
Universidade de Aveiro
2014
Departamento de Eletrónica Telecomunicações e Informática

VICTÓRIA CAROLINA CERQUEIRA PINTO DA CRUZ

DESENHO DE SOLUÇÕES DE ALTA DISPONIBILIDADE EM CENÁRIOS DE CATÁSTROFE


Universidade de Aveiro
2014
Departamento de Eletrónica Telecomunicações e Informática

VICTÓRIA CAROLINA CERQUEIRA PINTO DA CRUZ

DESENHO DE SOLUÇÕES DE ALTA DISPONIBILIDADE EM CENÁRIOS DE CATÁSTROFE

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Cláudio Teixeira, equiparado a Investigador Auxiliar do Departamento de Eletrónica Telecomunicações e Informática da Universidade de Aveiro e do Dr. Joaquim Manuel Henriques de Sousa Pinto, Professor auxiliar do Departamento de Eletrónica Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho aos meus pais e irmã bem como ao resto da minha família e amigos pelo constante apoio.

o júri

presidente

Prof. Dr. Joaquim Arnaldo Carvalho Martins
Professor Catedrático, Universidade de Aveiro

Prof. Dr. Fernando Joaquim Lopes Moreira
Professor Associado, Departamento de Inovação, Ciência e Tecnologia da Universidade Portucalense

Prof. Dr. Cláudio Jorge Vieira Teixeira
Equiparado a Investigador Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos

Gostaria de manifestar os meus mais sinceros agradecimentos a todos que de forma direta ou indireta contribuíram para a realização deste trabalho.

Ao meu orientador, Professor Cláudio Teixeira pelo apoio e interesse demonstrado ao longo da elaboração desta dissertação.

Ao meu coorientador, Professor Joaquim Sousa Pinto pela sua disponibilidade e pelo apoio que depositou neste trabalho.

Em especial à minha família amigos e colegas, pela compreensão, carinho, motivação e pela força nos momentos de desânimo.

palavras-chave

Desenho de soluções, alta disponibilidade, *cloud*, cenários de catástrofe, *back office*, sistemas críticos, *software, hardware,* falhas, *unplanned downtimes, anti-crash*.

resumo

Nos dias de hoje, as catástrofes têm sido mais regulares provocando muitas vezes enormes destruições materiais e diversas perdas económicas. Devido a isto, qualquer cenário de catástrofe tem de ser tido em conta no que toca a elaboração de um sistema considerado crítico. A probabilidade de falhas ocorrerem neste tipo de sistemas depende de onde os servidores de suporte estiverem localizados, mas a utilização destes noutros locais ou a sua deslocação nem sempre é uma alternativa concebível.

Esta dissertação incide sobre estudar e propor uma metodologia de sistemas críticos que permita mitigar as falhas de sistemas e infraestruturas, permitindo aos utilizadores continuar a usar o sistema ainda que de modo condicionado ao longo da fase de falha. Para isso foi estudada uma forma ampla para que fosse desenhada e posteriormente desenvolvida uma arquitetura de controlo que abrangesse diversos cenários de catástrofe. Nesta arquitetura são tidos em conta quatro casos de uso e também é demonstrado nesta dissertação como cada um destes é solucionado. Foi também elaborado uma plataforma de suporte *back office* para os utilizadores desta arquitetura, de forma a estes terem um controlo sobre os seus dados, bem como fornecerem informação necessária para o correto funcionamento desta.

keywords

Design of solutions, high availability, cloud, disaster scenarios, back office, critical systems, failures, software, hardware, unplanned downtimes, anti-crash.

abstract

Nowadays, disasters have been more regular often causing huge material destruction and diverse economic losses. Due to this, disaster scenarios have to be taken into consideration when it comes to developing a system considered critical. The probability of failures occurring in such systems depends on where the supporting servers are located, but the use of other locations or their displacement is not always a conceivable alternative.

This dissertation focuses on studying and proposing a methodology for critical systems that allows the mitigation of systems and infrastructure failures, allowing users to continue using the system even if partially conditioned during the phase of failure. To address this issue, it was studied a broad way to design and subsequently develop an architecture of control that could covers several disaster scenarios.

In this architecture there are four use cases taken into account as well as how they are should be addressed. It was also devised a back office support platform for the users of this methodology, so they are able to have control over their data, and consequently provide necessary information for its proper functioning.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

1. Introdução

Nos dias de hoje as falhas nos sistemas de informação críticos causam transtornos ocasionando muitas das vezes perdas económicas [1]. A disponibilização destes pressupõe a existência de equipamentos e infraestruturas capazes de assegurar o seu funcionamento correto na maioria dos cenários. Mas dependendo da localização física dos servidores de suporte, a probabilidade de falhas graves ao nível de infraestrutura pode aumentar de forma considerável. A mera deslocação física dos servidores, ou a utilização dos mesmos em outros locais com garantias de serviços melhores nem sempre é uma opção possível. Para especificar um sistema destes, é preciso compreender os riscos gerando também requisitos de confiabilidade para lidar com os mesmos. É no entanto, preciso saber identificar os riscos para poder aplicar a cada tipo de sistema.

Contudo num cenário de catástrofe, os riscos apresentados são maiores, surgindo assim a ideia de se propor uma solução enquadrada num cenário desta natureza, uma metodologia de controlo de sistemas críticos que permita mitigar as falhas de sistemas e infraestruturas, possibilitando aos utilizadores continuar a usar o sistema, ainda que de forma condicionada, durante os períodos de falha.

1.1 Motivação

A confiança de um sistema é uma qualidade indispensável e é esperada em sistemas considerados críticos. A confiabilidade implica disponibilidade, que é outro aspeto crucial neste tipo de sistema, e é precisamente a alta disponibilidade que vai permitir a continuidade das operações destes sistemas mesmo havendo falhas em um ou mais dos seus elementos aumentando assim a confiança dos utilizadores.

Quando estas operações param, consequentemente os lucros que estas fornecem também ficam suspensos, continuando a existir no entanto a necessidade de se proporcionar níveis crescentes de disponibilidade o que leva muitas das empresas a reestruturarem as suas soluções de forma a ganhar vantagem competitiva [2].

O tempo de inatividade pode levar à perda de produtividade, receitas, má publicidade, sem contar que o relacionamento com os clientes fica afetado [2] sendo que contornar estes obstáculos é a grande prioridade para quem fornece esse tipo de sistemas.

1.2 Objetivos

Esta solução de suporte será desenvolvida com o intuito de mitigar todas as falhas existentes num cenário de catástrofe permitindo aos utilizadores o uso do sistema sem que haja interrupções, ainda que de forma condicionada, diminuindo assim de certa forma alguns custos que estas falhas possam provocar. Sendo este o objetivo principal, para melhor conceção desta solução é relevante que esta se potencialize sendo crucial atender pontos como [1]:

- **Disponibilidade** – permitindo assim que esteja pronto a ser utilizado a todo momento, com a capacidade de disponibilizar serviços quando solicitado;
- **Confiabilidade** – fornecendo desta forma todos os serviços esperados pelos utilizadores conforme os mesmos foram especificados;
- **Segurança** – a solução tem de ser segura, tem de refletir a habilidade do sistema operar de forma condicionada ou não, sem ameaçar o ambiente;
- **Proteção** – refletindo a sua habilidade de proteger-se de um ataque externo;

Pontos estes que por si só já fazem parte da estrutura de um sistema considerado crítico porém esta solução também tem de atender aspetos como:

- **Rigor** – esta tem de ser uma solução que apresenta um padrão conciso, fazendo assim com que os sistemas que fizerem o uso desta se adaptem a este padrão, atendendo assim as condições necessárias para que o seu uso seja bem-sucedido.
- **Controlo** – nesta solução é permitido ao utilizador ter o controlo dos seus dados e informações, apresentando assim uma plataforma de suporte *back office* para que os utilizadores tenham acesso e domínio dos seus dados;

1.3 Estrutura da Dissertação

A presente dissertação está dividida em seis capítulos, sendo este o capítulo 1, onde é feita uma breve contextualização do projeto assim como a apresentação dos seus principais objetivos, também é feita uma pequena introdução a área em que este projeto se enquadra. No capítulo 2 são citados mecanismos, aplicações e ainda temas relacionados com a elaboração deste projeto. No capítulo 3 é feita uma visão geral deste projeto, são apresentados os casos de uso e como estes devem ser solucionados assim como a arquitetura geral relacionada a este projeto bem como as suas especificações e como estes casos de uso fluem nesta mesma arquitetura. No capítulo seguinte são referidos as condições necessárias para a utilização desta solução assim como é

apresentada a plataforma de suporte a esta, bem como as funcionalidades que dispõe. No capítulo 5 são descritos os aspetos relacionados com a implementação desta arquitetura e os testes que foram realizados para garantir o cumprimento dos objetivos. No capítulo 6 será apresentada uma conclusão da presente dissertação e uma breve descrição dos problemas encontrados ao longo do desenvolvimento desta solução e por fim é enunciada uma proposta de trabalho futuro para esta dissertação.

1.4 Sistemas Críticos e Ambientes Computacionais de Alta Disponibilidade

Com o progresso das tecnologias de informação em praticamente todas as áreas de negócio, a dependência aos sistemas aumentou consideravelmente surgindo assim o conceito de sistemas críticos.

Sistemas críticos são sistemas técnicos dos quais as pessoas dependem. São sistemas cujas principais características são: a confiabilidade, a disponibilidade, a proteção e a segurança [3]. Existem três tipos de sistemas críticos:

1. **Sistema crítico de segurança**, em que a sua falha pode proceder em prejuízos, danos ambientais bem como a perda da vida humana;
2. **Sistema crítico de missão**, em que a sua falha pode ocasionar problemas em atividades que contenham objetivos como por exemplo um sistema de navegação de uma nave espacial;
3. **Sistema crítico de negócio**, onde a sua falha pode resultar em custos elevados para a empresa que trabalha com o *software*.

A solução aqui apresentada se adapta melhor ao sistema crítico de negócio. Num sistema crítico, é preciso focar em todos os seus aspetos, tais como: *hardware, software* e processos operacionais, uma vez que qualquer falha pode causar problemas graves no futuro. Para além disso, é necessário pensar também na confiança visto que ela é fundamental no sistema considerado crítico [1].

1.5 Confiança no Sistema

A confiança em um sistema pode ser tida, como a confiança dos utilizadores em que o sistema não irá falhar. Como já referido acima, um sistema crítico tem de ser de confiança tanto é que as suas principais características bem como as dimensões para o sistema ser confiável são a disponibilidade, confiabilidade, segurança e proteção [3]. A Figura 1 mostra as propriedades de confiança de forma inter-relacionada.

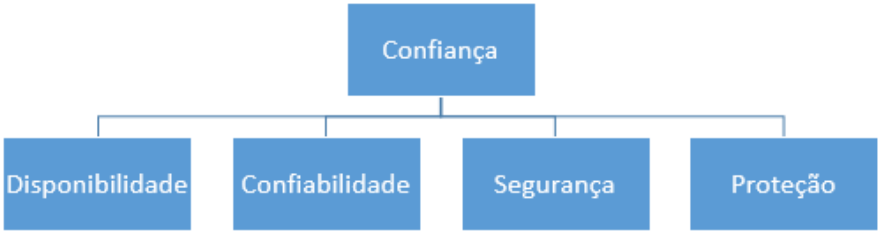

Figura 1- Dimensões da confiança [1].

Assim como estas dimensões existem também, outras propriedades do sistema consideradas aspetos de confiança, fundamentais na construção de sistemas designados críticos [3], sendo elas:

- **Facilidade de manutenção**: um sistema de fácil manutenção é aquele que pode ser atualizado com baixo custo e a sua adaptação não ocasiona erros;
- **Facilidade de reparo**: que é aperfeiçoada quando a equipa que faz uso do *software* tem acesso ao código fonte e estão habilitadas a elaborar o reparo;
- **Tolerância a erros**: quando é verificado um erro, o sistema deve detetar o mesmo e corrigi-lo;
- **Capacidade de sobrevivência**: sendo esta a capacidade do sistema continuar o serviço quando estiver a ser atacado.

1.6 Especificação de um Sistema Crítico

Para especificar um sistema crítico, é preciso compreender os riscos e gerar os requisitos de confiabilidade para lidar com eles. Esta especificação envolve alguns fatores [3]:

- **Identificação de riscos**: é um processo que na maioria das vezes apresenta alguma dificuldade e complexidade, devido ao facto de os riscos ocorrerem por causa das interações entre o *software* e as condições raras ambientais.

Técnicas como o *brainstorming* e experiências anteriores dos analistas que trabalharam no sistema crítico podem identificar estes riscos.

- **Análise e classificação de riscos**: uma análise é preciso ser feita para ver se o risco representa uma ameaça ao ambiente ou ao sistema. Também é preciso identificar o tipo de risco, tais como: intolerável, muito baixo e aceitável.
- **Decomposição de riscos**: neste processo descobrem-se as origens dos riscos no *software*. São diversas as técnicas propostas para a decomposição de riscos.
- **Avaliação de redução de riscos**: é preciso derivar os requisitos de confiabilidade do *software* ao identificar os riscos bem como as suas origens. Estes gerem os riscos e asseguram que estes acidentes não ocorram novamente mas para isso é preciso usar estratégias como a prevenção, deteção, eliminação de riscos bem como a limitação de danos.

A combinação destes fatores têm uma importância fundamental em um sistema crítico mas cabe no entanto ao elemento que faz esta especificação se adequar as necessidades de cada sistema [1].

1.7 Ambientes Computacionais de Alta disponibilidade

A falha de um sistema crítico pode causar severas repercussões financeiras às organizações desde atrasos nas suas operações, perdas irremediáveis bem como grandes prejuízos ao próprio nome e imagem diante dos seus clientes.

Estas interrupções não planeadas, de nome *unplanned downtimes* ou falhas não programadas, retratam a situação onde um cliente ou utilizador não consegue elaborar a sua tarefa por inacessibilidade ou perda de performance do sistema [4].

De modo a impedir estas falhas não programadas e minimizar as que são programadas para manutenções, foi estabelecido um outro conceito de nome ambientes computacionais de alta disponibilidade [5]. Consiste em uma série de tecnologias de redundância e de segurança aplicados aos componentes do sistema desde o ambiente onde os equipamentos apresentam-se instalados. *Nobreaks* ou UPS para evitar o desligamento inesperado por falha de energia nos servidores, dispositivos de rede e armazenamento de dados com componentes de *hardware* redundantes que continuam operacionais mesmo após apresentar algum tipo de falha, redundância de equipamentos, de conexões de rede até sistemas em cluster que mesmo com uma falha completa de um servidor ou componente da rede de dados mantém-se operacionais, sempre visando

eliminar os pontos únicos de falha ou *SPOF*. As falhas de *hardware* mais frequentes estão relacionadas aos discos rígidos devido os seus componentes eletromecânicos, as altas velocidades de rotação bem como a complexidade dos seus mecanismos [4]. E com o objetivo de evitar a perda ou corrupção de dados, adota-se dispositivos de armazenamento de dados com tecnologia RAID que possibilitam um grupo de discos operarem em conjunto como uma unidade lógica utilizando algoritmos de paridade ou espelhamento que mantém a integridade dos dados mesmo que ocorra a falha de um destes discos rígidos que compõe este grupo. [6].

A alta disponibilidade é acima de tudo, uma política de conceção, instalação, manutenção e administração de um sistema computacional de modo a maximizar o seu intervalo de tempo em que este permanece ligado, dentro dos padrões pré estabelecidos de necessidade.

2. Estado da Arte

Na atual temática onde organizações e empresas dependem cada vez mais das suas infraestruturas de tecnologias de informação, as aplicações e sistemas críticos às organizações necessitam de estar operacionais e disponíveis aos utilizadores e clientes vinte e quatro horas por dia durante todos os dias do ano. Desta forma, para além de requerer equipamentos (*hardware*), aplicações (*software*) e ambientes (*data centers*) de alta confiabilidade e qualidade também são necessárias mãos-de-obra capacitadas, instruídas assim como processos de gestão estruturados e serviços de suporte diferenciados capazes de evitar as falhas não programadas e responder atempadamente a estes incidentes diminuindo assim a indisponibilidade destes sistemas críticos e continuidade dos negócios [6].

Torna-se cada vez mais evidente a nível empresarial que não há mais espaço para estas falhas não programadas (ainda que momentâneas) neste tipo de sistemas. Manter a dinâmica destas organizações neste mercado extremamente competitivo, exige que sejam considerados aspetos fundamentais, como a estrutura que contempla estes equipamentos (*hardware*), as aplicações ou sistemas operacionais (*software*) e os ambientes onde estes equipamentos estão instalados e operacionais (*data centers*), a capacitação das pessoas que operam e suportam esta infraestrutura, o nível de serviços de suporte e assistência técnica com os fornecedores que derivam a mesma e também processos de liderança para gerir as pessoas e as infraestruturas. É importante frisar a existência de um processo de aprendizagem contínuo, que identifique a causa das falhas não programadas (*unplanned downtimes*) em sistemas e aplicações. Um processo capaz de apontar oportunidades de melhorias, tornando-se com isto importante apoio aos gestores nos momentos de tomar decisões. É dentro deste quadro que institutos de pesquisa focados no mercado de tecnologias de informação, como o Gartner nos EUA, realizam pesquisas de campo com o objetivo de subsidiar gestores de IT, para que as diversas empresas e organizações em que estes atuam possam efetuar um correto direcionamento de seus investimentos assegurando assim a continuidade das suas operações e disponibilidade dos seus sistemas e aplicações criticas [6].

Segundo uma pesquisa de abrangência mundial realizada pelo Gartner em 1998 [5], apontou-se que as causas das falhas não programadas eram em 20% relacionadas a falhas da tecnologia, ou infraestrutura de IT, o que também incluía falhas de *hardware* nos equipamentos, sistema operacional ou de ambiente incluindo desastres naturais. Ainda nesta pesquisa evidenciava-se que 40% estavam em falhas de aplicação (*software*) e,

finalmente, que os outros 40% estavam relacionados a erros operacionais (falha humana) como podemos observar na Figura 2.

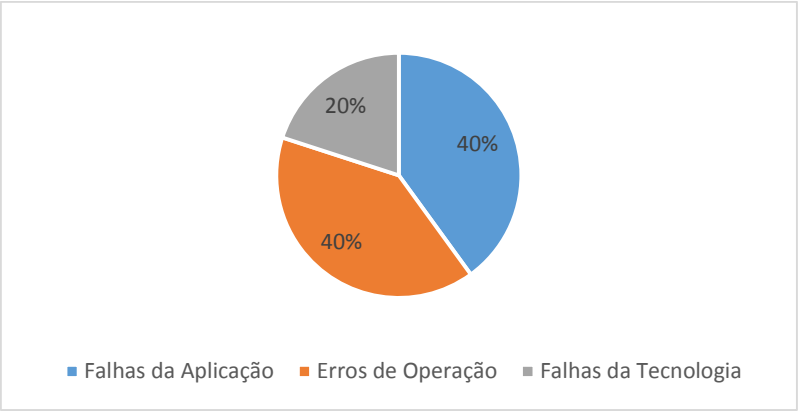

Figura 2 - Causa das falhas não programadas em sistemas [6].

No entanto de acordo a uma pesquisa feita pela Quorum[1], que é uma empresa dedicada a recuperação de desastres, no ano de 2013 este cenário alterou bastante como podemos observar na Figura 3 [7]. Os desastres naturais bem como as falhas de *hardware* hoje em dia têm mais incidência na causa destas falhas.

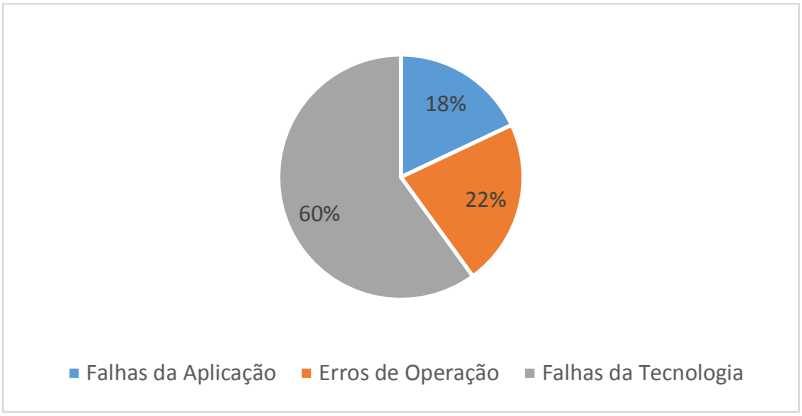

Figura 3 - Causas das falhas não programadas em sistemas nos dias de hoje [7].

Este aumento deve-se ao fato de as empresas hoje em dia requererem de um grande poder computacional e serviços de alta disponibilidade que com os seus requisitos corporativos entretanto aumentou a propensão a falhas de *hardware* [8].

[1] Http://www.quorum.net/

2.1 Mecanismos e aplicações de gestão de disponibilidade de sistemas críticos

Nos dias de hoje com o desenvolver das tecnologias bem com o aparecimento e crescimento de empresas que têm grande peso no mercado, todo o cuidado é pouco no que toca a tratar de sistemas considerados críticos. Existem diversas empresas bem como *softwares* e aplicações focados para o tratamento e gestão da disponibilidade destes sistemas considerados críticos.

É o caso do Double-Take Avaliability apresentado pela empresa Norte Americana Vision Solutions [9] e do CA ARCserve High Avaliability apresentado pela empresa Norte Americana ARCserve [10]. Mas no mercado ainda existem outros produtos nesta área porém alguns com menor impacto no mercado e menos funcionalidades.

2.1.1 Double-Take Avaliability

Com soluções para Windows, Linux, AIX e IBM i[2], este tem como objetivo manter o sistema disponível bem como resiliente. Oferece proteção com e sem agentes para servidores físicos, bem como virtuais e em nuvem. É uma solução independente de *hardware* e reduz custos operacionais gerais otimizando a infraestrutura existente.

Especialmente projetado para ambientes populares de virtualização, como o VMware vSphere, Microsoft Hyper-V, o Double-Take Availability também inclui um *hardware* robusto bem como *clustering* independente das aplicações, com ou sem replicação lógica subjacente [11].

Esta aplicação apresenta diversas características como [9]:

- Recuperação de desastres bem como fornecimento de alta disponibilidade;
- Oferece suporte bem como mobilidade para qualquer combinação de servidores físicos, virtuais ou em nuvem;
- Oferece suporte para diferentes tipos de *hardware*;
- Gere todo o ambiente de proteção em uma única consola unificada;
- Não apresenta limite de distância entre servidores;
- Garante a integridade dos dados replicados em tempo real;
- Faz a monitorização da rede e serviços dos servidores;
- Elimina interrupções no sistema de produção;

[2] Http://www-03.ibm.com/systems/power/software/i/about.html

- Elimina a necessidade de aquisição de *hardware* especial para proteção dos servidores;
- Não oferece perda de dados, entre outras características.

2.1.1.1 Como funciona

O Double-Take Avaliability oferece uma proteção de dados completa e possibilita a recuperação instantânea de qualquer falha do servidor. Este oferece também, mobilidade total da carga de trabalho bem como otimização e consolidação de recursos por meio da migração de cargas de trabalho entre plataformas de computação sem interrupções [11].

Com a captura contínua de alterações de bytes e sua replicação assíncrona em tempo real para qualquer armazenamento a qualquer distância local ou global, o Double-Take Availability garante que não se perderá nenhum dado nem se enfrentará problemas durante o tempo de inatividade. Também podem ser implementados *clusters* de *failover* sem armazenamento compartilhado ou limitações geográficas, eliminando o ponto único de falha e proporcionando assim a liberdade de localizar nós de cluster no local desejado [11].

Este monitoriza as alterações em todos os arquivos protegidos e replica apenas os bytes que foram alterados. É uma tecnologia que permite replicar para qualquer local de recuperação de desastres, através de redes IP padrão, para a máxima proteção contra a perda de dados.

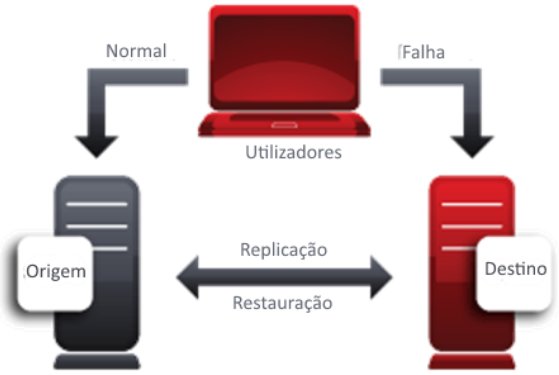

Figura 4 - Modo de Funcionamento do Double-Take Avaliability [9].

A Figura 4 faz uma ilustração de como funciona o Double-Take Avaliability para os ambientes Windows e Linux [12].

2.1.1.2 Plataformas Suportadas

O Double-Take Avaliability suporta os seguintes sistemas operativos e plataformas de virtualização [11]:

- Servidores Windows de 32 ou 64 bits/ Servidores Linux;
- Windows 2003 e 2003 R2;
- Windows 2008 e 2008 R2;
- Windows 2012 ;
- Red Hat Enterprise Linux e CentOS versões: 4.8, 4.9, 5.6, 5.7, 6.1, 6.2;
- Oracle Enterprise Linux versões: 5.6, 5.7, 6.1, 6.2;
- SUSE Linux Enterprise Server versões: 10.3, 10.4, 11.0, 11.1;
- VMware vSphere;
- Windows Server 2008 R2 Hyper-V;
- Windows Server 2003/2008 Storage Server Edition.

2.1.2 CA ARCserve High Avaliability

O CA ARCserve High Avaliability é um *software* que reduz o tempo de inatividade do sistema bem como a perda de dados ajudando assim a cumprir os acordos de nível de serviço exigentes e estratégias de continuidade de negócios bem como recuperação de desastres [10].

Este *software* protege os servidores Windows, Linux e UNIX, assim como aplicações e dados em ambientes de servidores físicos e virtuais nos seus *data centers* e escritórios remotos e pode ser implantado localmente bem como em nuvem.

Tem como características [13]:

- Minimizar o risco de inatividade dos negócios durante interrupções planeadas e não planeadas;
- Testes de recuperação automatizados;
- Reduzir o tempo de recuperação após a perda de dados ou danos;
- Replicações Multi-Stream;
- Melhorar qualquer estratégia de recuperação de desastres;
- Apresenta um gerenciamento unificado baseado na web e consola de relatórios com redundância interna para alta disponibilidade;
- Comunicações seguras;
- Sincronização *offline* para uma implementação mais rápida;

- Faz o retrocesso de dados para uma proteção de dados contínua (CDP) possibilitando assim uma recuperação fácil e rápida dos dados perdidos ou danificados, entre outras características.

2.1.2.1 Como funciona

O CA ARCserve High Avaliability monitoriza todos os eventos críticos, incluindo falhas no servidor global e todas as falhas do serviço de base de dados, e automaticamente ou com uma simples instrução, inicia uma transição.

Se o servidor principal ficar indisponível, as suas atividades podem ser comutadas automaticamente para um local remoto (réplica). A mudança, que é transparente para o utilizador, inclui inicialização imediata de uma base de dados standby sincronizada, e redireciona todos os utilizadores para lá imediatamente. Tudo isso é feito sem a necessidade de reconfigurar clientes ou a rede.

Este redireccionamento pode basear-se nos seguintes métodos:

- Mobilidade de IP (se o local de espera é implementado dentro da mesma rede segmento);
- Redirecionar o DNS, pode ser usado em uma rede local ou quando o local de espera remoto está localizado em uma rede IP diferente (comutação cross-network);
- Mudança do nome do servidor.

A seleção dos métodos de redireccionamento baseiam-se nos requisitos da aplicação a serem protegidos. Certos métodos podem não ser aplicados a um determinado cenário [14]. A Figura 5 faz uma ilustração de como este funciona.

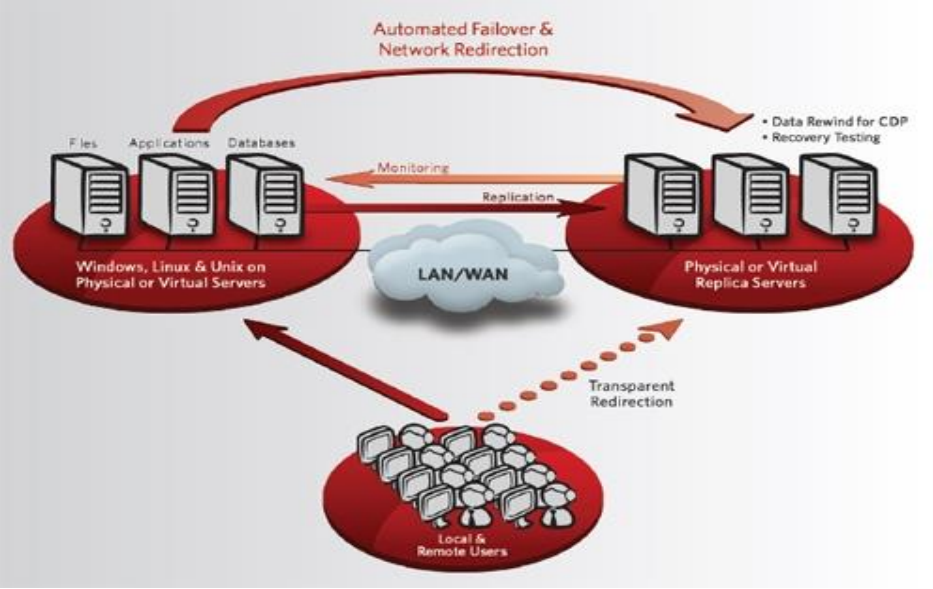

Figura 5 - Modo de Funcionamento do CA ARCserve High Avaliability [13].

2.1.2.2 Plataformas Suportadas

O CA ARCserve High Availability suporta uma ampla gama de sistemas operativos e plataformas de virtualização tais como [15]:

- Microsoft Windows Server 2003, 2008 e 2008R2 in Standard or Enterprise Edition (32-bits ou 64-bits);
- UNIX AIX versões 5.2, 5.3 6.1, Solaris vesões 9, 10, SP ARC, Intel;
- Linux RedHat versões 4,5, SUSE versões 9, 10;
- VMware Virtual Infrastructure e vSphere;
- Microsoft Hyper-V;
- Citrix XenServer.

2.1.3 Outras Soluções

Existem ainda no mercado outras soluções que tratam da gestão da alta disponibilidade de sistemas críticos, porém são de menor dimensão que as duas soluções citadas acima no que toca as suas características, funcionalidades e impacto no mercado.

evenRun Enterprise (Stratus) – Esta é projetada para funcionar em servidores x86, sendo uma solução de alta disponibilidade que automaticamente deteta, isola e lida com falhas com um *uptime* muito superior e desempenho em uma fração do custo das soluções de clusters ou de espera. É um *software* que suporta aplicações Microsoft *multi-core* e *multi-processor*. Esta solução de disponibilidade pode ser executada tanto em Windows bem como em Linux, sem alterar as suas aplicações. [16]. Principais Vantagens:

- Evita a inatividade da aplicação localmente;
- Não requer modificações para aplicações;
- Permite a conformidade e continuidade do negócio;
- Protege servidores inteiros ou apenas os dados selecionados;
- Opera em ambientes virtuais, físicos ou em nuvem;
- Oferece proteção tolerante a falhas ou de alta disponibilidade para multiprocessadores simétricos.

Stratus Avance High-Availability Software – É uma solução de *software* que combina dois servidores padrão em um único sistema à prova de falhas. Todos os dados e aplicações são sincronizados entre os dois servidores de forma contínua e em tempo real. Quando algo dá errado, um servidor pode assumir perfeitamente o trabalho do outro sem que o utilizador se aperceba. Ao contrário dos *clusters* de alta disponibilidade tradicionais, o Avance oferece disponibilidade muito superior e mais confiável, sem o custo e complexidade destes [17]. Principais Vantagens:

- Suporte a aplicações Windows e Linux;
- Proteção Cross-campus;
- Reduz o custo de compra e manutenção;
- Melhora a confiabilidade da infraestrutura de IT;
- Evita o tempo de inatividade do servidor e mantém as aplicações em execução.

PACSystems High Availability – É uma solução contruída sobre uma plataforma de controlo escalável, redundante e sincronizada que garante assegurar o controlo ininterrupto das aplicações e processos com transparência total. Esta oferece uma dupla redundância na sincronização de dados com módulos duplos [18]. Principais Vantagens:

- Diminui o tempo de inatividade devido a sua capacidade robusta de dupla redundância;
- Garante uma máxima proteção dos investimentos com arquiteturas escaláveis, abertas e flexíveis;
- Economiza tempo e reduz os custos de engenharia, com fácil configuração, inicialização rápida e manutenção;
- Aumento da integridade dos dados através da verificação e correção avançada de erros de memória.

Neverfail (High Availability and Disaster Recovery) – Concebida para evitar a paralisação de negócios, esta solução mantém os utilizadores finais constantemente conectados as suas aplicações de missão crítica, apesar de falhas como perda ou corrupção de dados, falhas na rede, aplicação ou no servidor bem como degradação da performance da aplicação [19]. Principais Vantagens:

- Não requer armazenamento partilhado;
- Faz replicação de dados sobre WAN / LAN;
- Tem um baixo custo e é fácil de gerir;
- É independente da marca ou configuração do servidor;
- Proteção do tipo *out-of-the-box* para todas as aplicações baseadas em Windows.

2.1.4 Principais Diferenças

Neste ponto apenas serão referidas as principais diferenças entre o Double-Take Availability e o CA ARCserve High Availability pois estes apresentam mais características, especificações bem como funcionalidades perante as demais soluções já acima citadas. Ambas as soluções apresentam características semelhantes mas tem aspetos que as distinguem, ambas têm uma extensa lista de recursos e providenciam um sistema de alta disponibilidade bem como recuperação de desastres abrangente para servidores físicos e virtuais.

Entre as principais diferenças temos que o Double-Take Availability oferece uma grande vantagem sobre o CA ARCserve quando se trata da capacidade de compactar os dados antes de enviá-los para o servidor alternativo. Esta capacidade de compressão de dados permite mais espaço livre para os dados adicionais a serem recuperados porém implica mais capacidade de processamento por parte do servidor.

O CA ARCserve High Availability é melhor no que toca a recuperação de desastres e também pode ser utilizado em conjunto com outros programas de recuperação e o seu preço é muito mais acessível do que o Double-Take Avaliability. A Tabela 1 mostra as principais diferenças entre os dois no que toca a características sobre replicação e alta disponibilidade com uma escala de pontuação de 0 a 5 [20].

Características	Double-Take Availability	CA ARCserve High Availability
Replicação	5	5
Alta Disponibilidade	4	5
Failback	1	5
Suporte (servidor)	5	5
Sincronização Offline	5	5
Proteção Contínua de Dados	3	5
Suporte (*cloud*)	3	4
RTO/RPO (para recuperação de desastres)	4	5
Integração de *Backup*	0	4
Total	3.3	4.7

Tabela 1 - Comparação de caraterísticas sobre replicação e alta disponibilidade [20].

Posto isto e fazendo um balanceamento entre os dois produtos, conclui-se que o CA ARCserve High Availability é melhor do que o Double-Take Availability fazendo também uma melhor relação preço/características.

O CA ARCserve High Availability tem claramente muitos pontos fortes na replicação e na alta disponibilidade e oferece maior tempo de atividade e disponibilidade para os seus servidores críticos, aplicações e dados. Além disso, tem capacidades que o Double-Take Availability carece completamente, como *backup* baseado em imagem e com base em arquivo.

Quando se sucede um desastre, o CA ARCserve High Availability é uma opção escalável, fácil de usar e confiável apresentando também uma interface de utilizador mais amigável bem como um preço mais acolhedor.

2.2 Tolerância a Falhas e Redundância

A prevenção e remoção de falhas não são suficientes quando o sistema em causa exige uma alta disponibilidade ou uma alta confiabilidade. As técnicas de tolerância de falhas são utilizadas em sistemas de computadores como uma forma de se lidar com condições de erro que podem surgir durante as suas vidas operacionais [21] .

Um sistema tolerante a falhas é aquele que, sem nenhuma intervenção manual é capaz de lidar com falhas operacionais e de projeto bem como de manter o desempenho do sistema dentro das suas especificações [22].

Estas técnicas são todas baseadas em redundância o que exige componentes adicionais ou até mesmo algoritmos especiais.

Sendo a redundância a palavra-chave, hoje em dia considerar um sistema tolerante a falhas, significa dizer que este é um sistema redundante. A aplicação desta para implementar técnicas de tolerâncias a falhas pode aparecer de diversas formas [23]:

Redundância de Informação:

Esta corresponde a duplicação de dados e/ou adição de informações redundantes a estes, de forma a permitir a verificação da consistência e/ou correção dos erros detetados, como acontece em códigos de deteção e correção de erros. Tem como exemplos:

- Códigos de deteção/correção de erros;
- RAID, etc.

Redundância Temporal:

Esta técnica consiste em executar a mesma instrução, em instantes distintos de tempo, a fim de se verificar a existência de falhas temporais no sistema. Esta é usada em sistemas onde o tempo não é crítico. Tem como exemplos:

- Re-execução de código em momentos distintos.

Redundância de Hardware:

Este tipo de redundância é obtido com a replicação de componentes eletrónicos, memorias, fontes de alimentação, entre outros, que assumem as tarefas do sistema em caso de falhas dos *hardwares* principais Tem como exemplos:

- NMR;
- TMR;
- SMR, etc.

Redundância de Software:

É o tipo de redundância onde várias versões de um *software* é implementado segundo a mesma especificação porém por equipas diferentes. Temos como alguns exemplos:

- N-Version (diversidade);
- Blocos de Recuperação;
- N-Self-Checking Programming, etc.

Todas estas formas de redundância apresentam um impacto no sistema, seja no custo, no desempenho ou mesmo na potência consumida sendo desta forma o uso desta bem ponderada [21].

A tolerância a falhas é um requisito vital em um sistema considerado crítico, sem o qual a sua alta disponibilidade e por consequência a sua confiabilidade são postas em causa.

2.3 VPS, Servidor Dedicado ou Servidor em Nuvem

Nos dias de hoje pequenas e médias empresas bem como os programadores de aplicações, plataformas e websites questionam-se cada vez mais acerca do tipo de modelo de hospedagem que irão utilizar.

É importante referir também, que qualquer um destes modelos permite a execução bem como a configuração destas aplicações, *softwares* ou websites porém, nem toda a solução necessita do mesmo tipo de recursos e desempenho estando assim a escolha pendente da importância e requisitos da mesma.

Nesta secção são descritas algumas das principais diferenças de um VPS, Servidor Dedicado ou Servidor em Nuvem [24].

VPS:

Um VPS é um servidor virtual privado e é um tipo de hospedagem compartilhada. Os recursos são uma parte de um todo e não é facilmente escalável. É importante saber que o desempenho dos recursos podem ser afetados pelo alto consumo de outras aplicações caso estes não apresentem reserva.

Este aparece para o utilizador como um servidor dedicado, mas é, na verdade, instalado em um computador que serve várias aplicações, websites, etc. Um único recurso físico pode ter vários VPSs.

Servidor Dedicado:

Um servidor dedicado é o oposto de um VPS. Não é de hospedagem compartilhada e o utilizador tem todos os recursos à sua disposição. É menos escalável que um VPS, no entanto, é potencialmente mais competitivo, porque os recursos não estão a ser compartilhados com ninguém. Os recursos são o todo e não uma parte de um todo. Este tipo de soluções de hospedagem é sempre mais caro, porque há mais *hardware* envolvido e todos os recursos estão disponíveis, mesmo quando não estão a ser usados. É altamente ineficiente, mas, por vezes, mais útil, especialmente quando é preciso um controle completo da plataforma.

Servidor em Nuvem:

Um Servidor em Nuvem é o tipo mais recente e mais flexível de hospedagem. Oferece disponibilidade sem nenhum ponto de falha, e é uma mistura de ambos os modelos acima. A escalabilidade de recursos é fácil e rápida. A principal característica deste modelo é a flexibilidade que ele oferece e a redução de custos.

A Tabela 2 mostra algumas diferenças existentes entre estes tipos de hospedagem.

	VPS	**Servidor Dedicado**	**Servidor em Nuvem**
Infraestrutura	Servidor físico único, utilizado por um número limitado de utilizadores	Servidor físico único, utilizado por um utilizador ou cliente	Recursos distribuídos entre vários servidores físicos
	Ambiente compartilhado, virtual e isolado	Ambiente isolado	Ambiente virtual isolado
Escalabilidade	*Upgrade* ou *downgrade* manual. Tempo de inatividade necessário.	Não atualizável. Se o utilizador precisar de mais recursos, tem que começar outro pacote definido pelo fornecedor da hospedagem.	Implantação rápida, *Upgrade* e *downgrade* feitos em segundos. Não apresenta tempo de inatividade.

Tabela 2 - Comparação entre Modelos de Hospedagem [24].

3. Visão Geral do Sistema

O objetivo principal desta dissertação consistiu em estudar bem como propor uma metodologia de sistemas críticos que permitissem mitigar falhas de sistemas assim como infraestruturas, possibilitando aos utilizadores a continuidade do uso do sistema mesmo que de forma condicionada ao longo do período de falha. Assim sendo, foi estudada uma forma ampla para que fosse desenhada e depois desenvolvida uma arquitetura de controlo que abrangesse diversos casos de uso. Nesta dissertação vão ser tidos em conta quatro casos de uso que vão ser descritos e detalhados ao longo deste capítulo.

Neste capítulo será também apresentada a arquitetura deste sistema assim como as suas especificações e como os casos de uso fluem na mesma.

3.1 Casos de Uso

Os casos de uso descrevem os requisitos externos de um sistema. São usados na fase de análise de requisitos que é um aspeto importante na gestão de projetos, fase esta responsável por recolher dados indispensáveis, necessários, exigências de que o utilizador precise para solucionar um problema e alcançar os seus objetivos. Sendo o principal requisito desta arquitetura a alta disponibilidade de um sistema, os casos de uso que serão apresentados de seguida espelham diferentes situações e cenários que têm de ser tomados em conta.

Primeiro caso de uso:

Descrição – O servidor do sistema-cliente encontra-se em disponibilidade, isto é, não apresenta qualquer tipo de falha ou problema que ponha em causa o seu bom funcionamento.

Figura 6 - Ilustração do Primeiro Caso de Uso.

Solução – Não há necessidade de uma intervenção externa por parte de qualquer tipo de solução visto que o sistema não apresenta qualquer falha.

Segundo caso de uso:

Descrição – O servidor do sistema-cliente encontra-se indisponível. Quando o servidor do sistema-cliente falha, têm de ser tidas em conta diversas soluções para colmatar esse problema.

Figura 7 - Ilustração do Segundo Caso de Uso.

Solução – Para este caso em que o servidor do sistema-cliente, ou melhor, o servidor principal encontra-se em baixa, existem diversas soluções e meios para rebater este problema tais como:

Replicação do servidor – É uma solução que melhora a disponibilidade e escalabilidade do sistema, sendo um método utilizado para eliminar a necessidade da intervenção humana como já pudemos constatar no Estado da Arte. Consiste na utilização de um servidor de apoio ao servidor principal de modo a substitui-lo em caso de falha.

- **Com partilha de armazenamento –** Um servidor redundante dedicado para substituir o servidor principal em caso de falha partilhando ainda o mesmo armazenamento. O servidor de apoio apenas fica ativo quando ocorre uma falha no servidor principal.
    - **Redundância de *Hardware* Local –** Quando os dois servidores se encontram no mesmo espaço físico.

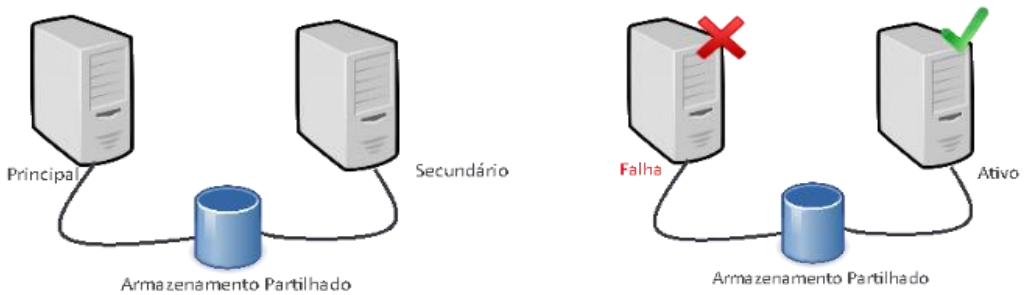

Figura 8 – Solução Segundo Caso de Uso.

- **Redundância de *Hardware* Remota –** Quando os dois servidores encontram-se em locais distintos.

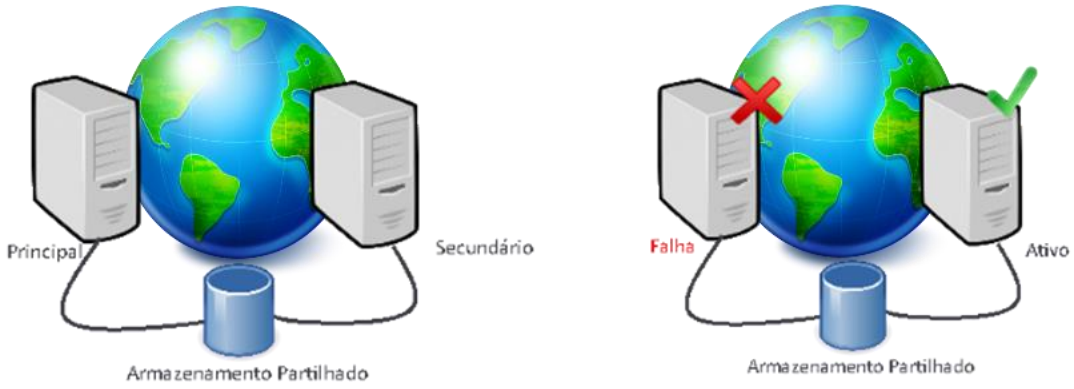

Figura 9 - Solução Segundo Caso de Uso (2).

- **Sem partilha de armazenamento –** Neste tipo de solução, cada servidor tem um armazenamento dedicado, o que implica uma troca de informações constante entre ambos os servidores (principal e secundário).
  - **Redundância de *Hardware* Local –** Quando os dois servidores encontram-se no mesmo espaço físico.

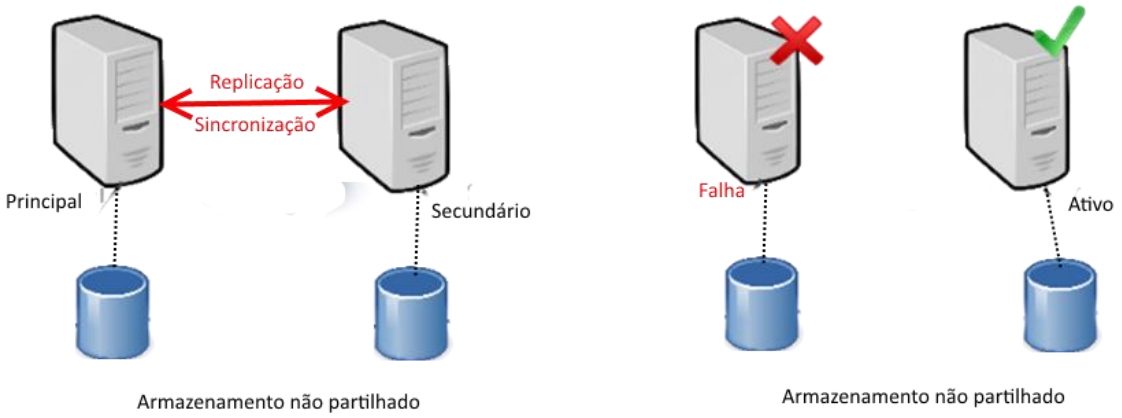

Figura 10 - Solução Segundo Caso de Uso (3).

- **Redundância de *Hardware* Remota –** Quando os dois servidores encontram-se em locais distintos.

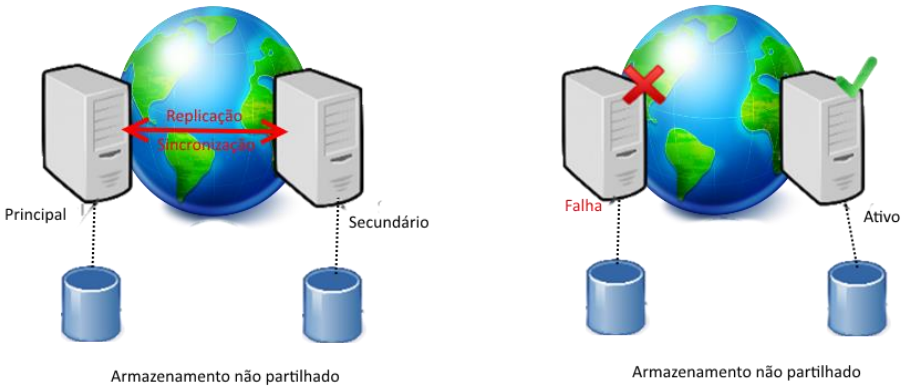

Figura 11 - Solução Segundo Caso de Uso (4).

Terceiro caso de uso

Descrição – Cenário em que o servidor do sistema-cliente encontra-se indisponível e o servidor da solução também está indisponível.

Figura 12 - Ilustração do Terceiro Caso de Uso.

Solução – Para este caso, o servidor da solução precisa de um servidor de apoio para substitui-lo em caso de falha voltando assim a solução do segundo caso de uso.

Quarto caso de uso

Descrição – Cenário em que o utilizador fica sem conexão com a internet.

Figura 13 - Ilustração do Quarto Caso de Uso.

Solução – Para este caso, o utilizador tem de fazer o uso de métodos que o possibilitem continuar a usar o sistema como por exemplo ter o modo trabalhar *offline* ativo no seu navegador web ou então a solução do sistema-cliente tem de fazer o uso do interface Application Cache[3] do HTML5 que possibilita que uma aplicação web fique armazenada em cache e que fique acessível ao utilizador mesmo sem este ter conexão com a internet [25]. Em ambos casos o utilizador tem de necessariamente ter acedido a página que se quer *anti-crash* anteriormente de maneira a que esta fique armazenada em *cache*.

3.2 Arquitetura do sistema

Atendendo as condições em causa e aos casos de uso em questão, a arquitetura deste sistema foi elaborada de modo a contornar os mais variados cenários de catástrofe bem como resolver estes casos de uso, tirando maior carga possível do lado do utilizador de maneira a que este continue a utilizar o sistema ainda que com restrições. Como já citado no estado de arte, um sistema considerado crítico e de alta disponibilidade tem de ser tolerante a falhas. E para esta arquitetura, a redundância de *hardware* é tida como escolha de modo a tolerar estas falhas de modo contínuo.

Também acordo com as características já citadas no estado de arte, para esta arquitetura, o modelo de hospedagem que mais se adequa é um servidor em nuvem. Esta é uma solução híbrida, resiliente bem como dinâmica, e ainda é económica e mais escalável do que os outros já apresentados, sem contar que não apresenta tempo de inatividade o que para um sistema considerado crítico é crucial para o seu funcionamento.

Posto isto, a estrutura geral desta arquitetura e as componentes de apoio a continuação do funcionamento da mesma encontram-se em *cloud*. Através de uma API Web são recebidos os dados que o utilizador quer inserir, onde de seguida é feita uma inserção destes dados em uma outra componente, a base de dados que também se encontra em *cloud*. Também faz parte das componentes de apoio um serviço em nuvem na qual através um *worker role* é verificado de tempo em tempo, nomeadamente 5 segundos, se o servidor do cliente já se encontra disponível fazendo a chamada a API Web (*web service*) do cliente enviando assim os dados, caso este esteja disponível. A Figura 14 ilustra como esta arquitetura encontra-se disposta assumindo que há conexão com a internet.

[3] Http://www.html5rocks.com/pt/tutorials/appcache/beginner/

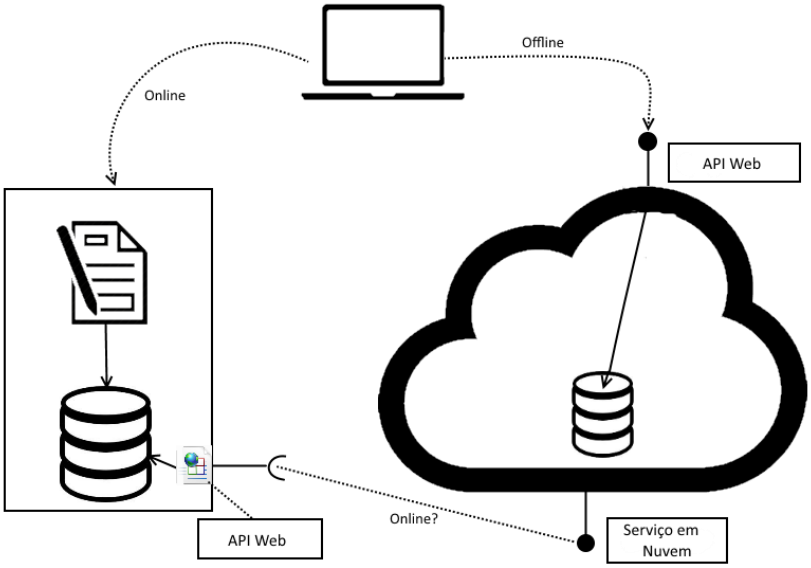

Figura 14 - Estrutura Geral.

Este serviço em nuvem está em constante funcionamento de modo a verificar se existe informação que precisa de ser tratada ou enviada para o servidor do sistema-cliente. O diagrama de sequência ilustrado na Figura 15, mostra como é feita esta verificação assim como o mesmo processa os dados.

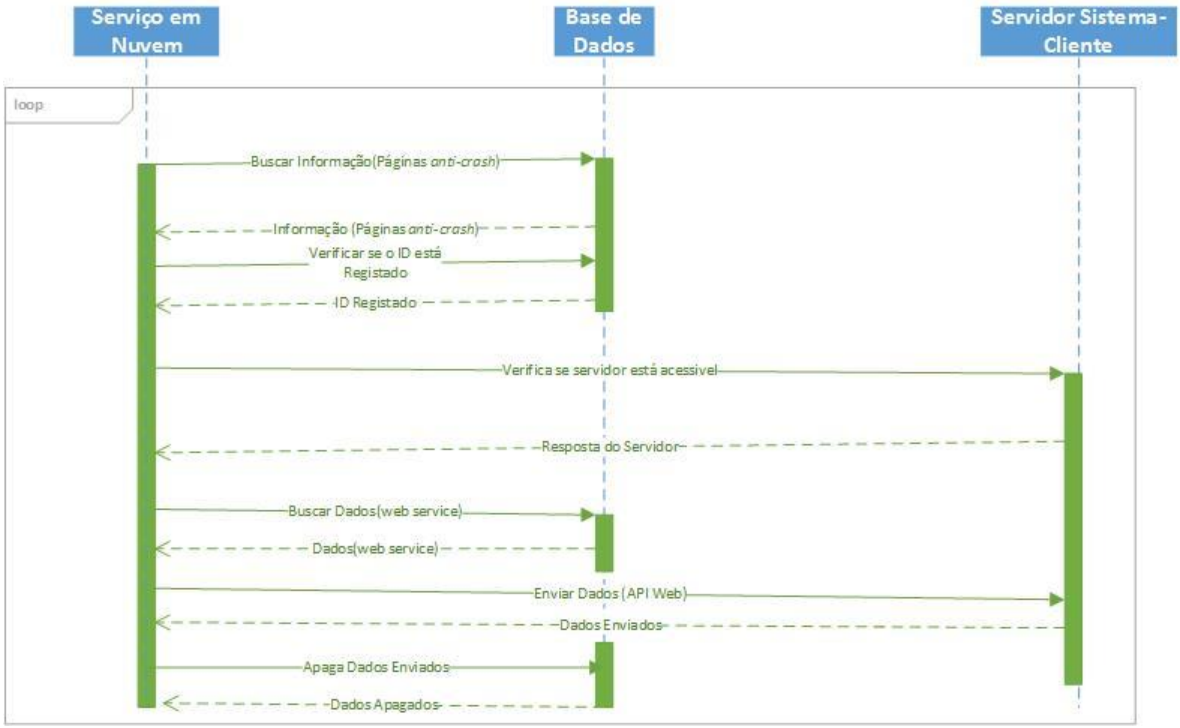

Figura 15- Diagrama de Sequencia do Funcionamento do Serviço em Nuvem.

Após verificar se existem informações a processar, este *worker role* faz uma verificação acerca da acessibilidade do *host* do servidor do cliente, adquirindo depois os dados relativos a este *host* que lhe permitam enviar a informação de volta, nomeadamente nome e parâmetros do *web service* do cliente para posteriormente ser feita uma chamada a este serviço que se encarregará de devolver a informação que por inacessibilidade do servidor do cliente não foi processada por este. Depois deste processo, estes mesmos dados são eliminados da base de dados do servidor da solução.

3.2.1 Tracking Code

Para que o sistema-cliente consiga fazer o uso desta solução, tem de fazer o uso de um script que possibilitará a invocação de determinados métodos para a verificação da conectividade dos servidores do cliente bem como os componentes que se encontram em *cloud*. Este script que funcionará como um *tracking code* à semelhança do que o Google Analytics dispõe. Este código de acompanhamento contém apenas os dados mínimos necessários para a identificação do cliente em causa, um ID de controlo bem como dados necessários para o seu arranque e funcionamento que são disponibilizados por um script externo. É preciso ser usada a biblioteca *jQuery* e para isso faz parte deste script a biblioteca do *jQuery* que é hospedada no CDN da Microsoft.

Cada sistema-cliente possui um identificador único por API web registada na plataforma de suporte *back office* desta arquitetura.

A caixa de texto abaixo contém um exemplo de como se apresenta o script a ser utilizado pelos sistemas-cliente. Este tem de ser utilizado nas páginas que se quer *anti-crash*.

```
<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.1.min.js"></script>

<script> var uiD= 'UD20140630092457'; </script>

<script src="http://websiteoffline.azurewebsites.net/webscript.js">  </script>
```

Caixa de Texto 1 – Tracking Code.

3.3 Conclusões

Nesta secção será apresentada uma breve comparação entre as soluções apresentadas para resolver cada caso de uso apresentado e a solução da arquitetura acima citada. Serão ainda dispostos diagramas a exemplificar como esta solução flui em cada caso de uso.

A tabela que se segue apresenta as vantagens bem como as desvantagens de cada solução.

Vantagens/ Desvantagens	Replicação do servidor				Solução da Arquitetura
	Com partilha de armazenamento		Sem partilha de armazenamento		
	Redundância de *Hardware*				
	Local	Remota	Local	Remota	
Primeiro Caso de Uso	✓	✓	✓	✓	✓
Segundo Caso de Uso	✗	✓	✗	✓	✓
Terceiro Caso de Uso	✗	—[4]	✗	—	✓
Quarto Caso de Uso	✗	✗	✗	✗	—

Tabela 3 - Análise Comparativa de Soluções.

Como podemos observar, a mera deslocação de um servidor nem sempre pode resolver uma falha no caso de uma catástrofe, questões como o armazenamento de informação e redundância de servidores devem ser tidas em conta. Em relação ao terceiro caso de uso, em que os dois servidores se encontram indisponíveis, a disponibilidade o não do sistema vai depender da dimensão da redundância existente, ou seja o número de servidores de *back up* existentes. Já no quarto caso de uso caso os utilizadores têm de fazer o uso de métodos que o possibilitem continuar a usar o sistema mesmo sem conexão com a internet como por exemplo ter o modo trabalhar *offline* ativo no seu navegador web ou então a solução do sistema-cliente tem de fazer o uso do interface Application Cache[5] do HTML5.

[4] Pode ou não resolver.
[5] Http://www.html5rocks.com/pt/tutorials/appcache/beginner/

A arquitetura apresentada aqui nesta dissertação, tem em conta a todos os intervenientes de um sistema de modo a que o utilizador ainda que de maneira condicionada consiga fazer o uso do sistema.

Posto isso, os diagramas seguintes apresentam a forma como a solução desta arquitetura deste sistema flui de acordo a cada caso de uso referido.

Primeiro Caso de Uso

Se o utilizador tiver conexão com a internet e o servidor do sistema-cliente estiver acessível, não há necessidade de ser encaminhado algum tipo de dados para a solução de suporte. O diagrama de fluxo ilustrado na Figura 16, mostra como é tratado este caso de uso.

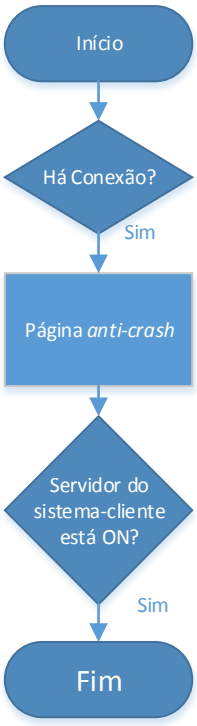

Figura 16 - Diagrama de Fluxo do Primeiro Caso de Uso.

Segundo Caso de uso

Caso o utilizador tenha conexão com a internet e o servidor do sistema-cliente estiver inacessível, os dados serão encaminhados para o servidor da solução, que se encontra em *cloud*, caso este esteja acessível. Posto isso é feita uma questão ao utilizador caso queira armazenar os seus dados em um servidor secundário (servidor da solução)

para posteriormente estes serem enviados para o servidor do sistema-cliente. Se a resposta for negativa não serão armazenados dados. Caso a mesma seja positiva, os dados serão depositados na base de dados que se encontra no servidor da solução, e é verificado se existem dados associados a utilizadores, isto é *web services* registados. Se sim, é verificado se o servidor do sistema-cliente já se encontra disponível e se os *hosts* são compatíveis. Caso sejam, é verificada se para aquele cliente em questão existem dados por enviar e caso a resposta seja positiva é feita uma chamada ao *web service* do sistema-cliente com os dados a enviar. O diagrama de fluxo ilustrado na Figura 17, mostra como é tratado este caso de uso.

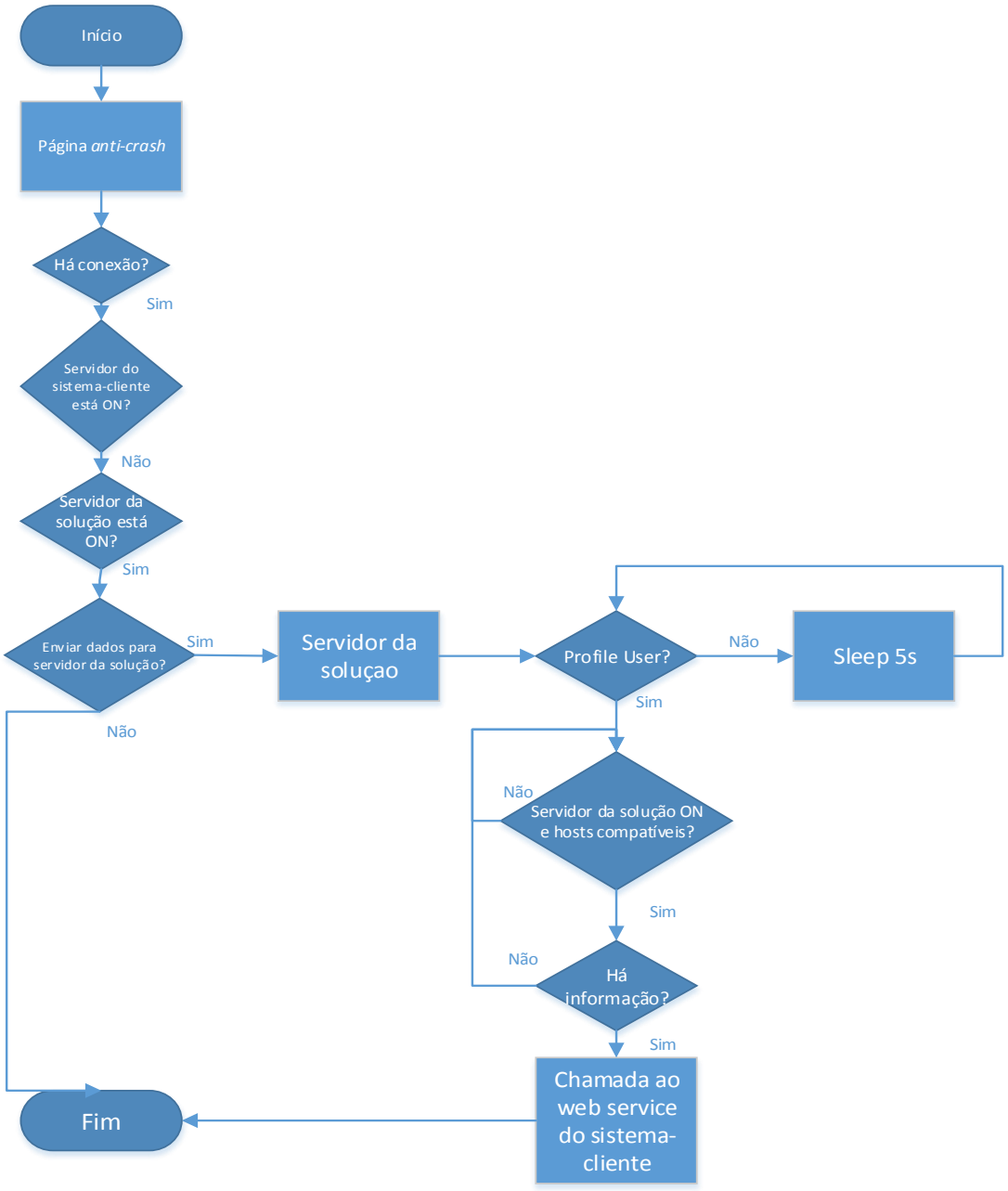

Figura 17 - Diagrama de Fluxo do Segundo Caso de Uso.

Como a Figura 18 ilustra, é sempre feita uma questão ao utilizador caso o servidor do sistema-cliente esteja inacessível, e como mostra o diagrama de fluxo do segundo caso de uso, diferentes decisões serão tomadas dependo da resposta a esta questão.

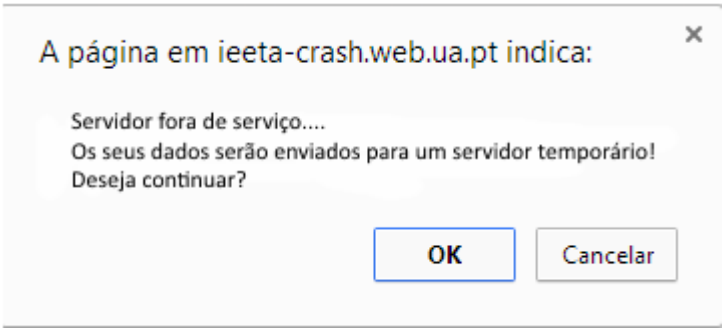

Figura 18 – Alerta de Inacessibilidade do Servidor do Sistema-Cliente.

Terceiro Caso de Uso

Caso o utilizador tiver conexão com a internet e o servidor do sistema-cliente estiver inacessível, os dados serão encaminhados para servidor da solução caso este esteja acessível. Se o mesmo estiver inacessível, os dados são armazenados no *browser* do utilizador (HTML5 local storage) e quando o mesmo voltar a aceder a esta página, é feita uma questão para saber-se se o utilizador quer enviar os dados que foram armazenados no seu *browser*. Se a resposta for negativa a mesma questão será feita novamente quando o utilizador voltar a aceder a página em questão. Caso a resposta seja positiva, é feita uma nova verificação acerca da disponibilidade do servidor da solução e caso ainda não esteja o processo é repetido. Se este estiver disponível, os dados são depositados na base de dados que se encontra no servidor da solução, e é verificada se existem dados associados a utilizadores, isto é *web services* registados. Se sim, é verificado se o servidor do sistema-cliente já se encontra disponível e se os *hosts* são compatíveis. Caso sejam é verificada se para aquele sistema-cliente em questão existem dados por enviar e caso a resposta seja positiva é feita uma chamada ao *web service* do sistema-cliente com dados a enviar. O diagrama de fluxo ilustrado na Figura 19, mostra como é tratado este caso de uso.

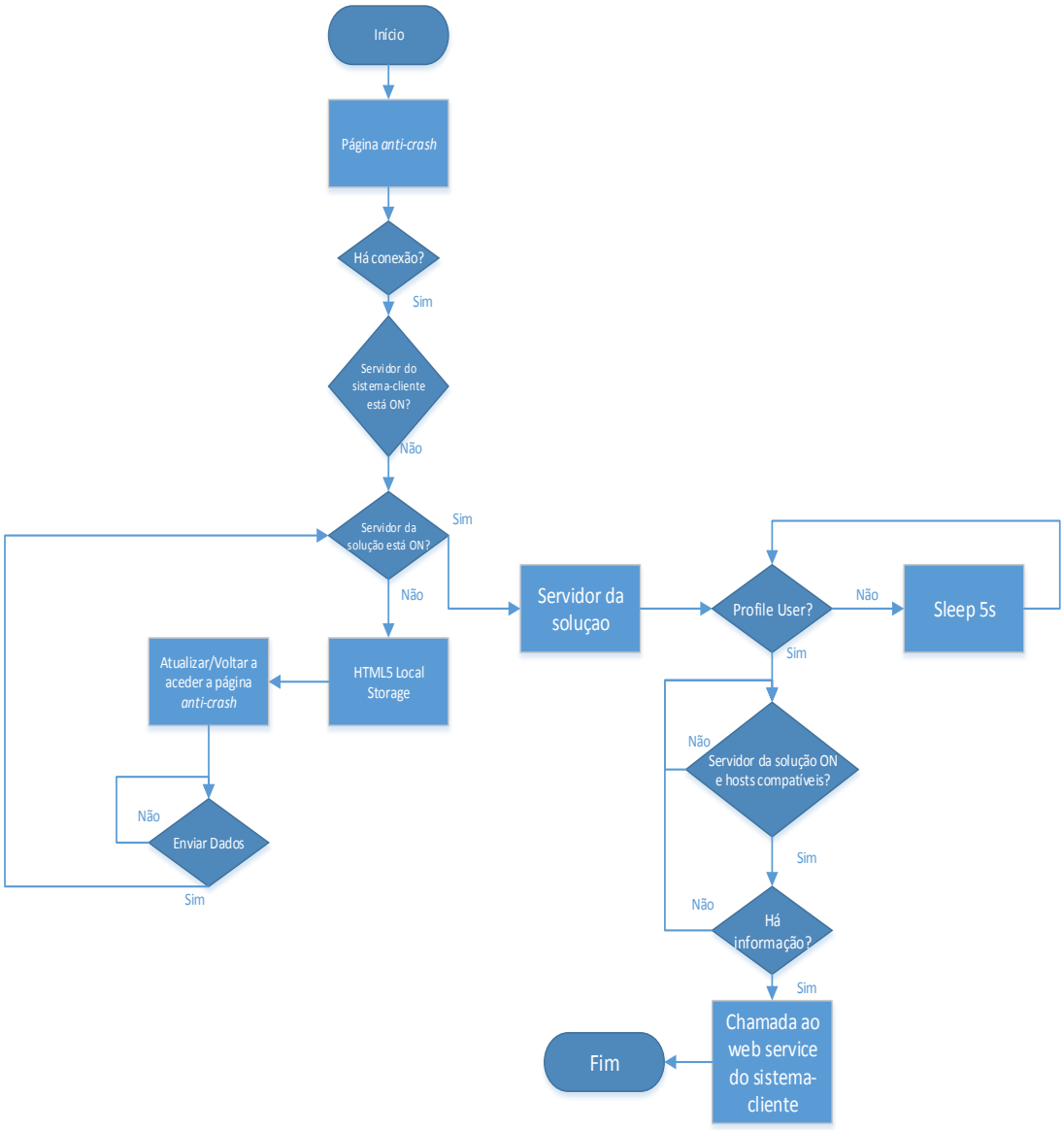

Figura 19 - Diagrama de Fluxo do Terceiro Caso de Uso.

A Figura 20 faz uma demonstração dos alertas e questões que são apresentadas e feitas ao utilizador caso o servidor do sistema-cliente e o servidor da solução estejam inacessíveis. Como mostra no diagrama de fluxo do terceiro caso de uso, sempre que o utilizador voltar a aceder a página web que se quer *anti-crash* e o servidor da solução já estiver acessível bem como os dados referentes a esta página *anti-crash* ainda não tiverem sido tratados, é feito um alerta de que existem dados por enviar e uma questão ao utilizador a saber se ele pretende enviar ou não estes dados onde diferentes decisões serão tomadas dependo da resposta a esta questão.

Figura 20 - Alertas Terceiro Caso de Uso.

Quarto caso de Uso

Caso o utilizador do sistema-cliente ficar sem conexão com a internet e o mesmo tiver acedido a página que se quer *anti-crash* anteriormente, os dados são armazenados no *browser* do utilizador (HTML5 local storage) e quando o mesmo voltar a aceder a esta página e a conexão com a internet já estiver estabelecida, é feita uma questão para saber-se se o utilizador quer enviar os dados que foram armazenados no seu *browser*. Se a resposta for negativa a mesma questão será feita novamente quando o utilizador voltar a aceder a página em questão. Caso a resposta seja positiva, estes dados são tratados. O diagrama de fluxo ilustrado na Figura 21, mostra como é tratado este caso de uso.

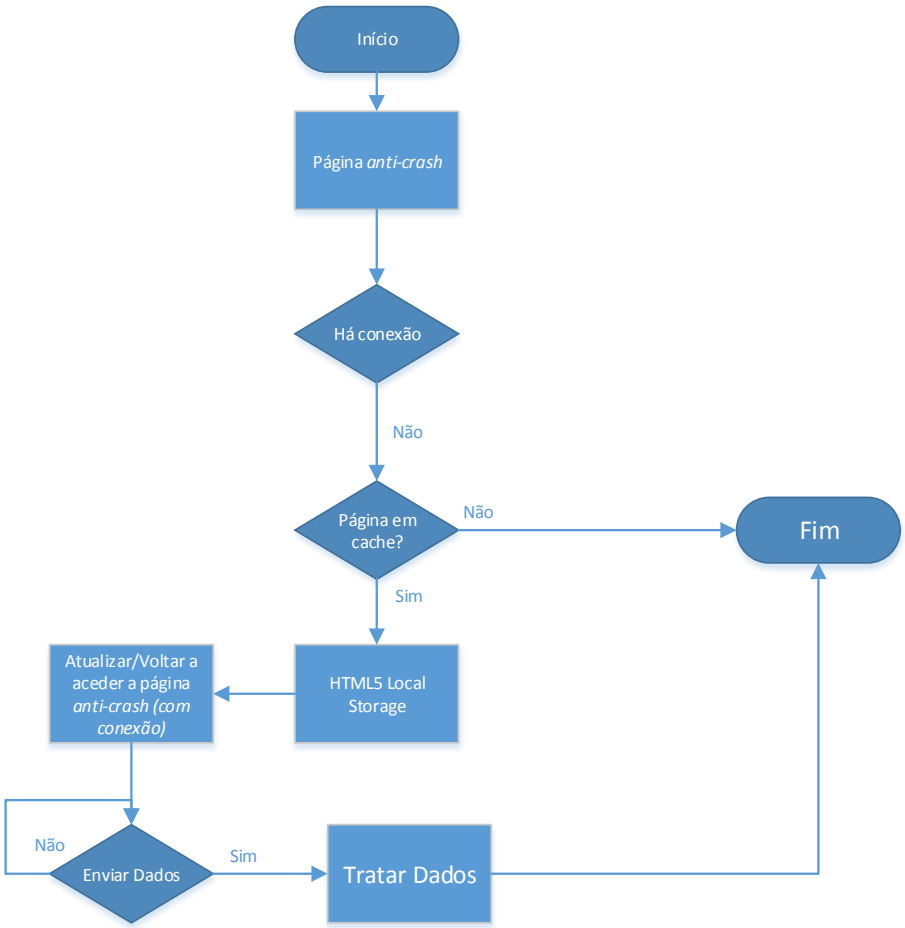

Figura 21 - Diagrama de Fluxo do Quarto Caso de Uso.

Porém isto só acontece caso o utilizador tenha o modo trabalhar *offline* ativo no seu navegador web ou então a solução do sistema-cliente tem de fazer o uso da interface Application Cache[6] do HTML5 para esta página *anti-crash* ficar em cache.

[6] Http://www.html5rocks.com/pt/tutorials/appcache/beginner/

4. Condições e Suporte

Neste capítulo vão ser apresentadas quais as condições necessárias para um cliente fazer o uso desta solução de forma correta bem como os porquês das decisões tomadas.

Será também apresentado como está disposta a plataforma de suporte *back office* assim como as suas funcionalidades e vistas que esta possui.

4.1 Condições da Solução

Esta solução apresenta um padrão, que têm um impacto mínimo, no qual todos os clientes que fizerem o uso desta têm de adaptar-se para encaixarem-se nas condições impostas pela solução. É fornecido aos sistemas-cliente um código de controlo básico para que estes possam ser identificados, como já foi citado em uma secção anterior. Assim, será possível fazer a recolha correta de dados relativos a esse sistema, bem como realizar e validar a conectividade dos servidores do cliente.

As condições que são necessárias para se fazer o uso correto desta solução são os seguintes:

- **Registo na plataforma de suporte:** para que os sistemas-cliente tenham um controlo sobre os seus dados, bem como disponibilizar informação para que seja gerado o código de controlo básico referente a cada serviço que inserirem de maneira a que a solução funcione de maneira correta.
- **API Web:** Para que os dados possam ser reencaminhados de volta para o servidor do sistema-cliente é preciso um serviço no lado do cliente que se encarregue disso, sendo assim este vai ser requisitado pelo serviço em nuvem assim que houver uma confirmação de que o servidor do sistema-cliente já se encontre acessível.

Tendo ainda em conta o quarto caso de uso, isto é, para que o utilizador do sistema-cliente ainda que sem conexão com a internet consiga parcialmente tratar dos seus dados, é necessário que este mesmo utilizador tenha o modo trabalhar *offline* ativo no seu navegador web ou então a solução do sistema-cliente tem de fazer o uso da interface Application Cache[7] do HTML5 para que esta página *anti-crash* fique em *cache*.

[7] Http://www.html5rocks.com/pt/tutorials/appcache/beginner/

4.2 Back Office

Para os utilizadores deste sistema terem um melhor controlo sobre os seus dados assim como fornecerem informação necessária para o correto funcionamento do mesmo, foi também elaborada uma plataforma suporte *back office*. Nesta plataforma é feito primeiro um registro do utilizador/cliente logo depois este é reencaminhado para a área de login em que clientes e administradores têm vistas distintas. Caso este utilizador já tenha um registo, o seu acesso para este suporte, é feito diretamente pela área de login.

4.2.1 Funcionalidades e Vistas

São apresentadas duas vistas distintas, a vista do cliente e a vista do administrador. Cada uma delas apresenta funcionalidades específicas.

A Tabela 4, mostra com distinção as funcionalidades dos diferentes tipos de utilizador deste suporte.

Funcionalidades	Administrador	Cliente
Ver e eliminar dados dos clientes (web services, números de identificação, *usernames*)	✓	✓
Ver e eliminar os utilizadores	✓	✗
Adicionar ou remover privilégios de administração	✓	✗
Alterar palavra passe	✓	✓
Validar e Registar webservices	✗	✓
***Logout* do sistema**	✓	✓

Tabela 4 - Tabela de Funcionalidades dos Utilizadores.

Para uma melhor compreensão do desenvolvimento realizado no *back office*, a Figura 23 representa um diagrama de contextualização das principais funcionalidades implementadas neste suporte.

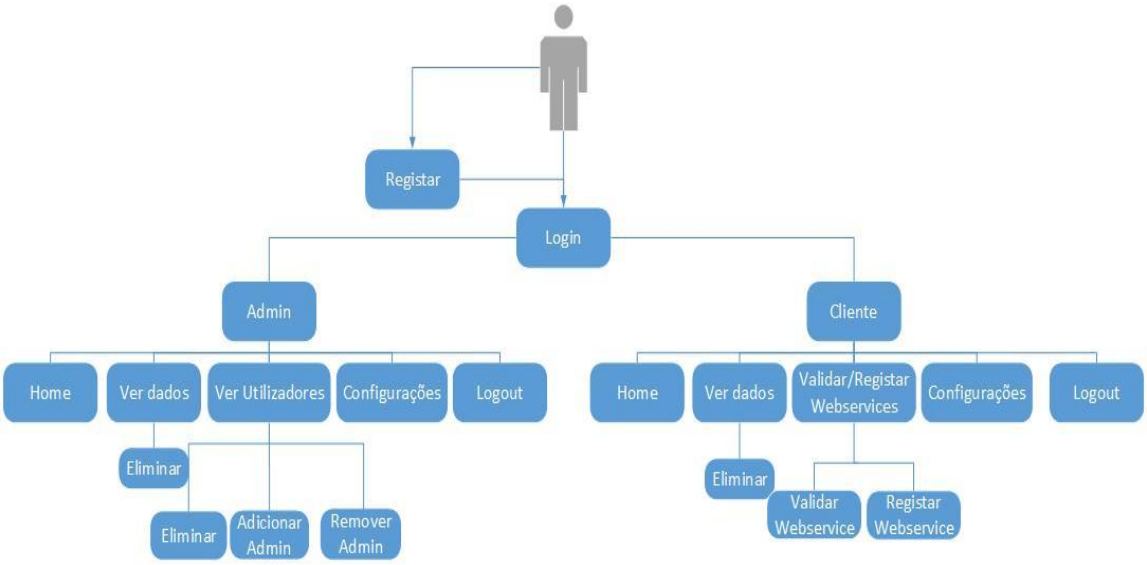

Figura 23 - Diagrama de funcionalidades do Back Office.

Cada uma destas funcionalidades vão ser abaixo ilustradas demonstrando assim como se apresentam as diferentes vistas e sequências deste suporte *back office* de acordo com as opções e operações escolhidas pelo utilizador.

Figura 24 - Sequência da Opção Registar para o Cliente.

Na Figura 24, podemos observar quais as vistas que aparecem quando se faz um registo bem como o que se sucede após este registo. Aparece uma vista em que aparecem diversas funcionalidades disponíveis para o cliente.

Figura 25 - Opção Login.

Na Figura 25 está representada uma funcionalidade que ambos utilizadores dispõe que é o *login*. Esta apenas está disponível para utilizadores que já estejam registados no sistema.

No diagrama de sequência representado na Figura 26, ilustra-se como funcionam os processos de registo, *login* e *logout* nesta aplicação de suporte.

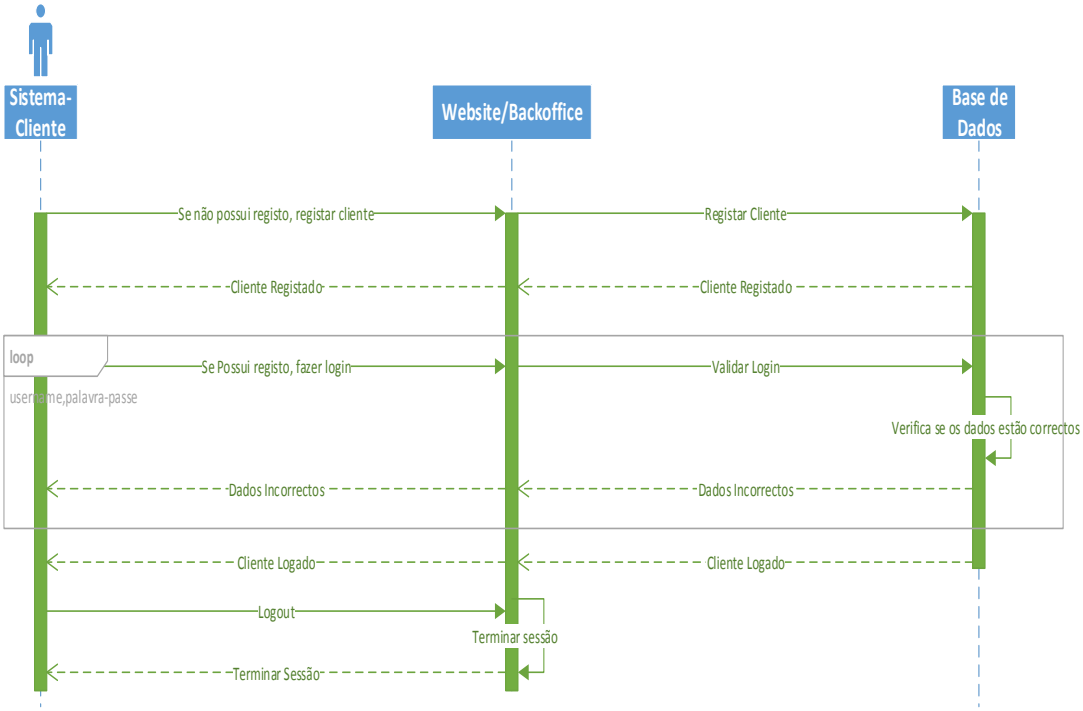

Figura 26 - Diagrama de Sequência Registo, Login, Logout.

Na Figura 27 observamos como se dispõe para o cliente os dados referentes a sua conta, apresentando também uma opção para eliminar estes dados. São apresentados o id do seu *web service* bem como dados referentes a este.

Figura 27 - Opção Ver Dados (Cliente).

Na Figura 28 está representada uma funcionalidade que ambos utilizadores dispõe que é a alteração da palavra passe.

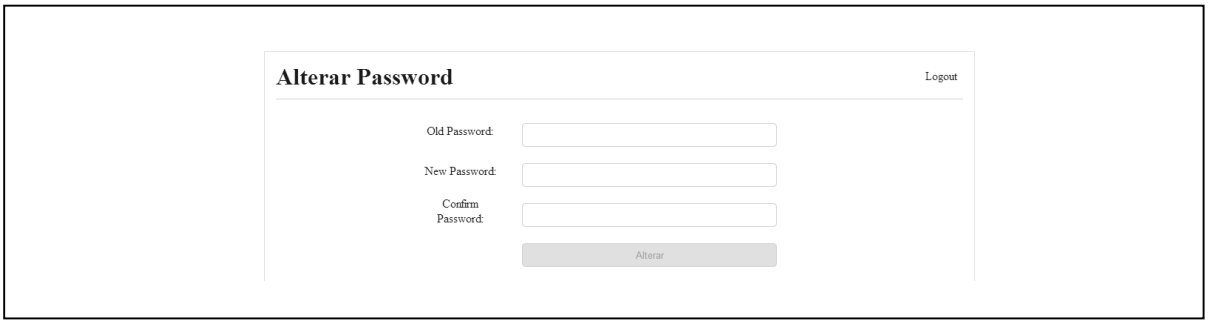

Figura 28 - Opção Alterar Password.

Na Figura 29 apresentam-se as vistas de acordo a funcionalidade que o cliente tem que é o registo ou validação do seu *web service*. Depois de preenchidos os dados necessários para o registo ou para a validação são apresentadas mensagens que indicam como ocorreu a opção escolhida. Para a opção registar apenas são necessários três dados, o nome e parâmetros do *web service* e o endereço do *host* e após o registo é fornecido um identificador para este *web service.*

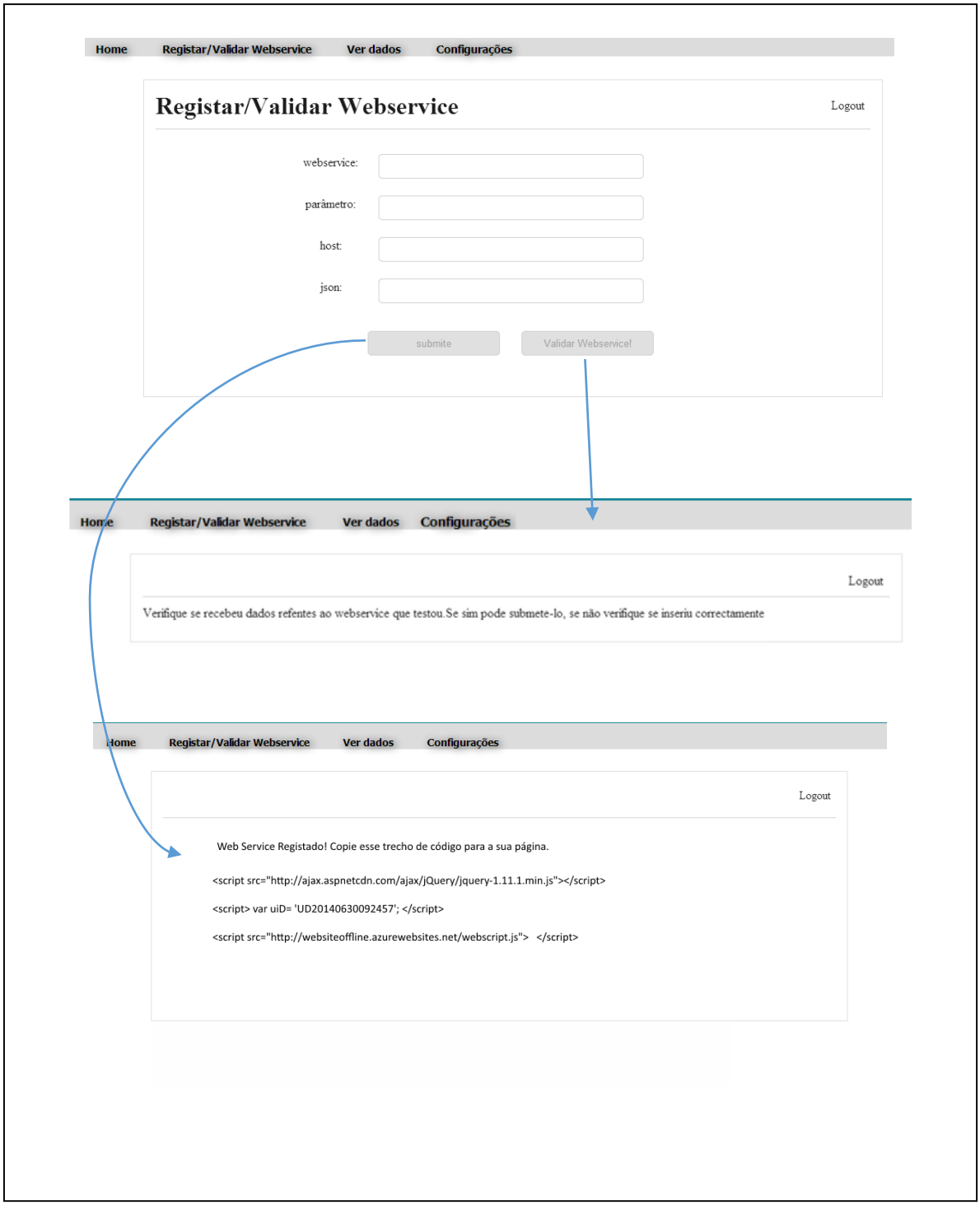

Figura 29 – Sequência da Opção Registar/Validar Web Service.

O diagrama de sequência representado na Figura 30, mostra como se comporta o suporte quando as opções de registar ou de validar o *web service* são acionadas.

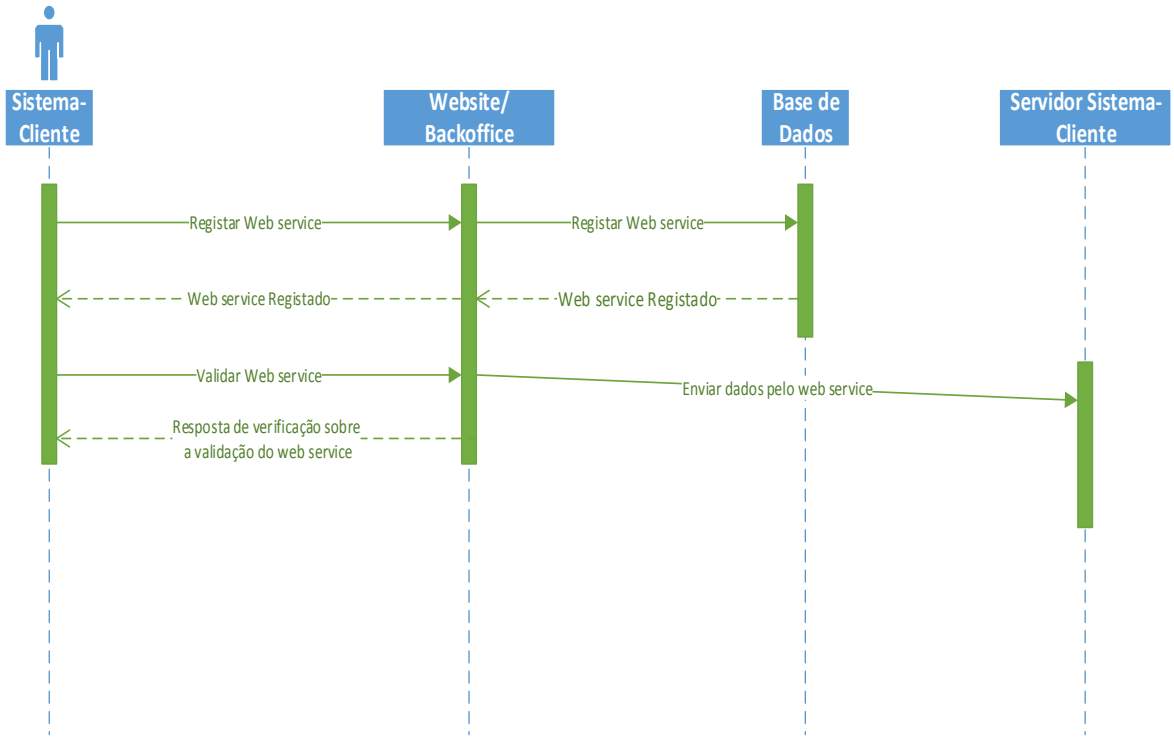

Figura 30 - Diagrama de Sequência Registar/Validar Web Service.

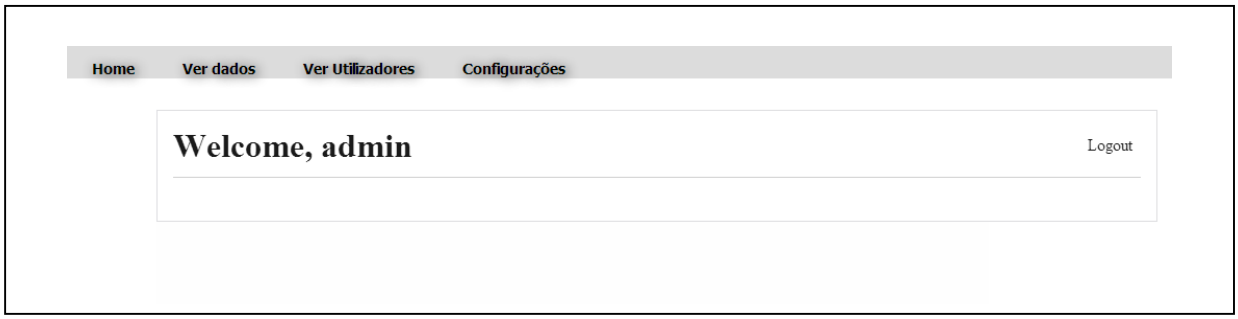

Figura 31 - Página Inicial do Administrador.

Na Figura 31 representa-se a vista que aparece ao administrador após o *login* efetuado, onde aparecem diversas funcionalidades disponíveis para um membro da administração.

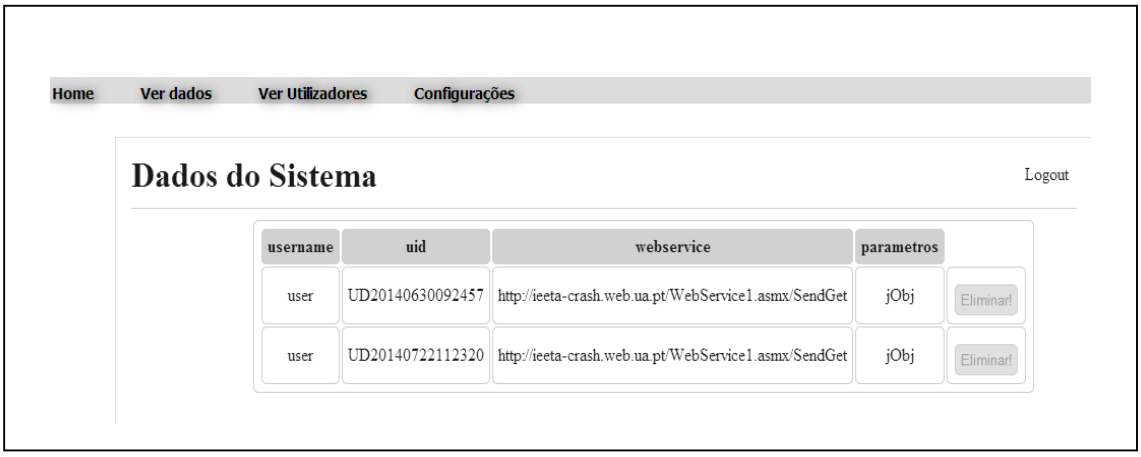

Figura 32 – Opção Ver dados (Administrador).

Na Figura 32 observamos como se dispõe para o administrador os dados referentes as contas dos clientes, apresentando também uma opção para eliminar estes dados. São apresentados os id dos *web services*, os nomes dos utilizadores bem como dados referentes a este.

Figura 33 – Opção Ver Utilizadores.

Na Figura 33 observamos como se dispõe para o administrador os dados referentes aos clientes, apresentando também uma opção para eliminar, bem como atribuir privilégios de administrador ou retirar estes mesmo privilégios.

4.3 Modelo de Base de Dados

Um dos aspetos a ter em conta na implementação de uma base de dados é que ela deve refletir cautelosamente as relações entre os recursos armazenados.

Atendendo aos requisitos necessários para um cliente fazer o uso correto da solução desta arquitetura assim como a disposição da própria arquitetura, a base de dados desta solução foi projetada de forma a suportar toda a informação relacionada a estes dois pontos.

Os parágrafos seguintes serão dedicados a explicar gradualmente como este modelo foi obtido.

Para guardar informação referente aos utilizadores da solução de suporte foi criada uma tabela de nome **Member** onde constam os seguintes atributos:

- **Id:** onde é armazenado o identificador da tabela Member;
- **Username:** onde é armazenada o nome de utilizador do utilizador do sistema;
- **Password:** onde é armazenada a palavra passe do utilizador do sistema;
- **Salt:** onde se armazena dados aleatórios que posteriormente acrescentarão complexidade a palavra passe;
- **Admin:** onde se indica se o utilizador possui ou não privilégios de administrador.

Estes utilizadores também têm associados a eles informações relativas aos serviços que eles adicionam as suas contas e para isso foi adicionada a tabela de nome **Infoid** onde estão presentes os seguintes atributos:

- **Id:** onde é armazenado o identificador da tabela Infoid;
- **RefIDMember:** onde é armazenada a chave estrangeira da tabela Infoid com relação a tabela Member;
- **UID:** onde é armazenado o número de identificação único referente aos dados de um cliente;
- **Webservice:** onde é armazenado o nome do web service;
- **Parameter:** onde é armazenado o nome do parâmetro do web service;
- **Host:** onde é armazenado o nome do host que utiliza esta metodologia.

E por fim, os dados que por inacessibilidade não são armazenados no servidor do cliente também precisam de ser tratados até que se resolva este problema e assim foi criada uma tabela de nome **Info**, onde fazem parte os seguintes atributos:

- **Id:** onde é armazenado o identificador da tabela Info**;**

- **Json:** onde é armazenado um json com a informação que não se conseguiu armazenar no servidor do cliente;
- **Host:** onde é armazenado o nome do host que utiliza esta metodologia;
- **UID:** onde é armazenado o número de identificação único referente aos dados de um cliente.

O modelo abaixo representado na Figura 34, demonstra como estão relacionadas estas tabelas entre si.

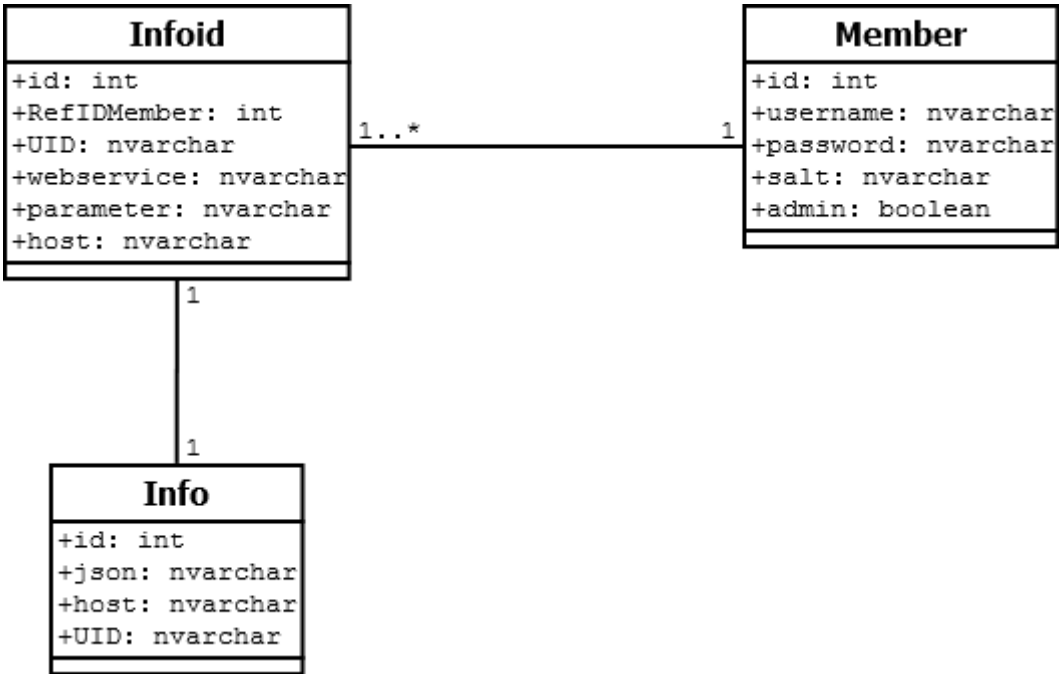

Figura 34 – Modelo de Base de Dados

5. Implementações e Testes

Neste capítulo serão apresentados os diversos componentes que foram implementados para construir e desenvolver a arquitetura apresentada até agora. Este dispõe de diversas secções que apresentam todo o processo de desenvolvimento, onde são apresentados todos os passos tomados durante esta fase de implementação, o porque da utilização de algumas tecnologias assim como estas foram solucionadas.

Também serão apresentados os testes que foram realizados a esta arquitetura.

5.1 Processo de Implementação

Após uma análise cuidada sobre os aspetos tratados no Estado da Arte, optou-se por hospedar esta arquitetura em um servidor em nuvem e ainda utilizar o serviço em nuvem para capacitar esta solução de acesso rápido e escalável. Como já citada na secção 3.2, esta arquitetura funcionará como um servidor de suporte para a solução principal e através de um *tracking code* que fará a invocação de determinados métodos para a verificação da conectividade dos servidores do cliente bem como os componentes que se encontram em *cloud* através de um script externo. É de se salientar também que uma API RESTful é que funciona como ponte de conexão entre o script externo e o servidor em nuvem. Como também já foram citados na secção 4.1, as condições necessárias para a utilização desta solução, implicam que o cliente faça o uso de uma API Web do tipo RESTful de maneiras a receber mensagens de requisição provenientes do serviço em nuvem.

Nunca foi objetivo desta dissertação estudar plataformas de computação em nuvem nem formas de hospedar serviços porém para hospedar o sistema desenvolvido que tem por base esta arquitetura foi escolhido o Windows Azure[8] pelo facto de este apresentar uma grande heterogeneidade no que toca as linguagens de programação, SDK's bem como as ferramentas que suporta. Como podemos ver nas seguintes referencias [26], [27], [28] [29], esta plataforma encaixa de maneira satisfatória para esta solução pois as linguagens assim como os recursos utilizados para a sua execução são suportados por esta plataforma. Além disso também foi feita uma análise comparativa com opções do mesmo género e que funcionalidades estas ofereciam com base nas seguintes referências [30], [31], [32], [33], [34], [35], [36], [37].

[8] https://azure.microsoft.com/pt-pt/

5.1.1 Serviço em Nuvem

Um serviço em nuvem é um serviço disponibilizado aos utilizadores através da Internet a partir de servidores de um fornecedor de computação em nuvem. Estes são projetados para fornecer um acesso fácil e escalável para aplicações, recursos e serviços, e são totalmente gerenciados por um fornecedor de serviços em nuvem.

Dos variados modelos de execução que a Azure oferece para soluções deste tipo, apresentada aqui nesta dissertação, se optou por fazer o uso de um serviço em nuvem principalmente para se conseguir uma solução de acesso rápido e escalável e também pelo tipo de tarefa que se pretendia executar. Entretanto, também foi feita uma análise destes modelos com base nas seguintes referências [38], [39], [40], [41] para se chegar a conclusão de que modelo seria melhor para esta solução.

Tal como as outras opções de computação do Azure, estes serviços em nuvem dependem de máquinas virtuais. Esta tecnologia oferece duas opções de máquinas virtuais ligeiramente diferentes:

- **Instâncias de *Web roles***: que fornece um servidor web de Serviços de Informações da Internet (IIS) dedicado, usado para hospedar aplicações web de front-end.
- **Instâncias de *Worker roles***: onde as aplicações podem executar tarefas assíncronas, de longa execução ou perpétuas, independentemente de interação do utilizador ou de entrada.

Em suma uma instância de *web role* corre em uma variante do Windows Server com o IIS, enquanto instâncias de *worker roles* executam a mesma variante do Windows Server sem IIS. Um serviço em nuvem do Azure conta com uma combinação dessas duas opções. A Figura 35 ilustra uma ideia de como é um serviço em nuvem do Azure [38].

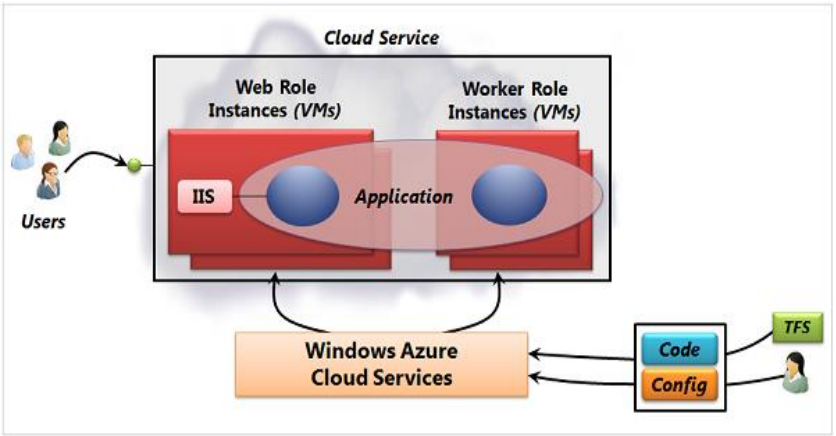

Figura 35 - Serviços em Nuvem do Azure [38].

Ainda que as aplicações sejam executadas em máquinas virtuais, é importante entender que os serviços em nuvem fornecem PaaS e não IaaS. Com o IaaS, primeiro deve-se criar e configurar o ambiente em que a aplicação será executada, em seguida, fazer o *deploy* da aplicação para esse ambiente. Já no PaaS é como que se este ambiente já existisse. A única coisa que tem de se fazer é o *deploy* da aplicação e ao contrário do IaaS não há necessidade de preocupar-se com a gestão nem as versões deste [38]. A referência [42] faz um complemento sobre o que foi daqui descrito nesta subsecção.
Desta forma, para esta arquitetura já citada na secção 3.2, é necessário um serviço que esteja sempre a executar tarefas perpétuas, uma funcionalidade que a instância de *worker role* realiza bastante bem acrescentando agilidade assim como facilidade na sua gestão. A Figura 15 da secção 3.2 ilustra a sequência de funcionamento deste.

5.2 Script Externo

A nível informático, um script é um conjunto de instruções em código que servem para controlar um determinado programa ou aplicação. Nesta solução são feitas uma série de verificações antes de se acionar a solução de suporte. Estas verificações são feitas através de um script externo que se encontra no tracking code que é disponibilizado aos clientes desta solução, e este contém um conjunto de instruções que possibilitam ver se há ou não a necessidade desta solução ser acionada. Por outras palavras este funciona como um controlador.

Este script pode dizer-se que esta estruturado em duas camadas, uma onde são armazenados os dados e uma outra onde são tratados esses dados.

Ele é o ponto de partida para indicar em que situação encontra-se o sistema o que depois leva para uma solução de acordo a cada situação.

A Figura 36 mostra como se comporta este controlador dependendo das situações encontradas:

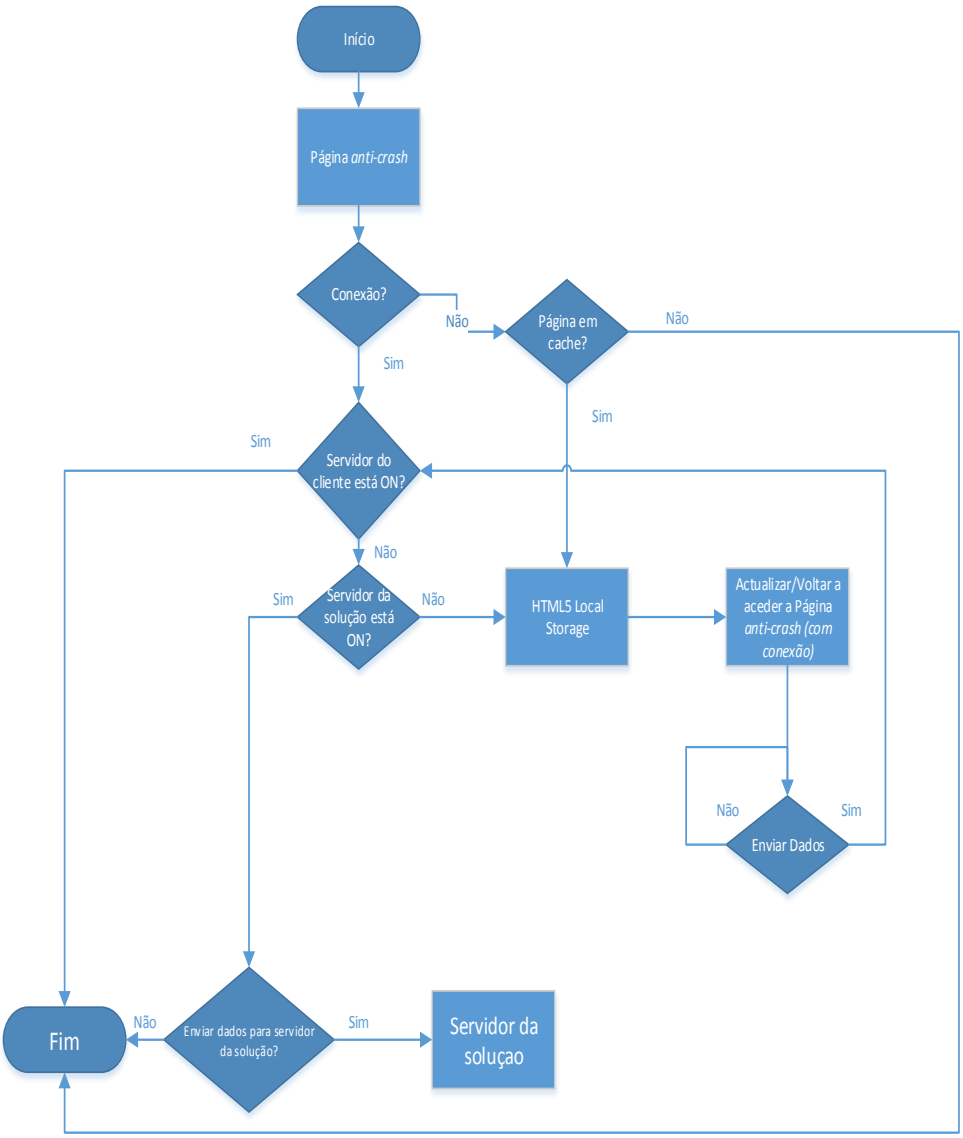

Figura 36 - Diagrama de Fluxo do comportamento do Script Externo.

De acordo com os casos de uso já apresentados na secção 3.1, este script têm como principais objetivos:

- Direcionar os dados do utilizador para:
    - O servidor do sistema-cliente, caso este esteja disponível;
    - O servidor da solução, caso o servidor do sistema-cliente esteja indisponível e o utilizador ainda assim pretender enviar os seus dados;
- Manter os dados em cache caso ambos os servidores estejam inacessíveis ou quando o utilizador do sistema-cliente não tem conexão com a internet. Esta parte específica faz parte da camada onde são armazenados estes dados a serem tratados, a camada de dados. Estes ficam armazenados em Local Storage, que é uma das mais novas funcionalidades do HTML5, até que um dos servidores fique disponível.

Estes dados circulam em formato JSON, formação esta que é feita de forma automática pelo método **JSON.stringify()** que tem como parâmetro os dados que vêm da página *anti-crash* do website do cliente que por sua vez são serializados pelo método **$.serializeObject** que tem como função converter formulários em um objeto JSON válido para poder ser utilizado em uma aplicação javascript.

Quando o servidor da solução está acessível e o servidor do cliente não, estes dados são encaminhados para a solução por um serviço RESTful. Na secção seguinte será explicado o porque da preferência de um serviço deste tipo.

A Figura 37 faz uma demostração de como está estruturado este controlador. Apresenta uma camada lógica onde ficam as funções bem como regras deste controlador e que por sua vez faz requisições por vezes a camada de dados onde se situam os dados que são armazenados pelo Local Storage.

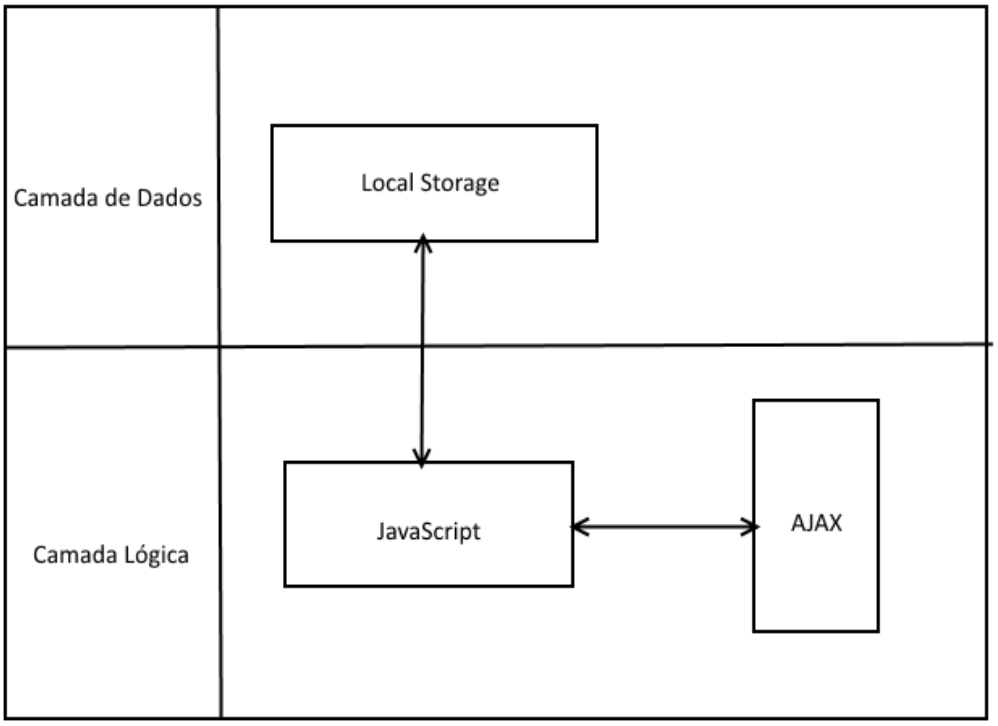

Figura 37 - Estrutura do Controlador da Solução.

5.3 API RESTful

No contexto de desenvolvimento web, uma API é um conjunto de mensagens de requisição e resposta HTTP que são geralmente expressados nos formatos XML ou JSON.

A arquitetura REST é executada principalmente através de HTTP e é comumente utilizado em APIs Web. Geralmente consiste em clientes, servidores, recursos e um vocabulário de operações HTTP conhecidos como métodos de solicitação [43].

As APIs RESTful são coleções de recursos que aderem aos princípios e limitações da arquitetura REST. Geralmente incluem [43]:

- A URI base para a API;
- O tipo de formato (tais como XML ou JSON) suportado pela API;
- O vocabulário das operações de solicitação, como GET, PUT ou DELETE;
- Componentes de hipertexto orientado;

Para haver uma troca de mensagens assim como informações entre o servidor do cliente e a solução aqui apresentada, são usadas API's web RESTful pelo simples facto de a arquitetura do tipo REST ser mais fácil do que outras abordagens como o SOAP, evitando ambiguidades e também por esta ser escalável, simples, fácil de modificar e a nível de performance é bem melhor também. É de realçar também que o REST suporta diferentes formatos de dados enquanto o SOAP apenas suporta o formato XML e nesta solução os dados circulam em formato JSON.

5.4 Testes

Neste ponto, serão analisados de uma forma geral os vários testes que foram realizados nesta solução. Vão ser apresentados testes de validação que têm como objetivo garantir o correto funcionamento desta, assim como alguns testes de compatibilidade que asseguram que a plataforma de suporte desta arquitetura assim como o controlador e consequentemente as ações que ele aufere são compatíveis com os diferentes tipos de navegadores web. Serão apresentados alguns diagramas que ilustrarão como a arquitetura deve-se comportar face a realização destes testes de validação.

5.4.1 Testes de Validação

Estes testes de validação são necessários para garantir que o produto atende aos requisitos especificados no final da fase de desenvolvimento. Por outras palavras, para garantir que o produto é construído de acordo com as necessidades do cliente. Estes não são automatizados mas são bastantes importantes pois permitem verificar se os casos de uso implementados funcionam como o esperado para o sistema.

Foram realizados os seguintes testes:

- **1º Teste – Servidor sistema-cliente acessível**

  Para a elaboração deste teste foram realizados os seguintes passos:

  - O servidor sistema-cliente foi ligado;
  - Foi verificado o comportamento do controlador quando o utilizador submeteu os seus dados.

A Figura 38 ilustra sequencialmente como a solução se comporta quando é realizado este teste. Visto que o servidor sistema-cliente está acessível, o controlador não tem necessidade de evocar a solução de suporte logo a ação termina.

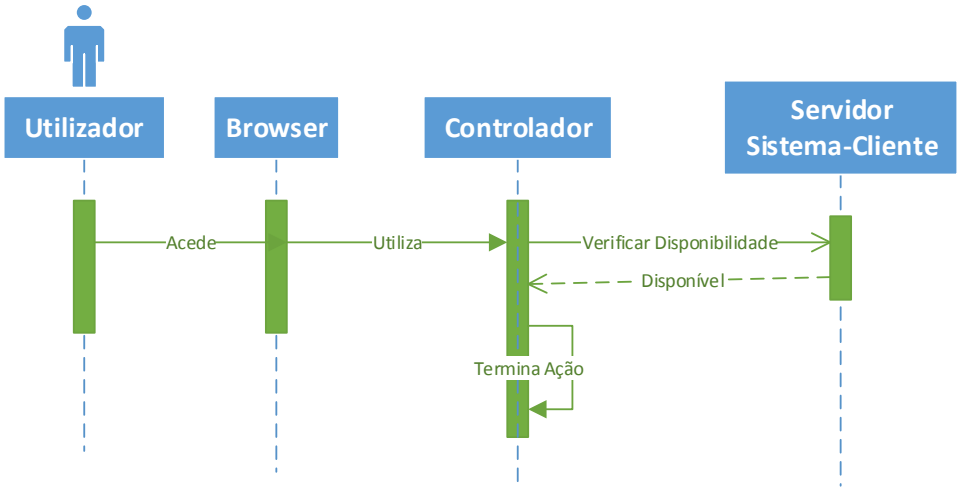

Figura 38 - Sequência do 1º Teste.

- **2º Teste – Servidor sistema-cliente inacessível e o utilizador não pretende enviar os seus dados para um servidor temporário**

  Para elaboração deste teste foram realizados os seguintes passos:

  - O servidor sistema-cliente foi desligado;
  - Foi verificado o comportamento do controlador quando o utilizador submeteu os seus dados;
  - Foi verificado o comportamento do controlador face a resposta negativa do utilizador para enviar os dados para um servidor temporário face a inacessibilidade do servidor sistema-cliente.

A Figura 39 ilustra as sequências que a solução apresenta quando é realizado este teste. Uma vez que o servidor sistema-cliente encontra-se indisponível, o controlador tem de verificar se o servidor da solução encontra-se disponível e tem de alertar posteriormente ao utilizador de que o servidor do sistema-cliente se encontra indisponível e se pretende que os seus dados sejam encaminhados para um servidor temporário. Quando o utilizador rejeita esta opção o controlador tem de terminar a ação.

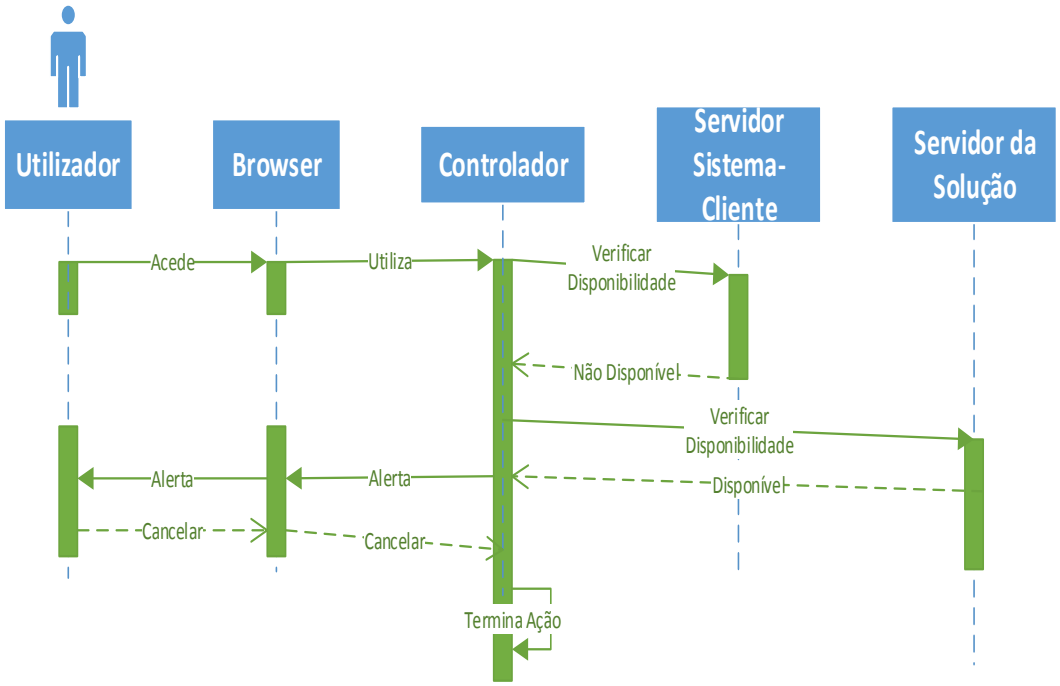

Figura 39 - Sequência do 2º Teste.

- **3º Teste – Servidor sistema-cliente inacessível e o utilizador pretende enviar os seus dados para um servidor temporário**

  Para elaboração deste teste foram realizados os seguintes passos:

  - O servidor sistema-cliente foi desligado;
  - Foi verificado o comportamento do controlador quando o utilizador submeteu os seus dados;
  - Foi verificado o comportamento do controlador face a resposta positiva do utilizador para enviar os dados para um servidor temporário face a inacessibilidade do servidor cliente;
  - Foi verificado se os dados submetidos pelo utilizador foram corretamente inseridos na base de dados da solução de suporte;
  - Foi verificado o comportamento do serviço em nuvem.

A Figura 40 ilustra as sequências que a solução apresenta quando é realizado este teste. Uma vez que o servidor do cliente encontra-se indisponível, o controlador tem de verificar se o servidor da solução encontra-se disponível e tem de alertar posteriormente ao utilizador de que o servidor sistema-cliente encontra-se indisponível e se pretende que os seus dados sejam encaminhados para um servidor temporário. Quando o utilizador aceita esta opção, o controlador tem de fazer uma chamada a API Web e os dados têm de ser inseridos na base de dados da solução de suporte. O serviço em nuvem tem de ir verificando se existem dados para tratar e se os servidores referentes a estes dados já se encontram disponíveis.

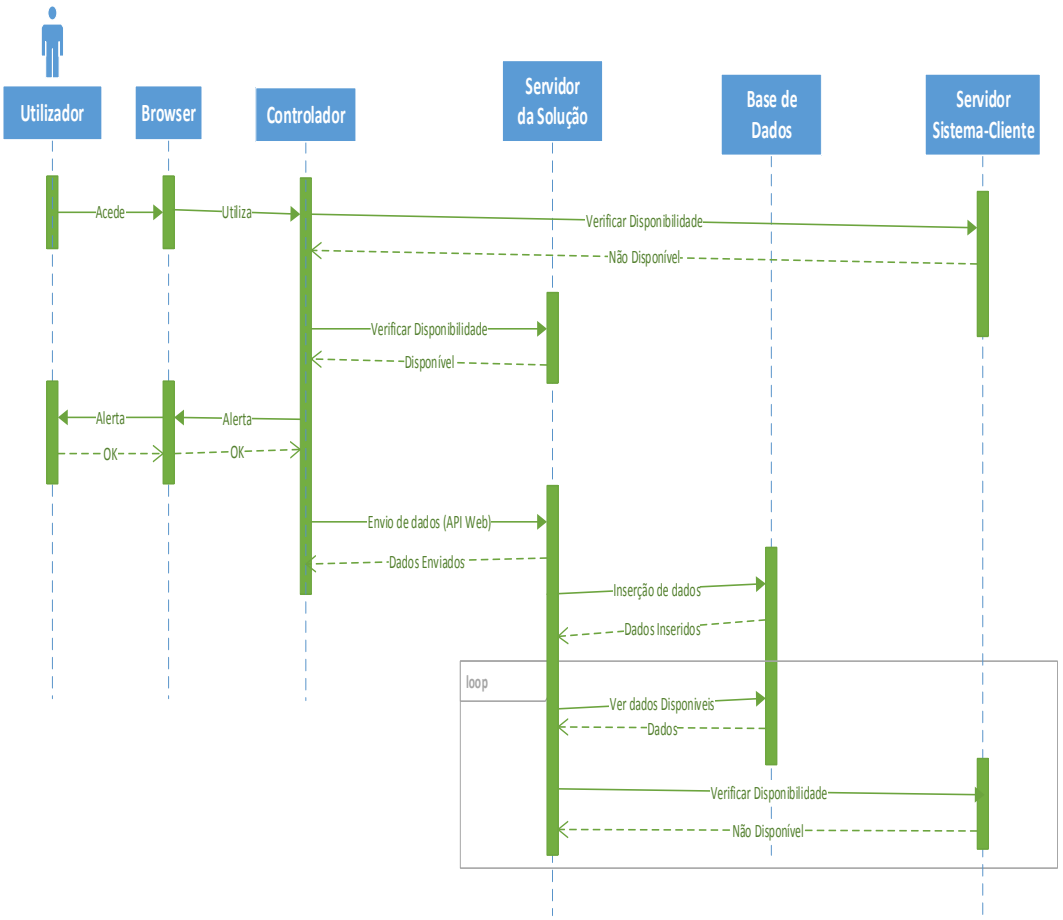

Figura 40 – Sequência do 3º Teste.

- **4º Teste – Funcionamento do Serviço em nuvem**

Este serviço em nuvem verifica a disponibilidade do servidor sistema-cliente, verifica também se o ID que se encontra no tracking code da página é válido e caso seja vai buscar a informação referente a este ID.

Para a elaboração deste teste foram realizados os seguintes passos:

- O servidor sistema-cliente foi ligado;
- Foi verificado se o serviço em nuvem confirmava o registo do ID;
- Foi verificado se o serviço em nuvem verificava a disponibilidade do servidor sistema-cliente;
- Foi verificado se a API Web do cliente era chamada.

A Figura 15 da secção 3.2 ilustra as sequencias que o serviço em nuvem apresenta quando este teste é realizado, ou seja quando o servidor sistema-cliente volta a estar disponível. O serviço em nuvem tem de verificar constantemente se existem dados a tratar e se existirem, verificar se os servidores referentes a estes

dados se encontram disponíveis. Estando o servidor disponível este tem de ir buscar informação referente a API Web deste cliente e de seguida enviar os dados para o servidor deste cliente. Depois de enviados, o serviço em nuvem tem de apagar estes dados da base de dados do servidor da solução.

- **5º Teste – Servidor sistema-cliente e da solução inacessíveis**

  Para a elaboração deste teste foram realizados os seguintes passos:

  - O servidor sistema-cliente foi desligado;
  - O servidor da solução foi desligado;
  - Foi verificado o comportamento do controlador quando o utilizador submeteu os seus dados;
  - Foi verificado se os dados são ou não enviados pelo controlador quando o utilizador volta a aceder a página. Havendo dados para envio, foi verificado se este alertava o utilizador e se o seu comportamento, em função das respostas do utilizador estava conforme o especificado.

  A Figura 41 ilustra as sequências que a solução apresenta quando é realizado este teste. Uma vez que o servidor sistema-cliente e da solução encontram-se indisponíveis, o controlador:

  1. Tem de alertar o utilizador desta situação;
  2. Tem de seguida fazer o uso do HTML5 Local Storage para armazenar os dados que estavam para ser submetidos pelo utilizador.

  Quando este mesmo utilizador voltar a aceder a página *anti-crash* o controlador:

  1. Tem de alertar que existem dados por enviar;
  2. O utilizador ao concordar, o controlador tem de verificar a disponibilidade dos servidores e caso o servidor sistema-cliente estiver disponível estamos perante o caso do 1º teste e caso apenas o servidor da solução estiver disponível estamos perante o 2º ou 3º teste dependendo da decisão do utilizador de enviar ou não os seus dados para o servidor temporário.

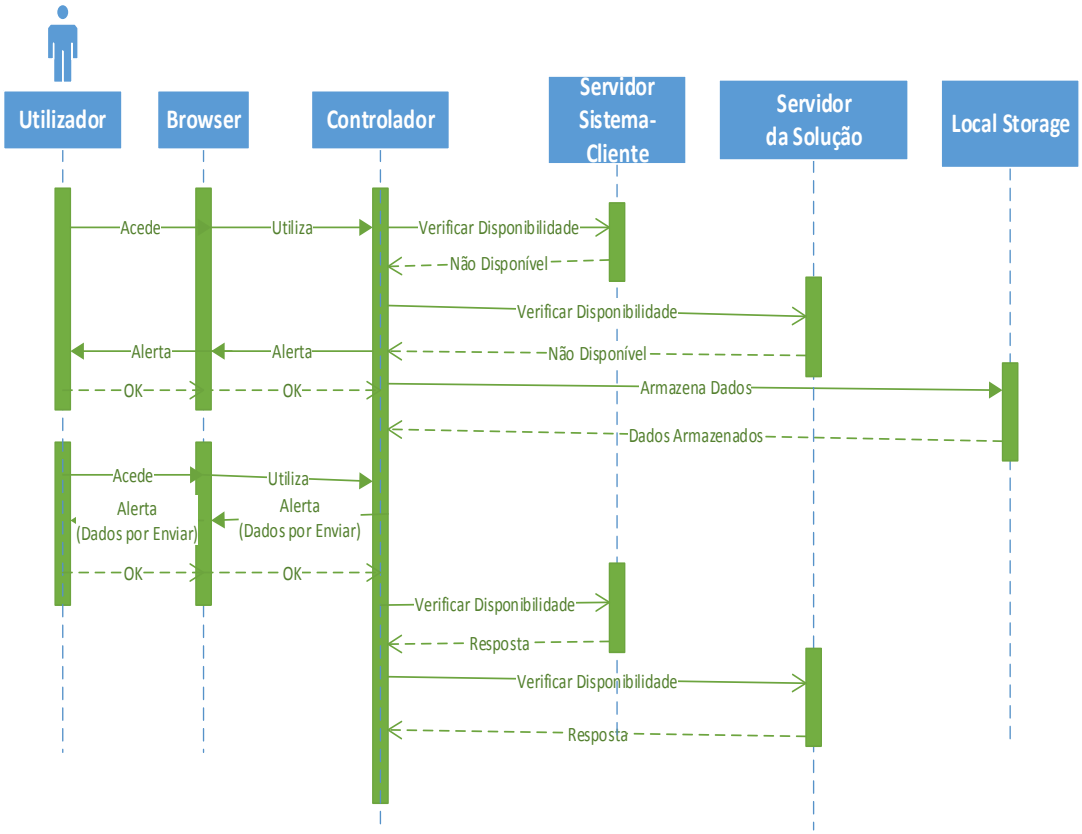

Figura 41 - Sequência do 5º Teste.

- **6º Teste – Utilizador sem conexão com a internet**

  Para a elaboração deste teste foram realizados os seguintes passos:

  - A página foi posta em cache;
  - O modo trabalhar *offline* foi ativado ou o sistema-cliente faz o uso da interface Application Cache[9] do HTML5;
  - A conexão com a internet foi desligada;
  - Foi verificado o comportamento do controlador quando o utilizador submeteu os seus dados;
  - Foi verificado se os dados são ou não enviados pelo controlador quando o utilizador volta a aceder a página (já com conexão com a internet). Havendo dados para envio, foi verificado se este alertava o utilizador e se o seu comportamento, em função das respostas do utilizador estava conforme o especificado.

[9] Http://www.html5rocks.com/pt/tutorials/appcache/beginner/

Quando é realizado este teste, os passos que têm de ser realizados pela solução são os seguintes:

Uma vez que não há conexão com a internet, o controlador:

1. Tem de verificar se a página está em cache;
2. Tem de seguida fazer o uso do HTML5 Local Storage para armazenar os dados que estavam para ser submetidos pelo utilizador.

Quando este mesmo utilizador voltar a aceder a página *anti-crash (*já com conexão com a internet*)*, o controlador:

3. Tem de alertar que existem dados por enviar;
4. O utilizador ao concordar, o controlador tem de verificar a disponibilidade dos servidores e caso o servidor sistema-cliente estiver disponível estamos perante o caso do 1º teste e caso apenas o servidor da solução estiver disponível estamos perante o 2º ou 3º teste dependendo da decisão do utilizador de enviar ou não os seus dados para o servidor temporário.

Ao longo do decorrer do desenvolvimento desta solução foram realizados estes testes com o fim de verificar que os casos de uso apresentados na secção 3.1 funcionam como o esperado.

5.4.2 Testes de Compatibilidade

Nestes tipos de teste é verificado o desempenho da plataforma de suporte para esta arquitetura nos diferentes navegadores web existentes, o funcionamento do Local Storage que é utilizado pelo controlador da solução desta arquitetura e ainda o desempenho das ações que este controlador aufere. Estes testes são necessários para definir que tipo de navegadores web podem ou não utilizar esta plataforma e que são compatíveis com as ações do controlador. Estes foram realizados com o auxílio de uma aplicação web de nome **browser shots**[10] que tem como objetivo fazer testes de compatibilidade de aplicações web e também com base na informação fornecida nas especificações apresentadas por estes navegadores. A Tabela 5 mostra os diversos pontos que foram tratados na elaboração destes testes.

[10] Http://browsershots.org/

	Inicialização	Funcionalidades funcionam corretamente?	É possível navegar sem o uso de plugins?	Local Storage	Application Cache	Modo Offline
	✓	✓	✓	✓	✓	✗
	✓	✓	✓	✓	✓	✓
	✓	✓	✓	✓	✓	✗
	✓	✓	✓	✓	✓	✗
	✓	✓	✓	✓	✓	✓

Tabela 5 - Testes de Compatibilidade.

Apenas foram feitos os testes nas últimas versões dos navegadores web apresentados aqui na Tabela 5. Podemos verificar que apenas dois navegadores conseguem suportar a opção de trabalhar em modo *offline*.

6.Conclusão e Trabalho Futuro

Hoje em dia, diferentes empresas bem como diferentes negócios apresentam diferentes níveis de risco associados à perda de informação e ao tempo de inatividade. Existem diversas soluções que podem ser utilizadas para fornecer vários níveis de proteção em relação às necessidades e características específicas de cada um desses negócios. A solução ideal consistia em não apresentar qualquer tempo de inatividade e não permitir a perda de dados. Estas soluções existem mas por vezes são dispendiosas, e os seus custos devem ser ponderados face ao impacto que um desastre destes pode causar a este negócio ou empresa.

6.1 Conclusão

Os utilizadores dos sistemas de computação têm-se tornado cada vez mais exigentes mostrando-se mais dispostos a enfrentar os custos adicionais das técnicas de tolerância a falhas. Deve-se portanto reconhecer exigências quanto a confiabilidade e a disponibilidade de uma determinada aplicação ou *software,* saber escolher qual a solução de menor custo que supera estas exigências assim como ter condições de desenvolver os mecanismos complementares de tolerâncias a falhas para se atingir a confiabilidade desejada.

A tolerância a falhas compreende muitas das técnicas que permitem aumentar a qualidade dos sistemas de computação apesar de não garantir o funcionamento correto na presença de todo e qualquer tipo de falha e as suas técnicas envolverem algum grau de redundância e de tornarem sistemas maiores e mais dispendiosos, ela permite alcançar a confiabilidade e a disponibilidade para estes sistemas [21].

O trabalho aqui apresentado nesta dissertação mostra que existem muitos desafios e muitos aspetos a ter em conta no desenho de uma solução de alta disponibilidade principalmente num cenário de catástrofe onde os níveis de risco incrementam consideravelmente em relação as situações consideradas normais. A arquitetura aqui descrita foi desenhada tendo em conta estes aspetos e tenta suprimir todas as barreiras que levam à não disponibilidade de um sistema fazendo assim com que o utilizador continue a fazer o uso deste ainda que de forma condicionada.

6.2 Problemas Encontrados

Durante a implementação da solução foram encontrados problemas ao nível do controlador quando se tratasse do quarto caso de uso, caso este em que o utilizador fica sem conexão com a internet. Nesta dissertação foram propostas duas formas para se solucionar este caso porém estas não cumprem em algumas partes o que era esperado, e uma das soluções (trabalhar em modo *offline*) não é suportada por todos os navegadores web. Ainda nestas duas formas de implementar esta solução foi notado que por vezes algumas das ações que são executadas pelo controlador não são disparadas. Isto deve-se ao facto de o evento que indica o estado do navegador web (`navigator.onLine`) apresentar alguns problemas em alguns navegadores web, como por exemplo o caso do navegador Firefox que apenas retorna false se o modo trabalhar offline estiver ativado neste browser e no navegador Chrome para Linux este evento é sempre retornado a true.

Também foi reparado que as cookies nunca podem ser limpas pois deste modo a API LocalStorage do HTML5 vai ser limpa também, isto é, os dados que esta armazenou serão apagados o que irá comprometer o funcionamento da solução para os casos em que não há conexão com a internet por parte do utilizador e quando o servidor do sistema-cliente não se encontra disponível.

6.3 Trabalho Futuro

Atualmente neste trabalho existe um aspeto de segurança que não se encontra totalmente funcional, no que toca a possibilidade de o JSON que circula com a informação que é submetida pelo cliente ser intercetado por terceiros. Esta pode ser tratada com a autenticação de entidades com recurso a certificados porém acrescentaria uma maior sobrecarga ao cliente.

É necessário também aprimorar o funcionamento da solução no que toca ao quarto caso de uso, isto é, tentar colmatar os problemas encontrados nesta solução.

7.Bibliografia

[1] C. F. Ribeiro, “Devmedia : Sistemas Críticos,” [Online]. Available: http://www.devmedia.com.br/sistemas-criticos/18952. [Acedido em 2014].

[2] R. Antônio, “TI Especialistas : Sistemas operacionais de alta disponibilidade,” 8 july 2013. [Online]. Available: http://www.tiespecialistas.com.br/2013/07/sistemas-operacionais-de-alta-disponibilidade/. [Acedido em 2014].

[3] I. Sommerville, Software Engineering, 8ª ed., Harlow: Pearson Addison-Wesley, 2007.

[4] E. Marcus e H. Stern, Blueprints for High Availability - Designing Resilient Distributed Systems, 2ª ed., Indianapolis, IN: Wiley Pub, 2003.

[5] P. S. Weygant, Clusters for High Availability - A Primer of HP Solutions, 2ª ed., Prentice Hall, 2001.

[6] M. Becker, M. P. d. Cruz, V. C. d. Oliveira e D. Bianchini, “AVALIAÇÃO DA APLICAÇÃO DOS RECURSOS EM TECNOLOGIA DA INFORMAÇÃO PARA MANTER A DISPONIBILIDADE DOS SISTEMAS CRÍTICOS DE NEGÓCIO,” pp. 3-4, 2012.

[7] H. Clancy, “Most common cause of SMB downtime? The answer may surprise you,” 11 February 2013. [Online]. Available: http://www.zdnet.com/most-common-cause-of-smb-downtime-the-answer-may-surprise-you-7000011137/.

[8] R. Hat, “Red Hat Enterprise Linux 4: Guia de Segurança,” 2005 . [Online]. Available: http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-sg-pt_br-4/ch-netprot.html. [Acedido em 2014].

[9] V. Solutions, “Products: Double-Take Availability,” Vision Solutions Inc., [Online]. Available: http://www.visionsolutions.com/Products/DT-Avail.aspx. [Acedido em 2014].

[10] Arcserve, “Products: Arcserve High Availability Solutions,” Arcserve, [Online]. Available: http://www.arcserve.com/us/products-solutions/products/arcserve-high-availability-solutions.aspx. [Acedido em 2014].

[11] V. Solutions, “Products: Vision Products Overview,” Vision Solutions Inc, [Online]. Available: http://www.visionsolutions.com/products/vision-products-overview.aspx. [Acedido em 2014].

[12] V. Solutions, “Products: Vision Products Overview,” Vision Solutions Inc, [Online]. Available: http://www.visionsolutions.com. [Acedido em 2014].

[13] C. ARCserve, “CA ARCserve High Availability,” Arcserve, [Online]. Available: http://www.stratus.com/~/media/Stratus/Files/Library/Collateral/CA-ARCserve-High-Availability-r16.pdf. [Acedido em 2014].

[14] C. ARCserve, “CA ARCserve® Replication and High Availability - Administration Guide,” 2010.

[15] T. D. Business, “Tiger Direct Software CA ARCserve high Availability,” [Online]. Available: http://biz.tigerdirect.com/p/software/software/server-so-ware/ca-arcserve-high-availability-r15-includes-ca-arcserve-assured-recovery-windows-enterprise. [Acedido em 2014].

[16] E. Enterprise, “Software: everRun,” Stratus Technologies, [Online]. Available: http://www.stratus.com/Products/Software/everRun. [Acedido em 2014].

[17] S. Avance, “Software:AvanceHA,” Stratus Technologies, [Online]. Available: http://www.stratus.com/Products/Software/AvanceHA. [Acedido em 2014].

[18] G. Electric, “Highavailability,” General Electric Company, [Online]. Available: http://www.ge-ip.com/highavailability. [Acedido em 2014].

[19] N. TM, “Solutions: Neverfail,” Tech-21 Systems Ltd, [Online]. Available: http://www.tech-21.com.hk/solutions_neverfail.asp#High_Availability_and_Disaster_Recovery. [Acedido em 2014].

[20] N. T. Labs, “CA ARCserve r16 vs Double-Take Availability 6.0,” 2012.

[21] T. S. Weber, “Um roteiro para exploração dos conceitos básicos de tolerância a falhas,” Instituto de Informática – UFRGS.

[22] J. Silveira, “Sistemas em Tempo Real Módulo 3: Tolerância a Falhas,” DETI - UFC.

[23] H. Albuquerque, J. Vitor, M. Cireno e T. Lima, “Centro de Informática UFPE,” [Online]. Available: http://www.cin.ufpe.br/~jvob/introducao.html. [Acedido em 2014].

[24] J. Simões, “Lunacloud,” 2 Fevereiro 2014. [Online]. Available: http://blog.lunacloud.com/vps-vs-dedicated-server-vs-cloud-server/. [Acedido em 2014].

[25] HTML5, “HTML5 Application Cache,” [Online]. Available: http://www.w3schools.com/html/html5_app_cache.asp. [Acedido em 2014].

[26] M. Azure, “What is Azure,” Microsoft, [Online]. Available: https://azure.microsoft.com/pt-pt/overview/what-is-azure/. [Acedido em 2014].

[27] C. Knighton e Z. Richardson, “gigaom: the great debate windows azure vs amazon web service,” 2011. [Online]. Available: https://gigaom.com/2011/09/04/the-great-debate-windows-azure-vs-amazon-web-services/. [Acedido em 2014].

[28] “Programmers Stackexchange Windows Azure vs Amazon ec2 vs Google App Engine,” [Online]. Available: http://programmers.stackexchange.com/questions/64727/windows-azure-vs-amazon-ec2-vs-google-app-engine. [Acedido em 2014].

[29] D. Shinder, “Techrepublic 10 things you should know about deploying Windows Azure VMs in a hybrid IT environment,” 2013. [Online]. Available: http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-deploying-windows-azure-vms-in-a-hybrid-it-environment/. [Acedido em 2014].

[30] M. Azure, “Campaigns : Azure vs AWS,” Microsoft, [Online]. Available: https://azure.microsoft.com/pt-pt/campaigns/azure-vs-aws/. [Acedido em 2014].

[31] A. W. Services, “Produtos e serviços,” Amazon Web Services, Inc, 2014. [Online]. Available: http://aws.amazon.com/pt/products/.

[32] N. Corporation, “The State of Cloud Storage,” Natick, 2014.

[33] C. Spectator, “Cloud Server Performance:: A Comparative Analysis of 5 Large Cloud IaaS Providers,” 2013. [Online]. Available: http://cloudspectator.com/2013/06/cloud-server-performance-a-comparative-analysis-of-5-large-cloud-iaas-providers-3/. [Acedido em 2014].

[34] A. W. Services, “What is AWS,” Amazon Web Services, Inc, [Online]. Available: http://aws.amazon.com/pt/what-is-aws/. [Acedido em 2014].

[35] O. O. Neto e R. C. Freitas, “COMPUTAÇÃO EM NUVENS,VISÃO COMPARATIVA ENTRE AS PRINCIPAIS PLATAFORMAS DE MERCADO,” Ceará, 2011.

[36] Cloudyn, “What’s behind the cloud vendors AWS, GCE and Azure?,” 2014. [Online]. Available: https://www.cloudyn.com/wp-content/uploads/2014/08/What%E2%80%99s-behind-the-cloud-vendors-AWS-GCE-and-Azure.pdf. [Acedido em 2014].

[37] B. Butler, “Cloud computing showdown: Amazon vs. Rackspace (OpenStack) vs. Microsoft vs. Google,” Network World, Inc, December 2012. [Online]. Available: http://www.networkworld.com/article/2161595/cloud-computing/cloud-computing-showdown--amazon-vs--rackspace--openstack--vs--microsoft-vs--google.html. [Acedido em 2014].

[38] M. Azure, “Documentation: Fundamentals Application Models,” Microsoft, [Online]. Available: http://azure.microsoft.com/pt-pt/documentation/articles/fundamentals-application-models/. [Acedido em 2014].

[39] M. Azure, “What is a cloud service?,” Microsoft, [Online]. Available: http://azure.microsoft.com/en-us/documentation/articles/cloud-services-what-is/. [Acedido em 2014].

[40] M. Azure, “Azure Websites, Cloud Services, and Virtual Machines comparison,” Microsoft, [Online]. Available: http://azure.microsoft.com/en-us/documentation/articles/choose-web-site-cloud-service-vm/. [Acedido em 2014].

[41] S. D. Mattia, “Microsoft Azure Cloud Services Part 1: Introduction,” April 2014. [Online]. Available: http://justazure.com/microsoft-azure-cloud-services-part-1-introduction/. [Acedido em 2014].

[42] S. D. Mattia, “Microsoft Azure Cloud Services Part Two: Anatomy of a Cloud Service,” June 2014. [Online]. Available: http://justazure.com/microsoft-azure-cloud-services-part-2-anatomy-cloud-service/. [Acedido em 2014].

[43] S. Objects, “Resources: Articles & Whitepapers,” Service Objects, Inc., [Online]. Available: http://www.serviceobjects.com/resources/articles-whitepapers/why-rest-popular. [Acedido em 2014].

[44] D. Scott, “Making Smart Investments to Reduce Unplanned Downtime,” Gartner, 1999.