FACULDADE DE TECNOLOGIA SENAC BLUMENAU
Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Denisson Augusto Silveira da Silva
Éderson Henrique Chimbida
Marcos Gabriel Mandel
Pablo Lucas Ramthun
Sandro Rodrigues

SOFTWARE COMO SERVIÇO: ARMAZENAMENTO DE DADOS PARA INTERNET DAS COISAS

Blumenau
2017

Denisson Augusto Silveira da Silva
Éderson Henrique Chimbida
Marcos Gabriel Mandel
Pablo Lucas Ramthun
Sandro Rodrigues

SOFTWARE COMO SERVIÇO: ARMAZENAMENTO DE DADOS PARA INTERNET DAS COISAS

Trabalho de Conclusão de Último Semestre apresentado à Faculdade de Tecnologia Senac Blumenau como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Professores (as): Cláudio Ratke
Fabiano Oss
Lariana Luy Peixoto

Blumenau
2017

Denisson Augusto Silveira da Silva
Éderson Henrique Chimbida
Marcos Gabriel Mandel
Pablo Lucas Ramthun
Sandro Rodrigues

SOFTWARE COMO SERVIÇO: ARMAZENAMENTO DE DADOS PARA INTERNET DAS COISAS

Trabalho de Conclusão de Último Semestre apresentado à Faculdade de Tecnologia Senac Blumenau como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

_____
Cláudio Ratke (Orientador)

_____
Fabiano Oss (Orientador)

_____
Lariana Luy Peixoto (Orientador)

_____
Luciano Marcelo França – Senac

Blumenau, 29 de junho de 2017

“As tecnologias mais importantes e duradouras são aquelas que desaparecem. Elas dissipam-se na vida do dia a dia, ao nosso cotidiano até serem indistinguíveis dele” (WEISER, 1991, p. 94).

RESUMO

Desde o início da internet, grandes oportunidades surgem. A tecnologia cresce e se adapta diariamente. Este trabalho acadêmico consiste no desenvolvimento de um protótipo de *software* implantado como serviço, voltado para o armazenamento de dados gerados por plataformas da Internet das Coisas (IoT). Com ele, o usuário será capaz de armazenar dados coletados por seus dispositivos. Com o crescimento do volume de dados gerados por dispositivos de internet das coisas, tanto na área corporativa, como de usuários finais, é justificável a criação de um *software* voltado para o armazenamento e centralização dos dados gerados por estes dispositivos. Neste trabalho, foram utilizadas diversas tecnologias, envolvendo desde a linguagem padrão escolhida para o desenvolvimento, até as ferramentas, *frameworks* e *softwares* escolhidos para a construção do sistema como um todo. Entre elas, o *Spring*, tecnologias de computação em nuvem, modelo de distribuição como serviço, microsserviços, MongoDB banco de dados não-relacional e MySQL banco de dados relacional. Toda a comunicação, tanto o envio para armazenamento como consultas para recuperação dos dados, feita pelos dispositivos IoT com o serviço de armazenamento, é realizado através de APIs. O protótipo de *software* desenvolvido possibilita a realização de cadastros de canais e campos através do painel administrativo. Estes campos cadastrados serão utilizados para o armazenamento dos dados no banco de dados não-relacional.

Palavras-chave: Armazenamento. Computação em nuvem. Internet das Coisas. Microsserviços.

ABSTRACT

Since the beginning of the internet, great opportunities have arisen. Technology grows and adapts daily. This academic work consists of the development of software prototype deployed as a service, turned to the storage of data generated by Internet of Things platform (IoT). With it the user will be able to store data collected by their devices. With the growing volume of data generated by Internet of Things devices, both in the corporate area and end users, it is justifiable to create software aimed at storing and centralizing the data generated by these devices. In this work, several technologies were used, starting from the standard language chosen for the development, to the tools, frameworks and software chosen for the construction of the system. Among them, Spring, cloud computing technologies, model of Software as a Service, microservices, MongoDB NoSQL database, MySQL relational database. All communication, both the sending for storage and queries for data recovery, made by IoT devices with the storage service, is performed through APIs. The software prototype developed is able to perform the registration of channels and fields through the administrative panel. These fields will be used to store the data in the non-relational database.

Keywords: Cloud Computing. Internet of Things. Microservices. Storage.

LISTA DE ILUSTRAÇÕES

Figura 1 – Principais métodos HTTP 16
Figura 2 – Ilustração da arquitetura REST 17
Figura 3 – Requisição utilizando JSON 18
Figura 4 – Arquitetura com as ferramentas Netflix OSS 22
Figura 5 – Exemplo de arquivo JSON utilizado com o MongoDB 25
Figura 6 – Disponibilização de *software* em *cloud* 27
Figura 7 – Arquitetura monolítica e de microsserviços 28
Quadro 1 – Requisitos funcionais do sistema 29
Quadro 2 – Requisitos não-funcionais do sistema 30
Figura 8 – Ilustração da arquitetura da solução 31
Figura 9 – Modelo de processo de acesso a aplicação 32
Figura 10 – Modelo de processo de cadastro de canais 33
Figura 11 – Entidades no banco relacional 34
Figura 12 – JSON utilizado para envio ao MongoDB 35
Figura 13 – JSON de resposta a uma consulta ao MongoDB 36
Figura 14 – Estrutura da coleção de dados 37
Figura 15 – Máquinas virtuais utilizados para os testes 41
Figura 16 – Lista de usuários cadastrados para os testes 41
Figura 17 – Informações do usuário Ederson 42
Figura 18 – Informações do canal Estação Meteorológica 42
Figura 19 – Informações do usuário Sandro 43
Figura 20 – Informações do canal Geladeira 43
Figura 21 – Informações do usuário Denisson 44
Figura 22 – Informações do canal Casa 44
Figura 23 – Informações do usuário Pablo 45
Figura 24 – Informações do canal Plantação de Tomates 45
Figura 25 – Informações do canal Plantação de Alface 46
Figura 26 – Informações do usuário Marcos 46
Figura 27 – Informações do canal Sistema de Irrigação 47
Figura 28 – Exemplo de JSON para o POST 48
Figura 29 – Resultado dos testes realizados na API 48

Figura 30 – Resultado do uso de CPU, memória e disco 49
Figura 31 – Uso de CPU dos servidores 50
Figura 32 – Uso de memória dos servidores 51
Figura 33 – Operações geradas pelo MongoDB 52
Figura 34 – Operações geradas pelo MySQL 52

LISTA DE SIGLAS

API – *Application Programming Interface*

BPMN – *Business Process Model and Notation*

CPU – *Central Processing Unit*

CRM – *Customer Relationship Management*

DI – *Dependency Injection*

DB – *Data Base*

GUID – *Globally Unique IDentifier*

HTTP – *Hypertext Transfer Protocol*

ID – *Identity*

IP – *Internet Protocol*

IoT – Internet das Coisas

JDBC – *Java Database Connectivity*

JPA – *Java Persistence* API

JSON – *JavaScript Object Notation*

NIST – *National Institute of Standards and Technology*

NoSQL – *Not Only* SQL

OSS – *Open Source Software*

REST – *Representational State Transfer*

RF – Requisitos Funcionais

RNF – Requisitos Não-Funcionais

SaaS – *Software as a Service*

SGBD – Sistema Gerenciador de Banco de Dados

SQL – *Structured Query Language*

TI – Tecnologia da Informação

UC – *Use Case*

UI – *User Interface*

URI – *Uniform Resource Identifier*

URL – *Uniform Resource Locator*

VM – *Virtual Machine*

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVO 13
1.1.1 Objetivo geral 13
1.1.2 Objetivos específicos 13
1.2 JUSTIFICATIVA 14
2 METODOLOGIA 15
3 TECNOLOGIAS 16
3.1 ARQUITETURA REST 16
3.2 NETFLIX *OPEN SOURCE* 18
3.3 *SPRING* 18
3.3.1 *Spring Framework* 19
3.3.2 *Spring Boot* 19
3.3.3 *Spring Security* 19
3.3.4 *Spring Cloud* 20
3.4 MySQL 23
3.5 MONGODB 23
3.6 REDIS 24
3.7 COMPUTAÇÃO EM NUVEM 25
3.8 *SOFTWARE* COMO SERVIÇO 26
3.9 MICROSSERVIÇOS 27
4 DESENVOLVIMENTO 29
4.1 REQUISITOS 29
4.2 DEFINIÇÃO DA ARQUITETURA 31
4.3 DEFINIÇÃO DA ESTRUTURA FUNCIONAL 31
4.4 ENTIDADES DO BANCO DE DADOS RELACIONAL 33
4.5 ESTRUTURA DO BANCO DE DADOS NÃO-RELACIONAL 35
4.6 DETALHAMENTO DOS MICROSSERVIÇOS 37
4.7 DIAGRAMAS DE CLASSE 38
4.8 PROTÓTIPOS DE TELAS 39
4.9 RESULTADOS E MÉTRICAS 40
5 CONSIDERAÇÕES FINAIS 54
REFERÊNCIAS 57

GLOSSÁRIO 59
APÊNDICE A – ESPECIFICAÇÕES DE CASOS DE USO 60
APÊNDICE B – DIAGRAMAS DE CLASSE 64
APÊNDICE C – PROTÓTIPOS DE TELAS DO SISTEMA 71
APÊNDICE D – TESTES DE LATÊNCIA REALIZADOS NA API 78
APÊNDICE E – TESTES REALIZADOS NA API POR POST 79
APÊNDICE F – TESTES REALIZADOS NA API POR GET 80
APÊNDICE G – ESTATÍSTICAS POR TIPO DE REQUISIÇÃO 81

1 INTRODUÇÃO

O advento da internet trouxe rápidas mudanças no mundo da tecnologia da informação e comunicação. Segundo Monteiro (2001), no Brasil a internet teve um crescimento espantoso, já no seu surgimento. Entre os anos de 1996 e 1997, aumentou em quase 1000% o número de usuários, passando de 170 mil (janeiro/1996) para 1,3 milhão (dezembro/1997). Em janeiro de 2000, eram estimados 4,5 milhões de usuários desta tecnologia.

A tecnologia em torno dos serviços de conectividade, trabalhando em conjunto com conceitos como o de Internet das Coisas (IoT), *hiperconectividade*, *cloud computing*, entre outros assuntos que comportam a mobilidade das informações, parece não ter um limite de crescimento para os próximos anos.

Números divulgados pelo grupo Gartner (2017), empresa que fornece pesquisa e consultoria relacionadas à obtenção de informações necessárias para tomada de decisões envolvendo a tecnologia da informação, relatam que 8,4 bilhões de aparelhos conectados à internet estarão circulando no mundo até o final deste ano.

Para o fim deste ano, estima-se que os usuários finais estejam gastando algo em torno de US$ 2,0 trilhões de dólares, na compra e manutenção da estrutura necessária, com dispositivos com potencial para se conectarem através da internet. Essa estimativa para o ano de 2017 propõe números que chegam a 31% de aumento se comparada com o valor no ano 2016 (GARTNER, 2017). Seguindo esta linha de crescimento, a quantidade de informações geradas será enorme, necessitando um local seguro para armazená-las.

O projeto desenvolvido neste trabalho tem por objetivo o desenvolvimento de um protótipo de software para armazenamento de dados gerados por dispositivos IoT. O software será disponibilizado em forma de serviço (SaaS), onde o usuário pagará mensalmente de acordo com o seu uso.

Com ele o usuário será capaz de armazenar diferentes tipos de dados coletados por seus dispositivos, possibilitando que estes dados sejam organizados em canais distintos. Em cada canal o usuário pode configurar campos para diferentes tipos de dados coletados.

Os dispositivos IoT se comunicarão com os serviços, através de uma API REST. Esta API (*Application Programming Interface*) será utilizada tanto para o

envio dos dados dos dispositivos para o serviço como para a consulta de dados armazenados.

O serviço disponibilizará um portal para o usuário, no qual ele terá acesso a diversas configurações, funcionando como um painel de controle onde pode-se definir o modo de armazenamento, tipos de dados armazenados, quantidade de canais, entre outras configurações relevantes para o uso do sistema.

Todo o *software* desenvolveu-se com tecnologias que trabalham em conjunto com a linguagem de programação Java, tecnologias estas como o *Spring e* outras. Na arquitetura do sistema, os conceitos de microsserviços e computação em nuvem são abordados.

1.1 OBJETIVO

A seguir serão descritos os objetivos em pauta relacionados a este projeto. Objetivos a serem alcançados para que este seja concluído com êxito.

1.1.1 Objetivo Geral

Este projeto estabelece como objetivo principal, implementar um protótipo de software como serviço, voltado para armazenar dados gerados de dispositivos conectados através da internet (Internet das Coisas). Um serviço que permite escalabilidade em um ambiente de armazenamento, gerando informações para serem usados posteriormente em relatórios passíveis à análise e tomada de decisão.

1.1.2 Objetivos Específicos

Aqui, estão relacionados alguns objetivos mais específicos e relevantes ao projeto deste trabalho:

a) Desenvolver um protótipo de sistema que seja capaz de armazenar dados coletados por dispositivos da Internet das Coisas;
b) Disponibilizar o *software* como serviço utilizando a arquitetura de microsserviços;

c) Implementar uma Interface de Programação de Aplicação (API) para integração com o serviço.

1.2 JUSTIFICATIVA

Conforme citado anteriormente, os números divulgados pelo grupo Gartner (2017), em relação ao crescimento do mercado de Internet das Coisas, mostram o aumento expressivo do volume de informações geradas nos próximos anos. Este fato promove a necessidade de cada vez mais possibilidades de armazenamento desses dados serem desenvolvidas.

Com esse crescimento do volume de informações no mercado de internet das coisas (IoT), tanto na área corporativa como de usuários finais, é justificável a criação de um *software* que seja capaz de armazenar os dados gerados por dispositivos IoT. Massruhá (2015) cita o uso da tecnologia na área da agricultura, destacando a necessidade de armazenamento das informações geradas por esses dispositivos:

> A busca pela otimização no uso dos recursos naturais e dos insumos fará com que a fazenda do futuro seja massivamente monitorada e automatizada. Sensores dispersos por toda a propriedade e interligados à internet configurarão a 'Internet das Coisas', em que os objetos ou aparelhos do mundo estarão ligados de um modo sensorial e inteligente e gerarão dados em grande volume (*big data*). Estes dados necessitarão ser filtrados, armazenados (computação em nuvem) [...] (MASSRUHÁ, 2015, p. 29-30).

Como visto, o volume de dados no mercado das tecnologias envolvendo a Internet das Coisas cresce a cada ano. Com isso, entende-se que o armazenamento dos dados é um importante desafio relacionado a este crescimento. O foco na capacidade de armazenamento dos dados traria maneiras de superar este desafio, que hoje, é um dos principais problemas a serem solucionados.

O desenvolvimento de uma alternativa para o armazenamento dos dados, por sua vez, traz uma oportunidade de arrecadação de receita, promovendo uma iniciativa que pode gerar lucros. A iniciativa proposta, parte da ideia de comercialização através de um modelo de software como serviço (SaaS) onde o usuário contrata conforme sua demanda de equipamentos ou quantia de dados coletados que precisa armazenar.

2 METODOLOGIA

Neste trabalho, a revisão de literatura é de grande importância. A metodologia utilizada se conceitua a partir da pesquisa exploratória, baseando-se em estudos de casos e pesquisa bibliográfica.

> As pesquisas exploratórias têm como principal finalidade desenvolver, esclarecer e modificar conceitos e idéias, tendo em vista a formulação de problemas mais precisos ou hipóteses pesquisáveis para estudos posteriores. De todos os tipos de pesquisa, estas são as que apresentam menor rigidez no planejamento. Habitualmente envolvem levantamento bibliográfico e documental, entrevistas não padronizadas e estudos de caso (GIL, 2008, p. 27).

A pesquisa bibliográfica é uma parte contundente do trabalho. “Qualquer que seja a pesquisa, a necessidade de consultar material publicado é imperativa” (GIL, 2008, p. 60). A busca bibliográfica que fundamenta este trabalho, basicamente, contém os conteúdos publicados de livros e artigos de diversos autores e partes de documentações das ferramentas e tecnologias, relevantes para o entendimento do projeto.

O desenvolvimento do protótipo considera a utilização de diversas ferramentas e técnicas de projeto de software. São utilizadas técnicas de levantamento de requisitos e elaboração de modelos de processos com diagramas BPMN (*Business Process Model and Notation*) para o entendimento da funcionalidade. Nos requisitos, as principais funções são detalhadas com diagramas de Caso de Uso (UC).

A construção da aplicação considera algumas técnicas de detalhamento das classes através de diagramas de classes. A partir do código-fonte da aplicação, as rotinas, métodos e demais recursos de implementação foram organizados formando um documento, para auxílio no entendimento das rotinas.

Os recursos utilizados no desenvolvimento do protótipo de software são abordados nos tópicos seguintes.

3 TECNOLOGIAS

A fim de se chegar ao objetivo proposto pelo projeto contido neste trabalho, foram utilizadas diversas tecnologias, envolvendo desde a linguagem padrão escolhida para o desenvolvimento, até as ferramentas, *frameworks* e *softwares* escolhidos para a construção do sistema como um todo. Nos tópicos que seguem, estão listados descritivamente as principais tecnologias utilizadas.

3.1 ARQUITETURA REST

Antes de descrever-se as demais tecnologias, inicialmente é preciso comentar sobre API REST (*Representational State Transfer*), uma vez que todos os recursos no projeto a utilizam.

Serviços na *web* são construídos especificamente para suportar as necessidades de um site ou que qualquer outra aplicação possui (MASSE, 2011). As APIs são utilizadas para realizar a comunicação entre a requisição vinda do cliente e do servidor da aplicação. “Uma API expõe um conjunto de dados e funções que facilitam a integração entre programas de computador e permitem que estes troquem informações” (MASSE, 2011, p. 5, tradução nossa). Resumidamente uma API trabalha diretamente recebendo e respondendo às requisições do cliente.

Serviços *RESTful*, como denominado quando se utiliza a arquitetura REST, segundo Rodriguez (2008) é caracterizado pelo uso explícito de métodos HTTP (*Hypertext Transfer Protocol*). Como exemplos pode-se considerar os métodos *POST*, *GET*, *PUT* e *DELETE*, conforme ilustrado na Figura 1.

Figura 1 – Principais métodos HTTP

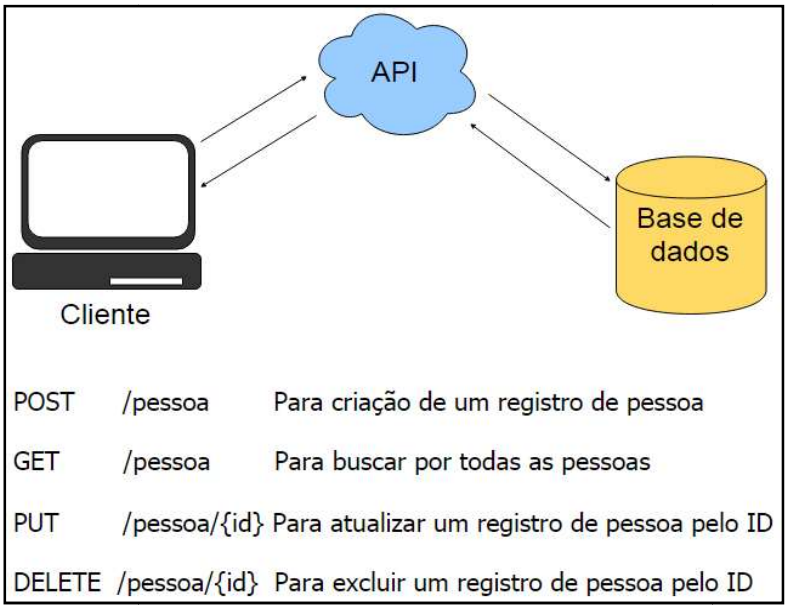
GET	Busca recursos
PUT	Atualiza um recurso que já existe
POST	Cria um recurso
DELETE	Exclui um recurso

Fonte: Elaborada pelos autores baseada em Rodriguez (2008).

Uma das características na arquitetura REST descreve que “dados e funcionalidades são considerados recursos e esses recursos são acessados usando URIs (*Uniform Resource Identifier*), normalmente *links* na *web*” (HAMAD; SAAD; ABED, 2010, p. 74, tradução nossa). Serviços com esta arquitetura mapeiam os métodos HTTP para executarem determinada ação por meio da aplicação.

Um aplicativo de serviço da *Web* REST inclui dentro dos cabeçalhos HTTP e do corpo de uma solicitação todos os parâmetros, contexto e dados necessários ao componente do lado do servidor para gerar uma resposta (RODRIGUEZ, 2008). Na Figura 2, pode ser visto um exemplo básico da chamada de um recurso.

Figura 2 – Ilustração da arquitetura REST

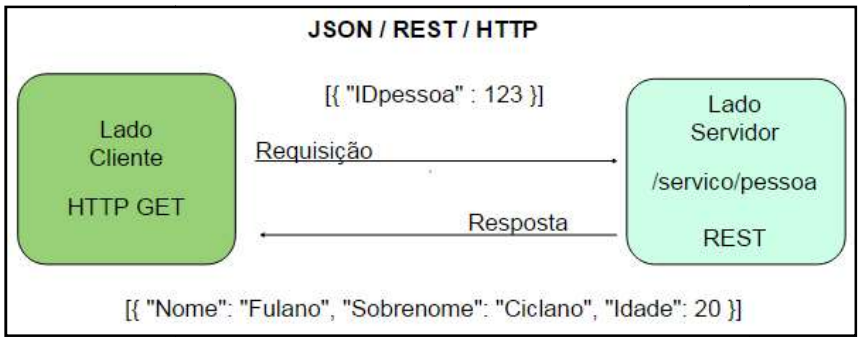


Fonte: Elaborada pelos autores baseada em Rodriguez (2008).

Uma representação dos recursos geralmente reflete o estado atual deste recurso e seus atributos, no momento em que a aplicação do cliente envia uma solicitação (RODRIGUEZ, 2008). Como exemplo, pode-se citar uma representação como um registro contido na base de dados ou uma chamada da aplicação por uma funcionalidade.

Nas transferências de dados em um serviço REST podem ser representados os recursos em diferentes tipos, como por exemplo, o JSON (*JavaScript Object Notation*). Na Figura 3 pode ser identificada, em um exemplo básico, a busca por informações do registro de uma pessoa na aplicação.

Figura 3 – Requisição utilizando JSON



Fonte: Elaborada pelos autores baseada em Rodriguez (2008).

Disponibilizar uma API que utiliza a arquitetura REST, é uma maneira flexível de oferecer serviços *web*, isto é, torna possível a realização de diversas funcionalidades e acesso a diferentes recursos.

3.2 NETFLIX *OPEN SOURCE*

Outro ponto para se abordar antes de falar efetivamente das tecnologias utilizadas, é o Netflix OSS. Este conjunto de ferramentas e serviços oferece bibliotecas para criação de aplicações voltadas para a arquitetura de microsserviços. O conjunto de ferramentas que fazem parte do Netflix OSS possibilita desenvolver aplicações utilizando recursos de armazenamento de dados na nuvem, foco em escalabilidade, confiabilidade, desempenho, segurança e outras características. De maneira efetiva auxiliam no desenvolvimento de aplicações que precisam contemplar os recursos citados (NETFLIX, 2016).

3.3 *SPRING*

O *Spring* pode ser caracterizado como uma maneira de tornar o processo de desenvolvimento de uma aplicação mais prática e fácil. Desta forma também, definido por Moreira Júnior e Afonso (2017), que caracteriza-o como não apenas um *framework*, “mas um conjunto de projetos que resolvem várias situações do cotidiano de um programador, ajudando a criar aplicações Java com simplicidade e flexibilidade” (MOREIRA JR; AFONSO, 2017, p. 12).

No *Spring*, existem muitas áreas cobertas por suas bibliotecas, desde o acesso ao banco de dados, projetos para segurança, e diversos outros projetos que vão de *cloud computing* até *big data*.

3.3.1 *Spring Framework*

Este se caracteriza por ser “o projeto do *Spring* que serve de base para todos os outros” (MOREIRA JR; AFONSO, 2017, p. 13). Possui o intuito de trazer às aplicações mais ênfase na regra de negócio e menos na infraestrutura. Nas funcionalidades se encontram, por exemplo, suporte para JDBC (*Java Database Connectivity*), JPA (*Java Persistence* API) e Injeção de dependências (DI Dependency *injection*).

3.3.2 *Spring Boot*

O *Spring Boot* é um *framework* que facilita a criação de aplicações, aumentando a produtividade e a qualidade. Este projeto “oferece uma estrutura voltada para os microsserviços REST, baseados em Java” (CARNELL, 2017, p. 5, tradução nossa).

Carnell (2017) descreve algumas características, quando expõe que o *Spring Boot* é a tecnologia principal usada na implementação dos microsserviços. Simplifica o desenvolvimento ao facilitar as tarefas centrais de construção de microsserviços REST. O *Spring Boot* simplifica muito o mapeamento de verbos HTTP (*GET*, *PUT*, *POST* e *DELETE*) para URLs (*Uniform Resource Locator*). O *Spring boot* oferece também a serialização de objetos do protocolo JSON para objetos Java. Outra característica está relacionada ao mapeamento das exceções Java retornando-as aos códigos de erro HTTP padrão, facilitando a manutenção das funcionalidades quando necessária.

3.3.3 *Spring Security*

A segurança é um fator extremamente importante quando se considera um projeto de sistema da informação. Deve ser pensada, projetada e adotada com as

melhores práticas para atribuir integridade ao sistema e principalmente às informações contidas nele.

O *Spring Security* oferece serviços de segurança para sistemas empresariais e é baseado na plataforma Java. Geralmente, aplicações que não o utilizam podem demandar maior esforço, por exemplo, no momento de configurar a segurança do seu aplicativo em um novo ambiente. O *Spring Security* supera esses problemas e também traz dezenas de outros recursos de segurança úteis e personalizáveis (ALEX et al., 2015).

O *Spring Security* é uma ferramenta do *Spring* que traz recursos para aplicar configurações envolvendo as ações de autenticação e autorização. Essas áreas são citadas por Alex et al. (2015) como as principais atividades na segurança de aplicativos, integrando o controle de acesso à determinada área da aplicação.

A autenticação é o processo de estabelecer acesso à aplicação. Este processo determina, por exemplo, se um determinado usuário, algum dispositivo ou até mesmo outra aplicação terá possibilidade de executar alguma ação na aplicação. A autorização refere-se ao processo de identificar e atribuir permissões na aplicação. Determina, por exemplo, se o usuário autenticado pode acessar determinada funcionalidade, ou executar uma determinada ação dentro da aplicação. Sendo assim, o processo de autorização é realizado após a autenticação (ALEX et al., 2015).

3.3.4 *Spring Cloud*

O *Spring Cloud* é um subprojeto do *Spring* e suas ferramentas são utilizados com o intuito de agilizar o desenvolvimento. Este conjunto de ferramentas implementa vários padrões comuns em sistemas distribuídos que, segundo Pivotal Software (2017), podem ser, desde gerenciamento de configuração, gerenciamento de serviços, *circuit breakers*, roteamento inteligente, *micro-proxy*, barramento de controle, *tokens* únicos, bloqueios globais, escolha de papéis, sessões distribuídas, gerenciamento de *cluster* e versionamentos.

Dentre os recursos que o *Spring Cloud* disponibiliza*,* Carnell (2017) expõe as principais ferramentas utilizadas na criação de microsserviços, isto é, na implementação dos padrões citados no parágrafo anterior. Recursos como o *Spring Cloud Config, Spring Cloud* Netflix *Eureka* (Descobrimento de serviços)*, Spring*

Cloud Netflix *Hystrix* e *Ribbon* (Balanceamento de carga)*, Intelligent Routing* (Netflix *Zuul*) *e Spring Cloud Security.* Essas bibliotecas são baseadas nos componentes do Netflix OSS.

Conforme citado anteriormente, um dos recursos no *Spring Cloud* é o *Spring Cloud Config.* Este trata o gerenciamento de dados de configuração por meio de um serviço centralizado, de modo que esses dados de configuração estejam separados em um microsserviço específico. Isso garante que, independentemente do número de instâncias de outros serviços que forem apresentados, elas sempre terão a mesma configuração (CARNELL, 2017).

O projeto *Spring Cloud Config* possui ainda integração com outras bibliotecas de código aberto, como o *Spring Cloud* Netflix *Eureka,* que basicamente é uma ferramenta de descoberta de serviço que permite a identificação das instâncias dos serviços na aplicação (CARNELL, 2017).

Um recurso importante é o oferecido pela biblioteca do Netflix *Hystrix.* Carnell (2017) comenta que com essas bibliotecas é possível implementar rapidamente um serviço seguindo os padrões de resiliência, como o *circuit breaker*.

O *circuit breaker* permite que um microsserviço continue em execução mesmo que outro esteja com falhas, impedindo assim uma total paralisação do serviço em forma de cascata. Este é um recurso que se torna extremamente importante em uma aplicação, mantendo desta forma a aplicação com o devido funcionamento.

O *Eureka*, comentado anteriormente, é uma ferramenta de descoberta de serviço. Para facilitar a integração com este recurso, é utilizado o Netflix *Ribbon.* Esta biblioteca também é utilizada para “balanceamento de carga de chamadas de serviço do lado do cliente” (CARNELL, 2017, p. 28, tradução nossa). Este recurso, por exemplo, permite que o cliente continue chamando outros serviços mesmo quando o serviço de descoberta está indisponível.

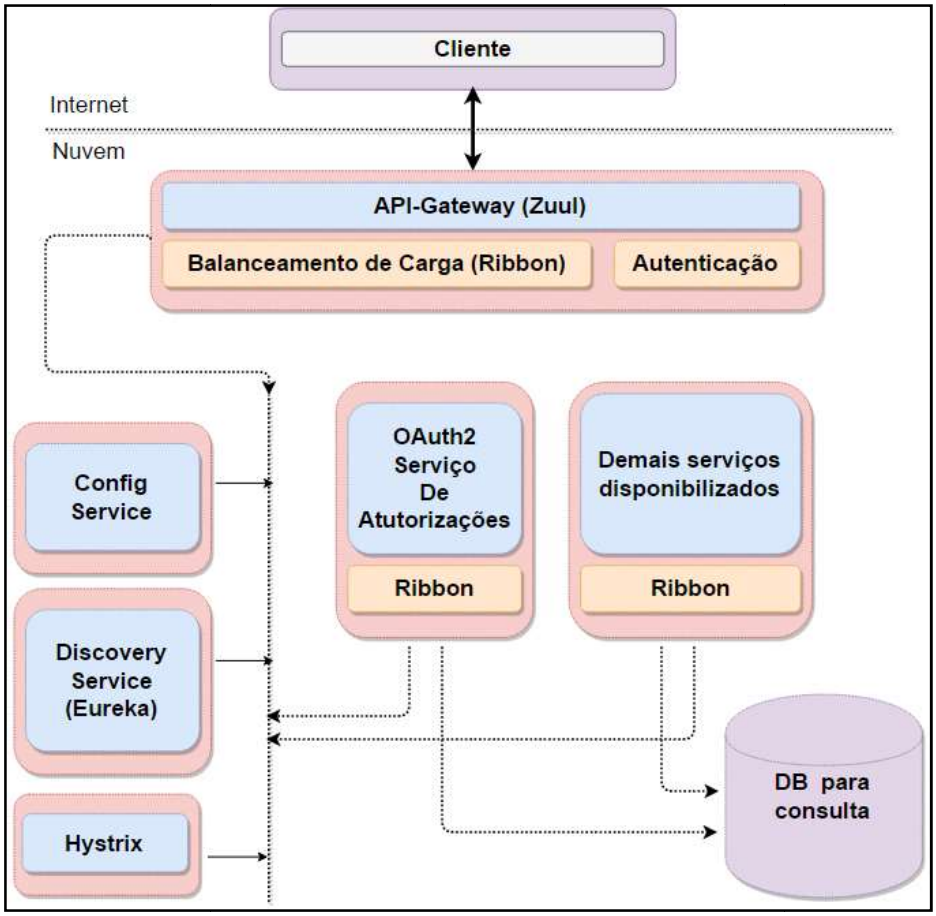
O recurso *Spring Cloud Security,* definido por Carnell (2017):

> [...] é uma estrutura de autenticação e autorização que controla quem pode acessar seus serviços e o que pode executar com seus serviços. O *Spring Cloud Security* é baseado em *token* e permite que os serviços se comuniquem entre si através de um *token* de identificação emitido por um servidor de autenticação. Cada serviço recebendo uma chamada pode verificar o *token* fornecido para validar a identidade do usuário e seus direitos de acesso neste serviço (CARNELL, 2017, p. 29, tradução nossa).

O projeto Netflix Zuul é utilizado para fornecer recursos de roteamento dos serviços para o seu aplicativo de microsserviços. O Zuul é um *Framework* que procura e garante que todas as chamadas para os microsserviços passem, por exemplo, por um único local antes da chamada do serviço seguinte ser invocado. Com esta centralização de chamadas, você pode impor políticas de serviço padrão como uma autenticação de autorização de segurança, filtragem de conteúdo e regras de roteamento (CARNELL, 2017).

Todas estas ferramentas oferecem recursos que, de certa forma, são parte importante da arquitetura de uma aplicação voltada aos microsserviços. Na Figura 4 há uma representação exemplificando como elas se dispõem dentro desta arquitetura.

Figura 4 – Arquitetura com as ferramentas Netflix OSS



Fonte: Elaborada pelos autores baseada em Carnell (2017).

3.4 MYSQL

Segundo Elmasri e Navathe (2005), considera-se banco de dados uma coleção de dados relacionados, que exista um sentido lógico, com coerência e com algum significado inerente. Bancos de dados são devidamente gerenciados por um sistema de gerenciamento de banco de dados (SGBD), que em resumo, é um conjunto de ferramentas para acessar e manipular os dados inter-relacionados (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

Para um projeto envolvendo armazenamento de dados é imprescindível a utilização de um gerenciador de banco de dados, e o MySQL, caracterizado como um banco de dados relacional, vem com este propósito.

> O MySQL é um servidor e gerenciador de banco de dados (SGBD) relacional, de licença dupla, [Uma é a de software livre e uma outra de uso comercial], projetado inicialmente para trabalhar com aplicações de pequeno e médio portes, mas hoje atendendo a aplicações de grande porte e com mais vantagens do que seus concorrentes. (MILANI, 2006, p. 22).

De certa forma, o MySQL se torna de grande utilidade, pois além de banco de dados, contém todas as características de um Sistema Gerenciador de Banco de Dados. Características estas como multi-acesso aos dados, gerenciamento de acesso, integridade dos dados, integridade relacional, concorrência, transações, segurança, entre outros (MILANI, 2006).

3.5 MONGODB

MongoDB é um sistema de gerenciamento de banco de dados não-relacional, que contempla o conceito utilizado para NoSQL. Conceito que dispõe de “sistemas de gerenciamento de banco de dados diferentes dos bancos de dados relacionais tradicionais em que os dados não são armazenados usando esquemas de tabela fixa” (BOICEA; RADULESCU; AGAPIN, 2012, p. 330, tradução nossa). Esses tipos de banco superam os modelos relacionais tradicionais no quesito performance, por isso se destacam em aplicações de grande escala, em que alto desempenho é fundamental.

O MongoDB é um banco orientado a documentos, que apresenta características como: alto desempenho, performance, recursos como tolerância a falhas de consistência, eficiência e também flexibilidade. Para Nayak, Poriya e Poojary (2013) é um banco de dados eficientemente adequado para aplicações como sistemas de gerenciamento de conteúdo, arquivamento e análise em tempo real.

Oliveira (2014) expõe algumas outras características:

> 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. Outra característica importante é que este modelo não depende de um esquema rígido, ou seja, não exige uma estrutura fixa como ocorre nos bancos relacionais. Assim, é possível que ocorra uma atualização na estrutura do documento, com a adição de novos campos, por exemplo, sem causar problemas ao banco de dados (OLIVEIRA, 2014, p. 190).

O MongoDB foi desenvolvido por uma empresa chamada 10gen e inicialmente lançado em 2009 sendo uma aplicação de código aberto. Os dados são armazenados principalmente no formato JSON. Esses documentos JSON, exemplo ilustrado na Figura 5, são formados por uma lista de elementos consistindo em nome e valor para cada um destes elementos (NAYAK; PORIYA; POOJARY, 2013).

3.6 REDIS

O Redis é um servidor de estrutura de dados que mantém um conjunto de dados armazenados na memória. Ele implementa essa estrutura de dados, permitindo que as chaves contenham cadeias, *hashes*, conjuntos ordenados binários, bem como listas (MACEDO; OLIVEIRA, 2011). Essa combinação de flexibilidade e aplicação faz do Redis uma ferramenta que se encaixa em aplicações que necessitam de um alto desempenho no armazenamento e consulta de dados.

Redis é um banco de dados não-relacional de alta velocidade que armazena um mapeamento de chaves para diferentes tipos de valores. O Redis suporta armazenamento persistente na memória, replicação para leitura em escala, bem como o suporte ao armazenamento de registros de dados em várias máquinas para dimensionar o desempenho do banco de dados (CARLSON, 2013).

Figura 5 – Exemplo de arquivo JSON utilizado com o MongoDB

```
{
    "_id" : ObjectId("5930b04bb3d4d334b2206d0b"),
    "_class" : "br.senac.tcs.iotdas.canal.domain.CanalMongo",
    "guid" : "ec973fac-0ab4-4371-b114-c40f9c0db27e",
    "canalId" : "4",
    "nome" : "Nome do Canal",
    "usuario" : {
        "_id" : "4",
        "usuario" : "usuario"
    },
    "campos" : [
        {
            "_id" : 13,
            "nome" : "nome campo 01",
            "guid" : "1ac32f72-6c9e-4a8c-b621-9933001749d2",
            "dados" : "dados para o campo 01"
        },
        {
            "_id" : 14,
            "nome" : "nome campo 02",
            "guid" : "b0e65d76-4319-4ffe-8028-3543fded15ff",
            "dados" : "dados para o campo 01"
        }
    ],
    "criado" : ISODate("2017-06-02T00:24:43.332Z"),
    "modificado" : ISODate("2017-06-02T00:24:43.332Z")
}
```

Fonte: Elaborada pelos autores baseada em Nayak, Poriya, Poojary (2013).

3.7 COMPUTAÇÃO EM NUVEM

O *Cloud* é um termo que em inglês se refere a “nuvem”. A relação com as tecnologias de informação está em uma série de padronizações em um modelo de disponibilização de tecnologias na internet. Segundo definição de Mell e Grance (2011), integrantes do *National Institute of Standards and Technology* (NIST):

> A computação em nuvem é um modelo que permite acesso ubíquo, conveniente e a pedido, através da rede em um conjunto de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente aprovisionados e liberados com o mínimo de esforço de gerenciamento e sem interação com o provedor de serviços (MELL; GRANCE, 2011, p. 2, tradução nossa).

Chee e Franklin Júnior (2013) definiram *cloud computing,* ou em tradução para o português, computação em nuvem, como um modelo de processamento de informação no qual recursos de computação administrada de forma centralizada são

oferecidos como serviço, à medida que são demandados, através da rede para diversos aplicativos com interatividade com o usuário.

Outras características em geral que podem ser levadas em consideração estão ligadas com a ideia que a *Cloud Computing* oferece recursos e serviços sob demanda, disponibilidade de acesso de qualquer lugar e momento através da internet, além de oferecer uma grande quantidade de recursos tornando assim a aplicação mais robusta e com maior valor agregado.

3.8 *SOFTWARE* COMO SERVIÇO

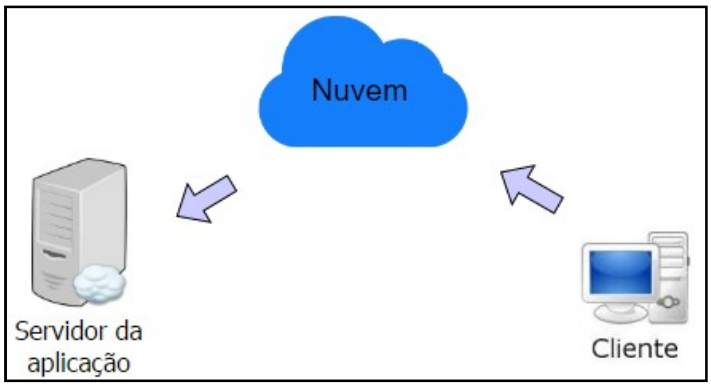
Software as a Service (SaaS) ou, em tradução para o português, *Software* como serviço, é uma forma de disponibilizar *software* em *cloud*. Pode ser considerada uma maneira de se utilizar softwares de maneira remota, conforme Figura 6. O *software* obtido como serviço é pago ao longo do tempo de utilização, em vez de adquirido e instalado em um computador ou servidor (FERREIRA, 2015).

Em resumo, de acordo com Sousa, Moreira e Machado (2009),

> No SaaS, o usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores, sistemas operacionais, armazenamento ou mesmo as características individuais da aplicação, exceto configurações específicas. Com isso, os desenvolvedores se concentram em inovação e não na infraestrutura, levando ao desenvolvimento rápido de sistemas de *software* (SOUZA; MOREIRA; MACHADO, 2009, p. 07).

Entre alguns exemplos de *softwares* que seguem esse conceito e são distribuídos como serviço, citados também por Opus *Software* (2015), podem ser o *Google Apps for Work*, que disponibiliza a seu usuário, planilhas, *e-mail*, apresentações acessados nos servidores do provedor de serviços. Podemos incluir também o *Microsoft Office* 365, seguindo a mesma linha do exemplo citado anteriormente, entre outros como o *SalesForce.com* que distribui uma forma de gestão de relacionamento com clientes através de um sistema CRM (*Customer Relationship Management*), entre outros exemplos.

Figura 6 – Disponibilização de *software* em *cloud*



Fonte: Elaborada pelos autores baseada em Souza, Moreira, Machado (2009).

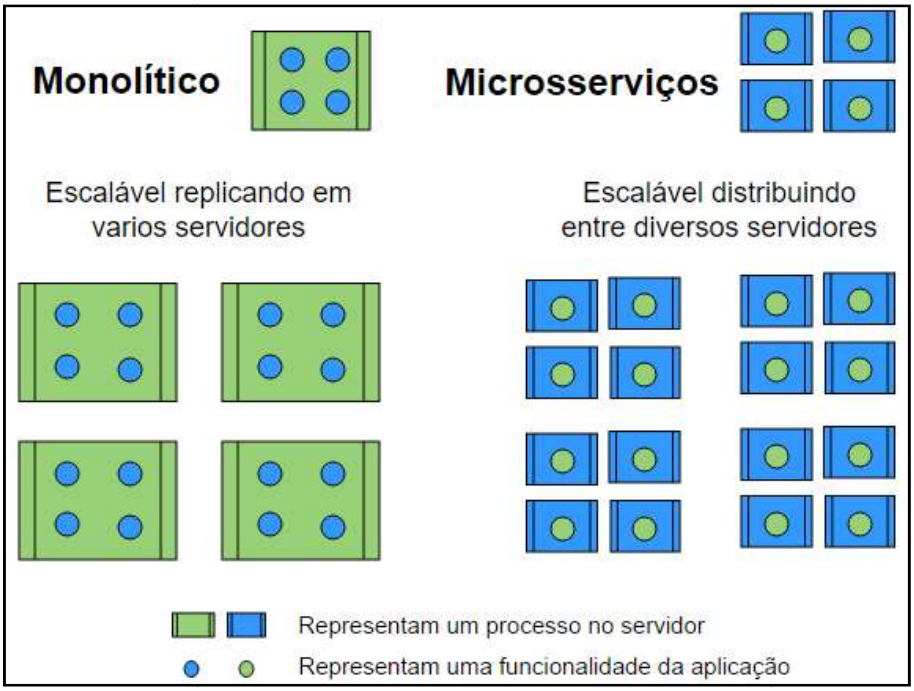
3.9 MICROSSERVIÇOS

O surgimento de novas tecnologias e metodologias de desenvolvimento levou as grandes empresas de TI (Tecnologia da Informação) a moldar novas estratégias e conceitos no que diz respeito à entrega e desenvolvimento de novos sistemas, exemplo das empresas como Google e Netflix. Estas empresas focam em versões contínuas e pequenas equipes, direcionadas em determinadas funcionalidades do projeto (NEWMAN, 2015).

De uma forma simples, porém efetiva, a arquitetura de microsserviços é definida como “uma abordagem para desenvolver um único aplicativo como um conjunto de pequenos serviços, cada um funcionando em seu próprio processo e se comunicando com mecanismos leves, muitas vezes uma API de recursos HTTP” (FOWLER; LEWIS, 2014, tradução nossa). Essa abordagem de dividir a aplicação em pequenos conjuntos de serviços facilita a organização das diversas funcionalidades que contemplam o projeto deste sistema.

Para falar sobre a arquitetura de microsserviços é interessante compará-la com a arquitetura chamada monolítica. Nesta, a aplicação é desenvolvida como uma única unidade construída basicamente em três partes principais: uma interface utilizada pelo cliente, um banco de dados e uma aplicação executada no servidor funcionando em um único processo, ao contrário dos microsserviços (FOWLER; LEWIS, 2014). Uma exemplificação pode ser vista na Figura 7.

Figura 7 – Arquitetura monolítica e de microsserviços



Fonte: Elaborada pelos autores baseada em FOWLER e LEWIS (2014).

Uma das características importantes, até levantada por Moreira e Beder (2015), descreve que:

> Cada micro serviço (sic) é uma entidade separada e autônoma, eles podem ser implantados em diferentes máquinas, e ainda assim trabalhar de forma coletiva, sendo capaz de executar seus processos de forma independente caso algum outro serviço falhe. Para garantir a separação entre eles, toda a comunicação é realizada por chamadas de rede (MOREIRA; BEDER, 2015, p. 210).

O objetivo principal dos microsserviços é desenvolver um projeto com funcionalidades independentes, com grande desempenho e escalabilidade, tornando a aplicação robusta e sua manutenção mais rápida (NEWMAN, 2015). Os microsserviços tornam-se uma forma eficaz e eficiente de implementação de um software, pois podem ser desenvolvidos em tecnologias de programação distintas e seguindo uma demanda e necessidade.

Segundo Coulouris et al. (2013), a escalabilidade de um sistema pode ser considerada quando há um aumento significativo no número de recursos e/ou no número de usuários e o sistema continua eficiente. Em outras palavras, escalabilidade é o quanto um sistema ou parte dele, está preparado para crescer.

4 DESENVOLVIMENTO

Em pauta no primeiro momento estão os requisitos necessários para o desenvolvimento da aplicação, baseados nos objetivos específicos de criação do sistema e desenvolvimento da API para integração das funcionalidades.

Seguido pelo momento posterior, em nível de projeto, usado para definição e criação do modelo de arquitetura da aplicação e baseado no objetivo de dispor a aplicação por meio da arquitetura de microsserviços.

Neste projeto de sistema, o armazenamento e a consulta de dados, são itens que possuem certa importância. Um banco de dados com potencial de armazenamento deve compor parte deste projeto. Neste caso, o MongoDB foi escolhido, ele apresenta essas características com um bom desempenho e eficiência.

A partir deste ponto, o desenvolvimento do projeto será descrito. Será abordado desde o início, com os requisitos levantados, até a devida implementação do *software*.

4.1 REQUISITOS

Todo projeto desenvolvido apresenta necessidades que precisam ser atendidas para a construção do mesmo. No Quadro 1 estão relacionados os principais requisitos funcionais e sua respectiva especificação de caso de uso relacionado. Por sua vez, as especificações dos casos de uso podem ser encontradas no apêndice A.

Quadro 1 – Requisitos funcionais do sistema

Identificação	**Requisitos**	**UC**
RF-01	O sistema deve permitir o controle, criação, alteração e exclusão de registros de usuário.	UC-01
RF-02	O sistema deve permitir que o usuário crie, altere e exclua o determinado canal para o armazenamento dos dados.	UC-02
RF-03	O sistema deve permitir que o usuário adicione, remova	UC-02

	e altere os campos dos canais para armazenamento.	
RF-04	O sistema deve permitir que o usuário adicione, remova e altere os dados do canal.	UC-02
RF-05	O sistema deverá enviar os dados por POST e GET.	UC-03
RF-06	Cada canal inserido deve possuir características / atributos equivalentes a: descrição, quantidade de campos definidos pelo usuário, nome para identificação.	UC-04

Fonte: Elaborado pelos autores (2017).

Algumas características importantes influenciam na qualidade do protótipo sendo desenvolvido e são representadas com os requisitos não-funcionais. No Quadro 2 estão descritos os principais requisitos não-funcionais implícitos no protótipo do sistema.

Quadro 2 – Requisitos não-funcionais do sistema

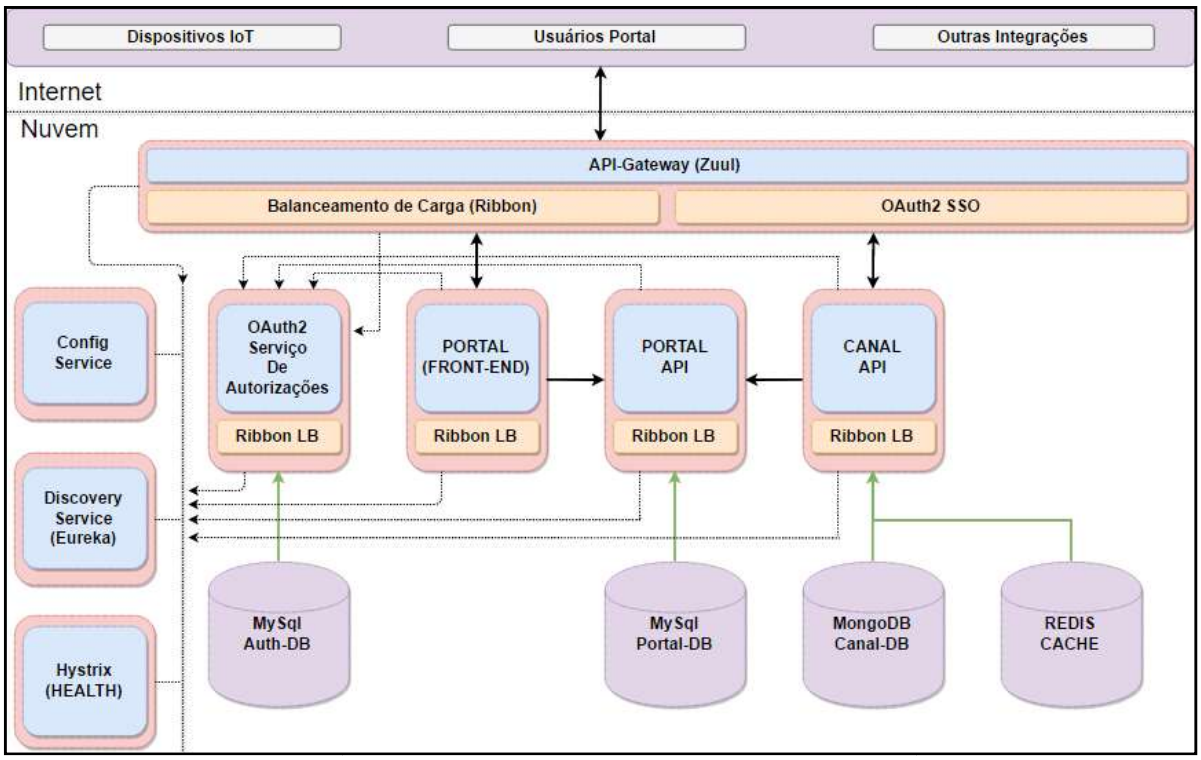
Identificação	**Requisitos**
RNF-01	O sistema deve possuir características que sigam os padrões de segurança da informação. Senhas de acesso e identificação. Diferentes tipos de usuários. Diferenciação de papéis na aplicação como, por exemplo: administrador do sistema, usuários finais etc.
RNF-02	O sistema deve utilizar como banco de dados relacional, que armazenará as informações cadastrais, o banco de dados MySQL.
RNF-03	O sistema deve utilizar como banco de dados não-relacional, que armazenará os dados gerais dos dispositivos IoT, o banco de dados MongoDB.
RNF-04	A arquitetura da aplicação deve seguir os conceitos da arquitetura de microsserviços.
RNF-05	A aplicação deve ser disponibilizada seguindo o modelo de Computação em nuvem.
RNF-06	A comunicação e integração entre os serviços na API devem utilizar a arquitetura REST.
RNF-07	Aliando as boas práticas, o projeto tende a seguir padrões de segurança ISO-IEC-27001.

Fonte: Elaborado pelos autores (2017).

4.2 DEFINIÇÃO DA ARQUITETURA

A base da arquitetura do projeto está estabelecida nos microsserviços. Cada um destes serviços está responsável por uma determinada funcionalidade dentro da aplicação. De maneira geral, a Figura 8 ilustra como é a base da estrutura dos microsserviços no projeto.

Figura 8 – Ilustração da arquitetura da solução



Fonte: Elaborada pelos autores (2017).

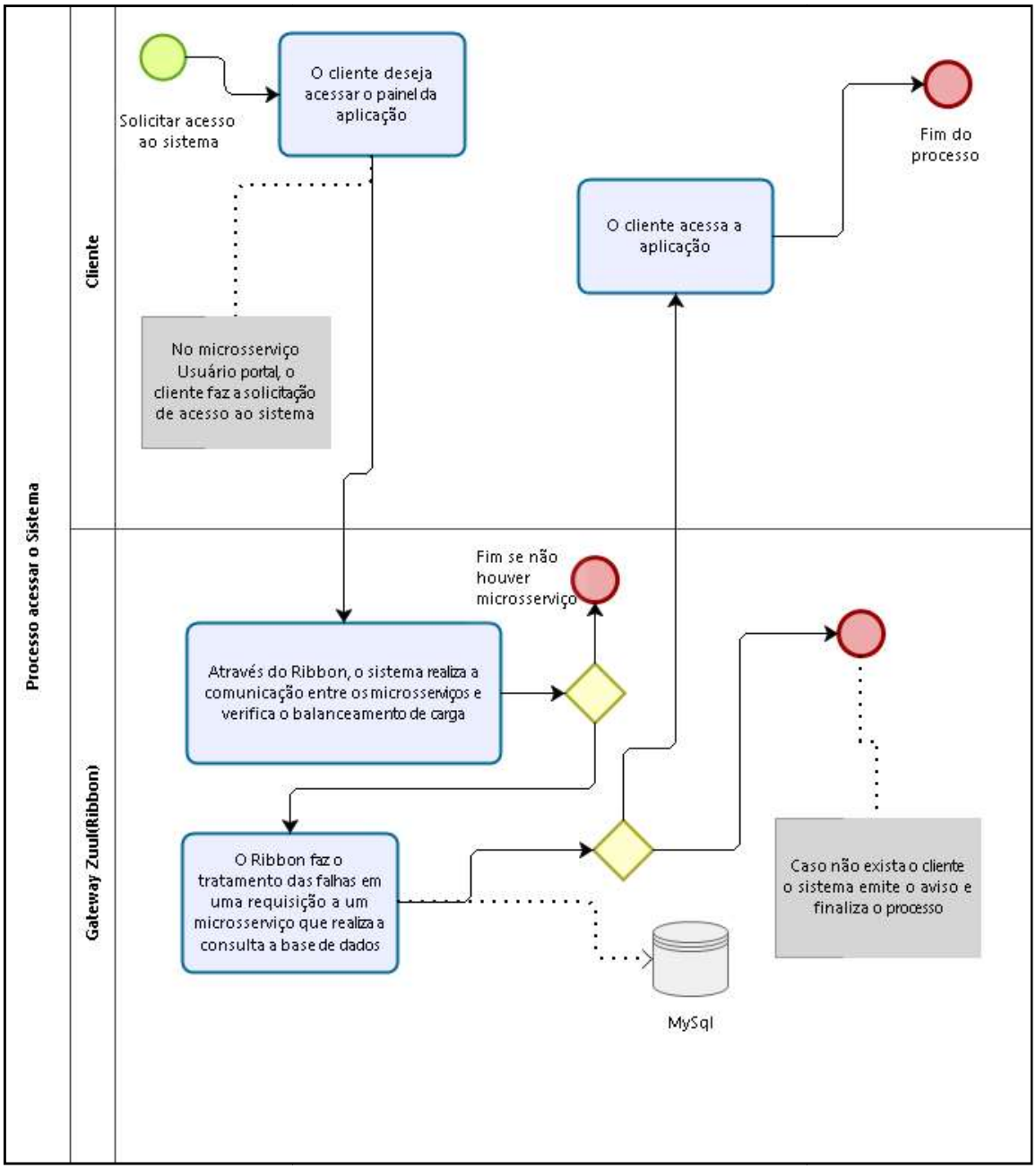
4.3 DEFINIÇÃO DA ESTRUTURA FUNCIONAL

A estrutura funcional se resume na arquitetura de microsserviços. Tem como ponto inicial a interface do cliente, onde o mesmo se conecta a API, para gerenciar seus dados. Através do *Gateway Zuul,* o sistema faz o roteamento das informações, designando cada requisição para seus respectivos serviços, armazenando os dados que se referem ao cliente em um banco MySQL, e o que refere-se aos canais, por ser uma grande quantidade de dados, serão armazenados no banco MongoDB.

Conforme demonstrado na Figura 9, o cliente faz o acesso ao painel da aplicação através de uma autenticação. O sistema faz uma requisição e a

verificação dos dados. Caso seja verdadeiro, retorna o *login* com sucesso, senão informa usuário inexistente.

Figura 9 – Modelo de processo de acesso a aplicação

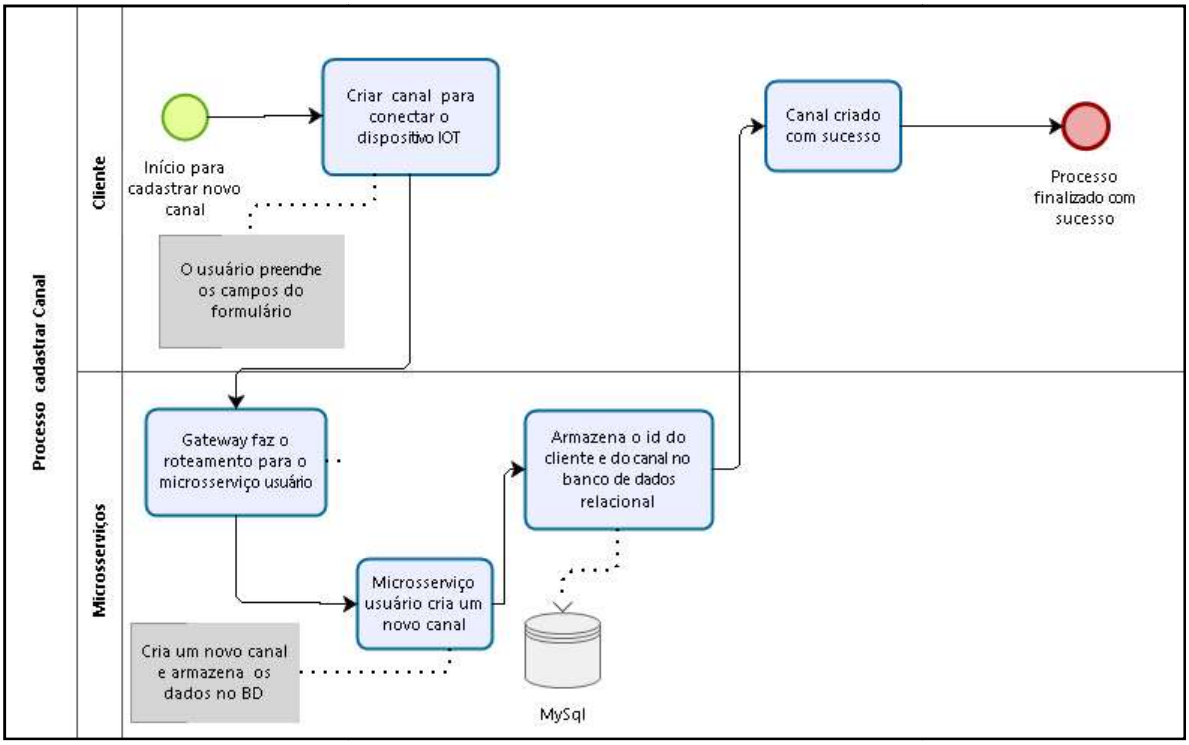


Fonte: Elaborada pelos autores (2017).

Na Figura 10 demonstra-se o processo de cadastrar um canal, quando o cliente faz a solicitação via painel da aplicação, o sistema faz uma solicitação a API que realiza o direcionamento, a validação dos dados e posteriormente o armazenamento no banco de dados MongoDB. O ID e o código GUID para

identificação do campo, canal e do usuário é armazenado no banco de dados relacional. No banco não-relacional ficam armazenados os dados que serão mantidos dos determinados canais.

Figura 10 – Modelo de processo de cadastro de canais

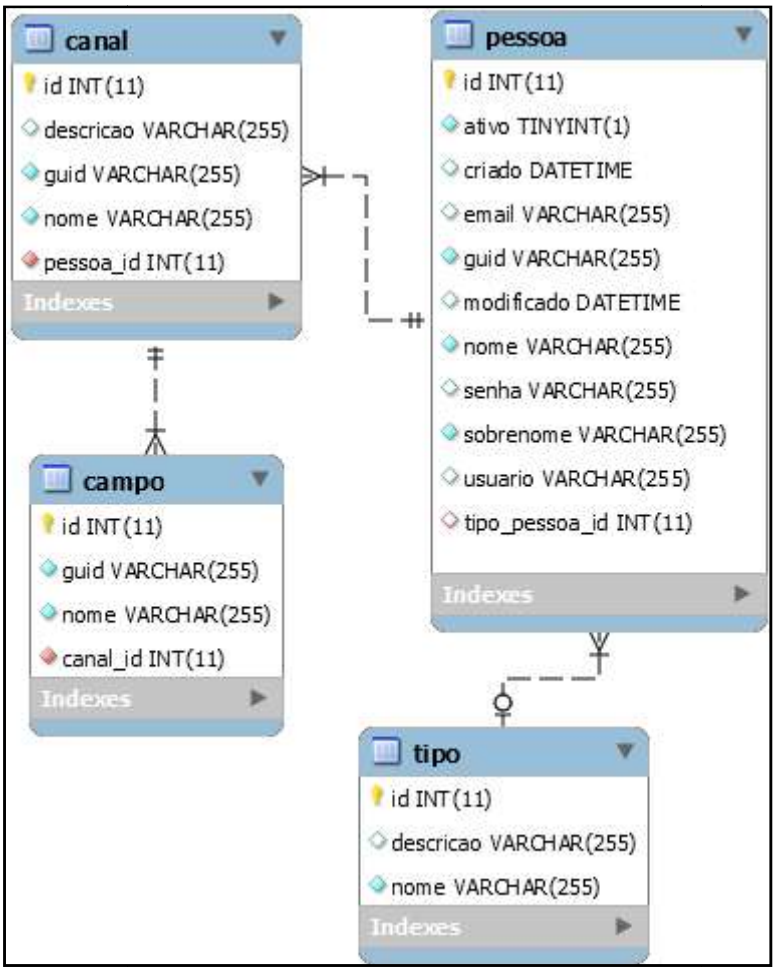


Fonte: Elaborada pelos autores (2017).

4.4 ENTIDADES DO BANCO DE DADOS RELACIONAL

A estrutura do banco de dados relacional, que compreende este protótipo, é de certa forma simples. Caracteriza-se basicamente pelas tabelas representadas no modelo ilustrado na Figura 11. Possui as tabelas nominadas: Pessoa, Canal, Campo e Tipo.

Figura 11 – Entidades no banco relacional



Fonte: Elaborada pelos autores (2017).

A entidade Pessoa é responsável por armazenar dados atribuídos aos clientes. Dados cadastrais compostos por: nome, sobrenome, e-mail, tipo de usuário (campo relacionado com a entidade Tipo), dados para acesso como *login* e senha, dados de criação e modificação dos registros de pessoa, um campo com identificador único para utilização com o banco não-relacional (GUID). Possui um relacionamento denominado um para muitos, para com a entidade Canal, representando que um cliente pode ter vários Canais associados a ele e o Canal possui apenas um cliente associado.

A entidade Canal é a responsável por armazenar atributos referentes ao canal como: descrição, nome, ID para identificação do cliente que está vinculado a este canal e GUID que é um código que será utilizado para identificação deste canal no banco de dados não-relacional. Esta entidade possui um relacionamento um para muitos, para com a entidade Campos, representando que um canal consegue

manter vários campos associados a ele e o Campo pertence a apenas um determinado canal.

A entidade Campo está com o propósito de armazenar dados de cadastro dos campos que cada canal possui. Manterá dados de descrição do campo, juntamente com o código de ligação com o MongoDB.

Neste modelo também existe a entidade Tipo. Esta foi criada com o intuito de normalizar os registros de identificação dos tipos de usuários. Ela mantém dados como: nome e descrição do respectivo tipo de usuário. A entidade Tipo está relacionada com a entidade pessoa.

4.5 ESTRUTURA DO BANCO DE DADOS NÃO-RELACIONAL

No banco não-relacional orientado a documentos (MongoDB), os dados são mantidos em formato de documentos. De maneira geral, basicamente o MongoDB recebe e retorna dados seguindo um arquivo JSON como o representado na Figura 12.

Figura 12 – JSON utilizado para envio ao MongoDB

Fonte: Elaborada pelos autores (2017).

Através do arquivo JSON enviado, o MongoDB persiste os dados referentes aos canais e os dados que cada um destes canais manterá. A Figura 12, por

exemplo, mostra um exemplo de JSON que é enviado para a aplicação através da API para a armazenagem no banco de dados não-relacional.

Ao realizar uma consulta, o MongoDB também retorna um documento JSON, desta vez como resposta a consulta. Este documento JSON é idêntico ao apresentado na Figura 13. Sendo retornado como uma coleção dos dados contidos no banco não-relacional.

Figura 13 – JSON de resposta a uma consulta ao MongoDB

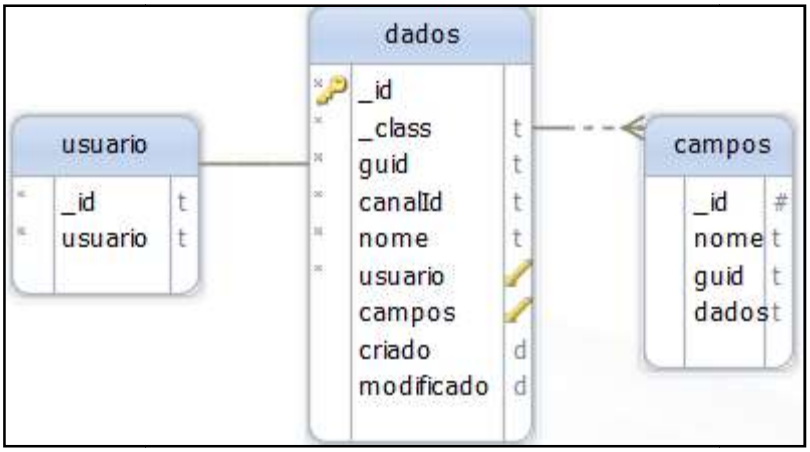

```
{
    "_id" : ObjectId("5929b0dd0d97f509f892abdd"),
    "_class" : "br.senac.tcs.iotdas.canal.domain.CanalMongo",
    "guid" : "200912db-419a-11e7-845c-f8a963665830",
    "canalId" : "235",
    "nome" : "Zoolab",
    "usuario" : {
        "_id" : "4",
        "usuario" : "zskarr3"
    },
    "campos" : [
        {
            "_id" : 57,
            "nome" : "sapien",
            "guid" : "28c62026-419a-11e7-845c-f8a963665830",
            "dados" : "-7186895738903539078"
        },
        {
            "_id" : 593,
            "nome" : "semper",
            "guid" : "36051ae7-419a-11e7-845c-f8a963665830",
            "dados" : "9216136058765651807"
        }
    ],
    "criado" : ISODate("2017-05-27T17:01:16.420Z"),
    "modificado" : ISODate("2017-05-27T17:01:16.420Z")
}
```


Fonte: Elaborada pelos autores (2017).

O esquema da estrutura do documento JSON de retorno do MongoDB pode ser representado com um diagrama que se parece com o modelo de entidade

apresentado para bancos relacionais. O modelo representado na Figura 14 mostra a estrutura da coleção de dados que este arquivo JSON possui.

Figura 14 – Estrutura da coleção de dados



Fonte: Elaborada pelos autores (2017).

4.6 DETALHAMENTO DOS MICROSSERVIÇOS

A estrutura da arquitetura dessa aplicação divide-se em várias partes, cada uma delas com uma responsabilidade diferente. Em seguida cada um dos microsserviços que constituem o projeto são detalhados e apresentados para entendimento de suas funcionalidades.

O primeiro microsserviço que deve ser inicializado na aplicação é o Serviço de Configuração. É nele onde todas as configurações necessárias para os demais microsserviços estarão centralizadas. É ele que vai prover as configurações relacionadas a qual porta os microsserviços vão utilizar, em que serviço de descoberta vão conectar, configurações para conexão com banco de dados, quais recursos e microsserviços vão chamar para prover as funcionalidades de autenticação e segurança, e quaisquer outras configurações relacionadas ao Spring Boot e ao Netflix OSS.

Após o serviço de configuração, deve ser inicializado o serviço de descoberta (*Discovery*). Utilizando o *framework Spring Cloud* Netflix Eureka, este serviço é responsável pela localização dos microsserviços com objetivo de balanceamento de carga e tratamento de falhas, e também torna as interações entre eles muito mais fáceis. Com ele é possível escalar múltiplos serviços de mesma funcionalidade para

atender uma demanda de crescimento, e todo o resto dos serviços vão se comunicar com eles, pois os clientes do Eureka também possuem um balanceador de carga incorporado que faz balanceamento básico de carga *round-robin*.

O serviço de usuário é o microsserviço responsável pela API de interação com o banco de dados relacional. Todo o cadastro, edição, exclusão e consulta das entidades (pessoa, tipo, canal e campo) é feita por este serviço, não existe interação direta de qualquer usuário ou dispositivo com ele, apenas outros microsserviços se comunicarão com ele através de sua API Rest.

O serviço que faz a integração com o usuário cliente (*front-end*) é o Serviço de *User Interface* (UI). Este microsserviço disponibiliza uma Interface Gráfica *web* onde são feitos todos os cadastros de usuários e configurações do sistema pelo administrador e também onde o usuário pode realizar a administração e visualização de seus canais e campos e chaves de identificação global.

O microsserviço que disponibiliza a API responsável pela comunicação com os dispositivos IoT e faz a interação com o banco de dados não-relacional, onde serão salvos os dados dos dispositivos IoT, é o Serviço de Canal. Quando um dispositivo for enviar algum dado para ser salvo será necessário enviar para a API uma requisição POST com os dados em formato JSON contendo os códigos GUID de canal e seus respectivos campos, essa requisição também deve ser feita através de uma URL que precisa conter o GUID do usuário dono do canal. Este serviço faz todas as validações dos códigos GUID de usuário, canais e campos chamando o Serviço de Usuário que se comunica com o banco de dados relacional, para assim gerar um documento para ser armazenado no banco de dados não-relacional.

O Ribbon é responsável por fazer um roteamento e balanceamento de carga dos microsserviços de forma mais avançada ele está incluso em cada microsserviço e utiliza informações do serviço de descoberta, verificando qual serviço necessita mais da aplicação, priorizando conforme a quantidade de requisições.

4.7 DIAGRAMAS DE CLASSE

A construção da aplicação envolveu uma série de classes utilizadas para seu desenvolvimento. Estas classes podem ser conferidas nos diagramas de classes contidos no Apêndice B. O conteúdo deste apêndice compreende os principais microsserviços deste protótipo de *software.*

4.8 PROTÓTIPOS DE TELAS

Neste tópico, serão abordadas as principais telas da aplicação. Serão descritas as funcionalidades de cada tela, referenciando devidamente a respectiva figura da mesma. As ilustrações se encontram no Apêndice C.

No painel administrativo o usuário terá acesso a todas as informações de gerenciamento de seus dispositivos. O usuário que é administrador do sistema consegue realizar a visualização de todos os usuários cadastrados na base de dados, conforme Figura 1. Nela é possível visualizar os usuários inativos, identificados na lista com o destaque em vermelho, e os ativos com a cor branca.

Nessa tela, também é possível realizar o cadastro de um novo usuário, conforme Figura 2. Posteriormente, conforme Figura 3, é possível realizar a edição do usuário e também a exclusão do mesmo, representado na Figura 4.

Na Figura 5, é possível visualizar a tela de informações do usuário, onde são listados todos os canais associados a ele. Nessa tela, é possível realizar o cadastro de um novo canal, visto na Figura 6, a edição de um canal, conforme Figura 7 e a exclusão de um canal, visto na Figura 8.

Conforme ilustrado na Figura 9, essa é a tela de informações do canal. Onde é possível gerar um exemplo de JSON para o envio das informações desse canal e seus respectivos campos para a API. Esse JSON é baseado nas informações de chave do canal e dos campos. Também é possível gerar uma nova chave para o canal nessa tela.

Todos os campos pertencentes ao canal, também são listados da tela da Figura 9. Onde o usuário tem a possibilidade de criar vários campos, onde cada um receberá uma informação que deseja coletar do dispositivo IoT, conforme demonstrado na Figura 10. Também é possível realizar a edição, visto na Figura 11, e a exclusão de algum campo do canal, apresentado na Figura 12.

Os campos também possuem uma tela para visualização das informações deles. Na Figura 13, é possível acompanhar essa tela, onde o usuário poderá gerar uma nova chave para o campo visualizado.

O administrador do sistema poderá criar tipos de papéis para os usuários, para posteriormente aplicar as permissões a cada usuário, conforme visto na Figura 15. Estes sendo listados para a visualização na Figura 14. O administrador também

terá acesso a editar os tipos de usuários, visto na Figura 16, e excluir eles, conforme Figura 17.

4.9 RESULTADOS E MÉTRICAS

Os resultados da aplicação estão em métricas que foram geradas a partir de um teste de *stress* efetuado na API. Os dados gerados à partir desse teste foram possíveis de serem analisados devido a algumas ferramentas que foram utilizadas no processo. Estas que auxiliam na mensuração e na quantificação de resultados desse teste feito na API.

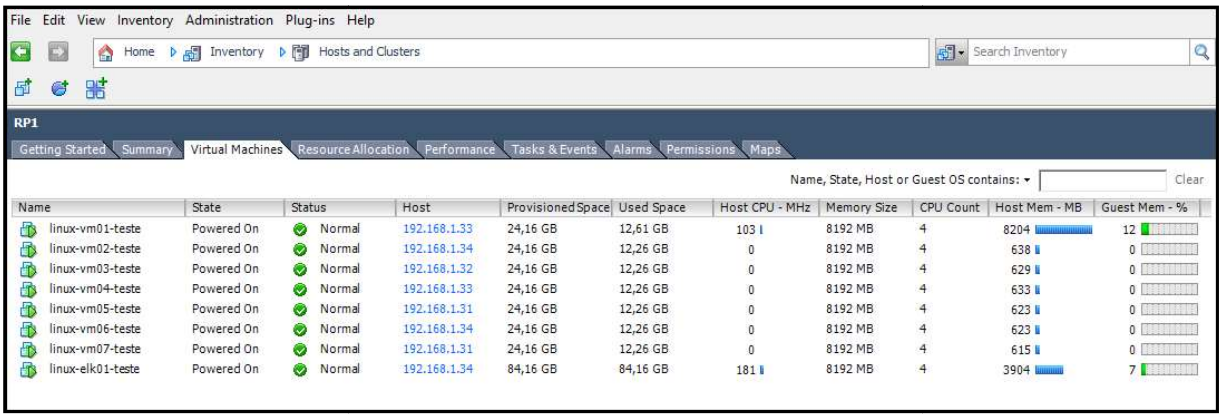
As métricas foram geradas a partir de algumas ferramentas. Entre elas o *Postman* utilizado para realização de teste de carga na APIs, simulando o envio e retorno de arquivos JSON gerando métricas de desempenho e falhas. A partir de uma outra ferramenta comercial chamada BlazeMeter, com ela foi gerado relatórios com testes de carga simulando acesso simultâneas de diversos usuários.

Para gerar as métricas relacionadas a utilização dos recursos das máquinas virtuais tais como uso de CPU, memória, estatísticas do banco foi utilizado: Elasticsearch (*document* DB), Kibana (*dashboard*, visualização) e Metricbeat (cliente que envia dados do *node* para o *Elasticsearch*). Todo estes conjuntos estavam disponível em uma máquinas virtuais conectadas na mesma nuvem.

Para executar os testes foi utilizado um ambiente real de nuvem, utilizando *software* de virtualização da VMWARE. Foram realizados em 4 servidores físicos, cada um com 2 processadores de 28 *cores* cada, 512Gb de memória, conectados a um *storage* de rede por ISCSI a 10Gbps. Na Figura 15, estão as 7 máquinas virtuais criadas para a virtualização e simulação de ambiente de nuvem, cada uma com 8gb de memória e 4 vCPUs.

Estas 7 máquinas virtuais (VM) foram divididas da seguinte maneira: 2 VMs para o microsserviço de usuário (linux-vm03-teste e linux-vm06-teste); 2 VMs para o microsserviço de canal (linux-vm04-teste e linux-vm05-teste); 1 VM para cada banco de dados, MySQL (linux-vm01-teste) e MongoDB (linux-vm02-teste); 1 VM para os microsserviços de *Config*, *Discovery*, *Gateway* e UI (linux-vm07-teste).

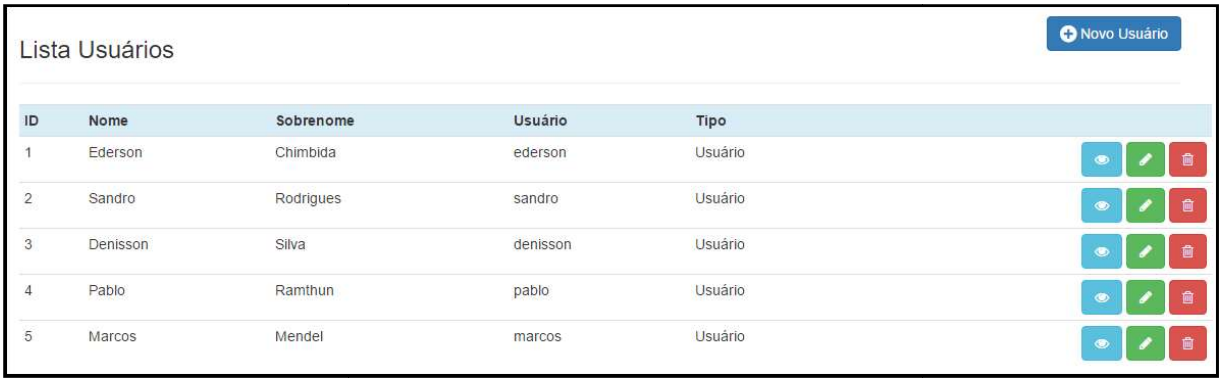
Figura 15 – Máquinas virtuais utilizados para os testes



Fonte: Elaborada pelos autores (2017).

Foram criados 5 usuário no sistema. Em cada um dos usuários foram criados canais, cada qual com seus campos. Conforme ilustrado na Figura 16.

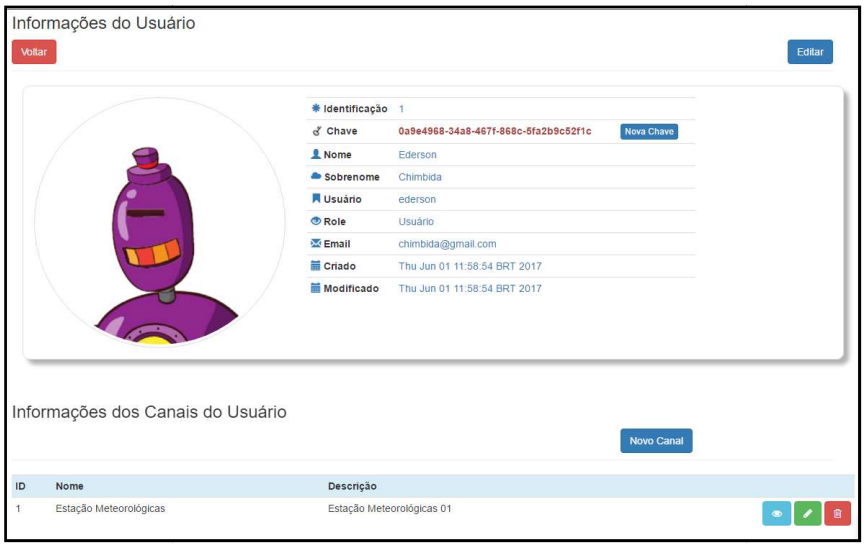
Figura 16 – Lista de usuários cadastrados para os testes



Fonte: Elaborada pelos autores (2017).

Para o usuário “Ederson” foi criado um canal para coleta de dados de uma “Estação Meteorológica”, Ilustrado na Figura 17.

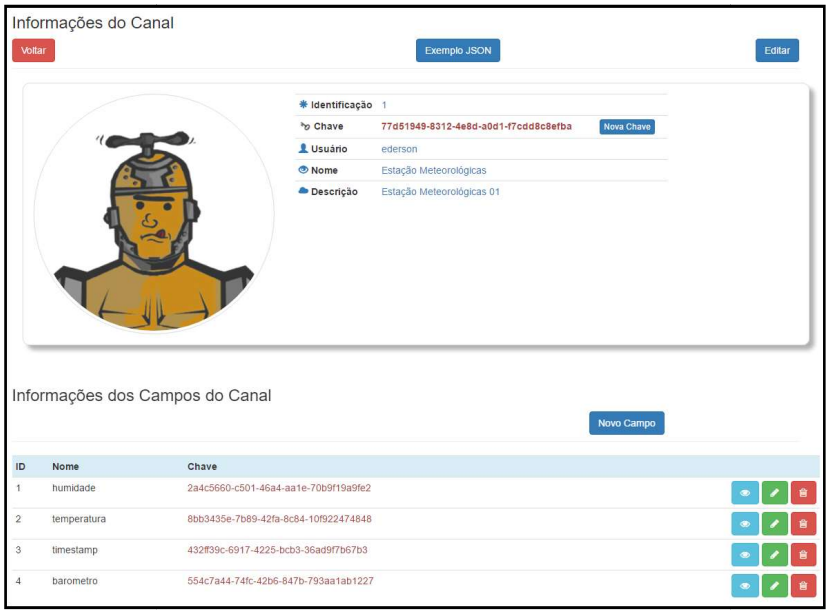
Figura 17 – Informações do usuário Ederson



Fonte: Elaborada pelos autores (2017).

No canal “Estação Meteorológica” foram criados 4 campos que corresponderão ao dados enviados para o sistema. Os campos “umidade”, “temperatura” e “barômetro” recebem valores inteiros randômicos e o campo “*timestamp”* vai receber um valor de tempo, conforme Figura 18.

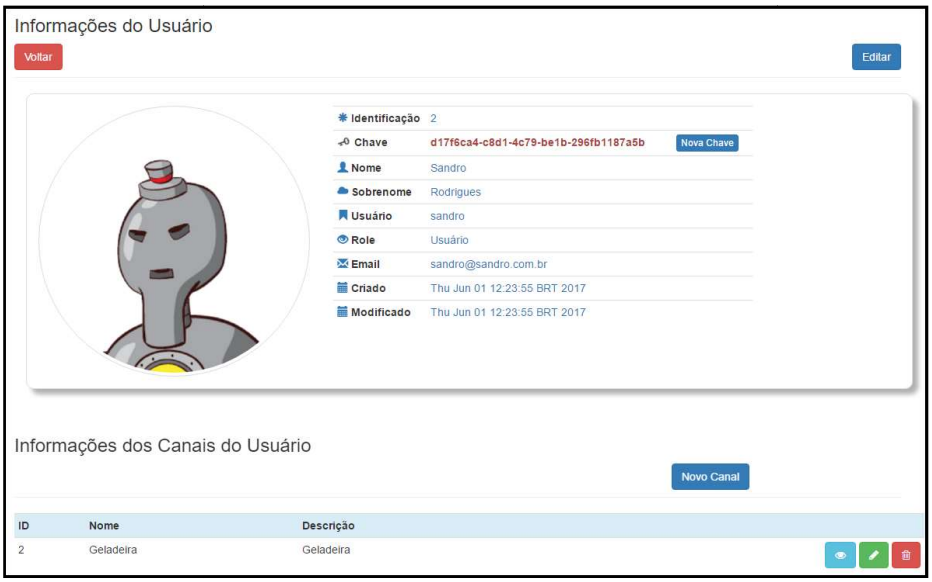
Figura 18 – Informações do canal Estação Meteorológica



Fonte: Elaborada pelos autores (2017).

Para o usuário “Sandro” foi criado um canal para coleta de dados de uma “Geladeira”, Ilustrado na Figura 19.

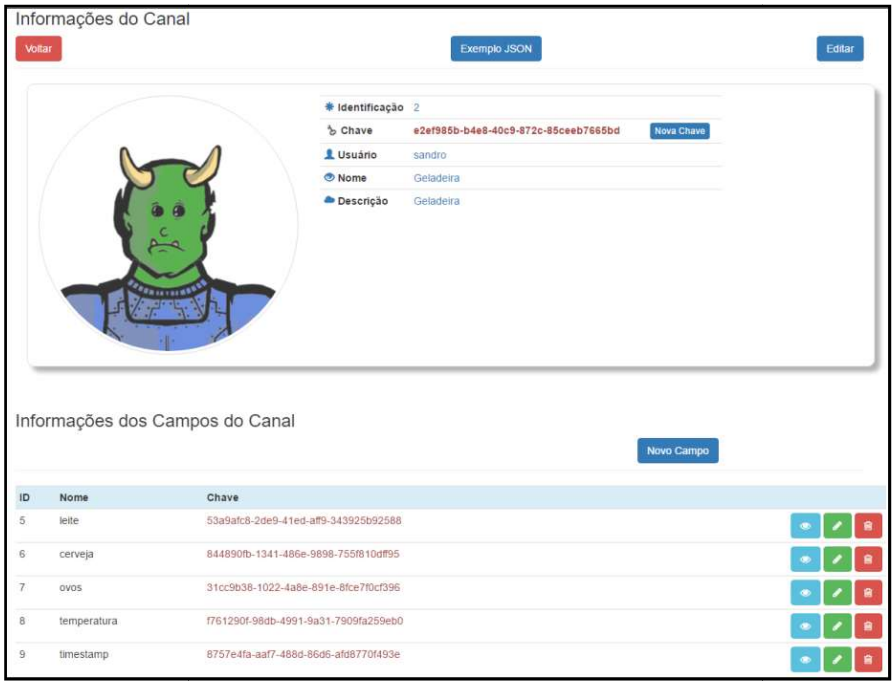
Figura 19 – Informações do usuário Sandro



Fonte: Elaborada pelos autores (2017).

No canal “Geladeira” foram criados 5 campos que corresponderão ao dados enviados para o sistema. Os campos “leite”, “cerveja”, “ovos” e “temperatura” recebem valores inteiros randômicos e o campo ”*timestamp”* vai receber um valor de tempo, conforme Figura 20.

Figura 20 – Informações do canal Geladeira



Fonte: Elaborada pelos autores (2017).

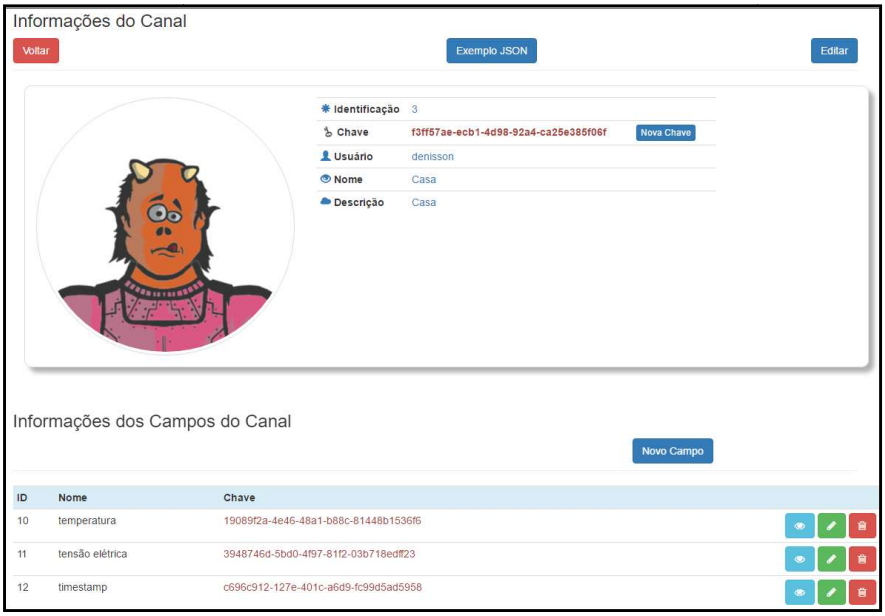
Para o usuário “Denisson” foi criado um canal para coleta de dados de uma “Casa”, ilustração na Figura 21.

Figura 21 – Informações do usuário Denisson

Fonte: Elaborada pelos autores (2017).

No canal “Casa” foram criados 3 campos que corresponderão ao dados enviados para o sistema. Os campos “temperatura” e “tensão elétrica” recebem valores inteiros randômicos e o campo “*timestamp*” vai receber um valor de tempo, conforme Figura 22.

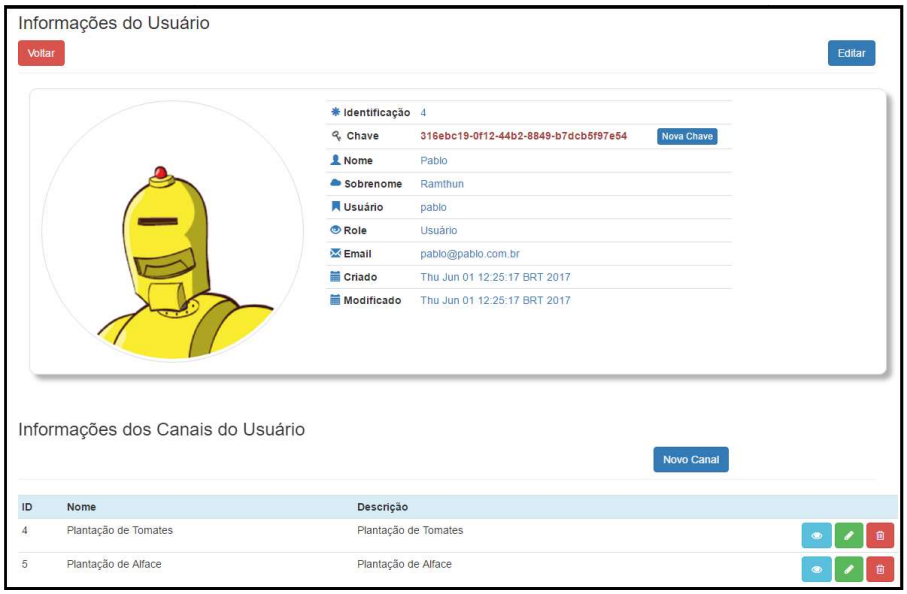
Figura 22 – Informações do canal Casa



Fonte: Elaborada pelos autores (2017).

Para o usuário “Pablo” foram criados 2 canais para coleta de dados de uma “Plantação de Tomates” e uma “Plantação de Alface”, conforme Figura 23.

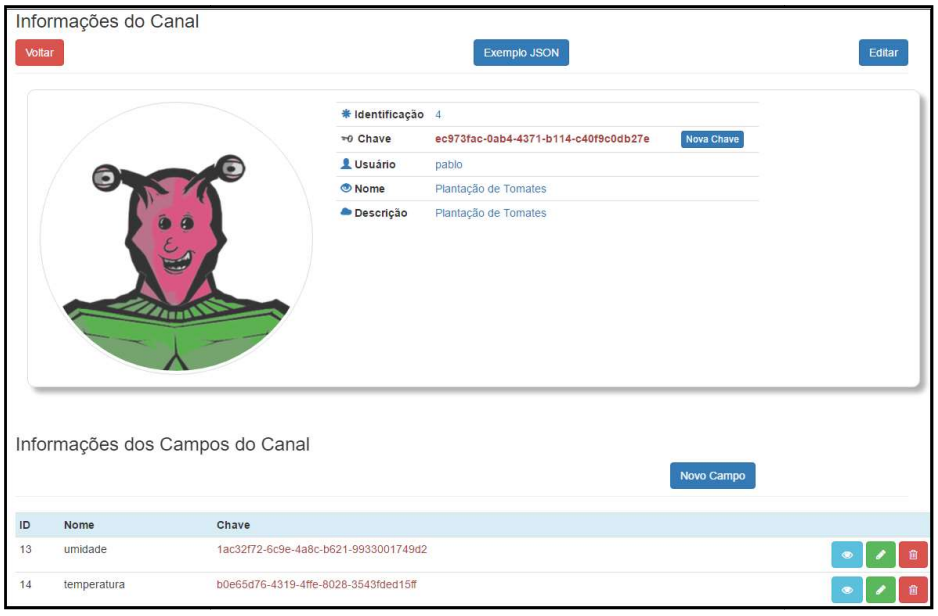
Figura 23 – Informações do usuário Pablo



Fonte: Elaborada pelos autores (2017).

No canal “Plantação de Tomates” foram criados 2 campos que corresponderão ao dados enviados para o sistema. Os campos “umidade” e “temperatura” recebem valores inteiros randômicos, conforme Figura 24.

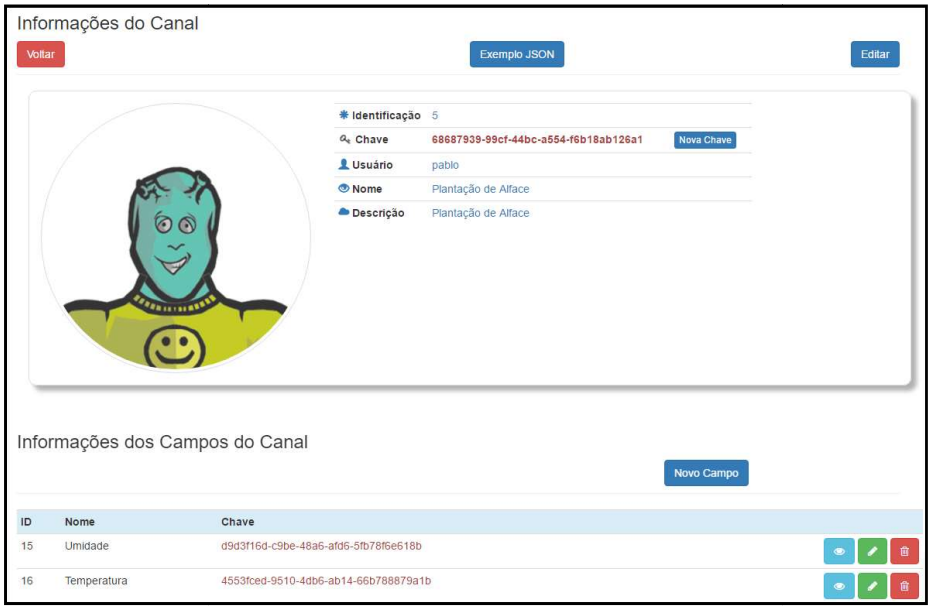
Figura 24 – Informações do canal Plantação de Tomates



Fonte: Elaborada pelos autores (2017).

No canal “Plantação de Alface” também foram criados 2 campos que corresponderão ao dados enviados para o sistema. Os campos “umidade” e “temperatura” recebem valores inteiros randômicos, conforme Figura 25.

Figura 25 – Informações do canal Plantação de Alface



Fonte: Elaborada pelos autores (2017).

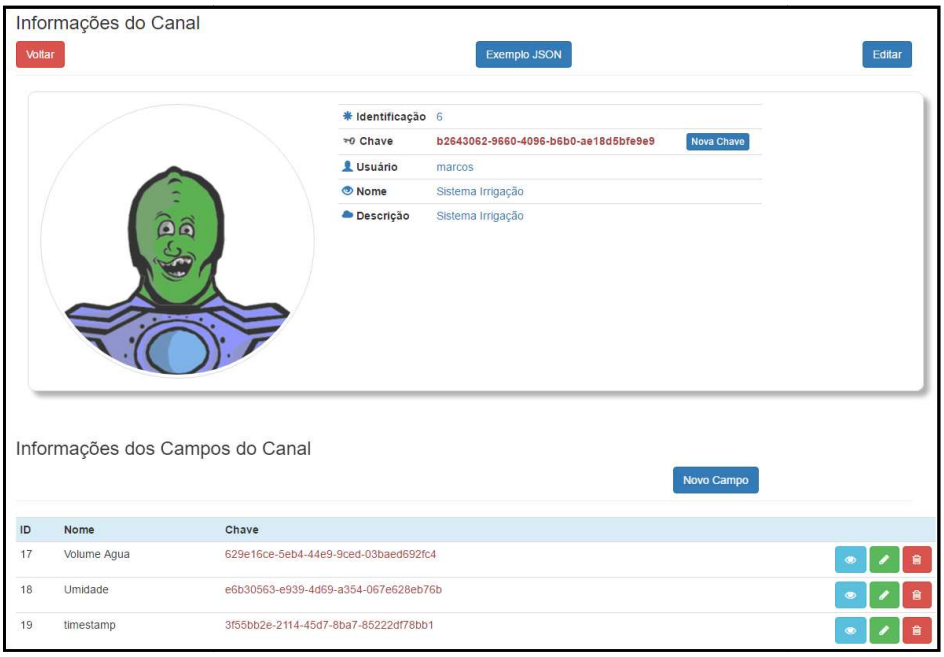
Para o usuário “Marcos” foi criado um canal para coleta de dados de um “Sistema de Irrigação”, conforme Figura 26.

Figura 26 – Informações do usuário Marcos

Fonte: Elaborada pelos autores (2017).

No canal “Sistema de Irrigação” foi criado 3 campos que corresponderão ao dados enviados para o sistema, conforme Figura 27. Os campos “volume água” e “umidade” recebem valores inteiros randômicos e o campo “*timestamp”* vai receber um valor temporal.

Figura 27 – Informações do canal Sistema de Irrigação



Fonte: Elaborada pelos autores (2017).

O teste foi realizado com uma sequência de 10 requisições de GET e POST. Estas requisições foram executadas durante 20 minutos em um servidor remoto hospedado em uma nuvem pública (Amazon EC2 - Instanciado em São Paulo), todas as requisições de POST foram enviadas em seu corpo informações no formato de um JSON para serem inseridas em um canal e simular um dispositivo IoT e para as requisições GET, todas elas foram chamadas fazendo paginação dos dados exibidos. Em todas as requisições foram feitos 2 testes, um verificando o código de status esperado e outra testando se a requisição levou menos de 200 milissegundos para ser executada.

Na Figura 28 pode-se verificar um exemplo de um arquivo JSON enviado para fazer o POST.

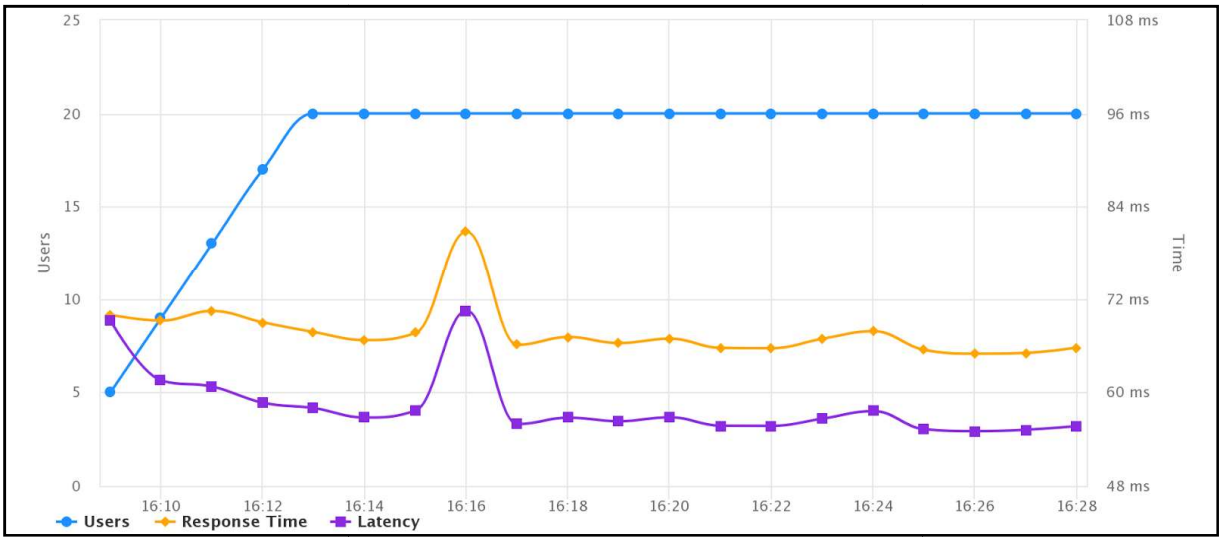
Figura 28 – Exemplo de JSON para o POST

```
curl -X POST \
  http://10.244.1.14:8080/api/0a9e4968-34a8-467f-868c-5fa2b9c52f1c \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/json' \
  -H 'postman-token: ace73d7c-6f99-7644-7d18-e805a95cfce4' \
  -d '{
    "guid" : "77d51949-8312-4e8d-a0d1-f7cdd8c8efba",
    "campos" :
    [
        {
            "guid" : "2a4c5660-c501-46a4-aa1e-70b9f19a9fe2",
            "dados" : "805"
        },
        {
            "guid" : "8bb3435e-7b89-42fa-8c84-10f922474848",
            "dados" : "805"
        },
        {
            "guid" : "432ff39c-6917-4225-bcb3-36ad9f7b67b3",
            "dados" : "1496415154"
        },
        {
            "guid" : "554c7a44-74fc-42b6-847b-793aa1ab1227",
            "dados" : "805"
        }
    ]
}'
```

Fonte: Elaborada pelos autores (2017).

As 10 requisições foram executadas simultaneamente simulando 20 usuários em um período de 20 minutos. Na Figura 29 são demonstrados os resultados obtidos através do teste realizado. Nele estão o tempo médio de respostas, os erros, quantos usuários estão conectados assim como todas as requisições foram finalizadas com sucesso.

Figura 29 – Resultado dos testes realizados na API



Fonte: Elaborada pelos autores (2017).

As 10 requisições foram executadas simultaneamente simulando 20 usuários em um período de 20 minutos. No Apêndice D, são demonstrados os resultados obtidos através do teste de latência realizado na API. Todas as requisições foram finalizadas com sucesso. No Apêndice E, verifica-se os resultados dos testes por requisições de POST, cada uma com seu tempo médio de resposta e quantos bytes consumiu. No Apêndice F, podem ser observados os resultados dos testes por GET, cada uma com seu tempo médio de resposta e quantos bytes consumiu.

No Apêndice G, segue uma tabela com estatísticas para cada um dos tipos de requisições GET e POST realizados, bem como, os indicadores e resultados identificados nas requisições. Além do acompanhamento da API, também foi gerado análises referentes aos servidores e aos bancos de dados utilizados nos testes.

Na Figura 30 pode-se ver o uso de CPU, memória e disco dos servidores utilizados durante o teste.

Figura 30 – Resultado do uso de CPU, memória e disco

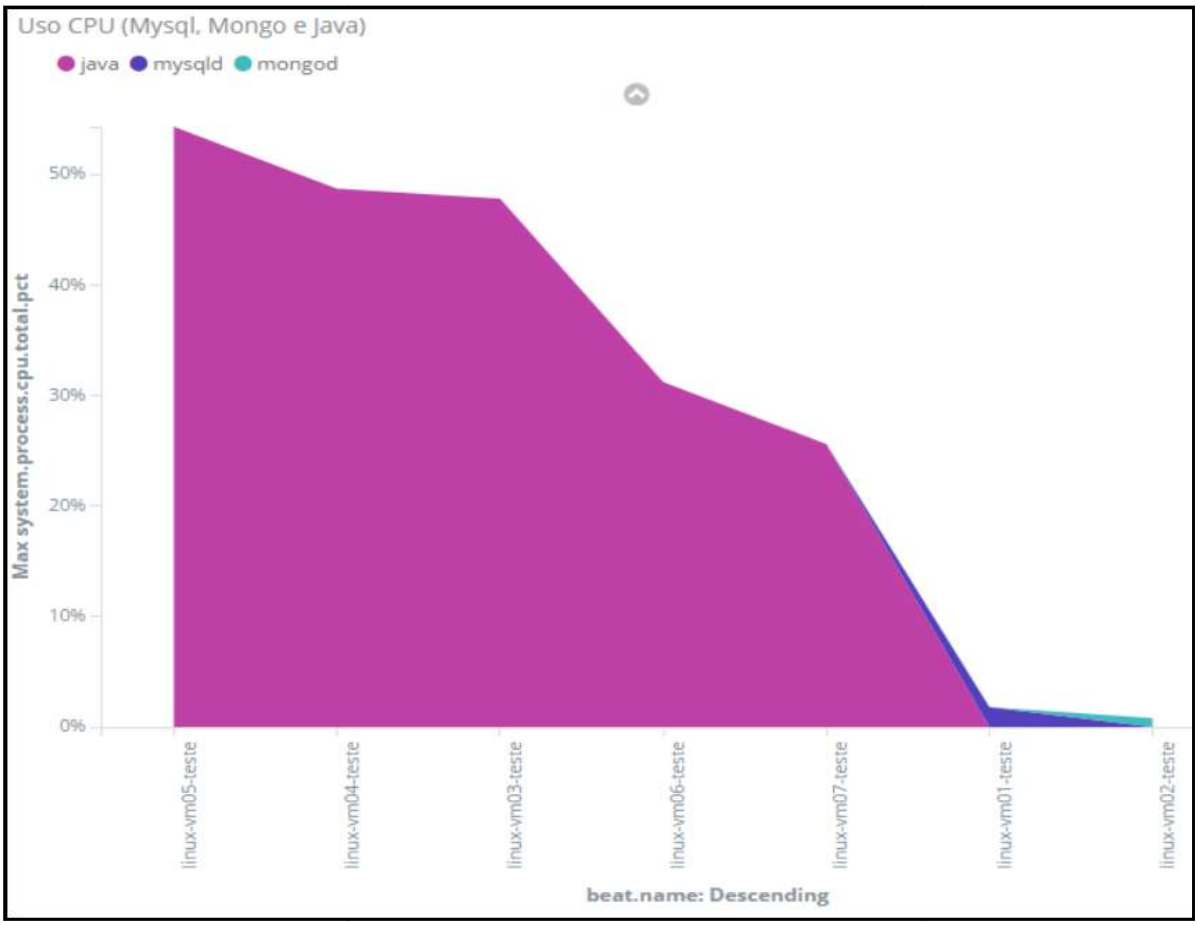
System overview by host - IOTDAS

Host	Used memory	Used Memory	Used disk space	System load / cores	CPU usage	IO Read Bytes	IO Write Bytes
linux-vm07-teste	53.568%	4.176GB	2.679GB	0.082	2.458%	120.7MB	199.125MB
linux-vm06-teste	24.757%	1.93GB	3.054GB	0.007	1.185%	116.398MB	506.98MB
linux-vm05-teste	22.61%	1.762GB	3.052GB	0.027	2.753%	69.679MB	160.897MB
linux-vm04-teste	25.021%	1.95GB	2.678GB	0.021	1.197%	120.382MB	161.928MB
linux-vm03-teste	34.331%	2.677GB	3.054GB	0.03	2.919%	349.557MB	383.338MB
linux-vm02-teste	10.118%	807.726MB	3.126GB	0	0.414%	100.689MB	265.182MB
linux-vm01-teste	19.736%	1.538GB	3.378GB	0	0.474%	207.823MB	566.812MB

Fonte: Elaborada pelos autores (2017).

Na Figura 31, temos um gráfico mais detalhado sobre o processamento da CPU dos servidores, onde pode-se verificar que a grande parte do processamento é realizado pelo Java, tendo um maior consumos nos servidores servidores que estavam rodando o microsserviço de canal, pois ele recebe as requisições, consome o microsserviço de usuário que acessa o banco de dados não relacional para assim conseguir acessar o banco de dados não-relacional.

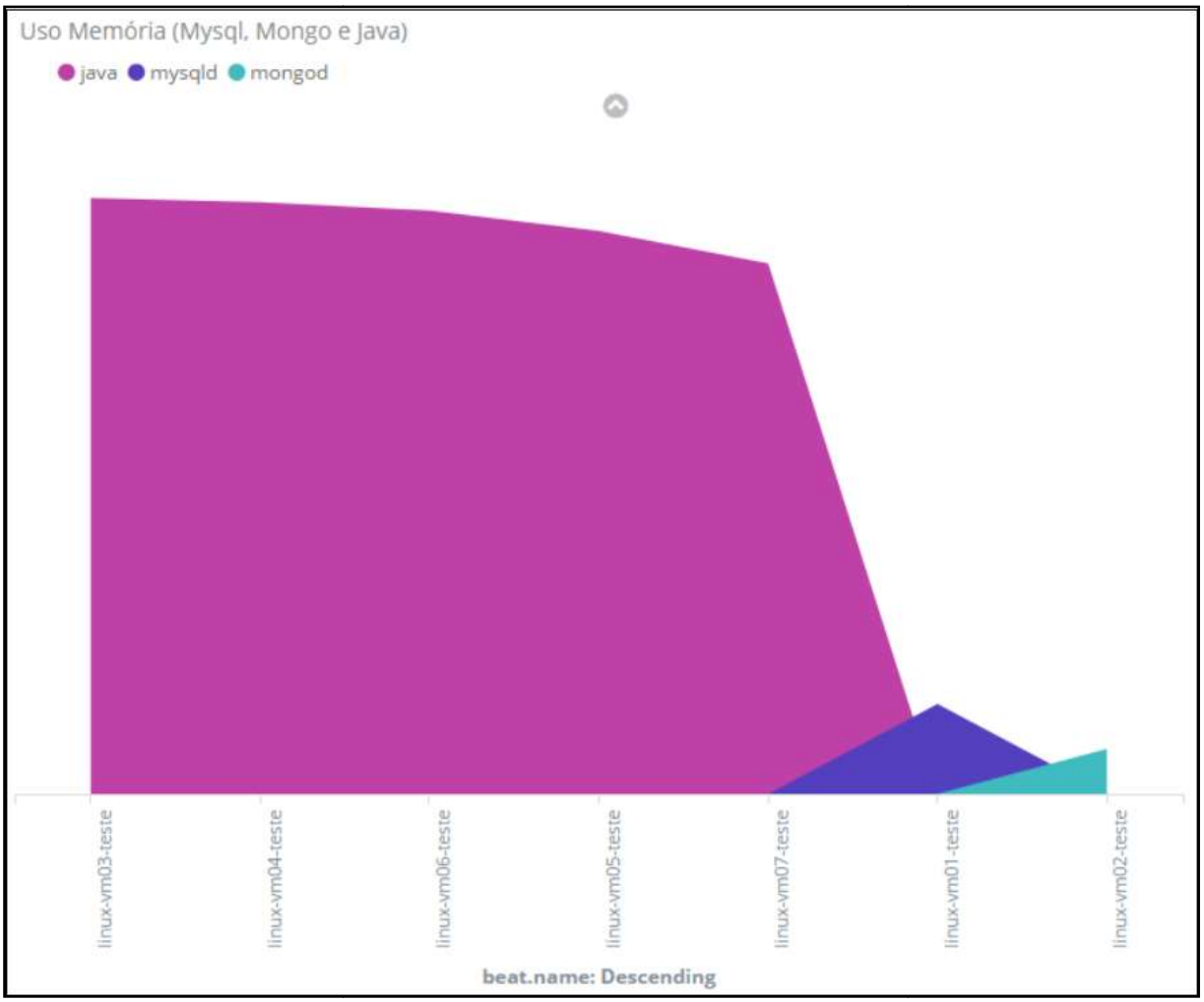
Figura 31 – Uso de CPU dos servidores



Fonte: Elaborada pelos autores (2017).

A utilização da memória dos servidores também é muito mais utilizada pelo Java, os microsserviços de usuário, que acessam o banco de dados não-relacional tem um consumo maior de memória, pois recebem todas as chamadas para validação do microsserviço de canal, conforme gráfico da Figura 32

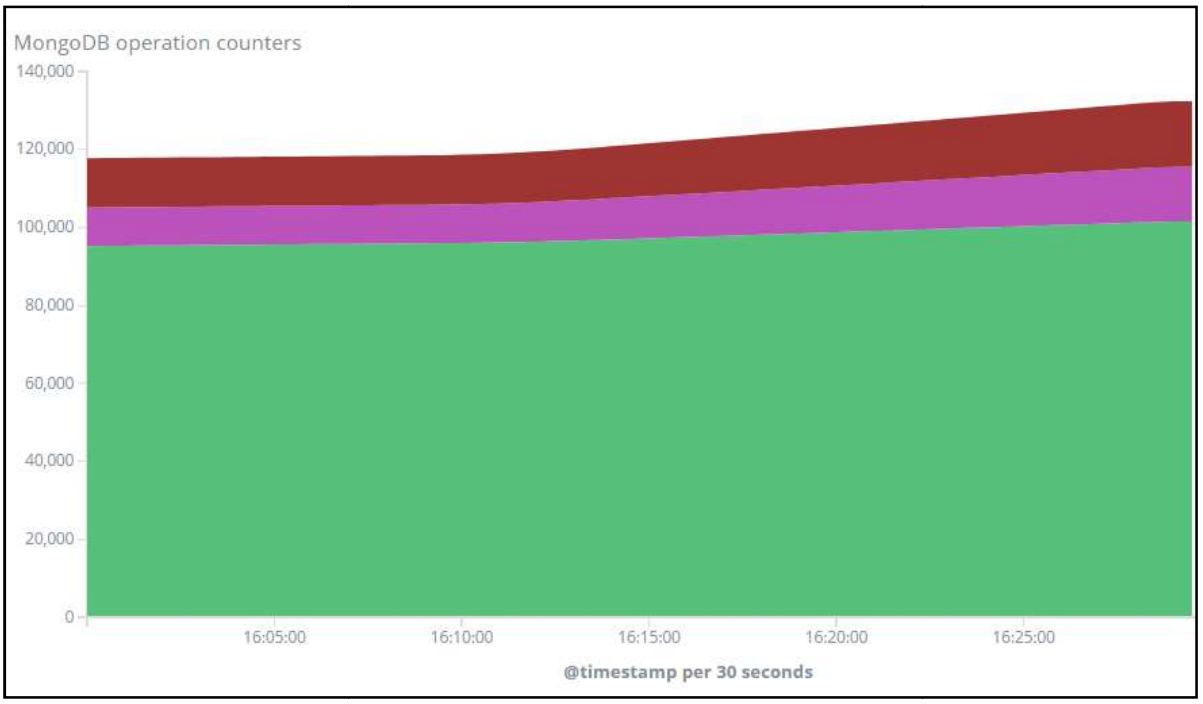
Figura 32 – Uso de memória dos servidores



Fonte: Elaborada pelos autores (2017).

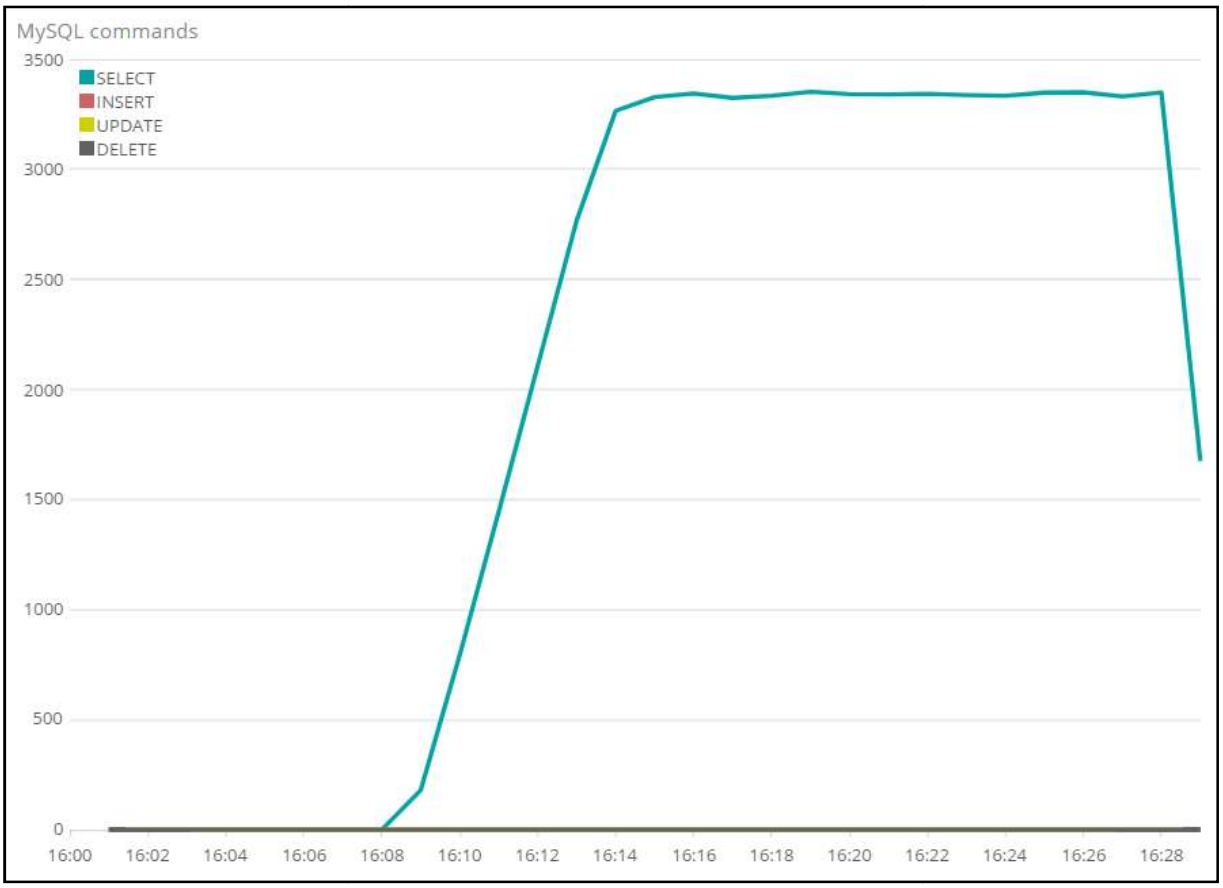
Os bancos de dados também foram monitorados durante o teste. A Figura 33 é um gráfico sobre as operações realizadas pelo MongoDB e a Figura 34 é um gráfico sobre as operações realizadas pelo MySQL.

Figura 33 – Operações geradas pelo MongoDB



Fonte: Elaborada pelos autores (2017).

Figura 34 – Operações geradas pelo MySQL



Fonte: Elaborada pelos autores (2017).

Gerando estas métricas ficou claro que os microsserviços que acessam o banco de dados não-relacional usam muito mais CPU, enquanto os microsserviços que acessam o banco de dados relacional consomem mais memória, desta forma podem ser provisionadas diferentes instâncias de servidores para cada propósito. Também ficou claro que conseguimos responder uma grande quantidade de requisições simultâneas com uma média de resposta de 20 milissegundos mesmo utilizando a metade dos recursos de servidor provisionados.

5 CONSIDERAÇÕES FINAIS

O volume de dados e informações gerados por dispositivos conectados à internet está cada vez maior. Existem dados sendo gerados, por exemplo, na agricultura, nas residências, nas empresas, desde as pequenas cidades, aos grandes centros. Um dos desafios na adesão à tecnologia envolvendo a Internet das Coisas é, justamente, o armazenamento deste volume de dados tão crescente.

A abordagem deste trabalho teve o foco em criar uma aplicação com capacidade de ser escalada de forma a atender a demanda de crescimento do armazenamento e disponibilização das informações que os dispositivos IoT estão gerando diariamente. Este objetivo foi alcançado utilizando a arquitetura de microsserviços e conjunto com o *Spring Cloud* e o *stack open source* da Netflix.

O desenvolvimento desse projeto foi um desafio para todos, pois as tecnologias utilizadas são relativamente recentes, muitas delas em constantemente aprimoramento, e recentemente tomadas como boa prática para atender os novos modelos de negócio que surgiram com advento da internet e dos dispositivos IoT.

Primeiramente tivemos que nos aprofundar no estudo destas tecnologias, para, conseguir entender todas as suas funcionalidades, dificuldades de implementação e limitações na utilização. E ainda assim, em muitas vezes, chegamos a algum ponto que tivemos que retroceder e adotar outra tecnologia.

Neste trabalho foi utilizado uma série de recursos para implementação. Um deles foi o *framework Spring*. Que facilitou muito a construção da aplicação, a implementação ficou muito prática, principalmente pela ampla diversidade de recursos que disponibiliza. A utilização do *Spring* foi fundamental para conseguir construir uma aplicação em forma de serviços independentes.

Além dos recursos do *Spring*, foram utilizadas também as ferramentas disponibilizadas pelo Netflix OSS que estão dentro do *Spring Cloud*. Ele que nos disponibilizou os recursos e facilidades para a construção de aplicação baseada em uma arquitetura de microsserviços. Utilizando as duas em conjunto, o *Spring* e as ferramentas do Netflix OSS, conseguimos criar uma aplicação bem estruturada e que segue padrões de resiliência, escalabilidade, balanceamento de carga, segurança entre outras características que uma aplicação deste tipo precisa.

No que diz respeito à resiliência de uma aplicação, o mau funcionamento de um determinado serviço não atinge o sistema como um todo, no efeito cascata, o que possibilita a maior concentração dos esforços na resolução dos problemas.

Também concluímos que os benefícios de uma aplicação desenvolvida com microsserviços são vários, que vão desde a facilidade na manutenção, aproveitamento de código, praticidade em implementar novas funcionalidades, a criação de um sistema distribuído e descentralizado que pode ser totalmente voltado para nuvem, que o uso de vários serviços em conjunto pode-se criar um sistema totalmente escalável e de alta disponibilidade. Esse tipo de arquitetura foi o ponto fundamental para conseguir entregar um sistema capaz de atender todos nossos objetivos.

Na parte do desenvolvimento da aplicação, uma das dificuldades no desenvolvimento do projeto foi a utilização do *framework Spring Cloud* e dos recursos do Netflix OSS que ele disponibiliza. Por tratar-se de uma tecnologia nova, as referências e documentos são muito escassos, e por estar em constante desenvolvimento, a curva de aprendizado foi muito grande, pois no decorrer do desenvolvimento do projeto houve algumas alterações de como elas operam em conjunto. Outra dificuldade foi em relação aos defeitos e problemas encontrados quando estas tecnologias são utilizadas em conjunto. Muitos destas possuem *bugs* de desenvolvimento que ainda não foram solucionadas.

Outro ponto relevante que pode ser citado foi a utilização da estrutura lógica de armazenamento de dados da aplicação. Com o propósito de tornar o projeto robusto, utilizou-se dois bancos de dados, um relacional e outro não-relacional, o que gerou a possibilidade para que o sistema seja escalado de forma a atender uma demanda em constante crescimento. Isso também permitiu que ele seja executado na nuvem de forma bem distribuída e descentralizada podendo atender regiões diferentes cada qual com suas instâncias de serviços e bancos de dados.

Com isso, apesar das dificuldades encontradas na busca por conhecimento nas tecnologias utilizadas, conclui-se que estas têm um papel importante na construção e no desenvolvimento de um protótipo de *software* como serviços que objetiva o armazenamento de dados para dispositivos IoT.

O protótipo de *software* foi inteiramente desenvolvido com base nos objetivos estipulados no escopo inicial do projeto. Isso nos dá margens para futuras implementações e até melhorias que podem ser desenvolvidas em cima da

aplicação já existente. Existem várias funcionalidades que podem ser acrescentadas, como por exemplo, um painel para visualização em gráficos das métricas de utilização de armazenamento e acesso aos dados, uma análise dos tipos de informação que estão sendo armazenadas, estruturas de decisão e gatilhos para quando determinado tipo de informação é inserida, tal como disparar um e-mail quando determinado campo recebe algum tipo de valor. Todas estas funcionalidades podem usufruir das facilidades que a arquitetura baseada em microsserviços possibilita para que qualquer implementação de funcionalidades seja feita sem afetar as demais.

Este trabalho foi muito importante para o nosso conhecimento e aprofundamento na prática de desenvolvimento e análise de *software,* pois nos permitiu conhecer uma diversidade de tecnologias para desenvolvimento de software, muitas deles ainda em ascensão, além de ter permitido colocar em prática técnicas de análise de software que conhecemos no decorrer do curso e assim nos preparar melhor para um mercado de trabalho tão exigente por profissionais qualificados.

REFERÊNCIAS

ALEX, Ben et al. **Spring Security Reference**. 2015. Disponível em: <http://docs.spring.io/spring-security/site/docs/5.0.0.M1/reference/htmlsingle/#what-is-acegi-security>. Acesso em: 26 mar. 2017.

BOICEA, Alexandru; RADULESCU, Florin; AGAPIN, Laura Ioana. MongoDB vs Oracle-Database Comparison. In: **EIDWT**. 2012. p. 330-335.

CARLSON, Josiah L. **Redis in Action**. Manning Publications Co., 2013.

CARNELL, John. **Spring Microservices in Action**. 8. ed. s. l.: Manning. 2017.

CHEE, Brian J. S.; FRANKLIN JR., Curtis. **Computação em Nuvem:** Cloud Computing, Tecnologias e Estratégias. 1. ed. São Paulo: M.Books do Brasil Editora Ltda, 2013

COULOURIS, George et al. **Sistemas Distribuídos**: Conceitos e projetos. 5 ed. São Paulo: Bookman editora Ltda, 2013.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de Banco de Dados**. 4. ed. São Paulo: Pearson Addison Wesley, 2005.

FERREIRA, António Miguel. **Introdução ao Cloud Computing:** IaaS, Paas, SaaS, tecnologia, conceito, e modelos de negócio. 1. ed. Lisboa, Portugal: FCA - Editora de Informática, 2015.

FOWLER, Martin; LEWIS, James. **Microservices**: a definition of this new architectural term. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 14 maio 2017.

GARTNER, inc. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. **Gartner**. Egham, U.K., fev. 2017. Disponível em <http://www.gartner.com/newsroom/id/3598917>. Acesso em: 04 mar. 2017.

GIL, Antonio Carlos. **Métodos e Técnicas de Pesquisa Social**. 6. ed. São Paulo: Editora Atlas S.A., 2008.

HAMAD, Hatem; SAAD, Motaz; ABED, Ramzi. **Performance Evaluation of RESTful Web Services for Mobile Devices**. Int. Arab J. e-Technol., v. 1, n. 3, p. 72-78, 2010.

KORTH, Henry F.; SILBERSCHATZ, Abraham; SUDARSHAN, S. **Sistemas de Banco de Dados**. 5. ed. Rio de Janeiro: Elsevier Editora Ltda, 2006.

MACEDO, Tiago; OLIVEIRA, Fred. **Redis cookbook**. O'Reilly Media, Inc., 2011.

MASSE, Mark. **REST API Design Rulebook:** Designing Consistent RESTful Web Service Interfaces. O'Reilly Media, Inc., 2011.

MASSRUHÁ, Silvia Maria Fonseca Silveira. Tecnologias da informação e da comunicação: O papel na agricultura. **AgroANALYSIS**, v. 35, n. 9, p. 29-31, 2015.

MELL, Peter; GRANCE, Tim. **The NIST definition of cloud computing**. Gaithersburg: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011.

MILANI, André. **MySQL:** Guia do programador. 1. ed. São Paulo: Novatec Editora, 2006.

MONTEIRO, Luís. A internet como meio de comunicação: possibilidades e limitações. In: **Congresso Brasileiro de Comunicação**. 2001.

MOREIRA, Pedro Felipe Marques; BEDER, Delano Medeiros. Desenvolvimento de Aplicações e Microsserviços: Um estudo de caso. **Revista T.I.S**. São Carlos, v.4, n.3, p. 209-2015, set./dez. 2015.

MOREIRA JR, Normandes José; AFONSO, Alexandre. **Produtividade no Desenvolvimento de Aplicações Web com Spring Boot**. 2. ed. Uberlândia: AlgaWorks Softwares, Treinamentos e Serviços Ltda., 2017.

NAYAK, Ameya; PORIYA, Anil; POOJARY, Dikshay. Type of NOSQL databases and its comparison with relational databases. **International Journal of Applied Information Systems**, v. 5, n. 4, p. 16-19, 2013.

NETFLIX, Inc. **Netflix Open Source Software Center.** 2016. Disponível em: <https://netflix.github.io/>. Acesso em: 13 maio. 2017.

NEWMAN, Sam. **Building Microservices: Designing fine-grained systems**. 1. ed. Sebastopol: O'Reilly Media, Inc, 2015.

OLIVEIRA, Samuel Silva de. Bancos de dados Não-Relacionais: um novo paradigma para armazenamento de dados em sistemas de ensino colaborativo. **Revista da Escola de Administração Pública do Amapá**, v. 2, n. 1, p. 184-194, 2014.

OPUS SOFTWARE. **O Que Você Realmente Precisa Saber Sobre Computação em Nuvem.** 1. ed. São Paulo: Opus Software Ltda., 2015.

PIVOTAL SOFTWARE, Inc. **Spring Cloud.** 2017. Disponível em: <http://projects.spring.io/spring-cloud/>. Acesso em: 30 abr. 2017.

RODRIGUEZ, Alex. **Restful web services:** The basics. IBM developerWorks, 2008.

SOUSA, Flávio RC; MOREIRA, Leonardo O.; MACHADO, Javam C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. **II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)**, p. 150-175, 2009.

WEISER, Mark. The computer for the 21st century. **Scientific american**, v. 265, n. 3, p. 94-104, 1991.

GLOSSÁRIO

Ações CRUD – acrônimo das expressões: Create (Criação), Retrieve (Consulta), Update (Atualização) e Delete (Exclusão). Utiliza-se para definir as operações básicas realizadas em um banco de dados.

Cluster – termo em inglês que significa “aglomerar”. Define uma arquitetura de sistema capaz combinar vários serviços para trabalharem em conjunto ou denominar o grupo em si de serviços combinados.

Código-fonte – escrita, os símbolos utilizados para codificar a aplicação.

Framework – abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.

Front-end – é a interface de usuário. Parte da aplicação que interage diretamente com o usuário.

Gateway – espécie de “ponte de ligação” intermediária destinada geralmente para interligar a requisição aos serviços da aplicação.

GUID – ou UUID é um acrônimo para *Globally Unique Identifier* ou Identificador Universalmente Único. É um número inteiro de 128 bits usado para identificar recursos.

Host – computador ou máquina conectado a uma rede, que conta com número de IP e nome definidos. Essas máquinas são responsáveis por oferecer recursos, informações e serviços aos usuários ou clientes.

Resiliência – capacidade da aplicação de manter níveis aceitáveis de operação frente a uma diversidade de problemas que podem ocorrer.

Rotina – representação de trechos de código fonte utilizados na aplicação.

Round-Robin – um sistema que consiste em alguns elementos, cada um representado por um processo ou uma máquina de um *cluster*, formando uma fila circular. É uma técnica utilizada se trabalhar com problemas de distribuição de carga.

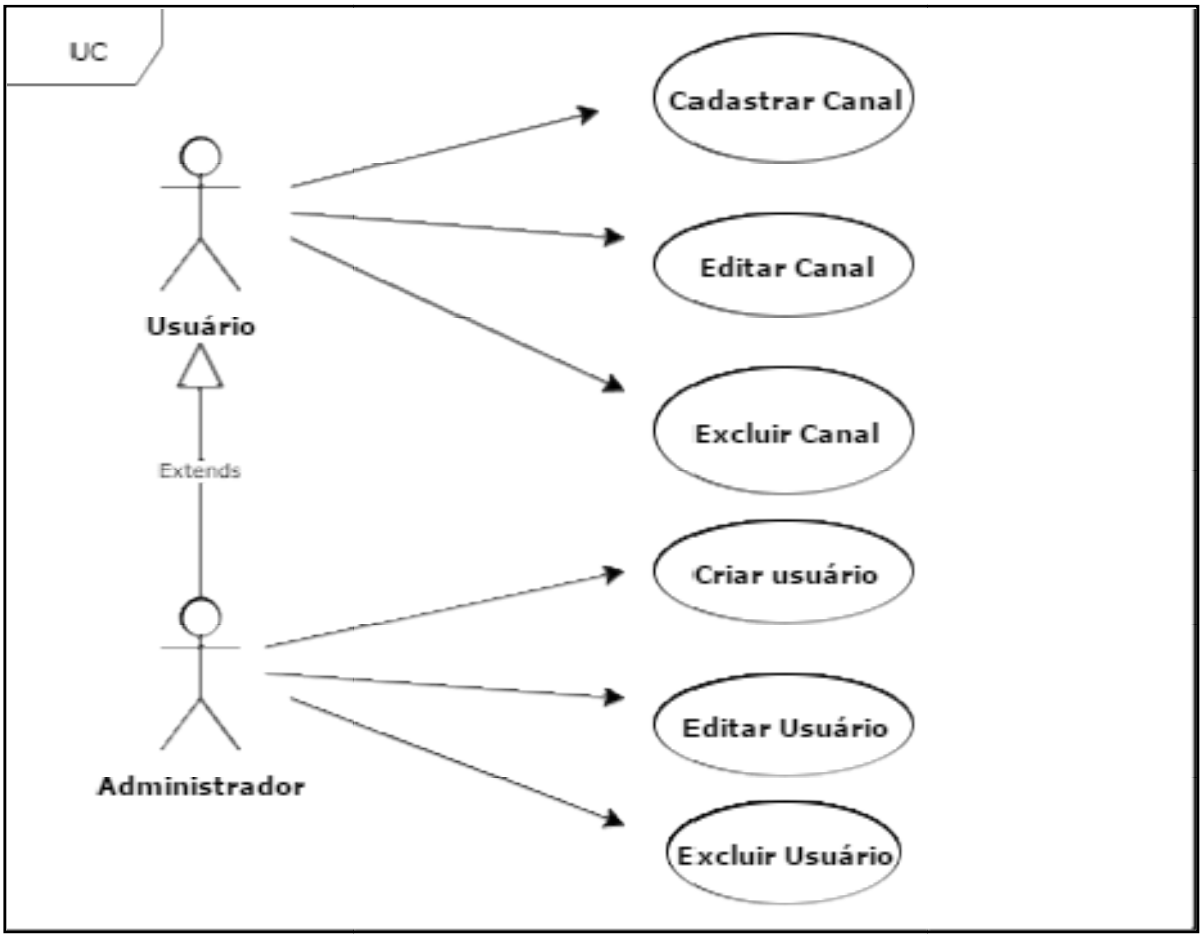
Serialização de JSON – processo de conversão de um objeto em um fluxo de *bytes* para armazenar o objeto ou fluxo na memória, em um banco de dados, ou em um arquivo. Sua finalidade principal é salvar o estado de um objeto para ser capaz de recriá-lo quando necessário. O processo inverso é chamado desserialização.

Token – chave, um código utilizado para segurança de acesso a aplicação.

Virtualização – simulação de uma plataforma de hardware, sistema operacional, dispositivo de armazenamento ou recursos de rede.

APÊNDICE A – ESPECIFICAÇÕES DE CASOS DE USO

Figura 1 – Ilustração dos casos de uso



UC-01 – Administrar aplicação.
Descrição: O sistema deve permitir o controle, criação, alteração e exclusão de registros de usuário.
Pré-condição: Usuário cadastrado no sistema.
Fluxo principal: 1. O usuário acessa o painel administrativo no sistema. 2. O sistema apresenta o painel de controle com menus de funcionalidades do sistema. 3. O administrador clica em usuários.

4. O sistema apresenta o menu editar, excluir ou novo usuário. 5. O usuário escolhe a opção que deseja e clica em salvar. 6. O Sistema salva modificações na aplicação.
Pós-condições: Alterações realizada com sucesso.

UC-02 – Administrar canal
Descrição: O sistema deve permitir que o usuário crie, altere e exclua o determinado canal para o armazenamento dos dados. O sistema deve permitir que o usuário adicione, remova e altere os campos dos canais para armazenamento. O sistema deve permitir que o usuário adicione, remova e altere os dados do canal.
Pré-condição: Usuário cadastrado no sistema.
Fluxo principal: 1. O usuário acessa o painel administrativo no sistema. 2. O sistema apresenta o painel de controle com menus de funcionalidades do sistema. 3. O usuário clica em canal. 4. O usuário clica em adicionar novo canal. 5. O usuário digita nome, descrição e o tipo de canal. 6. O usuário informa quais campos necessita e a chave key do canal. 7. O usuário clica em salvar. 8. O sistema emite uma mensagem de operação realizada com sucesso.
Fluxo de exceção: Editar canal. No passo 4 o usuário clica em Canal. 5. O usuário escolhe o canal que deseja editar e clica em editar canal. 6. O usuário realiza as alterações. O usuário retorna ao passo 7.
Fluxo de exceção: Excluir canal.

No passo 4 o usuário clica em Canal. 5. O usuário escolhe o canal que deseja excluir e clicar em excluir. 6. O sistema emite um alerta de canal excluido com sucesso.
Fluxo de exceção: Alterar campos. No passo 4 o usuário clica em Canal. 5. O usuário escolhe o campo que deseja editar, adicionar ou excluir. O usuário é direcionado ao passo 7.
Pós-condições: Operação realizada com sucesso.

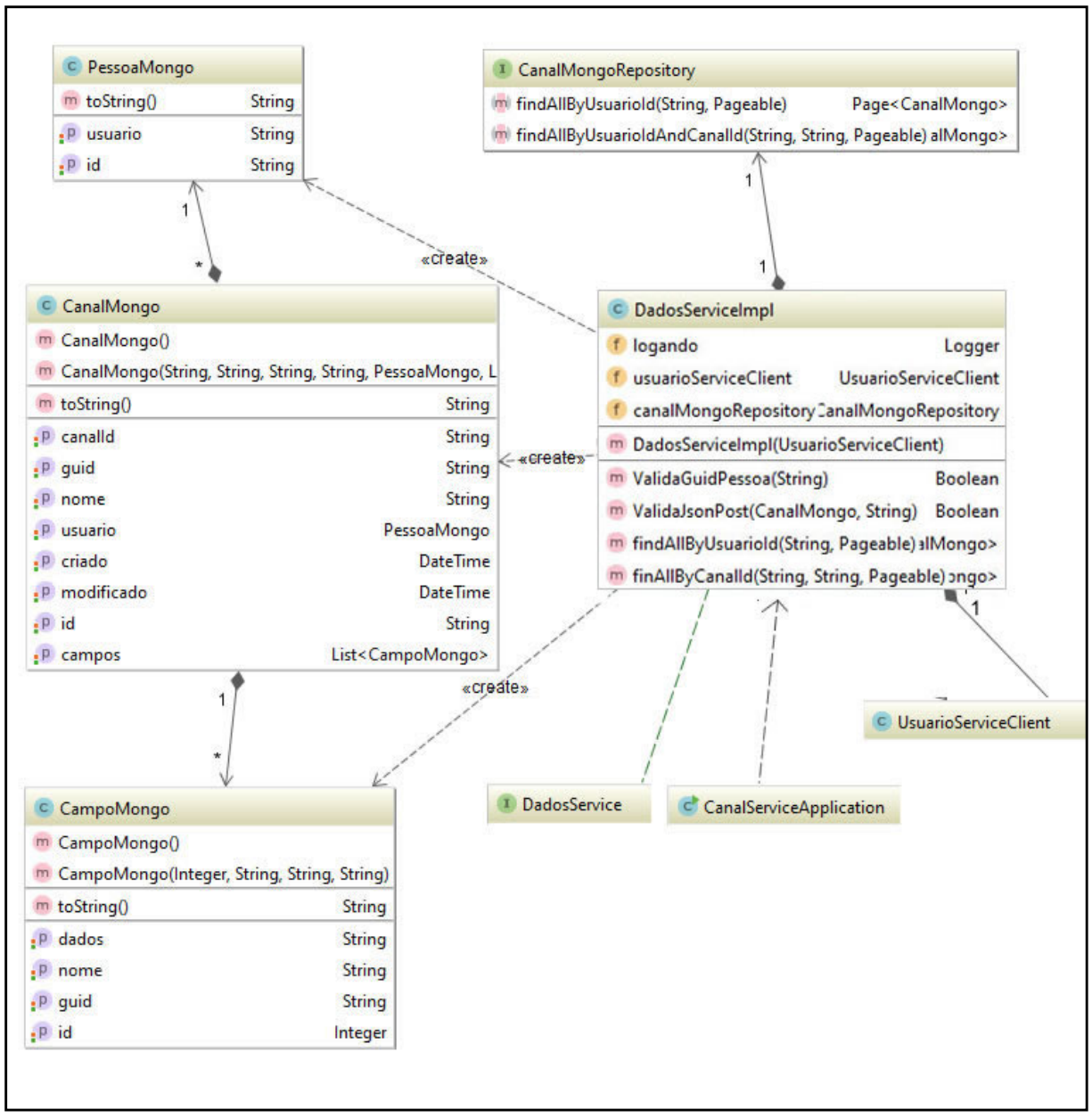
UC-03 – Comunicação do sistema
Descrição: O sistema deverá receber os dados do dispositivo IOT
Pré-condição: Dispositivo IoT conectado na Internet.
Fluxo principal: 1. O sistema conecta ao dispositivo através de uma chave única (Chave Key). 2. O sistema realiza solicitaçõe ao dispositivo. 3. O sistema exibe as informações do canal no painel administrativo.
Pós-condições: Operação realizada com sucesso.

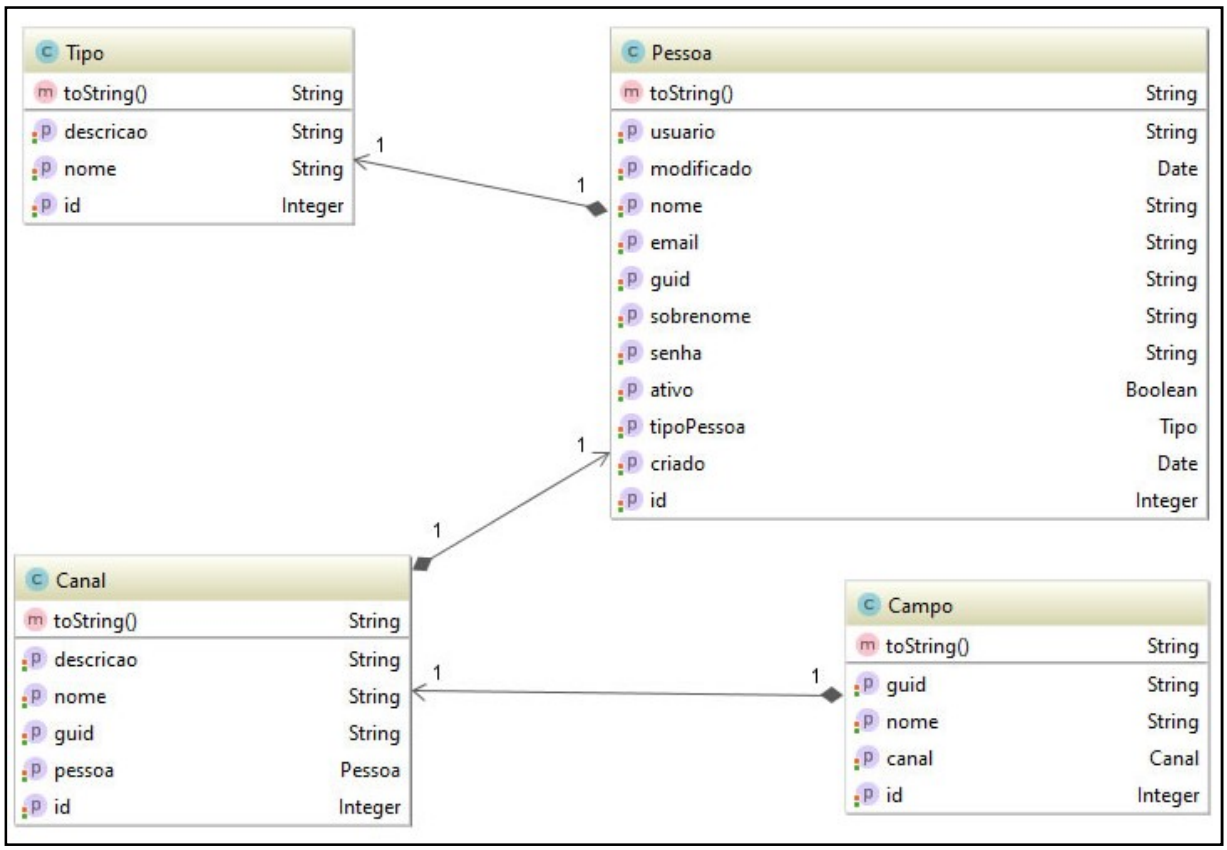
UC-04 – Dados do canal
Descrição: Cada canal inserido deve possuir características / atributos equivalentes a: descrição, quantidade de campos definidos pelo usuário, nome para identificação.
Pré-condição: Usuário cadastrado no sistema e na tela de canal aberta.

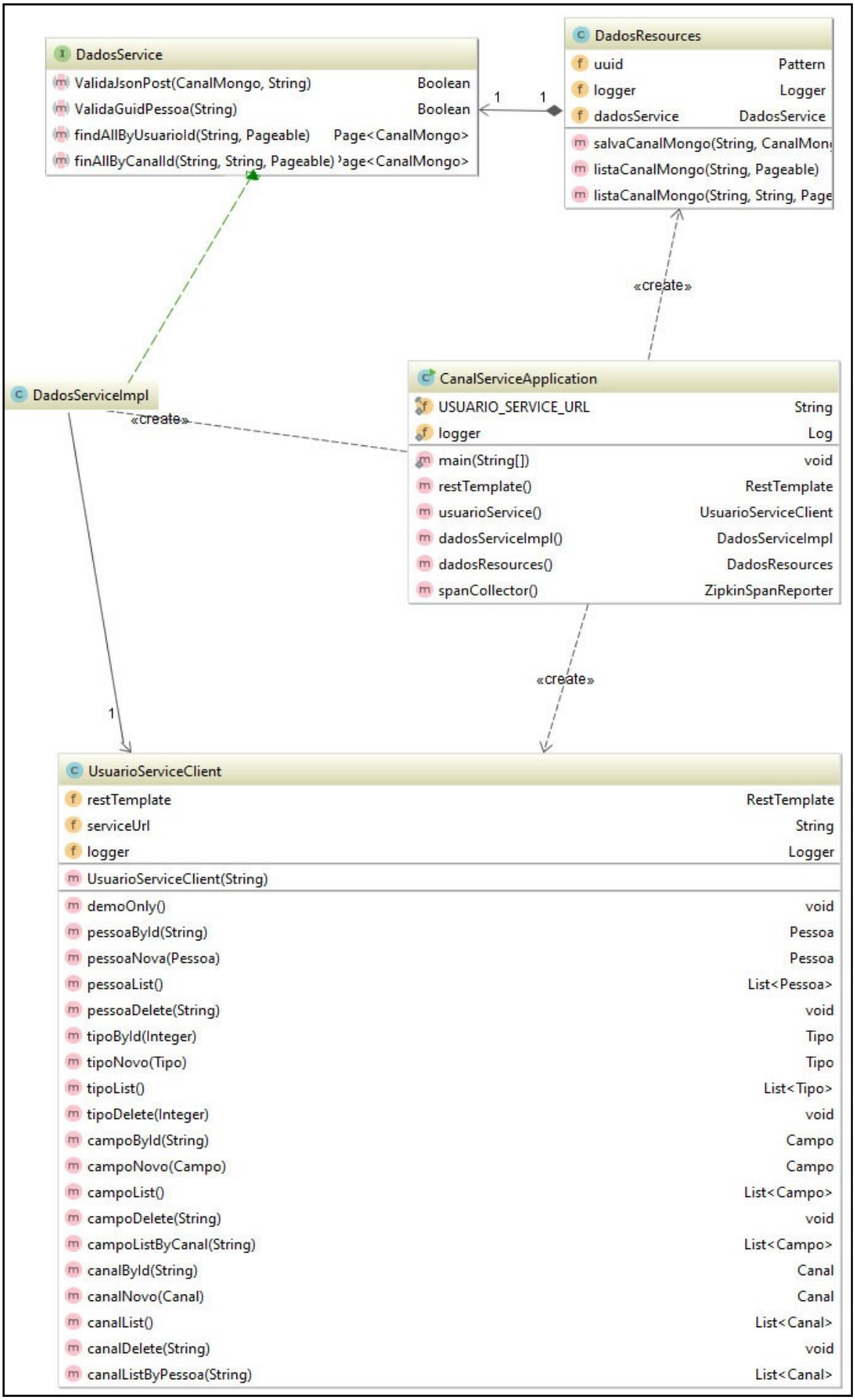
Fluxo principal: 1. O usuário digita nome, descrição e o tipo de canal. 2. O sistema possibilita somente os tipos de dados específicos para cada campo, onde for texto e valor numérico. 3. O usuário informa quais campos necessita e a chave key do canal. 4. O usuário clica em salvar. 5. O sistema emite uma mensagem de operação realizada com sucesso.
Fluxo de exceção: Campos obrigatórios. No passo 4, caso o usuário tenha esquecido de informar algum dado o sistema emite um alerta de campo obrigatório, retornando ao passo 3.

APÊNDICE B – DIAGRAMAS DE CLASSE

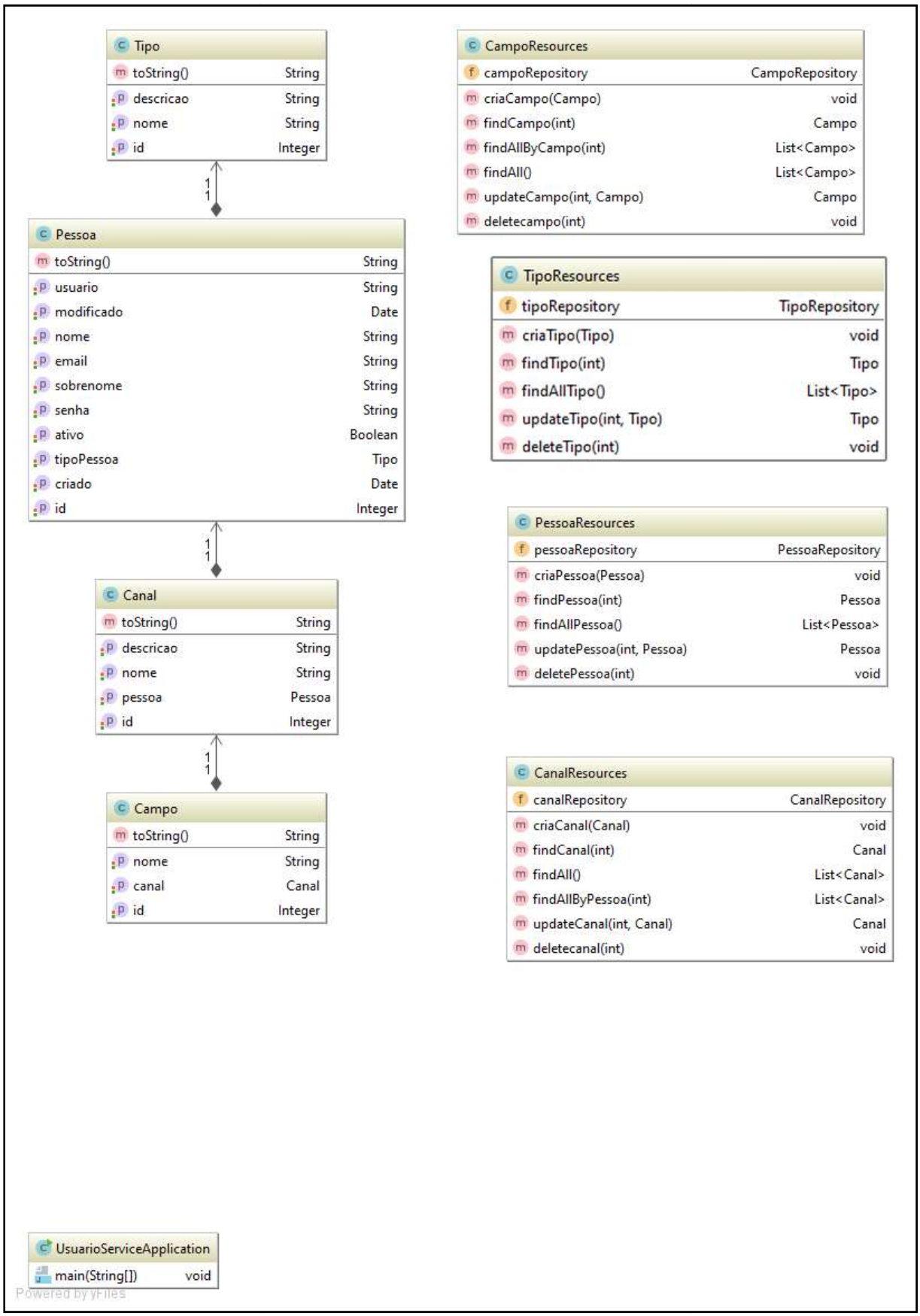
1 – Diagrama de classe do serviço de canal



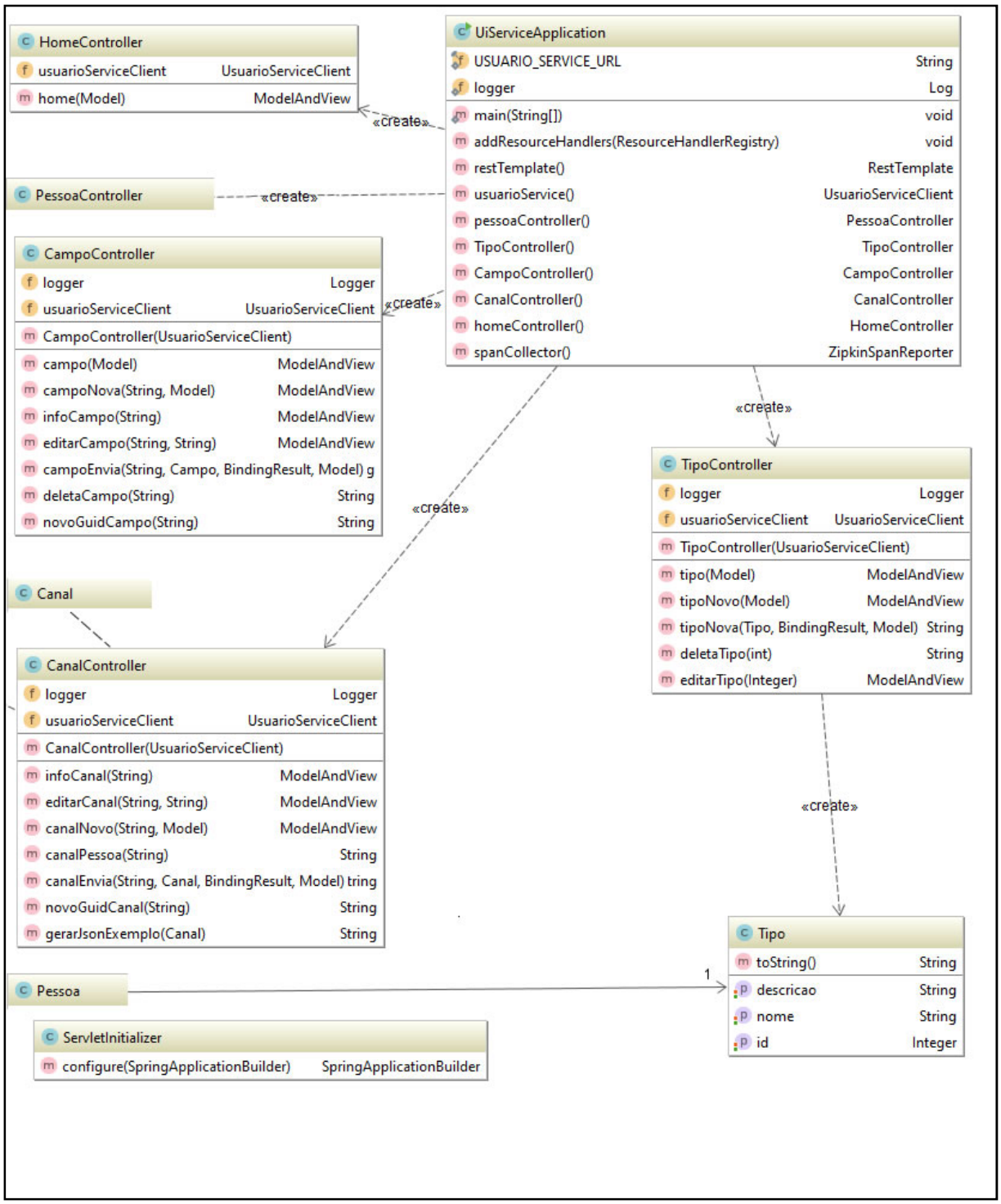

Tipo
toString() String
descricao String
nome String
id Integer
Pessoa
toString() String
usuario String
modificado Date
nome String
email String
guid String
sobrenome String
senha String
ativo Boolean
tipoPessoa Tipo
criado Date
id Integer
Canal
toString() String
descricao String
nome String
guid String
pessoa Pessoa
id Integer
Campo
toString() String
guid String
nome String
canal Canal
id Integer
1
1
1
1
1
1


DadosService
ValidaJsonPost(CanalMongo, String) Boolean
ValidaGuidPessoa(String) Boolean
findAllByUsuarioId(String, Pageable) Page<CanalMongo>
finAllByCanalId(String, String, Pageable) Page<CanalMongo>
DadosResources
uuid Pattern
logger Logger
dadosService DadosService
salvaCanalMongo(String, CanalMon
listaCanalMongo(String, Pageable)
listaCanalMongo(String, String, Page
1
1
«create»
DadosServiceImpl
«create»
CanalServiceApplication
USUARIO_SERVICE_URL String
logger Log
main(String[]) void
restTemplate() RestTemplate
usuarioService() UsuarioServiceClient
dadosServiceImpl() DadosServiceImpl
dadosResources() DadosResources
spanCollector() ZipkinSpanReporter
«create»
1
UsuarioServiceClient
restTemplate RestTemplate
serviceUrl String
logger Logger
UsuarioServiceClient(String)
demoOnly() void
pessoaById(String) Pessoa
pessoaNova(Pessoa) Pessoa
pessoaList() List<Pessoa>
pessoaDelete(String) void
tipoById(Integer) Tipo
tipoNovo(Tipo) Tipo
tipoList() List<Tipo>
tipoDelete(Integer) void
campoById(String) Campo
campoNovo(Campo) Campo
campoList() List<Campo>
campoDelete(String) void
campoListByCanal(String) List<Campo>
canalById(String) Canal
canalNovo(Canal) Canal
canalList() List<Canal>
canalDelete(String) void
canalListByPessoa(String) List<Canal>

2 - Serviço de Usuário



3 - Serviço de *interface* de usuário



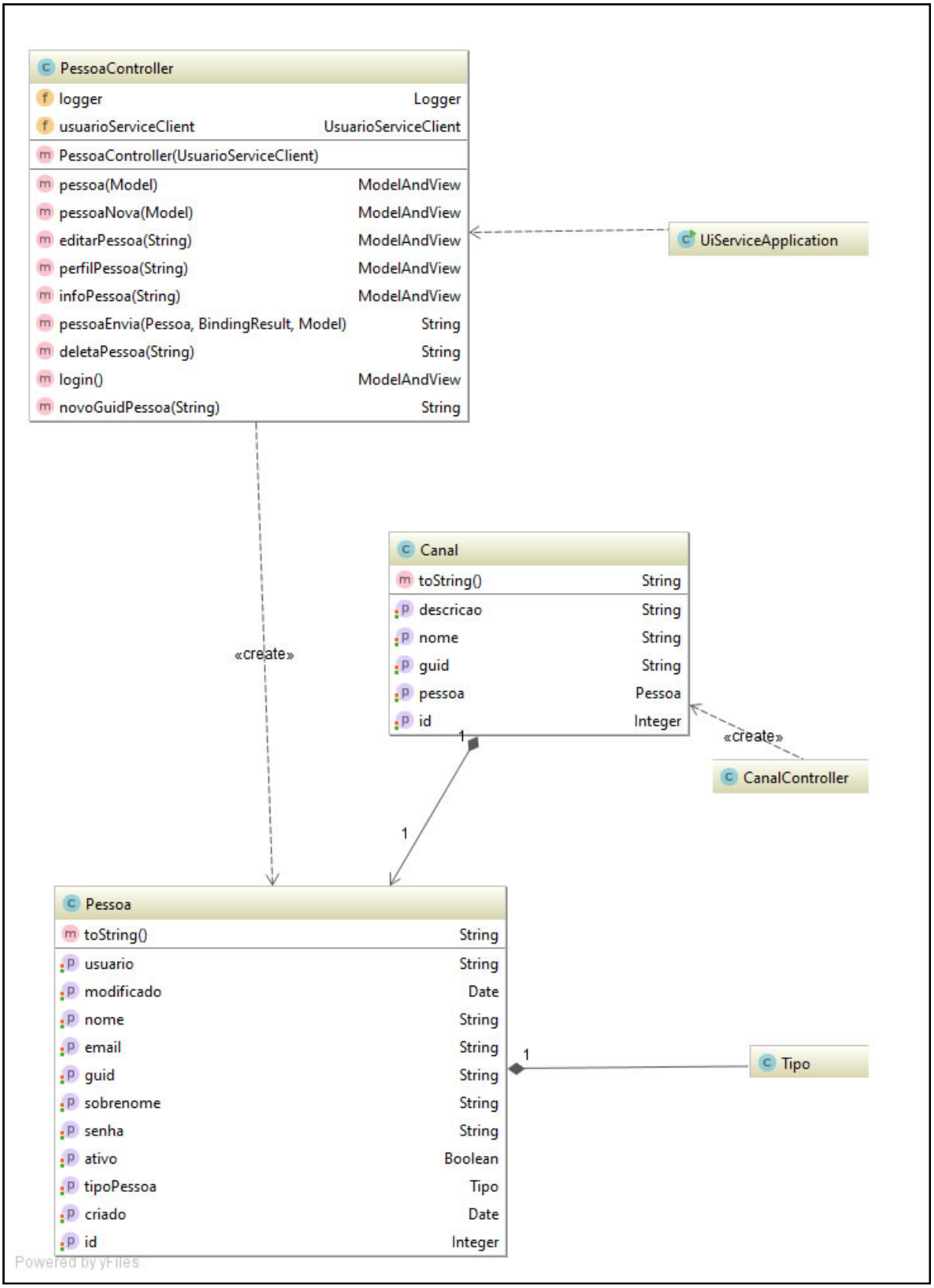
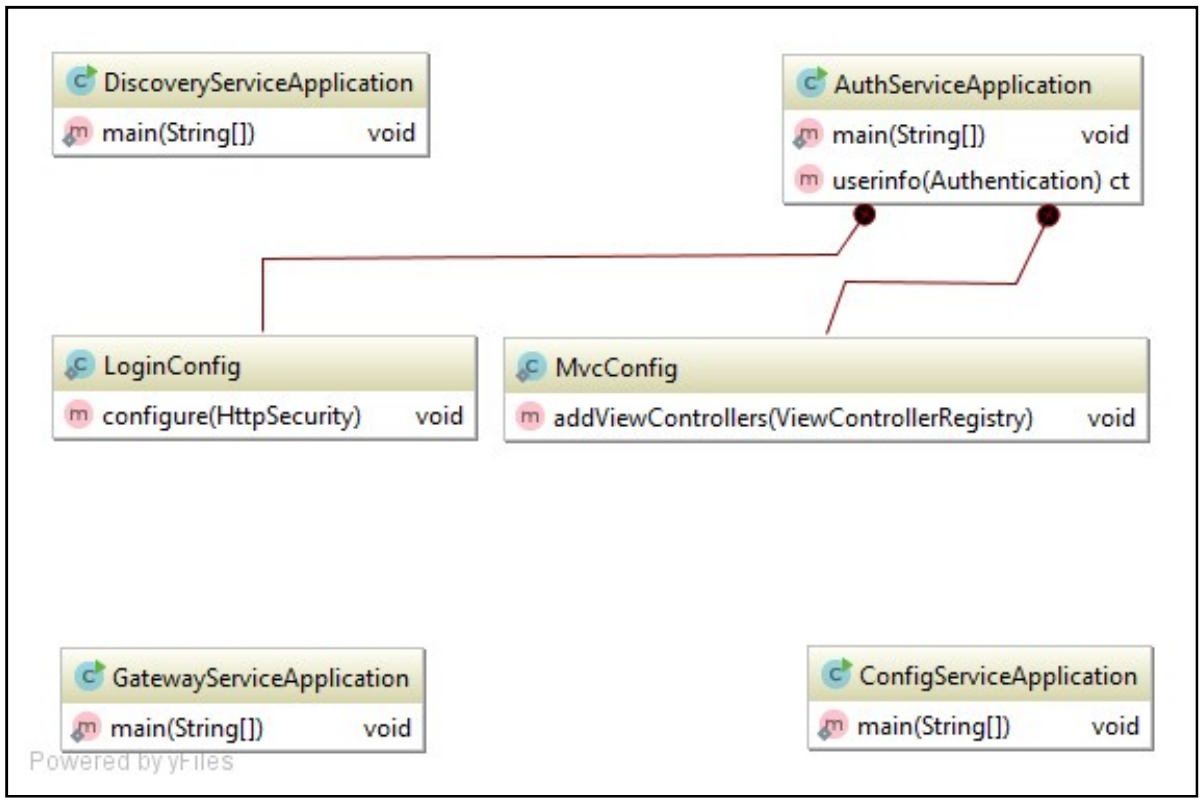

PessoaController
logger Logger
usuarioServiceClient UsuarioServiceClient
PessoaController(UsuarioServiceClient)
pessoa(Model) ModelAndView
pessoaNova(Model) ModelAndView
editarPessoa(String) ModelAndView
perfilPessoa(String) ModelAndView
infoPessoa(String) ModelAndView
pessoaEnvia(Pessoa, BindingResult, Model) String
deletaPessoa(String) String
login() ModelAndView
novoGuidPessoa(String) String
UiServiceApplication
«create»
Canal
toString() String
descricao String
nome String
guid String
pessoa Pessoa
id Integer
«create»
CanalController
1
1
Pessoa
toString() String
usuario String
modificado Date
nome String
email String
guid String
sobrenome String
senha String
ativo Boolean
tipoPessoa Tipo
criado Date
id Integer
1
Tipo
Powered by yFiles

Figura 4 - Serviço de autenticação



APÊNDICE C – PROTÓTIPOS DE TELAS DO SISTEMA

Figura 1 - Lista de usuários

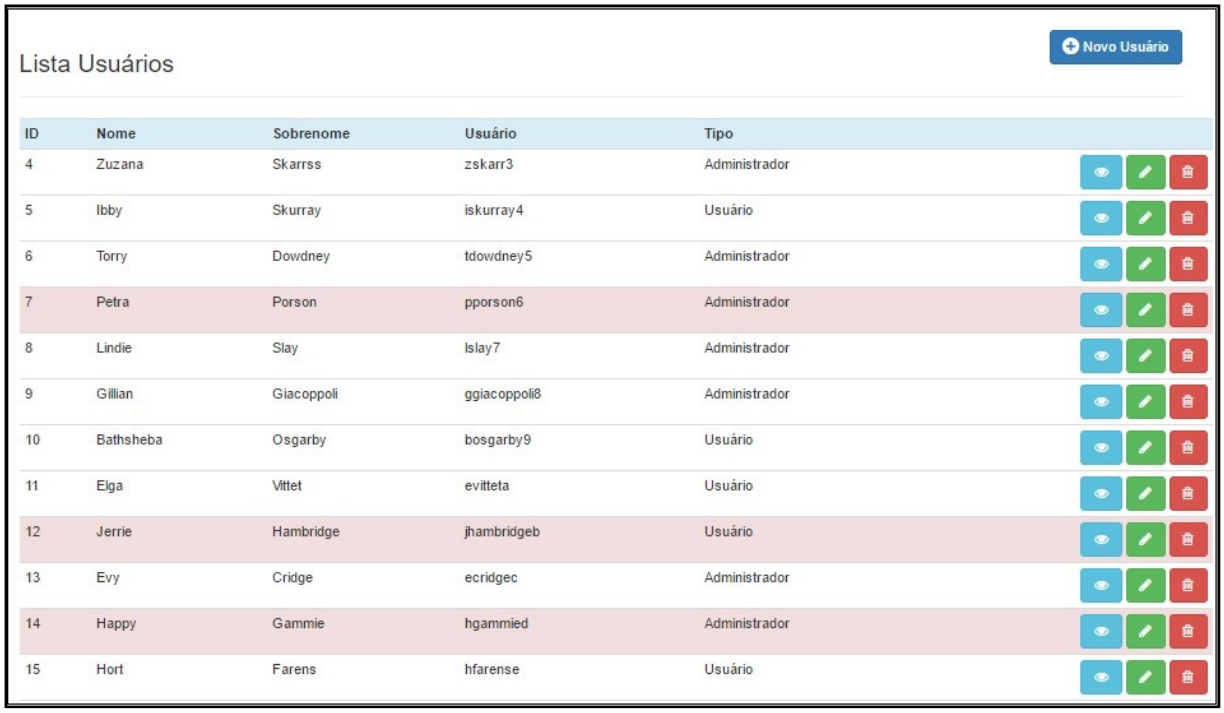


Figura 2 - Cadastrar novo usuário

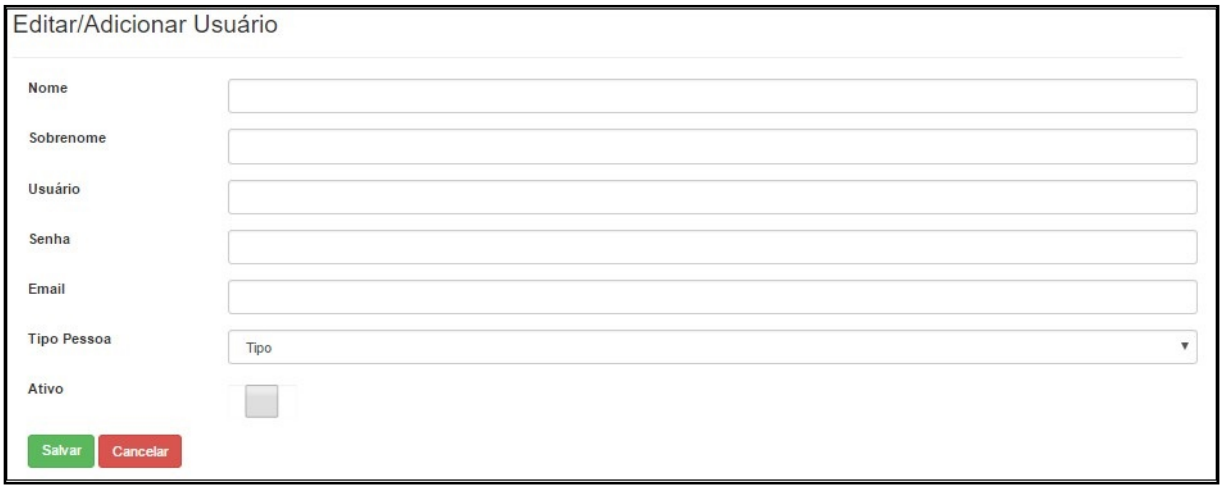


Figura 3 - Editar usuário

Figura 4 - Excluir usuário

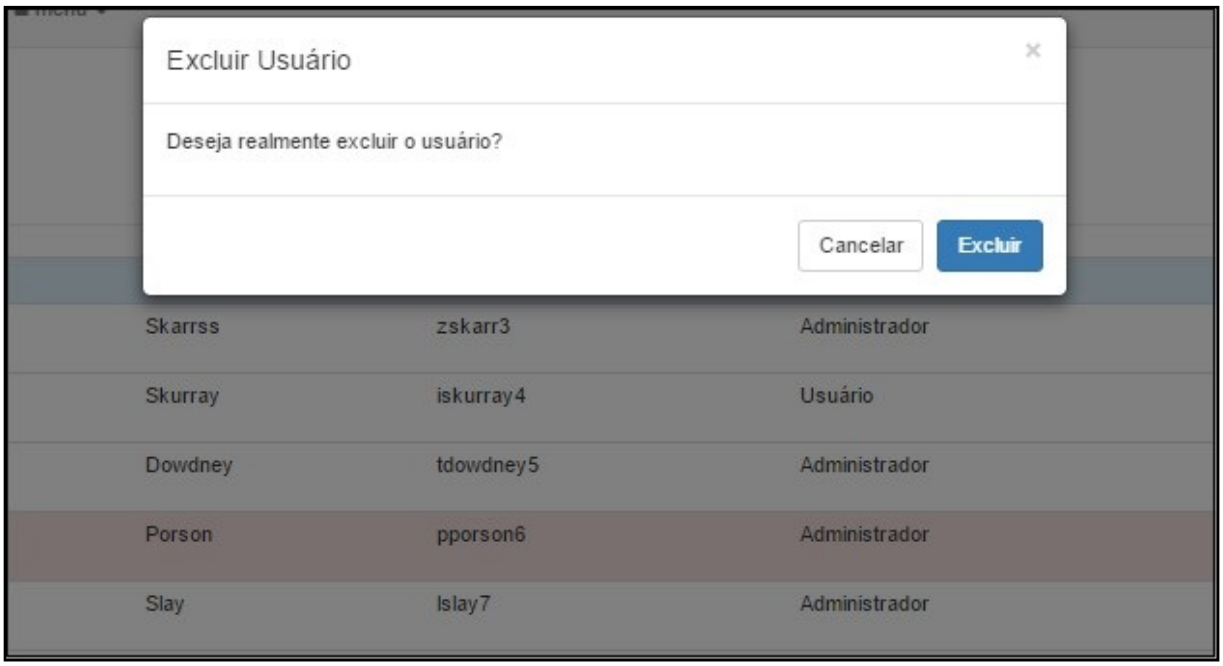


Figura 5 - Informações do usuário

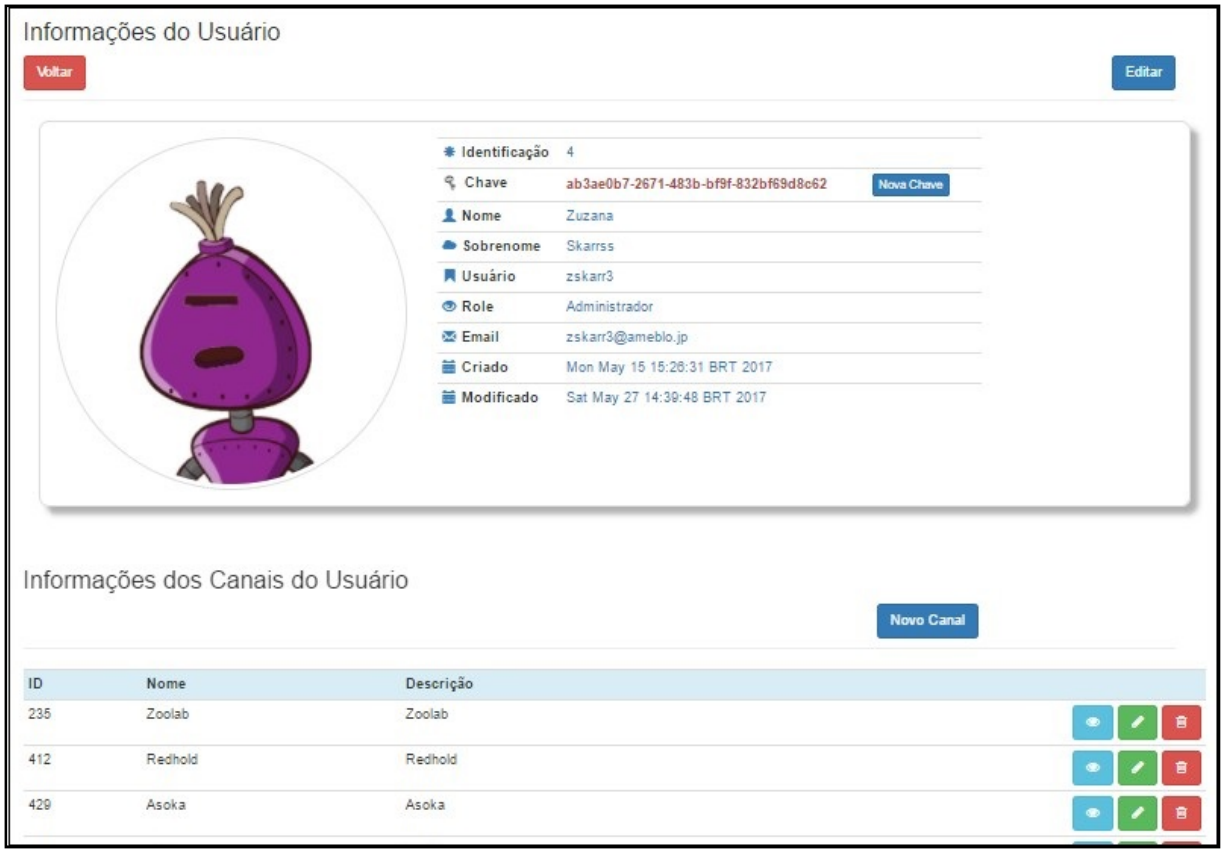


Figura 6 - Cadastrar novo canal

Figura 7 - Editar canal

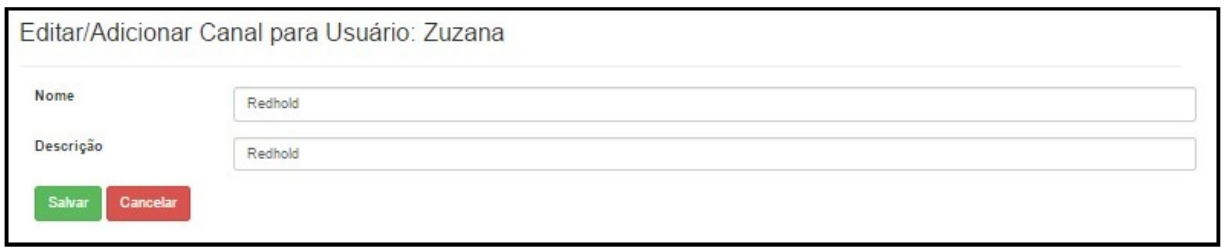


Figura 8 - Excluir canal

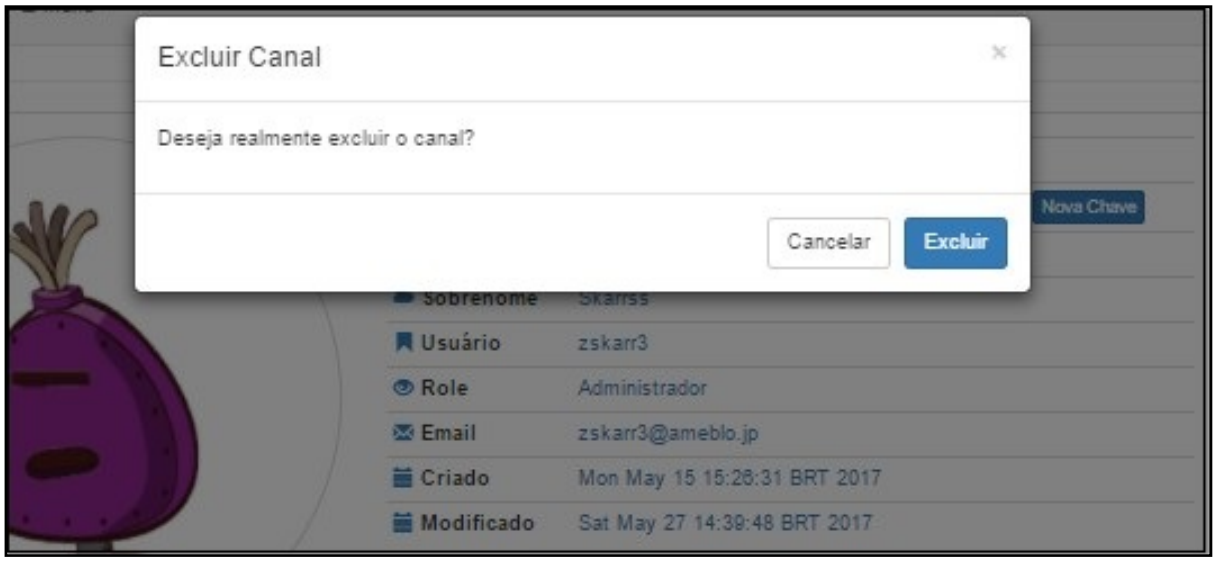


Figura 9 - Informações do canal

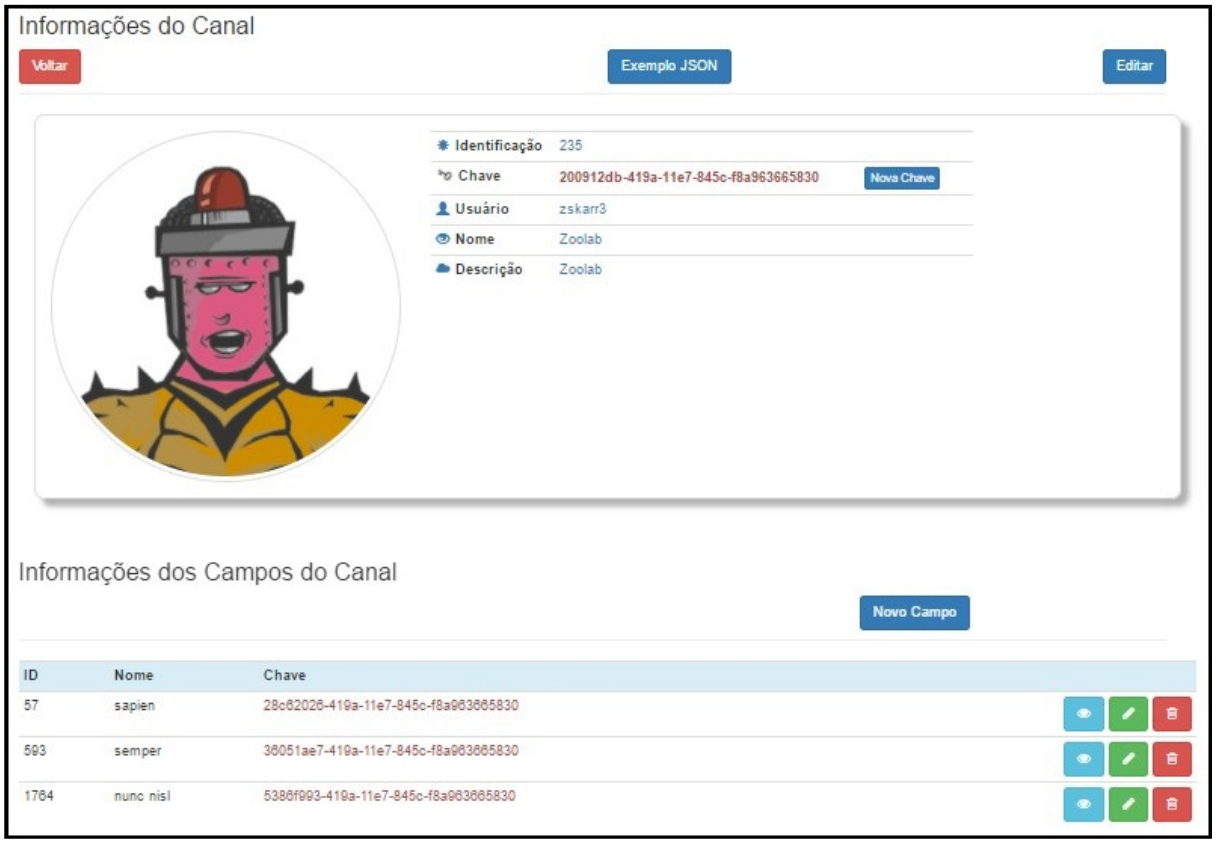


Figura 10 - Cadastrar novo campo

Figura 11 - Editar campo

Figura 12 - Excluir campo

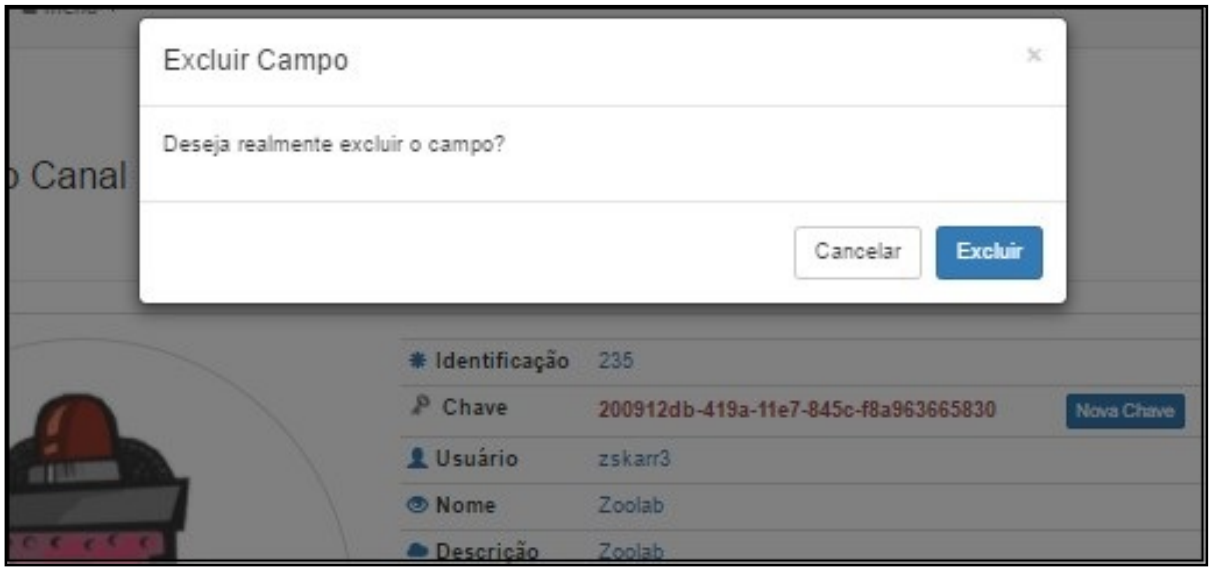


Figura 13 - Informações do campo

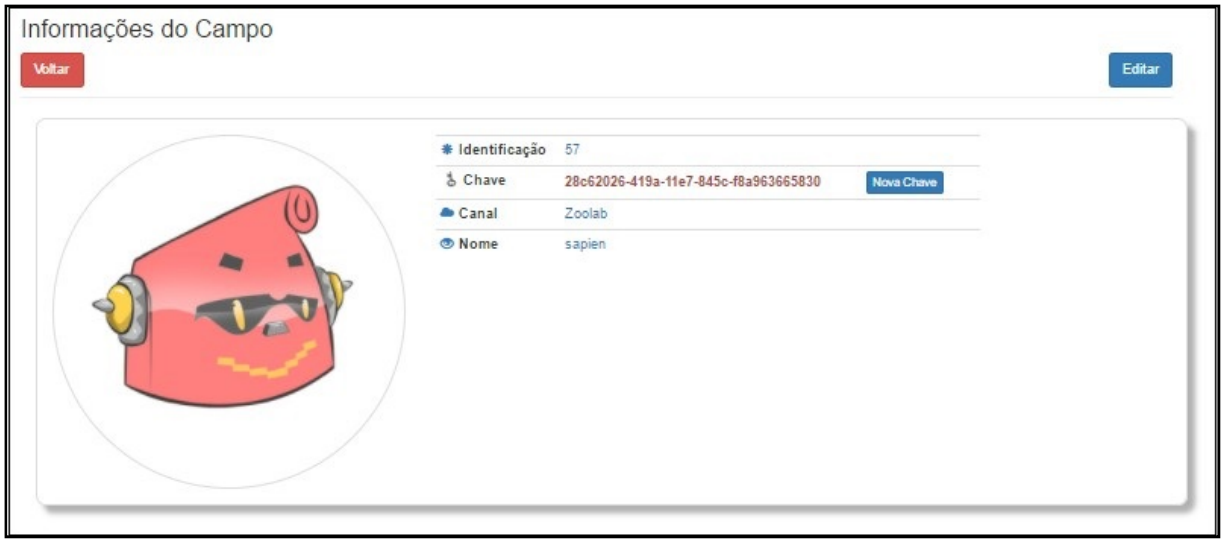


Figura 14 - Lista de tipos de usuário

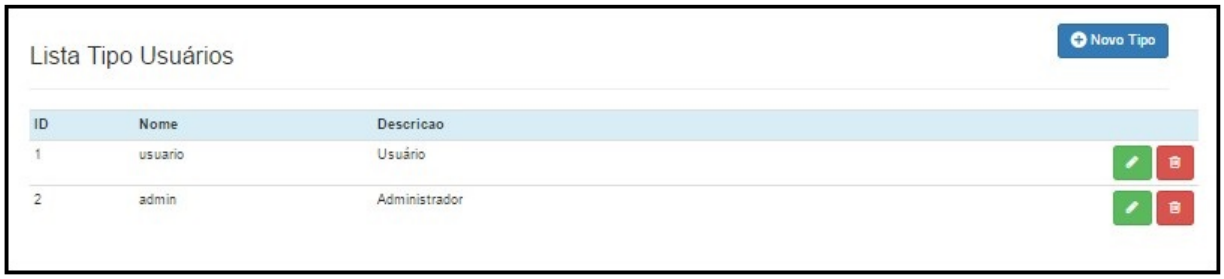


Figura 15 - Cadastrar novo tipo de usuário

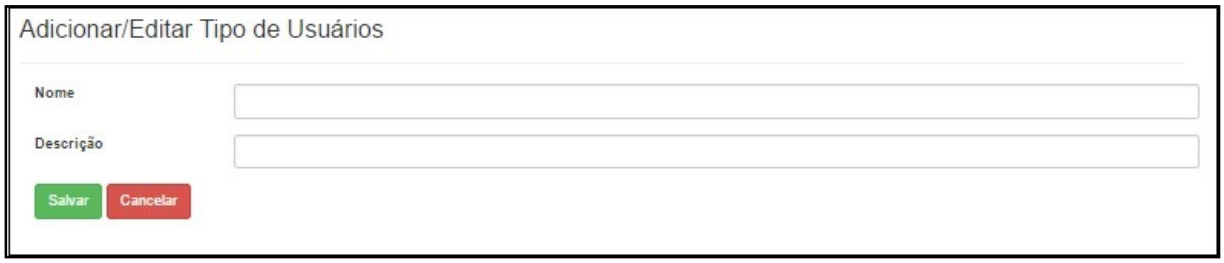


Figura 16 - Editar tipo de usuário

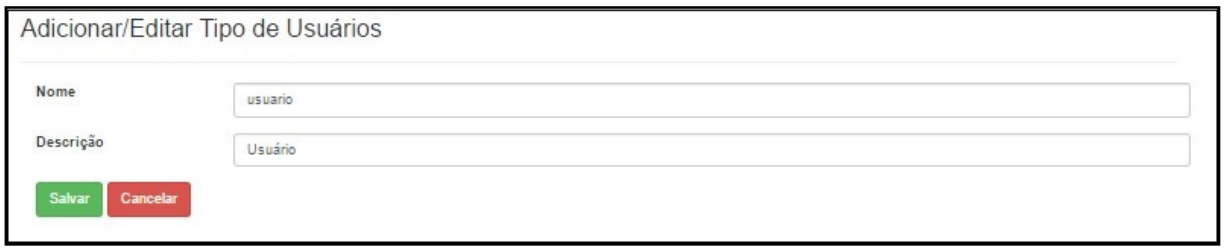
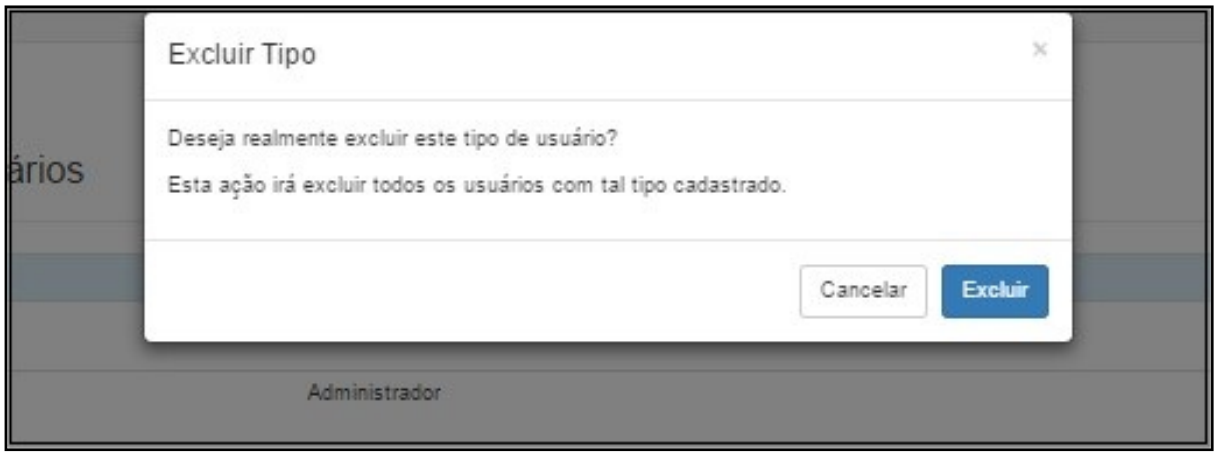
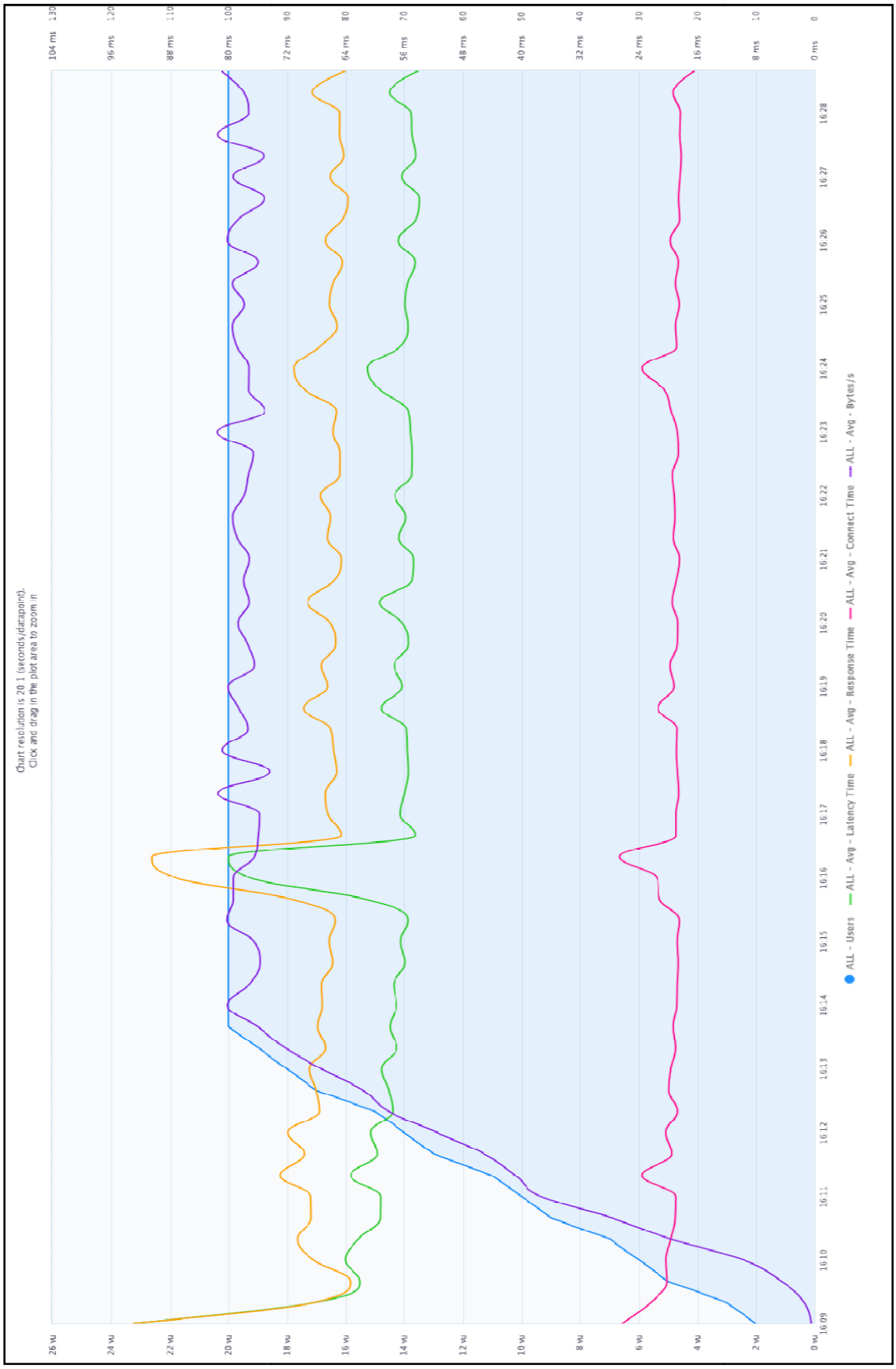


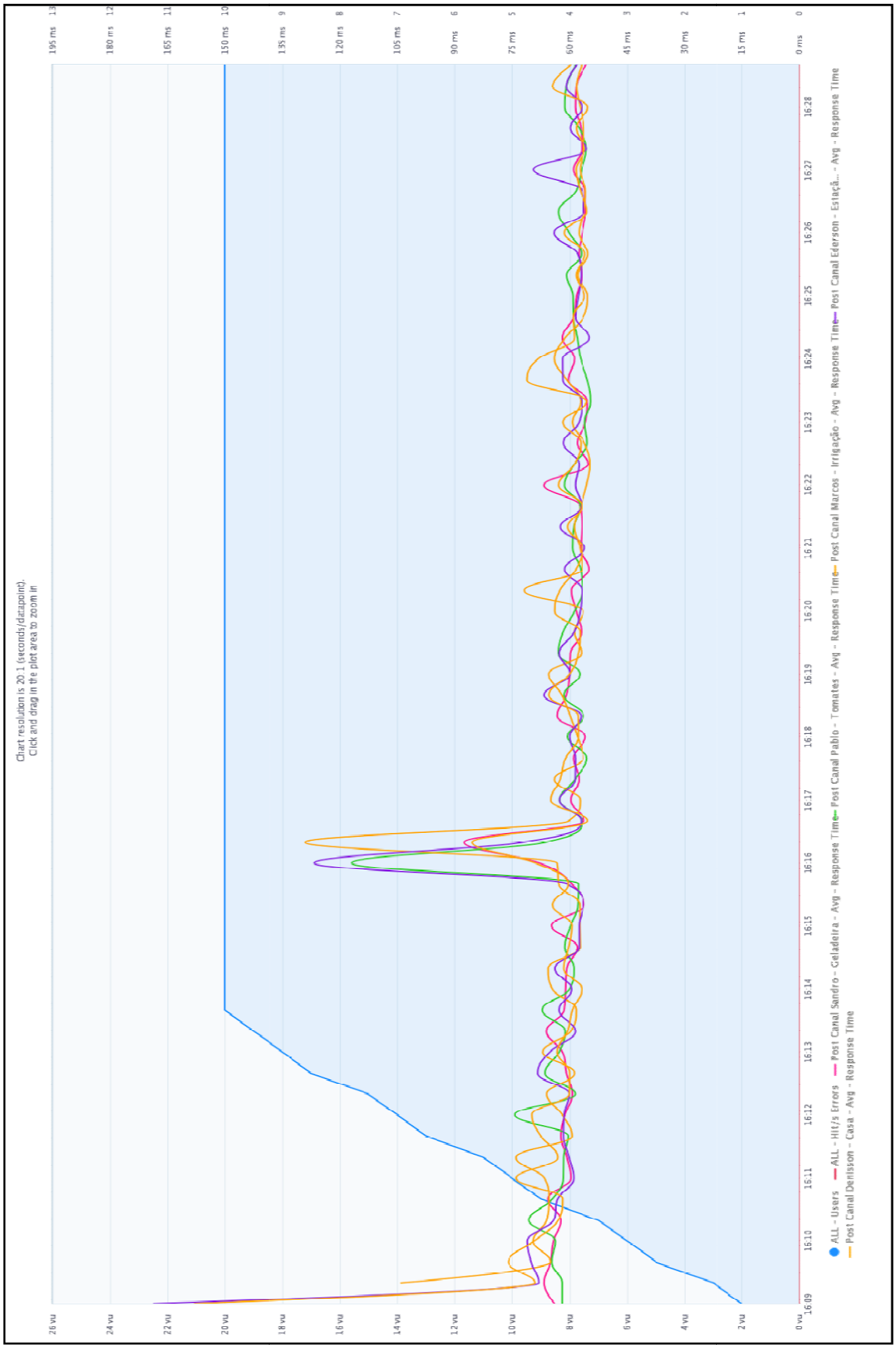
Figura 17 - Excluir tipo de usuário



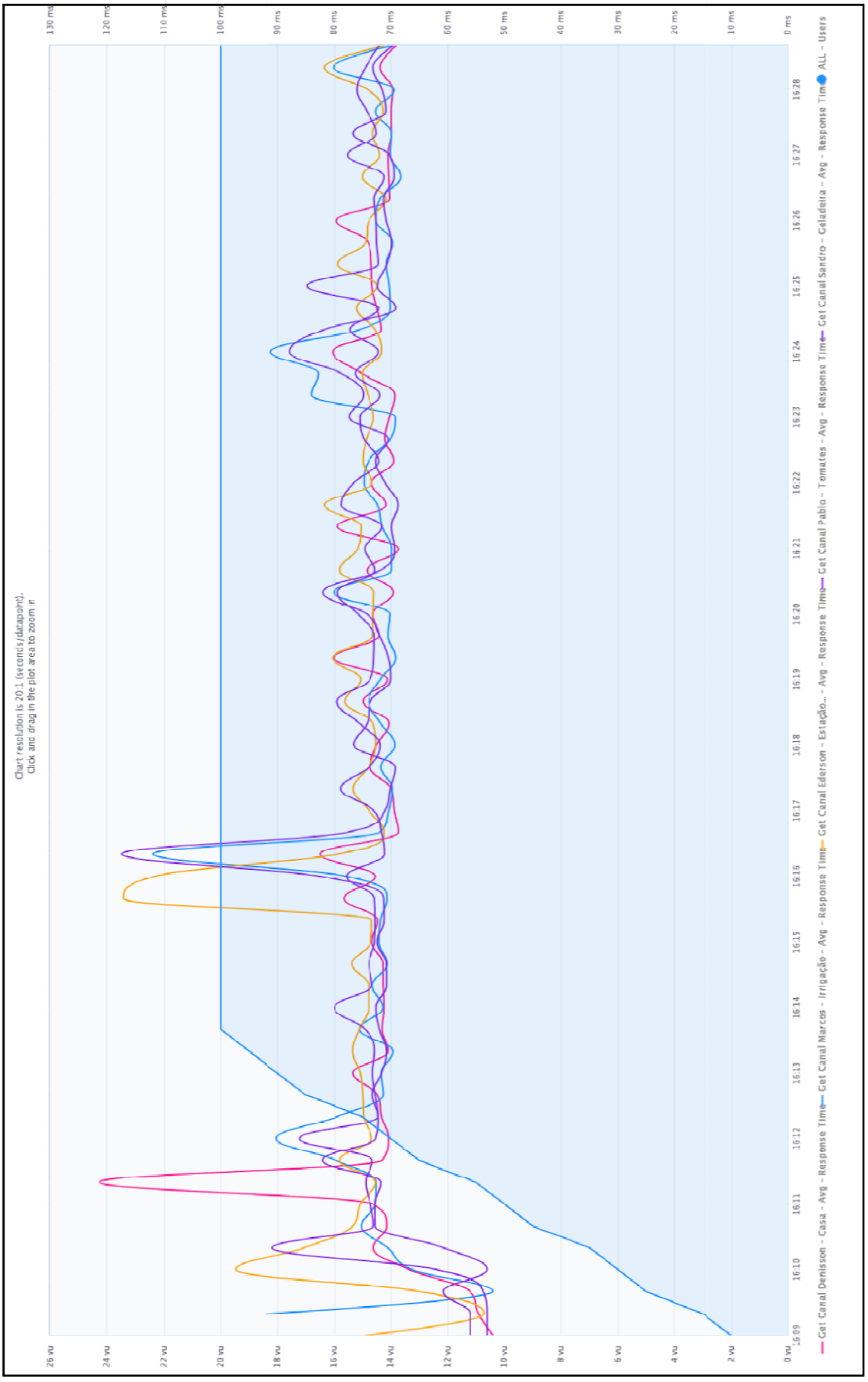
APÊNDICE D – TESTES DE LATÊNCIA REALIZADOS NA API



APÊNDICE E – TESTES REALIZADOS NA API POR POST



APÊNDICE F – TESTES REALIZADOS NA API POR GET



APÊNDICE G – ESTATÍSTICAS POR TIPO DE REQUISIÇÃO

Element Label	# Samples	Response Time (ms)	Avg. Hits/s	90% line (ms)	95% line (ms)	99% line (ms)	Response Time (ms)	Response Time (ms)	Bandwidth (KBytes/s)	Percentage
ALL	8246	67.51	6.89	76	83	148	47	558	85.37	0%
Get Canal Denisson - Casa	824	72.78	0.7	76	82	147	52	474	12.16	0%
Get Canal Ederson - Estação Meteorologia	832	76.34	0.7	79	92	145	53	558	25.44	0%
Get Canal Marcos - Irrigação	815	73.61	0.69	76	91	163	51	302	12.17	0%
Get Canal Pablo - Tomates	820	74.4	0.7	78	85	140	51	187	22.68	0%
Get Canal Sandro - Geladeira	828	73.49	0.7	76	86	161	48	365	13.23	0%
Post Canal Denisson - Casa	825	61.18	0.7	67	81	148	49	238	0.11	0%
Post Canal Ederson - Estação Meteorologia	833	61.63	0.7	68	82	156	48	300	0.11	0%
Post Canal Marcos - Irrigação	817	61.04	0.7	67	76	150	49	437	0.11	0%
Post Canal Pablo - Tomates	822	60.88	0.7	67	76	135	48	494	0.11	0%
Post Canal Sandro - Geladeira	830	59.78	0.7	66	71	107	47	428	0.11	0%