CENTRO UNIVERSITÁRIO UNIFACVEST
CIÊNCIA DA COMPUTAÇÃO
CAMILA MELO MACHADO

AUDIT.CRUD: FRAMEWORK PARA PROVENIÊNCIA DE DADOS DE LOGS DE AUDITORIA

Lages - SC
2020

CAMILA MELO MACHADO

AUDIT.CRUD: FRAMEWORK PARA PROVENIÊNCIA DE DADOS DE LOGS DE AUDITORIA

Projeto apresentado à Banca Examinadora do Trabalho de Conclusão de Curso II de Ciência da Computação para análise e aprovação.

Lages - SC

2020

CAMILA MELO MACHADO

AUDIT.CRUD: FRAMEWORK PARA PROVENIÊNCIA DE DADOS DE LOGS DE AUDITORIA

Trabalho de conclusão de curso apresentado à Banca Examinadora da Unifacvest como parte dos requisitos para obtenção do título de bacharel em Ciência da Computação.

Orientador: Marcelo Goulart Souza
Coorientadores:
Me. Márcio José Sembay

Lages, SC ___/___/2020.
Nota _____

_____

Coordenador do curso de graduação

Lages - SC

2020

Dedico este trabalho a todos os professores do curso, ao orientador do trabalho, aos familiares, amigos e todos aqueles que de certa forma contribuíram para realização desse projeto.

AGRADECIMENTOS

A Deus por me proporcionar perseverança durante toda a minha vida.

Aos meus pais Nelton e Rozeli por sempre me incentivarem e acreditarem que eu seria capaz de superar os obstáculos que a vida me apresentou.

Agradeço ao meu orientador Marcelo Goulart Souza por aceitar conduzir o meu trabalho.

Sou grata aos professores Márcio José Sembay e Igor Muzeka pelo apoio técnico prestado durante todo o desenvolvimento do projeto.

Agradeço a meu amigo Giovani que sempre me ajudou e incentivou com sua vasta experiência desde o início deste trabalho.

Também agradeço todos os meus amigos do curso de graduação que compartilharam dos inúmeros desafios que enfrentamos, sempre com o espírito colaborativo.

RESUMO

A integridade é um dos pilares da Segurança da Informação, assegurar que os dados sejam mantidos com precisão e consistência e protegidos contra modificações intencionais ou acidentais é uma tarefa desafiadora. Pesquisas recentes mostram que a utilização de proveniência de dados pode facilitar o gerenciamento de dados em ambientes computacionais. Neste contexto, usaremos a proveniência de dados para a produzir trilhas de auditoria, com objetivo de gerar um histórico das ações que usuários realizam dentro de um sistema, a fim de auxiliar na preservação de dados sensíveis e contribuir no mapeamento de fraudes e violações. Aplicar trilhas de auditoria em um sistema, pode causar impactos na arquitetura e na performance, além de produzir um grande volume de dados. Diante disso, o objetivo do presente trabalho foi a criação de um protótipo de *framework* em .NET Standard com o intuito de auxiliar a comunidade de desenvolvedores aplicarem logs de auditoria em Web APIs .NET Core, oferecendo agilidade e padronização na implantação, tanto como, compatibilidade com ferramentas emergentes e *open sources* capazes de gerenciar grandes volumes de dados. Esse *framework* deverá causar o menor impacto possível na arquitetura e performance da *Web API*, auxiliando na coleta, armazenamento e análise de dados, utilizando tecnologias *NoSQL, Elasticsearch, Kibana* e *containers*.

Palavras-chave: *Segurança da informação; Logs de auditoria; Gerenciamento de logs.*

ABSTRACT

Integrity is one of the pillars of Information Security, ensuring that data is maintained accurately and consistently and protected against intentional or accidental changes is a challenging task. Recent research shows that using data provenance can ease data management in computing environments. In this context, we will use data provenance to produce audit trails, aiming to generate a history of the actions that users perform within a system, in order to assist in the preservation of sensitive data and contribute to the mapping of fraud and violations. Applying audit trails in a system can impact in its architecture and performance, and also produce a large volume of data. Therefore, the objective of this work was to create a framework prototype using .NET Standard in order to assist the developers community in applying audit logs in .NET Core Web APIs, offering agility and standardization in the deployment, both as, compatibility with emerging and open sources tools capable of managing large volumes of data. This framework should have the least possible impact on the web API's architecture and performance, assisting in data collection, storage, and analysis, using NoSQL, Elasticsearch, Kibana, and containers technologies.

Keywords: *Information security; Audit logs; Logs management.*

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API	Application Programming Interface ou Interface de Programação de Aplicativos.
ABNT	Associação Brasileira de Normas Técnicas.
BDR	Banco de dados relacional.
ELK	Elasticsearch, Logstash, Kibana.
ES	Elasticsearch.
HTTP	Hypertext Transfer Protocol ou Protocolo de Transferência de Hipertexto.
HTTPS	Hyper Text Transfer Protocol Secure.
ISO	International Organization for Standardization.
JSON	JavaScript Object Notation ou Notação de Objetos JavaScript.
NBR	Norma Técnica Brasileira.
NoSQL	Not Only SQL ou Não Apenas SQL.
POO	Project Oriented Programming ou Programação Orientada a Objetos.
REST	Representational State Transfer ou Transferência de Estado Representacional.
SOAP	Simple Object Acces Protocol ou Protocolo Simples de Acesso a Objeto.
SQL	Structured Query Language ou Linguagem de Consulta Estruturada.
URI	Uniform Resource Identifier ou Identificador Uniforme de Recurso.
W3C	World Wide Web Consortium.
WEB	World Wide Web ou Rede de Alcance Mundial.
XML	eXtensible Markup Language.

SUMÁRIO

1 INTRODUÇÃO

O termo “proveniência” trata-se da origem ou procedência dos dados, ou seja, um registro histórico de derivação de dados, possibilitando a reprodução, interpretação e diagnóstico de problemas (LIM; LU; CHEBOTKO; FOTOUHI, 2010).

A proveniência de dados descreve a criação de um dado, o processo pelo qual passou, até seu resultado final, permitindo encontrar sua origem e as transformações que foram feitas, até seu momento atual (FREIRE et al., 2008). No contexto da segurança da informação, os dados de proveniência possibilitam a criação de trilhas de auditoria, que é um recurso fundamental para a proteção das informações contidas em softwares.

Um dos desafios enfrentados pela área de desenvolvimento de software é entrega de soluções de forma rápida e que gere valor para o cliente. É essencial que o processo de desenvolver *softwares* seja focado na entrega com qualidade e segurança, evitando que aconteçam incidentes futuros.

A auditabilidade é um dos princípios da segurança da informação, porém, a aplicação dela em *softwares* pode ser custosa e demandar uma grande quantidade de tempo. Uma das formas de driblar essa situação é a utilização de *frameworks* que auxiliem a implementação deste princípio.

Um *framework* pode ser definido como um conjunto de classes que incorpora um projeto abstrato, com objetivo de resolver um problema recorrente, usando uma abordagem genérica. Com *framework,s* as aplicações podem ser desenvolvidas mais rapidamente, e por conterem estruturas similares, a manutenção também se torna mais fácil (LIMA, 2011).

O presente estudo foca na criação de um *framework* para Web API’s .NET Core, fornecendo uma interface para a captura de dados de proveniência, capaz de gerar trilhas de auditoria e sustentar o pilar da auditabilidade levantado pela segurança da informação.

1.1 Justificativa

No mundo do desenvolvimento de *software*, existe uma corrida contra o tempo entre as organizações para lançar novos produtos ao mercado a fim de disputar o público-alvo. Devido a este tempo curto para entregas, muitas vezes a segurança é negligenciada.

A auditoria é um dos pilares da segurança da informação, assegurar que os dados sejam mantidos com precisão e consistência e protegidos contra modificações intencionais ou acidentais é uma tarefa crucial. Apesar da trilha de auditoria poder ser considerada um requisito não funcional, é de extrema importância, pois agrega valor e confiabilidade ao *software*, afinal, ter um histórico de tudo que foi feito ou consultado em um sistema de informação é um recurso precioso para qualquer administrador. Segundo Freire et al. (2008, p. 21) "explorar dados de proveniência pode reduzir o tempo de um *insight*". A proveniência de dados auxilia administradores rever conclusões ou chegar a novas descobertas, auxiliando na tomada de decisões.

O processo da inclusão de trilhas de auditoria em um sistema pode ser algo custoso, podendo exigir muitas refatorações ou até mesmo levando à redução de performance da solução. O protótipo de *framewok* para implantação de trilhas de auditoria desenvolvido neste trabalho tem como propósito oferecer uma interface de rápida e fácil implantação, com o menor impacto possível na arquitetura e performance ao software.

1.2 Importância

A seguir serão apresentadas as importâncias acadêmicas e sociais deste trabalho.

1.2.1 *Acadêmica*

Utilizar o conhecimento acadêmico adquirido na instituição e aprimorá-lo com o uso de tecnologias como Elastic Stack, Web API e *containers* Docker, para desenvolver um protótipo de *framework open source* capaz de auxiliar desenvolvedores na implantação de trilhas de auditoria e expandir a experiência na área.

1.2.2 *Social*

Contribuir com a comunidade de desenvolvedores, publicando um protótipo de *framework* de código aberto na plataforma de desenvolvimento NuGet, capaz de auxiliar a implantação de trilhas de auditoria em Web API's .NET Core.

2 OBJETIVO

Neste capítulo serão apresentados os objetivos gerais e específicos deste trabalho.

2.1 Objetivo Geral

Criar um protótipo de *framework* para auxiliar a comunidade de desenvolvedores a aplicar logs de auditoria em Web API's de forma ágil e padronizada, utilizando tecnologias capazes de suportar uma grande quantidade de dados.

2.2 Objetivos específicos

A partir do objetivo geral, pode-se definir os objetivos específicos, sendo estes:

a) Desenvolver um protótipo de *framework* para Web API's .NET CORE e publicá-lo no NuGet, com o intuito de auxiliar desenvolvedores a gerar trilhas de auditoria, oferecendo tanto agilidade e padronização na implantação, quanto compatibilidade com ferramentas emergentes e *open sources* capazes de gerenciar grandes quantidades de dados.

b) Criar uma documentação com objetivo de informar a comunidade de desenvolvedores como realizar a configuração e utilização do *framework*.

c) Desenvolver uma Web API .NET demonstrando a aplicabilidade do *framework*.

3 REVISÃO DE LITERATURA

Neste capítulo serão abordados os principais tópicos relacionados ao desenvolvimento do framework Audit.CRUD, tendo por objetivo realizar uma breve explicação sobre os conceitos utilizados para execução deste projeto, tanto como evidenciar a importância de trilhas de auditoria em sistemas de informação.

3.1 Sistemas Distribuídos

Coulouris et al. (2013) definem um sistema distribuído como aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens.

Segundo Tanenbaum (2007), sistema distribuído é um conjunto de computadores independentes que se apresentam aos seus usuários como um sistema único e coerente.

Com a disseminação da Internet, mudanças no desenvolvimento de aplicações foram adotadas. As aplicações distribuídas estão se estabelecendo quase como um padrão devido às necessidades atuais dos usuários (GEYER; VARGAS; AZEVEDO, 2001).

3.1.1 Cliente–Servidor

De acordo com Gomes (2018), a *Web* é um sistema baseado em Cliente-Servidor, onde existem partes distintas, ambas implementadas e implantadas independentemente de idioma ou tecnologia, contudo, que devem seguir a Interface Uniforme da Web.

Cliente-servidor é um tipo de aplicação distribuída que separa a interface de usuário do armazenamento de dados. Desta forma o componente servidor fica responsável pelo armazenamento dos dados e tratamento das requisições e o componente cliente pela parte da interface do usuário (FREITAS; SANTO; SILVA, 2019).

3.1.2 Web Service

Segundo o *W3C* (2004), *web services* fornecem um meio padrão de interoperar entre diferentes aplicativos de software, sendo executados em diversas plataformas e/ou estruturas.

Junior (2018, p. 28) define *Web Service* como uma tecnologia independente de plataforma, sendo possível, por exemplo, criá-lo em uma plataforma *Windows* e consumi-lo em uma plataforma *Linux*.

Web services são utilizados como forma de integração e comunicação de sistemas, de modo que um sistema possa realizar uma chamada para um serviço de outro sistema a fim de obter informações (LECHETA, 2015).

Para Moro, Dorneles e Rebonatto (2011), *Web Services* são componentes que permitem que aplicações enviem e recebam dados em formatos variados. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, como é o caso do formato *XML* ou *JSON*.

O desenvolvimento de *Web Services* permite disponibilizar formas de integrar sistemas distintos, modularizar serviços e capacitar a integração e consumo de informações. Com esta tecnologia, torna-se possível que aplicações diferentes interajam entre si e sistemas desenvolvidos em plataformas diferentes se tornem compatíveis (MORO; DORNELES; REBONATTO, 2011).

3.1.3 WebService Rest

O *REST* (*Representational State Transfer*) é uma arquitetura de *web services* proposta em uma tese de doutorado de Roy Fielding. Tal arquitetura utiliza o protocolo de comunicação *HTTP/HTTPS* e formatos de dados mais simples (*JSON*), quando comparado ao seu antecessor, o *SOAP*, que utiliza exclusivamente o formato de dados *XML* (MONTANHEIRO; CARVALHO; RODRIGUES, 2017).

Para Montanheiro, Carvalho e Rodrigues (2017), uma vez que o formato de dados é mais simplificado, os esforços para a comunicação são menores ao utilizar a arquitetura *REST*, resultando em melhor performance quando comparados à utilização de *SOAP*.

Em sua tese, Fielding (2000) propunha que os *web services* fossem implementados com base na utilização dos recursos oferecidos pelo protocolo *HTTP*, dessa forma o *REST* cria padrões para o protocolo *HTTP* e *URI* (Identificador Uniforme de Recursos), propondo novos métodos *HTTP* para que forneçam mais escalabilidade (GOMES; JANDL JUNIOR, 2009).

3.1.4 Propriedades de um serviço RestFul

Um serviço baseado em *REST* é conhecido como um serviço *RESTful*. Em geral, os serviços *RESTful* possuem as seguintes características e propriedades (CASTRO JUNIOR, 2017):

- **Representação:** Maneira de prover acesso aos recursos. Usualmente o acesso é provido através de *JSON*.
- **Mensagens:** O cliente comunica com o servidor através de mensagens usando protocolo *HTTP*. O cliente envia um pedido e o servidor devolve uma resposta.
- **URIs**: O *REST* requer que cada recurso seja identificável. A operação realizada neste recurso é determinada pelo verbo utilizado no *HTTP request*, como, por exemplo: *GET*, *POST* e *DELETE*. Isso permite chamar o mesmo recurso com diferentes verbos *HTTP* e realizar diferentes operações
- **Interface uniforme**: serviços *RESTful* devem possuir uma interface uniforme e isso é possível com o *HTTP/1.1*, em que são definidos os métodos de acordo com a tabela:

Quadro 1. Lista de métodos HTTP.

Método	Operação realizada no servidor
GET	Lê um recurso
PUT	Insere um novo recurso ou atualiza se já existir. Método idempotente. O cliente sabe a URL completa.
POST	Envia dados no corpo do pedido para processamento no servidor.
DELETE	Apaga um recurso.
OPTIONS	Lista as operações permitidas para um recurso.
HEAD	Responde apenas com o <Response Header>, sem o <Response B>

Fonte: CASTRO JUNIOR, 2017, p.9

- **Stateless:** Um serviço *RESTful* não mantém o estado da aplicação para o cliente, ou seja, um pedido não pode ser dependente de uma outra do passado. Cada pedido é tratado independentemente.
- **Caching:** Consiste em armazenar o resultado gerado e usá-lo futuramente em um pedido idêntico, em vez de gerá-lo novamente.

3.2 Programação orientada a objetos

Segundo Araújo (2017), programação orientada a objetos (*POO*) é um paradigma que viabiliza o desenvolvimento de aplicações fazendo uso de um conjunto de “coisas” que fazem parte de um contexto em estudo, normalmente agrupando-se em classes.

Para Carvalho (2016), o paradigma orientado a objetos tem como principal característica uma melhor e maior expressividade das necessidades do nosso dia a dia, pois possibilita criar unidades de código mais próximas da forma como pensamos e agimos.

De acordo com Aragão (2013), *POO* é uma metodologia que torna o processo de construção de sistemas menos custoso e mais produtivo, aumentando a eficiência no desenvolvimento, diminui o custo de construção de um software e facilita a manutenção do código fonte.

As principais vantagens da *POO* são a capacidade de reutilização de código, auxiliando na otimização da produtividade e no aumento da qualidade, e a manutenibilidade, ou seja, a facilidade na manutenção dos projetos (ARAÚJO, 2017).

3.3 Segurança da informação

Nesta seção serão descritos tópicos relacionados à segurança da informação, subdividido em concepções de informação, concepções da segurança da informação e pilares da segurança da informação.

3.3.1 Concepções de Informação

Vivemos na Era da Informação, em um mundo repleto de conexões e em constante mudança, onde cada vez mais a tecnologia se faz presente e a informação se torna um patrimônio muito valioso e necessário.

Lyra (2015, p. 16) define a Informação como um “conjunto de dados tratados e organizados de tal maneira que tragam algum significado ou sentido dentro de um dado contexto”.

Informação é o dado que possui utilidade ou aplicação para alguém. Dados são elementos que fornecem suporte lógico para a informação (LYRA, 2019).

A informação é um conjunto de dados devidamente tratados e analisados para produzir conhecimento relevante. Dados são a matéria-prima da informação, sozinhos não possuem um significado relevante (GUIMARÃES, 2019).

3.3.2 Concepções de segurança da informação

Devido ao avanço tecnológico, garantir que a informação esteja protegida é uma tarefa cada vez mais difícil de ser executada.

Fontes (2006, p. 11) conceitua segurança da informação como um "conjunto de políticas, normas, processos e atividades voltadas para a proteção da informação em empresas e organizações".

Segundo Sêmola (2003, p.43), podemos definir a segurança da informação como "uma área do conhecimento dedicada à proteção de ativos da informação, contra acessos não autorizados, alterações indevidas ou sua indisponibilidade". Nessa perspectiva, podemos perceber a importância da segurança da informação dentro de uma empresa ou organização. Prevenir incidentes que ponham os dados em risco é algo crucial nos dias atuais.

3.3.3 Pilares da segurança da informação

De acordo a norma ABNT NBR ISO/IEC 27002 (2005), os princípios básicos para garantir a segurança da informação são:

- **Autenticidade**: Garante que a informação que é trafegada tenha um proprietário e que este proprietário não foi alterado.
- **Confidencialidade**: Garante que a informação não é acessada por alguém não autorizado.
- **Disponibilidade**: Garante que a informação estará disponível no momento que for necessária.
- **Integridade:** Garante que a informação trafegada é original, não podendo ter nenhum dado alterado.

Segundo Laureano (2005), alguns autores defendem que, para uma informação ser considerada segura, o sistema que a administra ainda deve respeitar os seguintes critérios:

- **Autenticidade**: Garante que a informação ou o usuário da mesma é autêntico.

- **Não repúdio**: Não é possível negar o envio ou recepção de uma informação ou dado.
- **Legalidade**: Garante a legalidade (jurídica) da informação; a aderência de um sistema à legislação.
- **Privacidade**: Uma informação privada pode ser vista / lida / alterada somente pelo seu dono.
- **Auditoria**: Rastreabilidade dos diversos passos de um negócio ou processo, identificando os participantes, os locais e horários de cada etapa.

3.4 Proveniência de Dados

Nesta seção será descrito tópicos relacionados à proveniência de dados, subdividido em conceito, abordagem e nível de captura e modelos.

3.4.1 Conceito

O termo "proveniência" trata-se da origem ou procedência dos dados, ou seja, um registro histórico de derivação de dados, possibilitando a reprodução, interpretação e diagnóstico de problemas (LIM; LU; CHEBOTKO; FOTOUHI, 2010).

De acordo com PAULA (2012, p. 1), para garantir a proveniência de dados "se faz necessário guardar tanto a origem dos dados utilizados, quanto os processos que transformaram esses dados no produto final".

A proveniência de dados auxilia na tomada de decisões porque visa descrever os acontecimentos e insumos utilizados na geração de uma determinada informação, é importante para garantir a origem dos dados como para avaliar a sua precisão (ANDRADE, 2013).

3.4.2 Abordagem e nível de captura

Segundo Tan (2004) existem duas abordagens para obter a proveniência de dados:

- Abordagem Preguiçosa (*Lazy*): Nesta abordagem a obtenção da proveniência é executada somente no momento que é solicitada. Porém, torna o rastreamento mais difícil, já que a informação original pode não estar mais disponível.

- Abordagem Ansiosa (*Eager*): Nesta abordagem a proveniência é obtida durante a geração da informação e é armazenada para permitir futuras consultas, nesta abordagem a proveniência estaria prontamente disponível para quando o usuário solicitasse.

De acordo Freire et al. (2008) quando a proveniência é realizada de forma automática pode-se dividir o nível em que a captura é feita em:

- *Workflow*: Captura os dados dos diferentes processos executados.
- Atividade: Podem ser criados programas específicos para monitorar a execução de um determinado processo e capturar os dados de proveniência ou cada processo executado é alterado para capturar esses dados.
- Sistema Operacional: Utiliza os dados fornecidos pelo próprio sistema operacional como insumo para a proveniência.

3.4.3 Modelos

Vários modelos de proveniências de dados são encontrados hoje na literatura, porém todos têm como objetivo a gravação dos dados para proveniência. A seguir será feita uma breve revisão de alguns modelos.

3.4.3.1 Modelo W7

Modelo baseado na ontologia de Bunge, tem como objetivo descrever as propriedades de um objeto de carácter geral. Este modelo estruturou a proveniência de uma peça de dado através da resposta a sete perguntas, conforme quadro a baixo (PAULA, 2012).

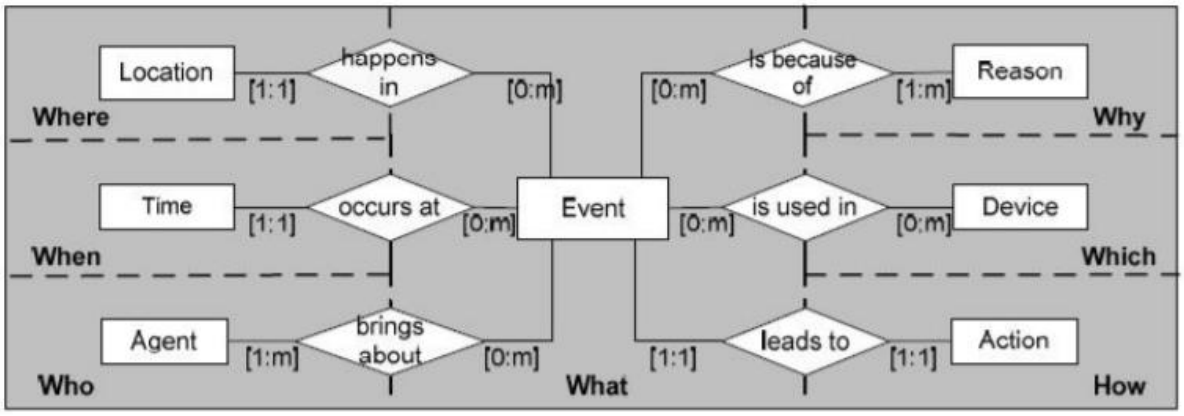
Quadro 2. Modelo W7

What?	Who?	When?	Where?	How?	Which?	Why?
O Que?	Quem?	Quando?	Onde?	Como?	Qual?	Por quê?

Fonte: Adaptado de PAULA (2012).

Na figura abaixo é possível perceber que todas as perguntas são interligadas com o *Event*.

Figura 1. Visão geral do modelo W7



Fonte: PAULA (2012, p. 16).

O lado positivo deste modelo é que este gera a maioria das respostas necessárias para uma boa proveniência, sua desvantagem é a complexidade da representação gráfica (ANDRADE, 2013).

3.4.3.2 *Provenance Vocabulary*

Este modelo foi descrito por Hartig e Zhao, é voltado para o problema da proveniência de dados publicados na *web*. Sua principal característica é fornecer classes e propriedades para que publicadores de dados para web possam armazenar metadados com informações úteis à proveniência (PAULA, 2012).

Destacam-se as três classes que dão estrutura principal a esse modelo (ANDRADE, 2013):

- Artefatos: Podem ser tanto entradas como produtos. Possuem três subclasses, *DataItem* (dado do artefato), *File* (conjunto de *DataItem*) e *CreationGuideline* (regras para criação do artefato).
- Execução: é a representação da execução completa de um processo. É composta de duas subclasses, *DataCreation* (processo de criação do dado) e *DataAccess* (processo pelo qual o dado foi acessado).
- Atores: representa uma entidade ativa que pode afetar a execução das ações ou dos processos. Possui duas subclasses, *HumanActor* (pessoa ou organização) e *NonHumamActor* (entidade não humana).

3.4.3.2 *Open Provenance Model*

É um modelo de dados *open source* do ponto de vista da interoperabilidade, que recebe contribuições da sua comunidade. Levando em conta que os objetos podem ser alterados por qualquer tipo de intervenção, foram definidos três tipos de nós (PAULA, 2012):

- Artefato (*Artifact*): Corresponde a um determinado objeto em um determinado estado no tempo. Pode ser usado ou gerado por um Processo;
- Processo (*Process*): Corresponde a uma ou mais ações executadas consumindo e/ou gerando artefatos.
- Agent (*Agent*): Corresponde a uma entidade, em um determinado contexto agindo como ativador do processo.

No modelo *Open Provenance Model*, as relações casuais entre os nós são chamadas de dependências. Para isso são definidos cinco tipos de dependências conforme abaixo (PAULA, 2012):

- *Used*: Indica que um determinado Artefato foi usado por um Processo;
- *WasGeneratedBy*: Indica que um Artefato foi gerado por um determinado Processo;
- *WasControlledBy*: Indica que um Processo foi controlado por um Agente;
- *WasTriggeredBy*: Indica que um Processo P1 foi acionado por um outro Processo P2.
- *WasDerivedFrom*: Indica que um Artefato A1 é derivado de um outro Artefato A2.

3.4.4 Considerações

De acordo com Paula (2012, p. 34), “os modelos W7, Open Provenance Model foram projetados para a descrição de sequência de passos na criação ou modificação de um dado”.

Apesar do modelo de dados *Provenance Vocabulary* ser voltado para dados publicados na *web*, e este trabalho se trata da criação de um *framework* direcionado para Web API’s, foi optado por utilizar o modelo W7 por ser mais genérico e poder abranger mais casos, independentemente da complexidade da representação gráfica deste modelo.

3.5 Trilhas de auditoria

Nesta seção serão descritos tópicos relacionados às trilhas de auditoria, subdividindo-se em logs, processo de auditoria e legislação e normas.

3.5.1 Logs

Arimatéa (2019, p.28) define log como “uma expressão utilizada para descrever o processo de registro de eventos relevantes num sistema computacional. Esse registro pode ser utilizado para restabelecer o estado original de um sistema ou para que um administrador conheça o seu comportamento no passado”.

Logs são mensagens utilizadas para registrar um acontecimento na execução de um sistema. O principal objetivo do uso dos logs é registrar informações detalhadas do comportamento do software para identificar problemas e suas causas (VIEIRA, 2018).

O propósito das mensagens de log é informar terceiros de informações detalhadas sobre a execução do programa, portanto, é crucial que as mensagens sejam descritas de forma clara e objetiva. Vieira (2018) ressalta que para escrever logs eficazes é necessário responder a regra dos 5W’s:

Quadro 3. Regra dos 5W’s

WHO	WHAT	WHEN	WHERE	WHY
Quem	O que	Quando	Onde	Porque

Fonte: Adaptado de VIEIRA (2018).

Os registros de log devem ser capazes de reconstruir um cenário de ação, com a identificação de quem fez a ação, quando e onde foi feita a ação. Registros de logs confiáveis permitem a identificação de violações e são essenciais em sistemas de informação (LIBERALI, 2012).

3.5.2 O processo de auditoria

A trilha de auditoria (também chamada de logs de auditoria), é o registro de atividades dos usuários e administradores dentro de um sistema. Os logs de auditoria auxiliam empresas e

organizações a manter um controle histórico das alterações realizadas na informação dentro de um sistema (LIBERALI, 2012).

A principal finalidade de um serviço de auditoria é oferecer armazenamento seguro e permanente dos registros de log, para que seja possível detectar quando uma falha de segurança ocorreu, auxiliando na garantia da integridade das informações contidas no sistema (RORATTO; DIAS, 2014).

Auditoria é considerada um dos controles básicos de segurança da informação, de acordo com Fontes (2016) tudo que for realizado no ambiente computacional deve ser obrigatoriamente registrado para facilitar uma decorrente auditoria. O autor complementa "qualquer uso da informação (leitura, alteração, inclusão ou remoção) deve ser registrado para possibilitar que futuras investigações identifiquem o que foi feito" (FONTES, 2016).

O log de auditoria também é um importante artefato no âmbito da computação forense, capaz de oferecer segurança e confiabilidade aos sistemas computacionais. Nos logs de auditoria podem ser registradas todas as atividades dos usuários e modificações de dados. Portanto, uma trilha de auditoria é capaz de possibilitar a análise e diagnóstico de incidentes de segurança, permitindo rastreabilidade da causa raiz de um problema, ou ataque, no sistema (CÂMARA, 2014).

A auditabilidade é essencial no processo de gestão da segurança da informação. Os documentos de log são uma importante fonte para acompanhar se as propriedades básicas da segurança da informação estão em ameaça de violação, pois auxiliam na redução das perdas de dados e na identificação de vulnerabilidades e riscos (ROCHA, 2018).

3.5.3 Legislação e Normas

É importante que as modificações nos sistemas e recursos de processamento da informação sejam controladas, uma vez que o controle insuficiente é uma causa comum de falhas se segurança ou de sistema (ABNT, 2001).

Tratando do controle de mudanças operacionais, a NBR ISO/IEC 17799 da ABNT (2001, p. 18) ressalta que:

> Convém que programas que estejam em produção sejam submetidos a um controle específico de modificações. Quando da mudança de programas, convém que seja realizada e mantida uma trilha de auditoria (registro) com todas as informações relevantes. Modificações no ambiente operacional podem causar impacto em aplicações.

Todas as atividades de uma organização que utiliza a informação devem ter medidas de controle adequadas para proteger essas informações. A Lei n° 12.846, de 1° de agosto de 2013 trata da responsabilização administrativa e civil de pessoas jurídicas (empresas) pela prática de atos de corrupção contra a administração pública, nacional ou estrangeira. No artigo 5° é citado a importância das trilhas de auditoria.

> Se um órgão de investigação solicitar da organização a trilha de auditoria de acesso a sistemas ou uso de ferramentas tipo correio eletrônico, e se esta organização não tiver os registros de auditoria gravados e guardados, com as identificações individuais e outros controles, com certeza poderá ser interpretada pelo judiciário como um empecilho para as investigações. Não ter cópias de segurança de informações e outros controles, recomendados pela Norma NBR ISO/IEC27002, poderá custar uma decisão judicial não favorável à organização, e ter seu nome enquadrado na Lista Negra das organizações que praticam corrupção.

Cabe à trilha de auditoria registrar exceções e outros eventos de segurança relevantes por um período de tempo acordado para auxiliar em futuras investigações e na monitoração do controle de acessos (ABNT, 2001). É recomendado que os registros de log de auditoria incluam:

- Identificação dos usuários;
- Datas e horários de entrada (log-on) e saída (log-off) no sistema;
- Identidade do terminal ou, quando possível, a sua localização;
- Registros das tentativas de acesso ao sistema aceitas e rejeitadas;
- Registros das tentativas de acesso a outros recursos e dados aceitas e rejeitadas.

3.6 Banco de dados

Nesta seção serão descritos tópicos relacionados a bancos de dados, subdividindo-se em conceitos de banco de dados, modelo relacional e não relacional e modelos NoSQL.

3.6.1 Conceitos

Heuser (2019, p. 3) conceitua banco de dados como “uma coleção de dados organizados, inter-relacionados e que atendem a uma aplicação ou família de aplicações”.

Date (2004) define um sistema de banco de dados como “um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários, busquem e atualizam essas informações quando as solicitar”.

Por meio do banco de dados é possível coletar informações e utilizá-las para que sejam realizadas análises, monitorar padrões, tendências, oportunidades, fraquezas e ameaças, possibilitando melhores tomadas de decisões a curto, médio e longo prazo (TEIXEIRA NETO, 2019).

3.6.2 Banco de dados relacional

O banco de dados relacional é fundamentado no modelo tabela, linhas e colunas. Sua definição teórica é baseada na teoria de conjuntos, ramo da matemática que estuda coleções de elementos (OLIVEIRA, 2014).

De acordo com a Microsoft (2020) “Os bancos de dados relacionais têm sido uma tecnologia predominante há décadas. Eles são maduros, comprovados e amplamente implementados”.

O banco de dados relacional contém algumas desvantagens; Custo, os valores são bem elevados; Profissionais caros, pois trata-se de um tipo de software sofisticado que requer desenvolvimento e desenho; Adaptação complicada, a construção é complexa e existe a chance de existirem problemas em possíveis adaptações (NASCIMENTO, 2018).

3.6.3 Banco de dados não relacional

Com a popularização da internet, o volume de dados gerados por aplicações web expandiu. O uso da tecnologia relacional para processamento de dados se tornou gradualmente mais complicada, e sua manutenção cada vez mais cara.

Com objetivo de evitar o alto custo dos sistemas de banco de dados relacionais, iniciou-se, ao longo do tempo, o desenvolvimento de bancos distribuídos capazes de gerenciar dados semiestruturados provenientes de diversas origens, de forma mais barata e menos complexa (OLIVEIRA, 2014).

Em 2009, os bancos não-relacionais se tornaram muito mais populares. O termo *NoSQL* (*Not Only SQL*), ou Não Somente *SQL* tomou força, essa nomenclatura deve-se a tentativa de descrever a popularização do uso de banco de dados não relacionais, uma provocação aos bancos relacionais mais populares da época como *MySQL*, *Ms SQL*,

PostgreSQL. Desde então, os bancos não relacionais passaram a ser conhecidos como *NoSQL*. (MELO, 2020).

Um banco de dados não relacional não usa o esquema de tabela, linhas e colunas como nos bancos relacionais. Em vez disso, usa um modelo de armazenamento otimizado para requisitos específicos do tipo de dados que será armazenado (MICROSOFT, 2018).

3.6.4 Modelos NoSQL

Os sistemas de gerenciamento de bancos de dados NoSQL tipicamente são classificados em quatro modelos: chave-valor, orientado a colunas, orientado a documentos e orientado a grafos.

3.5.4.1 Chave-valor (key-value)

Este modelo de banco de dados consiste em um conjunto de chaves, que são associadas a um único valor, possibilitando que os objetos possam ser pesquisados a partir de suas chaves (LÓSCIO, 2011).

Quadro 4. Exemplo de modelo chave-valor

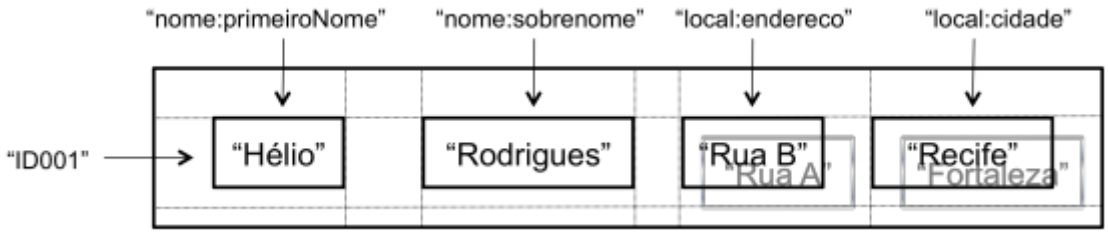
Nome	Camila
Sobrenome	Melo Machado
Idade	25
Sexo	Feminino
Telefone	99 9 9999-9999

Fonte: Adaptado de LÓSCIO (2011, p.6).

3.5.4.2 Orientado a colunas

Neste modelo os dados são indexados por linha, coluna e *timestamp*, onde as linhas e colunas são identificadas por chaves e o *timestamp* permite distinguir várias versões de um mesmo dado (LÓSCIO, 2011).

Figura 2. Exemplo de modelo orientado a colunas



Fonte: LÓSCIO (2011, p.6).

3.5.4.3 Orientado a documentos

Neste modelo os dados são coleções de documentos com atributos e valores, onde um atributo pode ser multivalorado (MELO, 2020).

No modelo orientado a documentos temos um conjunto de documentos e em cada documento temos um conjunto de campos (chaves) e o valor deste campo (LÓSCIO, 2011).

Quadro 5. Exemplo de modelo orientado a documentos

ID: P001
Assunto: "Monografia de Ciência da Computação"
Data: "03/11/2011"
Tags: ["NoSQL","BigData","Elasticsarch"]

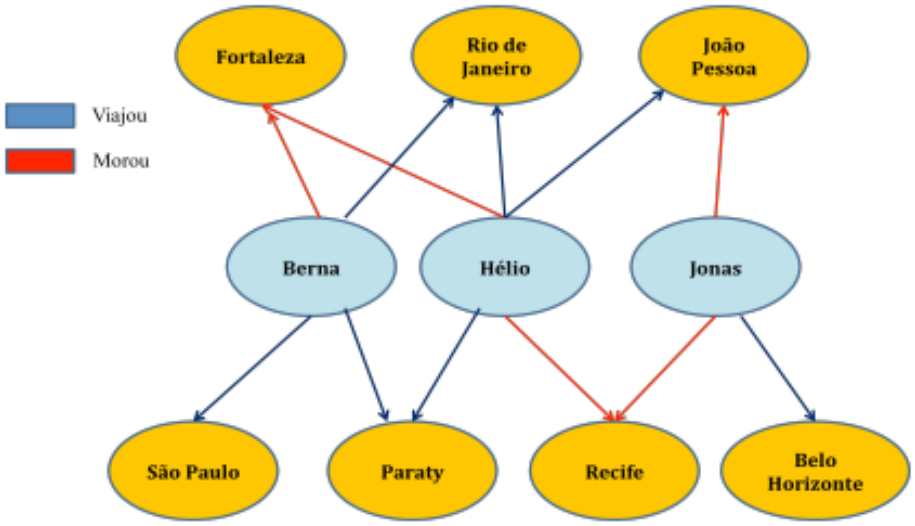
Fonte: Adaptado de LÓSCIO (2011, p.8).

3.5.4.4 Orientado a grafos

Segundo Lóscio (2011, p.8) "O modelo orientado a grafos possui três componentes básicos: os nós (são os vértices do grafo), os relacionamentos (são as arestas) e as propriedades (ou atributos) dos nós e relacionamentos".

O modelo de grafos é baseado na teoria dos grafos, ramo da matemática. Seu objetivo é representar os dados e/ou o esquema dos dados como grafos dirigidos, ou com estruturas que generalizem a noção de grafos (MELO, 2020).

Figura 3. Exemplo de modelo orientado a grafos



Fonte: LÓSCIO (2011, p.8).

3.7 Big Data

Nesta seção serão descritos tópicos relacionados a Big Data, subdividindo-se em conceitos e características.

3.7.1 Conceitos

Big Data é um termo que descreve um imenso volume de dados complexos, estruturados ou não, que podem ser utilizados para a obtenção de *insights* para tomada de decisões (MELO, 2020).

Silva (2016, p.74) define *Big Data* como "um termo usado para descrever conjuntos de dados, cuja captura, armazenamento, distribuição e análise requerem métodos e tecnologias avançadas, devido seu volume, velocidade e heterogeneidade".

O *Big Data* tem como objetivo extrair valor de um grande volume de dados, tornando possível a análise de grandes massas de dados para uma alta velocidade de captura (SILVA, 2016).

3.7.2 Características

Conforme Melo (2020) elucida, o *Big Data* é composto por 5 características, conhecidas como os 5 V's. São elas:

- Volume: grande quantidade de dados gerada a cada segundo.
- Velocidade: velocidade que os dados são criados.
- Variedade: os dados podem ser altamente estruturados, semiestruturados e totalmente desestruturados.
- Veracidade: garantir que a informação é verdadeira.
- Valor: gerar valor com o acesso a grande massa de dados.

4 FERRAMENTAS DO PROJETO

Nesta seção serão descritas as principais ferramentas a serem utilizadas no desenvolvimento do projeto.

4.1 C#

C# é uma linguagem de programação orientada a objetos desenvolvida em 2000 por uma equipe liderada por Anders Hejlsberg, da Microsoft (MKHITARYAN, 2017).

Segundo a Microsoft (2015), C# é uma linguagem orientada a objetos, fortemente tipada, que permite que os desenvolvedores criarem uma variedade de aplicativos executados no .NET.

C# é uma linguagem de programação popular sendo, segundo o site TIOBE (Programming Community Index Definition), uma das 6 linguagens mais utilizadas no mundo (TIOBE, 2019).

4.1.1 *IDE: Visual Studio*

Segundo Santos (2019), o IDE (Ambiente Integrado de Desenvolvimento) é um programa de computador, geralmente utilizado para aumentar a produtividade dos desenvolvedores de software, bem como a qualidade desses produtos.

O Visual Studio é um IDE que contém recursos para *Android*, *iOS*, *Windows*, *Web* e nuvem, oferece ferramentas de desenvolvimento para usar os padrões emergentes de design da Web em uma solução ASP.NET (MICROSOFT, 2017).

4.1.2 Framework: .NET CORE

Leite (2006) define framework como uma técnica da Orientação a Objetos, voltada para a reutilização que se beneficia de três características das linguagens de programação orientada a objetos: abstração, polimorfismo e herança.

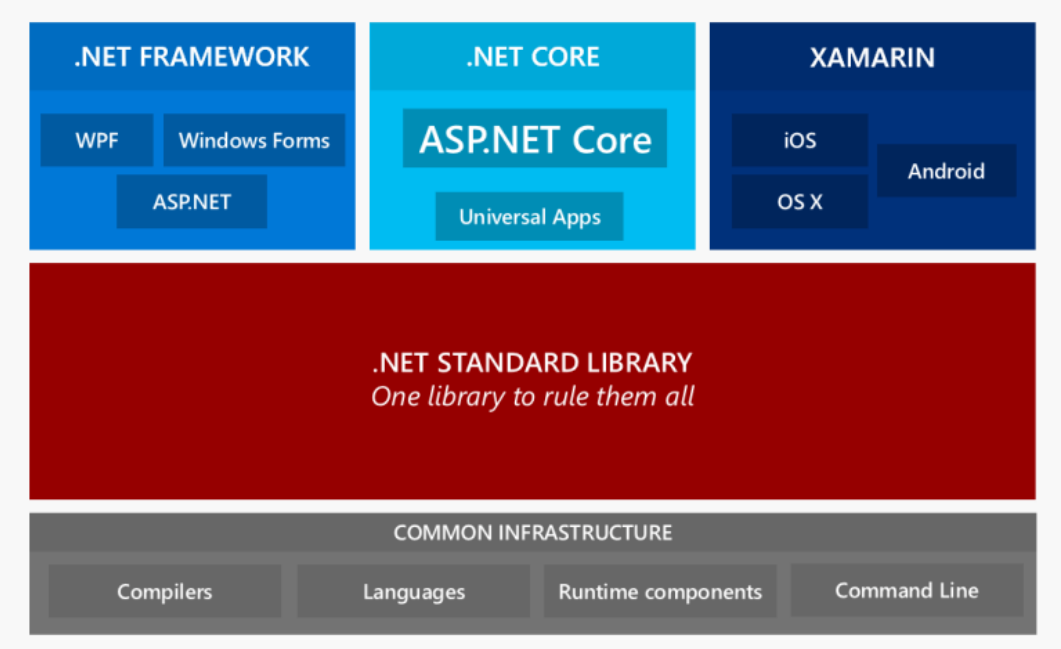
O .NET Core é uma plataforma de desenvolvimento de código aberto, mantida pela Microsoft e pela comunidade .NET no GitHub. Sua implementação é multiplataforma, podendo ser executado em Windows, Linux e macOS. Foi projetado para lidar com cargas de trabalho de servidor e na nuvem em escala (MICROSOFT, 2018).

4.1.3 Framework: .NET Standard

Como o passar dos anos, a Microsoft lança novos *frameworks* para a plataforma .NET usando como linguagem base o C#. Para que o .NET continue evoluindo rapidamente foi necessário evitar a fragmentação entre tantas versões de *frameworks* e proporcionar a escrita de um código único para qualquer plataforma. A solução foi implementar um padrão para que todas as versões dos frameworks suportem o mesmo conjunto de APIs (PIRES, 2017).

Pires (2017) define o .NET Standard como "uma espécie de interface que define as APIs que cada versão do .NET irá oferecer suporte". O código do .NET Standard não possui implementação de comportamento, apenas a declaração das classes e métodos.

Figura 4. Exemplificação lógica dos frameworks .NET



Fonte: PIRES (2017).

O .NET Standard é uma especificação formal das APIs do .NET que estão disponíveis em várias versões do .NET. Seu principal objetivo é estabelecer uma maior uniformidade no ecossistema .NET (MICROSOFT, 2020).

4.2 NuGet

NuGet é uma plataforma de desenvolvimento para .NET, a qual a Microsoft presta suporte, para que desenvolvedores possam criar, compartilhar e consumir código útil (MICROSOFT, 2019).

Nascimento (2020) define o NuGet como “um gerenciador de dependências para a plataforma .NET”. Tem como objetivo, definir como os pacotes dessa plataforma são criados, publicados e consumidos.

Para que um pacote seja compatível, é necessário que contenha *assemblies* criados para pelo menos um .NET *framework*. Os desenvolvedores podem criar pacotes específicos a uma estrutura, ou podem dar suporte a uma grande variedade de destinos. Para maximizar a compatibilidade de um pacote, os desenvolvedores direcionam para .NET Standard, que todos os projetos do .NET e .NET Core podem consumir (MICROSOFT, 2019).

O NuGet é uma plataforma de código aberto, o que significa que podemos contribuir com o seu desenvolvimento ou fornecer pacotes que podem ser usados em projetos para ajudar a comunidade desenvolvedores.

4.3 Git

Git é um sistema de código aberto utilizado pela grande maioria dos desenvolvedores para gerenciar projetos e controlar o versionamento de seus códigos. Com ele é possível criar um histórico de alterações no código do projeto e facilmente retornar para aquele (FERNANDES, 2017).

4.3.1 Pontos na história

Uma das principais funcionalidades do Git, é criar pontos de história em um projeto, esses pontos são chamados de *commits*, e são formados por um conjunto de alterações em um ou mais arquivos e ligados a um descritivo que resume as alterações nesse ponto (FERNANDES, 2017).

Cada vez que é executado um *commit*, está sendo gravado uma imagem do seu projeto onde é possível reverter ou comparar mais tarde (CHACON; STRAUB, 2014).

4.3.2 Ramificações

Chacon e Straub (2014, p. 71) definem *branch* como “um ponteiro móvel para uma sequência de *commits*”.

As ramificações ou *branchs*, são formas de desenvolver isoladamente, recebendo *commits* de diferentes fontes. Usando *branchs* é possível dividir a execução do projeto, e depois que finalizar combinar sua *branch* com outras (FERNANDES, 2017).

4.3.3 GitHub

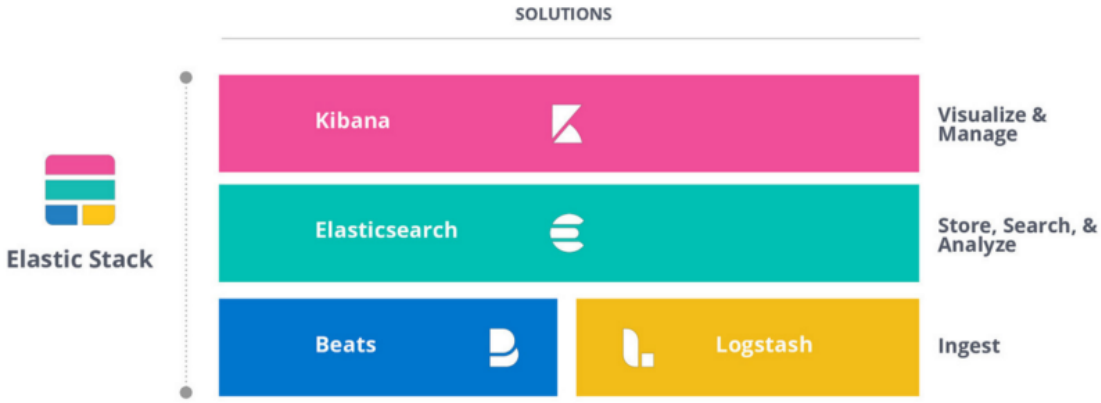
O GitHub é um serviço online de hospedagem de repositórios Git, com ele podemos gerenciar projetos, criar ramificações e compartilhar projetos com outros desenvolvedores (FERNANDES, 2017).

4.4 Elastic Stack

O Elastic Stack, também conhecido como “ELK”, é um conjunto de três projetos *open source*: **E**lasticsearch, **L**ogstash e **K**ibana. O Elasticsearch é um mecanismo de busca e análise dados. O Logstash é um *pipeline* de processamento de dados. O Kibana é um visualizador de dados (ELASTIC, 2020).

A arquitetura do Elastic Stack é baseada em “camadas”. A primeira camada é responsável pela ingestão de dados, através das soluções Logstash e Beats. A segunda camada é responsável pelo armazenamento, pesquisa e análise dos dados, através do Elasticsearch. A terceira camada é responsável visualização e gerenciamento dos dados, através do Kibana (RAMOS, 2020). Neste trabalho, será usado somente a segunda e a terceira camada.

Figura 5. Arquitetura Elastic Stack



Fonte: Adaptado de Ramos (2020).

O Elastic Stack foi desenvolvido para resolver problemas de análise, armazenamento e coleta de dados, contribuindo para que empresas transformem em tempo real seus dados em informações relevantes (ROCHA, 2018).

4.4.1 Elasticsearch

O Elasticsearch é considerado o coração do ELK, ele é um mecanismo de busca e análise de dados distribuído, consegue suportar grandes quantidades de dados e analisa-los em tempo real. Foi desenvolvido em Apache Lucene e lançado pela primeira vez em 2010, é reconhecido por sua velocidade de busca e indexação de dados (ELASTIC, 2020).

4.4.1.1 Conceitos básicos

O Elasticsearch, também conhecido como ES, pode ser considerado uma versão distribuída do *framework* Lucene. Devido essa relação, alguns conceitos básicos do ES são derivados do Lucene (SANTANA, 2015):

- Índice (*index*): define o endereço para acesso às informações armazenadas no ES.
- Tipo (*type*): recurso utilizado para nomear conjuntos de documentos armazenados em um índice.
- Documento: é um texto plano, organizado em campos delimitados com chaves e vírgulas, de acordo com o padrão JSON.
- Campo (*field*): unidade mínima de informação armazenada em um documento.
- Mapeamento (*mapping*): define a estrutura de um documento, contendo campos e a maneira como cada um deve ser armazenado e recuperado.
- Query DSL: é a linguagem de busca de dados.

Abaixo temos uma comparação entre os conceitos Lucene e os conceitos de um banco de dados relacional.

Quadro 6. Comparação de conceitos Lucene e BDRs

Lucene	**Banco de dados relacional**
Índice (index)	Esquema
Type	Tabela
Documento (JSON)	Linha
Campo (Field)	Coluna
Mapeamento (Mapping)	Estrutura da tabela
Query DSL	SQL

Fonte: Adaptado de SANTANA (2015).

4.4.1.2 Banco de dados não relacional

O ES é categorizado como um banco de dados não relacional, multiplataforma e facilmente integrável, trabalha com mecanismos de busca e análise dados, se comunicando através de uma *API RESTFull* (MACEDO, 2020).

O Elasticsearch é na sua essência um motor de buscas, mas que pode ser utilizado como uma ferramenta de armazenamento *NoSQL* orientado a documentos. Como a maioria dos sistemas de bancos de dados *NoSQL*, foi modelado para ser distribuído, escalável e realizar pesquisas em tempo real (RODRIGUES, 2016).

De acordo o site DB-Engines, o Elasticsearch ocupa a oitava posição em popularidade de sistemas de gerenciamento de banco de dados (DB-ENGINES, 2020).

4.4.1.2 Big Data

O Elasticsearch é um banco de dados orientado a documentos, constantemente listado com uma das ferramentas do Big Data, seu objetivo principal é apoiar o desenvolvimento de aplicações centradas em texto (SANTANA, 2015).

A arquitetura do ES foi projetada para trabalhar em *cluster* e suportar grandes quantidades de dados. Segundo Fernández (2009, p. 1) um *cluster* é “um conjunto de computadores interligados, instalados e programados de tal forma que os seus usuários tenham impressão de estar usando um recurso computacional único”.

Com o Elasticsearch é possível armazenar, buscar e analisar grandes quantidades de dados, devido sua velocidade, escalabilidade e flexibilidade se tornam uma solução poderosa

para observabilidade de sistemas e segurança (detecção e prevenção de ameaças) (DAVIS, 2020).

4.4.2 *Kibana*

O Kibana é uma interface de usuário, para visualizar dados do Elasticsearch e navegar no Elastic Stack. Fornece recursos de busca e visualização de dados indexados no Elasticsearch. Com ele é possível monitorar, gerenciar e proteger um *cluster* do Elastic Stack. Foi desenvolvido em 2013 pela comunidade do Elasticsearch, cresceu e se tornou a janela de acesso ao ELK, oferecendo um portal para usuários e empresas (ELASTIC, 2020).

4.5 Containers Docker

Pinto (2019, p. 6) define *containers* como “ambientes isolados dentro de um sistema operacional”.

Para Morais (2020, p. 78), *containers* são “uma tecnologia de empacotamento de código, bibliotecas e dependências em um único objeto”.

O principal objetivo de um *container* é prover micro serviços, sem a necessidade que todo um sistema operacional seja instalado para que este serviço funcione corretamente, trazendo maior agilidade ao processo (PINTO, 2020).

Atualmente existem algumas tecnologias para gerenciamento de *containers*, hoje em dia, o *Docker* é a mais consolidada, pois cria imagens (cópias completas dos *containers*) e as disponibiliza para *download* de forma simples e rápida (MORAIS, 2020).

Docker é uma tecnologia *open source*, lançada em 2013. Essa tecnologia de gerenciamento de containers trouxe portabilidade, flexibilidade e facilidade para implantar os mais diversos micro serviços (DOCKER, 2020).

Containers e máquinas virtuais são semelhantes, a principal diferença entre essas tecnologias é que; uma máquina virtual possui seu próprio *kernel*, binários e bibliotecas, já o *container*, compartilha o *kernel*, binário e bibliotecas do sistema operacional sob qual ele está sendo executado (PINTO, 2020).

Entre os benefícios da utilização de *containers* Docker, podemos citar a leveza, como uma das características mais fortes. Os *containers* compartilham o *kernel* do sistema

operacional da máquina, não necessitando um sistema operacional por aplicativo, dessa forma, aumenta a eficiência e reduz custos de implantação e licenciamento (DOCKER, 2020).

5 METODOLOGIA

A metodologia científica auxilia na compreensão das normas de pesquisa, colaborando para que os conhecimentos expandam por meio da coerência e coesão (CIECHOWICZ; CIECHOWICZ, 2019).

A pesquisa documental, ou de fonte primária, está relacionada à coleta de dados em documentos, sejam por escrito ou não, podendo ser escritos quando ou após o fato. Alguns exemplos de fontes de pesquisa documental são; documentos oficiais, como leis e ofícios, instituições de ordem privada, como registros e comunicados (MARCONI; LAKATOS, 2003).

A pesquisa bibliográfica, ou de fontes secundárias, está relacionada à toda bibliografia pública em relação ao tema de estudo. Alguns exemplos de fontes de pesquisa bibliográfica são; impressa escrita, como jornais e revistas, publicações, como livros, teses, monografias (MARCONI; LAKATOS, 2003).

Este trabalho foi baseado em sua maioria de pesquisa bibliográfica, como em artigos, livros, citações e teses de estudiosos do assunto, assim como pesquisa documental, como normas e legislação.

Com base na literatura, foi utilizada a metodologia de estudo de caso, que tem como objetivo identificar os elementos necessários para propor um framework para apoiar desenvolvedores a aplicarem trilhas de auditoria em *softwares*.

6 CRONOGRAMA

O seguinte cronograma foi utilizado para o desenvolvimento deste trabalho.

Quadro 7. Cronograma

Atividade	**Out**	**Nov**	**Dez**
Protótipo do resumo; Procurar referencial Teórico do Trabalho			
Criar referencial teórico; Ferramentas utilizadas			
Objetivo Geral e Específicos; Justificativa; Metodologia			
Pré-Projeto; Implementação do protótipo			
Pré-Projeto; Implementação do protótipo			
Pré-Projeto; Implementação do protótipo; Resultados			
Ajustes necessários; Entrega à coordenação			
Treinar apresentação			
Defesa TCC			

Fonte: Arquivo do Autor

7 PROJETO

Devido ao tempo curto para entregas de produtos de valor, as empresas de desenvolvimento de *softwares* ocasionalmente negligenciam as boas práticas de segurança no desenvolvimento, e apesar da inclusão de trilhas de auditoria ser um dos pilares da segurança da informação, muitas vezes é deixada de lado.

O uso de *frameworks* possibilita que desenvolvedores utilizem interfaces previamente prontas e apenas as configure em seu *software*, trazendo mais agilidade no desenvolvimento, fazendo que as entregas se tornem mais rápidas e código fique mais padronizado.

Com o objetivo de resolver a escassez de trilhas de auditoria em *softwares*, e auxiliar a comunidade de desenvolvedores aplicarem esse pilar tão importante para a segurança da informação, foi desenvolvido um protótipo de *framework*, utilizando tecnologias emergentes no mercado e *open sources*, visando trazer mais praticidade, agilidade e baixo custo na aplicação de trilhas de auditoria.

7.1 Diagramas UML

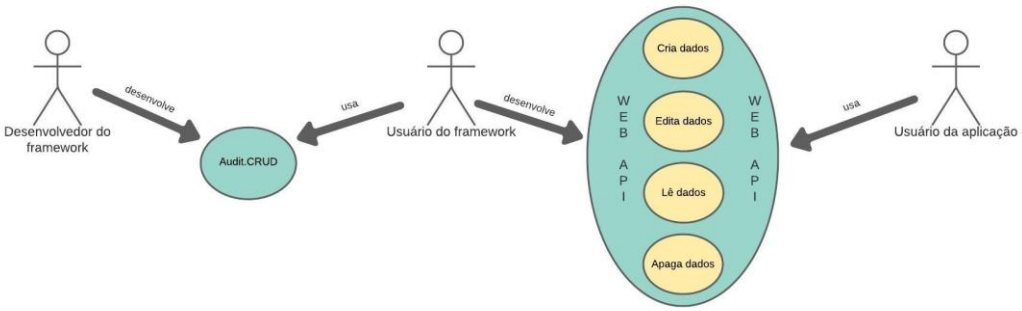
UML (Unified Modeling Language) é uma linguagem gráfica usada para visualizar, especificar e construir artefatos de sistemas de software complexos. Tem como objetivo fornecer um método padrão para a criação de projetos de arquitetura de sistema (BOOCH; RUMBAUGH; JACOBSON, 2006).

7.1.1 Diagrama de caso de uso

Os diagramas de caso de uso são utilizados para capturar os requisitos funcionais do software, ou seja, as funções que são identificadas como necessárias para que os interajam com o sistema. De forma resumida, o caso de uso expressa e registra o comportamento esperado da função do software (GUEDES, 2018).

O diagrama a seguir, expressa de forma simplificada o principal caso de uso do *framework* Audit.CRUD, onde um desenvolvedor cria um *framework*, para outro desenvolvedor aplicá-lo em uma Web API em pontos característicos de um CRUD (*Create, Read, Update and Delete*).

Figura 6. Diagrama de caso de uso



Fonte: Autoria própria

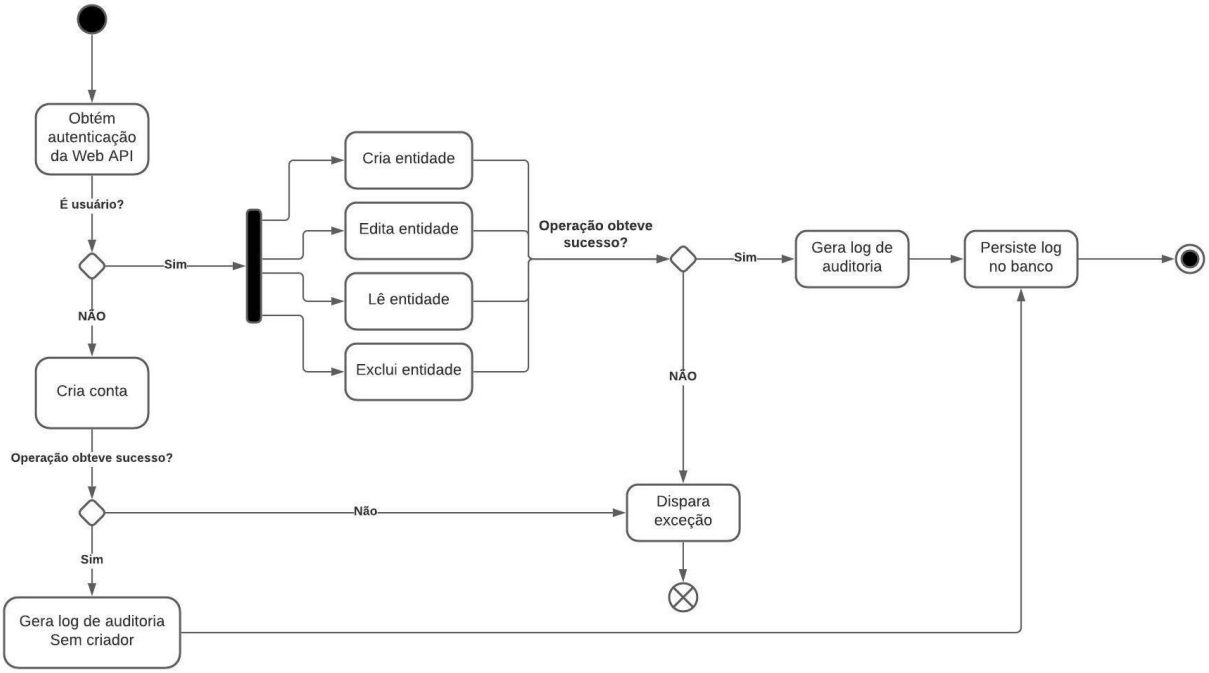
7.1.2 Diagrama de atividade

O diagrama de atividades enfatiza a sequência e as condições para coordenar que determinado sistema chegue a determinado comportamento. O diagrama de atividades pode ser considerado o diagrama mais detalhado porque possui muitas semelhanças com os fluxogramas usados para desenvolver a lógica de programação (GUEDES, 2018).

A figura 7, demonstra o fluxo realizado pelo usuário, iniciando pelo momento da sua autenticação no sistema, seguindo de alguma atividade relacionada a criar uma entidade, editar uma entidade, visualizar uma entidade ou excluir uma entidade, e por fim a persistência do log de auditoria, descrevendo quem realizo determinada atividade.

No diagrama de atividades da figura 7, é referido o objeto como “entidade” pois o *framework* busca ser algo genérico, e entidade pode ser um cliente, um endereço, dentre outros. O *framework* não sabe para qual entidade será configurado para gerar os logs de auditoria.

Figura 7. Diagrama de atividade



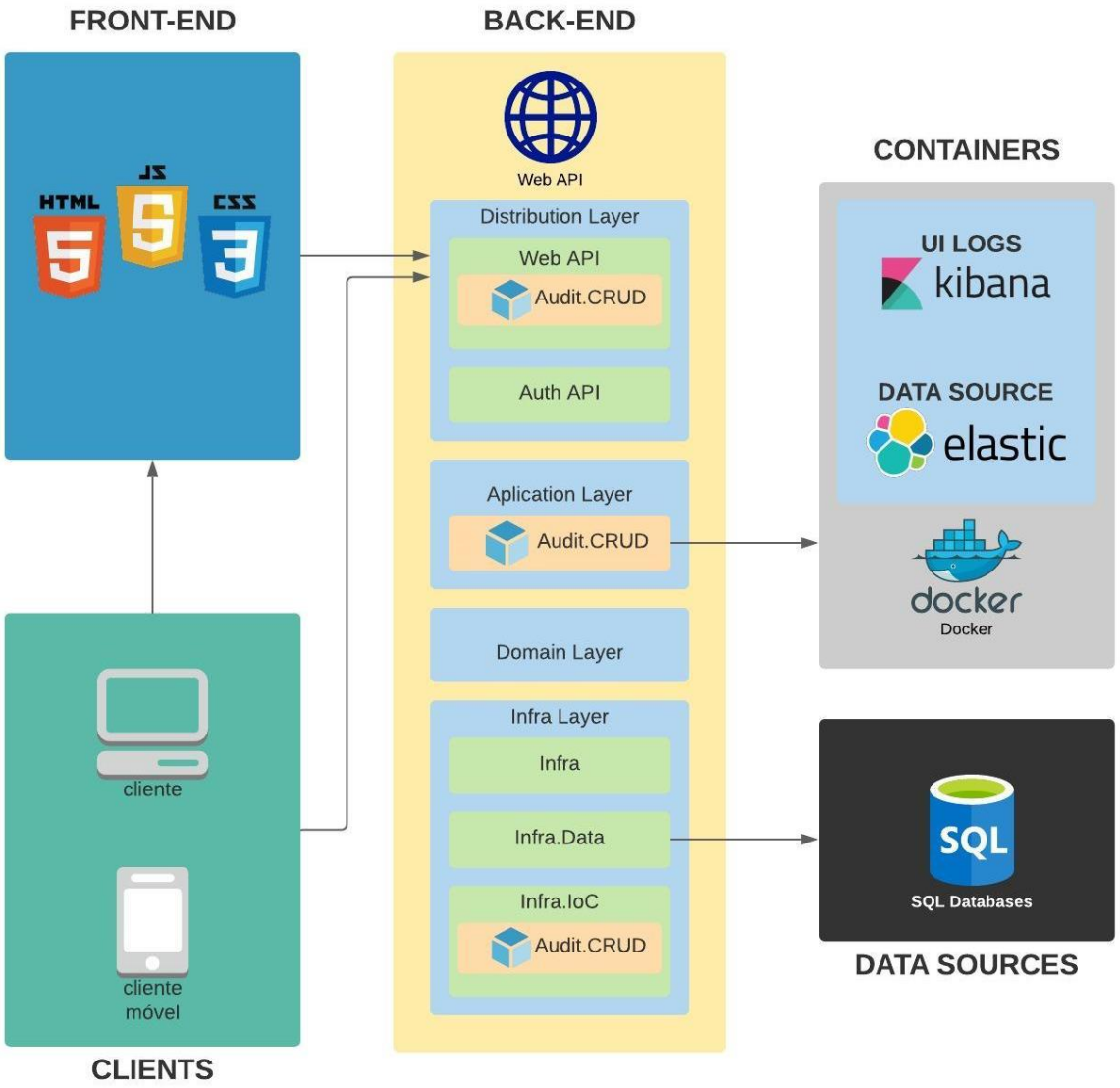
Fonte: Autoria própria

7.2 Diagrama de arquitetura

O diagrama de arquitetura tem como objetivo mostrar uma visão geral do contexto, demonstrar como os elementos de um sistema interagem entre si em um processo mais amplo. O diagrama de arquitetura de software descreve o que você está construindo, e como as partes interessadas interagem com ele e onde estão as restrições (GUTHRIE, 2020).

No diagrama de arquitetura da figura 9, é representado uma possível arquitetura que pode ser aderida ao utilizar o *framework* Audit.CRUD. Abaixo podemos observar que os *clients* podem se comunicar com a *Web* API através do *front-end* ou diretamente através de chamadas HTTP. Também pode-se observar que *Web* API está dividida em camadas, o Audit.CRUD é usado na camada de distribuição, onde o mesmo é configurado, também na camada de infraestrutura, para realizar a injeção de dependência da interface do Audit.CRUD e, por fim, também na camada de aplicação, onde será configurado o disparo dos logs de auditoria de acordo a necessidade do desenvolvedor/software. É nesta camada que o Audit.CRUD se comunica com o Elasticsearch que está sendo executado dentro de um *container* Docker.

Figura 8. Diagrama de arquitetura



Fonte: Autoria própria

8 TRABALHOS CORRELATOS

Nesta seção será descrito tópicos relacionado à trabalhos correlatos.

8.1 AUDIT.NET.Elasticsearch

O Audit.NET é um *framework* criado por Federico Daniel Colombo em abril de 2019, foi disponibilizado para a comunidade desenvolvedores através da plataforma Nuget. Atualmente é um pacote *open source* que recebe atualizações de diversos desenvolvedores através do GitHub.

Segundo Colombo (2020), com o “Audit.NET é possível gerar logs de auditoria com evidências para reconstrução e exame de atividades que afetam operações ou procedimentos específicos”.

O framework Audit.NET conta com vários pacotes, com suporte a diversas extensões de saída, como SQL, Mongo, Redis e Elasticsearch. O pacote responsável por sincronizar os dados com o Elasticsearch se chama Audit.NET.Elasticsearch.

O *framework* Audit.NET.Elasticsearch e Audit.CRUD tem o mesmo objetivo, que é proporcionar uma interface para desenvolvedores aplicarem logs de auditoria em softwares. Porém o Audit.CRUD, oferece uma interface de uso com os quatro métodos responsáveis por um CRUD (*create*, *read*, *upadate* e *delete*), para que de modo mais didático o desenvolvedor possa configurar os seus logs de auditoria.

Outro diferencial é que o Audit.CRUD, oferece uma interface de persistência dos dados de log mais enxuta, ou seja, busca persistir apenas informações relevantes ao modelo W7 da proveniência de dados, com objetivo de tornar mais fácil a leitura e compreensão dos logs, até para usuários menos avançados.

9 RESULTADOS

Neste trabalho foi apresentado um estudo demostrando a importância da aplicação de trilhas de auditoria em *Web* API's. Foi perceptível ao longo deste trabalho, a importância dos conhecimentos teóricos para a evolução deste *framewok*. Os estudos aqui realizados permitiram a criação de um protótipo de *framewok* para acoplar em Web API's REST, possibilitando a criação de trilhas de auditoria.

Um dos pontos relevantes deste estudo é o uso dos modelos de proveniência de dados. Com base nas características do modelo W7, foi elaborado um molde de persistência de dados para ser aplicado no *framewok* Audit.CRUD, com base nesse molde os logs de auditoria são sincronizados e persistidos no Elasticsearch.

Os ganhos da implantação do Audit.CRUD são: usabilidade, por ser um framework relativamente fácil de configurar e de aplicar; modularidade e reutilização, é possível utilizar o Audit.CRUD em diversos *Web* API's diferentes o que reduz o custo de produção; ganhos de segurança, por trazer a possibilidade de auditar as ações realizadas dentro de um sistema de informação; tomada de decisão, através da análise dos logs de auditoria é possível obter *insights* agilizando a tomada de decisões, controle dos acessos, ter um registro efetivo de qual usuário está acessando determinada informação.

A proposta de protótipo do *framewok* Audit.CRUD apresenta-se promissora para auxiliar desenvolvedores na aplicação logs de auditoria e atender os requisitos de segurança da auditabilidade e trafegabilidade de dados.

REFERÊNCIAS

ARAGÃO, Helder Guimarães. **JAVA E PROGRAMAÇÃO ORIENTADA A OBJETOS**: UMA ABORDAGEM DIDÁTICA. Salvador: Helder Guimarães Aragão, 2013. 63 p. Disponível em: <https://ler.amazon.com.br/?asin=B00E8SII90>. Acesso em: 30 jun. 2019.

ARAÚJO, Everton Coimbra de. **Orientação a Objetos em C#:** Conceitos e implementações em .NET. São Paulo: Casa do Código, 2017. 164 p. Disponível em: <https://ler.amazon.com.br/?asin=B077CWPJP6>. Acesso em: 30 jun. 2019.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO/IEC 17799**: Tecnologia da informação - Código de prática para a gestão da segurança da informação. Rio de Janeiro, 2001. 56 p. Disponível em: https://tororodeideias.files.wordpress.com/2012/03/nbr-iso-iec-17799.pdf. Acesso em: 25 out. 2020.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TECNICAS. **NBR ISO/IEC 27002**: ABNT NBR ISO/IEC 27002 – Tecnologia da informação – Técnicas de segurança – Código de prática para a gestão de segurança da informação. 1 ed. Rio de Janeiro, 2005. 120 p. Disponível em: http://www.fieb.org.br/download/senai/nbr_iso_27002.pdf. Acesso em: 15 out. 2020.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. 2. ed. Rio de Janeiro: Elsevier, 2006. 224 p. Disponível em: https://books.google.com.br/books?hl=pt-PT&lr=&id=ddWqxcDKGF8C&oi=fnd&pg=PR13&dq=uml+&ots=ffvMnehHRK&sig=cGLukL7fDObaAlAOJiBfTuA1ZPA#v=onepage&q=uml&f=false. Acesso em: 26 nov. 20.

BRASIL. Lei nº 12846, de 2013. **Lei Anticorrupção**. Brasil, 1 ago. 2013. Seção 5. Disponível em: https://www.planalto.gov.br/ccivil_03/_Ato2011-2014/2013/Lei/L12846.htm. Acesso em: 26 out. 2020.

CÂMARA, Sergio. Esquema de Estruturação SbC-EC para Log Seguro. **Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**, Belo Horizonte, v. 1, n. 1, p. 1-15, nov. 2014. Disponível em: https://www.researchgate.net/publication/271530020_Esquema_de_Estruturacao_SbC-EC_para_Log_Seguro. Acesso em: 19 out. 2020.

CÂMARA, Yzy Maria Rabelo; CAMPOS, Maria dos Remédios Moura; CÂMARA, Yis Rabelo. MUSICOTERAPIA COMO RECURSO TERAPÊUTICO PARA A SAÚDE MENTAL. **Cadernos Brasileiros de Saúde Mental**, Florianópolis, v. 5, n. 12, p. 94-117, 2013.

CARVALHO, Thiago Leite e. **Orientação a Objetos**: Aprenda seus conceitos e suas aplicabilidades de forma efetiva. São Paulo: Casa do Código, 2016. 238 p. Disponível em: <https://ler.amazon.com.br/?asin=B01LXHG8HX>. Acesso em: 30 jun. 2019.

CASTRO JUNIOR, Revailton de Souza. **Uma abordagem para evoluir sistemas web legados para web services**. 2017. 42 f. Dissertação (Mestrado) - Curso de Ciência de Computadores, Departamento de Ciência de Computadores, Faculdade de Ciências Universidade do Porto, Porto, 2017.

CIECHOWICZ, Marlene Perkoski; CIECHOWICZ, Franciele Cristina. A importância da disciplina metodologia da pesquisa no curso de pedagogia: um estudo de caso. **Revista Científica Multidisciplinar Núcleo do Conhecimento**, São Lourenço do Oeste, v. 4, n. 9, p. 05-25, set. 2019. Disponível em: https://www.nucleodoconhecimento.com.br/educacao/importancia-da-disciplina. Acesso em: 25 nov. 2020.

COLOMBO, Federico Daniel. **Audit.NET**. Disponível em: https://github.com/thepirat000/Audit.NET. Acesso em: 25 nov. 2020.

COULOURIS, George et al. **Sistemas Distribuídos**: Conceitos e Projeto. 5. ed. Porto Alegre: Bookman, 2013.

DATE, C.J.. **Introdução a sistemas de bancos de dados**. 8. ed. Rio de Janeiro: Elsevier Editora Ltda, 2004. 1417 p.

DB-ENGINES. **Classificação DB-Engines**. 2020. Disponível em: https://db-engines.com/en/ranking. Acesso em: 12 nov. 2020.

DOCKER. **O que é um container?** 2020. Disponível em: https://www.docker.com/resources/what-container. Acesso em: 18 nov. 2020.

ELASTIC. **O que é o ELK Stack?**: por que se chama elastic stack?. Por que se chama Elastic Stack?. 2020. Disponível em: https://www.elastic.co/pt/what-is/elk-stack. Acesso em: 10 nov. 2020.

ELASTIC. **O que é o Kibana?** 2020. Disponível em: https://www.elastic.co/pt/what-is/kibana. Acesso em: 16 nov. 2020.

FERNÁNDEZ, César Castelo. **Computação de Alto Desempenho usando Clusters**. 2009. 7 f. Monografia (Especialização) - Curso de Ciência da Computação, Instituto de Computação, Universidade Estadual de Campinas, Cidade Universitaria, 2009. Disponível em: https://www.ic.unicamp.br/~ducatte/mo401/1s2009/T2/089028-t2.pdf. Acesso em: 18 nov. 2020.

FONTES, Edison Luiz Goncalves. **SEGURANÇA DA INFORMAÇÃO**: o usuário faz a diferença. 5. ed. São Paulo: Saraiva, 2006. 223 p.

FONTES, Edison. **Segurança da Informação**: orientações práticas. São Paulo: Edição do Autor, 2016. 285 p.

FREIRE, Juliana *et al*. Provenance for Computational Tasks: A Survey. **Computing In Science & Engineering.** Salt Lake City, p. 11-21. abr. 2008. Disponível em: https://ieeexplore.ieee.org/document/4488060/authors#authors. Acesso em: 18 nov. 2020.

FREITAS, Damiana Galache de; SANTO, Danilo Moreira dos; SILVA, Patrick da Conceição da. **Um web service restful semântico para comparação de preços**. 2019. 85 f. TCC (Graduação) - Curso de Sistemas de Informação, Instituto Federal Fluminense, Campos dos Goytacazes, 2019.

GEYER, Cláudio F. R.; VARGAS, Patrícia Kayser; AZEVEDO, Silvana Campos de. **Sistemas Distribuídos.** 2001. Curso de Ciência da Computação, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.

GOMES, Eriton dos Santos. **Sense API: Uma Rest API para aplicações M-Health que utilizam sensores vestíveis para obter informações de saúde de pessoas**. 2018. 58 f. TCC (Graduação) - Curso de Engenharia de Software, Universidade Estadual da Paraíba, Campo Grande, 2018.

GOMES, Daniel Adorno; JANDL JUNIOR, Peter. **Web services SOAP e REST**. Intellectus, Campinas, p.88-114, jul. 2009.

GUEDES, Gilleanes T. A.. **UML 2**: uma abordagem prática. 3. ed. São Paulo: Novatec, 2018. 213 p. Disponível em: https://books.google.com.br/books?hl=pt-PT&lr=lang_pt&id=mJxMDwAAQBAJ&oi=fnd&pg=PA2&dq=+UML&ots=x9rV. Acesso em: 26 nov. 2020.

GUIMARÃES, Leandro. **Qual a diferença entre dado e informação?** 2019. Disponível em: https://www.knowsolution.com.br/diferenca-dado-e-informacao/. Acesso em: 13 out. 2020.

GUTHRIE, Georgina. **Everything you need to know about architectural diagrams (and how to draw one)**. 2020. Disponível em: https://cacoo.com/blog/everything-you-need-to-know-about-architectural-diagrams-and-how-to-draw-one/. Acesso em: 27 nov. 2020.

HEUSER, Carlos Alberto. **Banco de Dados Relacional**: conceitos, sql e administração. Porto Alegre: Edição do Autor, 2019. 714 p.

JUNIOR, Mauricio. Web API na linguagem C#: Usando Visual Studio. São Paulo: Mauricio Junior, 2018. 274 p. Disponível em: <https://ler.amazon.com.br/?asin=B07FPVJH32>. Acesso em: 29 maio 2019.

LAUREANO, Marcos Aurelio Pchek. SEGURANÇA COMO ESTRATÉGIA DE GESTÃO DA INFORMAÇÃO. **Revista Economia & Tecnologia**, Curitiba, v. 8, n. 3, p. 38-44, jan. 2005. Disponível em: https://roitier.pro.br/wp-content/uploads/2016/02/asti_ii_material_apoio_2_seguranca_informacao_texto_base2.pdf. Acesso em: 15 out. 2020.

LECHETA, Ricardo R.. **Web Services RESTFul.** São Paulo: Novatec, 2015. Disponível em: <https://docplayer.com.br/4483414-Ricardo-r-lecheta-novatec.html>. Acesso em: 29 maio 2019.

LEITE, Alessandre Ferreira. **Frameworks e Padrões de Projeto**. 2006. Disponível em: <https://www.devmedia.com.br/frameworks-e-padroes-de-projeto/1111>. Acesso em: 02 jul. 2019.

LIBERALI, Ernâni Teixeira. **UM MODELO PARA PROTEÇÃO DE TRILHAS DE AUDITORIA EM SISTEMAS DE IDENTIFICAÇÃO ELETRÔNICA**. 2012. 102 f. Dissertação (Mestrado) - Curso de Engenharia de Produção, Universidade Federal de Santa Maria, Santa Maria, 2012. Disponível em: https://repositorio.ufsm.br/handle/1/8241. Acesso em: 18 out. 2020.

LIM, Chunhyeok; LU, Shiyong; CHEBOTKO, Artem; FOTOUHI, Farshad. Prospective and Retrospective Provenance Collection in Scientific Workflow Environments. **2010 Ieee International Conference On Services Computing**, [S.L.], p. 449-456, jul. 2010. IEEE. http://dx.doi.org/10.1109/scc.2010.18. Disponível em: https://ieeexplore.ieee.org/document/5557202. Acesso em: 18 nov. 2020.

LIMA, Naísses Zoia. **Desenvolvimento de um framework para métodos sem malha**. 2011. 183 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, 2011. Disponível em: https://repositorio.ufmg.br/bitstream/1843/RCMA-8PXLUD/1/naisses_zoia_lima.pdf. Acesso em: 18 nov. 2020.

LÓSCIO, Bernadette Farias. NoSQL no desenvolvimento de aplicações Web colaborativas. **VIII Simpósio Brasileiro de Sistemas Colaborativos**, Paraty, v. 1, n. 1, p. 1-17, out. 2011. Disponível em: https://www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf. Acesso em: 03 nov. 2020.

LYRA, Mauricio Rocha. **Governança da Segurança da Informação**. Brasília: Edição do Autor, 2015. 208 p.

LYRA, Mauricio Rocha. **Segurança da Informação na Era da Conectividade: Ensaios e Reflexões**. Brasília: Edição do Autor, 2019. 130 p.

MACEDO, Savio de Paiva. **Desenvolvimento de uma Plataforma para Manipulação e Análise de Dados de Programas Sociais do Governo Federal**. 2020. 17 f. TCC (Graduação) - Curso de Engenharia de Software, Centro de Ensino Superior de Juiz de Fora, Juiz de Fora, 2020. Disponível em: https://www.researchgate.net/publication/340237459_Desenvolvimento_de_uma_Plataforma_para_Manipulacao_e_Analise_de_Dados_de_Programas_Sociais_do_Governo_Federal. Acesso em: 12 nov. 2020.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Fundamentos de metodologia científica**. 5. ed. São Paulo: Editora Atlas, 2003. 311 p.

MELO, Anaximandro Barbosa de. **Big Data e NoSQL**: ontologias e estado da arte (ciência de dados, inteligência artificial, machine learning e deep learning livro 1). São Paulo: Amazon, 2020. 49 p.

MICROSOFT. **Introdução à linguagem C# e ao .NET Framework**. 2015. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>. Acesso em: 30 jun. 2019.

MICROSOFT. **Relacional versus Dados NoSQL**. 2020. Disponível em: https://docs.microsoft.com/pt-br/dotnet/architecture/cloud-native/relational-vs-nosql-data. Acesso em: 29 out. 2020.

MICROSOFT. **Uma introdução ao NuGet**. 2019. Disponível em: https://docs.microsoft.com/pt-br/nuget/what-is-nuget. Acesso em: 08 nov. 2020.

MKHITARYAN, Armina. **Por que o C # está entre as linguagens de programação mais populares do mundo?** 2017. Disponível em: <https://medium.com/sololearn/why-is-c-among-the-most-popular-programming-languages-in-the-world-ccf26824ffcb>. Acesso em: 30 jun. 2019.

MONTANHEIRO MORO, Tharcis dal; DORNELES, Carina F.; REBONATTO, Marcelo Trindade. Web services WS-* versus Web Services REST. Reic - Revista de Iniciação Cientifica, Passo Fundo, p.36-51, jan. 2011.

MONTANHEIRO, Lucas Souza; CARVALHO, Ana Maria Martins; RODRIGUES, Jackson Alves. **Utilização de JSON Web Token na Autenticação de Usuários em APIs REST**. XIII Encontro Anual de Computação - Enacomp 2017, Morrinhos, p.186-192, maio 2017.

MORAIS, Anderson Melo de. **Simulação e Avaliação de Desempenho de uma Rede Blockchain Utilizando Containers Docker**. 2020. 15 f. Monografia (Especialização) - Curso de Informática Aplicada, Universidade Federal Rural de Pernambuco, Recife, 2020. Disponível em: https://www.e-publicacoes.uerj.br/index.php/cadinf/article/view/46934/35686. Acesso em: 18 nov. 2020.

NASCIMENTO, Allana. **Bancos NoSQL Vs Relacionais**. 2018. Mundo DevOps. Disponível em: https://mundodevops.com/blog/bancos-nosql-vs-relacionais/. Acesso em: 29 out. 2020.

NASCIMENTO, Wladimilson M.. **O que é o NuGet?** 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-o-nuget/. Acesso em: 08 nov. 2020.

OLIVEIRA, Samuel Silva de. BANCOS DE DADOS NÃO-RELACIONAIS: bancos de dados não-relacionais. **Revista Eletrônica da Escola de Administração Pública do Amapá**, Macapá, v. 1, n. 2, p. 184-194, dez. 2014. Disponível em: https://www2.unifap.br/oliveira/files/2016/02/35-124-1-PB.pdf. Acesso em: 29 out. 2020.

PAULA, Marília Alves de. **Gestão de custos como suporte para a formação do preço de venda numa microempresa varejista de roupas e calçados**. 2017. 53 f. TCC (Graduação) - Curso de Administração, Universidade Federal da Paraíba, João Pessoa, 2017.

PAULA, Renato de. **Proveniencia de Dados em Workflows de Bioinformatica**. 2012. 105 f. Dissertação (Mestrado) - Curso de Mestrado em Informática, Universidade de Brasília, Brasília, 2012. Disponível em: https://repositorio.unb.br/bitstream/10482/12699/1/2012_RenatodePaula.pdf. Acesso em: 21 nov. 2020.

PINTO, Walker Douglas Garcia. DOCKER –CONTAINERS NÃO SÃO VM's. **2º Seminário de Tecnologia Gestão e Educação**, Porto Alegre, v. 1, n. 1, p. 5-10, nov. 2019. Disponível em: http://raam.alcidesmaya.com.br/index.php/SGTE/article/view/21/24. Acesso em: 18 nov. 2020.

PIRES, Eduardo. **.NET Standard – Você precisa conhecer!** 2017. Disponível em: https://www.eduardopires.net.br/2017/06/net-standard-voce-precisa-conhecer/. Acesso em: 08 nov. 2020.

PIVA, Gustavo Dibbern; OLIVEIRA, Wilson José de. **Analise e gerenciamento de dados**. 3. ed. São Paulo: Centro Paula Souza, 2010. 106 p. Disponível em:

http://www.etecdesapopemba.com.br/assets/INFORM%C3%81TICA%20VOL.%203%20-%20AN%C3%81LISE%20E%20GERENCIAMENTO%20DE%20DADOS.pdf. Acesso em: 27 nov. 2020.

RAMOS, Gerson. **Introdução ao Elastic Stack**. 2020. Disponível em: https://medium.com/data-hackers/introdu%C3%A7%C3%A3o-ao-elastic-stack-914a89b5f07c. Acesso em: 10 nov. 2020.

ROCHA, Fernando da. **Benefícios da utilização de uma aplicação para concentração e análise em tempo real de logs dos servidores web de uma instituição de ensino superior**. 2018. 18 f. TCC (Graduação) - Curso de Gestão da Segurança da Informação, Unisul, Tubarão, 2018. Disponível em: https://riuni.unisul.br/handle/12345/4959?locale-attribute=pt_BR. Acesso em: 04 nov. 2020.

RODRIGUES, Matheus Dornelas. **ANÁLISE SOBRE BASES DE DADOS PARA ARMAZENAMENTO E CONSULTA DE DADOS NÃO ESTRUTURADOS E SEMIESTRUTURADOS.** 2016. 63 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal de Pernambuco, Recife, 2016. Disponível em: https://www.cin.ufpe.br/~tg/2016-1/mdr.pdf. Acesso em: 12 nov. 2020.

RORATTO, Rodrigo; DIAS, Evandro Dotto. SEGURANÇA DA INFORMAÇÃO DE PRODUÇÃO E OPERAÇÕES: um estudo sobre trilhas de auditoria em sistemas de banco de dados. **Journal Of Information Systems And Technology Management**, São Paulo, v. 11, n. 3, p. 717-734, 7 dez. 2014. TECSI. http://dx.doi.org/10.4301/s1807-17752014000300010. Disponível em: https://www.scielo.br/scielo.php?pid=S1807-17752014000300717&script=sci_arttext&tlng=pt. Acesso em: 19 out. 2020.

SANTANA, Luiz Henrique Zambom. **Elasticsearch: desenvolvendo Big Data com PHP**. 2015. Disponível em: https://www.devmedia.com.br/elasticsearch-desenvolvendo-big-data-com-php/31609. Acesso em: 18 nov. 2020.

SANTOS, Alexandro Klein dos. **Os IDE's (Ambientes de Desenvolvimento Integrado) como ferramentas de trabalho em informática**. Universidade Federal de Santa Maria, Santa Maria, v. 1, n. 1, p.4-10, jun. 2019. Disponível em: <http://www-usr.inf.ufsm.br/~alexks/elc1020/artigo-elc1020-alexks.pdf>. Acesso em: 30 jun. 2019.

SÊMOLA, Marcos. **Gestão da segurança da Informação**: uma visão executiva. Rio de Janeiro: Elsevier, 2003.

SILVA, Fabricio Alves Barbosa da. Big Data e Nuvens Computacionais: aplicações em saúde pública e genômica. **Journal Of Health Informatics.** São Paulo, p. 74-79. jun. 2016. Disponível em: http://www.jhi-sbis.saude.ws/ojs-jhi/index.php/jhi-sbis/article/view/336. Acesso em: 03 nov. 2020.

TAN, Wang Chiew. Why and where: A characterization of data provenance. **International Conference On Database Theory.** Philadelphia, p. 316-330. out. 2001. Disponível em: https://www.cis.upenn.edu/~sanjeev/papers/icdt01_data_provenance.pdf. Acesso em: 22 nov. 2020.

TANNENBAUM, A.S; STEEN, M.V. **Sistemas distribuídos: princípios e paradigmas**. São Paulo: Pearson/Prentice Hall, 2007.

TEIXEIRA NETO, Rodolfo. **Banco De Dados**: o que é e qual sua importância no armazenamento de arquivos?. O Que É E Qual Sua Importância No Armazenamento De Arquivos?. 2019. Disponível em: https://dominandoredes.com.br/banco-de-dados/#Qual_A_Importacircncia_Do_Banco_De_Dados. Acesso em: 05 nov. 2020.

TIOBE, Programming Community Index Definition. TIOBE Index for June 2019. 2019. Disponível em: <https://www.tiobe.com/tiobe-index/>. Acesso em: 30 jun. 2019.

TONG, Zachary. **O que é um índice Elasticsearch?** 2013. Disponível em: https://www.elastic.co/pt/blog/what-is-an-elasticsearch-index. Acesso em: 16 nov. 2020.

VIEIRA, Sinval. **Logs**. 2018. Disponível em: https://medium.com/@sinvalmendes/logs-a4025f227c46. Acesso em: 18 out. 2020.

YIN, Robert K.. **Estudo de caso**: planejamento e métodos. 2. ed. Porto Alegre: Bookman, 2001. 200 p. Disponível em: https://saudeglobaldotorg1.files.wordpress.com/2014/02/yin-metodologia_da_pesquisa_estudo_de_caso_yin.pdf. Acesso em: 25 nov. 2020.

W3C. **Web Services Architecture**. 2004. Disponível em: <https://www.w3.org/TR/ws-arch/>. Acesso em: 29 maio 2019.