Projeto Global

Gestão de Utilizadores

LICENCIATURA EM INFORMÁTICA

GONÇALO TEODORO OLIVEIRA

Aluno N.º 8134/A010

Diurno, Turma B

ORIENTAÇÃO:

PROFESSOR DOUTOR PEDRO BRANDÃO

PROFESSORA MARIA INÊS

2017/2018

Lisboa

Resumo

É objetivo deste projeto a criação de uma aplicação de gestão de utilizadores de uma biblioteca, através de cinco máquinas virtuais, juntamente com uma base de dados. Assim sendo, foram criadas cinco máquinas virtuais, sendo uma delas *Hyper-V* e encontrando-se, as restantes, dentro desta última.

As evoluções tecnológicas e as recentes conceções para a gestão de recursos de informação têm causado uma alteração no paradigma dos modelos tradicionais de gestão de utilizadores. O conceito de utilizadores de uma biblioteca apresenta-se como uma alternativa para entender as condições de busca, disponibilidade e recuperação de informação de maneira globalizada, qualitativa, pertinente e racional, aliando o acesso local ao acesso remoto, com base nas redes de telecomunicação disponíveis.

A tecnologia de Virtualização e o conceito de Cloud Computing têm crescido a um nível exponencial nos últimos anos. A palavra *Cloud Computing* surgiu na primeira década do Séc. XXI, mas a ideia por detrás desta começou a ser pensada nos anos 90.

A Virtualização é uma tecnologia que é usada para partilhar as capacidades de computadores físicos, dividindo os recursos entre os Sistemas Operacionais. Isto, através de uma máquina virtual, oferece inúmeras vantagens a uma infraestrutura de TI, destacando-se o ambiente completo, similar ao de uma máquina física, com *sistema operativo*, aplicações e serviços de rede.

Palavras-Chave: Hyper-V; Virtualização; Cloud; Cloud Computing; SQL; ADO.NET; Sistemas Operativos

Abstract

The purpose of this project is to to create a user management application for a library, through five virtual machines, along with a database. Thus, five virtual machines were created: one of them is *Hyper-V* and the remaining ones are within the *Hyper-V*.

The technological evolutions and the recent conceptions for the management of information resources have changed the paradigm of the traditional models of user management. The concept of users of a library presents itself as an alternative to understand the conditions of search, availability and retrieval of information in a globalized, qualitative, pertinent and rational manner, combining local access to remote access, based on available telecommunication networks.

Virtualization technology and the concept of Cloud Computing have grown exponentially in recent years. The word *Cloud Computing* came in the first decade of the 21st century, but the idea behind it began to be thought of in the 1990s.

Virtualization is a technology that is used to share the capabilities of physical computers, dividing resources between *Operating Systems*. This, through a virtual machine, offers many advantages to an IT infrastructure, with emphasis on the complete environment, similar to that of a physical machine with operating system, applications and network services.

Keywords: Hyper-V; Virtualization; Cloud; Cloud Computing; SQL; ADO.NET; Operating Systems

Índice

Índice de Imagens

(Destaca-se o facto de todas as imagens serem fonte do autor.)

Introdução

Para o projeto em curso, serão utilizadas tecnologias que tiram partido do potencial das tecnologias de informação, para a construção de uma aplicação, que permite interagir com máquinas virtuais para o acesso a um registo de utilizadores, mantida numa base de dados SQL.

Este relatório final, do projeto global está dividido em três parte: o primeiro é sobre o Estado da Arte onde é feito um estudo bibliográfico das tecnologias de virtualização, c cloud computing, SQL e ADO.NET, mostrando as principais caraterísticas e funcionalidades de cada uma destas tecnologias, bem como o impacto que têm e vão continuar a ter no presente e no futuro; a segunda parte é a Contextualização, onde é realizada uma breve explicação sobre o tema em si e a terceira parte é o Desenvolvimento, que como já foi referido, serve para explicar o código da aplicação que foi realizada em *Windows Form*. A aplicação tem um *CRUD,* máquinas virtuais que comunicam entre si, uma base de dados em SQL, para guardar os dados inseridos na aplicação e ainda o utiliza a tecnologia ADO.NET.

Estado de Arte

I. Virtualização

A virtualização é algo que poderá estar tanto no *desktop* de um entusiasta pelo assunto quanto no ambiente de *TI* de uma infinidade de empresas das mais variadas áreas. Não se trata de algo novo ou uma moda cíclica. Graças a este conceito, é possível, entre outros benefícios, economizar em equipamentos e obter resultados de determinadas tarefas executadas por computadores num tempo mais reduzido. A sua origem surgiu na década de 1960, propagando-se com mais força a partir 1970 sendo a virtualização extremamente importante para um mundo cada vez mais digital (Portnoy, 2017).

O seu conceito poderá ser definido soluções informáticas que permitem a execução de vários sistemas operativos e respetivas aplicações a partir de um equipamento, seja ele um *desktop* convencional, portátil ou um potente servidor. É como se existisse um ou mais computadores distintos dentro de um só. A diferença é que estas máquinas são virtuais, na prática elas oferecem resultados como qualquer outro computador, mas existem apenas num espaço lógico e não fisicamente. Cada máquina virtual traduz-se num ambiente completo para processamento de dados, praticamente todos os recursos do seu sistema operativo podem ser utilizados sendo possível executar diversas operações, conecta-las em rede e instalar aplicativos (*ibid*, pr.3).

Uma das principais razões para o desenvolvimento da virtualização deveu-se a que numa época em que os *mainframes* dominavam o panorama tecnológico e não havia computadores pessoais, por exemplo, não existia a prática de adquirir, instalar e usar um *software*, este era acompanhado de manuais e outros recursos que o tornavam quase como que exclusivos ao computador para o qual foi o desenvolvido originalmente. Desta forma, muitas vezes uma organização que implementava um novo sistema via-se obrigada a adquirir um equipamento apenas para executá-lo, em vez de simplesmente aproveitar o parque informático existente, no final todo este processo ficava extremamente caro (Postel, 1985).

Com a virtualização foi possível minimizar este problema: pode-se aproveitar um computador já existente para executar dois ou mais sistemas distintos, já que cada um trabalha dentro da sua própria máquina virtual. Evitam-se assim gastos com novos equipamentos e aproveitam-se os possíveis recursos ociosos do computador. Nos dias de hoje, a virtualização permite ainda, que uma empresa execute vários serviços a partir de um único servidor ou até mesmo que um utilizador doméstico teste um sistema operativo no seu computador antes de efetivamente instalá-lo. Do ponto de vista corporativo, atualmente destina-se a várias

aplicações como sistemas de *ERP*, serviços de processamento de dados nas nuvens e ferramentas de simulação, (*ibid*, pr. 5).

Embora tenham sido mencionadas algumas das vantagens da virtualização, a sua utilização oferece vários benefícios tais como um melhor aproveitamento da infraestrutura existente, ao executar vários serviços num servidor ou conjunto de máquinas, por exemplo, pode-se aproveitar a capacidade de processamento destes equipamentos o mais próximo possível da sua totalidade. O parque informático é menor, com o melhor aproveitamento dos recursos já existentes, a necessidade de aquisição de novos equipamentos diminui, assim como gastos inerentes com a sua instalação, espaço físico, refrigeração, manutenção, consumo de energia, entre outros. Desta forma pode-se ter a noção do impacto que esta vantagem pode ter num *datacenter* (Postel, 1985).

II. Cloud Computing

Processamento de dados nas nuvens não é um conceito claramente definido. Não é uma tecnologia que tenha saído dos laboratórios pelas mãos de um grupo de programadores que posteriormente foi disponibilizada no mercado. Esta característica faz com que seja difícil identificar com precisão a sua origem, mas há alguns indícios muito interessantes, um deles remete-se ao trabalho desenvolvido por John McCarthy. Falecido em outubro de 2011, foi um dos principais nomes por trás da criação do que conhecemos como inteligência artificial, com destaque para a linguagem *Lisp*, até hoje aplicada em projetos que utilizam tal conceito (Brinda & Heric, 2017).

Além deste trabalho, no início da década de 1960, John McCarthy tratou de uma ideia bastante importante: processamento de dados por *time sharing* (tempo compartilhado), onde um computador pode ser utilizado simultaneamente por dois ou mais utilizadores para a realização de determinadas tarefas, aproveitando especialmente o intervalo de tempo ocioso entre cada processo (McQuillan, 1975).

Pode-se entender que desta forma, é possível aproveitar melhor o computador (na época, um dispositivo muito caro) e diminuir gastos, uma vez que o utilizador somente paga, por exemplo, pelo tempo de uso do equipamento. É, de certa forma, uma ideia presente no processamento de dados nas nuvens (Brinda & Heric, 2017).

Quase que na mesma época, o físico Joseph Carl Robnett Licklider entrou para a história ao ser um dos pioneiros da internet. Isso porque, ao fazer parte da *ARPA* (*Advanced Research*

Projects Agency), lidou com a tarefa de encontrar outras utilidades para o computador que não fosse apenas a de ser uma "poderosa calculadora" (Leiner, Mills, & Postel, 1985).

Nesta matéria, Licklider foi um dos primeiros a entender que os computadores podiam estar interligados, permitindo a comunicação de forma globalizada e consequentemente, a partilha de dados. O seu trabalho determinou a criação da *Intergalactic Computer Network*, que posteriormente deu origem à *ARPANET*, que por sua vez "abriu as portas" para a internet (*ibid*, pr. 7).

Estão habituados a armazenar arquivos e dados dos mais variados tipos e a utilizar aplicações de maneira *on premise (no local)*, isto é, instalado nos computadores pessoais. No ambiente corporativo, este cenário é apenas um pouco diferente, já que nele é mais fácil encontrar aplicações disponíveis em servidores que podem ser acedidas por qualquer computador autorizado e autenticado por meio de uma rede (Brinda & Heric, 2017).

A principal vantagem deste modelo está no facto de ser possível, pelo menos na maioria das vezes, utilizar as aplicações mesmo sem acesso à internet ou à rede. Por outras palavras, é possível usar estes recursos em modo *off-line*. Entretanto, todos os dados gerados estão restritos a este computador, exceto quando partilhados em rede, coisa que não é muito comum no ambiente doméstico. Mesmo no ambiente corporativo, esta situação pode gerar algumas limitações, como a necessidade de ter que existir uma licença de um determinado programa para cada computador (Poole, Lambert, Woodford, Moschovitis, & Barbara, 2005).

A evolução constante das tecnologias de informação e telecomunicações está a permitir que o acesso à internet se torne cada vez mais amplo e cada vez mais rápido. Em países mais desenvolvidos, como o Japão, Alemanha e Estados Unidos, é possível ter acesso rápido à internet a baixo custo e em alguns de forma gratuita. Esta tendência cria a condição perfeita para a popularização da *cloud computing*, fazendo com que o conceito se torne conhecido em todo mundo, inclusive em Portugal (Brinda & Heric, 2017).

Com a *cloud computing* muitos aplicativos, assim como arquivos e outros dados relacionados, não precisam estar instalados ou armazenados no computador do utilizador ou num servidor próximo. Este conteúdo para a ficar disponível nas nuvens, isto é, na internet. Ao fornecedor da aplicação cabe todas as tarefas de instalação, armazenamento, manutenção, atualização, *backup* e escalonamento. O utilizador não precisa de se preocupar com nenhum destes aspetos, apenas com aceder e utilizar (*ibid*, pr. 5).

Um exemplo prático desta nova realidade é o Google *Docs*, serviço onde os utilizadores podem aditar textos, elaborar apresentações de slides, armazenar arquivos, entre outros, tudo pela internet, sem necessidade de ter programas como *Microsoft Office* ou *OpenOffice.org* instalados nos computadores. O que o utilizador precisa fazer é apenas abrir o explorador de internet e aceder ao endereço do Google *Docs* para começar a trabalhar, independentemente do sistema operativo ou do computador utilizado para esse fim. Neste caso, o único cuidado que o utilizador deve ter é o de utilizar um explorador de internet compatível, o que é o caso da maioria dos *browsers* (exploradores de internet) da atualidade (Leiner, 2000).

No Cloud Computing tal como já foi mencionado, uma das vantagens da *cloud computing* é a possibilidade de se utilizar aplicações diretamente da internet, sem que estas estejam instaladas no computador do utilizador. No entanto existem outros benefícios a ter em conta. Na maioria dos casos, o utilizador pode aceder a determinadas aplicações independente do seu sistema operativo ou de *hardware*. O utilizador não precisa de se preocupar com a estrutura para executar a aplicação: *hardware*, procedimentos de *backup*, controle de segurança e manutenção, ficam a cargo do fornecedor de serviços. Partilha de dados e trabalho cooperativo tornam-se mais fáceis, uma vez que todos os utilizadores acedem as aplicações e aos dados do mesmo lugar: a Cloud. Muitas aplicações do tipo já são criadas tendo em consideração essas possibilidades (Brinda & Heric, 2017).

Dependendo do fornecedor, o utilizador pode contar com alta disponibilidade, já que se um servidor parar de funcionar, os demais, que fazem parte da sua estrutura, continuam a oferecer o serviço. O utilizador pode contar com melhor controlo de gastos. Muitas aplicações em *cloud computing* são gratuitas, e, quando é necessário pagar, o utilizador só o fará em relação aos recursos que usar ou ao tempo de utilização. Não é, portanto, necessário pagar por uma licença integral de uso, tal como acontece no modelo tradicional de fornecimento de programas. Dependendo da aplicação, o utilizador pode precisar de instalar um programa cliente no seu computador. Mas nesse caso, todo ou a maior parte do processamento (e até mesmo do armazenamento de dados) fica por conta das Cloud. Pode-se observer que, independente da aplicação, com a *cloud computing*, o utilizador não necessita de conhecer toda a estrutura que há por trás, ou seja, não precisa de saber quantos servidores executam determinada ferramenta, quais as configurações de *hardware* utilizadas, como o escalonamento é feito, onde está a localização física do *datacenter*. O que importa ao utilizador é saber que a aplicação está disponível nas nuvens, não importa de que forma (Poole, Lambert, Woodford, Moschovitis, & Barbara, 2005).

Existem diversos tipos de serviços que são disponibilizados através de *Cloud Computing*, nomeadamente *Software as a Service (SaaS). Software* este que como serviço está ligado à *cloud computing*. Trata-se de uma forma de trabalho onde os programas são oferecidos como serviço, assim, o utilizador não precisa de adquirir licenças de uso para instalação ou mesmo comprar computadores ou servidores para executá-lo. Esta modalidade, no máximo, paga-se um valor periódico – como se fosse uma assinatura – somente pelos recursos utilizados e/ou pelo tempo de uso. Para entender os benefícios do *SaaS*, admitindo que uma empresa que tem vinte funcionários necessita de um programa de contabilidade. Há várias soluções prontas para isso no mercado, no entanto, a empresa terá que comprar licenças de uso do programa escolhido, e, dependendo do caso, até mesmo equipamento informático para executá-lo. Muitas vezes, o preço da licença ou mesmo dos equipamentos pode gerar um elevado custo e não ser compatível com a condição de porte pequeno da empresa (Alecrim, 2012).

Se, por outro lado, a empresa encontrar um fornecedor de programas de contabilidade que trabalhe com o modelo *SaaS*, a situação pode ficar mais fácil: esse fornecedor poderá, por exemplo, oferecer esse serviço por meio de *cloud computing* e cobrar apenas pelo número de utilizadores e /ou pelo tempo de uso. Desta forma, a empresa interessada paga um valor baixo pelo uso da aplicação. Além disso, o equipamento informático, instalação, atualização e manutenção, ficam por conta do fornecedor. Também é importante levar em conta que o intervalo entre a contratação do serviço e o inicio da sua utilização é extremamente baixo, o que não aconteceria se o programa tivesse que ser instalado nos computadores do cliente. Este só precisa de se preocupar com o acesso ao serviço (no caso, uma conexão à internet) ou, se necessário, com a simples instalação de algum recurso mínimo, como um conector no explorador de internet de suas máquinas. IBM e HP são dois exemplos de companhias que já oferecem soluções em *SaaS*: HP *SaaS*, IBM *SaaS* (*ibid,* pr. 6).

O mercado trabalha atualmente com conceitos derivados do *SaaS*, utilizados por algumas companhias para diferenciar os seus serviços. *Platform as a Service (PaaS)*, que em português significa, Plataforma como Serviço. Trata-se de um tipo de solução mais amplo para determinadas aplicações, incluindo todos (ou quase todos) os recursos necessários à operação, como armazenamento, base de dados, escalabilidade (aumento automático da capacidade de armazenamento ou processamento), suporte a linguagens de programação, segurança (Brinda & Heric, 2017).

O *Database as a Service (DbaaS)*, que em português significa Base de Dados como Serviço. Esta modalidade é direcionada ao fornecimento de serviços para armazenamento e acesso de volumes de dados. A vantagem é que o detentor da aplicação conta com maior flexibilidade para expandir a base de dados, compartilhar as informações com outros sistemas, facilitar o acesso remoto por utilizadores autorizados e autenticados. O *Infrastructure as a Service (IaaS)*, que em português significa Infraestrutura como Serviço. Parecido com o conceito de PaaS, mas aqui o foco é a estrutura de *hardware* o de máquinas virtuais, com o utilizador tendo inclusive acesso a recursos do sistema operativo. O *Testing as a Service (TaaS)*, que em português significa Teste como Serviços. Oferece um ambiente apropriado para que o utilizador possa testar aplicações e sistemas de maneira remota, simulando o comportamento destes em nível de execução (*ibid*, pr 8).

Os termos *cloud computing* e processamento de dados nas nuvens são relativamente recentes, mas ao analisar melhor, pode-se ter a perceção que a ideia não é, necessariamente, nova. Serviços de correio eletrónico, como *Gmail*, *Yahoo*! e *Mail*. Discos virtuais na internet, como *Dropbox*, espaços na internet destinados ao armazenamento e compartilhamento de fotos ou vídeos, como *Flickr* e *YouTube*. São exemplos de aplicações que, de certa forma, estão dentro do conceito do processamento de dados nas nuvens. Note-se que todos estes serviços não são executados no computador do utilizador, mas este pode aceder aos serviços em qualquer lugar, muitas vezes sem necessidade de instalar as aplicações ou de pagar licenças de programas. No máximo, paga-se um valor periódico pelo uso do serviço ou pela contratação de recursos adicionais, como maior capacidade de armazenamento de dados (Poole, Lambert, Woodford, Moschovitis, & Barbara, 2005).

Segue-se uma breve lista de serviços que incorporam claramente o conceito de *cloud computing*, tais como o Google *Apps*: este é um pacto de serviços que o Google oferece que conta com aplicativos de edição de texto, folhas de cálculo e apresentações (Google *Docs*), ferramenta de agenda (Google *Calendar*), comunicador instantâneo integrado (Google *Talk*), correio eletrónico com domínio próprio (por exemplo, geral@istec.pt). Todos estes recursos são processados pela Google – o cliente precisa apenas criar as contas dos utilizadores e efetuar algumas configurações. A Google *Apps* oferece pacotes gratuitos e pagos, de acordo com o número de utilizadores. Um dos maiores clientes do Google *Apps* é a *Procter & Gamble*, que contratou os serviços para mais de 130 mil funcionários (*ibid*, pr. 4).

A Amazon é um dos maiores serviços de comércio eletrónico do mundo. Para suportar o volume de vendas no período de natal, a empresa montou uma superestrutura de processamento e armazenamento de dados. Foi a partir daí que a companhia teve a ideia de alugar estes recursos, o que acabou por resultar em serviços como o *Simple Storage Solution* (S3) para armazenamento de dados e o *Elastic Compute Cloud* (EC2) para uso de máquinas virtuais. É possível saber mais sobre as soluções oferecidas pela Amazon nesta página (Brinda & Heric, 2017).

O Panda Cloud Antivirus, como o nome indica, é um programa de antivírus da Panda *Software*, mas com uma grande diferença: a maior parte do trabalho necessário à ferramenta para pesquisar e eliminar cavalos de Tróia fica por conta das nuvens. Com isso, de acordo com a Panda, essa solução acaba evitando que o antivírus deixe o computador lento (*ibid*, pr. 5).

O ICloud, anunciado em junho de 2011, trata-se de um serviço da Apple que armazena músicas, fotos, vídeos, documentos e outras informações do utilizador. O seu objetivo é fazer com que a pessoa utilize as nuvens em que de um computador na sua rede como "hub" para centralizar as suas informações. Com isso, se o utilizador atualizar as informações de um contacto no telemóvel *iphone*, por exemplo, o *iCloud* poderá enviar os dados alterados automaticamente para outros dispositivos (Alecrim, 2012).

Private Cloud (nuvem privada), até agora, o processamento de dados nas nuvens tem sido mencionado como um sistema composto de duas partes: quem fornece a solução e o utilizador, que pode ser uma pessoa, uma empresa ou qualquer outra organização. Podemos estender este contexto como um esquema de nuvem pública. No entanto, especialmente no que diz respeito ao segmento corporativo, é possível também o seu uso do que se conhece como nuvem privada (Leiner., 2000).

Do ponto de vista do utilizador, a nuvem privada (*private cloud*) oferece praticamente os mesmos benefícios da nuvem publica. A diferença está, essencialmente, no *datacenter*, uma vez que os equipamentos e sistemas utilizados para constituir a nuvem estão dentro da infraestrutura da própria corporação (*ibid,* pr. 3).

Por outras palavras, a empresa faz uso de uma nuvem particular, constituída e mantida dentro dos seus domínios. Mas o conceito vai mais além: a nuvem privada também considera a cultura corporativa, de forma que políticas, objetivos e outros aspetos inerentes às atividades da companhia sejam respeitados. A necessidade de segurança e privacidade é um dos motivos que levam uma organização a adotar uma nuvem privada. Em serviços de terceiros, cláusulas

contratuais e sistemas de proteção são os recursos oferecidos para evitar acesso não autorizado ou compartilhamento indevido de dados. Mesmo assim, uma empresa pode ter dados críticos por demais para permitir que outra companhia responde pela proteção e disponibilização destas informações. Ou, então, a proteção oferecida pode simplesmente não ser suficiente. Em situações como estas é que o uso de uma nuvem privada se mostra adequado (Alecrim, 2012).

Empresas como a Microsoft, IBM e HP oferecem soluções para nuvens privadas. As entidades interessadas, no entanto, devem contar com profissionais ou mesmo consultoria especializada na criação e na manutenção da nuvem, afinal, uma implementação mal-executada pode interferir negativamente no negócio (*ibid,* pr. 12).

Os custos de equipamentos, sistemas e profissionais da nuvem privada poderão ser elevados no inicio. Por outro lado, os benefícios obtidos a médio e longo prazo, como ampla disponibilidade, agilidade de processos e os já mencionados aspetos de segurança, compensarão os gastos, especialmente se a implementação for otimizada com virtualização, padronização de serviços, entre outros (Brinda & Heric, 2017).

Hybrid Cloud (nuvem híbrida) foi criada para a flexibilização de operações e até mesmo para maior controlo sobre os custos, as organizações podem optar também pela adoção de nuvens híbridas. Nestas, determinadas aplicações são direcionadas às nuvens públicas enquanto que outras, normalmente mais críticas, permanecem sob a responsabilidade da sua nuvem privada. Pode haver também recursos que funcionam em sistemas locais (*on premise*), complementando o que está nas nuvens (Proven, 2011).

Entenda-se que nuvens públicas e privadas não são modelos incompatíveis entre si. Não é preciso abrir mão de um tipo para usufruir do outro. Pode-se aproveitar o melhor dos dois mundos, razão pela qual as nuvens híbridas (hybrid cloud) são uma tendência muito forte nas corporações (*ibid*, pr. 15).

A implementação de uma nuvem híbrida pode ser feita tanto para atender a uma demanda contínua, quanto para dar conta de uma necessidade temporária. Por exemplo, uma instituição financeira pode integrar à sua nuvem privada um serviço público capaz de atender a uma nova exigência tributária. Ou então, uma rede de lojas pode adotar uma solução híbrida por um curto período para atender ao aumento das vendas numa época festiva. A eficácia de uma nuvem híbrida depende da qualidade da sua implementação. É necessário considerar aspetos de segurança, monitorização, comunicação, formação. Este planeamento é importante para avaliar inclusive se a solução híbrida é viável. Quando o tempo necessário para a

implementação é muito grande ou quando há grandes volumes de dados a serem transferidos para os recursos públicos, o seu uso pode não ser viável (Leiner., 2000).

Ao ser efetuada uma pesquisa em livros de redes, telecomunicações e afins, pode-se perceber que o desenho de uma nuvem é utilizado para fins de abstração. Neste sentido, a nuvem representa uma rede de algum tipo cuja estrutura não precisa de ser conhecida, pelo menos não naquele momento (*ibid*, pr. 9).

Por exemplo, se a ideia é a de explicar como funciona uma tecnologia de comunicação que interliga duas redes de computadores, não é necessário detalhar as características de cada uma. Assim, pode-se utilizar uma nuvem para indicar que há redes ali. O processamento de dados nas nuvens simplesmente absorveu esta ideia, mesmo porque o desenho de uma nuvem, seguindo a ideia de abstração, passou também a representar a internet (Proven, 2011).

Cloud computing é algo que não pode ser definido com uma exata precisão – pois as ideias por trás da noção de processamento de dados nas nuvens são muito novas e as opiniões de especialistas no processamento de dados ainda divergem (*ibid,* pr. 13).

Existe ainda muita coisa que necessita de ser tido em conta, o facto de a simples ideia de determinadas informações ficarem armazenadas em computadores de terceiros (no caso, os fornecedores de serviços), mesmo com documentos garantindo a privacidade e o sigilo, preocupam pessoas e principalmente empresas, motivo pelo qual este ponto precisa de ser melhor estudado e ultrapassado. Além disso, há outras questões, como o problema da dependência de acesso à internet: o que fazer quando a conexão cair? Algumas companhias já trabalham em formas de sincronizar aplicações *off-line* com *on-line* (conectado a uma rede de computadores estruturada), mas as tecnologias para isso precisam de evoluir bastante (Brinda & Heric, 2017).

De qualquer forma, o futuro aponta para esse caminho. Além das empresas mencionadas, companhias como Dell, Intel, Oracle e Microsoft que já estão a trabalhar nas mais variadas soluções para *cloud computing*. Esta última, por exemplo, já anunciou até o *Azure*, como uma plataforma própria para a execução de aplicações nas nuvens (*ibid,* pr. 11).

III. *SQL*

Teve o seu início em março de 1987 quando a Microsoft comprou os direitos do DataServer da Sysbase para o sistema operativo *OS/2*. O objetivo era provocar o interesse e chamar a atenção da comunidade do *dBase*. Por isso a Microsoft traçou um acordo com a

Ashton-Tate como forma de endossar esse novo processo. Assim, a primeira versão do *SQL Server* chamava-se *Ashton-Tate/Microsoft SQL Server* e chegou ao mercado na metade final de 1988. Em maio do ano seguinte a versão 1.0 para *OS/2* era lançada com parca participação da Microsoft (limitando-se apenas a poucas ferramentas, testes e o projeto visando tornar o aplicativo mais simples de ser instalado) (Trukhnov, 2008).

Na segunda metade de 1990 a união da Microsoft com a *Ashton-Tate* findou e a versão 1.1 do *SQL* Server passou a oferecer suporte para o *Windows 3.0* (que era uma novidade na época). Apesar disso, a base do *SQL* era produzida pela Sysbase e a Microsoft não tinha qualquer acesso ao código-fonte. Qualquer defeito tinha que ser relatado para a Sysbase e corrigido apenas por ela. Só em 1991 a Microsoft passou a poder aceder ao código do *SQL* apenas para leitura (Kriegel & Trukhnov, 2008).

Com o lançamento da versão 4.2 para *OS/2* no fim de 1991 a Microsoft já desenvolvia a base de dados em conjunto com a Sysbase, só foi alterado em agosto de 1993 quando a Microsoft finalmente conseguiu a totalidade da conceção do *SQL Server* e lançou a versão para o *Windows NT 3.1.* Essa nova versão já era em *32-bit*. Assim, pouco tempo depois, a parceria entre as duas empresas acabava. O aparecimento do *SQL* remota ao inicio da década de 70, na sequência de um trabalho intitulado "*A Relational Model of Data for Large Shared Data Banks*", da autoria de E. F. Codd, um investigador da IBM (Trukhnov, 2008).

Pretendia-se criar um modelo relacional que respondesse racional e eficientemente a inúmeros utilizadores a acederem a grandes volumes de dados. A IBM apresentou em 1974 um protótipo de base de dados relacional (*System/R*). o projeto terminaria em 1979, tendo o seu sucesso conduzido a resultados significativos. Estava desenvolvida uma linguagem para interrogação a múltiplas tabelas do *System/R* com suporte para múltiplos utilizadores. Foi designada *Structure English Query Language* (SEQUEL). Viria a evoluir para a designação atual (*Structured Query Language* – SQL) (Kriegel & Trukhnov, 2008).

A Oracle Corporation (inicialmente relational Software, Inc.) viria a desenvolver em 1979 a 1° versão comercial do SQL. A partir de então o sucesso do *SQL* conduziu ao aparecimento de *SQL/Data System* (SQL/DS) posteriormente *Database 2* (DB2). O emprenho da própria IBM determinou efetivamente a consolidação do *SQL*. Atualmente existem diversos esforços no sentido da criação de uma definição standard para integrar os múltiplos fabricantes. Todas as grandes marcas mundiais aderiram a este padrão, nomeadamente a Microsoft *SQL Server*, Oracle, Informix, Sybase e obviamente IBM DB2 (Kriegel, 2015).

Com o fim das parcerias a Microsoft lançou a versão 6.0 do *SQL Server* em 1995. A versão 6.0 aumentou o desempenho substancialmente provendo mecanismos internos de replicação e administração centralizada. A partir deste momento o *SQL* esteve sempre a evoluir (Kriegel & Trukhnov, 2008).

Em 1996, a Microsoft lançou a versão 6.5 do SQL Server. Essa versão trouxe melhorias significativas para a tecnologia disponibilizando diversas novas funcionalidades. Em 1997, a Microsoft lançou a versão empresarial do SQL 6.5. Depois em 1998, a Microsoft lançou o *SQL Server* 2000. Foi o lançamento mais importante do *SQL Server* até ao momento. Essa versão foi construída sobre o *Framework* só *SQL Server 7.0.* No ano de 2003, o *SQL Server* 2000 ganha a sua versão em *64-bit* podendo aceder a maiores quantidades de memória. Em 2005: é lançado o *SQL Server* 2005 (sob o nome de Yukon) com grande integração na plataforma *.NET.* O *SQL Server* dá mais um passo em direção às grandes plataformas corporativas. A Microsoft exibe alguns grandes casos de sucesso (como a Xerox que consegue realizar até 7.000.000 transações diárias utilizando o *SQL Server* 2005. Em 2007 a Microsoft divulga, numa feira mundial de *Business Intelligence*, o lançamento do *SQL Server* 2008. Em 2008 a criação do *Resource Governor, Transparent Data Encryption*, auditoria no *SQL* através do *SQL Audit.* Também foi criado o *Policy Based Management*, que anteriormente se chamava *Declarative Management Framework.* O suporte a *Spatial Data*, permitindo armazenamento de dados Geográficos e Geométricos nativamente, inclusivo com ótimos recursos já criados (Kriegel 2015).

Contrariamente ao que se possa imaginar, o *R2* não é um pacote de atualização para o *SQL Server* 2008. É uma edição nova, lançada no ano de 2010. Essa edição possibilitou utilizar powerpivot dentro do Excel. Foi a origem do *MDS* e *DQS* (*Master Data Services e Data Quality Services*) e também do *streaminsight.* Evolução do *Report Builder* para versão 3.0 (*ibid,* pr. 5).

Em 2012, o *SQL Server* 2012 proporciona confiança para soluções críticas com maior disponibilidade, alto desempenho e recursos de segurança aperfeiçoados: *insights* revolucionários com opção de escolha gerenciada e capacidades de visualizações de dados interativas que facilita o entendimento e integração do utilizador. Depois em 2014, *SQL Server* 2014 foi concebido a 18 de março de 2014, e comercializado dia 1 de abril de 2014. Até novembro de 2013, foram efetuadas algumas revisões, o *SQL Server* 2014 fornece uma nova capacidade em memória para tabelas que podem caber inteiramente na memória (também

conhecido como Hekaton). Embora pequenas tabelas poção estar na memória em todas as versões do *SQL Server*, elas também podem ficar no disco, por isso o trabalho efetuado pode ficar na memória volátil, efetuando toda a escrita e alterações no disco rígido (Kriegel 2015).

Em 1970, com a publicação do artigo intitulado "*A Relational Model of Data for Large Shared Data Banks*" e a implementação da linguagem *SEQUEL* que foi desenvolvida pela *IMB* e tinha por objetivo a implementação do modelo de *Cood.Com* a evolução desta linguagem deu origem ao *SQL* (Kriegel & Trukhnov, 2008).

A linguagem *SQL* é considerada um padrão dos sistemas gestores de base de dados relacionais. Por isso todos os fabricantes a integram nos seus produtos. A linguagem divide-se em cinco gerações, são elas: 1° Geração – Código Máquina, 2° Geração – *Assembley*, 3° Geração – *Pascal, C, Cobol, Fortan, Basic*, 4° Geração – *SQL*, 5° Geração – *C++, Java, Delphi, Visual Basic* (*ibid*, pr. 6).

A linguagem *SQL* é dividida em subconjuntos de acordo com as operações que queremos efetuar sobre uma base de dados, como por exemplo o DML= Linguagem de Manutenção de Dados, que é utilizado para realizar inclusões, consultas, alterações e exclusões de dados presentes em registos. O DDL= Linguagem de Definição de Dados que permite ao utilizador definir tabelas novas e elementos associados, os principais comandos são *DROP* (apagar) e *CREATE* (criar). O DCL= Linguagem de Controlo de Dados, linguagem essa que tem um comando que interage com áreas de controlo como transação, conexão e desconexão de qualquer cadeado ligado a dados. DQL= Linguagem de Consulta de Dados esta linguagem, embora tenha apenas um comando, essa linguagem é a parte da *SQL* mais utilizada. O comando *SELECT* que permite ao utilizador especificar uma consulta como uma descrição do resultado desejado. Esse comando é composto por várias cláusulas e opções, possibilitando elaborar consultas das mais simples às mais elaboradas (Kriegel 2015).

Dentro do mecanismo de base de dados o utilizador encontra serviços para armazenar, manipular efetuar copias e restauros dos dados, possui também uma avançada capacidade de segurança para proteger os seus investimentos com serviços que asseguram a máxima disponibilidade, inclusive a sua infraestrutura de dados pode ser estendida para trabalhar com dados não estruturados e fazer sincronização entre múltiplas cópias de uma base de dados (Trukhnov, 2008).

As principais vantagens do *SQL* são custos reduzidos de formação, portabilidade de aplicação, longevidade, comunicação entre sistemas, liberdade de escolha do consumidor.

Enquanto que as desvantagens são limitações na criatividade, *SQL* foge um pouco do ideal de ser uma linguagem relacional, algumas deficiências intrínsecas (*ibid*, pr. 4)

IV. ADO.NET

O *ADO.NET* pode ser considerado uma evolução da tecnologia *ADO*, e obteve algumas mudanças. Os motivos da evolução do ADO para o *ADO.NET* deu-se devido à necessidade de melhor integração com *XML* (*Extensible Markup Language*) e um novo modo de acesso. Com esta evolução, ler e utilizar XML no *ADO.NET* tornou-se mais simplificado (Sceppa, 2006).

Este novo modo de acesso aos dados expira as capacidades de memória do cliente, pois até aos modelos anteriores utilizava-se sempre dados no servidor e consultas a todo o momento, linha por linha. Já no modelo *ADO.NET* os dados podem ser reenviados por completo ao cliente, não a tabela inteira, mas sim apenas a seleção, que armazena na memória, no formato XML, e atualiza o servidor ao final das operações com a tabela (*ibid*, pr. 6).

Quanto à utilização de *XML* no *ADO.NET*, esta tornou-se mais simples, pois pode ler e gravar *XML* sem a necessidade de utilização do *XMLDOM (Document Object Model)*, um completo conjunto de instruções que ligam com o *XML* a nível de elementos e não como dados relacionais. *ADO.NET* é uma evolução desse modelo, e uma revolução pois foi construído a custo zero, sem reaproveitar a tecnologia *ADO*. A sua única herança é o nome, e mesmo assim apenas a abreviação, pois *ADO.NET* não significa *ActiveX Data Objects.NET*, mas sim, simplesmente, *ADO.NET (*Kriegel & Trukhnov, 2008).

A equipa da Microsoft procurou manter muitos dos conceitos do *ADO.NET* semelhantes aos do *ADO* tradicional, apesar de mudar totalmente a sua arquitetura. Com isso os programadores com experiência em *ADO* podem familiarizar-se com o *ADO.NET* mais rapidamente. Ainda é possível utilizar o *ADO* tradicional através da interoperabilidade com objetos *COM (Component Object Model)*, no entanto, recomenda-se a migração para o *ADO.NET*, a arquitetura mais madura e moderna para acesso a dados (*ibid,* pr. 10).

Pensado para um mundo conectado no qual vivemos hoje, a Microsoft desenvolveu o *ADO.NET* com características especiais para facilitar o desenvolvimento nesse contexto. O *ADO.NET* é um conjunto de *assemblies* que fazem parte da *.NET Framework* e que permitem a comunicação comas bases de dados realizando operações de leitura e atualização. *ADO.NET* utiliza dois tipos de objetos para modelo desconectado para aceder à base de dados: os objetos *Dataset*, que podem conter um ou mais *DataTable*, e os *.NET Data Provider* (Jones, 2002).

Para realizar esta operação, o *ADO.NET* tem vários clientes de fonte de dados, os quais se encontram no espaço de nomes – *namespace* – *System.Data*. Tais como o *System.Data.SqlClient* que permite o acesso à base de dados *SQL Server* 7.0 ou superior, o *System.Data.OleDb* que permite o acesso a qualquer outra fonte de dados exceto (não recomendado) para o *SQL Server* 7.0 ou superior, e ainda o *System.Data.Oracle* que permite o acesso à base de dados Oracle (*ibid*, pr. 4).

Criar uma conexão com uma base de dados significa usar o componente apropriado e fornecer informações ao componente para que ele possa encontrar a base de dados e poder aceder a informações a partir desta. Como exemplo, suponha-se que deseja criar uma conexão com uma base de dados *SQL Server* que está instalado na sua máquina local e aceder à base de dados *Northwind*. Nesse caso deve-se utilizar o fornecedor de dados *SQL Server .NET Data Provider do namespace System.Data.SqlClient,* que consiste no nome do servidor onde está a base de dados, o nome da base de dados e o nome do utilizador e a senha (Fonte: Do Autor).

Os adaptadores objeto (*DataAdapter*) estão encarregados de chamar os comandos de leitura, atualização, inclusão e exclusão de informações numa base de dados. Com o que foi descrito acima pode-se entender que cada adaptador deverá ter uma conexão para saber de onde irá aceder aos dados, um comando *Select* para saber quais os dados a aceder, um comando *Insert* para saber quais os dados a inserir, um comando *Update* para saber quais os dados a atualizar e um comando *Delete* para saber quais os dados a excluir (Kriegel & Trukhnov, 2008).

O código exemplo que um adaptador cria para o *SQL Server* usando uma das conexões acima mencionadas pode ser o seguinte: *Dim da As SqlDataAdapter = New SqlDataAdapter ("Select * from Clientes, conexao)* (Sceppa, 2006).

A linha de código acima cria uma instância do adaptador e atribui ao *SelectCommand* uma tarefa *SQL* que lê todos os registos da tabela de Clientes. (O resultado obtido com o *SelectCommand* pode ser usado para preencher um *DataReader* ou um *DataSet*.) (Jones, 2002).

Os comandos possuem a função que pode ser chamada para que um comando seja executado, podemos ainda informar quais os dados que desejamos receber. Dessa forma, ao executar um procedimento de armazenado, sem receber nenhum valor usa-se: *ExecuteNonQuery*. Se receber algum valor usa-se: *ExecuteScalar* (*ibid*, pr. 2).

Podem ter basicamente 4 comandos: Select, Update, Insert e Delete. Cada comando possui uma tarefa em SQL que dirá ao adaptador (DataAdapter) como realizar respetiva tarefa.

Para realizar tarefas simples usando comandos *SQL* podemos usar o objeto *CommandBuilder* que funciona como um construtor automático dos comandos *SQL* básicos. Como por exemplo: *Dim comandoBuilder As SqlComandBuilder = New SqlCommandBuilder(da).* Com esta linha do código criamos os comandos: *Insert, Update* e *Delete* para o Adaptador (Kriegel & Trukhnov, 2008).

ADO.NET, até à versão 1.1., fornecia dois objetos para devolver e armazenar dados em memória: *DataSet e DataReader*. *DataSet* fornece uma representação relacional em memória de dados, sendo um conjunto completo de dados que incluem tabelas que contém dados, restrições de dados e relacionamentos entre tabelas. O acesso é desconectado. *DataReader* fornece um acesso conectado somente-leitura e somente-para-frente a uma base de dados. *DataSet* fornece uma representação relacional em memória de dados, sendo um conjunto completo de dados que incluem tabelas que contém dados, restrições de dados e relacionamentos entre tabelas. O acesso é desconectado. Ao usar um *DataSet* frequentemente também se usa um *DataAdapter* (e possivelmente um *CommandBuilder*) para interagir com a sua fonte de dados. Ao usar um *DataSet* pode-se empregar um *DataView* para aplicar organização e filtragem nos dados do *DataSet*. O *DataSet* também pode ser herdado para criar um *DataSet* (*ibid*, pr. 9).

O ADO.NET *DataProvider* são bibliotecas que possibilitam uma maneira comum de interagir com uma base de dados específica. Cada biblioteca possui um prefixo que indica que fornecedor ela suporta. Criada para manipular os dados independentemente da origem. Pode receber os dados de bases de dados através do *DataAdapter*. Pode trabalhar diretamente com arquivos *XML*. Pode ainda trabalhar com base de dados diferentes dentro de um mesmo *DataSet*. Permite manipular os dados, efetuando leituras e alterações necessárias no modelo, sem a necessidade de manter a conexão aberta com a base de dados. Dessa forma consome menos recursos à rede e ao equipamento tornando o seu acesso mais rápido (Jones, 2002).

Totalmente integrado em *.NET Framework*, como é nativo, a *API* (Interface de Programação de Aplicações) pode ser utilizada com as diversas linguagens que fazem parte do *.NET Framework*: *Visual Basic, C#...*, é sucessor do *ADO*, porém, mais flexível, possui um conjunto de Classes, Interfaces, Estruturas e Enumerações que gerem o acesso a dados dentro do *.Net Framework*, a base de dados sempre foi um ponto forte na aplicação. Sendo que uma falha na instalação pode prejudicar o desempenho da própria aplicação. Portanto, é fundamental conhecer bem para que se possa preparar da melhor forma possível, tal como

dimensionar a solução a ser adotada. Existem diversas opções no mercado: *MC ACCESS* – é um programa de base de dados recomendado para o uso doméstico principalmente pela sua fácil utilização podendo considerar-se uma iniciação ao *MS SQL Server* (*ibid*, pr. 8).

MS SQL SERVER – este é um servidor de dados profissional e é atualmente um dos melhores do mercado. Tem como grande destaque a sua facilidade de utilização, resultando num ótimo produto com um custo baixo de manutenção, tal como custo de aquisição se comparando com produtos similares no mercado. Pode ser implementado de forma individual ou em paralelo (*Cluster*) com suporte a tolerância de falhas, desde que possua equipamento informático, sistema operativo e programas para isso (Kriegel & Trukhnov, 2008).

MS MSDE - este programa é uma versão gratuita e similar ao SQL Server. Para evitar o uso indevido, essa versão conta com um recurso que resulta da falta de desempenho no caso de mais de 5 transações simultâneas, tal como limitação do tamanho do aquivo de dados (*ibid*, pr. 12).

Logo no início, para ter acesso às informações da base de dados era necessário ter vasto conhecimento das *API's* de comunicação, e o programador acabava por implementar o seu próprio código exclusivamente para cada versão de driver como a *DBLIB* (*DataBase LIBrary – SQL Server*), dedicando com isso muito tempo a este tipo de implementação. Com o crescimento do mercado e a persistência desse problema, foi criado, em 1990, com o apoio da Microsoft e um consórcio de empresas, o padrão *ODBC* (*Open DataBase Connectivy*) (Sceppa, 2006).

Consiste numa camada intermediária encarregada de cuidar da comunicação com a *API* deixando para o programador uma interface único e padrão para todo o acesso à *SGDB* (sistema de gestão de base de dados). A partir do sucesso do *ODBC* e da experiência já estabelecida e necessidade de evolução foram surgindo outras propostas como o *DAO (Data Access Objects)* focado no *MS ACCESS*, sedo substituído logo depois pelo *RDO (Remote Data Objects)*, uma vez que o *DAO* era mais lento em paralelo com o *ODBC* (*ibid*, pr. 8).

O grande avanço da época deu-se em torno do *OLEDB* que se assemelhou muito com a arquitetura do *ODBC*, porém, trouxe a implementação de interfaces *COM* e a estratégia da *Microsoft UDA (Universal Data Access)* com objetivo de armazenamento distribuído. Semelhante ao *ODBC*, o *OLEDB* também teve a sua importância pois foi incorporado até por base de dados de padrão livre. Para facilitar a sua utilização, foi criado o *ADO (Activex Data*

Objects) com o objetivo de consumir os recursos oferecidos pela *OLEDB (Object Linking and Embedding Database)* (Jones, 2002).

Através do que foi mencionado pode-se observar que foram criadas várias camadas de acesso à base de dados, com o objetivo de simplificar e padronizar a sua utilização. Realmente foi fundamental para o desenvolvimento e evolução das aplicações deixando o padrão *OLEDB* em conjunto com o *ADO* na liderança no acesso aos dados devido a um melhor desempenho no acesso aos mesmos. Sendo utilizado por diversas soluções ficou como padrão *ODBC* para as soluções não compatíveis com *ODBC*. Assim sendo vale a pena ressalvar que mesmo usando *OLEDB* ou *ODBC*, a aplicação vai ter uma perde de desempenho pois vai ter uma camada intermediária entre a sua aplicação e as *API's* de acesso aos dados. (Fonte: Do autor)

ADO.NET fornece um conjunto mínimo de interface que permite implementar a sua própria ligação para acesso aos dados em *.NET* (Jones, 2002).

Contextualização

A razão que suscitou ter escolhido este tema deve-se ao progresso que estas tecnologias têm tido ao longo dos anos e ao impacto que têm hoje em dia no mercado. Os termos de Cloud Computing e de Virtualização são falados desde há muito décadas atrás, mas apenas neste século deu-se um crescimento exponencial sobre estas duas tecnologias e atualmente são muitas as empresas que oferecem serviços de nuvem, tal como a Amazon e a Google e também disponibilizam a criação de máquinas virtuais de servidores tal como o Azure.

Apesar de alguma desconfiança no início devido a uma certa falta de segurança por parte destes serviços, com a chegada de cada vez mais profissionais e especialistas na área, nos dias de hoje, a Virtualização deu um passo em frente e está presente em muitas empresas que possuem um bom conforto monetário para pagar os elevados custos iniciais de uma nuvem.

O ADO.NET, criado pela Microsoft, não é mais que um conjunto de classes que vai interligar através de código com o SQL Server para aceder aos dados armazenados numa base de dados remota, posto isto executa comandos e recupera resultados.

O SQL tem uma grande importância nos dias de hoje, tal como no futuro irá ter porque é a principal e a mais utilizada linguagem padrão para gerir e consultar numa base de dados. Devido à facilidade do seu uso, e o facto de ser muito intuitivo (ambiente de código e ambiente gráfico), faz com que o SQL, atualmente ainda continue a ser a referência quando se fala na manipulação de base de dados.

Se estão tecnologias separadas são importantes, então quando fazemos uma junção de todas elas torna-se muito mais interessante, que é exatamente o que vai acontecer na aplicação. Assim percebemos que real finalidade é ainda mais importante e necessário para o trabalho final.

Desenvolvimento

O Objetivo deste trabalho era a criação de cinco máquinas virtuais, dentro dessa máquinas e todas interligadas entre si iriam armazenar a aplicação. A partilha de rede e um Sevidor de *Domain Controller* para fazer a ligação entre as máquinas e ainda um servidor de SQL.

A máquina virtual principal é o *Hyper-V,* porque dentro foram criadas mais quatro máquinas – através da geração 1 do *Hyper-V* – uma de *Domain Controller*, para criar o domínio, outra de *Routing* para distribuir a rede por todas as máquinas do tipo NAT, uma SQL *Server* para armazenar o servidor de SQL, onde guarda a base de dados e, por fim uma máquina cliente com o Windows 8.1 instalado para correr a aplicação.

A máquina DC, que contem o *Domain Controller* instalador, que por sua vez está instalado no *Windows Server 2012 R2*. Esta maquina foi utilizada para criar o domínio (oliveira.local) com a rede IP: 10.1.1.1 e também para criar os *Users*, para poderem ter aceso à aplicação.

A máquina de *Routing* também tem o *Windows Server 2012 R2*, criada com uma placa interna para comunicar com o domínio e com uma placa rede externa, com o objetivo de, através da placa *NAT* consiga que todas as máquinas se liguem à internet. Foram criados dois tipos de *Virtual Switch*, um é *Private Virtual Switch*, que foi adicionado a fodas as máquinas virtuais dentro do Hyper-V, que tem como finalidade as máquinas comunicarem em si. O outro é um *External Virtual Switch* que apenas foi adicionado na máquina *Routing* para comunicar com a placa *NAT* e espalhar a rede.

A máquina SQL também tem o *Windows Server 2012 R2*, que por sua vez foi instalado o SQL *Server*. Esta máquina é muito importante, porque é nela que a base de dados vai estar guardada.

A máquina virtual cliente usa o *Windows 8.1* e é onde a aplicação é usada.

Aplicação Gestão de Utilizadores

A aplicação é composta por duas janelas aplicacionais.

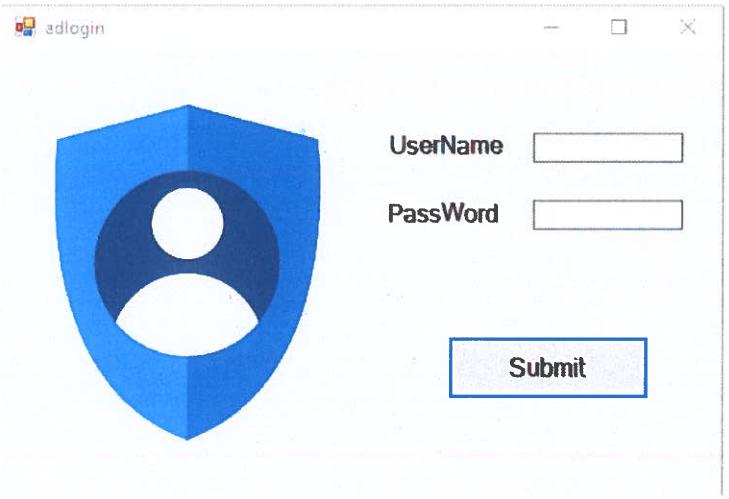

Figura 1. Janela de autenticação

Botão Submit – Através dos dados introduzidos na caixa texto do UserName e PassWord, faz uma consulta aos *Users* do *Active Directory*, da máquina virtual DC1 e verifica se o *User* existe.

```
string path = @"LDAP://DC.OLIVEIRA.LOCAL";
string domain = @"oliveira.local";
string user = txtBoxUser.Text.Trim();
string pass = txtBoxPassWord.Text.Trim();
string domUsu = domain + @"\" + user;
```

Figura 2. Excerto de código da figura 1

Excerto de código que conecta o form do login ao *Active Directoty do Domine Controller*. A criação das *strings* do evento do botão (Fig. 2), a *string path* contém o caminho do nosso LDAP. A *string domain* é o domínio que criamos na máquina DC1.

A *string user* e a *string pass* servem para armazenar os dados inseridos nas caixas de texto. A *string domUsu* vai juntar a *string domain* com a *string user*.

O método *Trim* remove os caracteres de espaço em branco à direita e à esquerda do objeto.

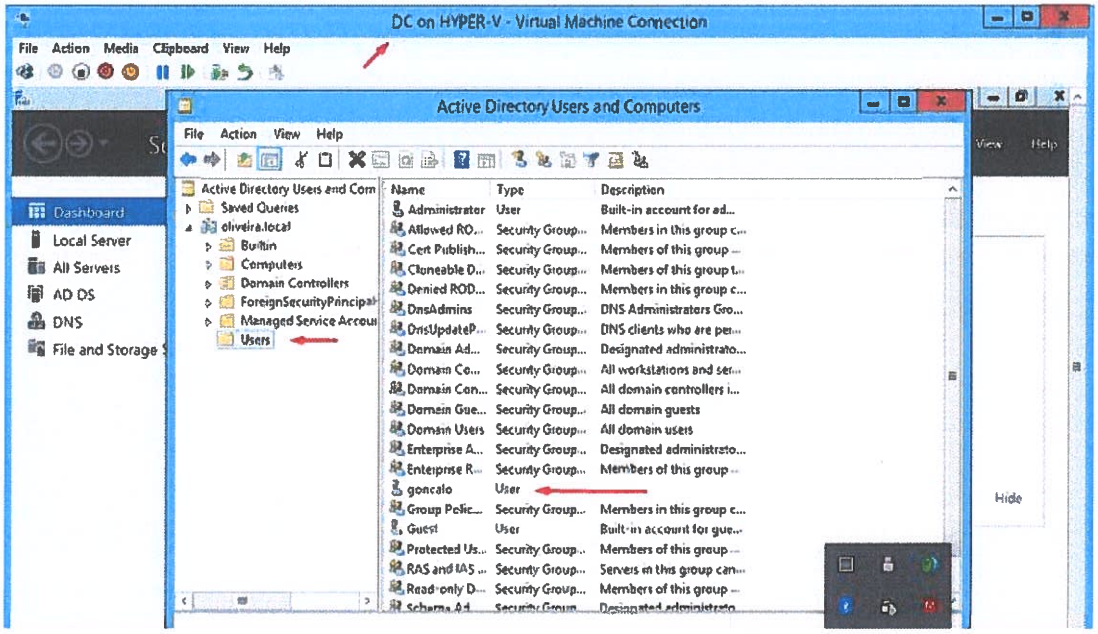

Figura 3. *Active Diretory* / *Users* referente à figura 1 / figura 2

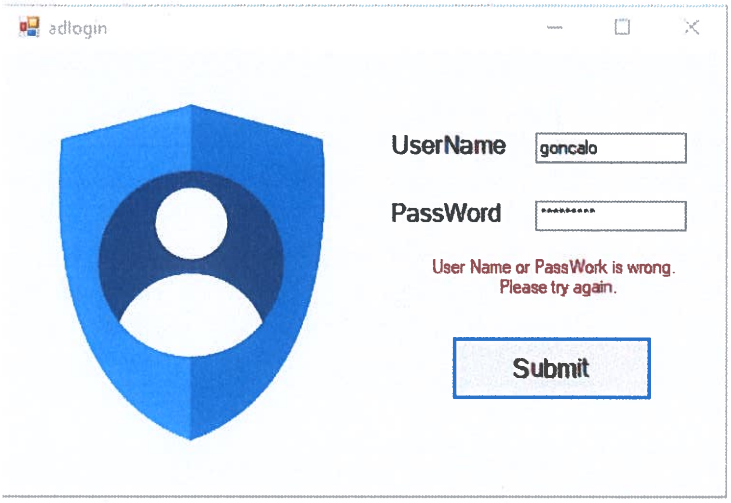

Figura 4. Janela de autenticação com erro

Caso o *User* não exista, a PassWord esteja incorreta ou o UserName esteja incorreto mostra uma mensagem de erro. Caso contrário avança para a próxima janela (Fig. 6).

Para Sair basta clicar no botão (x), no canto superior direito.

```
if (permission)
{
    this.Hide();
    Form1 form = new Form1();
    form.ShowDialog();
    errorlbl.Visible = false;
}
else
{
    errorlbl.Visible = true;
}
```

Figura 5. Excerto de código referente à figura 4

Neste excerto de código (Fig. 5) é verificado de o *User* tem permissão para executar o login, através da condição de que se for verdadeiro, a janela vai fechar e abrir-se-á uma nova janela (Fig. 6) e será aí que é possível introduzir os dados. Se não for verdadeira, retorna uma mensagem de erro.

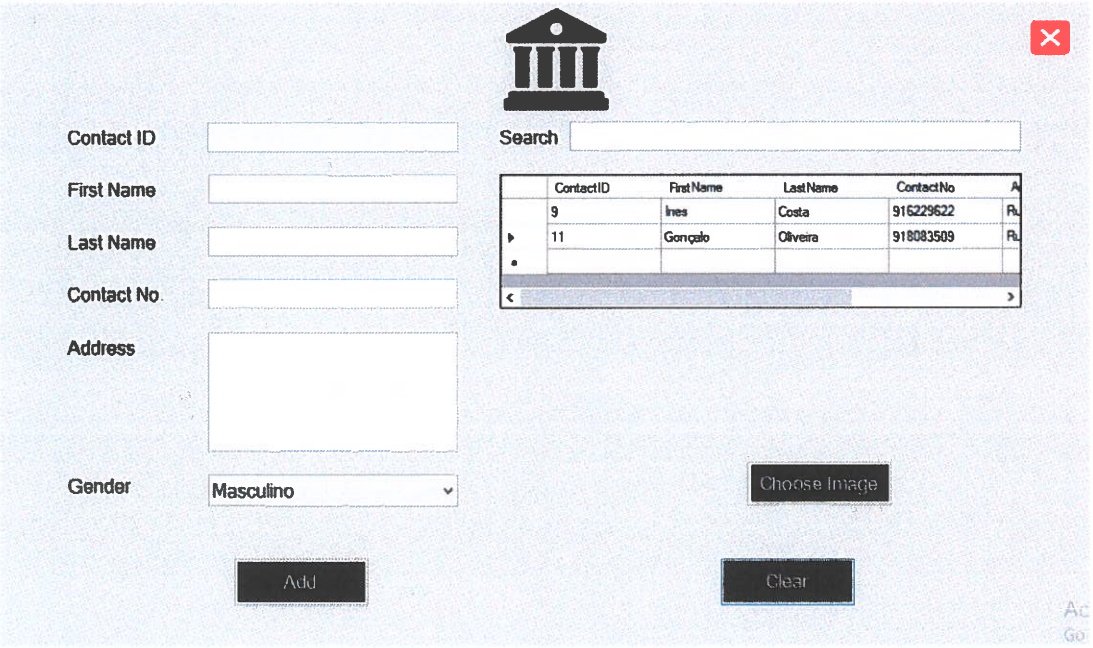

Figura 6. Janela Principal

Nesta janela, o utilizador cria um novo registo. Pode, também, editar, apagar ou até exportar para um documento PDF. No total, são 7 campos, sendo que só os campos First Name e Last Name são obrigatórios.

A interpretação de cada campo desta janela principal é feita da seguinte forma:

```
private void textBoxSearch_TextChanged(object sender, EventArgs e)
{
    //Recebe os dados das caixas de texto
    string keyword = txtboxSearch.Text;

    SqlConnection conn = new SqlConnection(myconnstr);
    SqlDataAdapter sda = new SqlDataAdapter("SELECT ContactID, FirstName, LastName, ContactNo, Address, Gender FROM tbl_contact
    DataTable dt = new DataTable();
    sda.Fill(dt);
    dgvContactList.DataSource = dt;
}
```

Figura 7. Excerto de código do Search

```
WHERE FirstName LIKE '%" + keyword + "%' OR LastName LIKE '%" + keyword + "%'OR Address LIKE '%" + keyword + "%'", conn);
```

Figura 8. Continuação do excerto de código do Search

Caixa de texto Search – Permite pesquisar por First Name, Last Name e Address. Basta escrever na caixa de texto o que pretende pesquisar e automaticamente aparece na *GridView*, como podemos verificar na (Fig. 9).

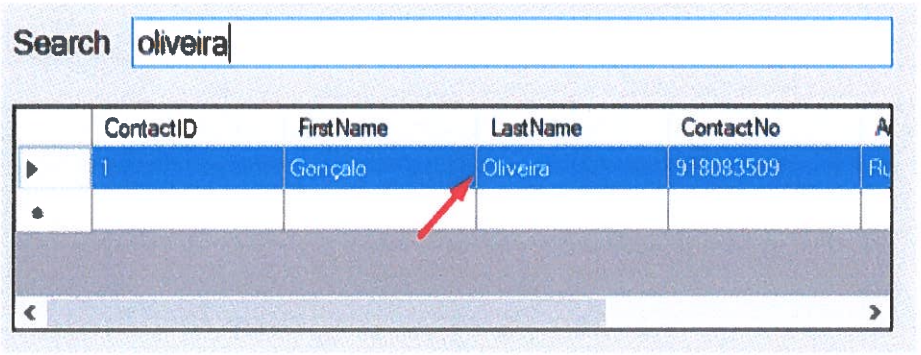

Figura 9. Pesquisa por Last Name

```
private void dgvContactList_RowHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)
{
    //Identifica a linha na GridView na qual foi clicada
    //Obtem os dados da linha selecionada da GridView e coloca-los nos campos respetivos
    int rowIndex = e.RowIndex;
    txtBoxContactID.Text = dgvContactList.Rows[rowIndex].Cells[0].Value.ToString();
    txtBoxFirstName.Text = dgvContactList.Rows[rowIndex].Cells[1].Value.ToString();
    txtBoxLastName.Text = dgvContactList.Rows[rowIndex].Cells[2].Value.ToString();
    txtBoxContactNo.Text = dgvContactList.Rows[rowIndex].Cells[3].Value.ToString();
    txtBoxAddress.Text = dgvContactList.Rows[rowIndex].Cells[4].Value.ToString();
    cmbGender.Text = dgvContactList.Rows[rowIndex].Cells[5].Value.ToString();
    ImageBox.Image = c.Selectimg(txtBoxContactID.Text);

    ContactID = dgvContactList.Rows[rowIndex].Cells[0].Value.ToString();
    FirstName = dgvContactList.Rows[rowIndex].Cells[1].Value.ToString();
    LastName = dgvContactList.Rows[rowIndex].Cells[2].Value.ToString();
    ContactNo = dgvContactList.Rows[rowIndex].Cells[3].Value.ToString();
    Address = dgvContactList.Rows[rowIndex].Cells[4].Value.ToString();
    Gender = dgvContactList.Rows[rowIndex].Cells[5].Value.ToString();

    btnAdd.Visible = false;
    btnUpdate.Visible = true;
    btnDelete.Visible = true;

    btnPDF.Visible = true;
}
```

Figura 10. Excerto de código da *GridView*

O código da figura 10 identifica qual a linha da *GridView* que foi clicada e preenche as respetivas caixas de texto com a devida informação que está guardada na Base de Dados. A figura 11 é a forma com vais buscar os dados à Base de Dados.

```
public DataTable Select()
{
    //Declara a variavel do tipo DataTable
    DataTable dt = new DataTable();
    try
    {
        //Escreve uma SQL Query
        string sql = "SELECT ContactID, FirstName, LastName, ContactNo, Address, Gender FROM tbl_contact";
        //Cria um comando que usa o SQL conection (conn)
        SqlCommand cmd = new SqlCommand(sql, conn);
        //Cria uma instancia da class SQL DataAdapter
        SqlDataAdapter adapter = new SqlDataAdapter(cmd);
        conn.Open();
        adapter.Fill(dt);
    }
    catch (Exception ex)
    {

    }
    finally
    {
        conn.Close();
    }
    return dt;
}
```

Figura 11. Código de dados da Base de Dados na *GridView*

Na *GridView*, pode observar-se os registos já efetuados, guardados na Base de Dados, com a respetiva informação guardada nos devidos campos. A função *select* retorna todos os dados da tabela *tbl_contact*, através do *sqlcommand*.

Caixa de texto Contact ID – Gera automaticamente o ID (número) de cada um dos utilizadores através do auto incremento.

Caixa de texto First Name – Primeiro nome do utilizador que está a efetuar o registo. Campo obrigatório.

Caixa de texto Last Name – Último nome do utilizador que está a efetuar o registo. Campo obrigatório.

Caixa de texto Contact No. – Contacto do utilizador.

Caixa de texto Address – Morada do utilizador.

ComboBox Gender – Escolher o sexo do utilizador. Pode ser Masculino ou Feminino.

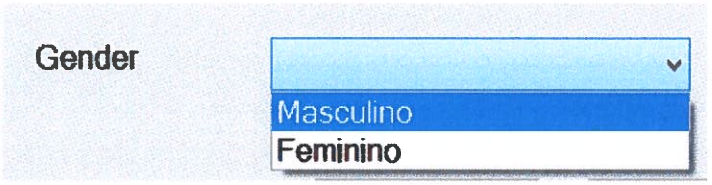

Figura 12. Gender

Botão Choose Image – Permite ao utilizador escolher uma imagem para concluir o registo. Campo obrigatório.

```
private void btnImage_Click(object sender, EventArgs e)
{
    try
    {
        OpenFileDialog dlg = new OpenFileDialog();
        dlg.Filter = "JPG Files (*.jpg)|*.jpg| PNG Files (*.png)|*png|All Files (*.*)|";
        if (dlg.ShowDialog() == DialogResult.OK)
        {
            LocImage = dlg.FileName.ToString();
            ImageBox.ImageLocation = LocImage;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Figura 13. Excerto de código do botão Choose Image

Na imagem acima (Fig. 13) podemos observar o excerto de código referente ao evento botão Choose Image.

Quando é pressionado o botão Choose Image é instanciado o componente *OpenFileDialog*, que permite ao utilizador escolher uma imagem guardada no seu computador, em formato .jpg ou .png. Posteriormente o caminho dessa imagem é guardado na variável LocImage, para mais tarde poder ser inserida na base de dados. Também é possível pré visualizá-la através do componente ImageBox.

```
public Image SelectImg(string ContactID)
{
    string sql = "SELECT Img FROM tbl_contact WHERE ContactID = @ContactID";
    //Cria um comando que usa o SQL e o connection (conn)
    SqlCommand cmd = new SqlCommand(sql, conn);
    cmd.Parameters.AddWithValue("@ContactID", Convert.ToInt32(ContactID));
    conn.Open();
    var img = cmd.ExecuteScalar();
    conn.Close();
    Byte[] ImageByte;
    Image imgFromByte;
    if (img != DBNull.Value)
    {
        ImageByte = (byte[])(img);
        MemoryStream ms = new MemoryStream(ImageByte);
        imgFromByte = System.Drawing.Image.FromStream(ms);
        return System.Drawing.Image.FromStream(ms);
    }
    imgFromByte = null;
    ImageByte = null;
    return null;
}
```

Figura 14. Excerto de código da imagem da Base de Dados

Retorna a imagem do utilizador selecionado da Base de Dados, através do ID selecionado. Na imagem acima (Fig. 14) podemos observar a forma como a imagem é convertida de *Bytes* para uma imagem pronta para poder ser visualizada.

```
public void Clear()
{
    txtBoxFirstName.Text = "";
    txtBoxLastName.Text = "";
    txtBoxContactNo.Text = "";
    txtBoxAddress.Text = "";
    cmbGender.Text = "";
    txtBoxContactID.Text = "";
    ImageBox.Image = null;
    btnAdd.Visible = true;
    btnUpdate.Visible = false;
    btnDelete.Visible = false;
    dgvContactList.ClearSelection();
}
```

Figura 15. Excerto de código do Clear Fonte

No código da figura 15, as caixas de texto estão todas vazias. Isto é, quando é feito o "clik" no botão Clear se as caixas de texto estiverem preenchidas passam a estar vazias. O mesmo acontece com a ImageBox, que fica vazia e, deixa de ter qualquer utilizador selecionado na *GridView*.

```
private void btnClear_Click(object sender, EventArgs e)
{
    //Chama o metedo clear para o evento do botão
    Clear();
    btnPDF.Visible = false;
}
```

Figura 16. Excerto de código do botão Clear

Botão Clear – Permite ao utilizador apagar os dados das caixas de texto todas de uma só vez, ao invés de ter que apagar uma por uma (Fig. 6) Na figura 16, o método *Clear* (Fig. 15) é chamado para o evento do botão Clear (Fig. 16).

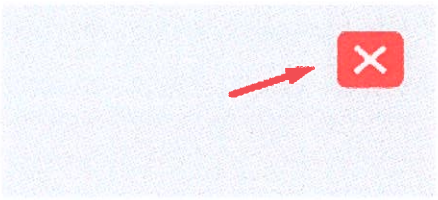

Figura 17. Fechar o programa

```
private void pictureBox2_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Figura 18. Excerto de código da PictureBox2

A PictureBox no canto superior direito permite fechar a aplicação (Fig. 17).

O código deste quadro principal (Fig. 6) poderá ser consultado no anexo form1.cs no bloco 49-232

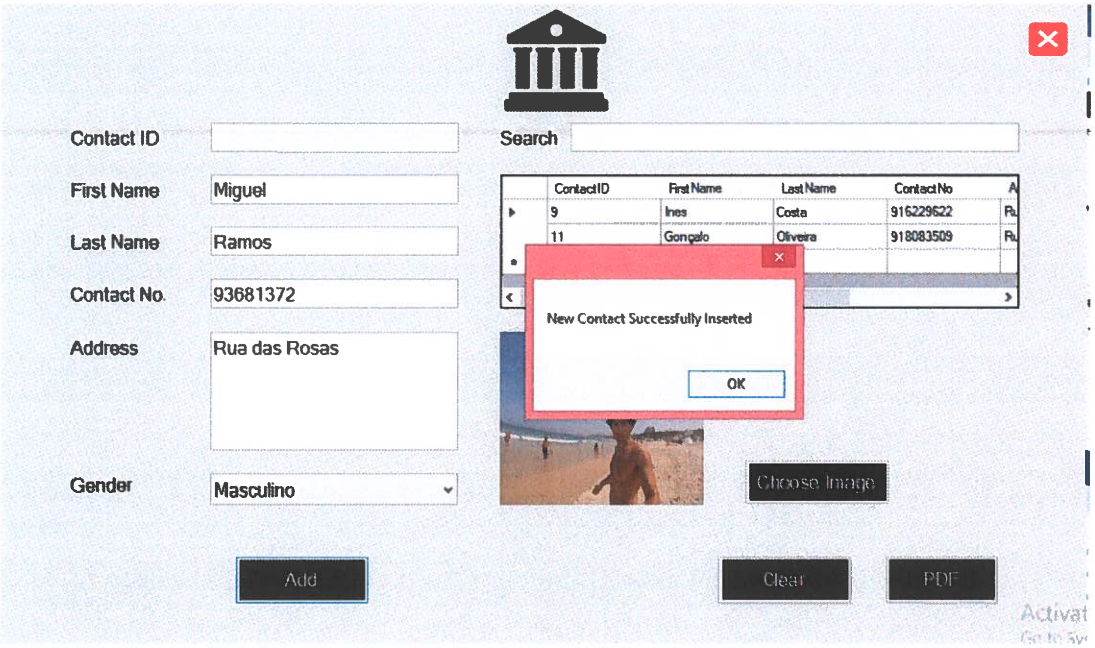

Figura 19. Novo registo

Ao preencher os campos e adicionar uma fotografia, basta clicar em Add para adicionar o registo à base de dados. Se o registo for bem feito aparece a mensagem acima no centro do ecrã, a dizer que o registo foi guardado com sucesso. De seguida basta clicar em OK e podemos verificar o registo na *GridView*, tal como na imagem (Fig.23).

```
try
{
    //Guardar a imagem em bytes na BD
    byte[] Image = null;
    if (LocImage != "")
    {

        FileStream fs = new FileStream(LocImage, FileMode.Open, FileAccess.Read);
        BinaryReader br = new BinaryReader(fs);
        Image = br.ReadBytes((int)fs.Length);
    }
    //Obtem os valores dos campos inseridos
    c.FirstName = txtBoxFirstName.Text;
    c.LastName = txtBoxLastName.Text;
    c.ContactNo = txtBoxContactNo.Text;
    c.Address = txtBoxAddress.Text;
    c.Gender = cmbGender.Text;
    c.ImageBytes = Image;
    //Insere os dados na Base de Dados
    c.Insert(c);
    //Inserido com sucesso
    MessageBox.Show("New Contact Successfully Inserted");
    //Chama o metedo Clear
    Clear();
    //Carrega os dados na GridView
    DataTable dt = c.Select();
    dgvContactList.DataSource = dt;
    LocImage = "";
}
catch (Exception ex)
{
    //Erro a adicionar novo registo
    MessageBox.Show("Failed to add New Contact. Try again! \n Error: " + ex.ToString());
}
```

Figura 20. Código referente a criar um novo registo

O código acima (Fig. 20) é referente ao evento do botão Add, onde se executa um *try,* em que dentro dele primeiro se a imagem não for nula, faz a conversão da imagem para bytes e guarda-a. De seguida obtém os valores dos campos inseridos e guarda-os na Base de Dados através da *class* "c". Posto isto mostra a mensagem de contacto inserido com sucesso, apaga os dados dos campos e mostra os valores na *GridView*. Se algum dos pontos acima não estiver correto, mostra a mensagem de erro, a dizer qual é o erro.

```
public void Insert(PFCRUDClass c)
{
    //Conecta à Base de Dados
    SqlConnection conn = new SqlConnection(myconnstrng);
    //Cria uma SQL Query para inserir dados
    string sql = "INSERT INTO tbl_contact (FirstName, LastName, ContactNo, Address, Gender, Img) VALUES (@FirstName,
    //Cria um comando em SQL usando sql e o conn
    SqlCommand cmd = new SqlCommand(sql, conn);
    //Cria parametro para adicionar dados
    cmd.Parameters.AddWithValue("@FirstName", c.FirstName);
    cmd.Parameters.AddWithValue("@LastName", c.LastName);
    cmd.Parameters.AddWithValue("@ContactNo", c.ContactNo);
    cmd.Parameters.AddWithValue("@Address", c.Address);
    cmd.Parameters.AddWithValue("@Gender", c.Gender);
    if (c.ImageBytes == null)
    {
        cmd.Parameters.AddWithValue("@Img", DBNull.Value);
    }
    else
    {
        cmd.Parameters.AddWithValue("@Img", c.ImageBytes);
    }

    //Abre a conecção
    conn.Open();
    cmd.ExecuteNonQuery();
    conn.Close();
}
```

Figura 21. Código referente a criar um novo registo na Base de Dados

```
@ContactNo, @Address, @Gender, @Img)";
```

Figura 22. Código referente a criar um novo registo na Base de Dados - continuação

O código da (Fig. 21 / Fig. 22) é como se processa a inserção de dados na Base de Dados, apos preencher os campos obrigatório e clicar no botão Add. Primeiro conecta à Base de Dados através do comando (*myconnstrng*), de seguida insere os valores dos respetivos campos. De seguida cria os parâmetros para adicionar os dados, incluindo a imagem. Por fim abre a conexão, excuta a *Query* e fecha a conexão.

ContactID	FirstName	LastName	ContactNo
1	Gonçalo	Oliveira	918083509
9	Ines	Costa	916229622
10	Miguel	Ramos	93681372

Figura 23. Registo guardado com sucesso na Base de Dados

Automaticamente, quando se efetua um registo com sucesso, ele fica logo selecionado na *GridView*. Daí como podemos verificar na (Fig. 19) o botão PDF já está visível.

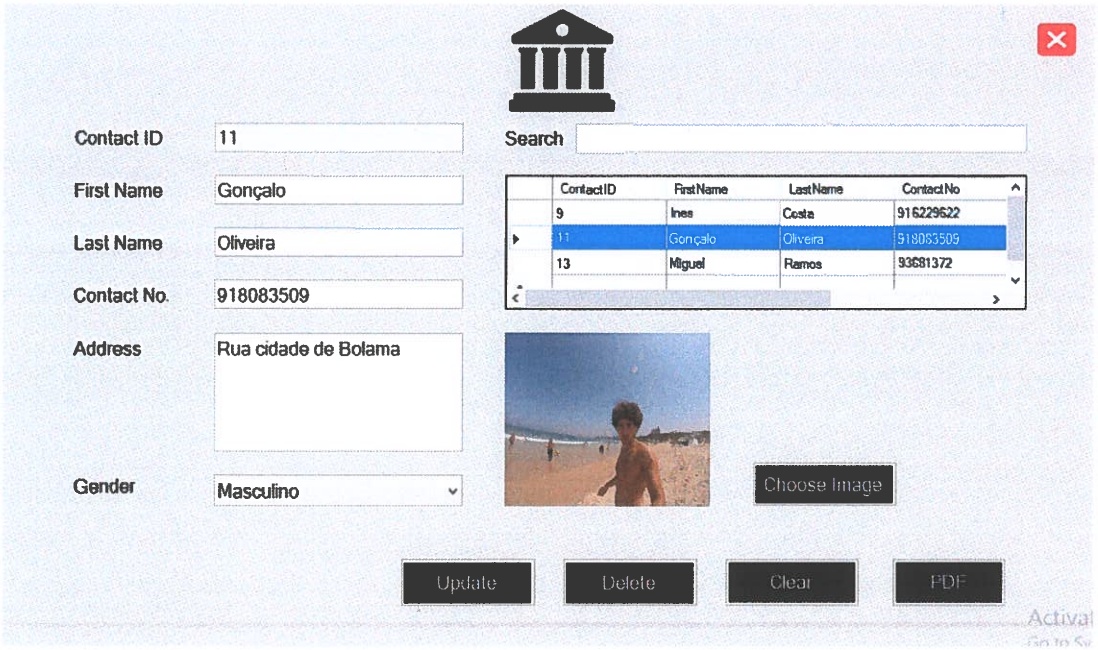

Figura 24. Janela com dados do utilizador

Nesta janela temos mais três botões, são eles:

Botão Update – Após o utilizador ser selecionado na *GridView*, os campos são automaticamente todos preenchidos, de acordo com os dados inseridos. De seguida o utilizador pode alterar os dados que pretende e para atualizar os dados basta clicar em Update.

```
try
{
    //Guardar a imagem em bytes na BD
    byte[] Image = null;
    FileStream fs = new FileStream(ImageBox.ImageLocation, FileMode.Open, FileAccess.Read);
    BinaryReader br = new BinaryReader(fs);
    Image = br.ReadBytes((int)fs.Length);

    //Recebe os dados das caixas de texto
    c.ContactID = int.Parse(txtBoxContactID.Text);
    c.FirstName = txtBoxFirstName.Text;
    c.LastName = txtBoxLastName.Text;
    c.ContactNo = txtBoxContactNo.Text;
    c.Address = txtBoxAddress.Text;
    c.Gender = cmbGender.Text;
    c.ImageBytes = Image;
    //Atualiza a Base de Dados
    c.Update(c);

    //Atualização bem sucedida
    MessageBox.Show("Contact has been seccessfully Updated.");
    //Carrega os dados na GridView
    DataTable dt = c.Select();
    dgvContactList.DataSource = dt;
    //Chama o metedo Clear
    Clear();
}
catch (Exception ex)
{
    //Erro ao atualizar
    MessageBox.Show("Failed to Updated Contact. Try again. \n Error: " + ex.ToString());
}
```

Figura 25. Código referente ao Update (Fig. 26)

Após a alteração dos dados volta a guardar a imagem em *bytes*, na Base de Dados, de seguida recebe os valores das caixas de texto e envia os novos valores através do metedo Update, da *class* "c". Se o processo for bem-sucedido aparece uma mensagem de sucesso no ecrã e os dados passam a estar visíveis na *GridView*. Posto isto é chamado o metedo *Clear* para limpar os campos. Se o código acima (Fig. 25) falhar aparecerá uma mensagem de erro a informar que o Update falhou e qual é o erro.

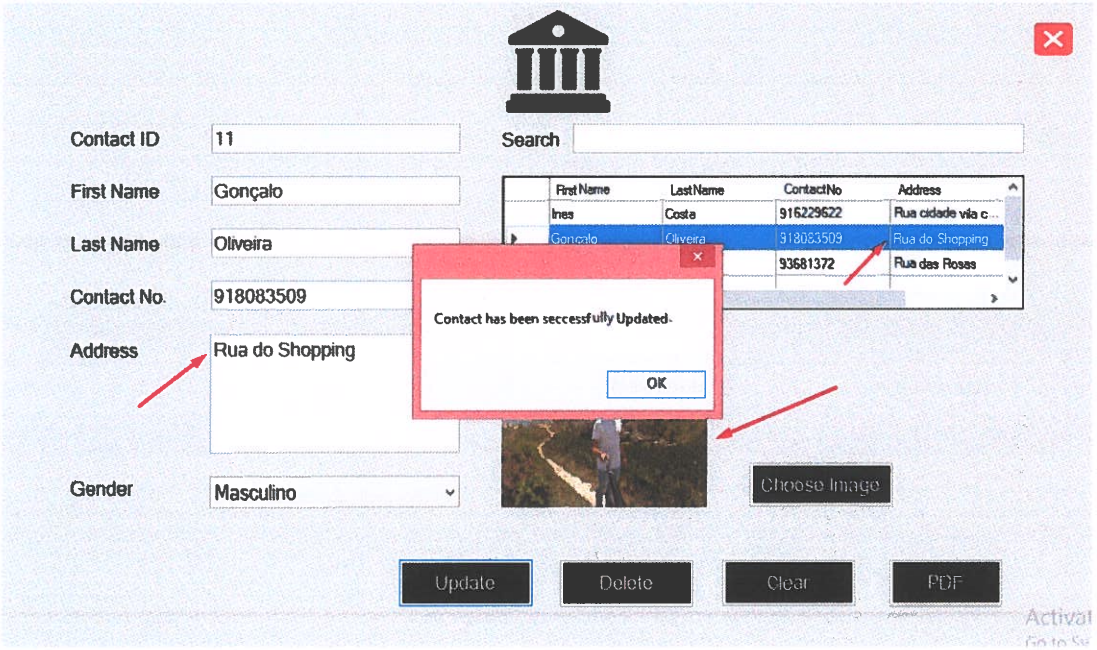

Figura 26. Atualização de morada e fotografia

Se o utilizador pretender atualizar a morada, tal como na imagem acima, poderá fazê-lo e em seguia a carregar no botão Update aparecerá uma mensagem no ecrã a informar que a atualização foi efetuada com sucesso.

Botão Delete – Com este botão podemos remover por completo e permanentemente o registo de um utilizador da Base de Dados.

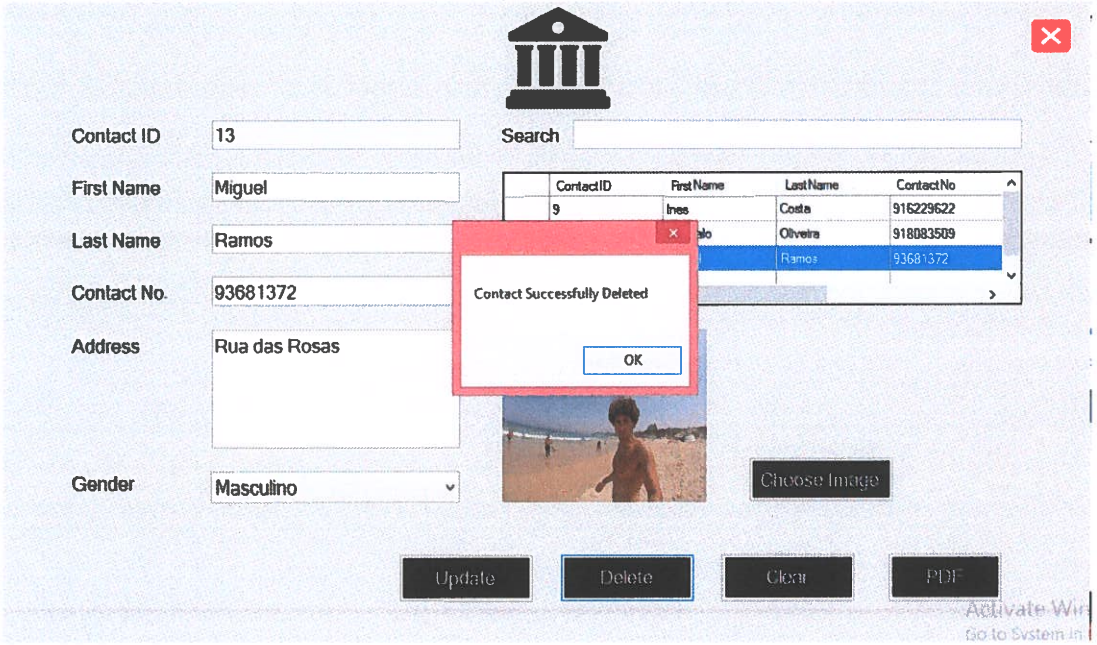

Figura 27. Mensagem de registo eliminado com sucesso

Quando o contacto é eliminado aparece uma mensagem no centro do ecrã a informar que o contacto foi eliminado com sucesso.

```
private void btnDelete_Click(object sender, EventArgs e)
{
    //Recebe o valor do ContactID da Base de Dados
    c.ContactID = Convert.ToInt32(txtBoxContactID.Text);
    bool success = c.Delete(c);
    if (success == true)
    {
        //Apagado com sucesso
        MessageBox.Show("Contact Successfully Deleted");
        //Atualiza a GridView
        //Carrega os dados na GridView
        DataTable dt = c.Select();
        dgvContactList.DataSource = dt;
        Clear();
    }
    else
    {
        //Erro ao apagar registo
        MessageBox.Show("Failed to Delete Contact. Try again.");
    }
}
```

Figura 28. Código referente ao Delete (Fig. 27)

Na figura 27 para o botão Delete funcionar, começa por receber o valor do ContactID, através da *class* "c", que por sua vez acede à Base de Dados para receber o valor do ContactID e identificar o utilizador que se pretende apagar. De seguida o utilizador é apagado, e se assim for aparecerá uma mensagem no ecrã a informar que o utilizador foi eliminado com sucesso. Posto isto atualiza a *GridView*, carrega os restantes dados e de seguida limpa as caixas de texto. Por fim se o código a cima (Fig. 28) não funcionar aparecerá uma mensagem de erro no ecrã.

Na imagem abaixo podemos verificar que o registo do Miguel Ramos já não é visível da *GridView* por já não existir na Base de Dados.

	ContactID	FirstName	LastName	ContactNo	A
▶	1	Gonçalo	Oliveira	918083509	Ru
	9	Ines	Costa	916229622	Ru
*					

Figura 29. GridView com o contacto apagado

Botão PDF – Permite ao utilizador guardar um PDF no seu computador com os dados do utilizador selecionado na *GridView*. O PDF contem todos os dados dos campos que estavam preenchidos, incluindo a fotografia.

Para criar um PDF basta clicar no botão PDF (Fig. 24) e de seguida abrir-se-á a janela onde poder-se-á guardar o PDF onde se pretender, como se pode verificar na imagem abaixo (Fig. 30). Esta máquina tem a DropBox instalada e a partir dessa ferramenta podemos extrair o documento para a máquina física. Apos guardar o documento aparecerá uma mensagem (Fig. 31) no centro do ecrã a informar que o PDF foi guardado com sucesso, e o caminho de onde foi guardado.

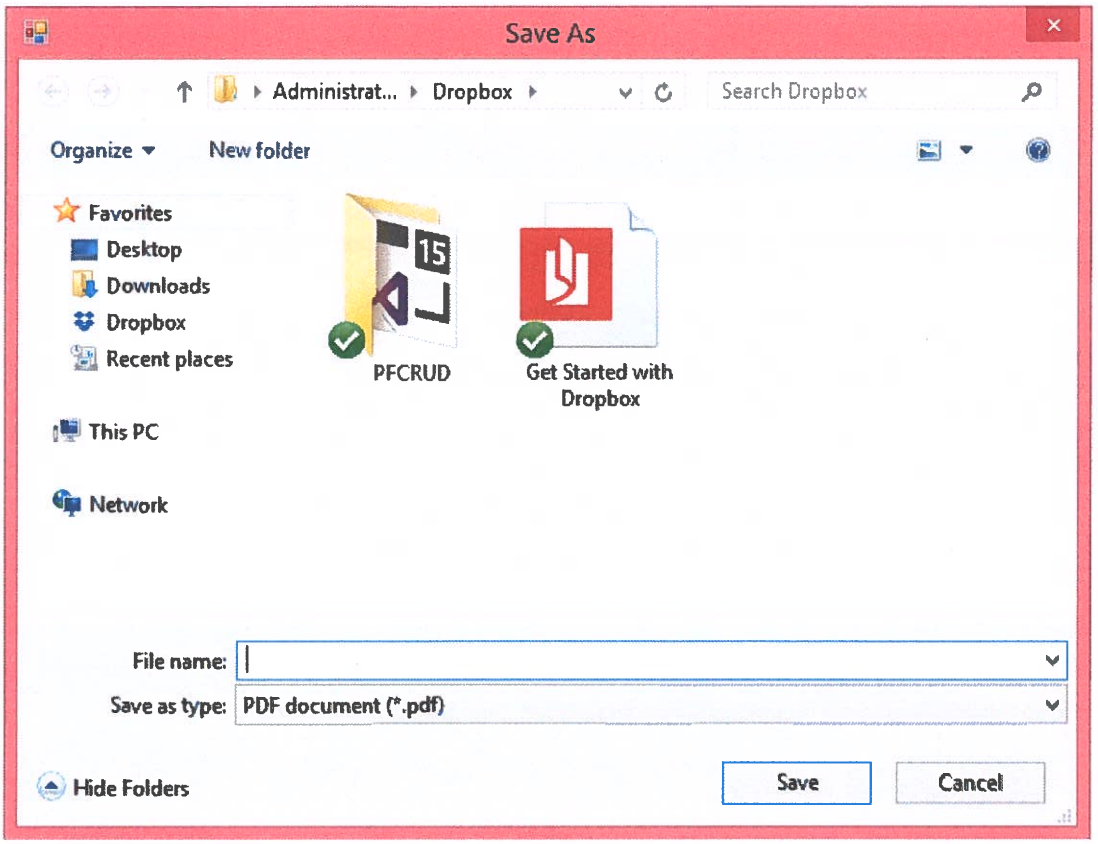

Figura 30. Janela para guardar o PDF

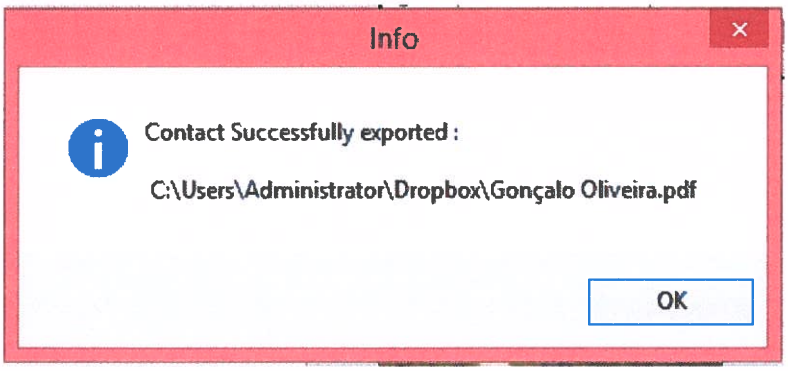

Figura 31. Mensagem de que o PDF foi guardado com sucesso

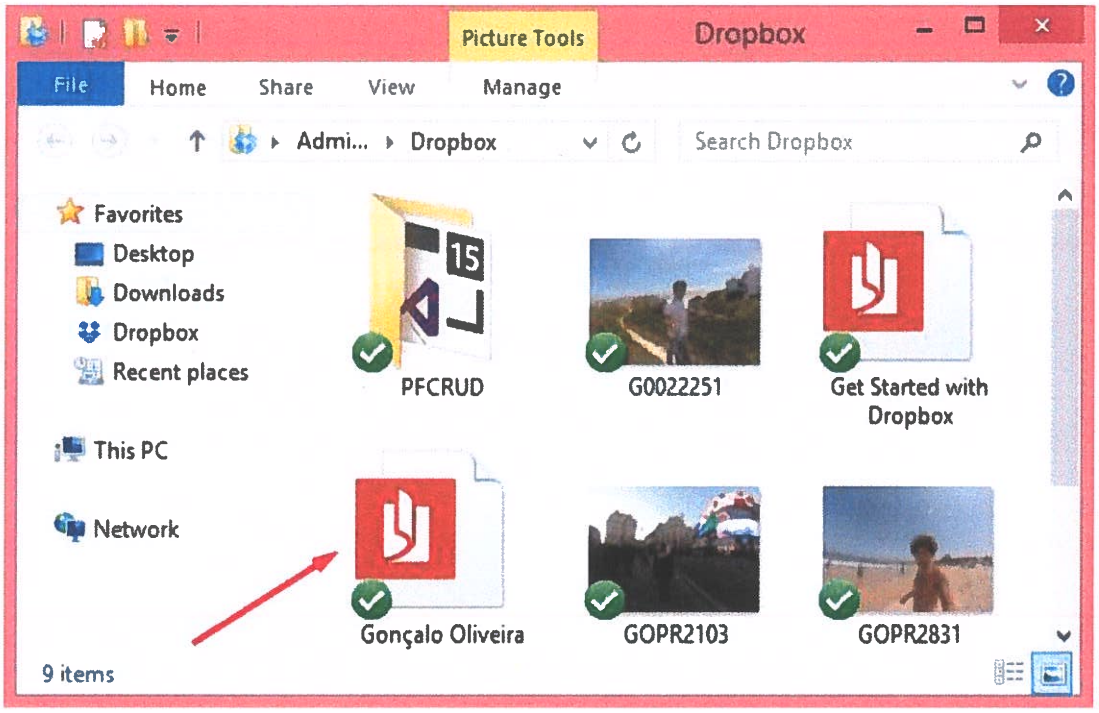

Figura 32. PDF guardado na pasta da DropBox

Como se pode observar na imagem acima (Fig. 32), o PDF já se encontra guardado numa pasta, neste caso na pasta da DropBox. De seguida como se pode verificar na imagem abaixo (Fig. 33), o PDF foi organizado da seguinte forma.

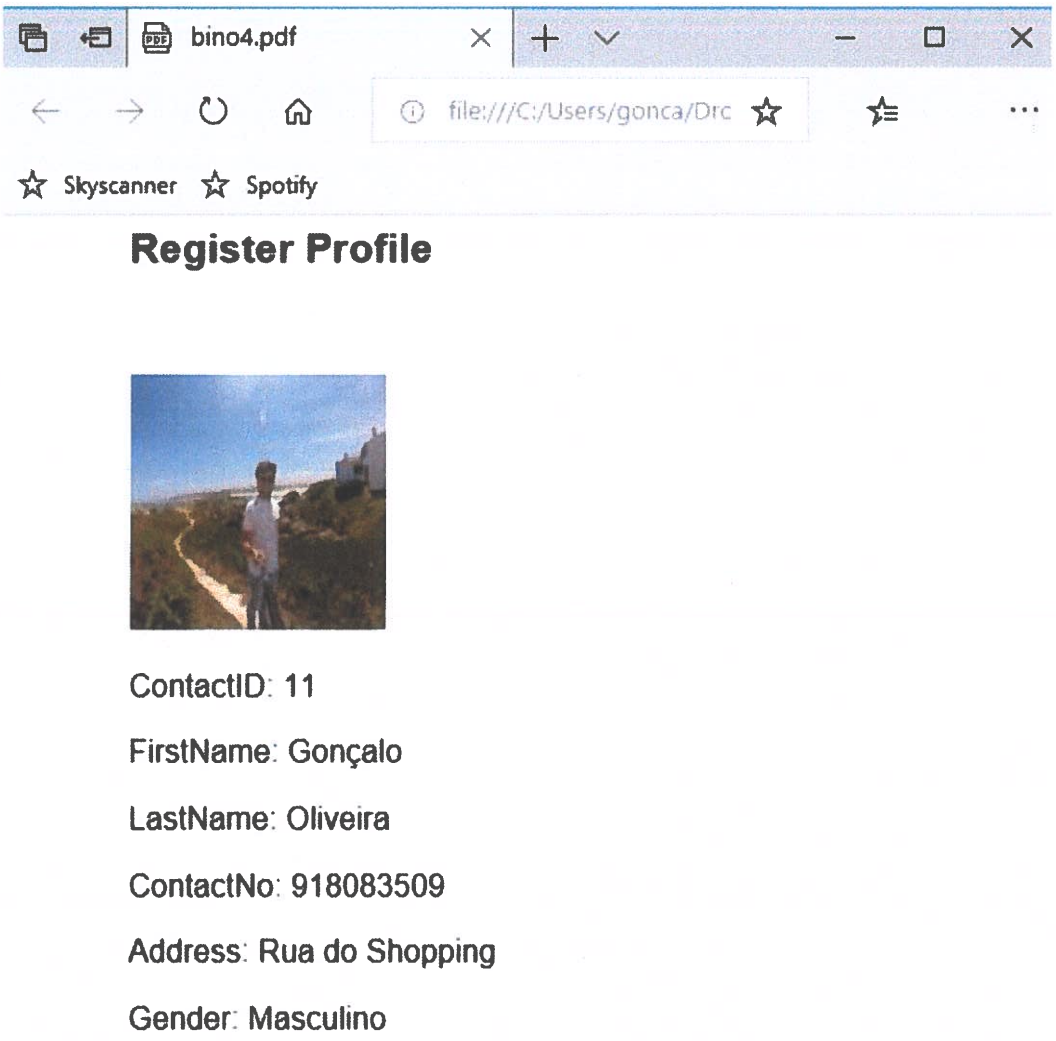

Figura 33. PDF aberto

Pdf exported at: 17/06/2018 23:15:49

Figura 34. PDF data/hora continuação (Fig. 33)

```
private void btnPDF_Click(object sender, EventArgs e)
{
    SaveFileDialog svg = new SaveFileDialog();
    svg.Filter = "PDF document (*.pdf)|*.pdf";
    DialogResult result = svg.ShowDialog();
```

Figura 35. Código apos clicar no botão PDF

Para extrair os campos da base de dados para um PDF, o utilizador ao clicar no botão PDF (Fig. 24) abrir-se-á um *OpenFileDialog* que permitirá ao utilizador escolher onde quer guardar o PDF e com que nome quer que seja guardado. O *Filter* diz-nos que o ficheiro será guardado no formato PDF (Fig. 35).

```
Document pdfDoc = new Document(PageSize.A4);
pdfDoc.SetMargins(50, 50, 50, 50);
PdfWriter.GetInstance(pdfDoc, stream);
pdfDoc.Open();
```

Figura 36. Código das especificações do PDF

No código da figura 36 pode-se observar as especificações do PDF, tais como o tamanho da página (A4), as margens (50, 50, 50, 50) e o método *Open*, para abrir o PDF. *pdfDoc* é a variável que guarda o documento.

```
//Inserir o titulo
Paragraph title = new Paragraph("Contact Profile");
title.SpacingAfter = 36f;
title.SpacingBefore = 36f;
title.Font = new iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.HELVETICA, 16f, iTextSharp.text.Font.BOLD);
pdfDoc.Add(title);
```

Figura 37. Código do titilo do PDF

No código da figura 37 é adicionado um paragrafo com margem antes e depois, de seguida atribui-se um estilo ao paragrafo e finalmente adiciona-se o paragrafo ao documento PDF através do método Add.

```
//Coloca a imagem no PDF
if (ImageBox.Image != null)
{
    iTextSharp.text.Image contact_pic = iTextSharp.text.Image.GetInstance(ImageBox.Image, System.Drawing.Imaging.ImageFormat.Jpeg);
    contact_pic.Alignment = Element.ALIGN_LEFT;
    contact_pic.ScaleAbsolute(100f, 100f);
    pdfDoc.Add(contact_pic);
}
```

Figura 38. Código para exportar a imagem para o PDF

Na figura 38 podemos observar que caso o registo não tenha imagem, não é adicionada nenhuma imagem ao documento PDF. Caso tenha imagem, é criada uma variável do tipo imagem, alinhada à esquerda com dimensões de 100x100. De seguida é adicionada à variável contact_pic.

```
//Exporta os dados para os campos
Paragraph dados = new Paragraph();
dados.SpacingBefore = 10f;
dados.Font = new iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.HELVETICA, 12f, iTextSharp.text.Font.NORMAL);
dados.Alignment = Element.ALIGN_LEFT;

dados.Add("ContactID: " + ContactID);
pdfDoc.Add(dados);
dados.Clear();
dados.Add("FirstName: " + FirstName);
pdfDoc.Add(dados);
dados.Clear();
dados.Add("LastName: " + LastName);
pdfDoc.Add(dados);
dados.Clear();
dados.Add("ContactNo: " + ContactNo);
pdfDoc.Add(dados);
dados.Clear();
dados.Add("Address: " + Address);
pdfDoc.Add(dados);
dados.Clear();
dados.Add("Gender: " + Gender);
pdfDoc.Add(dados);
dados.Clear();
dados.Add("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
pdfDoc.Add(dados);
Paragraph date = new Paragraph("PDF exported at: " + DateTime.Now.ToString());
pdfDoc.Add(date);
pdfDoc.Close();
stream.Close();

MessageBox.Show("Contact Successfully exported : \n \n " + svg.FileName, "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Figura 39. Código para exportar os dados das caixas de texto para PDF

Na figura 39 é possível verificar que é chamada a variável paragrafo, com uma margem de 10 no topo. É especificado o tipo e tamanho da letra e de seguida alinhado à esquerda. Podemos observar ainda que o ContactID é adicionado ao paragrafo, depois é adicionado ao documento PDF e por fim é apagado o seu conteúdo a partir do método *Clear*. Esta mesma abordagem é feita para os seguintes campos. Por fim é adicionada a data de exportação no final do documento, este é fechado através do método *Close*, assim como a *stream*. No final é apresentada uma mensagem a informar que o registo foi exportado com sucesso para o caminho inicialmente escolhido (Fig.30).

Base de Dados

Quanto à Base de Dados, é muito simples, tem os campos ContactID, FirstName, LastName, ContactNo, Address, Gender e Img (Imagem). Os únicos campos obrigatórios são o FirstName e o LastName, de resto são opcionais, como podemos ver na imagem abaixo.

Campo ContactID – Campo chave do tipo número inteiro com auto incremento. Serve como indentificador inequívoco de registo.

Campo FirstName – Campo do tipo texto com 50 caracteres máximo. Campo do primeiro nome do utilizador. Não suporta valores nulos.

Campo LastName – Campo do tipo texto com 50 caracteres máximo. Campo do último nome do utilizador. Não suporta valores nulos.

Campo ContactNo – Campo do tipo texto com 20 caracteres máximo. Campo do contacto do utilizador. Suporta valores nulos.

Campo Address – Campo do tipo texto com 255 caracteres máximo. Campo do endereço do utilizador. Suporta valores nulos.

Campo Gender – Campo do tipo texto com 10 caracteres máximo. Campo do género do utilizador. Suporta valores nulos.

Campo imagem – Campo do tipo imagem. Campo da imagem do utilizador. Suporta valores nulos.

Column Name	Data Type	Allow Nulls
ContactID	int	☐
FirstName	varchar(50)	☐
LastName	varchar(50)	☐
ContactNo	varchar(20)	☑
Address	varchar(255)	☑
Gender	varchar(10)	☑
Img	image	☑

Figura 40. Design da Base de Dados

Conclusão

A aplicação gestão de utilizadores, cujo seu desenvolvimento, implementação e utilização foi descrita neste documento, apresenta uma forma consistente do trabalho cooperativo em rede utilizando as tecnologias de informação e seus recursos para gerir os seus utilizadores. A aplicação permite armazenar e disponibilizar de forma detalhada todos os utilizadores nela inseridos, assim como os seus dados pessoais.

Conseguiu-se minimizar os elevados custos financeiros que uma estrutura desta requer ao nível do *hardware* recorrendo à virtualização. Todo o *software* utilizado que foi descrito neste projeto para suportar a aplicação, engloba as mais recentes tecnologias, ferramentas e as suas versões são bastante estáveis, desta forma a aplicação pode ser executada de forma clara e segura.

Referências bibliográficas

Jones, S. (2002). *Encyclopedia of New* Media*: Na Essential Reference to Communication. And.*

Kriegel, A., & Trukhnov, B. M. (2008). *SQL Bible*

Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., & Postel, j. (2000). ABrief History of the Internet, version 3.31. Institute for Information system and.

Leiner, R, Mills, B., & Postel, D. (1985). *The DARPA Internet Protocol.* Washinton D.C.

McQuillan. (1975). *The Evolution of Message Processing Techniques in the ARPA Network.* John Network Systems and Software Infotech State of the Art Report.

Poole, H. W., Lambert, L., Woodford, C., Moschovitis, C. J., & Barbara, S. (2005). *The Internet: A Historical Encyclopedia.* California: ABC-CLIO.

Portnoy, M. (2012). *Virtualization Essentials.*

Proven, L. (2011). *A Brief History of Virtualisation.*

Sceppa, D. (2006). *Programming Microsoft ADO.NET 2.0.*

Anexos e apêndices

Adlogin.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.DirectoryServices;

namespace PFCRUD
{
    public partial class adlogin : Form
    {
        public adlogin()
        {
            InitializeComponent();
        }
        private void btnSubmit_Click(object sender, EventArgs e)
        {
            string path = @"LDAP://DC.OLIVEIRA.LOCAL";
            string domain = @"oliveira.local";
            string user = txtBoxUser.Text.Trim();
            string pass = txtBoxPassWord.Text.Trim();
            string domUsu = domain + @"\" + user;

            bool permission = UserAuthentication(path, domUsu, pass);
            if (permission)
            {
                this.Hide();
                Form1 form = new Form1();
                form.ShowDialog();
                errorlbl.Visible = false;
            }
            else
            {
                errorlbl.Visible = true;
            }
        }
        private bool UserAuthentication(String path, String domUsu, String pass)
        {
            DirectoryEntry de = new DirectoryEntry(path, domUsu, pass,
AuthenticationTypes.Secure);
            try
            {
                DirectorySearcher ds = new DirectorySearcher(de);
                ds.FindOne();
                return true;
            }
            catch
            {
                return false;
            }
        }
    }
}
```

Form1.cs

```
using PFCRUD.PFClasses;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using iTextSharp;
using iTextSharp.text;
using iTextSharp.text.pdf;

namespace PFCRUD
{
   public partial class Form1 : Form
   {
      string LocImage = "";

      string ContactID;
      string FirstName;
      string LastName;
      string ContactNo;
      string Address;
      string Gender;
```

```
public Form1()
{
   InitializeComponent();
}
PFCRUDClass c = new PFCRUDClass();

private void Form1_Load(object sender, EventArgs e)
{
   //Carrega os dados na GridView
   DataTable dt = c.Select();
   dgvContactList.DataSource = dt;
   dgvContactList.ClearSelection();
   btnPDF.Visible = false;
   btnUpdate.Visible = false;
   btnDelete.Visible = false;
}

private void btnAdd_Click(object sender, EventArgs e)
{
   try
   {
      //Guardar a imagem em bytes na Base Dados
      byte[] Image = null;
      if (LocImage != "")
      {

         FileStream fs = new FileStream(LocImage, FileMode.Open, FileAccess.Read);
         BinaryReader br = new BinaryReader(fs);
         Image = br.ReadBytes((int)fs.Length);
      }
      //Obtem os valores dos campos inseridos
      c.FirstName = txtBoxFirstName.Text;
```

```
            c.LastName = txtBoxLastName.Text;
            c.ContactNo = txtBoxContactNo.Text;
            c.Address = txtBoxAddress.Text;
            c.Gender = cmbGender.Text;
            c.ImageBytes = Image;
            //Insere os dados na Base de Dados
            c.Insert(c);
            //Inserido com sucesso
            MessageBox.Show("New Contact Successfully Inserted");
            //Chama o metedo Clear
            Clear();
            //Carrega os dados na GridView
            DataTable dt = c.Select();
            dgvContactList.DataSource = dt;
            LocImage = "";
        }
        catch (Exception ex)
        {
            //Erro a adicionar novo registo
            MessageBox.Show("Failed to add New Contact. Try again! \n Error: " +
ex.ToString());
        }
    }

    private void pictureBox2_Click(object sender, EventArgs e)
    {
        this.Close();
    }
    //Metedo para limpar os campos
    public void Clear()
    {
        txtBoxFirstName.Text = "";
```

```
            txtBoxLastName.Text = "";
            txtBoxContactNo.Text = "";
            txtBoxAddress.Text = "";
            cmbGender.Text = "";
            txtBoxContactID.Text = "";
            ImageBox.Image = null;
            btnAdd.Visible = true;
            btnUpdate.Visible = false;
            btnDelete.Visible = false;
            dgvContactList.ClearSelection();
        }

        private void btnUpdate_Click(object sender, EventArgs e)
        {
            try
            {
                //Guardar a imagem em bytes na BD
                byte[] Image = null;
                FileStream fs = new FileStream(ImageBox.ImageLocation, FileMode.Open,
FileAccess.Read);
                BinaryReader br = new BinaryReader(fs);
                Image = br.ReadBytes((int)fs.Length);

                //Recebe os dados das caixas de texto
                c.ContactID = int.Parse(txtBoxContactID.Text);
                c.FirstName = txtBoxFirstName.Text;
                c.LastName = txtBoxLastName.Text;
                c.ContactNo = txtBoxContactNo.Text;
                c.Address = txtBoxAddress.Text;
                c.Gender = cmbGender.Text;
                c.ImageBytes = Image;
                //Atualiza a Base de Dados
```

```
            c.Update(c);

            //Atualização bem sucedida
            MessageBox.Show("Contact has been seccessfully Updated.");
            //Carrega os dados na GridView
            DataTable dt = c.Select();
            dgvContactList.DataSource = dt;
            //Chama o metedo Clear
            Clear();
        }
        catch (Exception ex)
        {
            //Erro ao atualizar
            MessageBox.Show("Failed to Updated Contact. Try again. \n Error: " +
ex.ToString());
        }
    }

    private void dgvContactList_RowHeaderMouseClick(object sender,
DataGridViewCellMouseEventArgs e)
    {
        //Identifica a linha na GridView na qual foi clicada
        //Obtem os dados da linha selecionada da GridView e coloca-los nos campos
respetivos
        int rowIndex = e.RowIndex;
        txtBoxContactID.Text = dgvContactList.Rows[rowIndex].Cells[0].Value.ToString();
        txtBoxFirstName.Text = dgvContactList.Rows[rowIndex].Cells[1].Value.ToString();
        txtBoxLastName.Text = dgvContactList.Rows[rowIndex].Cells[2].Value.ToString();
        txtBoxContactNo.Text = dgvContactList.Rows[rowIndex].Cells[3].Value.ToString();
        txtBoxAddress.Text = dgvContactList.Rows[rowIndex].Cells[4].Value.ToString();
        cmbGender.Text = dgvContactList.Rows[rowIndex].Cells[5].Value.ToString();
        ImageBox.Image = c.Selectimg(txtBoxContactID.Text);
```

```
    ContactID = dgvContactList.Rows[rowIndex].Cells[0].Value.ToString();
    FirstName = dgvContactList.Rows[rowIndex].Cells[1].Value.ToString();
    LastName = dgvContactList.Rows[rowIndex].Cells[2].Value.ToString();
    ContactNo = dgvContactList.Rows[rowIndex].Cells[3].Value.ToString();
    Address = dgvContactList.Rows[rowIndex].Cells[4].Value.ToString();
    Gender = dgvContactList.Rows[rowIndex].Cells[5].Value.ToString();

    btnAdd.Visible = false;
    btnUpdate.Visible = true;
    btnDelete.Visible = true;
    btnPDF.Visible = true;
}

private void btnClear_Click(object sender, EventArgs e)
{
    //Chama o metedo clear para o evento do botão
    Clear();
    btnPDF.Visible = false;
}

private void btnDelete_Click(object sender, EventArgs e)
{
    //Recebe o valor do ContactID da Base de Dados
    c.ContactID = Convert.ToInt32(txtBoxContactID.Text);
    bool success = c.Delete(c);
    if (success == true)
    {
        //Apagado com sucesso
        MessageBox.Show("Contact Successfully Deleted");
        //Atualiza a GridView
        //Carrega os dados na GridView
        DataTable dt = c.Select();
```

```
            dgvContactList.DataSource = dt;
            Clear();
        }
        else
        {
            //Erro ao apagar registo
            MessageBox.Show("Failed to Delete Contact. Try again.");
        }
    }
    //Conecta à Base Dados
    static string myconnstr =
ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;
    private void textBoxSearch_TextChanged(object sender, EventArgs e)
    {
        //Recebe os dados das caixas de texto
        string keyword = txtboxSearch.Text;

        SqlConnection conn = new SqlConnection(myconnstr);
        SqlDataAdapter sda = new SqlDataAdapter("SELECT ContactID, FirstName,
LastName, ContactNo, Address, Gender FROM tbl_contact WHERE FirstName LIKE '%" +
keyword + "%' OR LastName LIKE '%" + keyword + "%'OR Address LIKE '%" + keyword
+ "%'", conn);
        DataTable dt = new DataTable();
        sda.Fill(dt);
        dgvContactList.DataSource = dt;
    }

    private void btnImage_Click(object sender, EventArgs e)
    {
        try
        {
            OpenFileDialog dlg = new OpenFileDialog();
            dlg.Filter = "JPG Files (*.jpg)|*.jpg| PNG Files (*.png)|*png|All Files (*.*)|";
```

```
            if (dlg.ShowDialog() == DialogResult.OK)
            {
                LocImage = dlg.FileName.ToString();
                ImageBox.ImageLocation = LocImage;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void btnPDF_Click(object sender, EventArgs e)
    {
        SaveFileDialog svg = new SaveFileDialog();
        svg.Filter = "PDF document (*.pdf)|*.pdf";
        DialogResult result = svg.ShowDialog();
        if (result == DialogResult.OK)
        {
            using (FileStream stream = new FileStream(svg.FileName, FileMode.Create))
            {
                Document pdfDoc = new Document(PageSize.A4);
                pdfDoc.SetMargins(50, 50, 50, 50);
                PdfWriter.GetInstance(pdfDoc, stream);
                pdfDoc.Open();

                //Inserir o titulo
                Paragraph title = new Paragraph("Contact Profile");
                title.SpacingAfter = 36f;
                title.SpacingBefore = 36f;
                title.Font = new
iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.HELVETICA, 16f,
iTextSharp.text.Font.BOLD);
```

```
            pdfDoc.Add(title);


            //Coloca a imagem no PDF
            if (ImageBox.Image != null)
            {
               iTextSharp.text.Image contact_pic =
iTextSharp.text.Image.GetInstance(ImageBox.Image,
System.Drawing.Imaging.ImageFormat.Jpeg);
               contact_pic.Alignment = Element.ALIGN_LEFT;
               contact_pic.ScaleAbsolute(100f, 100f);
               pdfDoc.Add(contact_pic);
            }
            //Exporta os dados para os campos
            Paragraph dados = new Paragraph();
            dados.SpacingBefore = 10f;
            dados.Font = new
iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.HELVETICA, 12f,
iTextSharp.text.Font.NORMAL);
            dados.Alignment = Element.ALIGN_LEFT;


            dados.Add("ContactID: " + ContactID);
            pdfDoc.Add(dados);
            dados.Clear();
            dados.Add("FirstName: " + FirstName);
            pdfDoc.Add(dados);
            dados.Clear();
            dados.Add("LastName: " + LastName);
            pdfDoc.Add(dados);
            dados.Clear();
            dados.Add("ContactNo: " + ContactNo);
            pdfDoc.Add(dados);
            dados.Clear();
            dados.Add("Address: " + Address);
```

```
                pdfDoc.Add(dados);
                dados.Clear();
                dados.Add("Gender: " + Gender);
                pdfDoc.Add(dados);
                dados.Clear();
                dados.Add("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
                pdfDoc.Add(dados);
                Paragraph date = new Paragraph("PDF exported at: " +
DateTime.Now.ToString());
                pdfDoc.Add(date);
                pdfDoc.Close();
                stream.Close();

                MessageBox.Show("Contact Successfully exported : \n \n " + svg.FileName,
"Info", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }
  }
}
```

PFCRUDClass.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Drawing;

namespace PFCRUD.PFClasses
{
    class PFCRUDClass
    {
        //Transporta os dados da aplicação
```

```
        public int ContactID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string ContactNo { get; set; }
        public string Address { get; set; }
        public string Gender { get; set; }

        public Byte[] ImageBytes { get; set; }
        string LocImage = "";

        static string myconnstrng =
ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;
        SqlConnection conn = new SqlConnection(myconnstrng);
        //Seleciona os dados sa Base de Dados
        //Carrega os dados na gridview
        public DataTable Select()
        {
            //Declara a variavel do tipo DataTable
            DataTable dt = new DataTable();
            try
            {
                //Escreve uma SQL Query
                string sql = "SELECT ContactID, FirstName, LastName, ContactNo,
Address, Gender FROM tbl_contact";
                //Cria um comando que usa o SQL conection (conn)
                SqlCommand cmd = new SqlCommand(sql, conn);
                //Cria uma instancia da class SQL DataAdapter
                SqlDataAdapter adapter = new SqlDataAdapter(cmd);
                conn.Open();
                adapter.Fill(dt);
            }
            catch (Exception ex)
            {

            }
            finally
            {
                conn.Close();
            }
            return dt;
        }
        public Image Selectimg(string ContactID)
        {
            string sql = "SELECT Img FROM tbl_contact WHERE ContactID = @ContactID";
            //Cria um comando que usa o SQL e o connection (conn)
            SqlCommand cmd = new SqlCommand(sql, conn);
            cmd.Parameters.AddWithValue("@ContactID", Convert.ToInt32(ContactID));
            conn.Open();
            var img = cmd.ExecuteScalar();
            conn.Close();
            Byte[] ImageByte;
            Image imgFromByte;
            if (img != DBNull.Value)
            {
                ImageByte = (byte[])(img);
                MemoryStream ms = new MemoryStream(ImageByte);
                imgFromByte = System.Drawing.Image.FromStream(ms);
                return System.Drawing.Image.FromStream(ms);
            }
            imgFromByte = null;
            ImageByte = null;
            return null;
```

```
        }
        //Inserir dados na Base de Dados
        public void Insert(PFCRUDClass c)
        {
            //Conecta à Base de Dados
            SqlConnection conn = new SqlConnection(myconnstrng);
            //Cria uma SQL Query para inserir dados
            string sql = "INSERT INTO tbl_contact (FirstName, LastName, ContactNo,
Address, Gender, Img) VALUES (@FirstName, @LastName, @ContactNo, @Address, @Gender,
@Img)";
            //Cria um comando em SQL usando sql e o conn
            SqlCommand cmd = new SqlCommand(sql, conn);
            //Cria parametro para adicionar dados
            cmd.Parameters.AddWithValue("@FirstName", c.FirstName);
            cmd.Parameters.AddWithValue("@LastName", c.LastName);
            cmd.Parameters.AddWithValue("@ContactNo", c.ContactNo);
            cmd.Parameters.AddWithValue("@Address", c.Address);
            cmd.Parameters.AddWithValue("@Gender", c.Gender);
            if (c.ImageBytes == null)
            {
                cmd.Parameters.AddWithValue("@Img", DBNull.Value);
            }
            else
            {
                cmd.Parameters.AddWithValue("@Img", c.ImageBytes);
            }

            //Abre a conecção
            conn.Open();
            cmd.ExecuteNonQuery();
            conn.Close();
        }

        //Método para atualizar os dados na Base de Dados
        public void Update(PFCRUDClass c)
        {
            //Cria um valor de retorno por defeito e o define como falso
            SqlConnection conn = new SqlConnection(myconnstrng);

            //Comando de SQL para atualizar os dados na Base de Dados
            string sql = "UPDATE tbl_contact SET FirstName=@FirstName,
LastName=@LastName, ContactNo=@ContactNo, Address=@Address, Gender=@Gender, Img=@Img
WHERE ContactID=@ContactID";

            //Cria um comando SQL
            SqlCommand cmd = new SqlCommand(sql, conn);
            //Crias os parametro para adicional os dados
            cmd.Parameters.AddWithValue("@FirstName", c.FirstName);
            cmd.Parameters.AddWithValue("@LastName", c.LastName);
            cmd.Parameters.AddWithValue("@ContactNo", c.ContactNo);
            cmd.Parameters.AddWithValue("@Address", c.Address);
            cmd.Parameters.AddWithValue("@Gender", c.Gender);
            cmd.Parameters.AddWithValue("@ContactID", c.ContactID);
            cmd.Parameters.AddWithValue("@Img", c.ImageBytes);
            conn.Open();
            cmd.ExecuteNonQuery();
            conn.Close();
        }
        //Método para apagagar dados da Base de Dados
        public bool Delete(PFCRUDClass c)
        {
            //Cria um valor de retorno por defeito e o define como falso
```

```
            bool isSuccess = false;
            //Conecta à Base de Dados
            SqlConnection conn = new SqlConnection(myconnstrng);
            try
            {
                //Comando para apagar os dados na Base de Dados
                string sql = "DELETE FROM tbl_contact WHERE ContactID=@ContactID";

                //Cria um comando SQL
                SqlCommand cmd = new SqlCommand(sql, conn);
                cmd.Parameters.AddWithValue("@ContactID", c.ContactID);
                conn.Open();
                int rows = cmd.ExecuteNonQuery();
                //Se a consulta for executada com sucesso, o valor será maior que zero
                if (rows > 0)
                {
                    isSuccess = true;
                }
                else
                {
                    isSuccess = false;
                }
            }
            catch (Exception ex)
            {


            }
            finally
            {
                conn.Close();
            }
            return isSuccess;
        }
    }
}
```