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

DATACACHE: GESTÃO E PADRONIZAÇÃO NA COMUNICAÇÃO ENTRE APLICAÇÕES

Jackson Diesel Majolo

Lajeado, novembro de 2019.

Jackson Diesel Majolo

DATACACHE: GESTÃO E PADRONIZAÇÃO NA COMUNICAÇÃO ENTRE APLICAÇÕES

Monografia apresentada ao Centro de Ciências Exatas e Tecnológicas da Universidade do Vale do Taquari – UNIVATES, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Me. Fabrício Pretto

Lajeado, novembro de 2019.

Jackson Diesel Majolo

DATACACHE: GESTÃO E PADRONIZAÇÃO NA COMUNICAÇÃO ENTRE APLICAÇÕES

A Banca examinadora abaixo, aprova a Monografia apresentada na disciplina Trabalho de Conclusão de Curso II, da Universidade do Vale do Taquari – Univates, como exigência para obtenção do grau de Bacharel em Engenharia de Software:

Prof. Me. Fabrício Pretto – orientador

Prof. Me. Alexandre Stürmer Wolf

Prof. Me. Juliano Dertzbacher

Lajeado, novembro de 2019.

RESUMO

Em uma era onde cada vez mais recursos tecnológicos estão sendo utilizados para todos os tipos de negócios, as necessidades do mercado acabam mudando constantemente no que tange as tecnologias e sistemas empregados. Para que essas mudanças ocorram da forma mais rápida possível, aplicações modulares que se conectam a outras em busca de certas informações ou processos são desenvolvidas. Durante o processo de recuperação ou envio de informação, a falha e a lentidão representam insatisfações para o usuário e até mesmo a perda de credibilidade para o serviço. O objetivo do presente trabalho foi desenvolver uma plataforma para gerenciamento de *web services* com suporte a cache que melhore a qualidade de serviços de troca de informação, sendo capaz de organizar o fluxo da comunicação entre aplicações de uma organização. Essa plataforma monitora todas as requisições *web services* e emite alerta ao grupo de desenvolvedores quando há ocorrência de algum problema. Por meio de um painel administrativo, é possível acompanhar tempo e consumo de dados trafegados das requisições, além de gerar estatísticas de todo fluxo passante pela plataforma. Como resultado do projeto e da validação, a plataforma mostrou-se eficiente na quantificação das solicitações e gerenciamento via cache, promovendo melhor uso dos recursos de infraestrutura.

Palavras-chave: Sistemas de informações. Integração de Sistemas. Web Services. SOA. Cache.

LISTA DE FIGURAS

FIGURA 1 – PROCESSO DE TRANSFORMAÇÃO DO DADO EM INFORMAÇÃO.....18
FIGURA 2 – REPRESENTAÇÃO DA COMUNICAÇÃO FEITA POR SOAP.....26
FIGURA 3 – EXEMPLO DE ENVELOPE SOAP DE SOLICITAÇÃO.....27
FIGURA 4 – EXEMPLO DE ENVELOPE SOAP DE RESPOSTA.....27
FIGURA 5 – REPRESENTAÇÃO DA COMUNICAÇÃO FEITA POR REST.....28
FIGURA 6 – EXEMPLO DE ENVELOPE HTTP DE SOLICITAÇÃO.....30
FIGURA 7 – EXEMPLO DE ENVELOPE HTTP DE RESPOSTA.....30
FIGURA 8 – ARQUITETURA DA APLICAÇÃO DATACACHE.....41
FIGURA 9 – FLUXO DE UM *REQUEST/RESPONSE* NA APLICAÇÃO.....42
FIGURA 10 – MODELO DE ESTADOS.....45
FIGURA 11 – MODELO DE CASOS DE USO.....46
FIGURA 12 – BANCO DE DADOS – DATACACHE.....48
FIGURA 13 – BANCO DE DADOS LOG.....49
FIGURA 14 – CADASTRO DE GRUPOS DE DESENVOLVIMENTO.....50
FIGURA 15 – LISTAGEM DE APLICAÇÕES.....51
FIGURA 16 – DETALHES DA APLICAÇÃO.....52
FIGURA 17 – *DASHBOARD* DA APLICAÇÃO.....53
FIGURA 18 – TSOAPSERVER CLASSE DO SERVIÇO SOAP.....54
FIGURA 19 – TRESTSERVER CLASSE DO SERVIÇO REST.....55
FIGURA 20 – DCRESPONSESERVICE CLASSE DE SERVIÇO.....56
FIGURA 21 – RESPOSTAS DA TERCEIRA PERGUNTA, REFERENTE AO CONHECIMENTO DOS SERVIÇOS ATUAIS.....62
FIGURA 22 – RESPOSTAS DA QUARTA PERGUNTA, REFERENTE A IMPORTÂNCIA DE UMA FERRAMENTA.....63
FIGURA 23 – RESPOSTAS DA QUINTA PERGUNTA, REFERENTE A NOTA GERAL DA FERRAMENTA.....64
FIGURA 24 – RESPOSTAS DA SEXTA PERGUNTA, REFERENTE A PONTOS POSITIVOS DA FERRAMENTA.....64

FIGURA 25 – RESPOSTAS DA SÉTIMA PERGUNTA, REFERENTE A PONTOS NEGATIVOS DA FERRAMENTA.....65

LISTA DE QUADROS

QUADRO 1 – REQUISITOS FUNCIONAIS.....43
QUADRO 2 – REQUISITOS NÃO FUNCIONAIS.....44
QUADRO 3 – PROVEDORES DE SERVIÇOS.....58
QUADRO 4 – CONSUMIDORES DE SERVIÇOS.....58
QUADRO 5 – *RESPONSES* DE SERVIÇOS.....59
QUADRO 6 – MEDIA DE TEMPO DOS *RESPONSES* COM *CACHE*.....60

LISTA DE TABELAS

TABELA 1 – ATRIBUTOS USADOS NA DEFINIÇÃO DO QOS.....22
TABELA 2 – CÓDIGOS DE RETORNO HTTP.....29

LISTA DE ABREVIATURAS

API:	Application Programming Interface
BD:	Banco de Dados
CSS:	Cascading Style Sheets
FTP:	File Transfer Protocol
FW:	Framework
HTML:	HyperText Markup Language
HTTP:	HyperText Transfer Protocol
JS:	JavaScript
JSON:	JavaScript Object Notation
MVC:	Model View Controller
PHP:	Hypertext PreProcessor
QoS:	Quality of Service
REST:	Representational State Transfer

RF:	Requisito Funcional
RNF:	Requisito Não Funcional
SGBD:	Sistema Gerenciador de Banda de dados
SI:	Sistema de informação
SLA:	Service Level Agreement
SMTP:	Simple Mail Transfer Protocol
SOA:	Service Oriented Architecture
SOAP:	Simple Object Access Protocol
SQL:	Structured Query Language
TI:	Tecnologia da informação
TCP:	Transmission Control Protocol
UDDI:	Universal Description Language
URI:	Uniform Resource Identifier
URL:	Uniform Resource Locator
XML:	Extensible Markup Language
WS:	Web Service

SUMÁRIO

1 INTRODUÇÃO....13
1.1 Motivação....14
1.2 Objetivos....15
1.2.1 Objetivo geral....15
1.2.2 Objetivos Específicos....16
1.3 Estrutura do trabalho....16
2 REFERENCIAL TEÓRICO....18
2.1 Sistemas de informação....18
2.2 Integração de sistemas....20
2.3 SOA....20
2.4 Web services....21
2.5 Qualidade de serviços (QoS)....22
2.6 Tipos de dados....23
2.6.1 XML....24
2.6.2 JSON....24
2.7 Tipos de *Web services*....24
2.7.1 SOAP....25
2.7.2 REST....28
2.8 Cache....31
3 FERRAMENTAS RELACIONADAS....32
4 METODOLOGIA....34
4.1 Tipo de pesquisa....34
4.2 Ferramentas utilizadas....36
4.2.1 Apache....36
4.2.2 HTML e CSS....36
4.2.3 JavaScript....37
4.2.4 jQuery....37
4.2.5 PHP....37
4.2.6 Adianti Framework....38
4.2.7 PostgreSQL....38

5 ESPECIFICAÇÃO DA PLATAFORMA....40
5.1 Visão geral da implementação....40
5.2 Requisitos....43
5.2.1 Requisitos funcionais....43
5.2.2 Requisitos não funcionais....44
5.3 Modelos de estados....45
5.4 Modelo de casos de uso....46
5.5 Modelo relacional....47
5.6 Interfaces da aplicação....49
5.7 Processamento da API....53
6 RESULTADOS E DISCUSSÕES....57
6.1 Análise da implementação....57
6.2 Análise da pesquisa....61
7 CONSIDERAÇÕES FINAIS....66

1 INTRODUÇÃO

Em um mundo marcado por avanços tecnológicos e com a crescente evolução das organizações, softwares têm se tornado imprescindíveis nas atividades do dia a dia. Abrangendo diferentes áreas de conhecimento, cada vez mais infraestruturas e serviços são controlados por sistemas computacionais (SOMMERVILLE, 2011).

A Tecnologia da Informação – TI, conceituada como recurso não humano, é uma parte fundamental deste processo, pois as tecnologias auxiliam os sistemas de informações na capacidade e velocidade de coleta, armazenamento, processamento e até mesmo na distribuição das informações de uma organização (AUDY; ANDRADE; CIDRAL, 2007).

Com um objetivo claro de coletar, processar e disponibilizar informações precisas, os diferentes tipos de sistemas de informação e o grande aumento na utilização de softwares, impõe que os profissionais da área de TI usem tecnologias inovadoras para melhorar o desenvolvimento e a manutenção de sistemas de alta qualidade e versatilidade (PRESSMAN, 2010).

Atualmente, a informação representa um valor significativo para a pessoa ou a organização que a detém. Esta pode agregar valor ao produto fazendo com que ele se diferencie de sua concorrência. É possível afirmar que a informação está presente em todas as atividades relacionadas a processos, pessoas e tecnologias (REZENDE, 2013).

Tendo como base a forte competitividade do mercado, a gestão eficiente da informação pode dar a organização maior flexibilidade e competitividade no seu meio ambiente. Para que um software seja considerado de qualidade este deve cumprir com seus

objetivos, satisfazendo as metas empresariais alinhando recursos de TI com as necessidades da empresa (MECENAS; OLIVEIRA, 2005).

Uma das estratégias utilizadas para isso é criação de sistemas especialistas, onde cada um é responsável por organizar, consolidar e gerir a sua informação que no futuro servirá para diferentes propósitos dentro da organização. Embora essa informação esteja em locais diferentes, ainda é necessário um sistema ter acesso as informações de outro para poder manter sua qualidade (MARTINS, 2005). Tonsig (2008) complementa afirmando que sistemas que são construídos sem nenhum tipo de preocupação com a utilização de suas funcionalidades, podem ter maior complexidade no desenvolvimento ou até mesmo em sua manutenção.

Kuehne (2009) afirma que para padronizar e facilitar a comunicação feita entre aplicações distintas, surgem os conceitos e padronizações de *Web Services* – WS. Com a possibilidade de utilizar diferentes formatos de dados, um WS é considerado uma funcionalidade modular que pode ser invocada através de algum protocolo e assim interligar diferentes tipos de sistemas e até diferentes plataformas.

Pode-se afirmar então que sistemas de informação são desenvolvidos utilizando diferentes tipos tecnologia e as ferramentas que realizam a interligação destes sistemas são consideradas mecanismos de otimização e aumento de desempenho organizacional. Eles evitam a redundância de processos realizados ou sistemas que possuem grande nível de customização e visam reduzir custos, eliminar gargalos e desburocratizar processos. Com a integração de sistemas é possível executar processos em sistemas especialistas transmitindo ou requisitando informações de outros sistemas e, até mesmo, de fornecedores distintos.

1.1 Motivação

A maioria dos serviços *web* que necessitam trocar informações entre aplicações não possuem uma forma padronizada para realizar essa atividade. Empresas que desenvolvem aplicações que constantemente se comunicam com outros sistemas em busca de informações, estão um passo à frente de seus concorrentes, pois criam sistemas especialistas que permitem

interligar suas funcionalidades em diferentes tipos de sistemas, abrangendo assim várias áreas de negócio.

Porém este tipo de aplicação está mais suscetível a erros do que uma aplicação comum, devido à necessidade de ter algum tipo de ligação entre consumidores e provedores e não ter gerenciamento algum sobre esta comunicação. Atualmente existem poucas ferramentas disponíveis no mercado que fazem o controle de fluxo de requisições a servidores *web*. Ferramentas desse tipo podem auxiliar os administradores a reconhecerem possíveis incoerências em suas funcionalidades.

Um exemplo dessa problemática está relacionado com a comunicação entre sistemas via *web service*, de modo que os sistemas clientes ou consumidores necessitam consultar ou enviar dados para algum serviço disponível. Nesse cenário, o servidor que provê o *web service* está à disposição para fornecer o acesso, no entanto, não existem estatísticas de acesso, controle de acesso ou gerenciamento para prover contingência. Quando a falha no serviço for detectada, o setor de suporte será acionado, tratando do caso de forma totalmente reativa.

Visando resolver este problema, o presente trabalho busca auxiliar os administradores de sistemas para que tenham controle e estatísticas de todas as funcionalidades vinculadas ao serviço de *web service* e até mesmo melhorar algumas funções com o cacheamento de dados.

1.2 Objetivos

Nesta seção são apresentados os objetivos gerais e específicos do presente trabalho.

1.2.1 Objetivo geral

Desenvolver uma plataforma para gerenciamento das requisições entre sistemas distintos via *web services*, realizando a otimização da troca de dados por meio da geração de cache.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

a) Realizar uma pesquisa sobre sistemas de informação, integração entre sistemas, tipos de dados, cacheamento de dados e ferramentas relacionadas.

b) Definir requisitos funcionais e não funcionais para a implementação da ferramenta.

c) Implementar a ferramenta proposta.

d) Avaliar a ferramenta com profissionais da área de Tecnologia da Informação.

e) Coletar e analisar os dados resultantes da avaliação dos usuários.

f) Analisar os resultados obtidos.

1.3 Estrutura do trabalho

Para melhorar a compreensão do presente trabalho seus capítulos foram divididos em uma ordem de apresentação

O capítulo 1 apresenta uma visão geral do trabalho, seu tema, objetivos, justificativa e a sua estrutura.

O capítulo 2 contém o referencial teórico utilizado como embasamento para o desenvolvimento do trabalho, que conta com os seguintes subcapítulos: Sistemas de informação, Integração de sistemas, SOA, *Web services*, Qualidade em *web services*, Tipos de dados, SOAP, REST e Cache.

No capítulo 3 são apresentados alguns projetos e ferramentas relacionadas ao sistema desenvolvido.

No capítulo 4 é abordada a metodologia utilizada para a realização do trabalho, no qual é apresentado o método científico em que este trabalho se enquadra, acompanhado das tecnologias que serão utilizadas para a solução proposta.

O capítulo 5 apresenta uma visão geral da solução, especificações de requisitos e os artefatos gerados.

O capítulo 6, apresenta a avaliação dos resultados obtidos aplicando um questionário para um grupo de gestores da área de TI.

Por fim, o capítulo 7 contém as considerações finais que foram tiradas com a realização deste presente trabalho.

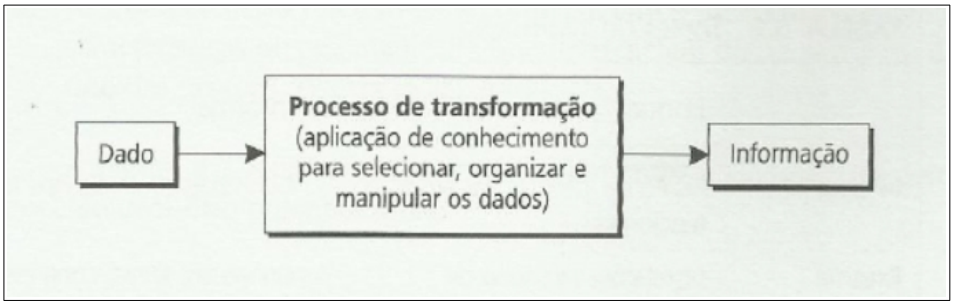
2 REFERENCIAL TEÓRICO

O presente capítulo tem como objetivo principal, contextualizar os temas relacionados ao desenvolvimento da plataforma. Para isso, este capítulo busca abordar temas como Sistemas de informação, Integração de sistemas, Web services, Tipos de dados, SOAP, REST e Cache.

2.1 Sistemas de informação

O termo sistema é utilizado de forma muito genérica e pode ser definido por todo o conjunto de elementos que interagem entre si. Já o termo informação pode ser definido como um conjunto de dados organizados de modo que tenham algum valor ou significado para o homem. Conceitualmente, um sistema de informação tem como objetivo coletar, processar e transmitir as informações, que ao final do processo estabelecido representem algo para o usuário, conforme demonstrado exemplo na Figura 1 (AUDY; ANDRADE; CIDRAL, 2007).

Figura 1 – Processo de transformação do dado em informação



Fonte: Audy, Andrade e Cidral (2007, pág. 95).

Os sistemas de informação são compostos por um subsistema social e outro automatizado, no qual o primeiro inclui processos, pessoas, informações e documentos, e o segundo automatização dos meios, computadores, redes de comunicação, máquinas, entre outros.

A concepção de um SI é maior que a simples concepção de sistema, pois abrange também a parte do hardware com o software, incluindo processos e seus atuadores, que podem ser executados fora das máquinas. Os seus principais objetivos são armazenar, tratar e fornecer as informações, de uma forma em que elas possam apoiar os processos e até mesmo as funções de uma organização (O´BRIEN, 2004).

> Hoje, todos admitem que conhecer sistemas de informação é essencial para os administradores, porque a maioria das organizações precisam deles para sobreviver e prosperar. Com os sistemas, as empresas podem aumentar o seu grau e alcance de participação no mercado, oferecer novos produtos, adequar-se internamente e, muitas vezes, transformar radicalmente o modo como conduzem seus negócios (LAUDON e LAUDON, 2004, p. 4).

Para Laudon e Laudon (2004), existem diferentes tipos de sistemas, isso porque há diferentes interesses, especialidades e níveis dentro de uma organização, sendo que nenhum sistema sozinho poderá oferecer e fornecer todas as informações das quais uma empresa precisa.

Entretanto, como Audy, Andrade e Cidral (2007) ressaltam, as classificações mais utilizadas, agrupam esses sistemas por finalidade principal do seu uso, sendo elas:

a) **Sistemas de Processamento de Transações (SPT):** Executam e registram as transações rotineiras em alguns processos de uma organização.

b) **Sistemas de Informação Gerencial (SIG)**: Sintetizam, registram e relatam a situação das operações de uma organização.

c) **Sistemas de Apoio à Decisão (SAD):** Auxiliam uma organização a tomar decisões estruturadas, com base em dados obtidos de um outro sistema.

d) **Sistemas de Informação Executiva (SIE):** Ajudam uma organização a tomar decisões não-estruturadas, possibilitando uma visão sobre a situação atual e até mesmo tendências sobre a área de negócio dessa organização.

2.2 Integração de sistemas

O conceito de integração em sistemas de informação é dado como a partilha de informações, processos e até mesmo fonte de dados. Isso acontece devido a vários motivos, por exemplo, a constante mudança no negócio de uma organização, cada vez mais um SI deve ser mais focado em suas funções, o crescimento de sistemas legados, entre outros (MARTINS, 2005).

Para Correia (2015), a necessidade de um dado ter que estar disponível entre diferentes tipos de sistemas já constitui um grande desafio para as organizações, pois podem haver redundância de informações e discrepâncias nas mesmas.

Para que haja integração entre duas aplicações distintas, deve existir uma forma de comunicação entre elas. É frequente encontrar organizações compostas de sistemas específicos para realização das atividades profissionais, onde a comunicação entre elas é feita através de canais de comunicações que não abrangem TI (MARTINS, 2005).

Existem várias formas de realizar essa troca de informação de forma automática no âmbito da tecnologia, sendo que uma das mais utilizadas é por meio de *web services* (COLÂNGELO, 2001).

2.3 SOA

O *Service Oriented Architecture* – SOA é um conceito utilizado na construção de sistemas distribuídos, formando um conjunto de serviços separados sem estado, no qual cada sistema trabalha de forma independente. Seguindo esse conceito as aplicações tornam-se acessíveis umas às outras, aumentando a reutilização de código e flexibilidade do software (SOMMERVILLE, 2011).

Sommerville (2011), acredita que o grande benefício de aplicações baseadas em serviços é a cooperação e compartilhamento de funções de negócio umas com as outras. A adoção universal desta interligação de sistemas fornece a empresas orientadas a serviço, uma

melhora significante na agilidade empresarial, na velocidade de inserção de novos produtos e serviços, fazendo com que reduza custos e aumente a capacidade dos processos operacionais.

Em outras palavras, a integração entre aplicações possibilita executar processos em sistemas especialistas, para transmitir ou requisitar informações sem que sejam de mesmo fornecedor.

2.4 *Web services*

Kuehne (2009) descreve os *web services* como uma implementação dos conceitos da arquitetura SOA, sendo basicamente um mecanismo para comunicação entre aplicações. Complementando Kuehne (2009), Duarte (2014), descreve um WS como sendo aplicações modulares que podem ser descritas, publicadas e até mesmos invocadas, sem que o solicitante se importe com a linguagem ou ambiente em que esta funcionalidade foi desenvolvida.

Essa implementação tem como seu principal objetivo interligar diferentes tipos de sistemas, tornando as aplicações cada vez mais modulares, encapsulando métodos que serão chamados remotamente por algum protocolo de comunicação, exemplo do protocolo HTTP, um integrante da família de protocolos de transporte TCP/IP, que é utilizado para realizar a comunicação do *browser* e o servidor web destino (KUEHNE, 2009).

Para Kuehne (2009), o ciclo de vida de um *web service* é basicamente constituído por quatro estados:

a) **Construção:** Fase inicial onde é contemplada o desenvolvimento e os testes da funcionalidade.

b) **Publicação:** Fase onde ocorre a publicação do serviço desenvolvido anteriormente.

c) **Execução:** Fase em que o WS já está publicado, na qual o consumidor já pode desfrutar de suas funcionalidades.

d) **Gerenciamento:** Fase de administração do serviço criado, na qual deve ser levado em consideração requisitos como segurança, disponibilidade, desempenho e qualidade.

Assim como um *web service* deve ser arquitetado de maneira inteligente, pensando no reúso e na flexibilidade, como mostrado anteriormente neste capítulo, ele também pode ser construído pensando em sua qualidade.

2.5 Qualidade de serviços (QoS)

Devido às características imprevisíveis da *web* e com a popularização dos *web services*, garantir a qualidade de serviços que são executados é uma tarefa muito complicada e deve ser tratada com grande importância pois interfere diretamente na satisfação do usuário (KUEHNE, 2009).

Para isso, um conjunto de atributos não funcionais são referenciados para definir a qualidade de serviços, detalhados na Tabela 1.

Tabela 1 – Atributos usados na definição do QoS.

Atributo	**Descrição**
Disponibilidade	Disponibilidade é um aspecto de qualidade de serviço no qual um *web service* está presente ou pronto para uso imediato, representado como uma porcentagem de tempo disponível de um serviço em um período de observação e está relacionado com sua confiabilidade. Para serviços frequentemente acessados um pequeno valor de período de observação fornece uma aproximação mais precisa para sua disponibilidade.
Confiabilidade	É a probabilidade que um serviço responderá no período de tempo esperado pelo consumidor e probabilidade do serviço executar sua funcionalidade de maneira correta. Este atributo caracteriza a confiança que um consumidor de serviços pode assumir no provedor de serviços.
Custo	O Valor monetário cobrado pelo provedor de serviços pelo acesso feito ao serviço.
Throughput	Representa o número de requisições que são finalizadas por determinado período de tempo. O aumento do *throughput* está diretamente

	relacionado ao tempo de resposta do sistema, pois quanto mais clientes ele conseguir atender em menos tempo, menor será a sobrecarga para receber novas requisições.
Tempo de resposta	É o tempo que uma requisição demora a partir do momento que ela é iniciada para ser atendida até o momento em que o requisitante recebe a resposta dessa requisição. Este tempo inclui toda a sobrecarga e possíveis atrasos que ocorram na rede.
Latência	Tempo gasto entre a chegada da requisição e o envio da resposta.
Desempenho	É a medida feita utilizando métricas como Tempo de Resposta e *Throughput*. O desempenho é considerado bom quando o *throughput* é alto e o tempo de resposta é baixo.
Segurança	A habilidade para garantir o não-repúdio das mensagens. Este atributo é importante para conseguir confiabilidade dos provedores de serviços.
Regulatoriedade	Aspecto de qualidade de serviço que mostra se um *web service* está em conformidade com as leis, regras, padronizações e acordos de níveis de serviços estabelecidos.
Robustez/Flexibilidade	A habilidade de um serviço em contornar entradas inválidas, incompletas e conflitantes e gerar o resultado correto.
Precisão	Avalia a probabilidade do serviço gerar erro.
Reputação	É medido pela avaliação que os clientes do serviço atribuem a ele. No final de cada acesso isso pode ser coletado por meio de uma enquete sobre satisfação do

Fonte: Adaptado em 19/05/2019 de Kuehne. Modelos e algoritmos para composição de Web Services com qualidade de serviço, 2009.

A qualidade dos *web services*, normalmente está associada *Service Level Agreement* – SLA, no qual é definido formalmente os níveis do serviço entre clientes e provedores.

2.6 Tipos de dados

Media types são informações presentes em todas as requisições feitas, para informar ao servidor qual o tipo de dados que o cliente espera receber. Um ambiente de *web service*, pode abranger diferentes tipos de *media types*, como o exemplo do *text/xml*, conhecido como XML, e o *application/json*, conhecido como JSON (SAUDATE, 2014).

2.6.1 XML

Extensible Markup Language – XML é uma linguagem de marcação extensível onde podemos expressar quase todo tipo de informação. Esse tipo de dado é muito parecido com o *HyperText Markup Language* – HTML, possuindo somente um elemento principal, podendo ser nomeado com qualquer nome. Nesse tipo de dado é possível criar prólogos que são interpretados pelo XML mas não fazem parte do dado, normalmente utilizado para definir versão e codificação do arquivo e tem o formato "<? instrução ?>". Além dos prólogos, podemos definir *tags* com atributos e valores no seguinte formato "<tag atributo=valor></tag>" (SAUDATE, 2014).

2.6.2 JSON

Como o XML é muito complexo para ser lido por humanos, o *JavaScript Object Notation* – JSON representa uma alternativa de formato de dados, representando uma forma de serializar objetos e vetores para texto simples, que poderá ser lido por qualquer um (BALDUINO, 2013).

Saudate (2014) afirma que uma das principais melhorias do modelo JSON em relação ao modelo XML é o seu tamanho reduzido sem perder nenhum tipo de informação. Atendendo ao modelo onde um objeto pode possuir zero ou mais membros; um membro possui zero ou mais pares e zero ou mais membros; um par possui uma chave e um valor; e ainda um membro pode ser um *array*, acaba se tornando uma ótima alternativa como modelo de dados.

2.7 Tipos de *Web services*

Conforme já mencionado, um WS trata-se de uma funcionalidade modular, auto suficiente que pode ser invocada através de uma rede. Correia (2015) complementa que *web service* é uma solução utilizada principalmente para integrações entre sistemas sendo sua

implementação baseada em tecnologias e protocolos onde os usuários não necessitam conhecer sua estrutura ou até mesmo a linguagem de programação em que a mesma foi desenvolvida.

2.7.1 SOAP

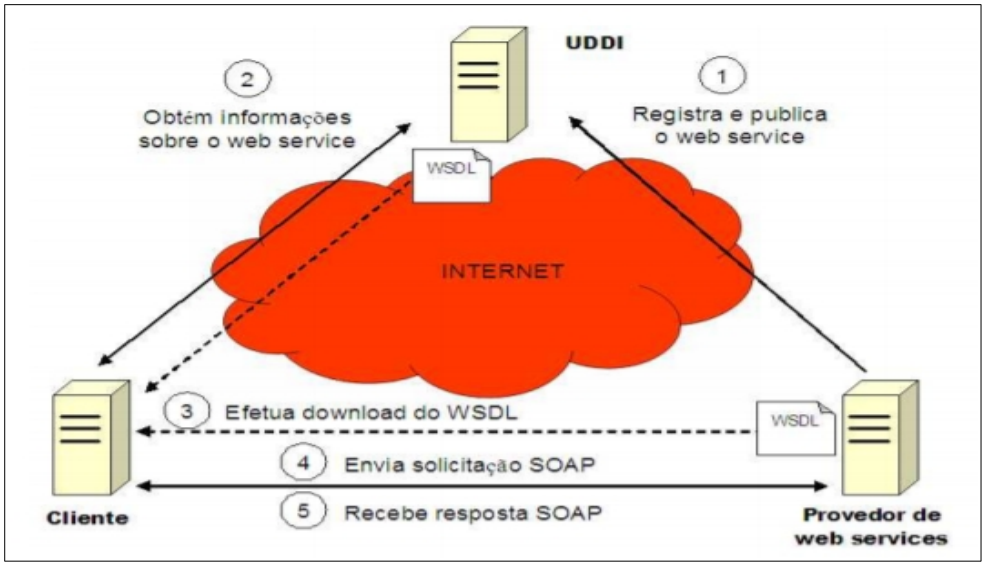
O *Simple Object Access Protocol* – SOAP é um protocolo simples para comunicação de um *web service*. Sua estrutura é baseada em um documento do tipo XML, regido pela W3C que estabelece convenções para a sua composição, que é baseado em um modelo de envelope (W3C, 2000).

A descrição de um serviço SOAP é padronizada pela linguagem *Web Service Description Language* – WSDL, ao contrário do REST que não possui padronização. Nesse XML é informado ao cliente como utilizar o serviço fornecido, o conteúdo é referente ao nome dos métodos, parâmetros e do tipo de dados e retorno de dados do serviço (W3C, 2000).

Universal Description Language – UDDI, é o serviço de armazenamento de serviços SOAP, onde organizações podem publicar suas API's e consultar outros serviços, de forma padronizada (CHAPPELL; JEWELL, 2002).

Conforme Chappell e Jewell (2002), uma das grandes promessas dos *web services* é a integração empresarial de forma automática, quando utilizadas as três tecnologias em conjunto tal promessa é comprida, onde cada pedaço da tecnologia deve descobrir, acessar, e integrar as informações sem intervenção humana. Essa integração dinâmica dos três elementos SOAP, WSDL e UDDI é apresentada na Figura 2.

Figura 2 – Representação da comunicação feita por SOAP



Fonte: Gomes e Jandl. Web Services SOAP e REST. Intellectus No 6, 2009.

De acordo com W3C (2000), só existe um tipo de formato de dados possível em WS baseados em SOAP, o XML. Estruturado de uma forma padronizada através de três elementos principais formados pelo envelope, o cabeçalho e o corpo, sendo que o cabeçalho e o corpo ficam dentro do envelope.

Toda a informação necessária para um *web service* SOAP está embutida no envelope, assim todo o processamento feito é baseado no arquivo XML, dessa maneira pode ser transportado por vários meios, independentemente do tipo de protocolo, ou seja, podem trafegar sobre HTTP, *Simple Mail Transfer Protocol* – SMTP, *File Transfer Protocol* – FTP, *Transmission Control Protocol* – TCP puro ou, qualquer outro protocolo, ao contrário do REST que trafega apenas por HTTP. (COUTINHO, 2013).

Conforme visto na Figura 3, a tag <S:Envelope é a raiz do XML, incorporando o corpo e o cabeçalho do serviço. O cabeçalho, <S:Header, contém a descrição de elementos para a comunicação do serviço, tais como, destinatário, remetente. Já na tag <S:Body, considerada o corpo de todo o envelope SOAP, contém todos os dados pertinentes ao conteúdo da mensagem enviada (W3C, 2000).

Figura 3 – Exemplo de envelope SOAP de solicitação

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header>
    </S:Header>
    <S:Body>
        <ns2:listarConta xmlns:ns2="http://webservices.soap.ws.policamp/">
            <nDoc>2</nDoc>
        </ns2:listarConta>
    </S:Body>
</S:Envelope>
```

Fonte: Adaptado em 07/05/2019 de Gomes e Jandl. Web Services SOAP e REST. Intellectus No 6, 2009.

Como o SOAP interage em um formato simples de r*equest/response*, a leitura do envelope *request* é feita pelo servidor destino, o mesmo será processado de alguma forma, e deverá retornar uma resposta ao seu cliente. Esta resposta será um novo envelope do padrão SOAP, destinado ao cliente (GOMES; JANDL, 2009).

A Figura 4 mostra que em seu *response*, esse envelope deve conter o elemento <return>, que é responsável por retornar o resultado processado pelo *web service*. Ainda como elementos secundários do SOAP, podemos encontrar o *fault*, elemento responsável pelo tratamento de erros (W3C, 2000).

Figura 4 – Exemplo de envelope SOAP de resposta

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:listarContasResponse xmlns:ns2="http://webservices.soap.ws.policamp/">
            <return>
                <cedente>SABESP</cedente>
                <descricao>CONTA DE AGUA</descricao>
                <dpagto></dpagto>
                <dvenc>10/11/2008</dvenc>
                <ndoc>2</ndoc>
                <valor>34.5</valor>
            </return>
        </ns2:listarContasResponse>
    </S:Body>
</S:Envelope>
```

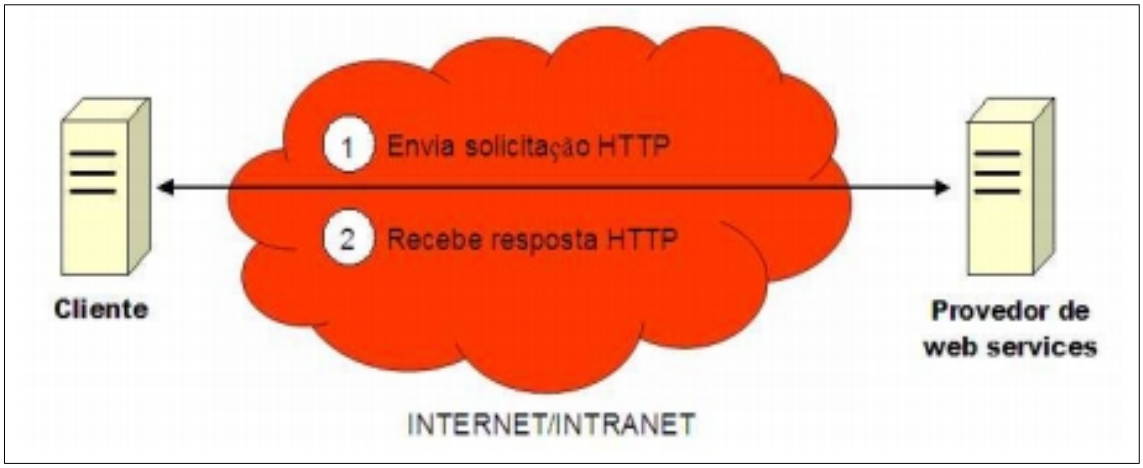
Fonte: Adaptado em 07/05/2019 de Gomes e Jandl. Web Services SOAP e REST. Intellectus No 6, 2009.

2.7.2 REST

Proposto por Roy Fielding em meados dos anos 2000 na sua tese de doutorado, o *Representional State Transfer* – REST também pode ser caracterizado como um padrão simples para comunicação de um *web service* (GOMES; JANDL, 2009). Saudate (2014) complementa que este padrão foi criado para ser uma alternativa mais simples e leve, do que os padrões já estabelecidos anteriormente (SOAP).

O REST, em seu conceito, não é considerado como um protocolo, mas sim um conjunto de ideais e conceitos que possibilitam sua utilização como padrão de comunicação entre aplicações. Sua comunicação é feita através do protocolo de transferência de dados *HyperText Transfer Protocol* – HTTP, conforme representada na Figura 5, e de *Uniform Resource Locator* – URL, endereço virtual que é utilizado como método de acesso para as funcionalidades criadas (SAUDATE, 2014).

Figura 5 – Representação da comunicação feita por REST



Fonte: Gomes e Jandl. Web Services SOAP e REST. Intellectus No 6, 2009.

Segundo Saudate (2014), até a versão 1.1 do protocolo HTTP, existem oito métodos ou verbos, especificados em cada uma das requisições feitas pelo cliente. Eles são responsáveis por indicar a ação realizada para um certo recurso criado, porém, somente com quatro desses métodos é possível algum tipo de interação, ou seja, provocam alterações nos recursos identificados pela URL, sendo eles:

a) **GET:** Recupera os dados identificados pela URL.

b) **POST:** Cria um novo recurso.

c) **PUT:** Atualiza um recurso.

d) **DELETE:** Apaga um recurso.

Quando o cliente faz uma requisição HTTP para o servidor, a mesma deve retornar um código de status, informando o resultado da chamada. Esses códigos são divididos em cinco famílias: 1xx, 2xx, 3xx, 4xx e 5xx, onde os seus principais retornos estão representados pela Tabela 2 (SAUDATE, 2014).

Tabela 2 – Códigos de retorno HTTP.

Código	**Significado**
200	Sucesso
201	Criado
204	Sem conteúdo
400	Requisição inadequada
401	Não autorizado
403	Acesso negado
404	Não encontrado
412	Falha na pré-condição
500	Erro Interno no servidor
501	Não implementado

Fonte: Adaptado em 19/04/2019 de Gomes e Jandl. Web Services SOAP e REST. Intellectus Nº 6, 2009.

Diferente do protocolo SOAP que trabalha somente com um formato de dados, o REST pode utilizar diversos formatos, sendo que os mais utilizados são o JSON e o XML. A Figura 6 mostra o envelope de uma solicitação onde as informações transmitidas serão feitas no formato XML, bastando somente alterar a informação *Content-Type* presente no *header* do envelope para informar o formato dos dados enviados ao protocolo (GOMES; JANDL, 2009).

Figura 6 – Exemplo de envelope HTTP de solicitação

```
GET http://localhost:8080/ContasAPagarWSRest/resources/pagaments/2/
Protocol HTTP/1.1
Host localhost:8080
User-Agent Mozilla/4.0
Accept application/xml
Accept-language pt-br
Accept-encoding gzip, deflate
Connection Keep-Alive
```

Fonte: Adaptado em 07/05/2019 de Gomes e Jandl. Web Services SOAP e REST. Intellectus No 6, 2009.

Conforme explica Gomes e Jandl (2009), a requisição feita pelo cliente sempre haverá um retorno do servidor, pois um *web service* do padrão REST se baseia exclusivamente nos recursos do protocolo HTTP, que trabalha sempre no formato *request/response*.

Neste padrão, o *web service* também irá ler o conteúdo que está dentro do envelope, porém pelo protocolo HTTP. Fará um processamento desta requisição, e retornará uma resposta ao cliente no mesmo padrão solicitado, colocando o resultado dentro de outro envelope, representado pela Figura 7 (GOMES; JANDL, 2009).

Figura 7 – Exemplo de envelope HTTP de resposta

```
<?xml version="1.0" encoding="UTF-8"?>
<pagament uri="http://localhost:8080/ContasAPagarWSRest/resources/pagaments/2/">
    <cedente>SABESP</cedente>
    <descricao>CONTA DE AGUA</descricao>
    <dpagto></dpagto>
    <dvenc>10/11/2008</dvenc>
    <ndoc>2</ndoc>
    <valor>34.5</valor>
</pagament>
```

Fonte: Adaptado em 07/05/2019 de Gomes e Jandl. Web Services SOAP e REST. Intellectus No 6, 2009.

Caso ocorra algum erro no processamento ou até mesmo erro de acesso não autorizado, este será tratado pelo status enviado no *header* do protocolo, conforme já mencionados pela Tabela 2.

2.8 *Cache*

Em seu conceito o *Cache* pode ser considerado como o armazenamento físico de dados já processados, sejam eles em memória ou até mesmo em disco. Geralmente tem aspecto temporário e o acesso é efetivamente rápido. Seu funcionamento é bastante simples, após o processo de requisitar uma informação, esta precisa ser carregada ou processada, então, o sistema deve automaticamente salvar este dado, para que em uma próxima requisição semelhante, este seja apenas recuperado (AMAZON, 2019).

Amazon (2019) complementa que quando este conceito é utilizado toda a carga que iria para um certo processamento de uma lógica de negócio ou até mesmo de uma consulta complexa, passe a ser somente uma carga de recuperação, pois os dados solicitados, de alguma forma já foram carregados ou processados.

Segundo Amazon (2019), entre os principais benefícios que o *cache* proporciona a um sistema, destacam-se os seguintes:

a) O aumento de performance.

b) A redução da carga no servidor *back-end*.

c) Simplicidade no acesso aos dados.

3 FERRAMENTAS RELACIONADAS

Neste capítulo serão apresentados alguns exemplos de ferramentas que buscam o gerenciamento de funcionalidades *web services* que se assimilam ao software desenvolvido pelo presente trabalho.

A primeira ferramenta abordada é chamada de “Application Manager”, sendo mantida e desenvolvida pelo grupo Zoho. Com o foco no monitoramento de aplicações e recursos de TI, monitorando tanto serviços do protocolo SOAP quanto do protocolo HTTP utilizando o REST, esta aplicação pode acompanhar o tempo de resposta e a disponibilidade de cada uma das funcionalidades gerenciadas (ZOHO, 2019).

Assim como na plataforma desenvolvida pelo presente trabalho, esta ferramenta também se propõem a entregar estatísticas de utilização das funcionalidades gerenciadas, porém não possibilita o cacheamento das requisições e não contém a possibilidade de conversão no formato dos *web services*.

Outra ferramenta que pode ser mencionada, refere-se a um trabalho acadêmico, que teve como seu objetivo a construção de um *plugin* para monitoramento e gerenciamento de *web services*. Este trabalho batizado de “Um Plugin para Monitoramento e Gerenciamento de Web Services baseados em SOAP”, intercepta *web services* somente do protocolo SOAP em aplicações desenvolvidas na linguagem de programação Java com a extensão da biblioteca JAX-WS (ROHLING; GARCIA, 2012).

Diferente da aplicação desenvolvida, está ferramenta necessita que seja feita uma implementação via código fonte de cada um dos provedores de serviços, para que comece a

monitorar as funcionalidades. Além disso, no caso de monitoramento de *web services* em aplicações distintas, essa informação não estará em um local centralizado, mas em cada um destes provedores.

Além das aplicações para monitoramento, existem ferramentas com a capacidade de converter certos formatos de *web service*, que é caso da plataforma “DreamFactory”. Com a possibilidade transformar qualquer serviço SOAP em REST e qualquer serviço REST em SOAP, os desenvolvedores não precisam implementar dois tipos de protocolo, o que torna o processo de desenvolvimento mais rápido, podendo assim integrar o maior número de funcionalidades ao seu sistema (DREAMFACTORY, 2019).

Todas as ferramentas analisadas buscam gerar estatísticas de uso para um melhor gerenciamento das funcionalidades disponibilizadas pela organização, porém algumas destas aplicações possuem maior versatilidade devido as suas funcionalidades extras. O presente trabalho propõe uma plataforma que seja capaz de gerenciar os *web services* em um local centralizado, melhorar as funcionalidades dos sistemas quando possível o uso do *cache* de dados e até mesmo transformar os serviços para o protocolo de escolha do requisitante.

4 METODOLOGIA

No processo de pesquisa científica é necessário definir e seguir uma metodologia, para que assim ela dite os caminhos que devem ser percorridos para este projeto. Neste capítulo serão apresentadas a metodologia utilizada no desenvolvimento do presente trabalho, junto a sua abordagem e as ferramentas que serão utilizadas na solução proposta.

4.1 Tipo de pesquisa

Conforme apresentado nos objetivos, o presente trabalho buscará construir uma ferramenta para gerenciar toda a comunicação entre aplicações distintas feitas por *web services*. Para poder chegar a este resultado, é inevitável que seja feita uma pesquisa exploratória dos temas relacionados.

> Todas as ciências caracterizam-se pela utilização de métodos científicos; em contrapartida, nem todos os ramos de estudo que empregam estes métodos são ciências. Podemos concluir que a utilização de métodos científicos não é da alçada exclusiva da ciência, mas não há ciência sem o emprego de métodos científicos (Marconi; Lakatos, 2003. p 20).

Para Marconi e Lakatos (2010), o estudo exploratório pode ser realizado através de documentos, e basicamente é formulado por três objetivos básicos, que são: a) Desenvolver hipóteses; b) Aumentar o conhecimento sobre o assunto a ser estudado; c) Identificar se é fato ou fenômeno.

Segundo, Prodanov e Freitas (2013), quando a pesquisa estiver na fase preliminar, o método exploratório proporciona mais informações sobre o assunto estudado, e em geral, assume as formas de pesquisa bibliográfica e estudo de casos.

Para que a plataforma proposta estivesse em conformidade com os conceitos já estabelecidos, foi necessário utilizar o método de pesquisas bibliográficas. Gil (2002) coloca que na pesquisa bibliográfica são buscadas premissas já referenciadas em bibliografias a respeito do tema da pesquisa, fazendo-se uma análise crítica quanto à situação atual. Complementando Gil (2002), Marconi e Lakatos (2010), afirmam que a pesquisa bibliográfica tem quatro principais fontes bibliográficas sendo elas: imprensa escrita, meios áudios visuais, materiais cartográficos e publicações.

Para poder avaliar o presente trabalho, foram utilizados dois tipos de abordagens diferentes, a experimental, quando da implementação da ferramenta em uma universidade local, e uma pesquisa com questões qualitativas e quantitativas feita com os gestores do setor de desenvolvimento desta organização.

A abordagem experimental é de natureza investigatória, que se apoia somente em experiências vividas na qual o pesquisador controla as variáveis independentes e observa os resultados nas variáveis dependentes (KAHLMEYER-MERTENS; FUMANGA; TOFFANO; SIQUEIRA, 2007).

Segundo Marconi e Lakatos (2010), a pesquisa experimental utiliza projetos que incluem os seguintes fatores: grupo de controle, seleção de amostra por técnica probabilística e manipulação das variáveis independentes.

A produção deste trabalho não visa analisar e quantificar um grande número de dados retirados de uma massa de usuários. Porém, para poder avaliar o seu uso, também será utilizada uma abordagem de pesquisa qualitativa. Prodanov e Freitas (2013), consideram que a pesquisa qualitativa tem relação direta entre o real e o sujeito tendo vínculo a qualidade do assunto abordado e não em dados numéricos.

> A pesquisa qualitativa é particularmente útil como uma ferramenta para determinar o que é importante para os clientes e porque é importante. Esse tipo de pesquisa fornece um processo a partir do qual questões-chave são identificadas e perguntas são formuladas, descobrindo o que importa para os clientes e porquê (Eduardo Moresi, 2003. p 69).

Os resultados foram obtidos através desta pesquisa e utilizados para validar a ferramenta desenvolvida neste trabalho. Após análise dos resultados, as devidas conclusões são apresentadas.

4.2 Ferramentas utilizadas

Nesta seção são elencadas as principais ferramentas e tecnologias utilizadas no desenvolvimento do presente trabalho.

4.2.1 Apache

Atualmente mantido pela fundação Apache Group, o Apache é um servidor HTTP muito popular entre as aplicações *web*. Com uma configuração simples e rápida, ele é responsável por controlar com segurança todas as transações entre cliente e servidor. Fatores como ser um software livre e suportar diferentes linguagens de programação, como PHP, Java, Python e outras, foram essenciais para a popularização do *software* (MARCELO, 2005).

Atualmente o Apache encontra-se na versão 2.4.39, lançada em 2018, e já faz parte de aproximadamente 43% dos ambientes web da internet (*Usage statistics and market share of Apache for websites*, W3Techs, 2019).

4.2.2 HTML e CSS

Toda a *web* é formada por três pilares básicos: 1) Esquema de nomes para localização das fontes de informações, as URI; 2) Protocolo de acesso para acessar essas fontes de informações, o HTTP; 3) Uma linguagem para fácil navegação entre as fontes de informação, o HTML. O *Hypertext Markup Language* – HTML é uma linguagem baseada em marcação, utilizada como padrão para interfaces de softwares na internet, que são facilmente interpretadas pelos navegadores e exibidas aos usuários (EIS; FERREIRA, 2012).

O *Cascading Style Sheets* – CSS é responsável por toda a estilização da informação mostrada ao usuário, desde cor, o tamanho e até mesmo os comportamentos dos componentes gerados pelo HTML. Em poucas palavras ele é o principal responsável pela interface que é vista pelo usuário que está acessando a aplicação (EIS; FERREIRA, 2012).

4.2.3 JavaScript

Uma linguagem de programação criada em 1995, que é fortemente usada no *front-end* (*processada pelo navegador*), sendo uma das linguagens mais populares do mundo. Disponibilizada em quase todos os navegadores da atualidade, pode ser utilizada para poder manipular elementos HTML sem que necessite de um processamento do lado do servidor, tornando assim a interface do sistema muito mais interativa para o usuário (CROCKFORD, 2008).

4.2.4 jQuery

Uma biblioteca rápida, pequena e que com muitos recursos para a linguagem JavaScript. Utilizada para o desenvolvimento *front-end*, o jQuery foi criado com a intenção de simplificar e agilizar o desenvolvimento das principais interações que uma aplicação *web* deve ter, conseguindo até mesmo generalizar algumas funcionalidades que são diferentes de cada navegador e que ocasionariam alguma incompatibilidade com o usuário final (BALDUINO, 2013).

4.2.5 PHP

Uma linguagem de programação *back-end* (processada pelo servidor) muito popular, rápida e flexível. Utilizada especialmente para o desenvolvimento *web* como sites e blogs, o PHP passou por muitas evoluções ao longo do tempo (PHP, 2019).

No princípio o PHP era chamado de *Personal Home Page Tools / Forms interpreter* e era composto por um conjunto de *scripts* desenvolvidos em linguagem C, para criação de páginas dinâmicas. Após anos de desenvolvimento e evolução em sua estrutura, o PHP passa a ser chamado de *Hypertext Preprocessor*. Com uma API muito simplificada e bastante dinâmica, a linguagem de programação acaba ganhando uma nova realidade e um propósito mais amplo (DALL'OGLIO 2016).

Segundo Dall'Oglio (2016) estima-se que seu uso ultrapasse os 80% da grande parte dos servidores da *web*, o que a torna a linguagem de programação mais utilizada para o desenvolvimento *web*.

4.2.6 Adianti Framework

O Adianti é um *framework* de código aberto para linguagem de programação PHP, e é baseado no padrão de projetos *Model View Controller – MVC*. Este padrão busca uma separação das camadas lógicas da aplicação, o que deixa o código gerado pelo desenvolvedor muito mais organizado e estruturado (DALL'OGLIO 2015).

Segundo Dall'Oglio (2015) este FW foi criado com enfoque em aplicações de negócio, ou seja, não prima por configurações visuais a seus componentes, mas sim em funcionalidades para o *back-end*, o que não impede de ser usado para tal. Feito para ser pequeno e fácil de ser utilizado, ele provê de um conjunto de classes que busca reduzir drasticamente o custo de desenvolvimento de um projeto, já oferecendo alguns componentes e até mesmo *templates* de alto nível para que o desenvolvedor não inicie o projeto do zero.

4.2.7 PostgreSQL

PostgreSQL é um Sistema Gerenciador de Banco de dados – SGBD objeto relacional, criado em 1986 e é responsável por armazenar e gerir todas as informações de um sistema de informação. Além de ser conhecido por ser muito estável e confiável, este SGBD tem o seu

código fonte aberto, o que possibilita um maior crescimento pois pode ter o apoio de sua comunidade de desenvolvedores (MILANI 2008).

Compatível com diversos sistemas operacionais, o PostgreSQL utiliza a linguagem *Structured Query Language* – SQL e oferece suporte a transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), entre outros recursos que atestam sua qualidade como banco de dados (MILANI 2008).

5 ESPECIFICAÇÃO DA PLATAFORMA

O objetivo do presente capítulo é apresentar a plataforma desenvolvida, desde a sua visão geral, à definição de seus requisitos, modelo de casos de uso, modelo de estados, modelo relacional e os protótipos de interface.

5.1 Visão geral da implementação

Como já abordado anteriormente, o objetivo da plataforma desenvolvida é ser uma ferramenta *web* que consiga auxiliar, padronizar e melhorar a comunicação entre aplicações, batizada de DataCache.

Nessa plataforma, os usuários podem cadastrar informações pertinentes a sistemas e até mesmos serviços, que serão executados em algum momento nos padrões de comunicação REST ou SOAP. Nestes cadastros é possível definir um certo nível de permissões, no qual uma funcionalidade pode estar disponível para uma ou mais aplicações.

O propósito da solução é atuar como um intermediário na comunicação entre duas aplicações, podendo assim separar as camadas entre aplicação final e os fornecedores de dados. Isso permite um melhor gerenciamento das requisições, pois quando uma requisição é feita através do DataCache, todo o registro desta transação é gravado em seu banco de dados, podendo assim gerar estatísticas dos serviços gerenciados. Para melhor entendimento, o fluxo da aplicação está dividido em três partes, sendo elas:

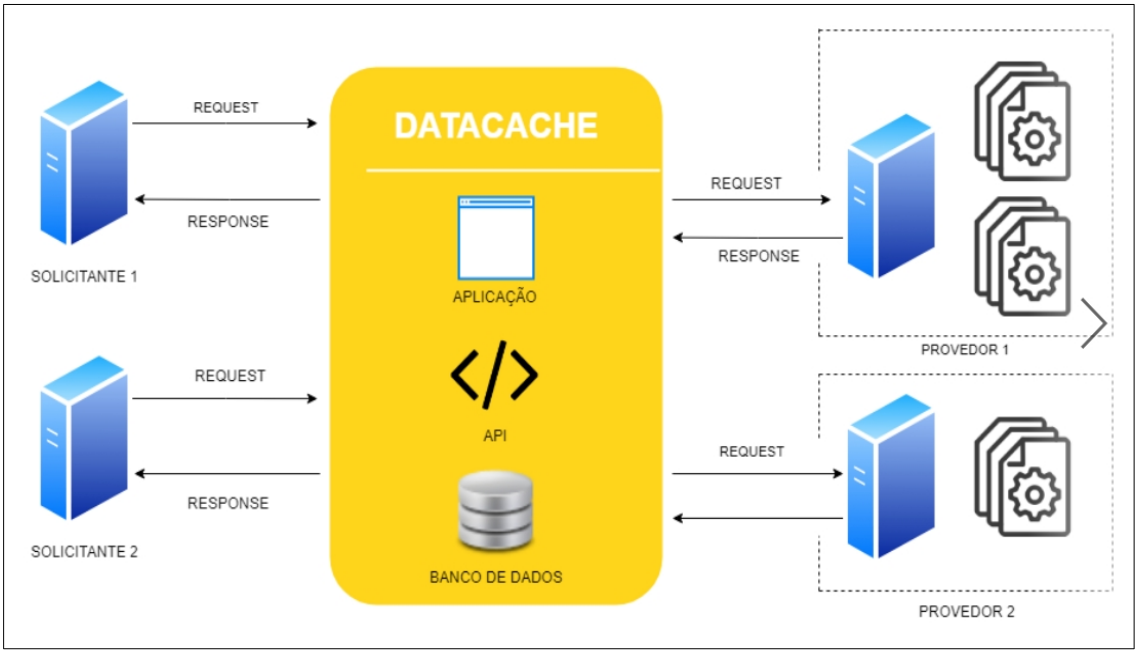
a) **Solicitantes:** Sistemas que necessitam utilizar algum recurso de um provedor.

b) **DataCache:** Solução proposta que separa as camadas entre solicitante e provedor de serviços.

c) **Provedores:** Sistemas provedores de serviços.

A Figura 8 representa a arquitetura do sistema, onde aplicações distintas solicitam alguma informação de um determinado serviço. Porém, em vez de fazer essa chamada direto ao provedor deste serviço, esta será feita através do DataCache, onde é possível ter um melhor controle do que acontece com cada requisição, sendo ela um erro ou um sucesso.

Figura 8 – Arquitetura da aplicação DataCache

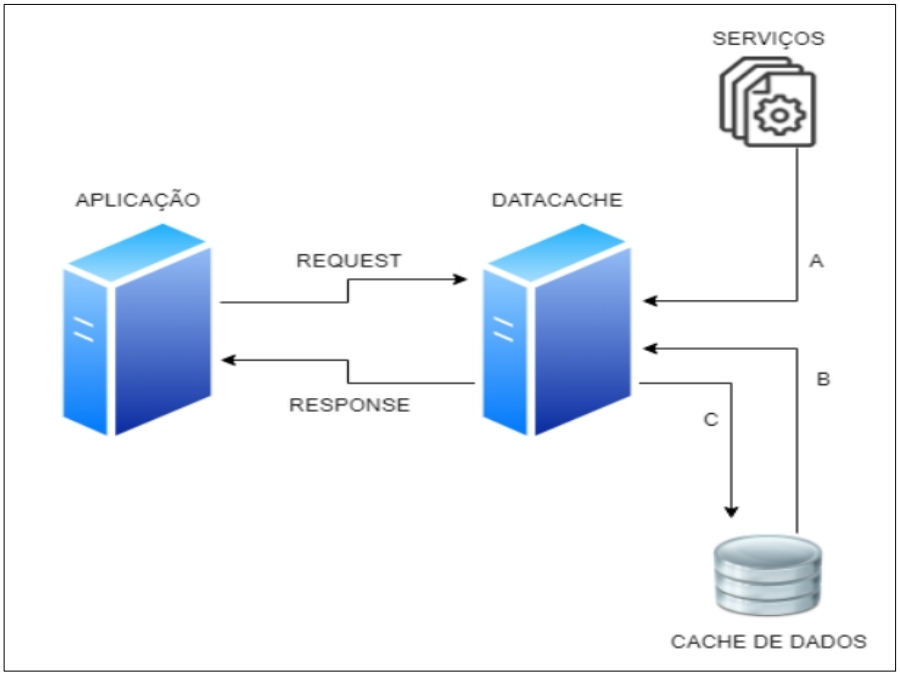


Fonte: Elaborado pelo autor, 2019.

A arquitetura da solução gerencia todo o fluxo de requisições por meio da integração de uma Aplicação, uma API e um Banco de Dados. A aplicação é responsável pelos cadastros feitos pelos gerenciadores dos sistemas, no qual devem fornecer os dados pertinentes às aplicações e seus serviços disponibilizados com suas permissões de execução, além de poder informar que certas funcionalidades utilizam o *cache* dos dados, definindo uma vida útil para que estes expirem. Já a parte da API é responsável por receber todas as requisições solicitadas pelas aplicações. O banco de dados é responsável por gravar os dados tanto do *request* quanto *response*, para que possam ser geradas estatísticas de usabilidade de cada um dos serviços adicionados na ferramenta.

Através da Figura 9, é possível exemplificar o fluxo de uma requisição onde, primeiramente, é realizada uma verificação se as informações estão no banco de dados do DataCache a fim de retornar os dados da requisição anterior, desde que estas estejam dentro do período de validade. Caso sejam encontrados dados válidos, estes são disponibilizados imediatamente (B) sem haver a necessidade de solicitar o serviço para o provedor novamente. Entretanto, caso não sejam encontrados será executado o serviço destino (A). Ao retornar as informações do serviço, as informações serão gravadas no banco de dados do DataCache (C) para que possam ser utilizadas posteriormente por outra requisição. Após este processo, elas passam a ser disponibilizadas para a aplicação que as requisitou.

Figura 9 – Fluxo de um *request/response* na aplicação



Fonte: Elaborado pelo autor, 2019.

Caso algum erro por parte do provedor da informação ocorra, o grupo responsável desta aplicação (definido no cadastro do sistema) será informado desta incoerência por e-mail com todos os *logs* da transação, sendo eles a funcionalidade executada com seus parâmetros e a exceção gerada, podendo assim fazer um plano de contingência para este problema.

5.2 Requisitos

Existem dois tipos de requisitos, os funcionais, que podem ser considerados como funcionalidades ou serviços que o sistema deve contemplar, e os não funcionais, que são uma necessidade funcional, ou seja, que definem como as funcionalidades serão implementadas.

5.2.1 Requisitos funcionais

Para que o desenvolvimento da ferramenta fosse considerado como concluído, alguns requisitos funcionais foram atendidos. A seguir (Quadro 1), serão elencados os principais grupos de requisitos funcionais da aplicação:

Quadro 1 – Requisitos funcionais

RF01 – Permitir cadastro de aplicações	**Prioridade**
Permitir cadastro de aplicações, podendo ser um tanto um provedor de serviços quanto um consumidor de serviços	Alta
RF02 – Permitir cadastro de serviços da aplicação	**Prioridade**
Permitir cadastro de serviços no qual uma aplicação pode ter vários serviços	Alta
RF03 – Permitir cadastro de permissões de serviços	**Prioridade**
Permitir cadastro de permissões de serviços no qual um serviço pode estar disponibilizado a uma ou mais aplicações consumidoras	Alta
RF04 – Desenvolver API de integração entre sistemas	**Prioridade**
Desenvolver API de comunicação da plataforma para que possam ser feitas as integrações entre os sistemas finais e os provedores de serviços e informações	Alta
RF05 – Permitir visualização de estatísticas de utilização de serviços	**Prioridade**
Disponibilizar um painel com estatísticas de utilização das funcionalidades e as suas respectivas taxas de sucesso e erro	Alta
RF06 – Alertas de erros em serviços	**Prioridade**

Permitir a visualização de alertas por meios de notificações no sistema e e-mails, quando ocorrer algum erro em uma funcionalidade cadastrada na plataforma	Média
RF07 – Cadastrar grupos de permissões aplicações	**Prioridade**
Manter o cadastro de permissões aos sistemas baseados nos grupos de usuários já cadastrados pelo administrador	Alta
RF08 – Permitir o cache de dados	**Prioridade**
Permitir que a plataforma utilize quando necessário o *cache* de dados de informações já processadas	Alta

Fonte: Elaborado pelo autor, 2019.

5.2.2 Requisitos não funcionais

A seguir (Quadro 2), serão dispostos os principais requisitos não funcionais que foram impostos para a plataforma:

Quadro 2 – Requisitos não funcionais

RNF01 – Utilizar HTML5, CSS3	**Prioridade**
Desenvolver a interface *front-end* seguindo os padrões HTML5 e CSS3	Alta
RNF02 – Utilizar PHP 7.1	**Prioridade**
Desenvolver utilizando PHP versão 7.1 ou superior como linguagem de programação *back-end*	Alta
RNF03 – Utilizar JavaScript	**Prioridade**
Desenvolver utilizando JavaScript como linguagem de programação *front-end*	Alta
RNF04 – Utilizar Adianti Framework	**Prioridade**
Utilizar Adianti Framework no desenvolvimento do *back-end*	Alta
RNF05 – Utilizar jQuery	**Prioridade**
Utilizar biblioteca JavaScript jQuery no desenvolvimento do *front-end*	Alta

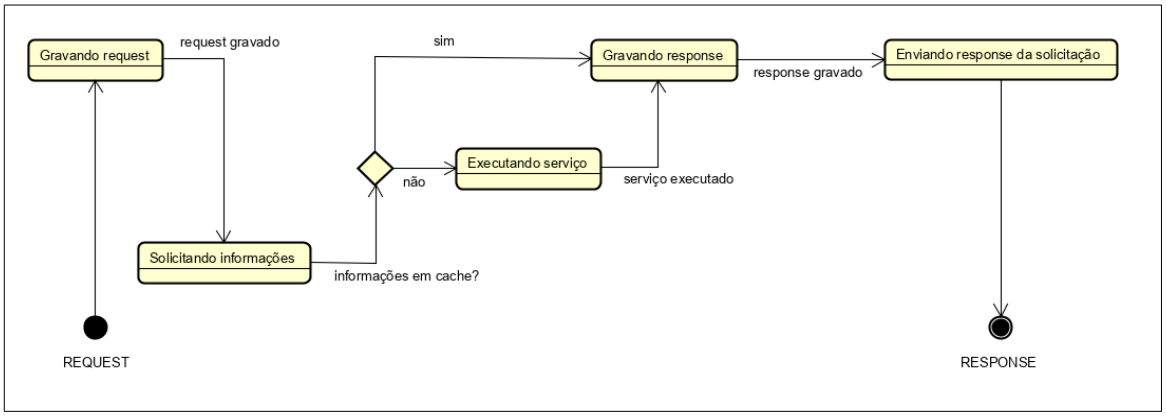
RNF06 – Utilizar PostgreSQL	**Prioridade**
Utilizar como banco de dados da aplicação o PostgreSQL versão 11	Alta
RNF07 – Compatível com navegador Google Chrome	**Prioridade**
Ser compatível com navegadores Google Chrome (versão 70 ou superior)	Alta

Fonte: Elaborado pelo autor, 2019.

5.3 Modelos de estados

A Figura 10 representa um diagrama de estados, no qual busca reproduzir o cenário de uma aplicação distinta solicitando um determinado serviço para a plataforma. Após receber a requisição desta solicitação, imediatamente a ferramenta DataCache grava em seu BD todas as informações necessárias referentes ao *request*. Após efetuar os registros, verifica se a solicitação já está ou não no *cache* de dados. Caso não encontre as informações necessárias em seu banco de dados, deverá executar o serviço requerido. Após obter os dados, sejam eles oriundos do *cache* de dados ou executando o serviço de um provedor, esses dados são armazenados no BD da plataforma, e só então enviados no *response* para o solicitante.

Figura 10 – Modelo de estados



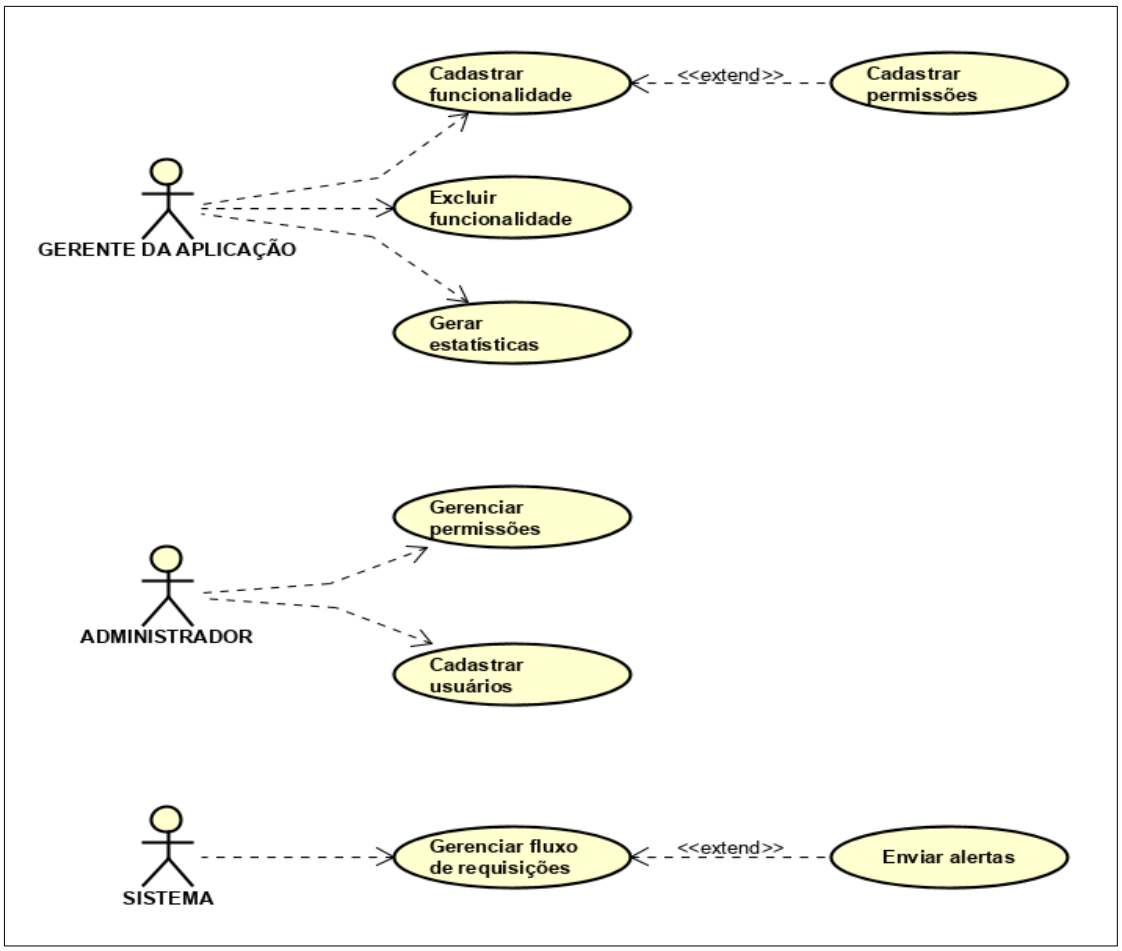
Fonte: Elaborado pelo autor, 2019.

5.4 Modelo de casos de uso

Para demonstrar os principais papéis dentro do DataCache será utilizado o diagrama de casos de uso. Conforme visto na Figura 11, a plataforma contém basicamente dois grupos de usuários, o grupo de gerente de aplicações, responsável por todos os cadastros e análises dos dados pertinentes às funcionalidades inseridas na ferramenta, e o grupo de administradores do sistema, que é responsável por gerenciar os acessos na plataforma proposta.

Além dos grupos de usuários podemos considerar que o sistema também é responsável por algumas tarefas na ferramenta, como por exemplo, gerenciar todo o fluxo de requisições e quando necessário alertar aos usuários sobre possíveis instabilidades.

Figura 11 – Modelo de casos de uso



Fonte: Elaborado pelo autor, 2019.

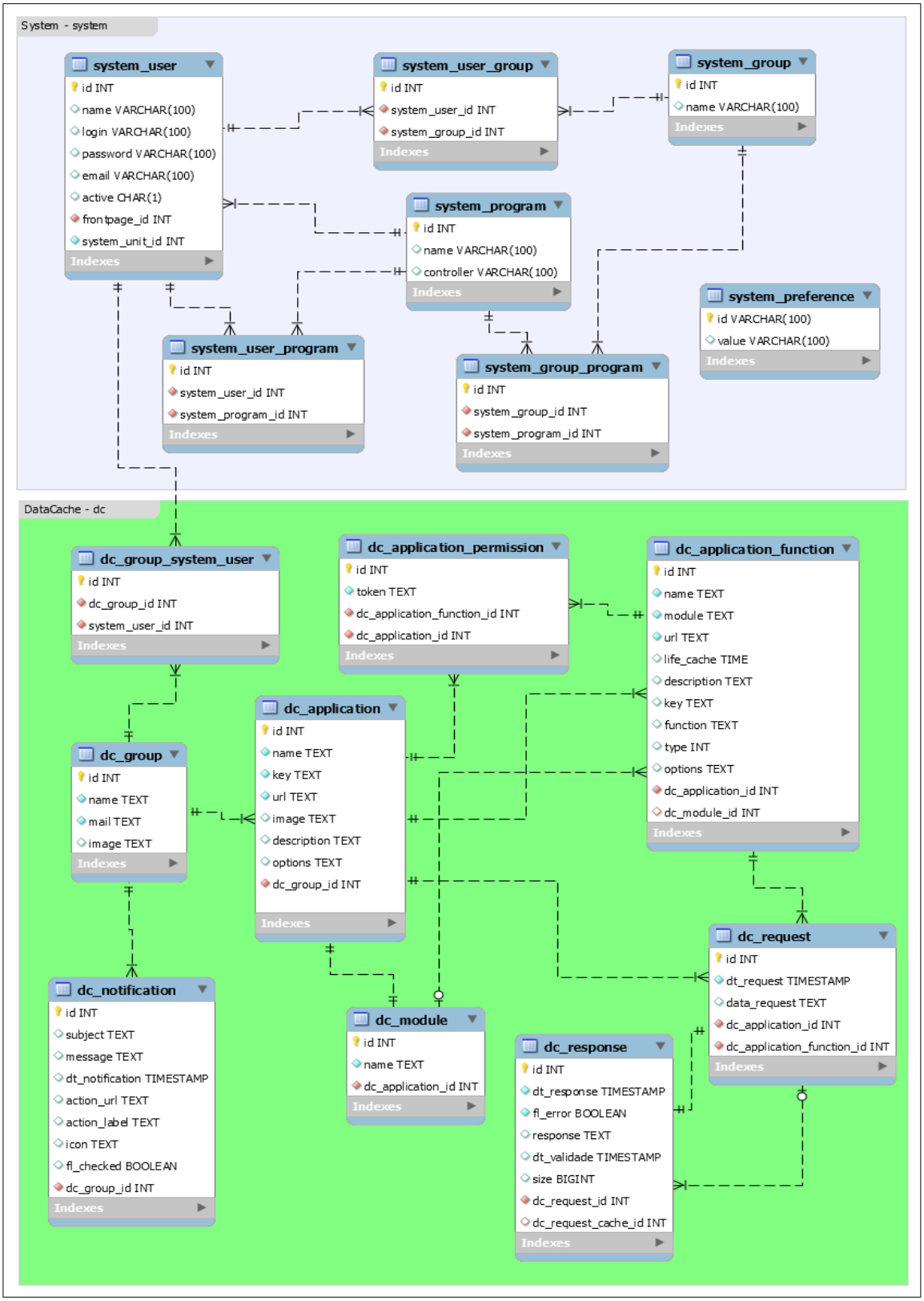
5.5 Modelo relacional

Para poder representar visualmente como as informações são armazenadas na aplicação, foram elaborados dois modelos relacionais. Um destes modelos é responsável por logs e registros vinculados com auditorias, e o outro é responsável por todos os cadastros pertinentes ao sistema, desde permissões de acessos até mesmo cadastro de aplicações e funcionalidades.

Na Figura 12, que representa o banco de dados do DataCache, o qual comporta duas estruturas de tabelas totalmente distintas em um mesmo banco de dados. Pode-se notar que há um padrão utilizado para separar estas tabelas, que visa determinar suas responsabilidades dentro do sistema. Para representar tabelas que compõem o cadastro de permissões do sistema, usuários, grupos páginas, dentre outros, é utilizado o prefixo “system_”. Quanto ao restante do DataCache, e as tabelas responsáveis por manterem os registros de aplicações e funcionalidades utilizados para as integrações, utilizam os prefixos são representadas por “dc_”.

Uma das principais tabelas deste modelo relacional é a tabela representada por “dc_application”, na qual serão atreladas todas as funcionalidades, permissões e requisições da aplicação. Como podemos ver no diagrama, existem duas tabelas que representam grupos, porém suas finalidades são diferentes. A tabela “system_group” é a responsável pelas permissões de acesso às páginas da aplicação, que será cargo dos administradores do sistema e dificilmente serão alteradas. Já a tabela “dc_group” será alimentada pelos gerentes de aplicações é a responsável por informar os usuários que pertencem a um certo grupo de desenvolvimento, na qual será utilizada para listar as aplicações que o usuário logado tem acesso.

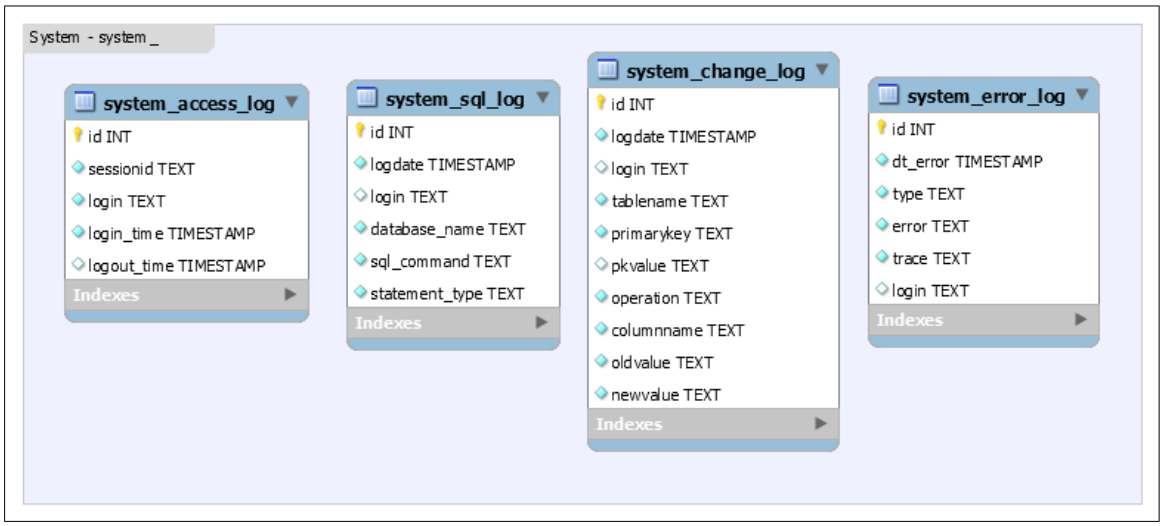
Figura 12 – Banco de dados – DataCache



Fonte: Elaborado pelo autor, 2019.

Na Figura 13 é apresentado o banco de dados Log que é responsável por armazenar todas as informações de auditoria e erros, no qual também utilizam o prefixo “system_” e são armazenados em um banco de dados diferente da aplicação devido ao seu constante crescimento. Para que não ocorra redundância nas informações, as únicas tabelas que não são contempladas nos logs de auditoria são as tabelas de “dc_request” e “dc_response” do banco de dados DataCache, onde somente são realizadas instruções de *insert* pela API da plataforma.

Figura 13 – Banco de dados Log



Fonte: Elaborado pelo autor, 2019.

Essa estrutura é um padrão utilizado pelo Adianti Framework no qual as tabelas “system_access_log”, “system_sql_log” e “system_change_log” são responsáveis por toda a auditoria do sistema na qual gravam acessos, alterações, e a instrução SQL gerada. Já a tabela “system_error_log” é uma implementação que em seu princípio deve guardar todas as exceções geradas internamente pela aplicação, permitindo que os administradores do sistema corrijam eventuais problemas da forma mais rápida possível.

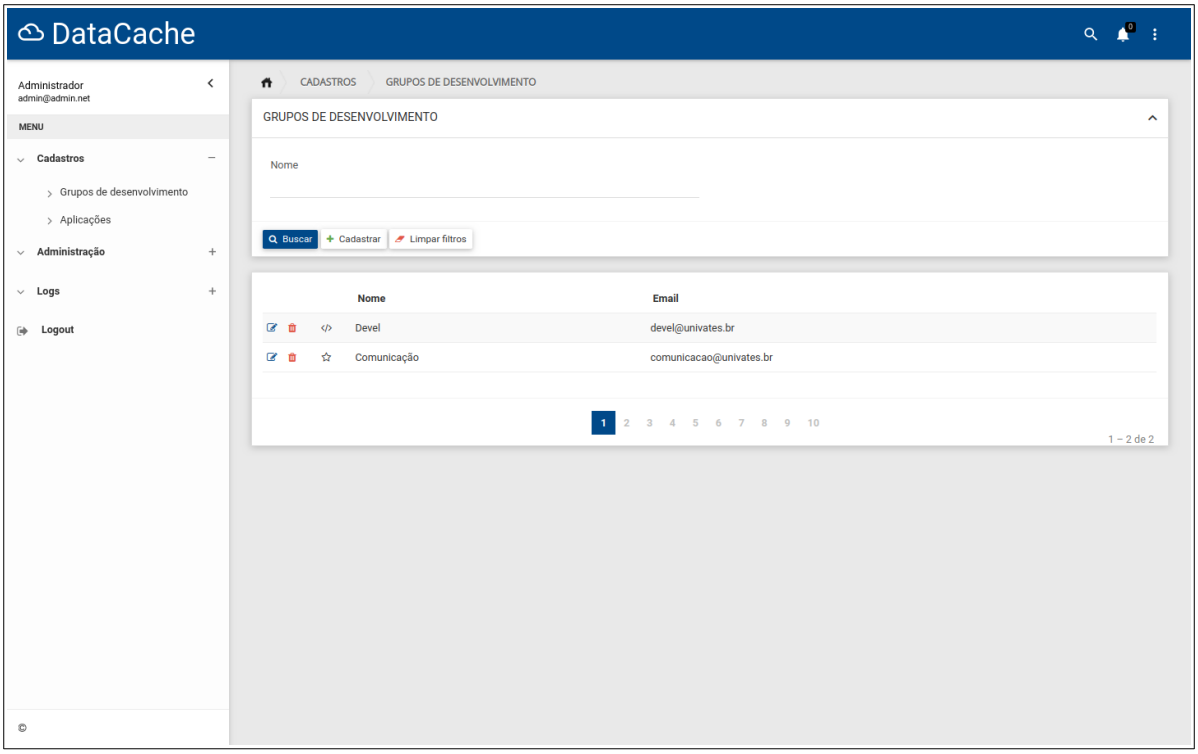
5.6 Interfaces da aplicação

Para demonstrar melhor o ambiente da aplicação, esta seção abordará as principais interfaces do sistema. As telas criadas foram feitas com base no *template* disponibilizado pela ferramenta Adianti Framework. Os dados contidos nas interfaces são fictícios e foram

substituídos para fins de documentação. A informação original validada dos testes está sendo preservada a pedido da empresa.

Na Figura 14 é demonstrado o cadastro de grupos de desenvolvimento, neste momento são informados o nome do grupo, o e-mail do grupo, um ícone para que este seja destacado nos demais locais do sistema e as permissões de acesso para os usuários que já necessitam visualizar as aplicações do grupo, onde estes usuários já devem estar previamente cadastrados no sistema.

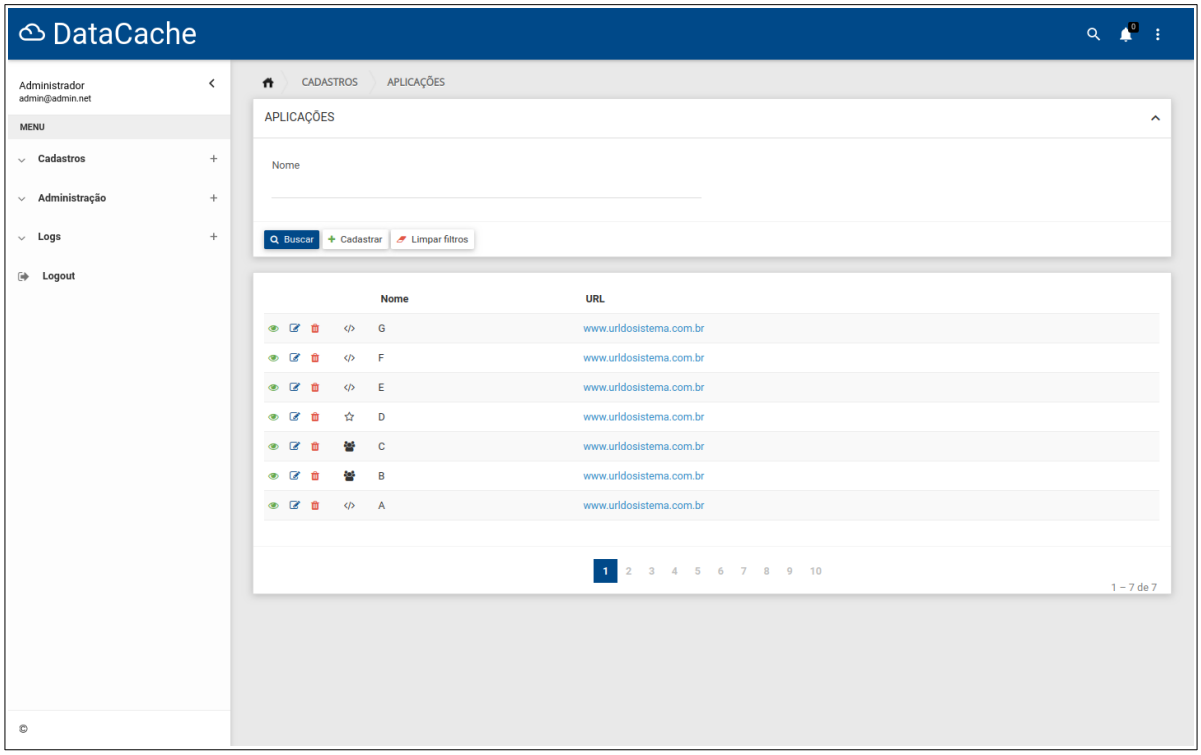
Figura 14 – Cadastro de grupos de desenvolvimento



Fonte: Elaborado pelo autor, 2019.

A Figura 15 exibe a listagem principal do sistema, onde é mostrado ao usuário logado na plataforma todas as aplicações no qual o mesmo tem acesso. A permissão e a disposição dos itens nesta visualização será feita a partir dos grupos do usuário e as classificações de cada uma das aplicações. Para que o usuário consiga visualizar uma aplicação, ele deve fazer parte do grupo em que ela pertence. Além das disso, a plataforma também disponibilizará notificações de instabilidades no qual serão visualmente encontradas no canto superior do sistema.

Figura 15 – Listagem de aplicações



Fonte: Elaborado pelo autor, 2019.

A Figura 16 mostra a principal tela do sistema, onde por meio desta é possível cadastrar todos os dados pertinentes de uma aplicação. Ao acessar essa página o usuário poderá alterar todos os cadastros vinculados a uma aplicação e seus respectivos módulos podendo adicionar funções e definir se usa ou não o *cache* dos dados e até mesmo alterar suas permissões de acesso. As ações desta página no que diz respeito a aplicação selecionada, como por exemplo, editar o cadastro da aplicação e adicionar uma nova funcionalidade, são agrupadas pelo menu à esquerda.

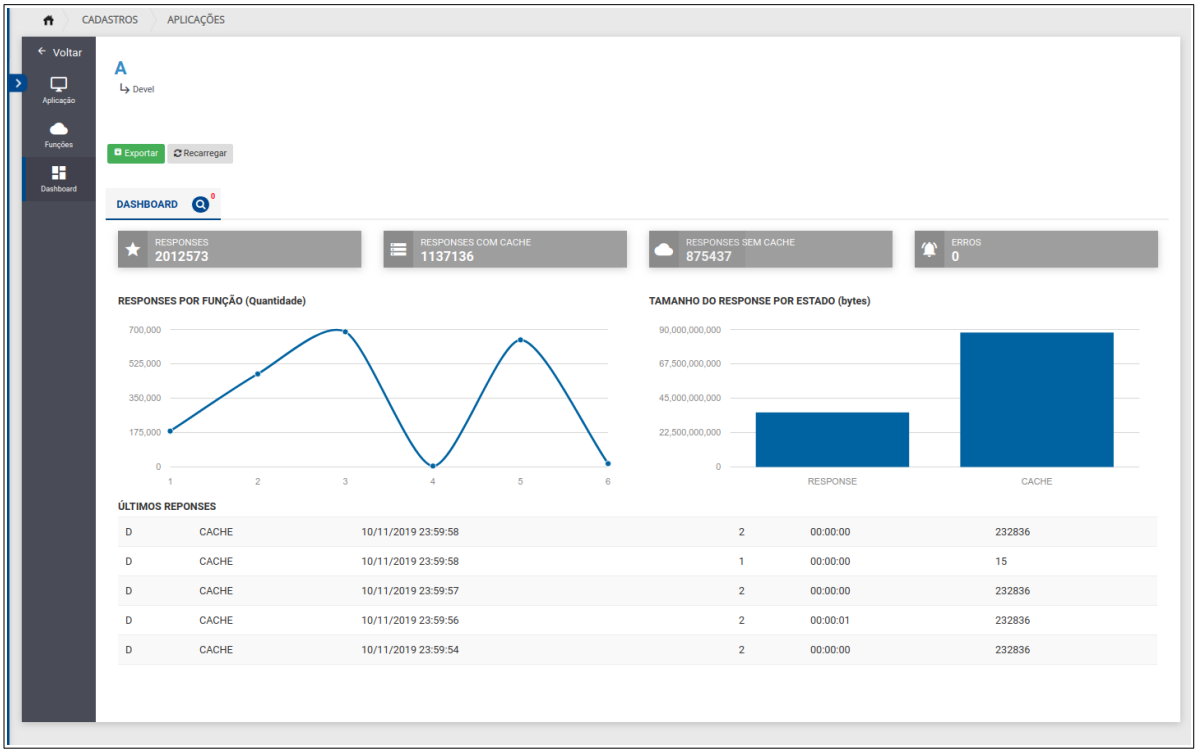
Figura 16 – Detalhes da aplicação

Fonte: Elaborado pelo autor, 2019.

Na Figura 17 é exibida a interface do Dashboard da aplicação. Esta interface permite uma visão macro das estatísticas de uso da aplicação que o usuário está acessando. Caso necessite de uma visão mais detalhada, pode ser gerado um relatório de todas as requisições pelo botão [Exportar].

Ao clicar no botão [Exportar], o sistema gerará um arquivo com a extensão “.cvs”, onde é possível buscar qualquer tipo de informação, contendo as seguintes colunas: requester_id, requester_name, provider_id, provider_name, provider_function_id, provider_function_name, request_id, request_dt_request, response_id, response_dt_response, response_delay, response_size, response_state.

Figura 17 – *Dashboard* da aplicação



Fonte: Elaborado pelo autor, 2019.

5.7 Processamento da API

A API do DataCache suporta dois tipos de protocolos para conexão de seus *web services*, sendo eles o SOAP e o HTTP utilizado via REST. Para poder suportar a estes tipos de serviços foram criadas duas classes distintas para receber estas requisições e uma classe do tipo *service*, que tem a responsabilidade de identificar um cache ativo.

Para exemplificar o fluxo interno da aplicação mencionado na Figura 9, onde temos um *request/response* gerenciado pela API do DataCache, a Figura 18 representa a classe TSoapServer que é responsável por receber todas as requisições do tipo SOAP. Todas essas requisições invocam o método “executaWS” que em sua lógica verifica o *token* utilizado como parâmetro e identifica qual a funcionalidade a ser executa e se o mesmo tem a devida permissão para executá-la.

Figura 18 – TSoapServer classe do serviço SOAP

```php
<?php
require_once 'init.php';

class TSoapServer extends SoapServer
{
    private static $database = 'datacache_api';

    public function __construct()
    {
    }

    public function executaWS($arguments, $token, $nocache = FALSE)
    {
    }
}

$server = new TSoapServer();
$server->handle();
?>
```

Fonte: Elaborado pelo autor, 2019.

A Figura 19 representa a classe TRestServer, responsável por receber as requisições do tipo REST. Diferente da classe TSoapServer onde é seguida uma implementação de uma classe nativa do PHP (SoapClient), que proporciona uma série de funcionalidades já prontas. Na classe TRestServer foi necessário o desenvolvimento de alguns métodos adicionais, como por exemplo o "handle" que é responsável por receber a requisição e identificar os parâmetros e após chamar o método principal da classe "executaWS".

Como o serviço REST pode trabalhar com diferentes formatos de dados, foi desenvolvido um método chamado "response", que após executar a funcionalidade requerida antes de retornar o *response* para o requisitante, o sistema executará este método podendo ou não transformar os dados para o formato utilizado, que é definido pelos administradores nas preferências do sistema.

Figura 19 – TRestServer classe do serviço REST

```php
<?php
require_once 'init.php';

class TRestServer
{
    private static $database = 'datacache_api';

    public function __construct()
    { ...
    }

    public function handle($params)
    { ...
    }

    private function response($result)
    { ...
    }

    public function executaWS($arguments, $token, $nocache = FALSE)
    { ...
    }
}

$server = new TRestServer();
print $server->handle($_REQUEST);
?>
```

Fonte: Elaborado pelo autor, 2019.

Tanto o serviço disponibilizado em REST quanto em SOAP devem executar as mesmas funcionalidades. Para isso foram desenvolvidas classes de serviço e adicionadas essas funcionalidades em comum, podendo assim, minimizar os trechos de código duplicados em todo o sistema.

A Figura 20 mostra a classe DCResponseService responsável por criar o registro do *response* no DataCache utilizando o método “createResponse” e buscar um *response* anterior válido pelo método “getResponseFromCache”. Para buscar e validar um *request* que ainda possa ser utilizado como base, para entregar o serviço de forma mais rápida e sem consumir os recursos do servidor destino utilizando o *cache*, destacam-se as seguintes regras:

a) O *request* realizado deve ser da mesma funcionalidade e conter os mesmos parâmetros.

b) O seu *response* não pode ser um *cache* ou um erro.

c) O tempo de validade do *request* ainda deve estar válido.

Figura 20 – DCResponseService classe de serviço

```php
<?php

class DCResponseService
{
    public static function createResponse(DCRequest $request, $response, $fl_error, $response_cache = NULL)
    { ...
    }

    public static function getResponseFromCache($request)
    { ...
    }
}
```

Fonte: Elaborado pelo autor, 2019.

6 RESULTADOS E DISCUSSÕES

Neste capítulo serão apresentados os resultados obtidos e os métodos de validação da plataforma. A validação deste trabalho foi feita a partir de uma implementação real da ferramenta e através da análise de uma pesquisa com os gestores da área de desenvolvimento da organização.

6.1 Análise da implementação

Para poder validar o seu uso e realizar uma análise dos dados obtidos, a plataforma DataCache foi instalada em um dos servidores da organização. Para que não fosse prejudicada nenhuma operação, foram utilizados nomes fictícios para os sistemas e suas funções, mantendo assim a privacidade da organização. Ao total foram monitoradas oito rotas diferentes de *web services*, servindo a um total de sete sistemas durante um período de 40 dias.

O Quadro 3 reapresenta uma visão geral dos sistemas que são provedores de serviços e as informações sobre suas funcionalidades. Neste quadro temos a descrição de cada uma das funcionalidades possibilitando um maior entendimento do cenário proposto. Além disso temos a informação sobre utilização ou não o *cache* de dados, representada pela coluna "Validade do *cache*", que contém a quantidade de minutos que o *request* ficará válido para os critérios da busca.

Quadro 3 – Provedores de serviços

Sistema	**Função**	**Descrição**	**Validade do *cache***
A	1	Busca próxima aula	360
A	2	Foto pessoa	1440
A	3	Tipos aluno	1440
A	4	Histórico e notas	360
A	5	Frequência e notas	0
A	6	PDF do histórico da graduação	360
B	1	Consultar CEP	10080
C	1	Consultar CEP	10080

Fonte: Elaborado pelo autor, 2019.

Podemos observar que apenas uma função de todas as monitoradas não utilizou *cache* de dados. A função A/5 contém informações precisas sobre as frequências e notas de um aluno, portanto necessita sempre estar atualizada. Caso o professor registre uma falta para aluno este recebera uma notificação por e-mail e poderá conferir suas frequências.

No Quadro 4, são exibidos os consumidores dos serviços monitorados. São monitorados cinco sistemas consumidores em um total de dez funcionalidades, onde por exemplo a funcionalidade A/2, responsável por retornar a foto de uma pessoa, é invocada pelos sistemas D, E e F.

Quadro 4 – Consumidores de serviços

Sistema	**Sistema/Função**	**Tipo de *web service***
A	B/1	REST
A	C/1	SOAP
D	A/1	SOAP
D	A/2	SOAP
D	A/4	SOAP
D	A/5	SOAP
E	A/2	SOAP

E	A/3	SOAP
F	A/2	SOAP
G	A/6	SOAP

Fonte: Elaborado pelo autor, 2019.

De todas funcionalidades da organização apenas a função B/1 utilizada o padrão REST como meio de comunicação. Está função é utilizada para validar o CEP informado em formulários, e atualmente está servindo como alternativa mais rápida e simples para a função C/1 que utiliza o protocolo SOAP.

O Quadro 5 apresenta a quantidade de *responses* e o consumo medido em Bytes de cada uma das funcionalidades que foram monitoradas durante o período de testes, com e sem o *cache* de dados.

Quadro 5 – *Responses* de serviços

Sistema /Função	**Responses sem cache**	**Bytes sem cache**	**Responses com cache**	**Bytes com cache**
A/1	93.858	74.168.769	88.707	72.628.426
A/2	85.775	16.189.883.838	387.661	76.989.429.230
A/3	36.749	2.937.669	651.095	64.840.284
A/4	3.786	147.119.741	1.110	41.636.086
A/5	646.670	8.195.933.968	0	0
A/6	8.599	11.079.267.341	8.563	10.787.979.095
B/1	5.303	1.137.508	32.338	6.417.509
C/1	508	110.226	221	40.159

Fonte: Elaborado pelo autor, 2019.

Podemos notar que a função A/3 tem elevados números de acesso e somente é invocada pelo sistema E, diferente da função A/2 que também tem os números elevados porém é invocada pelos sistemas D, E e F. A utilização do *cache* na maioria das funcionalidades, proporcionou um considerável ganho em desempenho e economia de

recursos do serviço destino. Com a implementação da ferramenta, foram deixados de trafegar aproximadamente 88 Gigabytes no período de tempo de sua execução.

Para mensurar o ganho na velocidade em que o DataCache consegue retornar uma requisição em cache, o Quadro 6 apresenta uma comparação da média de tempo das funcionalidades que utilizam cache, mostrando a média de uma *response* que é atendida pela aplicação final (serviço de destino) e a média da *response* atendido pelo próprio DataCache com o *cache*.

Quadro 6 – Media de tempo dos *responses* com *cache*

Sistema /Função	**Média de tempo sem cache**	**Média de tempo com cache**
A/1	00:00:00.546187	00:00:00.129652
A/2	00:00:00.278053	00:00:00.131391
A/3	00:00:00.591417	00:00:00.126825
A/4	00:00:00.83439	00:00:00.109009
A/6	00:00:04.154088	00:00:00.207871
B/1	00:00:00.82161	00:00:00.154833
C/1	00:00:00.364173	00:00:00.140271

Fonte: Elaborado pelo autor, 2019.

A função A/6 foi a funcionalidade que teve a maior diferença na resposta de tempo com e sem *cache*, cerca de 3,9 segundos. Foi constatado que devido à complexidade na geração do documento e no volume de dados que é transportado pelo protocolo é normal que está funcionalidade demore um pouco mais que as outras.

Constatado que o tempo de requisição e a resposta melhora em dados passíveis de *cache*, em algumas funcionalidades este ganho de tempo não é tão considerável, pois como quase todas as funcionalidades encontra-se na mesma rede, o tempo para a conexão é mínimo, o que leva o tempo de execução da função na aplicação destino, como sendo o principal fator de elevação.

6.2 Análise da pesquisa

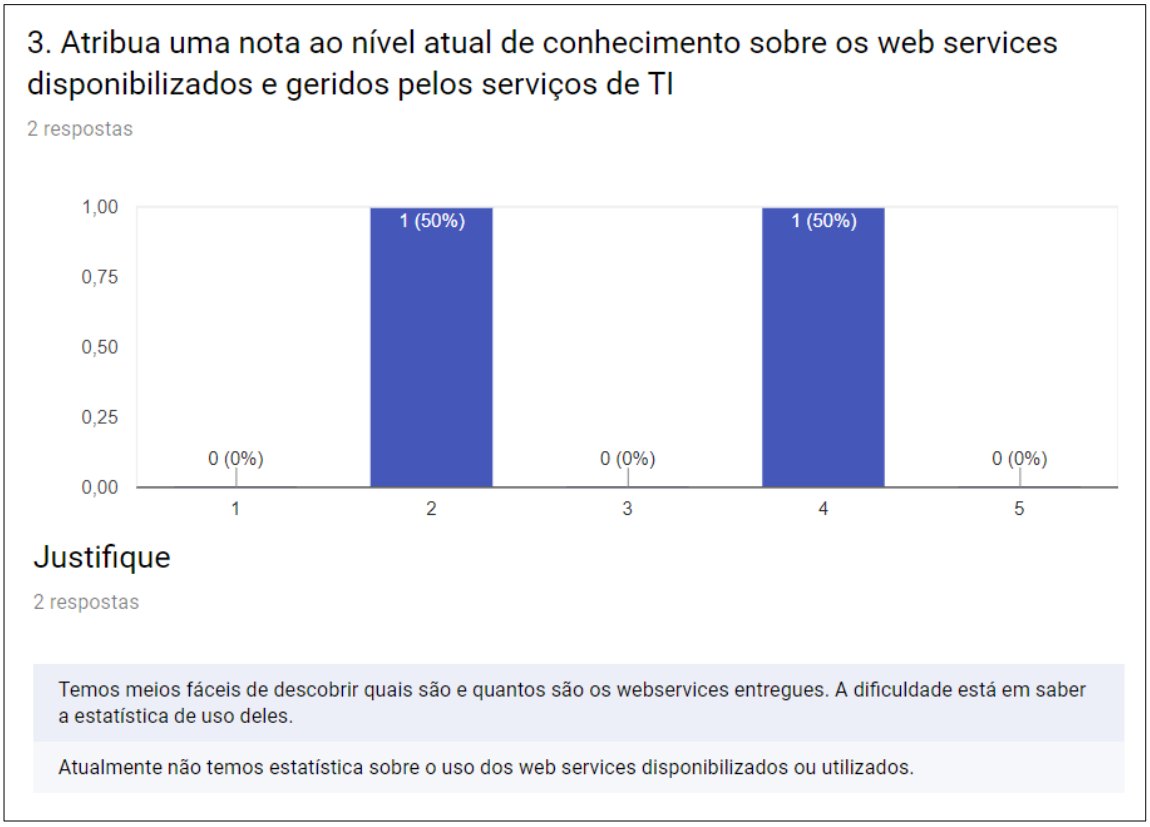
Para poder avaliar tanto o cenário atual da organização quanto a ferramenta desenvolvida, foi elaborado um formulário com questões quantitativas e qualitativas, que no total somam sete questões, exibidas no Apêndice A deste trabalho. O questionário continha uma breve apresentação da ferramenta, explicando o objetivo principal da mesma, sendo aplicado com os gestores da área de desenvolvimento da organização.

Antes de iniciar o questionário, foi realizada a explicação sobre os conceitos e os objetivos da ferramenta, realizando uma apresentação da ferramenta implementada, mostrando desde a fase inicial onde é realizada os cadastros e permissões da aplicação quanto a fase final onde são geradas as estatísticas exibindo as informações das funcionalidades monitoradas com dados coletados até o momento.

As questões um e dois tinham o objetivo de coletar o perfil da pessoa que estava respondendo, solicitando ao respondente quanto tempo de trabalho ele já tinha dentro da organização e qual o seu respectivo cargo. Dentre os entrevistados, todos eles tinham mais de seis anos dentro da organização com cargos de gestão e coordenação.

Na questão três, objetivou-se identificar o cenário atual da organização. Para isso foi solicitado aos avaliadores que atribuíssem uma nota ao conhecimento atual de seus serviços disponibilizados via *web services* e que fizessem uma justificativa de sua resposta. Com diferentes notas atribuídas por cada um dos entrevistados, porém com justificativas similares, que podem ser vistas na Figura 21, percebe-se que, conforme o esperado, atualmente a organização detêm conhecimento sobre as suas funcionalidades disponibilizadas, mas não tem estatísticas e nem informações sobre a utilização das mesmas.

Figura 21 – Respostas da terceira pergunta, referente ao conhecimento dos serviços atuais

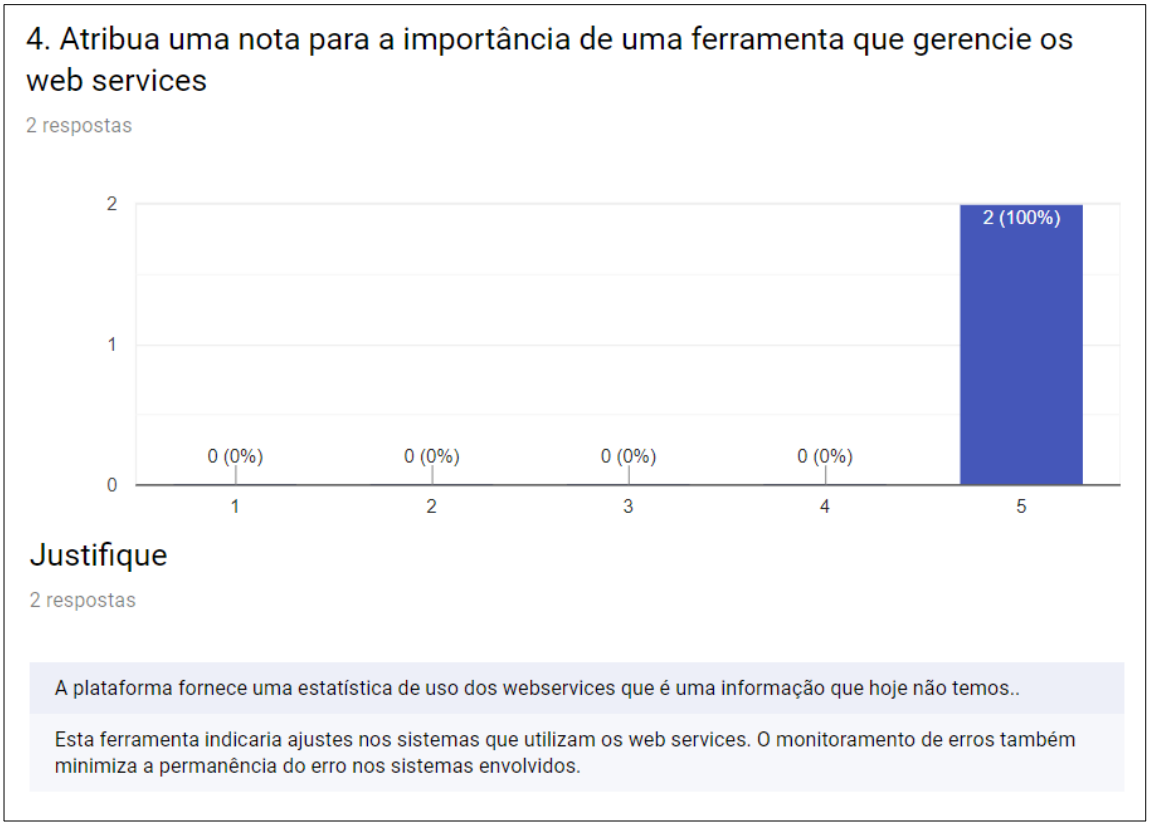


Fonte: Elaborado pelo autor, 2019.

A pergunta quatro, tinha um objetivo mais específico de validar a ideia da aplicação, solicitando ao avaliador classificação da importância de uma ferramenta que gerenciasse a deficiência identificada na pergunta três.

A Figura 22, exibe a unanimidade nas respostas, onde todos os avaliadores conseguiram identificar a importância de ter o controle e acesso as estatísticas dos serviços disponibilizados e a importância na identificação para prevenção mais efetiva de futuros problemas nessas funcionalidades.

Figura 22 – Respostas da quarta pergunta, referente a importância de uma ferramenta

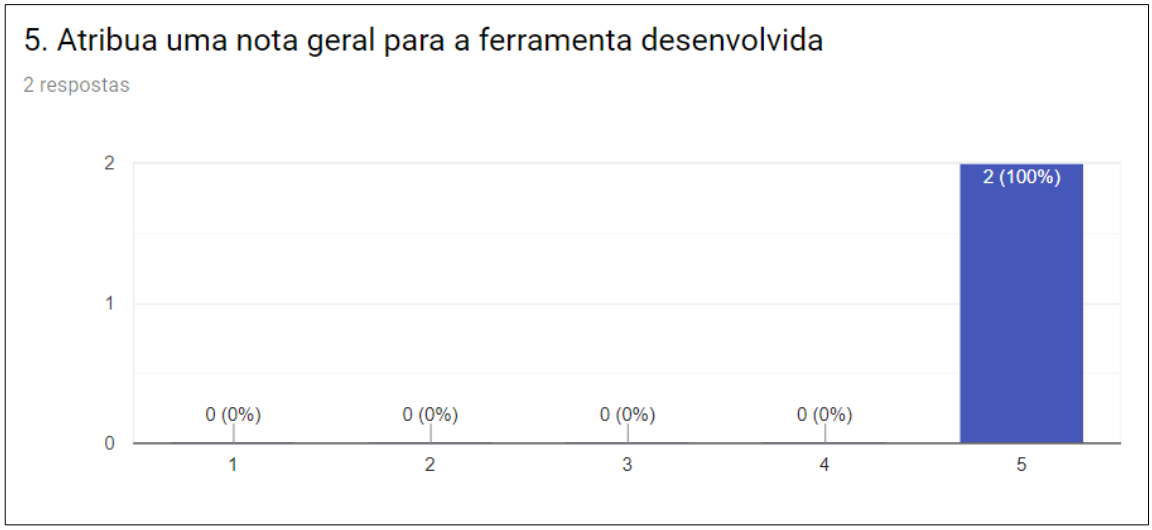


Fonte: Elaborado pelo autor, 2019.

Após a pergunta quatro, novamente alterava-se o foco do questionário, abordando com maior ênfase a ferramenta desenvolvida. A questão cinco buscava verificar se a aplicação desenvolvida teve aceitação solicitando para os avaliadores atribuírem uma nota de um a cinco.

Como a necessidade de ter as estatísticas necessárias dos serviços entregues foi identificada pelas questões três e quatro, a solução DataCache conseguiu nota máxima de aceitação e suas respostas podem ser vistas pela Figura 23.

Figura 23 – Respostas da quinta pergunta, referente a nota geral da ferramenta

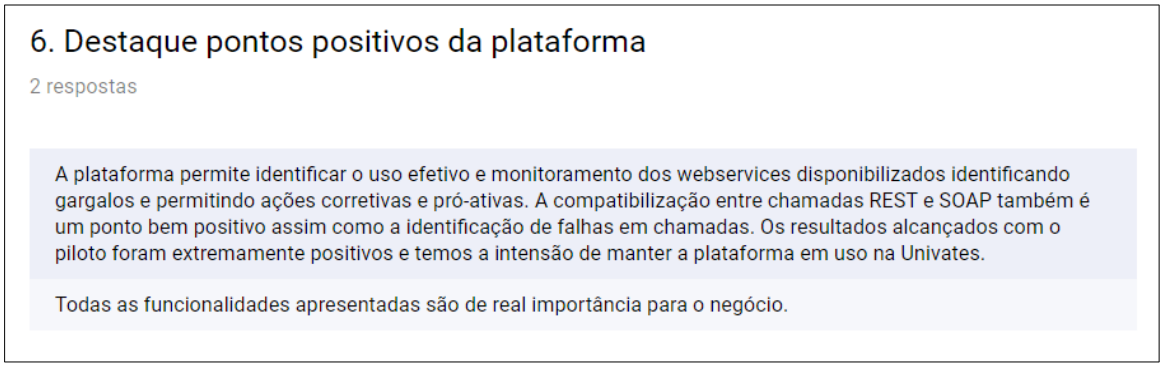


Fonte: Elaborado pelo autor, 2019.

Para poder complementar a questão cinco e finalizar o questionário, as questões seis e sete solicitaram aos entrevistados comentários tanto dos pontos positivos quanto dos pontos negativos da plataforma desenvolvida. As respostas podem ser vistas nas Figuras 24 e 25.

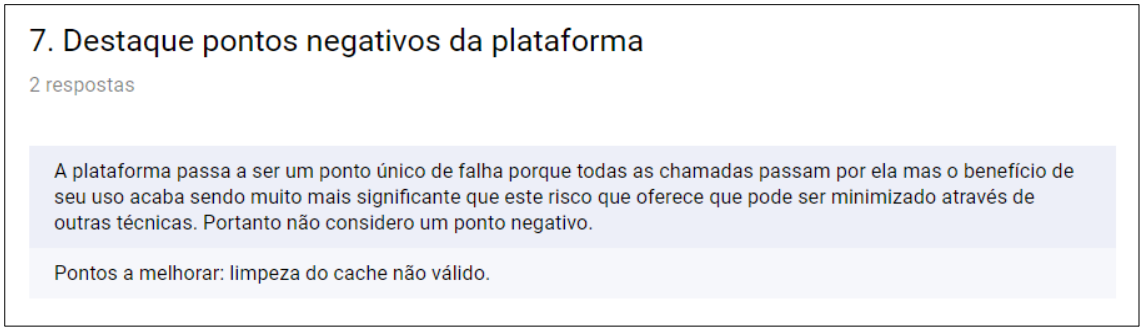
Estas questões visam saber a opinião dos entrevistados, onde destacam-se como pontos positivos compatibilização entre as chamadas, os avisos sobre as falhas nas funcionalidades e a identificação de gargalos. Quantos aos destaques negativos, não ter a possibilidade de limpar dados antigos e ser um ponto único para falhas.

Figura 24 – Respostas da sexta pergunta, referente a pontos positivos da ferramenta



Fonte: Elaborado pelo autor, 2019.

Figura 25 – Respostas da sétima pergunta, referente a pontos negativos da ferramenta



Fonte: Elaborado pelo autor, 2019.

Após finalizado a análise das questões feitas com as pessoas envolvidas na pesquisa, conforme já esperado, o cenário atual da organização não dispõem de nenhum tipo de ferramenta gerencie seus *web services*, mesmo que haja concordância unânime entre os gestores na importância de tal.

Além disso é possível afirmar que a plataforma DataCache teve um grande desempenho e uma grande aceitação perante a sua proposta. Nas questões descritivas foram elencadas novas funcionalidades desejadas no sistema, que vão ser adotadas como implementações futuras.

7 CONSIDERAÇÕES FINAIS

Como a maioria dos sistemas que construímos atualmente são sistemas especialistas, estes têm a necessidade de conectar-se a outros sistemas para realizar a troca de informações. Para que isto ocorra, existe a necessidade de ter algum tipo de ligação entre eles, que na maioria das vezes é feita através de *web services*.

Por meio dos estudos concretizados, foi possível certificar a importância de uma plataforma para gerenciar os serviços de WS de uma organização, fornecendo uma forma de padronizar as chamadas de sistemas e gerar dados e estatísticas que no futuro podem ajudar a melhorar estes serviços.

Durante o período de coleta da ferramenta pode-se observar algumas informações que até então passavam despercebidas pela organização, como por exemplo as chamadas desnecessárias feitas via WS, a quantidade significativa de dados trafegados no servidor da aplicação destino e a possibilidade de cacheamento de dados das requisições para melhora na performance de algumas funcionalidades.

Outro ponto que afirma o êxito no objetivo da aplicação é que ao final do período de testes os gestores da área alegaram o interesse em dar continuidade a este projeto e torná-lo uma ferramenta mantida pela organização.

Através das avaliações feitas com estes profissionais da área da gestão após o período de coleta, pode se perceber a dificuldade em gerenciar esses tipos de serviços até então e uma grande satisfação dos usuários com a ferramenta implementada. Algumas considerações feitas

foram adotadas como implementações futuras como por exemplo a limpeza de dados que não vão mais ser utilizados para cache.

Com a possibilidade de criação de novas funcionalidades dentro da ferramenta, uma implementação possível seria ter a redundância de serviços, promovendo, caso haja falha no serviço “A” uma forma de, imediatamente, fazer uma chamada para o serviço “B”. Outra implementação futura seria o fato de criar conexões diretas com a base de dados da aplicação cadastrada, para que possam ser realizadas consultas pré-cadastradas disponibilizando-as via WS.

REFERÊNCIAS

AMAZON. **Database Caching**, 2019. Disponível em: <https://aws.amazon.com/pt/caching/>. Acesso em 22 abr. 2019.

AUDY, Jorge Luis Nicolas; ANDRADE, Gilberto Keller de; CIDRAL, Alexandre. **Fundamentos de sistemas de informação**. Porto Alegre: Bookman, 2007.

BALDUÍNO, Plínio. **Dominando JavaScript com jQuery**. São Paulo: Casa do Código, 2013.

CHAPPELL, David; JEWELL, Tyler. **Java TM Web Services**. O'Reily & Associates Inc. 2002.

CORREIA, Karina da Silva. **Evolução arquitetural de um web service: transformação de código e avaliação da arquitetura**. 2015. Disponível em: <https://repositorio.ufpe.br/bitstream/123456789/17765/1/Dissertação_Karina_CIN.pdf>. Acesso em 30 abr. 2019.

CROCKFORD, Douglas. **O melhor do JavaScript**. Rio de Janeiro: Alta Books, 2008.

COLÂNGELO, L. **Implantação de sistemas ERP (Enterprise Resources Planning)**. São Paulo: Atlas, 2001;

COUTINHO, Paulo César. **REST Tutorial**. 2013. Disponível em: <https://www.devmedia.com.br/rest-tutorial/28912>. Acesso em 26 mai. 2019.

DALL'OGLIO, Pablo. **Adianti Framework para PHP**. São Paulo: Novatec, 2015. ed 5.

DALL'OGLIO, Pablo. **PHP: programando com orientação a objetos**. São Paulo: Novatec, 2016. ed. 3.

DREAMFACTORY, 2019. Disponível em: <https://www.dreamfactory.com/>. Acesso em 15 de out. 2019.

DUARTE, Felippe Roberto Bayestorff. **Integração de Sistemas**. 2009. Disponível em: <https://repositorio.ufsc.br/handle/123456789/184496>. Acesso em 25 fev. 2019.

EIS, Diego; FERREIRA, Elcio. **HTML5 & CSS3 com farinha e pimenta**. 1. ed. São Paulo: Tableless, 2012. E-book. Disponível em: <https://books.google.com.br/books?id=nDKMAwAAQBAJ&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em 30 mar. 2019.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa**. 4. ed. São Paulo: Atlas, 2002.

GOMES, Daniel Adorno; JANDL, Peter Jr. **WEB SERVICES SOAP E REST**. 2009. Intellectus, Ano V, Nº6. Disponível em: <http://www.revistaintellectus.com.br/DownloadArtigo.ashx?codigo=7>. Acesso em 05 mar. 2019.

KAHLMEYER-MERTENS, Roberto; FUMANGA, Mario; TOFFANO, Claudia; SIQUEIRA, Fabio. **Como elaborar projetos de pesquisa linguagem e método**. 1.ed. Rio de Janeiro: Editora FGV,2007.

KUEHNE, Bruno Tardiole. **Modelos e algoritmos para composição de Web Services com qualidade de serviço**. 2009. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-05042010-111224/publico/bruno.pdf>. Acesso em 08 mar. 2019.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. **Fundamentos de metodologia científica**. 7. ed. São Paulo: Atlas, 2010.

LAUDON, Kenneth; LAUDON, Jane. **Sistemas de informação gerenciais: administrando e empresa digital**. 5. ed. São Paulo: Prentice Hall, 2004.

MARTINS, Victor Manuel Moreira. **Integração de Sistemas de Informação**. 2005. Disponível em: <https://repositorium.sdum.uminho.pt/bitstream/1822/5657/3/tese_mestrado_victor_martins_2005.pdf>. Acesso em 05 mar. 2019.

MARCELO, Antonio. **Apache: configurando o servidor WEB para linux**. São Paulo: Brasport, 2005.

MECENAS, Ivan; OLIVEIRA, Viviane. **Qualidade em software: uma metodologia para homologação de sistemas**. Rio de Janeiro: Alta Books, 2005.

MILANI, André. **PostgreSQL: guia do programador**. São Paulo: Novatec, 2008.

MORESI, Eduardo. **Metodologia da Pesquisa**. 2003. 108 f. Pós-Graduação Stricto Sensu, Universidade Católica de Brasília, Brasília, 2003.

O´BRIEN, James. **Sistemas de informações e as decisões gerenciais na era da Internet**. 9.ed. São Paulo: Saraiva, 2004.

PHP. **PHP**, 2019. Disponível em: <https://www.php.net/>. Acesso em 20 mar. 2019.

PRESSMAN, Roger; PENTADO, Rosângela. **Engenharia de software**. 6. ed. Porto Alegre: AMGH, 2010.

PRODANOV, Cleber Araujo; FREITAS, Ernani Cesar. **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho acadêmico**. 2013. Novo Hamburgo Rio Grande do Sul: Universidade FEEVALE, 2. ed.

REZENDE, Denis; ABREU, Aline. **Tecnologia da informação aplicada a sistemas de informação empresariais: o papel estratégico da informação e dos sistemas de informação nas empresas**. 9. ed. São Paulo: Atlas, 2013.

ROHLING, Adair; GARCIA, Vinicius. **Um Plugin para Monitoramento e Gerenciamento de Web Services baseados em SOAP**. 2012. Disponível em: <https://pt.scribd.com/document/101523024/Um-Plugin-Para-Monitoramento-e-Gerenciamento-de-Web-Services-Baseados-Em-SOAP>. Acesso em 05 set. 2019.

SAUDETE, Alexandre. **REST: Construa API's inteligentes de maneira simples**. 2014. Casa do Código. eBook Kindle.

SAUDETE, Alexandre. **SOA aplicado: integrando web services e além**. 2012. Casa do Código. eBook Kindle.

SOMMERVILLE, Ian. **Engenharia de software**. 2011. São Paulo, SP: Pearson Prentice Hall, 9. ed.

TONSIG, Sérgio Luiz. **Engenharia de software: análise e projeto de sistemas**. 2008. Rio de Janeiro: Ciência Moderna Ltda. 2. ed.

ZOHO. **Applications Manager**, 2019. Disponível em: <https://www.manageengine.com/products/applications_manager/>. Acesso em 20 out. 2019.

W3C. **Simple Object Access Protocol (SOAP) 1.1**. Disponível em: <https://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Acesso em 12 mar. 2019.

APÊNDICES

Apêndice A – Questionário de avaliação da ferramenta desenvolvida

1. Há quantos anos você trabalha na organização?

( ) 1 ano ou menos

( ) 1 a 2 anos

( ) 2 a 4 anos

( ) 4 a 6 anos

( ) 6 anos ou mais

2. Qual o seu cargo atual dentro da organização?

( ) Gestor

( ) Desenvolvedor

( ) Testador

( ) Analista

( ) Coordenador

3. Atribua uma nota ao nível atual de conhecimento sobre os web services disponibilizados e geridos pelos serviços de TI

FRACO - ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5 - ALTO

Justifique

4. Atribua uma nota para a importância de uma ferramenta que gerencie os *web services*

PEQUENA - ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5 - GRANDE

Justifique

5. Atribua uma nota geral para a ferramenta desenvolvida

RUIM - ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5 - ÓTIMA

6. Destaque pontos positivos da plataforma

7. Destaque pontos negativos da plataforma