

UNIVERSIDADE DO VALE DO TAQUARI - UNIVATES
CURSO DE ENGENHARIA DE SOFTWARE

GLFS: SOLUÇÃO PARA GERENCIAMENTO DE ARQUIVOS EM NUVEM

Tales Igor Ebert

Lajeado, junho de 2018

Tales Igor Ebert

GLFS: SOLUÇÃO PARA GERENCIAMENTO DE ARQUIVOS EM NUVEM

Trabalho de Conclusão de Curso
apresentado ao curso de Engenharia de
Software da Universidade do Vale do
Taquari – Univates, para obtenção do título
de Bacharel em Engenharia de Software.

Orientador: Prof. Me. Fabrício Pretto

Lajeado, junho de 2018

RESUMO

As soluções de computação devem atender um modelo de negócio de alto crescimento. Conceitos de escalabilidade e elasticidade estão presentes nos mais diversos segmentos, tais como, infraestrutura, processos e serviços. No mercado de desenvolvimento de software não é diferente, existe uma grande preocupação quanto ao crescimento em escala das aplicações e dos recursos necessários para atender a demanda de dados a serem armazenados e processados. A computação em nuvem pode contribuir para este cenário à medida que sejam criadas ferramentas que permitam às organizações utilizarem estes recursos de forma transparente. Na área de gerenciamento de dados e informações de grande volume, as soluções disponíveis no mercado são poucas e limitadas, e ao mesmo tempo em que resolvem alguns problemas criam novos, como por exemplo, a dependência de um serviço único e proprietário. A solução GLFS visa tornar simples armazenar, recuperar, migrar e analisar os objetos de qualquer formato em qualquer lugar. Esses recursos vão possibilitar o gerenciamento em escala, de forma segura e mantendo a privacidade do conteúdo, independentemente do local armazenado. Esse trabalho é um estudo de caso que busca qualificar o processo de gerenciamento de dados de clientes com a ferramenta GLFS em uma empresa de desenvolvimento de software. Serão apresentados os resultados da implementação e implantação da solução, e os benefícios decorrentes dos novos recursos.

Palavras-chave: Armazenamento de objetos binários; banco de dados; computação em nuvem; escalabilidade; otimização; segurança.

ABSTRACT

Computing solutions must serve a high growth business model. Scalability and elasticity concepts are present in the most diverse segments, such as infrastructure, processes and services. In the software development market is no different, there is great concern about the scale-up of applications and the resources required to meet the demand for data to be stored and processed. Cloud computing can contribute to this scenario as tools are created that enable organizations to use these resources transparently. In the area of high volume data and information management, the solutions available in the market are few and limited, and at the same time solving some problems create new ones, such as dependence on a single and proprietary service. The GLFS solution aims to make it simple to store, retrieve, migrate and parse objects of any format anywhere. These features will enable scalable management, securely and maintaining content privacy, regardless of where it is stored. This work is a case study that seeks to qualify the process of managing customer data with the GLFS tool in a software development company. The results of the implementation and implementation of the tool will be presented, along with the benefits derived from the new resources.

Key-Words: Storage of binary objects; database; cloud computing; Scalability; Optimization; safety.

LISTAS DE ILUSTRAÇÕES

LISTA DE FIGURAS

Figura 1: Cenário antes do GLFS	35
Figura 2: Cenário depois da implantação do GLFS.....	36
Figura 3: Diagrama de classes - GLFS - Bloco das classes de negócio do GLFS.	43
Figura 4: Diagrama de sequência da interação entre classes internas da API para criação de um depósito.	46
Figura 5: Figura 4: Diagrama de sequência da interação entre classes internas da API no envio de um arquivo.....	46
Figura 6: Diagrama de sequência da interação entre classes interna da API na recuperação de um arquivo.	47
Figura 7: Diagrama de sequência da interação entre classes interna do fluxo de execução do método responsável por fazer uma limpeza do GLFS.....	48
Figura 8: Exemplo de código mostrando a simplicidade no uso do GLFS.....	54
Figura 9: Gráfico gerado com base nos metadados e logs do GLFS.	58

LISTA DE QUADROS

Quadro 1: Características do PostgreSQL.....	28
Quadro 2: Comparação entre GLFS e soluções similares.....	32

LISTA DE ABREVIATURAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	Application Programming Interface
FTP	File Transfer Protocol
GPL	General Public License
GLFS	GreenApp Large file System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
PHP	PHP Hypertext Preprocessor
REST	Representational State Transfer
SaaS	Software as a Service
SSL	Secure Sockets Layer
SGBD	Sistema de Gerenciamento de Banco de Dados
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
URI	Unified Resource Identifier
WWW	Word Wide Web
WS	Web Services

SUMÁRIO

1	INTRODUÇÃO	1
1.1	Motivação	3
1.2	Objetivo	4
1.2.1	Objetivos específicos	4
1.3	Organização do trabalho	5
2	REFERENCIAL TEÓRICO	6
2.1	Engenharia de Software	6
2.1.1	Padrões de projeto de software	7
2.1.1.1	Padrões de criação	8
2.1.1.2	Padrões estruturais	8
2.1.1.3	Padrões comportamentais	8
2.1.1.4	Padrões de persistência	9
2.1.1.5	Padrões de controle	9
2.1.1.6	Padrões de apresentação	9
2.1.2	Arquitetura de Software	10
2.1.3	Arquitetura de comunicação para aplicações Web	10
2.1.4	Modelo cliente-servidor	11
2.1.5	Modelo MVC (<i>Model View Control</i>)	11
2.1.6	Arquitetura orientada a SOA (<i>Service-Oriented Architecture</i>)	12
2.1.6.1	SOAP (<i>Simple Object Access Protocol</i>)	12
2.1.6.2	Web Services Description Language (WSDL)	13
2.1.6.3	REST (<i>Representational State Transfer</i>)	13
2.1.7	SaaS - Software como Serviço	14
2.2	Computação em Nuvem	15
2.2.1	Riscos Serviços de armazenamento em nuvem	15
2.2.2	Riscos dos serviços terceirizados em nuvem	16
2.3	Sistemas de arquivos	17
2.3.1	Banco de dados e SGBD	18
2.3.2	Catálogo de metadados de um banco de dados	19
2.3.3	Transações	19
2.4	Modelo de referência de segurança	20
2.4.1	Integridade	21

2.4.2	<i>Hashing</i>	21
2.4.3	Confiabilidade	21
2.4.4	Disponibilidade	22
2.4.5	Auditoria	23
2.4.6	Criptografia	23
2.4.6.1	Algoritmo AES	24
2.4.7	Intrusão ou acesso não autorizados	24
2.4.7.1	Detecção e prevenção de intrusão	25
3	METODOLOGIA	26
3.1	Ferramentas	27
3.1.1	PostgreSQL	27
3.1.2	Características do PostgreSQL	27
3.1.3	Limitações do PostgreSQL	28
3.2	Linguagem de programação - PHP (Hypertext Preprocessor)	29
3.3	Tecnologias utilizadas como apoio	29
3.4	Estudo de caso do Software ERP GreenApp	30
3.5	Trabalhos relacionados	30
4	IMPLEMENTAÇÃO DO GLFS	33
4.1	Cenário do ERP GreenApp antes do GLFS	33
4.2	Cenário do ERP GreenApp com a implantação do GLFS	35
4.3	Levantamento de requisitos	36
4.3.1	Requisitos funcionais	36
4.3.2	Requisitos não funcionais	38
4.3.3	Limitações do escopo do projeto	39
4.4	Desenvolvimento	39
4.4.1	Gerenciamento dos dados	39
4.4.1.1	Controles do GLFS ao armazenar um arquivo	41
4.4.2	Modelagem/Diagrama de classes da API	43
4.4.2.1	Controles do GLFS ao recuperar um arquivo	44
4.4.3	Diagrama de sequência da API	45
4.4.3.1	Acesso a URL de <i>download</i> do arquivo	49
4.4.3.2	Outros métodos no GLFS	49
4.4.4	Segurança do GLFS	51
4.4.4.1	Comunicação	51
4.4.4.2	Criptografia dos arquivos	51
4.4.4.3	Verificação de vírus	52

4.4.4.4	Token, controle de <i>download</i>	52
4.4.5	Extensibilidade	53
4.4.6	Transparência e facilidade na utilização do GLFS pelo ERP.....	54
4.4.7	Métodos para usos específicos	55
4.4.8	Estatísticas	55
4.4.9	Testes e validações	57
4.4.10	Avaliação.....	58
5	CONSIDERAÇÕES FINAIS	60
	REFERÊNCIAS	62

1 INTRODUÇÃO

Com o avanço tecnológico das últimas décadas, os softwares exigem mais recursos computacionais, ou seja, maior capacidade de processamento de modo que as informações sejam obtidas rapidamente. Essas informações ajudam na tomada de decisão e garantem que a organização sobreviva no mercado global (ROB; CORONEL, 2011).

Para Pressman (2011) o software precisa lidar com o avanço tecnológico do hardware, que evolui muito mais rápido devido ao entusiasmo da mídia e entusiasmo popular, logo os aplicativos precisam amadurecer e ultrapassar o hardware para obter sucesso.

Buyya, Yeo e Vnugopal (2009) descreve a Computação em Nuvem como uma forma de fornecer os serviços da TI (Tecnologia da Informação) conforme a demanda e com pagamento conforme o uso. A Nuvem fornece aos usuários um servidor centralizado e compartilhado que é acessado entre todos os membros da organização internamente, mas que por razões de limitações de recursos e acessibilidade, com o passar do tempo foram adaptados e transformados em uma grande rede que chamamos de Nuvem, seja o serviço público ou privado.

A Computação em Nuvem surge como um paradigma da computação, e o nome Nuvem é uma definição para a Internet, baseia-se em um serviço que oculta toda a complexidade por traz da infraestrutura de hardwares e softwares

necessários, que normalmente são alocados em servidores compartilhados, sejam eles próprios ou terceirizados (BUYA; YEO; VNUGOPAL, 2009).

Para Pressman (2011) os aplicativos baseados na Web possuem os seguintes atributos:

- Uso intensivo da rede: Deve atender a necessidade de acesso e a comunicação mundial (isto é, Internet), o acesso não mais limitado ao ambiente local de uma corporação;
- Simultaneidade: Possibilitar um grande número de usuários acessarem ao mesmo tempo muitas vezes a mesma informação;
- Carga não previsível: O número de usuários que se conectam aleatoriamente a qualquer momento, por exemplo: algumas dezenas de usuários no início de uma manhã, a centenas ao final do dia;
- Desempenho: Usuários procuram desempenho e usuários modernos não querem esperar pelo sistema, muitas vezes podendo até procurar outra solução;
- Disponibilidade: Aplicações Web devem estar disponíveis 27 horas por dia e 7 dias por semana;
- Evolução contínua: Diferente de aplicativos de desktop lançam atualizações e versões planejadas, as aplicações Web normalmente possuem atualizações frequentes, às vezes no intervalo de horas;
- Segurança: Pelo fato de estarem disponíveis na Internet, torna-se difícil controlar quem tem acesso ao ambiente Web, logo existe uma necessidade de proteger o conteúdo ou acesso com fortes mecanismos e técnicas de segurança;
- Estética: A parte inegável das soluções Web que pesa aos olhos do usuário final tanto quanto a parte técnica.

Com a Computação em Nuvem, os aplicativos e soluções Web precisam de uma infraestrutura contínua, e para sua comercialização surgiram alguns modelos de comercialização, sendo eles:

- IaaS (Infraestrutura como Serviço): Oferece uma infraestrutura para executar alguma solução ou aplicação;

- PaaS (Plataforma como Serviço): Oferece plataformas para desenvolvimento de soluções e produtos;
- SaaS (Software como Serviço): Oferece acesso ao sistema mediante ao pagamento de uma assinatura.

O SaaS é uma forma de comercialização voltado para Web mundialmente utilizado, é um modelo interessante para o consumidor final e para o fornecedor, pois esconde toda complexidade de infraestrutura necessária para o serviço e possibilita automatizar todos os processos para manter o software, tais como: constantes atualizações, cópias de segurança, monitoramento, criação de novos ambientes e manutenção remota.

1.1 Motivação

O crescimento no volume de informação requer sistemas computacionais capazes de armazenar, recuperar e garantir a integridade dos dados, dessa forma, a computação em todos os aspectos deve ser flexível e escalável para atender a demanda de armazenamento necessária. É difícil encontrar um setor que não faça uso permanente dos dados e das informações a qualquer momento, isso reforça o conceito que muito além de armazenar, de forma segura e íntegra os sistemas precisam ser capazes de recuperar os dados e informações. O software de gestão empresarial GreenApp¹ passa por esses mesmos desafios, e nos dias atuais passa a lidar com um problema específico no armazenamento de arquivos no formato digital.

Atualmente, existem várias ferramentas que têm como objetivo gerenciar e armazenar arquivos em nuvem e equipamentos específicos para armazenamento locais conhecidos como *storages*, porém, ao optar por uma ou outra solução, passamos a depender da solução escolhida que muitas vezes não atende a todos os requisitos necessários, principalmente no que diz respeito a monitorar estatísticas de armazenamento, integração entre diferentes soluções e a extensibilidade a diferentes tecnologias. As ferramentas que mais se aproximam do objetivo deste estudo de caso têm um alto custo de investimento, e uma vez optando-se por uma

1 GreenApp é um software ERP de gestão empresarial comercializado na modalidade SaaS.

solução proprietária esse investimento se transforma em custo mensal e tende a crescer no mesmo ritmo que a utilização dos recursos.

O armazenamento de arquivos dentro do GreenApp é comercializado e cobrado com base no uso dos recursos de armazenamento, logo, é vital para os administradores saberem as estatísticas de uso dos recursos do espaço utilizado de cada cliente da GreenApp. Sendo assim, tornou-se uma motivação construir a ferramenta GLFS (*GreenApp Large File System*), uma ferramenta que faz o controle, gerenciamento e monitoramento dos arquivos anexos no GreenApp ERP.

1.2 Objetivo

O objetivo geral do presente trabalho é apresentar uma solução para gerenciar o armazenamento de arquivos em nuvem. Esse armazenamento não deve ficar limitado ao uso de recursos próprios, mas sim ser flexível para integrar-se com diferentes soluções de armazenamento em nuvem disponível no mercado.

1.2.1 Objetivos específicos

Para atingir o objetivo principal, foram mapeados os seguintes objetivos específicos que deverão ser atendidos:

- a) Compreender os princípios de armazenamento de dados, no que diz respeito à integridade dos dados, escalabilidade e segurança da informação;
- b) Extrair informações de uso dos recursos computacionais agrupados por vários formatos sejam por cliente, por módulo, por local de armazenamento;
- c) Desenvolver uma solução (GLFS) para o gerenciamento de arquivos em nuvem de fácil acoplamento ao ERP GreenApp a outros sistemas;

- d) Possuir uma ferramenta no GLFS de *back-end* capaz extrair estatísticas de uso dos recursos, gerenciamento e manutenção da solução;
- e) Validar a solução no ambiente de produção;
- f) Avaliar os resultados obtidos.

1.3 Organização do trabalho

Com a finalidade de facilitar o entendimento do trabalho, os capítulos são apresentados na seguinte formato.

O capítulo 2 explicará a base de conhecimento e os conceitos utilizados no desenvolvimento da solução, tais como, conceitos sobre escalabilidade, flexibilidade, segurança, integridade, conceitos utilizados pelos bancos de dados atuais no que diz respeito à integridade dos dados e comunicação pela Internet de forma segura.

No capítulo 3 é apresentada a metodologia do trabalho, contendo o método de pesquisa e as ferramentas utilizadas no desenvolvimento.

O capítulo 4 contempla o desenvolvimento do trabalho, onde são tratadas as etapas do projeto e as funcionalidade da ferramenta. Ainda no capítulo 4 é apresentada a validação da ferramenta contendo os testes e estatísticas de uso.

Por fim no capítulo, 5 são apresentadas as considerações finais e possíveis trabalhos futuros sobre o tema.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão abordados os principais conceitos necessários para o desenvolvimento da solução proposta. Foram realizados estudos em artigos e livros. Destacam-se os conceitos sobre as propriedades ACID utilizados em bancos de dados, estratégias da Engenharia de Software no desenvolvimento Web, conceitos de computação em nuvem, escalabilidade, flexibilidade, integridade, segurança de informações na Internet e comunicação de dados em redes.

2.1 Engenharia de Software

A Engenharia de Software é um processo que reúne um conjunto de práticas, métodos e ferramentas, com o objetivo de desenvolver softwares de qualidade. A Engenharia de Software capacita os profissionais para criar soluções de alta qualidade atendendo a prazos e requisitos (SOMMERVILLE, 2012).

Uma das abordagens mais eficazes em software é o reuso não se limitando ao código, mas sim a solução por completo. Sempre que possível deve-se criar linhas de componentes, aplicações ou serviços que possam ser utilizados em diferentes aplicações. Um exemplo clássico é um *framework*, que é utilizado por várias aplicações de forma simples, rápida e fácil. Geralmente essas linhas de produtos de softwares que são projetadas para serem comuns entre várias aplicações sem passar por modificação de código fonte. Linhas de produtos e

softwares de serviços, nem sempre precisam ser criadas utilizando uma abordagem de orientação a objetos (SOMMERVILLE, 2012).

2.1.1 Padrões de projeto de software

Sommerville (2011) afirma que padrões de projetos facilitam a construção de aplicações de forma eficiente no que diz respeito à reutilização de código em diferentes sistemas. Ainda Fowler (2003) sugere que um padrão não é simplesmente um código ou algoritmo, e sim a descrição abstrata de uma solução para resolver um determinado problema que se repete em diversas situações, sendo que essa solução pode ser utilizada muitas vezes em diferentes contextos.

Para Pressmann (2011), os padrões de projeto ajudam na obtenção de softwares de qualidade, através não somente na redução da manutenção de código, mas também na melhoria da comunicação da equipe e curva de aprendizagem ao se reutilizar classes e bibliotecas.

Gamma e Salgado (2008) descreve 23 padrões no livro *Design Patterns: Elements of Reusable Object-Oriented Software*, e Folwer (2003) 51 padrões catalogados no livro *Patters of enterprise application architecture*. Folwer (2003) classifica os padrões de grupos, sendo eles: padrões de criação, padrões estruturais, padrões comportamentais, padrões de persistência, padrões de controle e padrões de apresentação.

- Padrões de criação: Abstract Factory, Builder, Prototype, Factory Method e Singleton;
- Padrões estruturais: Adapter, Composite, Business Delegate, Bridge, Decorator, Flyweight, Facade e Proxy;
- Padrões de comportamentais: Chain of Responsibility, Iterator, State, Command, Mediator, Strategy, Interpreter, Observer e Visitor.

2.1.1.1 Padrões de criação

Existe uma divisão de dois grupos para os padrões de criação, um que lida com a criação de classes e outro que lida com a criação de objetos. Os padrões de criação de classes sugerem a utilização de orientação a objetos, como por exemplo, a herança. Já os padrões de criação de objetos devem criar os objetos delegados a outros objetos (GAMMA; SALGADO, 2008).

Um exemplo do padrão criação é o Singleton, que tem o objetivo de garantir que somente exista uma única instância de determinada classe; outro exemplo é o padrão Factory que encapsula a criação de um conjunto de objetos onde subclasses podem decidir quais classes devem ser instanciadas (GAMMA; SALGADO, 2008).

2.1.1.2 Padrões estruturais

Conforme Gamma e Salgado (2008), padrões estruturais estão relacionados à maneira como se devem estruturar as classes, através de interfaces que são herdadas na implementação definindo-se assim um modelo de estrutura para criação de funcionalidades.

Como exemplo de padrão estrutural, temos o Facade, que tem o objetivo de fornecer uma interface ou mais para um conjunto de subsistemas. Esse padrão tem o objetivo de reduzir o acoplamento de um sistema aos seus subsistemas, desta forma códigos que estão atrás do Facade podem sofrer alterações sem impacto aos sistemas aos quais que estão ligados (GAMMA; SALGADO, 2008).

2.1.1.3 Padrões comportamentais

Conforme Gamma e Salgado (2008), os padrões comportamentais deve sugerir modelos específicos de como devem ocorrer a comunicação entre objetos e classes com a separação do comportamento. Esses padrões ajudam a controlar o fluxo de execução entre as classes tornando a comunicação mais flexível.

O Strategy é um exemplo de padrão comportamental, que permite que o algoritmo de uma classe, independentemente dos clientes que a utilizam, seja implementado conforme o comportamento. Cada comportamento é encapsulado e pode ser intercambiável durante a execução através de chamadas dos métodos (GAMMA; SALGADO, 2008).

2.1.1.4 Padrões de persistência

Conforme Fowler (2003), um padrão de projeto de persistência tem como objetivo criar uma camada responsável pela comunicação entre as demais camadas do software. Esse grupo compreende a maneira pela qual a lógica do domínio irá trocar informações com o bando de dados.

O Repository é um exemplo de padrão de persistência que encapsula a lógica de consultas e manipulação dos dados através de um mapeamento dos dados que são representados através de objetos (FOWLER, 2003).

2.1.1.5 Padrões de controle

Conforme Fowler (2003), esse padrão tem o objetivo de gerenciar o fluxo das informações de um sistema. Um bom exemplo desse padrão é o Unit of Work, que cria uma lista de objetos que precisam ser afetados, coordenando a concorrência durante o fluxo de execução e da persistência. Esse padrão garante que tudo seja executado dentro de uma transação, ou seja, caso ocorra um problema inesperado o padrão deve restaurar o estado do início do processo para que se mantenha a integridade.

2.1.1.6 Padrões de apresentação

Conforme Fowler (2003), esse padrão tem o objetivo de definir a forma e quais informações são apresentadas para o usuário. O padrão deve garantir que a camada de apresentação seja independente do restante da aplicação.

Um bom exemplo é o Model-View-Controller (MVC), que divide a aplicação em três camadas para isolar a interface de apresentação da regra de negócio da aplicação (FOWLER, 2003).

2.1.2 Arquitetura de Software

Uma meta de um projeto de software é possuir um quadro geral da arquitetura do sistema, essa arquitetura permite que o Engenheiro possa visualizar a organização do sistema e tomar decisões de projeto mesmo antes do software ser concebido (PRESSMAN, 2011).

Pressman (2011) cita que existe debate em torno da concepção da arquitetura do software, o modelo conceitual pode vir antes e separado do projeto, ou ser parte integrante do projeto onde a cada mudança do projeto ou requisitos esse modelo deve ser atualizado. Independentemente do momento da concepção da arquitetura, todos os autores concordam que ele traz os seguintes benefícios:

- Melhoria na comunicação: é muito mais fácil todos envolvidos no projeto discutirem sobre arquitetura e de projeto;
- Sucesso do projeto: o modelo conceitual de arquitetura ajuda a visualizar problemas de concepção.

2.1.3 Arquitetura de comunicação para aplicações Web

A tecnologia Web Services teve seu início na década de 90 e foi criada por um grupo de empresas filiadas ao W3C que sentiram a necessidade de integrar aplicações utilizando o meio de comunicação da Internet (GOMES, 2010).

Conforme a definição do World Wide Web Consortium - W3C (W3C, 2017), um serviço Web é um subsistema capaz de proporcionar a comunicação entre duas máquinas diferentes através da rede. Para possibilitar essa comunicação, o serviço utiliza uma interface descrita em um formato específico, a exemplo o WSDL (Web Services Description Language) que é um descritor que permite encapsular as

mensagens através de um protocolo, como por exemplo, o SOAP (Simple Object Access Protocol).

Os Web Services são amplamente utilizados para comunicação na internet, utilizam padrões abertos como XML (Extensible Markup Language) e HTTP (HyperText Transfer Protocol), o que permite que eles possam ser implementados em qualquer linguagem e manter a compatibilidade entre eles (ROVARIS, 2007). Embora a tecnologia Web Services tenha surgido como a principal proposta de integração de sistemas em ambientes heterogêneos (GOMES, 2010), ela pode ser utilizada em redes locais privadas.

2.1.4 Modelo cliente-servidor

Nas últimas décadas muitas aplicações passaram a utilizar este modelo de cliente-servidor, as aplicações Web são um bom exemplo. Uma grande vantagem deste modelo é possibilitar distribuir recursos na rede, compartilhando entre um ou mais servidores e centralizar os recursos para os clientes. Pressman (2011) define as seguintes funcionalidades desta arquitetura:

- Funcionalidades e interfaces são implementadas no lado do cliente;
- Base de dados são responsabilidades e tarefas do servidor;
- Algumas funcionalidades podem rodar no servidor independentemente do cliente;
- Segurança e controle devem ser implementados tanto no cliente como no servidor.

2.1.5 Modelo MVC (*Model View Controller*)

Pressman (2011) sugere que este modelo amplamente utilizado para criação de aplicações. É um modelo que separa a arquitetura em três camadas: Modelo, Visão e Controlador. Cada camada tem responsabilidades específicas:

- *Model*: Camada com a lógica da aplicação que tem acesso a fonte de dados;

- *View*: Camada com a lógica responsável pela interface de apresentação do sistema para o usuário;
- *Controller*: Camada que faz a ligação entre a *View* e a *Model*, controlando o fluxo de dados entre as duas camadas.

De maneira geral, tudo se inicia pela requisição do cliente, essa é enviada para camada de *controller*, a partir dela a requisição é enviada ao *model* para alterar ou consultar dados, esses dados são retornados para camada *controller* que envia os dados para uma *view* executar o processamento para o usuário (PRESSMAN, 2011).

2.1.6 Arquitetura orientada a SOA (Service-Oriented Architecture)

Essa arquitetura é um modelo que disponibiliza um serviço através de um protocolo de comunicação. Esse modelo possui uma interface de comunicação bem definida que fornece um serviço mais complexo para diversas aplicações (ENDREI *et al.*, 2004).

Essa arquitetura pode ser implementada em diversas tecnologias de comunicação, sendo os Web Services a tecnologia predominante e mais difundida nas aplicações Web, os quais utilizam o protocolo HTTP como meio de comunicação (ENDREI *et al.* 2004).

2.1.6.1 SOAP (Simple Object Access Protocol)

O SOAP é um protocolo simples que é utilizado para descrever estruturas para comunicação através de mensagens na Internet. O SOAP obriga que as mensagens sejam baseadas em XML. Embora o SOAP possa encapsular outros protocolos, é muito comum utilizar o WSDL para dizer quais os serviços estão disponíveis, localização e qual a funcionalidade (GOMES, 2010).

2.1.6.2 Web Services Description Language (WSDL)

O WSDL é uma tecnologia com a estrutura de um XML que foi criado pela W3C com a única finalidade de descrever todos os serviços de um Web Service, incluindo os métodos de operações com duas entradas e saídas (GOMES, 2010). Lopes e Ramalho (2014) referem-se ao WSDL como uma interface de acesso às aplicações descrevendo tudo o que o utilizador do serviço precisa saber.

Um WSDL define as interfaces, esquemas, operações, codificação entre outros, em um modelo para que o *Requester* saiba o padrão dos métodos a serem invocados e seus parâmetros. WDSL é a linguagem utilizada nesses documentos para estes fins (MEDYK, 2006).

2.1.6.3 REST (Representational State Transfer)

O REST é uma abstração da arquitetura WWW e constitui em um conjunto de componentes e conectores dentro de um sistema distribuído. A arquitetura REST tipicamente é encontrada em Web Services para disponibilizar uma API de recursos na Web que normalmente é através do emprego de URI (Unified Resource Identifier), HTTP e XML ou JSON (JavaScript Object Notation) (RICHARDSON; RUBY, 2007).

Para ser acessado, o REST deve estar vinculado a um nome e a um servidor onde é hospedado. Por isso é utilizado um nome de recurso, desta forma, um recurso precisa ter ao menos um identificador para que se torne acessível (TOBALDINI, 2008).

As URI's (Unified Resource Identifier) devem ser utilizadas de forma estruturada, intuitiva e uniforme. Um exemplo de prática não recomendada é apresentar nas URI's as informações técnicas de documentação sobre REST ou informações sobre técnicas dos protocolos (RICHARDSON; RUBY, 2007).

A arquitetura REST surgiu como alternativa ao SOAP, ela sobrecarrega menos o servidor e tem menos interações ao consumir um serviço (TOBALDINI,

2008). No ano de 2008, a Google descontinuou o Google SOAP Search API e lançou o AJAX Search API uma interface simples de REST (GOOGLE, 2017).

A utilização de uma API baseada em arquitetura REST é bastante simples, como exemplo, a API da Google onde é necessário apenas fazer uma solicitação GET e processar uma resposta JSON, isso possibilita que qualquer aplicativo que implementa o protocolo HTTP possa consumir esses recursos (GOOGLE, 2017).

2.1.7 SaaS - Software como Serviço

As tendências de Software (Software as a Service - SaaS) se intensificaram com a globalização da Internet. Conforme Sommerville (2012), é normal que os usuários e empresas pensem cada vez mais em migrar parte ou toda sua infraestrutura para nuvem. A geração que está acostumada a utilizar serviços remotos onde estiverem não questionam mais onde está o servidor físico. Ainda Sommerville (2012) reforça que empresas que comercializam aplicações na modalidade SaaS são responsáveis pelas correções de erros, desempenho da aplicação, sendo de forma direta responsável por garantir que a solução SaaS estará disponível ao usuário quando ele necessitar.

Sommerville (2012), sugerem que ao fornecer um serviço como SaaS, a arquitetura deve estar preparada para ser escalável. Sommerville (2012) ainda traz como preocupação da arquitetura atender a múltipla locação e a configurabilidade, como segue:

- **Configurabilidade:** Deve permitir um nível aceitável de configuração para cada cliente a fim de garantir a flexibilidade para atender diferentes perfis de usuário;
- **Multilocação:** Garantir a apresentação do serviço ao usuário como sendo único serviço, e ao mesmo tempo, fazer uso de recursos simultâneos;
- **Escalabilidade:** Projetar um sistema capaz de atender ao número de usuários conforme a demanda necessária.

2.2 Computação em Nuvem

Chee e Franklin (2012) definem a nuvem como uma palavra que esconde toda a complexidade de sistemas, subsistemas e infraestrutura de um sistema. Antes da existência da nuvem, toda empresa precisava se preocupar com todas as questões de TI, desde software até infraestrutura. Hoje em dia, quando uma empresa fala que armazena suas informações em nuvem, quer na verdade dizer que suas informações estão hospedadas e são fornecidas por meio do acesso como um serviço na rede mundial de computadores. De fato, a nuvem surgiu graças a Internet, e a grande nuvem se tornou a grande rede da Internet.

Chee e Franklin (2012) ainda ressaltam que, exceto os casos específicos, onde ainda existam justificativas para manter um servidor local, nos demais casos não existe mais a necessidade de se ter uma infraestrutura própria, que pode ser substituído por serviços em nuvem.

A nuvem é uma alternativa para empresas que desejam utilizar serviços por um tempo programado ou conforme a existência de demanda por tempo indeterminado. A maioria destes serviços é cobrada por uso, um dos argumentos e justificativas que provedores como Amazon e Google usam como vantagem na hora de comercializar seus produtos (CHEE; FRANKLIN, 2012).

2.2.1 Riscos Serviços de armazenamento em nuvem

Existem serviços específicos para armazenamento de objetos em nuvem e que disponibiliza grandes espaços de armazenamento de forma segura pelo valor que você utilizar. Conforme Chee e Franklin (2012), embora os preços sejam atraentes, podem surpreender os usuários mesmo aqueles que calculam o custo antes de contratar o serviço.

Chee e Franklin (2012) comentam que o que eleva o custo dos serviços em nuvem é na largura de banda entre os provedores. É possível simular perfeitamente o espaço em disco que vai utilizar, mas jamais será possível prever o tráfego que os

acessos aos arquivos (frequência e origem de acesso dos usuários) vão gerar (CHEE; FRANKLIN, 2012).

A transferência de dados *in* e *out* referem-se à transferência para dentro e para fora de uma localização da Amazon S3. (Isto é US ou EU). Dados copiados dentro de uma localização da Amazon S3 são gratuitos. Dados transferidos entre Amazon S3 e outras localidades são cobrados taxas regulares conforme a demanda (Amazon, 2017, texto digital).

Um exemplo desta oscilação de preços é o caso da Amazon, aonde os valores chegam a ser 2,5 vezes maior do que os calculados pela ferramenta da Amazon, isso se deve ao fato que a calculadora não leva em conta em seu cálculo todas as variáveis que são cobradas, e essas taxas variam muito conforme o uso e rotas que os usuários estão usando para enviar e baixar arquivos (CHEE; FRANKLIN, 2012).

2.2.2 Riscos dos serviços terceirizados em nuvem

É bom sempre refletir, conforme Chee e Franklin (2012), situações imprevistas podem tirar do ar até mesmo aquele grande provedor de serviços bem preparado, a exemplo, terremotos de Northridge, na Califórnia, em 1994 tirou do ar grande provedores de serviços por danificar praticamente todas as rotas de Internet dos Estados Unidos, a única lição que aprendemos neste dia, é que não se deve colocar todos os serviços ou dados no mesmo fornecedor (CHEE; FRANKLIN, 2012).

Segundo Chee e Franklin (2012), a computação em nuvem gera diferentes opiniões, para alguns, computação em nuvem proporciona muitos benefícios, entretanto esconde muitos detalhes para que efetivamente possam confiar todos os serviços na nuvem, mas isso é uma questão de contrato de prestação de serviço parte do provedor em torno das garantias do serviço. Existem outras coisas ocultas, um exemplo de uma renomada Universidade que desistiu de usar a computação em

nuvem da Google por suspeitar que seus dados estivessem sendo expostos de alguma forma.

Se o provedor do serviço lhe promete 100% de disponibilidade e por problemas desconhecidos resolver permanecer no escuro por 72 horas, independente de quem é a responsabilidade, quem acaba pagando por isso é a contratante, por isso é importante possuir redundância dos serviços, ou na ausência de redundância, um bom contrato exclusivo que garanta os direitos da contratante (CHEE; FRANKLIN, 2012).

2.3 Sistemas de arquivos

Para implementar um sistema de banco de dados em arquivos, alguns conceitos de implementações de sistemas de arquivos atualmente existentes são válidos. Conforme Deitel (1992), os arquivos são estruturas de dados distintos e organizados, de forma que estes possam ser utilizados pelos processos. Da mesma forma como os sistemas operacionais controlam as operações sobre esses arquivos, organizando-os, e cumprindo papéis definido por Deitel (1992) como:

- Métodos de acesso: formato como são armazenados e recuperados nos sistemas;
- Gerenciamento de arquivos: conjunto de técnicas e mecanismos de referência, acesso compartilhado e segurança;
- Integridade: Técnicas que asseguram que os dados armazenados são íntegros.

Ainda Deitel (1992) cita que um sistema de arquivos deve suportar alguns requisitos como:

- Garantir que os arquivos possam ser criados, modificados, eliminados, recuperados;
- Possibilitar o acesso simultâneo ao mesmo conjunto de arquivos;
- Suportar operações de backups;
- Permitir a adoção de técnicas e conceitos de segurança e integridade;

- Oferecer, no que diz respeito ao acesso aos arquivos, uma forma transparente por meio de nomes simbólicos, o que garante uma independência do tipo de armazenamento implementado.

Um sistema de arquivos nem sempre precisa entender o que é armazenado, conforme Tanenbaum (1995), um sistema de arquivos pode armazenar um conjunto de *bytes* em sequência não interpretados pelo sistema que o implementa e depende diretamente de outra aplicação que faz a utilização desse sistema de arquivo para o correto entendimento do conteúdo.

2.3.1 Banco de dados e SGBD

Conforme define Silberschatz, Korth e Sudarshan (2012), qualquer estrutura de armazenamento que tenha uma coleção de dados que tenha algum tipo de relacionamento com outras partes e que trate de mesmo domínio pode ser chamado de um banco de dados.

O banco de dados tem um objetivo principal de armazenar e permitir que usuários busquem e atualizem informações quando necessário. Essas informações são dados que tenham algum significado (DATE, 2004).

Segundo Ramakrishnan e Gehrke (2008), o Sistema de Gerenciamento de Banco de Dados ou repositório é um software que auxilia na manipulação e organização de um conjunto de dados, esse deve ser capaz de cumprir funções mínimas necessárias para manipular esses dados, tais como: armazenar, recuperar e excluir. Tipicamente um SGBD é um programa que gerencia a estrutura do banco de dados e controla os acessos aos dados armazenados (ROB; CORONEL, 2011).

Ainda Rob e Coronel (2011) sugere que SGDB serve como uma interface de comunicação entre o banco de dados e usuário. Em sua estrutura interna, os dados são armazenados em um formato organizado e a única forma de acessá-los é por meio do SGDB.

2.3.2 Catálogo de metadados de um banco de dados

Segundo Elmasri e Navathe (2011) o Sistema Gerenciador de Banco de Dados (SGBD) é um sistema projetado para atender diversas aplicações de banco de dados, e sempre que for necessário conhecer a estrutura de um banco de dados, o SGBD deve recorrer ao catálogo, onde se encontra os metadados do esquema, tais como informações de controle interno e dos dados armazenados.

O catálogo possui todas as informações sobre a estrutura do banco de dados, informações das estatísticas utilizadas pelo SGBD para melhorar o plano de execução das consultas, ainda as estatísticas podem ser usadas pelo administrador de banco de dados para identificar problemas de desempenho, tais como índices não utilizados e a ocorrência de *deadlocks*² (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

Uma consulta convencional retorna informações existentes em tabelas, já uma consulta no catálogo retorna informações sobre os bancos, os objetos dos bancos, os campos de tabelas, seus tipos de dados, seus atributos, dentre outros. Uma consulta no catálogo do PostgreSQL, por exemplo, retorna informações como o nome e o esquema de dados de todas as tabelas de um banco de dados (POSTGRESQL, 2017).

2.3.3 Transações

Uma transação é definida por Ramakrishnan e Gehrke (2008) como uma lista de ações que devem ser executadas. Cada transação deve especificar ao final do processo uma marcação que indica sucesso ou insucesso. Em caso de falha (insucesso), o SGBD dispara o procedimento de desfazer as alterações até o momento. As transações apenas interagem com operações de gravação ou leitura do banco de dados, elas não podem trocar mensagens entre si (RAMAKRISHNAN; GEHRKE, 2008).

² *Deadlocks* são interbloqueio, blocagem ou impasse, no contexto de sistemas operacionais, refere-se a uma situação em que ocorre um impasse, e dois ou mais processos ficam impedidos de continuar suas execuções.

Siberschartz, Korth e Sudarshan (2012) define que SGDB deve ter subconjuntos de operações internas que são executadas pelo usuário como uma única operação. Por exemplo, a transferência de valores uma conta bancária para uma conta poupança é uma operação única sob o ponto de vista do usuário, mas que dentro do SGDB envolve várias operações e controles.

Uma transação altera vários dados e é o resultado de um fluxo de execução de um software. Cada transação tem marcadores que indicam seu início e fim, e dentro desse bloco varias outras operações serão executadas como sendo um único bloco (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

Conforme Siberschartz, Korth e Sudarshan (2012) para garantir a integridade dos dados, um banco de dados tem as seguintes propriedades bem definidas.

- a) Atomicidade: Ou todas as operações da transação são executadas com sucesso no banco de dados ou nenhuma será executada;
- b) Consistência: A execução de uma transação isolada, ou seja, se há execução concorrente de outra transação, preserva a consistência do banco de dados;
- c) Isolamento: Sabendo que diversas transações podem ser executadas de forma concorrente o sistema deve garantir que cada transação não toma conhecimento de outras transações concorrentes no sistema e todas as transações são executadas em ordem;
- d) Durabilidade: Depois da transação finalizada com sucesso, as mudanças que ela fez no banco de dados persistem até mesmo se houver falhas no sistema.

2.4 Modelo de referência de segurança

Para entender o que é segurança, devemos entender os conceitos envolvidos na computação que os sistemas implementam, Moraes (2012) e Commer (2007) definem alguns dos principais conceitos.

2.4.1 Integridade

A integridade de qualquer sistema consiste em garantir que a informação recebida ou enviada é integrada em ambas as pontas. A mensagem deve chegar ao destino sem nenhum tipo de modificação. Conforme Moraes (2012) existe um modelo de segurança que define uma arquitetura confiável que programa uma série de regras, procedimentos, autorizações e negações que garantam a confiabilidade do sistema. Para entender o que é segurança, devemos entender os conceitos envolvidos na computação que os sistemas implementam. Moraes (2012) define sete dos principais conceitos a baixo.

2.4.2 Hashing

O *hashing* é uma técnica amplamente utilizada para verificação e a garantia de integridade de dados. O resultado do *hashing* é o valor calculado matematicamente sobre a combinação dos dados, ou seja, a mensagem após percorrer um caminho sofre diversas interferências do meio de comunicação, e pode ser checada no destino final calculando-se o *hashing* novamente e conferindo se este é mesmo da mensagem original (MORAES, 2012).

Uma chave *hash* deve ser computada de tamanho fixo e sem colisão. Ou seja, a função *hash* deve garantir que estas chaves não vão colidir. Uma função é amplamente difundida nas linguagens de programação é o MD5, que foi criado em 1991 por Rivers da RSA em 1991.

2.4.3 Confiabilidade

Moraes (2012) comenta que *Hackers* podem tentar descobrir senhas ou acessar sistemas e recursos por portas de acessos que foram negligenciadas pela equipe e principalmente pela arquitetura, tornando pública uma informação sigilosa e destruindo a reputação de um sistema. Ainda Moraes (2012) reforça que a confiabilidade está relacionada à autorização de acesso aos recursos e processos. Um sistema deve garantir que somente a pessoa ou processo autorizado terá

acesso, logo isso se refere à privacidade dos dados. Moraes (2012) descreve aspectos importantes da confiabilidade são a identificação, a autorização e a autenticação, sendo eles definidor da seguinte forma:

- **Identificação:** A identificação é um processo que antecede a autenticação, podemos dizer que programa ou processo deve disponibilizar sua identidade para que se conecte e permita que uma autenticação seja realizada;
- **Autenticação:** Após prover a identificação do sistema de origem, os recursos e serviços podem ser acessados mediante a autenticação do usuário, este deve fornecer uma chave ou combinação de usuário e senha para acesso autorizado;
- **Autorização:** O último processo que ocorre após a autenticação é a verificação dos privilégios que o usuário vai ter. Um usuário autenticado deve possuir acesso somente aos recursos vinculados a sua autenticação;
- **Autorização:** O último processo que ocorre após a autenticação é a verificação dos privilégios que o usuário vai ter. Um usuário autenticado deve possuir acesso somente aos recursos vinculados a sua autenticação.

2.4.4 Disponibilidade

A disponibilidade é a capacidade de garantir que um sistema está sempre acessível. As ameaças mais comuns estão relacionadas não propriamente a inovação, mas sim em derrubar um sistema de forma proposital, um ataque conhecido é o DoS (*Denial of Service*), onde o sistema é bombardeado por tentativa de negação de serviço até que servidores e conexões sobrecarregam e se tornem indisponível. Deve existir um controle sobre essas requisições de negação, bloqueando o ataque antes que o sistema se torne indisponível (MORAES, 2012).

2.4.5 Auditoria

É um importante mecanismo, com a criação de um serviço de *Log* (arquivo de registro de atividades), é possível se fazer um controle preventivo e corretivo nas ações enquanto elas ocorrem. Os Logs servem também para verificação de anormalidades e irregularidades dos sistemas (MORAES, 2012).

O *Log* deve garantir a verificação do que foi feito, por quem, quando e o que foi afetado e em aplicações críticas deve ser detalhado o suficiente para ajustar e restabelecer um sistema danificado ou comprometido (MORAES, 2012).

2.4.6 Criptografia

A criptografia é muito anterior aos computadores, é uma ciência matemática que existe há séculos na humanidade. A criptografia consiste em tornar um código ilegível ou torná-lo legível somente pela pessoa que possua chave para fazer essa engenharia (MORAES, 2012).

Conforme Moraes (2012), algoritmos de criptografia são eficientes do ponto de vista computacional, ou seja, deve ser computacionalmente fácil de criptografar e descriptografar com o conhecimento da chave, e computacionalmente falando, difícil de descriptografar sem a chave.

O sistema de chave secreta é o mais seguro, baseia-se nos conceitos de que apenas o transmissor e receptor possua a chave compartilhada, logo a segurança está na chave, uma vez o conteúdo criptografado, somente o conhecedor da chave saberá descriptografar o conteúdo (MORAES, 2012).

Ainda Moraes (2012) cita que o armazenamento da chave é algo que deve ocorrer, mas que essas não devem estar acessíveis a qualquer pessoa, deve estar encapsulado e serem exibidas, se necessário, apenas quando precisar recuperar alguma informação.

A força da chave de criptografia está relacionada com o seu tamanho, em resumo, quanto maior o tamanho de caracteres diferentes e aleatórios, maior o tempo para descobrir essa chave (MORAES, 2012).

2.4.6.1 Algoritmo AES

Esta seção tem o objetivo de explicar a principal criptografia que manterá os dados armazenados de forma segura. Embora possamos armazenar conteúdos criptografados sobre diferentes algoritmos e técnicas, o Algoritmo AES é um padrão de criptografia definido e utilizado pelo governo dos Estados Unidos para proteger seus documentos secretos e confidenciais, logo se entende como seguro, robusto e confiável.

O governo utiliza um bloco de 256 bits para garantir a força da chave AES, conhecida como AES-256 (INTEL, 2012). Ainda conforme a Intel, 2012, a criptografia AES pode ser implementada utilizando o OpenSSL que é uma interface mundialmente conhecida por gerar chaves de criptografia para servidores em diferentes plataformas, incluindo de serviços da Intel (INTEL, 2012).

Diversos equipamentos baseiam sua segurança em AES, a diferença é que equipamentos de menor capacidade de processamento implementam variações do AES com blocos de 64 bits ou 128 bits, com é o caso de roteadores sem fio, citado por Moraes (2012), que implementam autenticação reforçada com AES.

2.4.7 Intrusão ou acesso não autorizados

A intrusão ou acesso não autorizado a um sistema pode ser feito por pessoa legítima ou não. Essa tentativa normalmente ocorre por pessoas de fora da organização, mas também pode ser por pessoas de dentro da organização. Conforme Moraes (2012) existem softwares capazes de detectar esses ataques e bloquear a origem de forma preventiva.

Normalmente são várias técnicas e ferramentas para conseguir invadir o sistema alvo. O invasor primeiramente busca utilizar informações sobre o sistema

vítima utilizando serviços específicos para essa finalidade, tais como: *Whois*, *Nslookup*, *Scanning* de portas e IPs, identificação de serviços, ataques de buffer, injeção de conteúdo, injeção de softwares maliciosos (MORAES, 2012).

Quando estamos falando em aplicações em nuvem, a segurança é mais delicada, estamos lidando com várias camadas, desde sistemas operacionais, aplicativos dos servidores e também da própria aplicação. Moraes (2012) comenta que os atacantes buscam por vulnerabilidades na própria aplicação do servidor, como é o caso do Apache, ISS da Microsoft e etc. Logo estes devem estar sempre atualizados. Mas os atacantes também podem explorar vulnerabilidades na camada de sistema operacional, explorando vulnerabilidades da camada de rede, protocolos e serviços. Por último, um ataque muito comum é o *SQL Injection*, que busca explorar as falhas na camada da aplicação, em entradas não tratadas.

2.4.7.1 Detecção e prevenção de intrusão

Conforme Moraes (2012) existem três técnicas que devem ser observadas em diferentes camadas da solução, são elas:

- Prevenção: roda na camada de infraestrutura e é responsável por gerenciar IPS, *firewalls*, encriptação, autorização;
- Detecção: normalmente na camada de sistema operacional, responsável por auditoria, registro de atividades, detecção de vírus e *scripts* maliciosos;
- Reação: a camada dos aplicativos, políticas de segurança, procedimentos e resposta automática aos ataques.

Os sistemas de detecção e prevenção de intrusão são importantes quando precisamos deixar portas abertas para o mundo externo, como no caso da porta de acesso às páginas de sistemas HTTP. Na camada de aplicação também podemos implementar mecanismos dinâmicos de controle, prevenção e reação às falhas de acesso aos sistemas, através de bloqueios do atacante. Conforme Moraes (2012), um sistema de detecção e prevenção de intrusão deve ser capaz de identificar padrões maliciosos e atividades anormais do uso. Esse sistema, caso detectar anomalias, deve acionar um perito para intervenção manual.

3 METODOLOGIA

De acordo com Gil (1999, p. 42) a pesquisa é um “processo formal e sistemático de desenvolvimento do método científico. O objetivo fundamental da pesquisa é descobrir respostas para os problemas utilizando procedimentos científicos”.

Uma pesquisa exploratória aumenta o conhecimento do pesquisador sobre um determinado assunto. Ela pode ser composta, entre outros, por entrevistas, revisões de literaturas (GIL, 1999).

O presente trabalho apresenta a construção de forma experimental de um sistema capaz de armazenar arquivos em nuvem privada. Para isso existe um *engine*³ que é responsável por estruturar o armazenamento em disco objetos de forma binária e uma API de acesso para os sistemas realizarem operações neste repositório de arquivos ou em repositórios de serviços de terceiros, que neste trabalho é chamado apenas de GLFS. É requisito que o GLFS tenha flexibilidade para adaptarem-se facilmente a outros serviços de armazenamento, como exemplo, serviços de arquivamento de terceiros.

3 *Engine* é um conjunto de funções prontas para abstrair a complexidade e que facilitam na implementação de novos módulos.

Foram realizados estudos e testes dos conceitos relacionados no referencial teórico com a finalidade de desenvolver uma ferramenta em conformidade com os conceitos existentes.

3.1 Ferramentas

Nesta seção serão abordadas as principais ferramentas utilizadas na construção da ferramenta proposta.

3.1.1 PostgreSQL

O PostgreSQL é considerado um SGBD objeto-relacional e é compatível com diferentes sistemas operacionais, tais como: Windows, Linux, MacOS, Solaris. Além disso, tem suporte a diferentes plataformas e linguagens de programação, por exemplo: Java, C, Python, PHP e Ruby (MILANI, 2008).

De acordo com Milani (2008), o PostgreSQL possui os principais recursos existentes nos sistemas gerenciadores de banco de dados pagos do mercado e com capacidade para suprir desde pequenas a grandes aplicações.

3.1.2 Características do PostgreSQL

A seguir, Milani (2008) cita algumas características existentes no PostgreSQL.

- a) Suporte a transações: O PostgreSQL possui suporte a operações ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Cada uma destas propriedades garante a qualidade dos serviços disponibilizados pelo SGBD;
- b) Alta disponibilidade: Visando expansão da capacidade de processamento o PostgreSQL possibilita que atue como um *cluster* de informações;
- c) *Multithreads*: O PostgreSQL possibilita mais de uma conexão com o banco de dados, por meio de recurso de *Multithreads*, ou seja, permite que mais de

uma pessoa possa acessar a mesma informação simultaneamente sem ocasionar atrasos ou filas de acesso;

- d) Segurança e criptografia: O PostgreSQL possui suporte nativo a SSL, possibilitando conexões seguras através destes canais para trafegar informações consideradas sigilosas;
- e) SQL: Adota os padrões ANSI SQL na implementação de suas funcionalidades;
- f) Incorporável em aplicações gratuitamente: Por utilizar a licença BSD (Berkeley Software Distribution), pode ser livremente incorporado em aplicações pessoais e/ou comerciais;
- g) Capacidade de armazenamento: O PostgreSQL suporta de maneira eficiente e confiável grandes tamanhos de informações não tendo limite máximo para um banco de dados.

3.1.3 Limitações do PostgreSQL

Segundo Milani (2008) o PostgreSQL é flexível no que diz respeito ao processador e a plataforma. O SGBD pode ser instalado de forma estável em qualquer tipo de hardware com diversos tipos de sistemas operacionais entre eles Linux, UNIX (AIX, BSD, HP-UX, SGI ARIX, MAC OS X, Solaris, Tru64), e Windows. O Quadro 1 apresenta algumas características.

Quadro 1: Características do PostgreSQL

Limite	Valor
Tamanho máximo do banco de dados	Ilimitado
Tamanho máximo de uma tabela	32 TB
Tamanho máximo de uma linha	1.6 TB
Tamanho máximo de uma coluna	1 GB
Quantidade de linhas por tabela	Ilimitado
Quantidade de colunas por tabela	250 à 1600
Quantidade de index por tabela	Ilimitado

Fonte: POSTGRES (2017).

3.2 Linguagem de programação - PHP (Hypertext Preprocessor)

Conforme Dall'Oglio (2015), PHP é uma Linguagem de programação que significa Personal Home Page Tools e nasceu em 1994. Rasmus Lerdorf, o criador do PHP precisava de uma solução para monitorar o acesso ao seu currículo na Internet. Em pouco tempo muitas pessoas passaram a utilizar o PHP e em 1995 já contava com recursos de banco de dados. Em 1997 1% da Internet já utilizava a linguagem PHP. Conforme PHP Group (2017), duas décadas depois, o PHP conquistou vários adeptos tornando-se a linguagem mais utilizada do mundo para desenvolvimento Web.

O PHP é a principal linguagem de programação utilizada na implementação, atualmente uma linguagem moderna e implementa todos os conceitos de orientação a objetos. A orientação a objetos é um paradigma de programação que serve para representar o mundo real, que em programação é chamado de entidade (DALL'OGGIO, 2015).

3.3 Tecnologias utilizadas como apoio

Para o desenvolvimento da solução proposta, foram utilizadas diversas outras tecnologias e ferramentas de apoio, são elas:

- a) Apache HTTP(S) Server: Software de código aberto que disponibiliza páginas à Web. O Apache⁴ é utilizado como servidor Web para os usuários terem acesso ao sistema;
- b) HTML: (HyperText Markup Language) é uma linguagem da Internet e é utilizada para desenvolver páginas na Web. O HTML⁵ será utilizado para criar a interface da plataforma;
- c) JavaScript: É uma linguagem de programação do lado cliente processada pelo próprio navegador. O JavaScript⁶ será utilizado para

4 <http://httpd.apache.org/>

5 <http://www.w3.org/html/>

6 <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

criar efeitos nas interfaces da ferramenta para proporcionar uma maior interatividade;

- d) Bootstrap: É um *framework* que facilitar a criação sistemas Web. O Bootstrap⁷ é um conjunto de estilos e será utilizado para deixar a ferramenta mais intuitiva em páginas de apresentação ao usuário;
- e) JQuery: É uma biblioteca JavaScript que interage com o HTML desenvolvida para simplificar os scripts no navegador do cliente. O JQuery⁸ será utilizado para aumentar a compatibilidade entre navegadores e plataformas, redução e desempenho de código.

3.4 Estudo de caso do Software ERP GreenApp

GreenApp é um sistema de gestão empresarial Web, comercializado na modalidade SaaS. Diversos módulos do GreenApp possuem telas com opções de anexar arquivos, estes arquivos eram armazenados na camada de aplicação e somavam 660 Gigabytes fragmentados entre todos os clientes, isso incluía arquivos anexos pelos usuários e cópias de backup com 60 dias retroativos. A ferramenta GLFS foi implementada e validada no ERP GreenApp para melhorar a gestão destes arquivos anexos, melhorando a arquitetura do sistema e permitindo uma maior escalabilidade em nuvem.

3.5 Trabalhos relacionados

Graham e Lio (2014) demonstram a facilidade que os sistemas podem se conectar aos provedores de serviços em Nuvem através de uma API, no entanto, ao mesmo tempo em que essa API ajuda no conceito de elasticidade associado à Nuvem, há uma tendência de que essas APIs sejam específicas para cada provedor tornando-as limitadas por falta de padrões entre os provedores. Um método para

7 <http://getbootstrap.com/>

8 <https://pt.wikipedia.org/wiki/JQuery>

superar essa falta de padrões são as APIs abertas ou genéricas. Em seu artigo, Graham e Lio (2014) propõe uma avaliação das APIs Cloudify e jClouds sobre 8 critérios e 5 sub-critérios. Uma constatação durante os testes é que as APIs Cloudify e jClouds não adicionam nenhum fator adicional de segurança, ou seja, ela apenas limitam-se em fazer uma ponte entre os subsistemas e nuvem, mesmo assim concluem que APIs genéricas são mais positivas do que negativas.

Artigas (2017) sugere que os serviços de armazenamento do Personal Cloud (PC), como o Dropbox ou o Google Drive, tornaram-se cada vez mais populares nos últimos anos. Infelizmente, esses serviços ainda não são seguros mesmo assumindo a confidencialidade de dados. O compartilhamento seguro de um conteúdo nesses serviços ainda é um problema, pois se baseia em atributos de usuário arbitrários e atributos do arquivo. O trabalho é uma implementação de um software livre que oferecer suporte ao compartilhamento de dados privados refinados por meio de criptografia baseada em atributos antes de enviar ao servidor remoto. O resultado foi o primeiro projeto e implementação de um serviço de sincronização de arquivos, chamado StackSync, em que o usuário retém o controle total sobre seus próprios dados ao compartilhá-lo. Nunca antes, a criptografia baseada em atributos havia sido implementada e testada em um serviço de PC real. Os resultados mostram que o StackSync é seguro e eficiente.

Graupner *et al.* (2015), propõe uma interface para acesso aos provedores de armazenamentos em nuvem existentes e uma nova técnica de lidar com os desafios de conectar padrões de autenticação. Foi destacado que os provedores de nuvem se comportam de maneira muito diferente no que diz respeito ao processamento de arquivos e com diferentes padrões de comunicação, segurança e desempenho. Portanto, os sistemas de software que se propõe a suportar diferentes provedores de serviços devem ser capazes de alternar dinamicamente entre serviços de forma eficaz.

Andrade (2014) em seu trabalho faz uma avaliação da ferramenta de armazenamentos de arquivos OwnCloud capaz de gerenciar um repositórios local e se integrar a outros serviços em nuvem. Andrade (2014) faz um estudo aprofundado sobre as tecnologias de *cloud computing*, analisando seus aspectos e funcionalidades para aplicação da ferramenta. Os resultados mostram que a

ferramenta tem um grande potencial a ser explorado e que é uma ótima alternativa para armazenamento em nuvem, por permite ser customizada.

Quadro 2: Comparação entre GLFS e soluções similares.

		OwnCloud	jClouds	StackSync	GLFS
1	Melhora a segurança dos dados	sim	não	sim	sim
2	Permite criptografia de dados	sim	não	sim	sim
3	Permite integrar com nuvem de terceiros	sim	sim	sim	sim
4	Possui repositório e gerenciamento próprio	sim	não	não	sim
5	Permite redundância de dados internamente	sim	sim	não	sim
6	Possui relatórios detalhados de uso	não	não	não	sim
7	Permite customização	sim	sim	sim	sim
8	Permite download de arquivos por URL	sim	não	não	sim
9	Permite controles de download dos arquivos	não	não	não	sim
10	Permite configuração de diferentes links	sim	não	não	sim
11	Permite diferentes criptografias	sim	sim	sim	sim
12	Permite diferentes verificações de vírus	não	não	não	sim
14	Possui controle interno de versionamento	sim	não	não	não

Fonte: Elaborado pelo autor (2018).

Entre as soluções dos trabalhos relacionados, o Quadro 2 mostra um comparativo entre características do GLFS e três soluções similares. Essas características foram selecionadas com base nas necessidades do GreenApp.

Ainda o Quadro 2, demonstra que há pequenas diferenças entre as soluções e quer demonstrar que o GLFS, além de atender a necessidades específicas do GreenApp, aproveitou e implementou os características, conceitos e funcionalidades das outras soluções, com o intuito de ser mais completa e robusta.

4 IMPLEMENTAÇÃO DO GLFS

Neste capítulo são apresentados os detalhes da implementação e validação da solução para armazenamento de arquivos em nuvem privada batizada como GLFS.

4.1 Cenário do ERP GreenApp antes do GLFS

É muito comum possuir em diversas partes do sistema, opções para anexar arquivos de diferentes tipos e tamanhos. Quando o sistema é instalado no cliente, normalmente esse arquivamento é local e o problema de espaço, integridade e *backup* são de responsabilidade do cliente, mas quando a solução é comercializada como SaaS em nuvem, essa preocupação passa a ser de responsabilidade do fornecedor. No início do projeto de ERP GreenApp foi decidido que os arquivos anexados pelos usuários ficariam armazenados em um diretório dentro da aplicação, fazendo parte da estrutura de diretórios aplicação Web, e a justificativa para essa decisão era simples:

- 1) Os arquivos anexos seriam pouco utilizados;
- 2) Os serviços em nuvem para arquivamento eram caros e limitados;
- 3) O número de clientes usuários era muito baixo;
- 4) Entendia-se os anexo como sendo arquivos de tamanho pequenos;

5) A função de anexar arquivos dentro do GreenApp seria pouco utilizado.

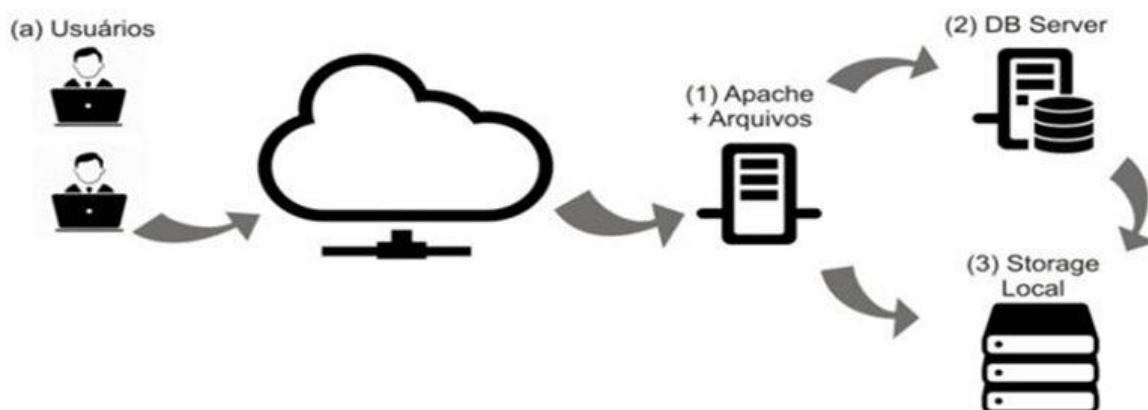
Em pouco tempo, cerca de três anos, essa decisão mostrou-se ser um problema típico de escalabilidade, pois diversos módulos do sistema foram atualizados e passou a ter a opção de anexar arquivos, o número de clientes aumentou e muitos deles passaram a adotar a prática de anexar os arquivos dentro do GreenApp como tarefa comum do dia a dia devido a organização e facilidade em se localizar um arquivo. Nesse novo cenário, é possível enumerar em ordem os principais problemas que o cenário antes do GLFS apresentava:

- Escalabilidade vertical: A estrutura da aplicação Web (arquivos de clientes mais arquivos da aplicação) rapidamente se torna grandes, impedindo que se criem instâncias virtuais para distribuir a carga entre servidores, pois cada instância levaria consigo uma cópia dos arquivos da aplicação;
- Escalabilidade horizontal: A estrutura inchada de arquivos dentro da camada da aplicação limitou o aumento de performance do servidor, pois tornou-se difícil encontrar um hardware que atenda às características de alto desempenho para servidor Web e de alta capacidade de armazenamento para arquivamento;
- Backup: A estrutura de *backup* se tornou tão grande quanto à estrutura do servidor Web e difícil de gerenciar;
- Gargalo nos *links*: O uso dos mesmos *links* para aplicação Web, *download* e *upload* de arquivos, quando simultâneos acarretam em lentidão do sistema.

A Figura 1, representa o cenário antes do GLFS que era composto por 3 servidores. Esta arquitetura apresentava um problema de escalabilidade, pois um único servidor (1) era responsável por armazenar os arquivos da aplicação e os arquivos anexados ao sistema, isso dificultava a virtualização de várias instâncias do servidor Web uma vez que carregava todos os arquivos em anexo dos usuários em cada máquina virtual que fosse criada. Também este formato impedia o crescimento em escala dos recursos necessários para armazenar grandes volumes de arquivos, o espaço ficava limitado ao tamanho dos discos do hardware que não estava preparado para crescer como os equipamentos de armazenamento (*storage*). Outro

problema era que todas as requisições de *download* e *upload* utilizavam o mesmo *link* de acesso do ERP, e gerava gargalos na aplicação quando usuários simultâneos estavam gerando muito tráfego de arquivos.

Figura 1: Cenário antes do GLFS



Fonte: Elaborado pelo autor (2018).

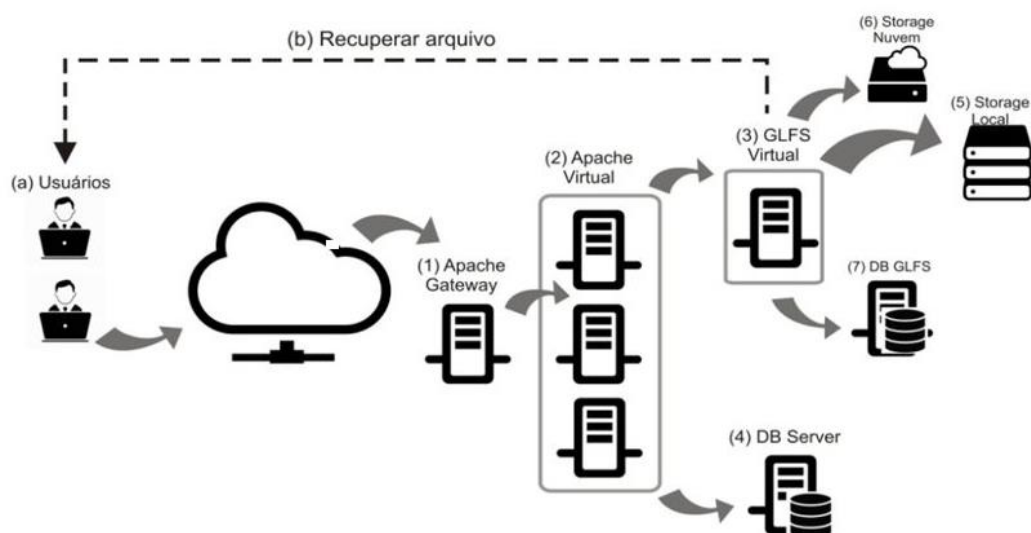
4.2 Cenário do ERP GreenApp com a implantação do GLFS

Na Figura 2 vemos o cenário após a implantação do GLFS, que permite uma arquitetura mais flexível no requisito escalabilidade. Nela a estrutura de arquivos fica separada da camada da aplicação, e passa a ser gerenciado pelo servidor ou instância virtual do GLFS, desta forma o servidor da aplicação Web (2) pode ser escalado, além de verticalmente, horizontalmente com várias instâncias do servidor Web (replicadas).

Para resolver o problema de escalabilidade de recursos para arquivamento de larga escala (crescimento vertical e/ou horizontal), o GLFS pode ser configurado para usar duas estratégias; a primeira é usar o *storage* local de alta capacidade, onde o GLFS criar uma estrutura própria de arquivamento nos seus próprios discos físicos; a segunda estratégia é utilizar um serviço de terceiros de arquivamento em nuvem. Para executar essa solução, basta carregar uma instância da máquina virtual em qualquer *host*. Ele é auto suficiente, pois carrega consigo seu próprio banco de dados para gerenciamento interno, bem como um servidor Web e demais dependências necessárias para seu funcionamento; além disso, ele precisa ter

conectividade com os repositórios que vai utilizar e que as portas HTTPS esteja aberta e encaminhada para o servidor do GLFS.

Figura 2: Cenário depois da implantação do GLFS.



Fonte: Elaborado pelo autor (2018).

A implantação do GLFS também resolveu de uma forma simples o problema de gargalos em *links*, pois utiliza um canal de comunicação destinado exclusivamente para esse serviço sem prejudicar os canais de comunicação do ERP na recuperação dos arquivos.

4.3 Levantamento de requisitos

Para que a ferramenta proposta atingisse os objetivos desejados foram mapeados alguns requisitos. Os requisitos são divididos em funcionais e não funcionais e podem ser visualizados nas seções seguintes.

4.3.1 Requisitos funcionais

- RF-001: Possuir uma API para utilização do serviço com os seguintes recursos: criar e excluir depósito, criar e excluir galeria, enviar arquivos, recuperar arquivos, excluir arquivos, restaurar arquivos, verificar

integridade dos arquivos, configurar a API e retornar informações de um repositório;

- RF-002: Armazenar metadados para o *back-end* gerar relatórios do tipo: depósitos mais utilizados, tráfego de um depósito, erros, logs de auditoria, informações gerais de um depósito;
- RF-004: Implementar uma classe para gerenciar um repositório local com garantia de integridade e atomicidade dos arquivos armazenados localmente; o repositório local deve seguir os conceitos *ACID*, logo tratado como um banco de dados próprio;
- RF-005: Implementar uma classe de verificação de vírus com antivírus local e em nuvem; para opção de nuvem pode-se optar por uma solução que tenha no mercado que disponibiliza uma API, como exemplo da Virustotal (REF); para os testes pode-se utilizar um arquivo texto com uma string (EICAR - The Anti-Virus test file) que é detectado como infectado;
- RF-006: Implementar classes para criptografia dos arquivos com algoritmos AES 256 e base64. Essas classes são utilizadas para criptografar e descriptografar os arquivos; essa implementação vai garantir o sigilo dos dados quando forem armazenados por serviços de terceiros;
- RF-007: Permitir configurar diferentes *links* para o *download* dos arquivos; esses *links* devem ser utilizados para criar o *link* de acesso aos arquivos;
- RF-008: O GLFS deverá ser flexível para funcionar com vários repositórios, ou várias classes de criptografia; para cada arquivo armazenado, devem ser armazenadas todas as informações necessárias para a recuperação destes arquivos, menos a chave de descriptografia;
- RF-009: Os *links* de acesso aos arquivos deve ser um *link* estático e descartável, seja por tempo configurável ou por *download* único; o sistema deve garantir que o acesso ao arquivo seja unicamente pelo *link*, uma vez ele destruído, o arquivo não tem acesso direto de nenhuma outra forma;

- RF-010: Possuir um *back-end* para Web que permita gerenciar o GLFS;
- RF-011: Implementar uma lixeira local, onde os arquivos excluídos são mantidos temporariamente; isso vai ser utilizado para dois casos, recuperação de arquivos deletados que não tenham finalizado o processo completo, e como lixeira para arquivos excluídos com sucesso; logo, aplicações que trabalham com o conceito de exclusão por software (*soft delete*⁹), os arquivos podem ser recuperados.

4.3.2 Requisitos não funcionais

- RNF-001: Utilizar banco de dados PostgreSQL encapsulado para armazenamento dos *metadados*;
- RNF-002: Ser implementado utilizando a linguagem PHP orientado a objetos;
- RNF-003: Utilizar o padrão *singleton* para a API, e/ou priorizar classes estáticas;
- RNF-004: O GLFS não deve armazenar a chave de descryptografia, uma vez sendo gerada e entregue ao cliente, ele é o responsável por cuidar e garantir a existência dela quando for necessário recuperar um arquivo;
- RNF-005: Para o repositório local, o servidor de arquivos deve ser um *storage*, hardware com discos destinado para essa finalidade;
- RNF-006: Toda comunicação, desde o usuário final até ERP GreenApp, e entre o GreenApp e o GLFS, deve ser somente por um túnel criptografado por SSL, conhecido na Web por HTTPS;
- RNF-007: A arquitetura deve permitir a implementação de classes para utilizar novos repositórios, classes para no suporte a novos algoritmos de criptografia e verificação de vírus;
- RNF-007: Existir um *log* de erros interno de todos os métodos; isso vai servir para validação do projeto e para resolver futuros problema.

9 *Soft-delete*: Técnica adotada para facilitar a recuperação de dados que consiste em apenas fazer uma marcação no item excluído para uma futura exclusão definitiva programada;

4.3.3 Limitações do escopo do projeto

A solução proposta limita-se em melhorar a arquitetura de software do ERP GreenApp que impede que ele seja escalável, ou seja, alguns itens não fazem parte do projeto proposto pois já existe um cuidado especial pela empresa mantedora do ERP GreenApp nos seguintes itens:

- 1) Infraestrutura, desempenho e dimensionamento de rede e servidores;
- 2) Backup de servidores, máquinas virtuais;
- 3) Redundância de *links* e serviços;
- 4) Gerenciamento do projeto (versionamento de código, documentação, orçamento e cronograma).

4.4 Desenvolvimento

Foi desenvolvida uma solução que permite o gerenciamento de arquivos fora da camada de aplicação do ERP GreenApp. A solução tem o objetivo de receber requisições de armazenamento e recuperação de arquivos, e de forma segura, íntegra e privada fazer o armazenamento tanto repositório local ou na nuvem de terceiros.

A solução GLFS é composta por um conjunto de tecnologias, sendo elas: uma API contendo toda a lógica para manipulação das solicitações e recuperação das informações; uma *engine*, onde se encontram os algoritmos de criptografia; escaneamento anti-vírus; verificação de integridade de dados (via MD5), dentre elementos que serão abordados nessa seção.

4.4.1 Gerenciamento dos dados

O GLFS efetua o controle do armazenamento dos arquivos físicos por meio de transações, seguindo os princípios de gerenciamento dos Sistemas de Bancos de Dados. Cada operação realizada sobre os arquivos, como: armazenamento, alteração ou exclusão, segue a estratégia de realizar a operação por completa,

garantindo a integridade dos dados, ou seja, caso tudo ocorra bem, a operação é finalizada com sucesso, caso contrário, cabe ao sistema desfazer a sequência de tarefas (rollback) e levar o sistema para um estado seguro e estável novamente.

Para o GLFS um repositório representa uma estrutura macro para organização dos arquivos, ele refere-se a um serviço de terceiro ou a um hardware específico para armazenamento. Dentro deste repositório é possível criar diversos depósitos, onde não obrigatoriamente, pode significar um cliente em uma relação de 1 para 1. Dentro dos depósitos de cada cliente podem ser criadas diversas galerias que representam um terceiro nível de organização. As galerias, por sua vez, serão formadas pelos arquivos de dados dos usuários. Sendo assim, dentro de um repositório previamente configurado, é possível criar n Depósitos dentro organizados em n Galerias, essas formadas por n arquivos. Como exemplo pode-se ilustrar uma estrutura empresarial, definida como: Em um repositório local, existe um depósito da Empresa XYZ com as Galerias Backup, Cadastros, Documentos e ISO.

Para garantir a integridade dos arquivos armazenados, o GLFS aplica alguns conceitos de segurança, sendo os principais:

- MD5: é utilizado para garantir que os dados estejam íntegros em várias etapas do GLFS;
- Transação: garante que o processo foi concluído, ou em qualquer falha que ocorrer o GLFS vai desfazer a operação, definimos isso como transação atômica;
- Metadados: é um pequeno banco de dados que armazena dados referentes aos arquivos para as operações internas;

4.4.1.1 Controles do GLFS ao armazenar um arquivo

Para o ERP armazenar um arquivo no GLFS, ele deverá instanciar uma classe responsável por implementar as chamadas para o Web Service do GLFS, consideramos que nesse ponto já ocorreu a autenticação no Web Service. Para fazer uma chamada ao método *push()*, deve-se enviar como parâmetro um objeto contendo os seguintes atributos:

- *filename*: nome original do arquivo: esse atributo serve para o GLFS armazenar em seu banco de metadados o nome original do arquivo para futura recuperação do mesmo e exibição com o nome original, pois em sua estrutura de terceiros e interna o arquivo é armazenado com o nome do *token*;
- *deposit*: nome do depósito: é o diretório principal do cliente, é com base nele que o GLFS saberá onde armazenar o arquivo;
- *gallery*: nome da galeria: indica que o arquivo deve ser armazenado dentro de um sub nível dentro de um depósito;
- *base64*: arquivo transformado em base64: esse atributo é onde o arquivo seja enviado ao GLFS no formato de base64, um padrão utilizado na transmissão de binários na internet;
- *MD5*: MD5 do arquivo original: vital para garantir a integridade dos arquivos, esse atributo é coletado no lado do cliente e enviado ao GLFS;
- *hash*: *hash* da *string* transformada em base64: é um *hash* calculado com base no arquivo transformado em base64 para o GLFS verificar se o arquivo chegou no lado do servidor de forma íntegra;

- *password*: uma senha para proteção do arquivo: a senha indicada aqui é utilizada para criptografia do arquivo e não é armazenada no lado do servidor.

É importante salientar que o MD5 do arquivo gerado neste momento ainda é de responsabilidade do ERP, bem como todas as informações sobre o arquivo, em posse das informações o método *push()* é invocado, e uma vez que o processo foi concluído com sucesso, o retorno será um objeto contendo o retorno de sucesso e um *token*. Essa combinação de *token* juntamente com a senha (*password*), deve ser armazenado pelo ERP e é a única forma de se recuperar o arquivo no futuro ou fazer novas operações com ele. Também é importante dizer que o ERP não sabe para onde o arquivo foi enviado, se ficou armazenado localmente no GLFS ou enviado para um servidor de terceiros, essa escolha é feita pelo GLFS conforme sua configuração.

Já no lado do GLFS, considerando um cenário de configuração de repositório local, o método *push()* vai primeiramente fazer alguns controles, o primeiro é verificar a integridade do arquivo em base64 recebido, para isso o GLFS gera o *hash* localmente e conferir com o *hash* enviado pelo ERP, em caso de falha na conferência, o GLFS retorna erro com a etapa que o processo falhou, isso vale para as próximas etapas sempre que ocorrer um erro.

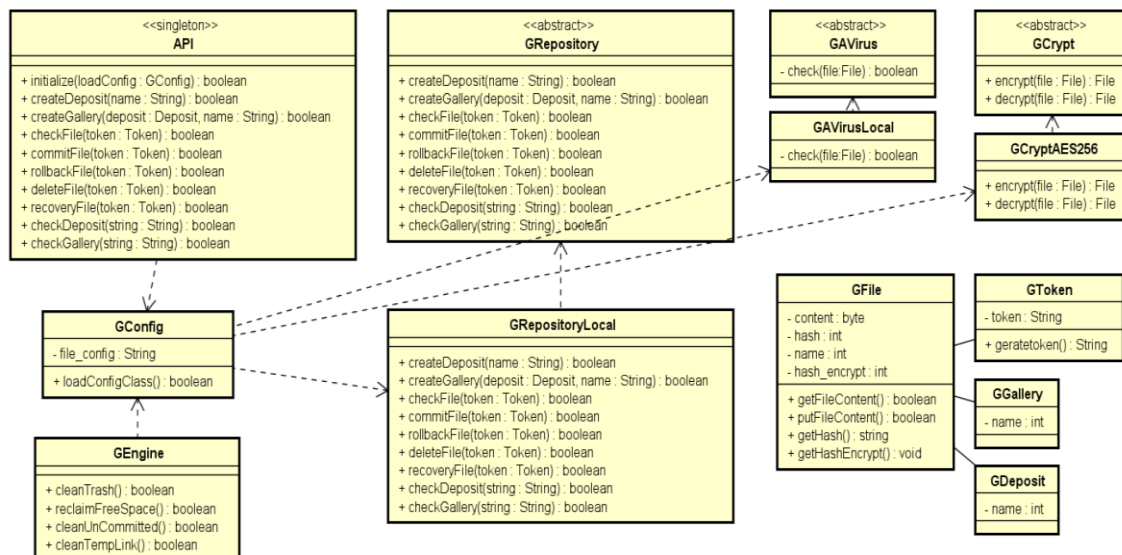
Em caso de sucesso é verificado a existência do depósito e da galeria, se existirem, o passo seguinte é gravar o arquivo em disco em sua forma original, guardar a informação do tamanho do arquivo e gerar um MD5 do arquivo e comparar com o MD5 do arquivo original enviado pelo ERP. Considerando que as conferências estão corretas, o GLFS agora vai iniciar o processo de verificação de vírus e criptografia. A classe de vírus não vai retornar um erro, apenas um alerta de que existe uma infestação no arquivo original, já a criptografia implementa alguns controles que podem retornar sucesso ou falha, após criptografar o arquivo com a senha enviada pelo ERP, será gerado um MD5 do arquivo criptografado que é armazenado no banco de metadados do GLFS juntamente aos dados gerais do arquivo: tais como: classe utilizada na criptografia, tamanho original do arquivo,

tamanho do arquivo criptografado, MD5 do arquivo original, MD5 do arquivo criptografado, depósito, galeria entre outras informações de metadados.

Na operação de *push()*, o GLFS implementa um controle de transação, que funciona de forma simples e funcional. No início do processo de recebimento de um arquivo, o servidor mantém um registro que essa operação foi iniciada (início da transação) e vai aguardar que o ERP envie uma confirmação do recebimento do *token*, o envio do próprio *token* recebido de volta ao servidor (fim da transação), marcando a transação como concluída com sucesso. Se algo algum erro ocorrer e o GLFS não receber a confirmação do ERP o processo fica marcado como pendente e será limpo na próxima iteração dos métodos de limpeza.

4.4.2 Modelagem/Diagrama de classes da API

Figura 3: Diagrama de classes - GLFS - Bloco das classes de negócio do GLFS.



Fonte: Elaborado pelo autor (2018).

Na Figura 3 são apresentadas as principais classes de negócio do GLFS, no modelo não são exibidas as classes que fazem parte da persistência no banco de metadados e classes de utilitários.

Existem três classes abstratas que seguem uma arquitetura que permite extensibilidade e integração com outros repositórios externos (a classe que

implementa um repositório, uma criptografia e uma verificação de vírus). Quando for necessário estender o GLFS para usar, por exemplo, a AmazonS3, será necessário apenas criar uma classe *GAmazonS3* estendendo a classe abstrata *GReposotory* que vai garantir que sejam implementados todos os métodos necessário para a API.

Uma vez implementadas as três classes, estas podem ser configuradas na classe *GConfig* que são automaticamente reconhecidas e carregadas para serem utilizadas pela API. O mesmo padrão de projeto é adotado para implementação de novas classes de criptografia e de verificação de vírus.

4.4.2.1 Controles do GLFS ao recuperar um arquivo

Para recuperar um arquivo o ERP deve invocar o método *pull()* ou o seu apelido *gerateLink()* passando para ele um objeto com os atributos:

- *token*: o *token* que foi gerado pelo GLFS no momento do armazenamento;
- *password*: a senha indicada pelo ERP no momento do armazenamento;
- *deposit*: o depósito que o arquivo está armazenado;
- *galeria*: a galeria que o arquivo está armazenado;
- *validate*: data de validade limite para fazer *download* o arquivo;
- *one_download*: indicação se o arquivo pode ser feito *download* uma única vez.

Se tudo estiver correto, o ERP vai receber uma lista com uma ou mais URLs para *download* seguro do arquivo.

No lado do GLFS, ao receber uma solicitação para recuperar o arquivo através da geração de URL estática, o processo inicia-se pela validação do *token*, depósito e galeria; se algo falhar o retorno indicará a etapa que o processo falhou e assim para as próximas etapas. Se tudo estiver correto, o GLFS gerar um novo *token* para usar nas URLs de *download*. A validade da URL é controlada por uma tabela auxiliar de metadados, contendo a validade da URL ou se o *download* foi indicado como único. Essa etapa não requer que o GLFS consulte onde está o arquivo, esse processo é feito apenas se a URL for acessada para economia de recursos.

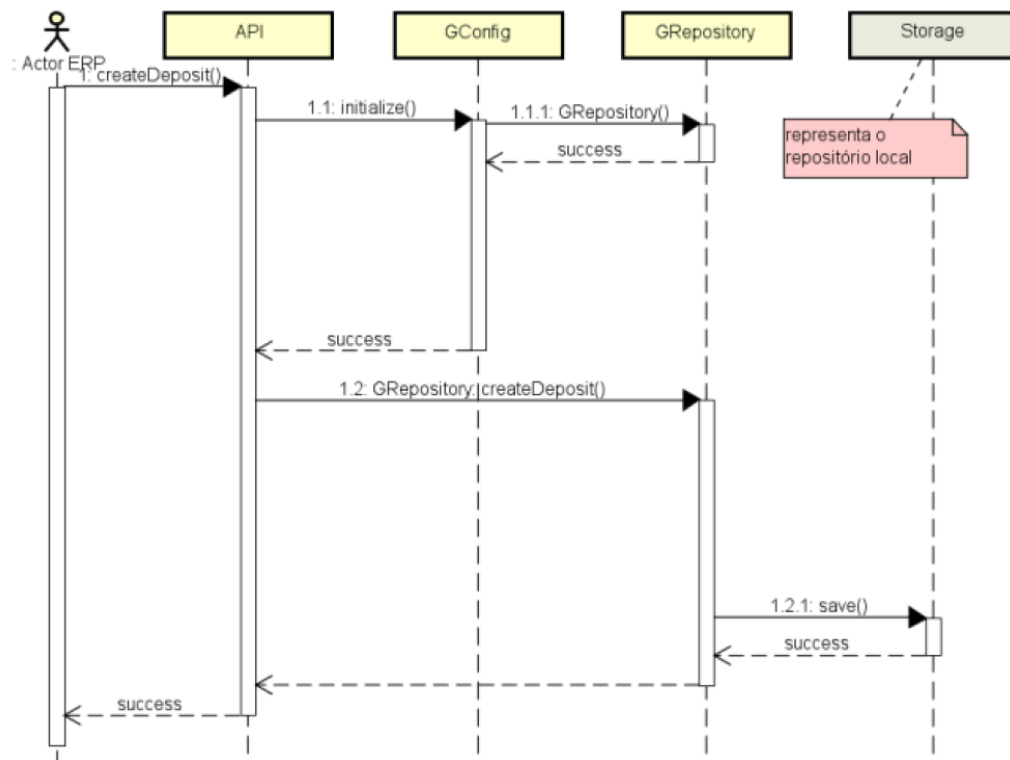
4.4.3 Diagrama de sequência da API

Os diagramas de sequência têm o objetivo de demonstrar o funcionamento de alguns processos do GLFS, tais como: enviar arquivo (*push*), gerar *links* para acesso aos arquivos (*gerateLink*), checar integridade do arquivo (*check*) e enviar configuração ao GLFS (*config*).

São demonstradas as interações entre as principais classes, no processo ainda existem dezenas de pequenas classes e métodos privados auxiliares que não aparecem nesse fluxo.

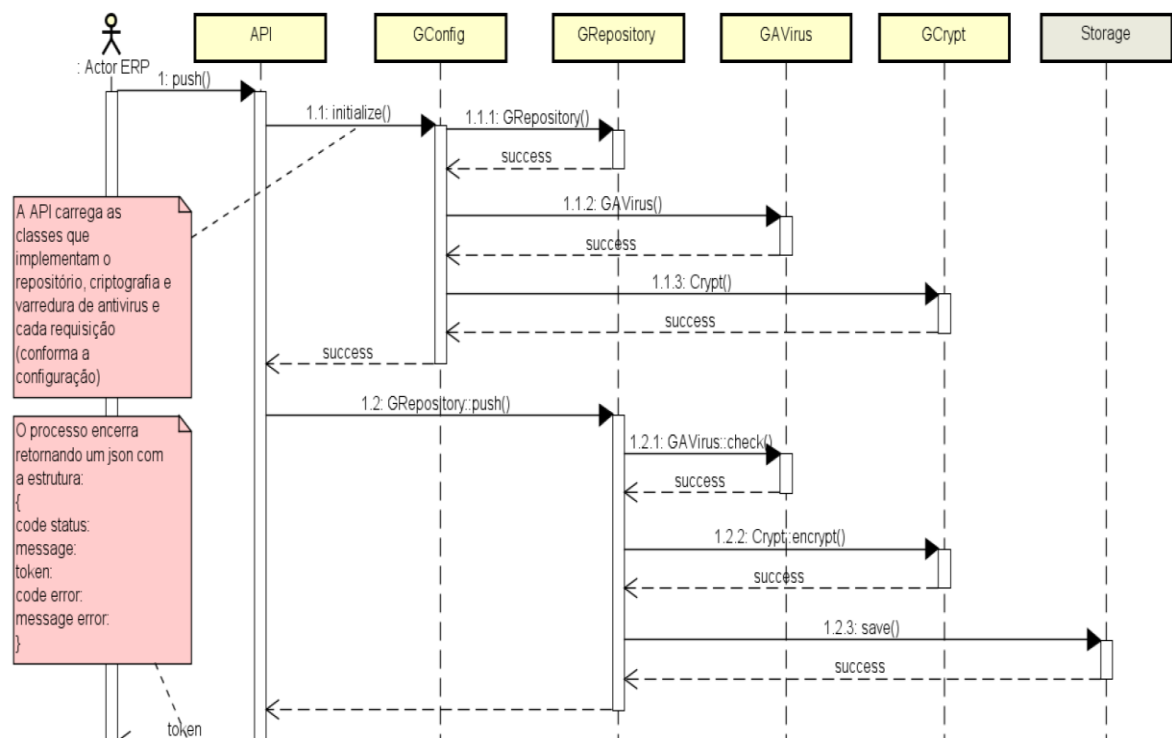
A Figura 4 mostra que a criação de um depósito é muito simples, bastando existir a classe que implementa um determinado repositório e estar devidamente configurada na classe de configuração do GLFS. A única implementação que vai exigir um nível de controle mais detalhado é no caso do repositório ser local, neste caso deve-se ter o cuidado para garantir que dentro da classe que implementa o método de criação garanta que o diretório de fato foi criado, caso contrário todo o processo vai sair pela *exception* (erro de execução) retornando um código de erro.

Figura 4: Diagrama de sequência da interação entre classes internas da API para criação de um depósito.



Fonte: Elaborado pelo autor (2018).

Figura 5: Diagrama de sequência da interação entre classes internas da API no envio de um arquivo.

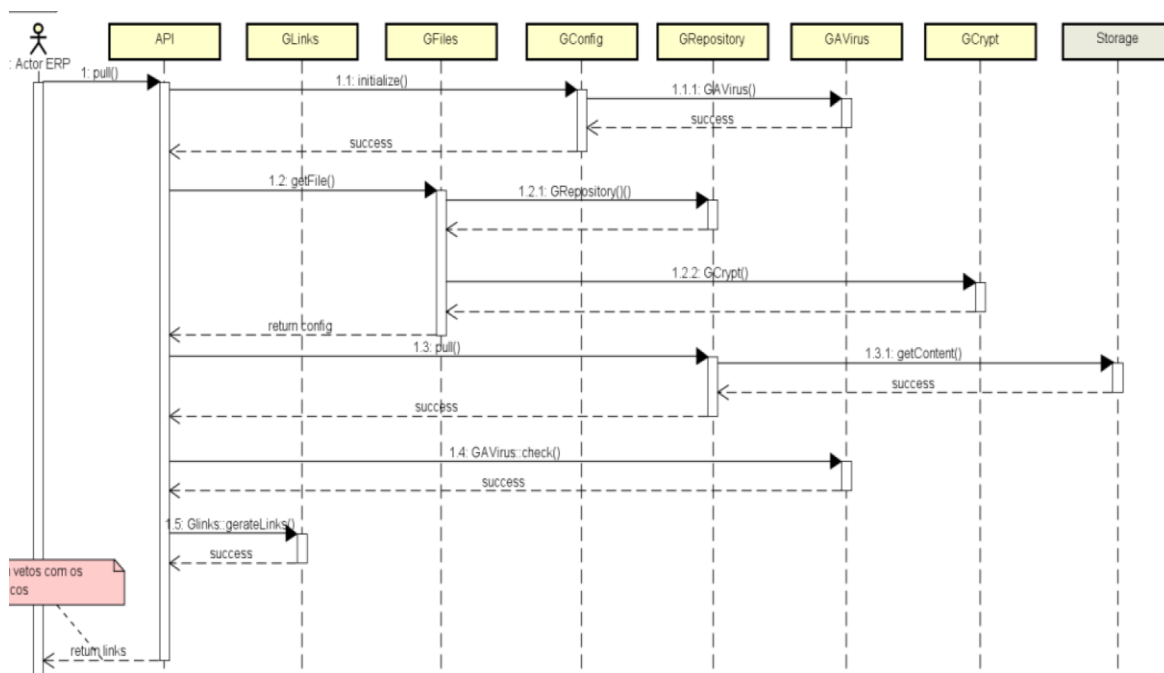


Fonte: Elaborado pelo autor (2018).

Na Figura 5 é possível observar que o GLFS inicializa o processo carregando as classes de comportamento do GLFS com base nas configurações, neste momento são *carregadas* as classes responsáveis por implementar um repositório, uma verificação de vírus e também uma implementação de criptografia. Após o *carregamento* das classes, todos os métodos são invocados das classes de comportamento que implementam as funções específicas de cada operação, esse padrão de carregamento baseia-se no padrão FACTORY, e as classes que implementam o comportamento da API no padrão STRATEGY.

Por exemplo, se a API for configurada para utilizar o serviço da Amazon, a classe AmazonS3 será *carregada* e terá a implementação da função *push()*, *gerateLink()*, *delete()*, *recovery()* e assim para todas as funções necessárias.

Figura 6: Diagrama de sequência da interação entre classes interna da API na recuperação de um arquivo.

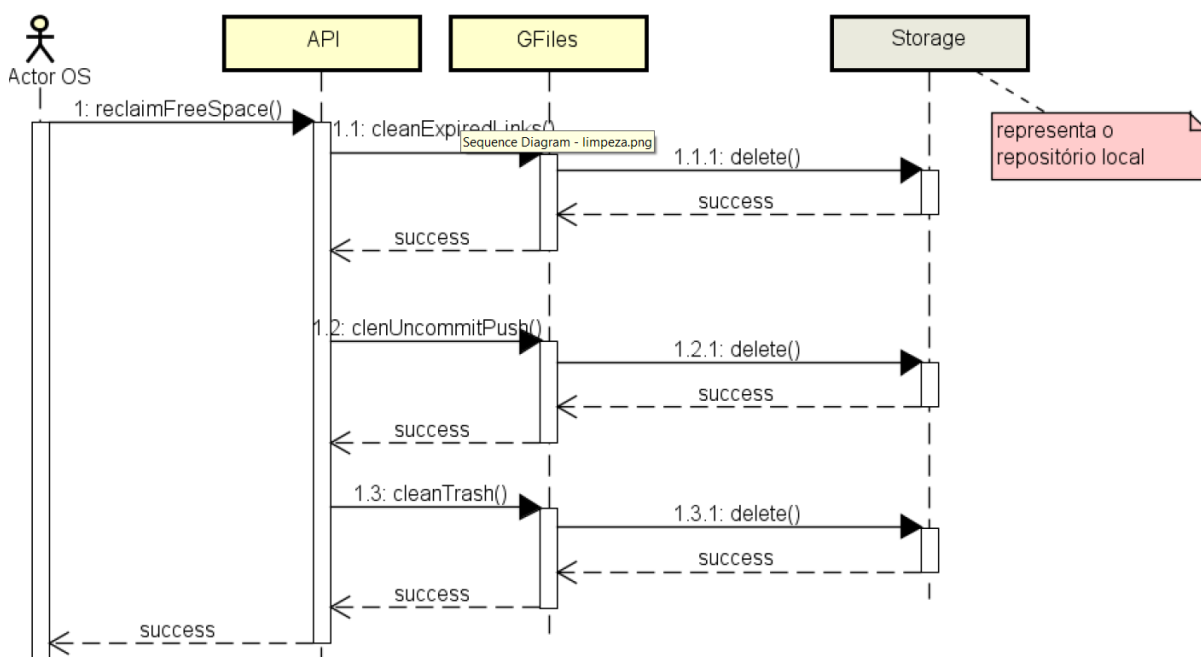


Fonte: Elaborado pelo autor (2018).

O digrama de sequência da Figura 6, mostra o processo de obtenção de acesso a um arquivo através de *links*. Nesse processos o GLFS carrega das configurações apenas a classe de verificação de vírus, para saber a classe que deve ser utilizada na descriptografar e a classe que implementa o local onde o arquivo esta armazenado, o GLFS faz uma consulta em seu banco de dados de metados na classe GFile. Após obter o arquivo criptografado do local armazenado, o GLFS

descriptografa o arquivo e com base na informações dos metadados originais, faz uma verificação da integridade utilizando as técnicas de MD5 e *hash*. O processo continua com a verificação da existência de vírus e finaliza com a geração dos *links* na classe Glinks; o GLFS devolve os *links* como resposta da solicitação inicial. É importante ressaltar que os *links* sempre são gerados com validade e isso é controlado pelo GLFS com base nos parâmetros que são enviados na requisição.

Figure 7: Diagrama de sequência da interação entre classes interna do fluxo de execução do método responsável por fazer uma limpeza do GLFS.



Fonte: Elaborado pelo autor (2018).

Na Figura 7, demonstra como é o funcionamento de limpeza do GLFS quando configurado para utilizar o repositório local. Existem métodos específicos para fazer a limpeza tais como: limpar arquivos excluídos que estão na lixeira (*trash*), limpar *links* temporários que estão com a validade expirada ou *download* único como realizado e fazer a limpeza de lixos de processos que não foram concluídos com sucesso. Esses métodos poderão ser executados sozinhos, ou todos de uma única vez, bastando invocar o método `reclaimFreeSpace()`, que significa reclamar por espaço livre fazendo uma limpeza geral. Esses recursos são configurados normalmente na cama da de sistema operacional em funções conhecidas como agendamento de tarefas.

Quando a implementação de um repositório for baseada em um serviço em nuvem, os métodos devem invocar essa limpeza conforme a API de terceiros, e caso não existir, ainda podem implementar algum tipo de limpeza com base nos dados locais do GLFS de falhas que ocorreram na transmissão.

4.4.3.1 Acesso a URL de *download* do arquivo

Quando um usuário do ERP requisita a URL de *download* do arquivo, o processo no lado do GLFS inicia-se pela verificação de validade, caso seja uma URL de *download* único é exibida uma mensagem que o *download* já foi realizado, ainda existem mensagem de que o *download* expirou ou de que não existe mais.

Se a URL está dentro da validade, o GLFS precisa verificar em seus metadados a classe que implementou o repositório, a criptografia e decompor a chave da URL que também é criptografada, dentro dessa chave contém o *token* do arquivo e a senha para descriptografia, observando que esse *token* não corresponde ao original do arquivo. Considerando que a classe que implementou o repositório é a do repositório local, é invocado desta a recuperação do arquivo criptografado que é descriptografado em tempo de execução, e liberado o link para *download*. Para economizar recursos e garantir a integridade, existem duas checagens de MD5, uma é antes de descriptografar e outra do arquivo descriptografado, o GLFS tem essas duas informações para comprar os MD5, qualquer um que venha a falhar, é exibido uma mensagem ao usuário de que o arquivo está corrompido.

Se o GLFS for configurado para verificar vírus, ao descriptografar o arquivo em sua forma original, uma verificação é iniciada em paralelo de forma assíncrona e é exibida ao usuário as informações assim que a verificação termina.

4.4.3.2 Outros métodos no GLFS

O GLFS possui outros métodos para manipulação dos arquivos e manutenção dos repositórios, sendo as principais:

- *cleanUncommitted*: utilizado para fazer a limpeza de processos que não foram confirmados o seu término (encerramento da transação);
- *cleanExpiredTempLink*: faz a limpeza de URLs que estão fora da data de validade ou que o *download* já foi executado;
- *createDeposit*: cria um novo depósito;
- *createGallery*: cria uma nova galeria;
- *checkFile*: verifica se um arquivo esta íntegro;
- *commitFile*: confirma o final da transação;
- *rollbackFile*: desfaz a operação em caso de falha;
- *deleteFile*: exclui um arquivo;
- *recoveryFile*: recupera um arquivo excluído;
- *checkDeposit*: verifica se um depósito existe;
- *checkGallery*: verifica se uma galeria existe;
- *getFileHex*: recupera um arquivo em formado hexadecimal;
- *getFileBase64*: recupera um arquivo em formado base64;

- *inTrash*: retorna se um arquivo está na lixeira;
- *getSizeDeposit*: retorna o tamanho de um depósito;
- *getSizeGallery*: retorna o tamanho de uma galeria;
- *deleteDeposit*: exclui um depósito;
- *deleteGallery*: exclui uma galeria.

4.4.4 Segurança do GLFS

Nessa seção, é explicado como o GLFS mantém os arquivos de forma segura seja em seu repositório local ou em serviços externos.

4.4.4.1 Comunicação

A comunicação do GLFS é feita em HTTPS (SSL), não são aceitas conexões em HTTP, isso evita que dados sejam capturados no momento do envio do arquivo. O Web Service do GLFS é autenticado através do próprio cabeçalho de envio (SOAP Header), isso permite que uma vez autenticado, o ERP possa fazer chamadas para diversos métodos enquanto a conexão não for quebrada ou encerrada.

4.4.4.2 Criptografia dos arquivos

É possível configurar o GLFS para manter os arquivos no estado original ou de forma criptografada. Para isso existem classes específicas que implementam diferentes tipos de criptografia, dentre elas a mais forte do tipo SSL AES-256-CBC, que usa uma chave de 256 bits baseada em uma senha.

A classe de encriptação foi projetada para que cada arquivo seja encriptado por uma senha única que é fornecida pelo ERP, a decisão de mudar a senha a cada envio é do ERP, mas pode-se utilizar a mesma palavra-passe para todos os arquivos.

No lado do GLFS a senha para descriptografar o arquivo não é armazenada, ficando sobre total responsabilidade do ERP guardar essa senha juntamente do *token*. Essa estratégia busca permitir que o GLFS possa ser instalado em servidores de terceiros ou replicados para servidores, por exemplo, da Google, suspeitos por minerar informações de clientes, mantendo alto sigilo das informações.

Caso seja implementada uma classe que utiliza um serviço de terceiros que já usa algum tipo de criptografia no armazenando, o GLFS pode ser configurado para não criptografar os arquivos, dessa forma não fica redundante.

4.4.4.3 Verificação de vírus

Outra forma de manter os arquivos seguros é aplicar uma configuração para que o GLFS faça a varredura de vírus no envio e na recuperação do qualquer arquivo, isso é muito importante para controlar uma possível disseminação de vírus sem o conhecimento do usuário.

A verificação funciona mesmo em arquivos compactados, mas essa é uma característica da ferramenta utilizada para verificação, dentre as implementadas, o ClamAV e VirusTotal têm essa peculiaridade.

4.4.4.4 Token, controle de *download*

O *token* é o principal conceito envolvido ao se trabalhar com o GLFS, é ele quem identifica um arquivo. O *token* é gerado pelo GLFS ao se armazenar um arquivo, a partir dele é possível fazer todas as outras operações disponíveis.

Para o *download* sempre é gerado um *token* novo, esse *token* é um nome simbólico e temporário que aponta para o arquivo original. A URL fornecida para

download é seguida de um prazo de validade, na hora de gerar as URLs é obrigatório informar a data de validade e se o *download* pode ser feito uma única vez.

4.4.5 Extensibilidade

O GLFS foi pensado desde o início para ser estendido a outras tecnologias. Conforme explicado no tópico 4.3.4.1, toda comunicação é feita por Web Service, isso permite que o GLFS seja utilizado por qualquer aplicação que tenha suporte a Web Services, o que é muito trivial entre as linguagens de programação atuais. Essa arquitetura deixa GLFS com baixo acoplamento ao ERP e permite que ele possa ser implementado em qualquer linguagem e modificado a qualquer momento, desde que mantenha os métodos de acesso iguais (padrão FACADE).

Internamente o GLFS adota o padrão de fábrica para carregamento das classes responsáveis por implementar a criptografia, verificação de vírus e o repositório de armazenamento. Todas essas classes são estendidas de classes abstratas que forçam que elas tenham os métodos implementados. Por exemplo, se precisamos adotar um novo repositório de terceiros, basta criar uma classe do novo serviço estendendo a classe abstrata *GRepository* e implementar todos os métodos para atender ao novo serviço, feito isso, a classe já é carregada automaticamente pela *engine* do GLFS e esta pronta para ser usada, bastando indicar no arquivo de configuração essa classe.

É importante salientar que existem níveis diferentes para cada implementação de repositório. Ao implementar um repositório da AmazonS3 é necessário apenas utilizar a API de terceiros e toda complexidade de integridade e segurança é de responsabilidade deles, o GLFS preocupa-se apenas em garantir a existência de metadados para recuperação de arquivos. Já ao implementarmos um repositório utilizando o disco local, vai existir toda uma preocupação com a integridade do arquivo, ou seja, a classe que implementa o repositório local terá uma série de

métodos privados para garantir que o arquivo não se corrompa em algum dado momento.

Esse padrão fábrica (Factory) é adotado nas classes de verificação de vírus, criptografia e repositório, são os três recursos que provavelmente vão necessitar de novas implementações em um futuro.

4.4.6 Transparência e facilidade na utilização do GLFS pelo ERP

Para o ERP, a utilização da API do GLFS é uma tarefa muito simples, bastando-se configurar o arquivo de configuração que contem também os dados de autenticação; feito isso o ERP já pode manipular arquivos, criar depósitos, galerias, enviar, excluir, recuperar e todas as funções descritas no tópico 4.3.3.4.

Figura 8: Exemplo de código mostrando a simplicidade no uso do GLFS.

```

33
34
35 /* Send new file */
36 $glfs = new GLFS();
37 $glfs->setDeposit('deposito.tales');
38 $glfs->setGallery('arquivos');
39 $glfs->setPassEncrypt('5q5yEHnG49quyGYp');
40
41 $response_object = $glfs->push('tmp/file_656.png');
42 $token = $glfs->getToken();
43
44 $glfs->commitFile($token); // commit file;
45
46 /* Delete exist file */
47 $glfs = new GLFS();
48 $glfs->setDeposit('deposito.tales');
49 $glfs->setGallery('arquivos');
50 $response_object = $glfs->deleteFile('195844b1-15c7-459d-a4ad-304d2b9fd968');
51
52 /* Recovery file */
53 $glfs = new GLFS();
54 $glfs->setDeposit('deposito.tales');
55 $glfs->setGallery('arquivos');
56 $response_object = $glfs->recoveryFile('195844b1-15c7-459d-a4ad-304d2b9fd968');
57

```

Fonte: Elaborado pelo autor (2018).

Na Figura 8, o trecho entre as linhas 35 a 44 mostra a forma como é armazenado um arquivo, detalhe na linha 44 onde é necessário chamar o método

commitFile() passando o *token* recebido para que a transação seja finalizada. Essa transação é aberta automaticamente no início de cada processo. O trecho da linha 46 a 50 demonstra o processo de exclusão de um arquivo a partir do *token*; e da linha 52 a 56 o processo de recuperação de um arquivo excluído, normalmente utilizado em *try{} catch{}* onde ocorre o tratamento de erros que se faz necessário desfazer uma exclusão.

4.4.7 Métodos para usos específicos

Para um software é muito comum existirem necessidades específicas, uma delas diz respeito ao armazenamento e recuperação de imagens, logo o ERP GreenApp faz uso deste recurso armazenando localmente apenas imagens miniaturizadas e a imagem original no GLFS.

Quando o ERP precisar mostrar a imagem ao usuário, ele faz uma chamada a um método específico *getFileBase64()*; isso permite que o ERP receba uma string em base64 da imagem e utilize diretamente na camada de visão (front-end) da aplicação, com isso o ERP trabalha somente com a memória RAM aumentando a performance, do contrário seria necessário fazer algumas outras etapas como salvar a imagem em um diretório temporário para depois carregá-la. Isso é possível por que o HTML permite incorporar uma imagem em HTML *inline*, ou seja, dentro da declaração de imagem embutir a string em base64.

4.4.8 Estatísticas

Um dos objetivos da solução é permitir que o nível estratégico possa responder perguntas do tipo: Quais são os clientes que mais trafegam dados?; Quais os clientes que mais armazenam conteúdo?; Quais são os horários de picos que geram possíveis gargalos?

Para extrair essas informações, foi necessário criar um banco de dados para armazenar os dados estatísticos. Nesse banco de dados são armazenadas informações como:

- Dados sobre o arquivo enviado: repositório, galeria, tamanho do arquivo bruto, tamanho do arquivo criptografado, status da transação, data, hora, ip de origem, classe de criptografia utilizada, *token*;
- Dados sobre a geração de links: repositório, galeria, tamanho do arquivo bruto, tamanho do arquivo criptografado, data, hora, IP de origem, *token*, *token* temporário, link utilizado, URL, data de expiração, indicação de *download* único;
- Dados sobre acessos a URL de download: repositório, galeria, tamanho do arquivo bruto, tamanho do arquivo criptografado, verificação de vírus, data, hora, ip de origem, *token*, *token* temporário, link utilizado, URL, data de expiração, indicação de *download* único, dados do navegador do cliente;
- Dados sobre exclusão: repositório, galeria, tamanho do arquivo bruto, tamanho do arquivo criptografado, data, hora, ip de origem, *token*;
- Dados sobre recuperação de arquivos: repositório, galeria, tamanho do arquivo bruto, tamanho do arquivo criptografado, data, hora, ip de origem, *token*;
- Dados sobre a manutenção de limpeza: repositório, galeria, tamanho do arquivo bruto, tamanho do arquivo criptografado, data, hora, ip de origem, *token*, origem da limpeza (transação incompleta, lixeira ou link expirado).

Os dados de estatísticas parecem redundantes, pois em cada tabela os dados estão repetidos, no entanto essa modelagem é necessária por não existir relação entre as tabelas, ou seja, no momento do registro é armazenado tudo referente aquele arquivo. Também existe um motivo para essa modelagem, a facilidade em agrupamentos, contagens e a montagem de relatórios sobre diferentes perspectivas de visão em um grande volume de dados.

4.4.9 Testes e validações

O GLFS foi colocado em produção depois de uma sequência de testes em homologação em novembro de 2017. Para esse teste foram selecionados 49 bases de dados com 208 usuários ativos, 9.174 mil arquivos anexos totalizando 27 gigabytes. A solução foi monitorada durante 3 meses. Para esse grupo de usuários o GLFS operou de forma paralela, ou seja, para cada arquivamento ou exclusão de um arquivo dentro do GreenApp, o GLFS era invocado para executar a mesma tarefa, desta forma o processo não afetou o funcionamento do ERP nem comprometeu ou acarretou perda de dados dos usuários.

Quando um cliente acessava um arquivo do ERP, em primeiro momento a busca ocorria no GLFS para gerar os *links* de acesso ao arquivo, em caso de insucesso, o ERP usa o arquivamento das pastas locais e todos os erros eram armazenados nos Logs. Nestes processos de falhas qualquer erro ocorrido do GLFS era encapsulado e não era exibido em tela, apenas monitorado pelos programadores. Pequenas correções e ajustes foram necessários durante os testes, e nos últimos 45 dias o GLFS operou sem interrupções nos processos.

O GLFS utiliza um *link* de conectividade de alta velocidade exclusivo para *download* dos arquivos, esse *link* permite que arquivos atinjam até 100 megabits *full-duplex*, ou seja, tráfego para *upload* e *download*, permitindo que arquivos sejam recuperados o mais rápido possível mesmo que primeiro precisem ser baixados da nuvem de terceiros. O servidor de repositório é um *storage* de alto desempenho com conectividade *Gigabit* que alcança até 125 megabits por segundo.

4.4.10 Avaliação

Um dos objetivos da ferramenta é possibilitar a análise dos dados mesmo que o local do armazenamento não forneça tais indicadores, a exemplo dos serviços terceirizados, para isso, o GLFS faz esta análise com base nos dados de metadados e arquivos de *log* internos, mesmo se os arquivos estiverem espalhados em mais de um serviço de arquivamento o GLFS será capaz de agrupar essas informações.

Na Figura 9 um gráfico foi gerado com base nos dados dos metadados, corresponde a um exemplo de como podem ser exibidos gráficos e agrupamentos dos dados que estão sendo gerenciados no GLFS. Nesse exemplo foram realizados agrupamentos e somatórios, tornando possível saber que existe com precisão 14.772 arquivos, com um total de 40,91 Gigabytes de *upload* e 1.51 Gigabytes de *download*. Ainda observa-se com base nos *links* gerados e *links* acessados que foram economizados 0.12 Gigabytes (aproximadamente 120MB) de *links* que os destinatário receberam mas não clicaram para acessar, isso se dá pelo fato de que nem todo mundo acessa os *links* que recebem.

Figura 9: Gráfico gerado com base nos metadados e logs do GLFS.



Fonte: Elaborado pelo autor (2018).

Com os metadados e *logs* é possível analisar cliente a cliente, montando gráficos capazes de indicar como anda o comportamento do cliente, por exemplo, um cliente que tenha apenas 1GB armazenado em disco pode gerar muito mais

tráfego que um cliente que esteja com 10GB de arquivos armazenados, isto significa que ele terá um custo mensal de manutenção muito mais elevado, afinal, tráfego de internet é pago através de franquia de dados, por isso a importância de além do espaço em disco utilizado, a forma como ele trafega e utiliza esses dados.

5 CONSIDERAÇÕES FINAIS

No contexto de software de gestão empresarial, as empresas têm a necessidade de armazenar e acessar as informações em um volume de dados cada vez maior. A mobilidade e facilidade no acesso da informação é um requisito nos sistemas de gestão modernos que contribuem para que as empresas e usuários estejam à frente daqueles que possui sistemas *offline*. A utilização dos serviços de Computação em Nuvem vem sendo uma das opções mais utilizadas para suprir essa demanda, seja pela segurança, facilidade, rapidez ou flexibilidade. Recursos como compartilhamento, edição simultânea, acesso remoto a partir de qualquer local e quase de qualquer dispositivo, motivam os usuários a armazenar seus dados e arquivos em sistemas que estão sempre *online*. O fator segurança sempre é questionado a cada nova solução proposta para a Nuvem, no entanto, a cada dia mais empresas estão aderindo a esse modelo de armazenamento e compartilhamento de dados devido à maturidade e à confiança fornecida pelos provedores desses serviços.

Os sistemas de informação, de maneira análoga e geral, buscam oferecer aos seus usuários acesso rápido, seguro e eficiente aos dados e informações, e até mesmo aos arquivos extras anexados. A facilidade de fazer a gestão da informação por meio de dispositivos móveis e rapidamente obter acesso a um boleto, ao *download* de arquivo ou obtenção de uma informação, estimula o uso de soluções que executem todos os serviços na Web. Empresas com filiais espalhadas

geograficamente não sentem a distância devido à transparência oferecida pelas soluções dos sistemas via Internet.

A arquitetura proposta pela solução GLFS demonstra a viabilidade de estruturar os negócios da área de Tecnologia de Informação e tê-los conectados aos serviços de armazenamento em nuvem de forma segura e transparente. Por meio de sua estrutura de classes, há facilidade de escalar e gerenciar serviços, hardware e *links* de acesso, de maneira fácil e rápida.

Quando comparados solução antiga e solução atual, a solução GLFS aprimora o processo otimizando os recursos e oferecendo segurança através de criptografia e verificação antivírus. No intuito de tornar a solução GLFS mais robusta, mapeou-se como requisitos futuros:

- recuperação de arquivos de áudio e vídeo em formato de *streamming*,
- trabalhar com *multicasting* uma transmissão para vários clientes;
- implementar controle de versão, permitindo arquivos serem recuperados baseado em data de alteração e versionamento;
- *download* de todos os arquivos de um cliente, mesmo que em diferentes repositórios.

REFERÊNCIAS

AMAZON. **Pricing 2017**. Disponível em: <<http://aws.amazon.com/s3/#pricing>>. Acesso em: set. 2017.

ANDRADE H. Vinicius. **Avaliação da ferramenta Owncloud como solução de cloud computing privada**. Lavras: Monografia em Sistemas de Informação. UFLA, 2014.

ARTIGAS Marc Sánchez; COTES Cristian; RODRIGUES Marc R; LOPEZ Pedro García. **StackSync: Attribute-based data sharing in file synchronization services**. Concurrency and Computation Practice and Experience. Trabalho de conclusão de curso, dez. 2017.

BUYYA, R., YEO, C. VNUGOPAL. **Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing**, 2009.

CHEE Brian J.S.; FRANKLIN Curtes Jr. **Computação em nuvem - Cloud Computing. Tecnologias e estratégias**. Editora MBooks 247 p., 2012.

COMMER Douglas E. **Redes de Computadores e Internet**.4 e.d. Editora Bookman 608 p., 207.

DALL’OGLIO, Pablo. **PHP: Programando com orientação à objetos**. 3 ed. São Paulo: Novatec, 2015.

DATE, C. J. **Introdução a sistemas de bancos de dados**.8.ed. Rio de Janeiro: Campus, 2004.

DEITEL Harvey M. **An Introduction to Operating Systems**, 2 ed. Addison-Wesley, Reading, MA, 1992.

ELMASRI, R; NAVATHE, S. B. **Sistemas de banco de dados**. 6 ed. Pearson, 2011.

ENDREI Mark; ANG Jenny; ARSANJANI Arsanjani; CHUA Sook; COMTE Philippe; KROGDAHL PØI; LUO Min; NEWLING Tony. **Patterns: Service-Oriented Architecture and Web**. New York: IBMRedbooks, 2004.

FOWLER, Martin. **Patterns of enterprise application architecture**. Bostom: Addison Wesley, 2003.

GAMMA, Erich; SALGADO, Luiz A. Meirelles. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2008.

GIL, Antonio Carlos. **Como Elaborar Projetos de Pesquisa**. 4 ed. Editora Atlas,

1999.

GRAHAM Steve Thomas; LIO Xiaodong. **Critical Evaluation on jClouds and Cloudify Abstract APIs against EC2, Azure and HP-Cloud**. International Computer Software and Applications Conference Workshops. 39 ed. Vasteras: 2014, p. 510-515.

GRAUPNER Hendrik; TORKURA Kennedy; BERGER Philipp; MAINEL Christoph. **Secure Access Control for Multi-Cloud Resources**. Annual IEEE Conference on Local Computer Networks. 40 ed. Florida: 2015, p. 722-729.

GOMES, Daniel A. **Web Services SOAP em Java. Guia prático para o desenvolvimento de web Services em Java**. 1 ed. São Paulo: Novatec, 2010.

GOOGLE. **API REST**. Disponível em <<https://developers.google.com/fit/rest/>>. Acesso em: ago. 2017.

GOOGLE. **App Engine**. Disponível em:<<http://code.google.com/appengine/>>. Acesso em: ago. 2017.

INTEL. Intel® **Advanced Encryption Standard (AES) New Instructions Set**. Disponível em <<https://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf>> Acesso em: ago. 2017.

JSON. **Introducing JSON**. Disponível em <<http://json.org>>. Acesso em: ago. 2017.

LOPES, Carlos J.F.; RAMALHO, José C. **Web Services: Metodologia de Desenvolvimento**. Porto, 2004. Disponível em: <<http://www4.di.uminho.pt/~jcr/XML/publicacoes/artigos/2004/LR04.pdf>>. Acesso em ago. 2017.

MEDYK, Sérgio. **Web Services em Gerência de Redes**. Paraná, 2006. Disponível em: <<http://ppgia.pucpr.br/~jamhour/Download/pub/Mestrado%202006/Web%20Service.pdf>> Acesso em ago. 2017.

MILANI, A. **POSGRES: Guia do Programador**. São Paulo: Novatec, 2008.

MORAES, Alexandre Fernandes. **Segurança em redes. Fundamentos**. São Paulo Érica 262p, 2012.

PRESSMAN, R. **Engenharia de Software: Uma Abordagem Profissional**. 7 ed. McGraw-Hill, 2011.

POSTGRESQL. **PostgreSQL 9.1.24 Documentation. The PostgreSQL Global Developer Group**. Disponível em: <<https://www.postgresql.org/files/documentation/pdf/9.1/postgresql-9.1-A4.pdf>>. Acesso em: ago. 2017.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistemas de Gerenciamento de Banco de**

Dados. 3 ed. McGraw-Hill, 2008.

RICHARDSON, Leonard; RUBY, Sam. **RESTFull Web Services. Web Services for the real world.** O'Reilly Media: Sebastopol, 2007.

ROB, P.; CORONEL, C. **Sistemas de Bando de Dados:** Projeto, implementação e administração. 8 ed. CENGATE: Learning, 2011.

ROVARIS, Felipe M. **Automatização na cotação de livros utilizando Web Services**, 2007. Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_583/TCC%20FELIPE%20pronto.pdf>. Acesso em: ago. 2017.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de Bancos de Dados.** 3 ed. Makron Books, 2012.

SOMMERVILLE, Ian. **Engenharia de Software.** 3 ed. Editora Pearson Education Brasil, 2012.

TANENBAUM, Andrew S. **Distributed Operating Systems.** Prentice Hall, Upper Saddle River, NJ, 1995.

TOBALDINI, Ricardo Ghisi. **Aplicação do Estilo Arquitetural REST a um Sistema de Congressos.** Florianópolis: Monografia em Ciências da Computação – UFSC, 2008.

W3C. **Padrões.** Disponível em: <<http://www.w3c.br/Padroes/>>. Acesso em: out. 2017.