

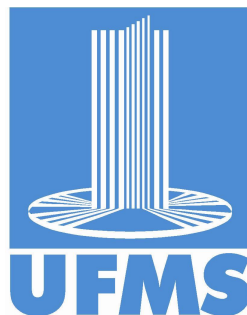
Trabalho de Conclusão de Curso

API para monitoramento de servidores
Linux.

Eduardo Balan

Orientação: Prof. Me. Kleber Kruger

Bacharelado em Sistemas de Informação



Sistema de Informação
Universidade Federal de Mato Grosso do Sul
15 de dezembro de 2017

API para monitoramento de servidores Linux.

Coxim, 15 de dezembro de 2017.

Banca Examinadora:

- Prof. Me. Kleber Kruger (CPCX/UFMS) - Orientador
- Prof.
- Prof.

Resumo

Abstract

Agradecimientos

Conteúdo

Lista de Figuras	8
Lista de Tabelas	9
Lista de Quadros	10
1 Introdução	11
1.1 Justificativa	12
1.2 Objetivos	12
1.2.1 Objetivo Geral	12
1.2.2 Objetivos Específicos	12
1.3 Organização da Proposta	14
2 Fundamentação Teórica	15
2.1 Arquitetura Cliente-Servidor	15
2.2 <i>Web Services</i>	16
2.3 Banco de Dados	17
2.4 HTTP/1.1 <i>HyperText Transfer Protocol</i>	18
2.4.1 Conexões	18
2.4.2 Métodos	18
2.4.3 Códigos de Resposta	19
2.4.4 Exemplo de utilização do HTTP/1.1	19
2.5 Arquitetura ReST	21
2.6 Testes Automatizados	22
2.6.1 Testes de Unidade	22

3	Metodologia	23
3.1	Spring Framework	23
3.1.1	<i>Spring Boot</i>	23
3.1.2	spring-boot-test	24
3.1.3	Spring-Data	25
3.2	Apache Maven	25
3.3	Boost	26
4	Desenvolvimento	27
4.1	MonitorWeb-API	27
4.1.1	Diagrama De Classes	28
4.1.2	Configurações Apache Maven	39
4.1.3	Configurações application.properties	40
4.1.4	Estrutura do Projeto	41
4.1.5	Consumindo serviços do MonitorWeb-API	46
4.2	MonitorWeb-Cli	46
5	Resultados	47
5.1	Desempenho da Biblioteca <i>FaultRecovery</i>	47
5.2	Desempenho e Eficiência da Classe TData	47
6	Conclusão	48
	Referências Bibliográficas	49
	Apêndices	50
A	Anexos	51
B	Anexos	55

Lista de Figuras

2.1	Arquitetura Cliente-Servidor.	16
4.1	Diagrama de classe resumido 1.	29
4.2	Diagrama de classe resumido 2.	32
4.3	Diagrama de classe resumido 3.	34
4.4	Diagrama de classe resumido 3.	36
4.5	Diagrama de classe completo.	38
4.6	Estrutura de pastas do projeto.	42
4.7	Pasta Repository.	43
4.8	Pasta Business.	44
4.9	Pasta Service.	44
4.10	Pasta Entitybbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb.	46

Lista de Tabelas

2.1	Métodos da solicitações HTTP.	19
2.2	Códigos de resposta da solicitação HTTP.	20
4.1	Variáveis da classe Servidor e suas descrições.	28
4.2	Variáveis da classe ServidorConfig e suas descrições.	30
4.3	Variáveis da classe ServidorConfigDb e suas descrições.	33
4.4	Variáveis da classe ServidorConfigInformacoesDb e suas descrições.	35
4.5	Descrição das pastas do projeto.	42
4.6	Variáveis da classe ServidorConfigInformacoesDb e suas descrições.	45
A.1	Variáveis da classe InformacoesCpu e suas descrições.	51
A.2	Variáveis da classe InformacoesMemoria e suas descrições.	51
A.3	Variáveis da classe InformacoesSwap e suas descrições.	51
A.4	Variáveis da classe MonitoramentoCpu e suas descrições.	52
A.5	Variáveis da classe MonitoramentoMemoria e suas descrições.	52
A.6	Variáveis da classe MonitoramentoSwap e suas descrições.	52
A.7	Variáveis da classe MonitoramentoPostgres e suas descrições.	53
A.8	Variáveis da classe ServidorConfigInformacoesDb e suas descrições.	53
A.9	Variáveis da classe Usuario e suas descrições.	54
A.10	Variáveis da classe Dominio e suas descrições.	54

Lista de Quadros

1	Formato de uma requisição HTTP	20
2	Exemplo de uma requisição HTTP utilizando o método GET.	20
3	Formato de uma resposta HTTP	20
4	Exemplo de uma resposta HTTP com status 200.	21
5	pom.	25
6	Arquivo POM com as principais dependências do projeto.	39
7	Arquivo application.properties com as principais configurações do projeto.	41
8	Entidade genérica da aplicação GenericEntity.	55
9	Entidade genérica GenericBO.	56

Capítulo 1

Introdução

A Internet é uma rede de computadores que interconecta milhares de dispositivos computacionais ao redor do mundo com centenas de milhares de usuários. Há pouco tempo, esses dispositivos eram basicamente computadores de mesa e servidores que realizavam diversas tarefas, como armazenamento e distribuição de dados e arquivos, gerenciamento de impressão e de usuários, conexão a outras redes, transmissão de informações tais como páginas da *web* e mensagens de e-mail, além de outras funcionalidades [1]. Com frequência, essas máquinas assim chamadas servidores são instaladas e mantidas em um local central de uma empresa por um administrador de sistemas [2].

Nos últimos tempos houve uma mudança, que é o uso da *web* não apenas para comunicação, mas como uma forma de executar aplicativos. Agora temos processadores de texto, planilhas e outros programas sendo executados como uma aplicação *web* em um navegador, em que suas principais informações ficam armazenadas nos servidores [3]. Uma vez que os dados dos usuários estão armazenados em locais remotos algumas vantagens podem ser obtidas, tais como *self-service* sob demanda e amplo acesso à rede, [4] mas por outro lado, manter estes dados seguros é imprescindível e o acompanhamento destes servidores é uma forma de garantia. É fácil perceber que a queda de um servidor pode comprometer a produtividade de usuários, principalmente se esse servidor for o único disponível ou se estiver executando serviços vitais [5].

Cabe observar, que segundo Carlos E. Marimoto, pouco a pouco, a internet tem se tornado o verdadeiro computador, e os computadores passam a ser cada vez mais um simples terminal, cuja única função é mostrar informações processadas por servidores remotos. Isso se tornou possível devido à popularização da ADSL (*Asymmetrical Digital Subscriber Line*), *wireless* e outras formas de acesso rápido e contínuo à internet. Futuramente, a tendência é que mais aplicativos passem a ser usados via *web*, tornando um computador desconectado cada vez mais limitado e inútil. Eventualmente, é possível que o próprio computador seja substituído por dispositivos mais simples e baratos, que sirvam como terminais de acesso [3].

1.1 Justificativa

Muitas empresas detêm valiosas informações guardadas em seus servidores, que podem ser de ordem técnica (por exemplo, o projeto de um novo *chip* ou novo *software*), comercial (como estudos sobre competidores ou planos de *marketing*), financeira (planos para uma venda de ações), jurídica (documentos sobre uma possível fusão ou aquisição), entre outras possibilidades. Além das ameaças causadas por invasores, dados valiosos podem ser perdidos por acidente. Algumas das causas comuns de perda acidental de dados são fenômenos naturais, como enchentes, terremotos, guerras, motins; erros de hardware ou de software (defeitos na CPU, discos ou fitas com problemas de leitura, surtos ou falhas de energia, sujeira, erros de programas e temperaturas extremas); e erros humanos (entrada incorreta de dados, montagem incorreta de disco ou fita, execução de programas errado, entre outro) [6, 7].

Em 27 de dezembro de 2005, um incêndio destruiu seis dos dez andares do prédio do INSS (Instituto Nacional de Seguro Social), em Brasília [8]. Segundo o ministro da Previdência e Assistência Social da época, Nelson Machado, as maiores perdas foram de informações de receita previdenciária, informações do sistema central e processos físicos, dívidas de empresas, processos de fraudes e autos de infração [9]. O presidente da Comissão de Fiscalização e Controle da Câmara dos Deputados, deputado Alexandre Cardoso, estimou, que os prejuízos, naquela época, referentes à processos administrativos foram equivalentes a R\$ 60 bilhões, tendo a previdência cópias de pelo menos R\$ 53 bilhões, concluindo que a união teria perdido em torno de R\$7 bilhões [10].

A maioria dessas causas podem ser tratadas com a manutenção adequada dos *backups*, preferivelmente em lugar distante dos dados originais. Embora proteger dados de perda acidental possa parecer banal, se comparado a proteger contra invasores inteligentes, na prática provavelmente mais danos são causados pelo primeiro que pelo ultimo [2, 7].

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste trabalho é estudar as formas utilizadas para prevenir erros, e aplicar essas técnicas na criação de ferramentas de monitoramento para servidores. Todas as informações de monitoramento ficarão a disposição de seus usuários através de um *web* site central, facilitando o trabalho de acompanhamento das rotinas dos servidores, e prevenções de problemas futuros.

1.2.2 Objetivos Específicos

- Pesquisar as principais formas utilizadas para prevenir os erros em servidores e identificando os pontos mais vulneráveis.
- Estudar como fazer leituras de informações do hardware, tais como dados dos discos

rígidos, processador, memória, temperatura da placa mãe, etc...

- Criar um *web* site para visualizar as informações dos servidores.
- Criar uma API para sincronizar as informações entre o sistema e *web* site.

Capítulo 2

Fundamentação Teórica

Neste Capítulo são apresentados os conceitos utilizados neste trabalho de acordo com a literatura estudada. Na seção 2.1 explica-se os conceitos da arquitetura cliente-servidor. Na Seção 2.2 explicase os conceitos de um *web-service*. Na Seção 2.3 explica-se o conceito de Banco de Dados. Na Seção 2.4 explica-se o conceito do protocolo HTTP/1.1 seus metodos forma de utilização e respostas esperadas. Na Seção 2.5 é apresentada o estilo de desenvolvimento da arquitetura ReST e na Seção 2.6 são mostradas as principais vantagens e melhorias adquiridas por meio dos testes automatizados.

2.1 Arquitetura Cliente-Servidor

Em uma arquitetura cliente-servidor há um hospedeiro sempre em funcionamento, denominado servidor, que atende à requisições de muitos outros hospedeiros, denominados clientes. Estes podem estar em funcionamento esporadicamente ou a todo o tempo. Um exemplo clássico é a aplicação *web* na qual um servidor *web* que está sempre em funcionamento atende à requisições de *browsers* de hospedeiros clientes. Ao receber uma requisição de um objeto de um hospedeiro cliente, o servidor *web* responde enviando o objeto requisitado a ele. Observe que, na arquitetura cliente-servidor, os clientes não se comunicam diretamente uns com os outros: por exemplo, na aplicação *web*, dois *browsers* não se comunicam diretamente. Outra característica da arquitetura cliente-servidor é que o servidor tem um endereço fixo, bem conhecido, denominado endereço IP (*Internet Protocol*). Devido a esse característica do servidor e devido ao fato de ele estar sempre em funcionamento, um cliente sempre pode contatá-lo enviando um pacote ao endereço do servidor. Algumas das aplicações mais conhecidas que empregam a arquitetura cliente-servidor são *web*, FTP (*File Transfer Protocol*), Telnet e e-mail. Essa arquitetura cliente-servidor é mostrada na Figura 2.1 em que diversos clientes utilizando, computadores, notebooks e celulares realizam comunicação com um servidor [1].

Em aplicações cliente-servidor, muitas vezes acontece de um único hospedeiro servidor ser incapaz de atender a todas as requisições de seus clientes. Um site *web* pode ficar rapidamente saturado se tiver apenas um servidor para atender grande número de requisições. Por essa razão, um grande conjunto de hospedeiros - às vezes coletivamente chamados *data center* - freqüentemente é usado para criar um servidor virtual poderoso em arquitetura cliente-

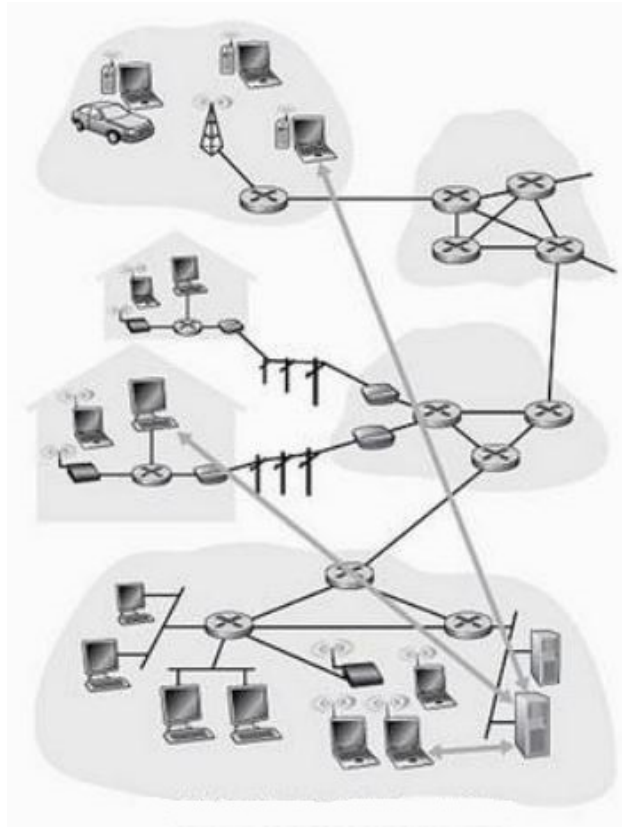


Figura 2.1: Representa a arquitetura cliente-Servidor, onde diversos tipos de cliente se comunicam com um mesmo servidor.

servidor, aumentando a capacidade de resposta de requisições [1].

2.2 *Web Services*

O *Web Service* é um *software* que atende a uma função de negócio específica para seus clientes. Ele recebe requisições de clientes e as responde ocultando todo o detalhamento do seu processamento. Normalmente as transações estão em formato XML (*Extended Markup Language*), e são transmitidas utilizando o protocolo HTTP (*HyperText Transfer Protocol*), entretanto podem ser utilizados outros protocolos de transporte [4].

Os *Web Services*, aliados aos padrões estabelecidos da Internet, podem fazer com que os dados fluam entre as várias entidades, como o governo e empresas interagindo entre si, reduzindo custos de modo a possibilitar maior facilidade e rapidez na troca de informações [11]. Um exemplo utilizado nacionalmente é a NF-e (Nota Fiscal Eletrônica) em que diversas empresas com *softwares* diferentes enviam arquivos xml utilizando o protocolo https (*Hyper Text Transfer Protocol Secure*) para o *web service* da receita federal, aguardam a realização dos processamentos de validação e recebem um arquivo xml de retorno com as informações desejadas.

Um *web wervice* deve executar unidades completas de trabalho, não dependendo do estado de outros componentes externos. Outra definição importante é que eles devem ser

"*Stateless*" ou seja, cada requisição é considerada uma transação independente que não está relacionada a qualquer requisição anterior de forma que a comunicação consista de pares de requisição e resposta independentes [4, 12].

2.3 Banco de Dados

Um Sistema de banco de dados é basicamente um sistema computadorizado de manutenção de registros. O banco de dados, por si só, pode ser considerado como o equivalente eletrônico de um armário de arquivamento; ou seja, ele é um repositório ou recipiente para uma coleção de arquivos de dados computadorizados. Os usuários de um banco de dados podem solicitar que o sistema realize diversas operações envolvendo tais arquivos, por exemplo:

- Acrescentar novos arquivos ao banco de dados
- Inserir dados em arquivos existentes
- Buscar dados de arquivos existentes
- Excluir dados de arquivos existentes
- Alterar dados em arquivos existentes
- Remover arquivos existentes do Banco de dados

As informações contidas no banco de dados em questão podem ser qualquer coisa que tenha algum significado ou indivíduo ou à organização a que o sistema deve servir, ou seja, qualquer coisa que seja necessária para auxiliar no processo geral das atividades desse indivíduo ou dessa organização

Os sistemas de bancos de dados Estão disponíveis em máquinas que variam desde pequenos computadores de mão (hand-helds e celulares) ou computadores pessoais até os maiores mainframes ou clusters de computadores de grande porte. Normalmente sistemas em máquinas grandes costumam ser multiusuários, enquanto os que estão em máquinas menores tendem a ser de monousuários. Um sistema monousuário é um sistema em que no máximo um usuário pode acessar o banco de dados em determinado momento; um sistema multiusuário é aquele em que muitos usuários podem acessar o banco de dados ao mesmo tempo; porém, a distinção é irrelevante para a maioria dos usuários, pois um dos objetivos dos sistemas multiusuários, em geral, é que cada usuário se comporte como se estivesse trabalhando com um sistema monousuário.

As informações contidas no banco de dados são persistentes, ou seja, uma vez que um operação de inserir é aceita pelo SGBD (Sistema de Gerenciamento de Banco de Dados), ela só poderá ser removida por outra requisição explícita ao SGBD, e não como um mero efeito colateral, como algum programa concluindo sua execução apagar todos os registros.

As transações realizadas pelo SGBD utilizam o conceito de ACID(Atomicidade, Consistência, Isolamento e Durabilidade):

- Atomicidade quer dizer que um bloco de transações deve ter todas as suas operações executadas em caso de sucesso ou nenhuma operação executada, mesmo que o sistema venha a falhar.
- Consistência diz que a execução de uma transação deve levar o banco de dados de um estado consistente a um outro estado consistente, dessa forma, uma transação deve respeitar as regras de integridade dos dados
- Isolamento quer dizer que transações paralelas não interferem umas nas outras, ou seja, se um usuário tentar alterar o salário de um funcionário e outro usuário tentar alterar o mesmo salário ao mesmo tempo uma transação só iniciará quando a outra for completamente terminada.
- Durabilidade diz que as transações com sucesso devem persistir no banco de dados mesmo em caso de falha.

2.4 HTTP/1.1 *HyperText Transfer Protocol*

O protocolo HTTP (*HyperText Transfer Protocol*) é o protocolo utilizado em toda a *World Wide Web*. Ele especifica o formato das mensagens que os clientes enviam aos servidores e também o formato de respostas que receberão. Cada interação consiste em uma solicitação ASCII, seguida por uma resposta RFC 822.

Na utilização do protocolo HTTP/1.1 todos os clientes e todos os servidores devem obedecer a esse protocolo que é definido na RFC 2616 [2].

2.4.1 Conexões

O modo habitual de cliente entrar em contato com um servidor é estabelecer uma conexão TCP (*Transmission Control Protocol*) para a porta 80 da máquina servidora, embora esse procedimento não seja exigido formalmente. A vantagem de se usar o TCP é que ambas as partes não têm de se preocupar com mensagens perdidas, mensagens duplicadas, mensagens longas ou confirmações. Todos esses assuntos são tratados pela implementação do TCP [2].

O HTTP/1.1 diferentemente de seus antecessores, admite conexões persistentes. Com elas, é possível estabelecer uma conexão TCP, enviar uma solicitação e obter uma resposta, e depois enviar solicitações adicionais e receber respostas adicionais. Amortizando o custo da instalação e da liberação do TCP por várias solicitações, o processamento relativo devido ao TCP é muito menor por solicitação. Também é possível transportar as solicitações por *pipeline*, ou seja, enviar a segunda solicitação antes de chegar a resposta da primeira [2].

2.4.2 Métodos

O HTTP foi criado de modo mais geral que simplesmente para utilização na *web*, visando às futuras aplicações orientadas a objetos. Por essa razão, são aceitas operações chamadas métodos, diferentes da simples solicitação de uma página da *web*. Cada solicitação consiste

em uma ou mais linhas de texto ASCII, sendo a primeira palavra da primeira linha o nome do método solicitado. Os métodos internos estão listados na Tabela 2.1 [2].

Método	Descrição
GET	O método GET solicita ao servidor que envie a página (ou objeto, no caso mais genérico; na prática, apenas um arquivo). A grande maioria das solicitações a servidores da <i>web</i> tem a forma de métodos GET.
HEAD	O método HEAD solicita apenas o cabeçalho da mensagem, sem a página propriamente dita. Esse método pode ser usado para se obter a data da última modificação feita na página, para reunir informações destinadas à indexação, ou apenas para testar a validade de um URL (Uniform Resource Locator).
PUT	O método PUT grava em uma página já existe. Esse método possibilita a criação de um conjunto de páginas da <i>web</i> em um servidor remoto. O corpo da solicitação contém a página.
POST	O método POST grava novos dados e são "anexados" a ele, em um sentido mais genérico [2].
DELETE	O método DELETE exclui a página. Não há garantia de que DELETE tenha sido bem-sucedido pois, mesmo que o servidor HTTP remoto esteja pronto para excluir a página, o arquivo subjacente pode ter um modo que impeça o servidor HTTP de modificá-lo ou excluí-lo [2].
TRACE	O método TRACE serve para depuração. ele instrui o servidor a enviar de volta a solicitação. Esse método é útil quando as solicitações não estão sendo processadas corretamente e o cliente deseja saber qual solicitação o servidor recebeu de fato [2].
CONNECT	O método CONNECT não é usado atualmente. Ele é reservado para uso futuro [2].
OPTIONS	O método OPTIONS fornece um meio para que o cliente consulte o servidor sobre suas propriedades ou sobre as de um arquivo específico [2].

Tabela 2.1: Métodos das solicitações HTTP e suas descrições [2].

2.4.3 Códigos de Resposta

Toda solicitação obtém uma resposta que consiste em uma linha de status e, possivelmente, informações adicionais (por exemplo, uma página da *web* ou parte dela). A linha de status contém um código de status de três dígitos informando se a solicitação foi atendida e, se não foi, o motivo. O primeiro dígito é usado para dividir as respostas em cinco grupos importantes, como mostra-se na Tabela 2.2 [2].

2.4.4 Exemplo de utilização do HTTP/1.1

Este protocolo foi desenvolvido de maneira a ser o mais flexível possível para comportar diversas necessidades diferentes. Sempre que uma requisição é enviada ela segue o formato das requisições que pode ser visto no Quadro 1 [13].

Código	Descrição
1xx	Raramente são usados na prática [2].
2xx	Solicitação foi tratada com sucesso [2].
3xx	Informam ao cliente que ele deve procurar em outro lugar, usando uma URL diferente [2].
4xx	Solicitação falhou devido a um erro do cliente, como uma solicitação inválida ou uma página inexistente [2].
5xx	O próprio servidor tem um problema, seja causado por um erro em seu código ou por uma sobrecarga temporária [2].

Tabela 2.2: Códigos de resposta da solicitação HTTP [2].

```

1 <método> <URL> HTTP/<versão>
2 <Cabeçalhos - Vários cabeçalhos podem ser enviados mas um em cada linha>
3
4 <corpo da requisição>

```

Quadro 1: Formato de uma requisição HTTP

Quando um GET para `http://www.site.com.br/clientes` é realizado uma requisição é criada como pode ser visto no Quadro 2. Nesta requisição pode ser visto em sua primeira linha o método GET da solicitação seguido pela URL a qual esta sendo feito a solicitação e pela versão do protocolo HTTP. O servidor responsável por receber a solicitação e a porta são passado no cabeçalho da requisição precedido por "Host:". Na terceira linha pode ser visto um cabeçalho *Accept*, que serve para informar ao servidor qual tipo de dados a requisição espera receber [13].

```

1 GET /clientes HTTP/1.1
2 Host: www.site.com.br:80
3 Accept: text/xml

```

Quadro 2: Exemplo de uma solicitação HTTP utilizando o método GET, para `http://www.site.com.br/clientes` com o pedido de uma arquivo XML de resposta.

Assim que o servidor receber a solicitação ele irá processar as informações, e retornará uma resposta com um dos código mostrado no Tabela 2.2. A resposta retornada possuirá o formato mostrado no Quadro 3 [13].

```

1 HTTP/<versão> <código de status> <descrição do código>
2 <cabeçalhos>
3 <resposta>

```

Quadro 3: Formato de uma resposta HTTP

Para a solicitação mostrada no Quadro 2 uma resposta velida seria a representada pelo Quadro 4. Nesta resposta pode ser visto em sua primeira linha a versão do protocolo HTTP.

seguido pelo código de resposta e descrição do código. Na segunda linha temos um cabeçalho Content-Type que indica qual o formato da resposta, através de um tipo conhecido como *Media Type*. A terceira linha tem um cabeçalho Content-Length que informa qual o tamanho da resposta. A partir da quarta linha temos a resposta em XML da solicitação [13].

```
1 HTTP/1.1 200 OK
2 Content-Type: text/xml
3 Content-Length: 232
4 <clientes>
5   <cliente>
6     <id>1</id>
7     <nome>Alexandre</nome>
8     <dataNascimento>2012-12-01</dataNascimento>
9   </cliente>
10  <cliente>
11    <id>2</id>
12    <nome>Paulo</nome>
13    <dataNascimento>2012-11-01</dataNascimento>
14  </cliente>
15 </clientes>
```

Quadro 4: Exemplo de uma resposta HTTP com status 200, possuindo um Content-Type:text/xml informando que a resposta é em formato XML, e a resposta a solicitação.

2.5 Arquitetura ReST

Os princípios básicos de ReST (*Representational State Transfer*) são estilos de desenvolvimento de *web services* que teve origem na tese de doutorado de Roy Fielding. Este, por sua vez, é co-autor de um dos protocolos mais utilizados no mundo, o HTTP/1.1 (*HyperText Transfer Protocol*) [14] [13]. Assim, é notável que o protocolo ReST é guiado pelo que seriam as boas práticas de uso do HTTP/1.1 [13]:

Em ReST, cada recurso deve ter uma URL bem definida. Por exemplo, o conjunto dos usuários de um sistema pode ter mapeada para si uma URL `http://www.site.com/usuarios`. Caso queiramos apenas o usuário de ID 1, essa URL se torna `http://www.site.com/usuarios/1` [15].

Parâmetros adicionais que não fazem parte da definição do recurso propriamente dito e/ou sejam opcionais podem ser passados em formato de *query string*, ou seja, dados “anexados” à URL. Esses dados devem ser passados após o ‘?’ e contendo um nome, seguido de ‘=’ e seu respectivo valor, outros dados adicionais devem ser separados por ‘&’. Por exemplo, para utilizar paginação em um listagem de usuários, deve ser passado um *query string* contendo os valores desejados da paginação, como na requisição `http://www.site.com/usuarios?pagina=1&itemPorPagina=20` [15].

Note que as URLs seguem uma estrutura hierárquica, ou seja, o elemento seguinte obedece a um relacionamento com o elemento anterior. Ou seja, se quisermos o endereço do usuário,

devemos obter primeiro o usuário em questão e, depois, o endereço. Assim sendo, a URL seria `http://www.site.com/usuarios/1/endereco`. No entanto, se quiséssemos obter apenas o endereço de todos os usuários, aí teríamos uma URL `http://www.site.com/usuarios/endereco`. O endereço obedece a uma estrutura hierárquica em relação a usuários [15].

2.6 Testes Automatizados

Todo desenvolvedor de software já escreveu um trecho de código que não funcionava. E muitas vezes só descobriu que o código não funciona quando o cliente reporta o *bug*. Nesse momento o desenvolvedor perde confiança no código, e seu cliente perde a confiança na equipe de desenvolvimento [16].

Uma maneira para conseguir testar o sistema todo de maneira constante e contínua é automatizando os testes. Ou seja, escrevendo um programa que testa o seu programa. Esse programa invocaria os comportamentos do seu sistema e garantiria que a saída é sempre a esperada. Se isso for feito, teríamos diversas vantagens. O teste executaria muito rápido (afinal, é uma máquina!). Se ele executa rápido, logo ele seria rodado constantemente. Se for rodado constantemente, os problemas serão encontrados mais cedo. [17].

2.6.1 Testes de Unidade

Um teste de unidade não se preocupa com todo o sistema; ele está interessado apenas em saber se uma pequena parte do sistema funciona. Ele testa uma única unidade do nosso sistema. Geralmente, em sistemas orientados a objetos, essa unidade é a classe.

Testes automatizados são fundamentais para um desenvolvimento de qualidade. Sua existência traz diversos benefícios para o software, como o aumento da qualidade e a diminuição de *bugs* em produção [17].

Capítulo 3

Metodologia

Neste Capítulo são apresentadas as ferramentas utilizados neste trabalho de acordo com a literatura estudada. Na seção 3.1 explica-se sobre o *Spring Framework* e seus diversos módulos.

3.1 Spring Framework

O *Spring Framework* fornece um modelo abrangente de programação e configuração para aplicativos corporativos modernos baseados em Java - em qualquer tipo de plataforma de implantação. Um elemento-chave do *Spring* é o suporte infra-estrutural no nível de aplicação: o *Spring* se concentra no *core* (Núcleo) de aplicativos corporativos para que as equipes possam se concentrar na lógica comercial de nível de aplicativo [18].

Spring é um projeto de código aberto. Possui uma comunidade grande e ativa que fornece *feedback* contínuo com base em uma ampla gama de casos de uso do mundo real. Isso ajudou a *Spring* a evoluir com êxito [18].

O *Spring Framework* é dividido em módulos. As aplicações podem escolher quais módulos eles precisam. No *core* são os módulos do núcleo, incluindo um módulos de configuração e um mecanismo de injeção de dependência. Além disso, o *Spring Framework* fornece suporte fundamental para diferentes arquiteturas de aplicativos, incluindo mensagens, dados transacionais e persistência, e *web*. Inclui também a estrutura *web Spring MVC* baseada em *Servlet* [18].

3.1.1 Spring Boot

A primeira versão do Spring Boot veio da necessidade de o Spring Framework ter suporte a servidores web embutidos. Depois, a equipe do Spring percebeu que existiam outras pendências também, como fazer aplicações prontas para nuvem (cloud-ready applications) [14].

Atualmente o *Spring Boot* é considerado um facilitador para a criação de aplicativos baseados em *spring*, para que não seja necessário ficar perdendo tempo configurando diversas

recursos nem mesmo um servidor de aplicação [19]. O *Spring Boot* é capaz de interagir com diversos banco de dados, mainframe, realiza transação distribuída, e torna qualquer plataforma confiável para executar os seus sistemas [14].

O *Spring Boot* tem um conceito na especificação JEE, que acelera o desenvolvimento e simplifica bastante a vida de quem trabalha com aplicações do *Spring Framework* [14].

3.1.2 spring-boot-test

O *Spring boot test* é a biblioteca *Spring* responsável pelos testes automatizados que foram visto no Seção 2.6. A intenção dessa biblioteca é fazer os testes ficarem o mais fáceis possíveis através de anotações e injeção de dependência para tornar seu código menos dependente de diversos *Framework* do que seria com o desenvolvimento Java EE tradicional. O *Spring Boot Test* incorpora em seu projeto diversas bibliotecas, algumas delas podem ser vistas a seguir [20]:

JUnit

A plataforma JUnit é um *framework* para facilitar a criação de testes de unidade e em especial sua execução. Ele possui alguns métodos que tornam o código de teste bem legível e fácil de fazer as asserções [21].

Uma asserção é uma afirmação. Algumas vezes em determinados pontos do teste é preciso garantir que uma variável tenha um determinado valor, caso isso não ocorra, o teste deve indicar uma falha a ser reportada para o programador, indicando um possível bug [21].

Hamcrest

É uma biblioteca que trabalha com tratamento de objetos matcher que nada mais é do que uma classe cuja função é verificar se um dado objeto tem as propriedades desejadas. [22].

Mockito

Um teste unitário deve testar uma funcionalidade isoladamente. Os efeitos secundários de outras classes ou do sistema devem ser eliminados se possível. Isso pode ser feito através da utilização do Mockito. Com ele é possível simular que métodos foram chamados, criar objetos falsos, simular uma resposta do banco de dados e simular respostas de métodos [23].

JsonPath

É um DSL(Domain Specific Languages) para ler documentos JSON, ela oferece aos desenvolvedores uma maneira simples de extrair dados específicos de um json [24].

3.1.3 Spring-Data

A missão da *Spring Data* é fornecer um modelo de programação familiar e consistente, baseado em *Spring*, para acesso a dados. Este módulo facilita o uso de tecnologias de acesso a dados, bancos de dados relacionais e não-relacionais, estruturas de redução de mapas e serviços de dados baseados em nuvem [25]. Ele abstrai para o desenvolvedor aqueles detalhes repetitivos das implementação de acessos a dados, através de *templates* [26].

Este projeto contém muitos subprojetos específicos de banco de dados. Os projetos são desenvolvidos trabalhando em conjunto com muitas das empresas e desenvolvedores dessas tecnologias [25].

Spring-Data-jpa

Spring Data JPA visa melhorar significativamente a implementação da camadas de acesso a dados, reduzindo o esforço para a quantidade mínima necessária. Suas interfaces de repositório incluindo métodos de busca personalizados, e o *Spring* irá fornecer a implementação de acesso aos dados automaticamente [27].

3.2 Apache Maven

Apache Maven é uma ferramenta que pode ser usada para construir e gerenciar qualquer projeto baseado em Java. Ele permite que um projeto seja construído usando um arquivo POM.xml (Modelo de Objeto de Projeto) dentro desse arquivo são declaradas as dependências e características do seu projeto. Quando o Maven é executado ele faz a leitura desse arquivo e realiza o *download* das dependências em formato JAR (Java ARchive) necessários para construir. Os arquivos JAR ficam em um reposiciono central e isso permite aos usuários do Maven reutilizar JARs em todos os projetos e incentiva a comunicação entre projetos para garantir que os problemas de compatibilidade com versões anteriores sejam tratados [28].

No Quadro 5 pode ser visto um exemplo de um arquivo POM e o comentário (`<!--` comentário `-->`) em cada linha.

```
1 <project> <!-- Informa o inicio do arquivo -->
2   <modelVersion>4.0.0</modelVersion><!-- Versão do arquivo POM -->
3   <groupId>br.com.monitorweb</groupId><!-- Grupo ao qual o projeto pertence -->
4   <artifactId>monitorweb-api</artifactId><!-- Nome do projeto -->
5   <version>1.0</version><!-- Versão do projeto -->
6
7   <dependencies><!-- Informa o inicio das dependências -->
8
9       <dependency><!-- Informa o inicio de uma dependência. -->
10         <groupId>org.springframework.boot</groupId><!-- Grupo ao qual o
11         projeto a dependência -->
12         <artifactId>spring-boot-starter</artifactId><!-- Nome da dependência
13         -->
14         <version>1.5.9.RELEASE</version><!-- Versão da dependenteia -->
```

```
13         </dependency><!-- Informa o fim de uma dependência. -->
14
15     </dependencies><!-- Informa o fim das dependências -->
16
17 </project><!-- Informa o fim do arquivo -->
```

Quadro 5: pom.

3.3 Boost

Asio

Ptree

Capítulo 4

Desenvolvimento

Neste trabalho foi desenvolvido um sistema para monitoramento de servidores Linux, utilizando a arquitetura cliente-servidor. O sistema consiste em duas aplicações, um *web service* desenvolvido em Java para o qual foi dado o nome de MonitorWeb-API, e outra aplicação em C++, executada nos servidores Linux como cliente, tendo o nome de MonitorWeb-Cli.

O MonitorWeb-Cli realiza leitura dos dados de seu hospedeiro, tais como CPU (*Central Processing Unit*), memória, banco dados e *swap*. Esse procedimento é realizado de acordo com a configuração de tempo desejada pelo usuário. Para cada leitura realizada, o sistema realiza o envio dos dados para o MonitorWeb-API. Ele também pode realizar rotinas de *backups* (cópia de segurança) e *vaccum* (processo de limpeza no banco de dados) do banco dados PostgreSQL, tanto no hospedeiro quanto em outro computador a qual tenha acesso pela rede. Após efetuar esses procedimentos, o MonitorWeb-Cli envia mensagens para o MonitorWeb-API que por sua vez armazena essas informações.

Um MonitorWeb-API é responsável por receber os dados de diversos MonitorWeb-Cli e realizar a persistência no banco de dados. Também é responsável por armazenar as configurações dos clientes, tais como o intervalo de envio dos dados de monitoramento e as informações dos procedimentos de *backup*. Essas configurações são capturadas periodicamente conforme as configurações do usuário. A **API** também pode ser utilizada para disponibilizar os dados dos servidores para aplicações de *frontend* (DEFINIR O QUE É FRONTEND).

4.1 MonitorWeb-API

O MonitorWeb-API é um *web service* desenvolvido utilizando a tecnologia *Spring-boot* para o desenvolvimento de uma aplicação ReST que utiliza o protocolo HTTP/1.1 na comunicação com sistema clientes. A forma de passar dados entre os sistema foi utilizado o JSON (*JavaScript Object Notation*) em vez do XML por ter sua estrutura menor e consumir menos tráfego na rede [13]. Para a persistência dos dados foi utilizado o banco de dados PostgreSQL [14].

Um MonitorWeb-API possui a capacidade de trabalhar com diversos MonitorWeb-Cli simultaneamente. A classe Servidor, descrita na Tabela 4.1 representa o MonitorWeb-Cli.

A Subseção 4.1.5 descreve como realizar o cadastro dos recursos ReST do *web-service*.

Variáveis	Descrições
id	Número único para cada objeto do tipo Servidor. (Gerado Automaticamente)
dominio	A qual domínio o servidor pertence. (Não obrigatório). Esse atributo será tratado na Seção 4.1.1
dthr_cadastro	Data e hora do cadastro. (Valor gerado automaticamente)
nome	Nome que o usuário deseja dar ao servidor.
empresa	Nome da empresa (Não obrigatório)
observacao	Alguma observação a fazer sobre o servidor (Não obrigatório)

Tabela 4.1: Variáveis da classe Servidor e suas descrições.

4.1.1 Diagrama De Classes

Um diagrama de classe pode ser visto na Figura 4.1, em que é mostrado resumidamente o relacionamento da classe Servidor com as outras classes da aplicação. Esse diagrama está separado em três partes enumeradas, descritas a seguir.

- Figura 4.1, Grupo 1. As classes desse grupo possuem o prefixo Servidor e correspondem às configurações da aplicação e as configurações de rotinas do banco de dados.
- Figura 4.1, Grupo 2. Classes desse grupo possuem o prefixo Informacoes. Estas, representam registros cadastrados automaticamente pela aplicação MonitorWeb-Cli e fazem referência às informações do *hardware* (componentes físicos de um computador). Os registros dessas classes são armazenados apenas no momento de inicialização do sistema, pois os valores não podem ser alterados sem a reinicialização da máquina. Exemplos desses valores são o modelo do processador e a quantidade de memória física. A descrição de cada classe desse grupo pode ser vista no Apêndice A.
- Figura 4.1, Grupo 3. Classes desse grupo possuem o prefixo Monitoramento. Os objetos dessas classes representam registros gerados automaticamente pelo MonitorWeb-Cli com os valores de desempenho de *hardware* e *software* monitorados periodicamente. Exemplos dos valores armazenados por objetos dessas classes são: quantidade de memória e CPU que estão sendo utilizadas, situação dos procedimentos de *backups*. A descrição de cada classe desse grupo pode ser vista no Apêndice A

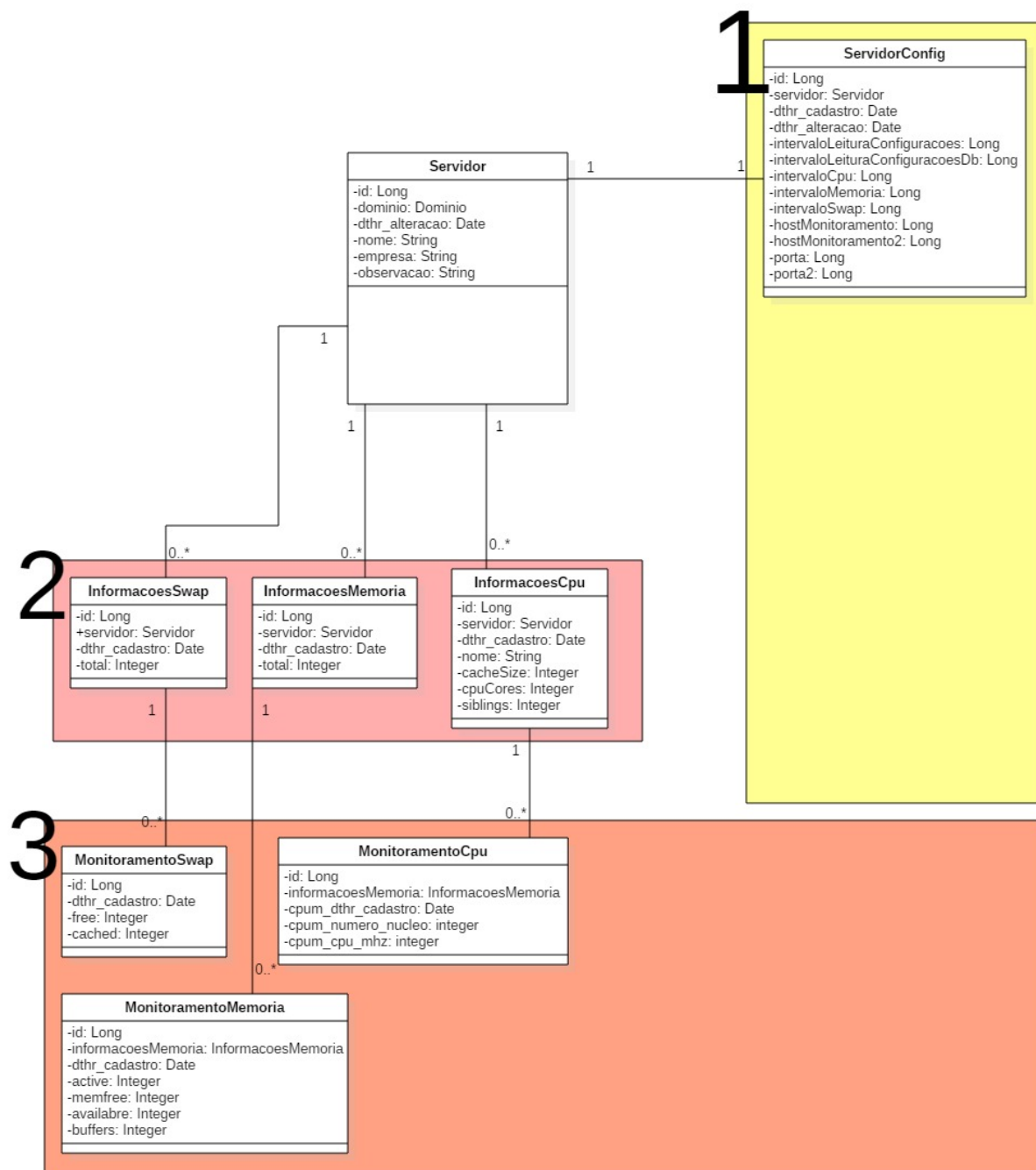


Figura 4.1: Diagrama de classe resumido 1, separados pelos grupos 1 à 3 que correspondem, respectivamente, às configurações que o usuário deve fazer, informações geradas pelo MonitorWeb-Cli e monitoramentos gerados pelo MonitorWeb-Cli

Classe ServidorConfig

Como pode ser visto na Figura 4.1, a classe **ServidorConfig** possui um relacionamento de um-para-um com a classe **Servidor**, indicando assim que um objeto de **Servidor** pode ter somente um **ServidorConfig**. Essa classe é vital para o funcionamento do MonitorWeb-Cli. Dentro

dela estão as configurações mínimas para que um MonitorWeb-Cli saiba como trabalhar. Na Tabela 4.2 explica-se detalhadamente a classe ServidorConfig.

Na Tabela 4.2, o valor padrão das variáveis são apenas sugestões. O usuário pode alterar qualquer um deles, porém, é altamente recomendado que esses valores nunca sejam iguais a 0. Caso isso aconteça, o MonitorWeb-Cli enviará o máximo de informações que puder sem qualquer intervalo de tempo. Isso acaba consumindo o máximo de recurso (memória, cpu e disco) do sistema operacional hospedeiro.

Variáveis	Descrições
id	Número único para cada objeto do tipo ServidorConfig. (Gerado Automaticamente)
servidor	Informa com qual servidor esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
dthr_alteracao	Data e hora que foi realizado a ultima alteração. (Gerado Automaticamente)(Essa funcionalidade não foi implementada)
intervaloLeituraConfiguracoes	Intervalo em segundos para o MonitorWeb-Cli ler as configurações da classe ServidorConfig, e se reconfigurar(Valor padrão 120 s).
intervaloLeituraConfiguracoesDb	Intervalo em segundos para o MonitorWeb-Cli ler as configurações da classe ServidorConfigDb, e se reconfigurar(Valor padrão 120 s). Essa classe é mostrada na Seção 4.1.1
intervaloCpu	Intervalo em segundos para o MonitorWeb-Cli enviar um registro da classe MonitoramentoCpu.(Valor padrão 1 s)
intervaloMemoria	Intervalo em segundos para o MonitorWeb-Cli enviar um registro da classe MonitoramentoMemoria.(Valor padrão 1 s)
intervaloSwap	Intervalo em segundos para o MonitorWeb-Cli enviar um registro da classe MonitoramentoSwap.(Valor padrão 1 s)
hostMonitoramento	IP que o MonitorWeb-Cli deve se comunicar.
hostMonitoramento2	IP secundário que o MonitorWeb-Cli deve se comunicar caso o servidor principal não esteja funcionando.(Essa funcionalidade não foi implementada)
porta	Porta da aplicação para o MonitorWeb-Cli saber em que porta o MonitorWeb-API esta rodando.
porta2	Porta da aplicação secundária para o MonitorWeb-Cli saber em que porta o MonitorWeb-API esta rodando.(Essa funcionalidade não foi implementada)

Tabela 4.2: Variáveis da classe ServidorConfig e suas descrições.

Classe ServidorConfigDb

No diagrama de classe mostrado na Figura 4.2 existem duas novas classes: a `ServidorConfigDb` no grupo um e a `MonitoreamentoPostgres` no grupo três. Além das classes, um novo grupo número 4 é mostrado. Nesse grupo temos as *enums* da aplicação, utilizadas em algumas classes.

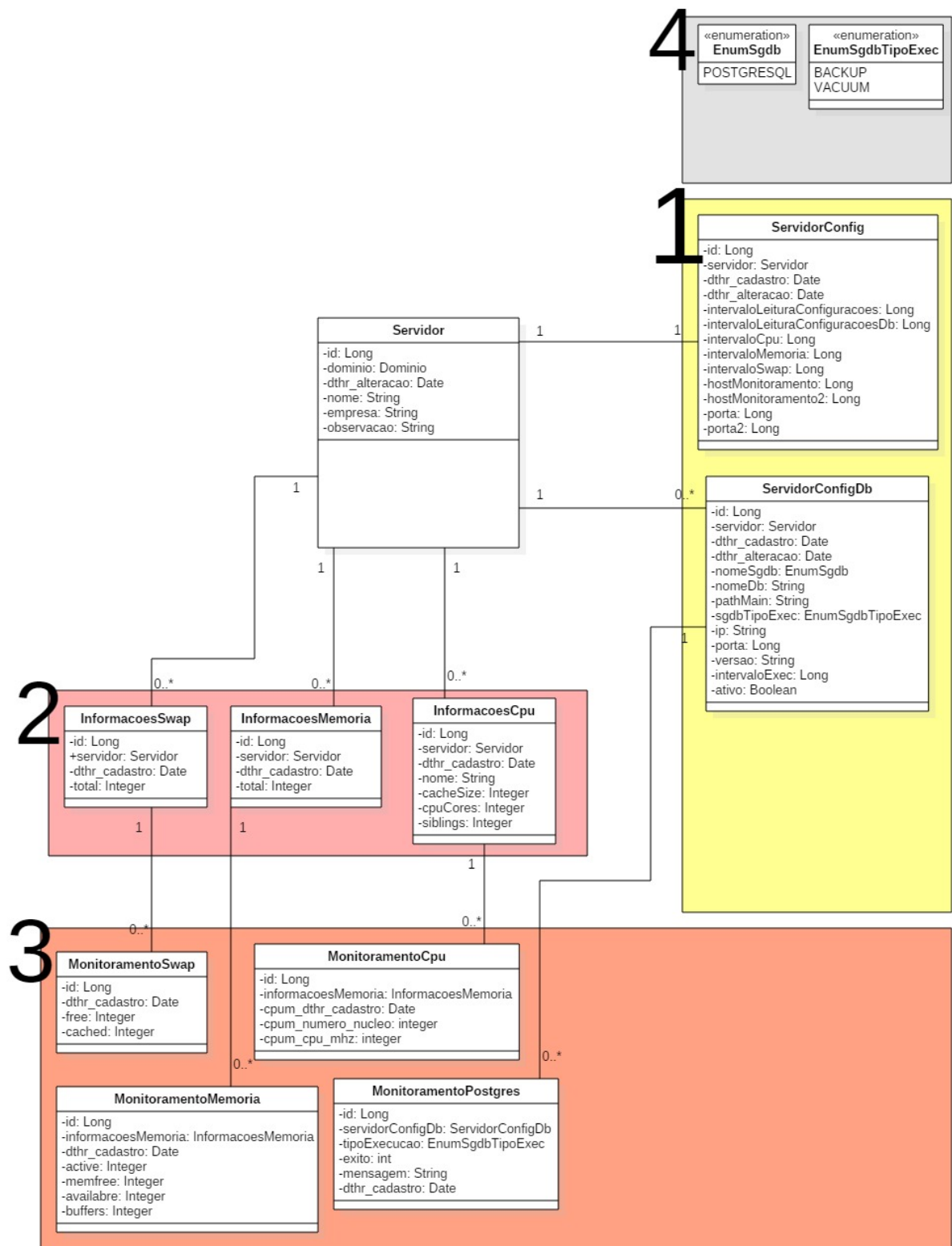


Figura 4.2: Diagrama de classe resumido 2, separados pelos grupos 1 à 4 que correspondem, respectivamente, às configurações que o usuário deve fazer, informações geradas pelo MonitorWeb-Cli, monitoramentos gerados pelo MonitorWeb-Cli e *enums* da aplicação.

A classe `ServidorConfigDb` é responsável por criar rotinas de *backup* e *vacuum* para o `MonitoreWeb-Cli` executar. Essa classe possui um relacionamento de muitos-para-um com a classe `Servidor`, o que indica que em um servidor pode ter diversas rotinas de *backup* e *vacuum*. Na Tabela 4.3 é explicada de forma minuciosa a classe `ServidorConfigDb`.

Essa classe também pode ser utilizada para o `MonitoreWeb-Cli` executar o procedimento de *backup* e *vacuum* em outro computador, para isso basta ser informado o IP da máquina que se deseja executar o procedimento e que a máquina esteja habilitada para fazer esse procedimento pela rede.

A classe `MonitorementoPostgres` é responsável por armazenar os resultados dos *backups* e dos procedimentos de *vacuum* da classe `ServidorConfigDb`. A descrição de cada variável dessas classe pode ser vista no Apêndice A.

Variável	Descrição
id	Número único para cada objeto do tipo <code>ServidorConfigDb</code> . (Gerado Automaticamente)
servidor	Indica com qual objeto <code>Servidor</code> esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
dthr_alteracao	Data e hora que foi realizado a ultima alteração. (Gerado Automaticamente)
nomeSgdb	É uma enum do tipo <code>EnumSgdb</code> , que informa em qual SGDB o procedimento será executado. Atualmente somente o <code>postgresql</code> é suportado.
nomeDb	Nome do banco de dados que o procedimento será executado.
pathMain	Caminho para as libs que executam o backup.
sgdbTipoExec	É uma enum do tipo <code>EnumSgdbTipoExec</code> o qual informa se o procedimento é um backup ou um vacuum.
ip	IP para onde devera ser executado o procedimento.
porta	Porta que sera executado o procedimento.
versao	Versão do SGDB.
intervaloExec	Intervalo de tempo que o <code>MonitoreWeb-Cli</code> deve executar esse procedimento.
ativo	Controle para ativar e desativar o procedimento.

Tabela 4.3: Variáveis da classe `ServidorConfigDb` e suas descrições.

Classe `ServidorConfigInformacoesDb`

No diagrama de classe mostrado na Figura 4.3 existem duas novas classes: a `ServidorConfigInformacoesDb` no Grupo 1 e a `MonitorementoPostgresInformacoes` no Grupo 3.

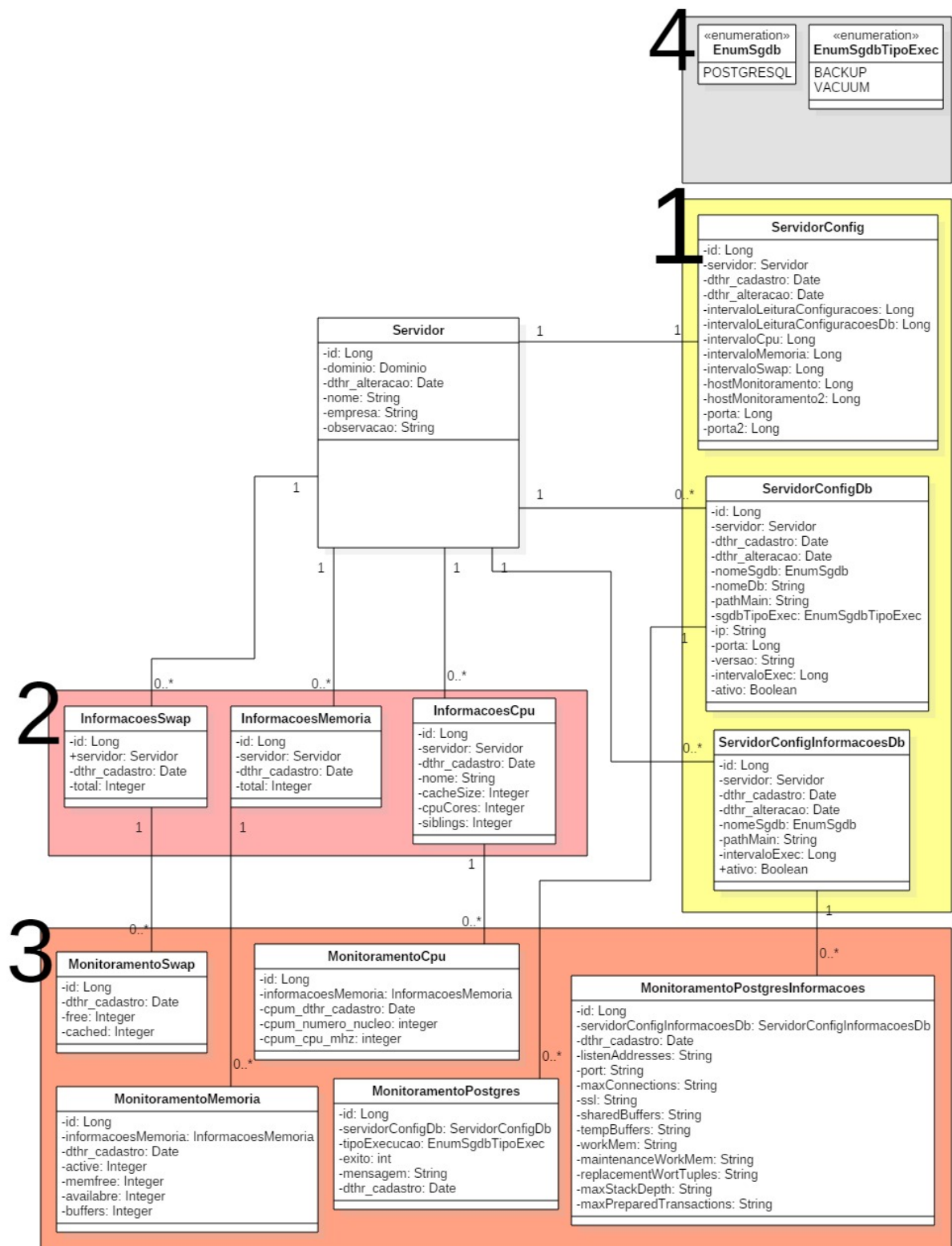


Figura 4.3: Diagrama de classe resumido 3, separados pelos grupos 1 à 4 que correspondem respectivamente a configurações que o usuário deve fazer, informações geradas pelo MonitorWeb-Cli, monitoramentos gerados pelo MonitorWeb-Cli e enums da aplicação.

A classe `ServidorConfigInformacoesDb` é responsável por criar uma rotina para ler o arquivo com as configurações principais do PostgreSQL (`postgresql.conf`). Essa classe possui um relacionamento de muitos-para-um com a classe `Servidor`, uma vez que o servidor pode ter várias versões do banco de dados. Na Tabela 4.4 descreve-se detalhadamente a classe `ServidorConfigInformacoesDb`.

A classe `MonitorementoPostgres` é responsável por armazenar os resultados da classe `ServidorConfigInformacoesDb`. A descrição de cada variável da classe `MonitorementoPostgres` pode ser vista no Apêndice A.

Variável	Descrição
id	Número único para cada objeto do tipo <code>ServidorConfigDb</code> . (Gerado Automaticamente)
servidor	Indica com qual objeto <code>Servidor</code> esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
dthr_alteracao	Data e hora que foi realizado a ultima alteração. (Gerado Automaticamente)
nomeSgdb	É uma enum do tipo <code>EnumSgdb</code> , que informa em qual SGDB o procedimento sera executado. Atualmente somente o PostgreSQL é suportado.
pathMain	Caminho para o arquivo <code>postgresql.conf</code> .
intervaloExec	Intervalo de tempo que o <code>MonitorWeb-Cli</code> deve executar esse procedimento.
ativo	Controle para ativar e desativar o procedimento.

Tabela 4.4: Variáveis da classe `ServidorConfigInformacoesDb` e suas descrições.

Diagrama de classe Usuários

No diagrama de classe mostrado na Figura 4.4 existe um Grupo 5. Esse grupo está relacionado a um cadastro de usuário. Ele não é utilizado pelo `MonitorWeb-Cli` mas pode ser utilizado por implementações futuras para uma aplicação *frontend* para o sistema.

No Grupo 5 existem duas classes: a `Usuario`, que corresponde ao cadastro de contas de usuário na aplicação, e a `Dominio`, que corresponde a uma classe para dividir os usuários por grupo. Dessa forma, uma empresa que tenha diversos setores pode separar mais facilmente qual usuário tem acesso a qual grupo de servidores.

Além das duas classes, existe a classe `UsuarioHasDominio` que é simplesmente uma classe para uma ligação de muitos-para-muitos entre as classes `Usuario` e `Dominio`. A descrição de cada variável da classe `Usuario` e `Dominio` pode ser vistas no Apêndice A.

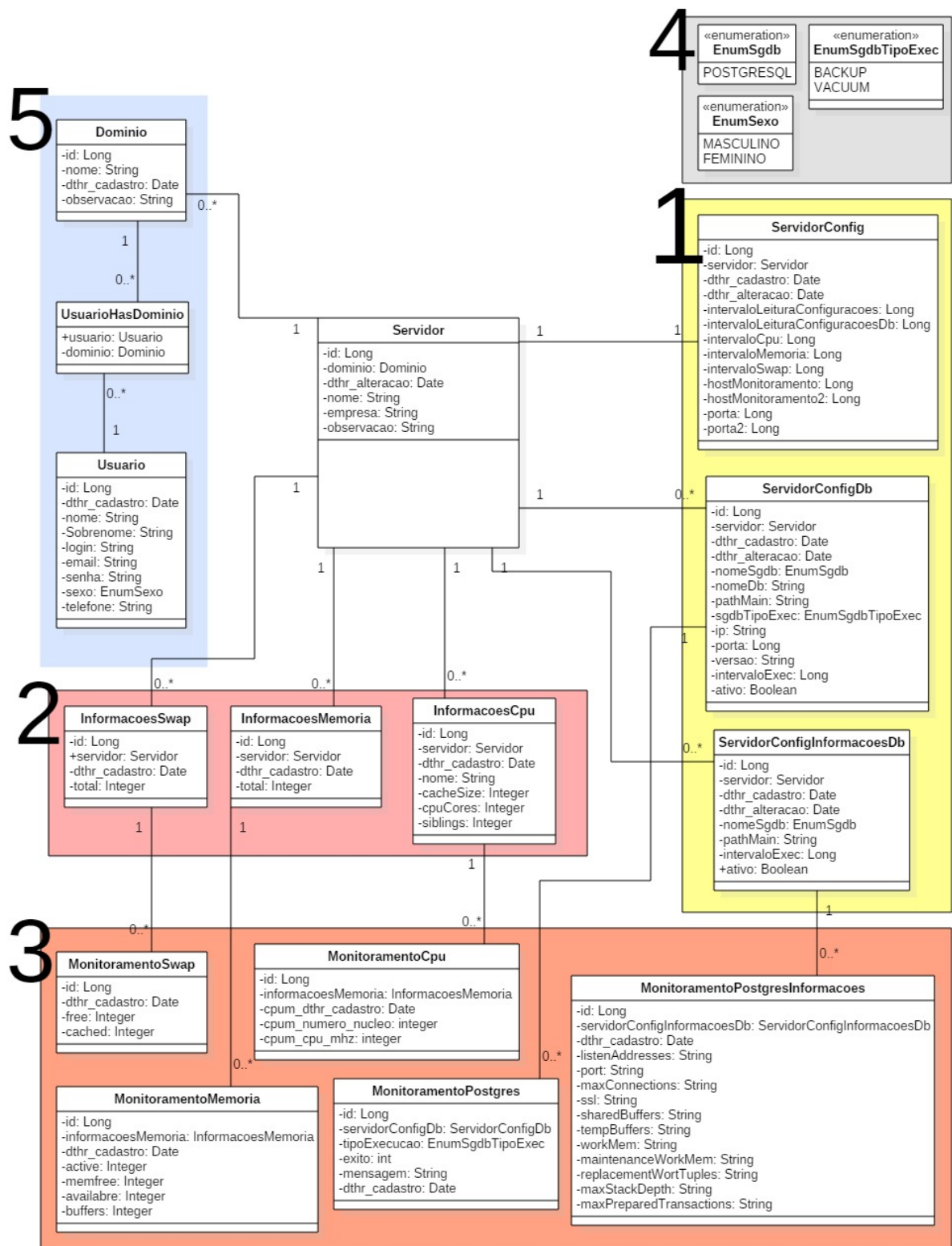


Figura 4.4: Diagrama de classe resumido 4, separados pelos grupos 1 à 5 que correspondem respectivamente a configurações que o usuário deve fazer, informações geradas pelo MonitorWeb-Cli, monitoramentos gerados pelo MonitorWeb-Cli, enums da aplicação e cadastro de usuários.

Diagrama de classe Completo

No diagrama de classe mostrado na Figura 4.5 existe uma nova classe abstrata chamada de `GenericEntity`, sendo essa a superclasse para todas as entidades do sistema. A sua função é implementar os métodos *equals* e *hashCode* da classe `Object`. EXPLICAR O PORQUE!!! Além disso, ela cria os métodos abstratos `getId`, `setId`, `getDthr_cadastro` e `setDthr_cadastro` obrigando todos os métodos que estenderem dela a implementá-los.

A classe `GenericEntity` recebe um tipo `T` genérico que estende de `Serializable`. Esse tipo `T` representa o tipo da variável `id` da entidade, para que a aplicação saiba como gerar determinados métodos. A classe `GenericEntity` pode ser vista no Apêndice B.

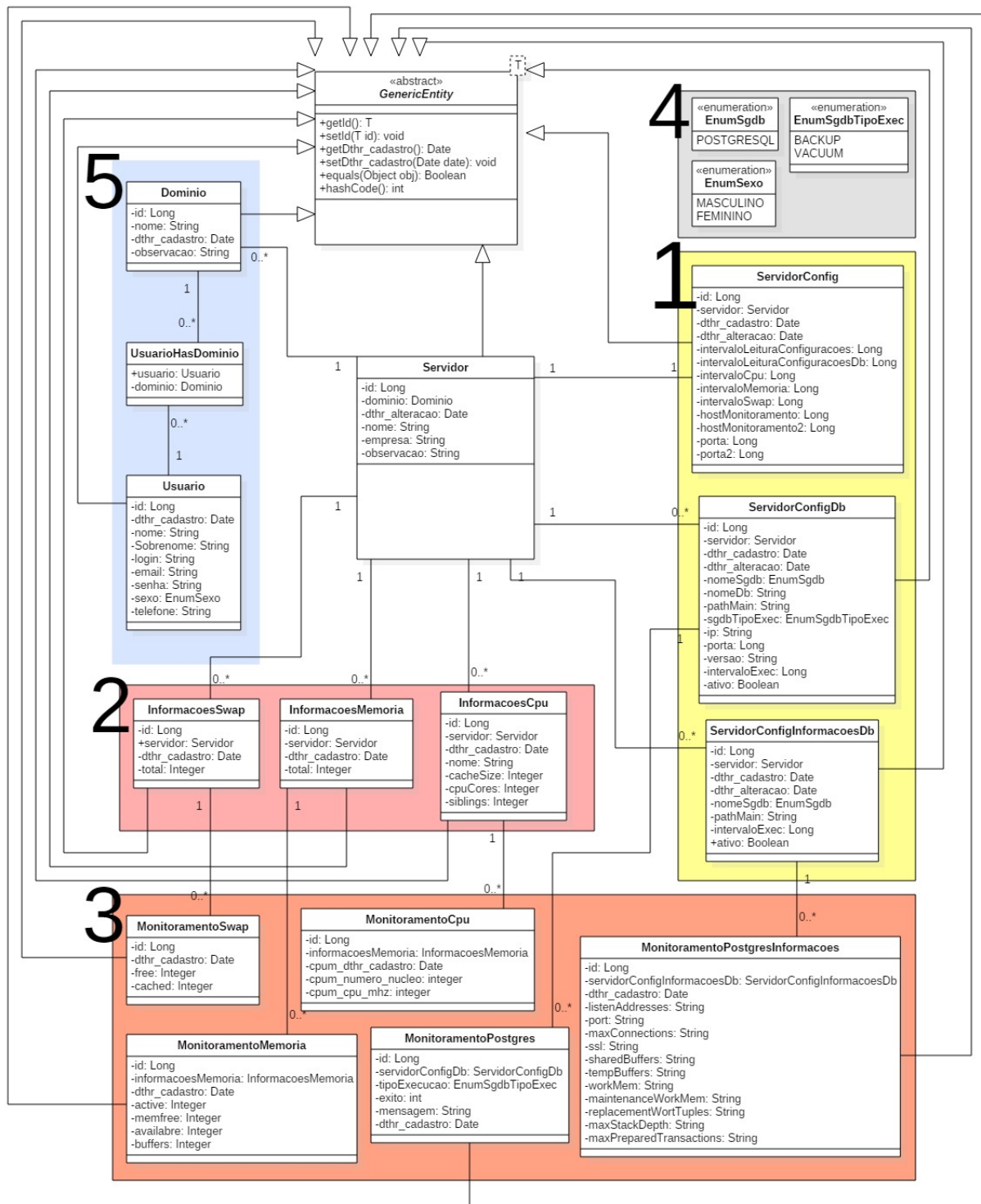


Figura 4.5: Diagrama de classe completo, separados pelos grupos 1 à 5 que correspondem respectivamente a configurações que o usuário deve fazer, informações geradas pelo MonitorWeb-Cli, monitoramentos gerados pelo MonitorWeb-Cli, enums da aplicação e cadastro de usuários.

 ----- kleber continuar a revisão daqui -----

4.1.2 Configurações Apache Maven

Como visto na Seção 3.2 o Apache Maven é um gerenciador de pacotes, que gerencia as dependências por meio do arquivo POM. Esse arquivo pode ser encontrado na raiz do projeto /pom.xml. No Quadro 6, é mostrado as principais dependências contidas nesse arquivo, a seguir é explicado cada uma das dependências.

- Quadro 6, linha 3 - spring-boot-starter-parent, Dependência do Spring-boot. Ele também cria um "parent" que faz com que não seja necessário especificar as versões nos outros pacotes do spring, pois ele já identificara a versão automaticamente [19].
- Quadro 6, linha 10 - spring-boot-starter-data-jpa, Dependência que spring-data-jpa. Ele traz automaticamente todas as dependências necessárias para o seu funcionamento [27].
- Quadro 6, linha 15 - spring-boot-starter-web, Essa dependência identifica que nossa aplicação será uma aplicação *web*, rodará usando tomcat, utilizará ReSTfull, seguindo padrão spring MVC (*Model View Controller*) [19].
- Quadro 6, linha 20 - spring-boot-starter-logging, Dependência responsável pelo registro de log automático [19].
- Quadro 6, linha 25 - spring-boot-starter-test, Dependência do Spring Boot com bibliotecas de teste unitário, incluindo JUnit, Hamcrest e Mockito [19].
- Quadro 6, linha 31 - postgresql, Dependência que fornece um conjunto padrão de interfaces para bancos de dados compatíveis com SQL [29].

```
1  <parent>
2      <groupId>org.springframework.boot</groupId>
3      <artifactId>spring-boot-starter-parent</artifactId>
4      <version>1.3.3.RELEASE</version>
5      <relativePath/>
6  </parent>
7
8  <dependency>
9      <groupId>org.springframework.boot</groupId>
10     <artifactId>spring-boot-starter-data-jpa</artifactId>
11 </dependency>
12
13 <dependency>
14     <groupId>org.springframework.boot</groupId>
15     <artifactId>spring-boot-starter-web</artifactId>
16 </dependency>
```

```
17
18 <dependency>
19   <groupId>org.springframework.boot</groupId>
20   <artifactId>spring-boot-starter-logging</artifactId>
21 </dependency>
22
23 <dependency>
24   <groupId>org.springframework.boot</groupId>
25   <artifactId>spring-boot-starter-test</artifactId>
26   <scope>test</scope>
27 </dependency>
28
29 <dependency>
30   <groupId>org.postgresql</groupId>
31   <artifactId>postgresql</artifactId>
32   <version>${postgresql.version}</version>
33 </dependency>
34
35 <dependency>
36   <groupId>com.github.springtestdbunit</groupId>
37   <artifactId>spring-test-dbunit</artifactId>
38   <version>${spring-test-dbunit.version}</version>
39   <scope>test</scope>
40 </dependency>
41
42 <dependency>
43   <groupId>com.h2database</groupId>
44   <artifactId>h2</artifactId>
45   <scope>test</scope>
46 </dependency>
```

Quadro 6: Arquivo POM com as principais dependências do projeto.

4.1.3 Configurações application.properties

O *Spring Boot* fornece um arquivo de configuração com o nome de `application.properties`, e pode ser encontrado na pasta `/src/main/resources/application.properties`. Dentro desse arquivo pode ser feitas configurações de acordo com a necessidade do projeto [19]. No Quadro 7 pode ser visto o `application.properties` do `MonitoreWeb-API` e a seguir uma lista contendo a descrição de cada configuração.

- Quadro 7 linha 2, `server.port` - Porta que a aplicação ira rodar [19].
- Quadro 7 linha 5, `spring.jackson.date-format` - Formato que a data sera apresentada [19].
- Quadro 7 linha 10, `logging.path` - Indica o local que ficara o arquivo de *log* [19].

- Quadro 7 linha 11 a 14, `logging.level.*` - Indique se aquele nível do spring terá o log ativado ou não. Alguns dos valores que eles podem receber são *TRACE*, *DEBUG*, *INFO*, *WARN*, *ERROR*, *FATAL*, ou *OFF* [19].
- Quadro 7 linha 19, `spring.datasource.url` - URL de conexão para o banco de dados [19].
- Quadro 7 linha 20, `spring.datasource.username` - Usuario do banco de dados [19].
- Quadro 7 linha 21, `spring.datasource.password` - Senha do banco de dados [19].
- Quadro 7 linha 22, `spring.jpa.database-platform` - Vários bancos de dados têm mais de um *Dialect*, e essa propriedade especifica que dialeto do hibernate ira utilizar [19].
- Quadro 7 linha 23, `spring.datasource.driverClassName` - Nome do *driver* que sera utilizado para conexão [19].

```

1  ## Portas
2  server.port=8081
3
4  ## Formatador de datas do jackson
5  spring.jackson.date-format= yyyy-MM-dd'T'HH:mm:ss.SSSZ
6
7  ## -----
8  ## LOGGING
9  ## -----
10 logging.path=../
11 logging.level.root=INFO
12 logging.level.org.hibernate = INFO
13 logging.level.org.springframework.web = INFO
14 logging.level.org.camunda=INFO
15
16 ## -----
17 ## POSTGRES
18 ## -----
19 spring.datasource.url=jdbc:postgresql://localhost/webmonitor
20 spring.datasource.username=postgres
21 spring.datasource.password=postgres
22 spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
23 spring.datasource.driverClassName=org.postgresql.Driver

```

Quadro 7: Arquivo `application.properties` com as principais configurações do projeto.

4.1.4 Estrutura do Projeto

O projeto é dividido em seis pastas que podem ser visto na Figura 4.6, e uma classe com nome `WebMonitorApp.class`, a qual é responsável por iniciar a aplicação. A Tabela 4.5 contem o nome das pastas e suas descrição.

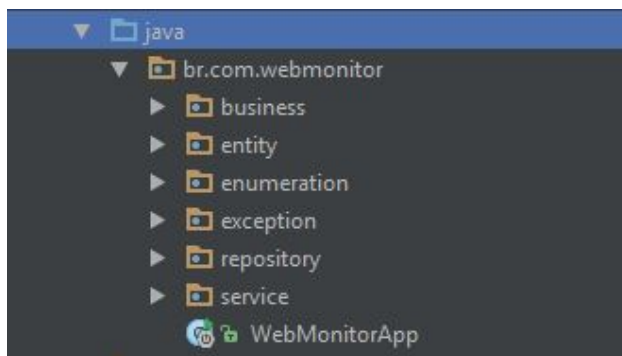


Figura 4.6: Estrutura de pastas do projeto.

Pasta	Descrição
business	Responsável pelas regras de negocio. Seção 4.1.4.
entity	Contem as entidades do sistema. Subseção 4.1.1.
enumeration	Contem as enum da aplicação.
exception	Contem as exception da aplicação.
repository	Interfaces para gerar os acessos ao banco de dados. Seção 4.1.4.
service	Responsáveis pelos serviços da aplicação. Seção 4.1.4.

Tabela 4.5: Descrição das pastas do projeto.

Repository

A pasta repository contem as interfaces de acesso ao banco de dados que são controladas pelo *Spring Data*. Todas essas interfaces devem estender da classe *JpaRepository* dessa forma diversos consultas são geradas automaticamente pelo *Spring Data*.

Na Figura 4.7 pode ser visto que cada classe da aplicação possuem uma interface repository seguindo o padrão de nomenclatura nome da classe mais Repository.

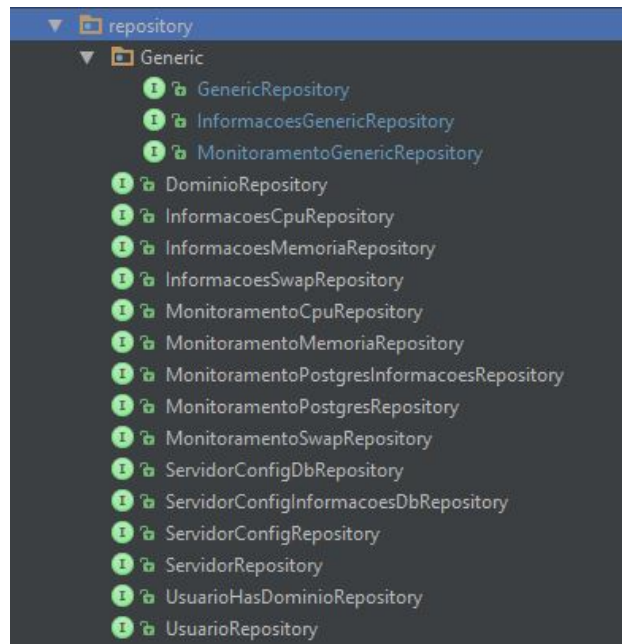


Figura 4.7: Pasta Repository.

Business

A pasta business é responsável pelas regras de negocio da aplicação. Cada entidade da aplicação contem uma classe business correspondente que segue o padrão de nomenclatura nome da classe mais BO, como pode ser visto na Figura 4.8.

Todas as classes da business estendem da classe GenericBO que contem os métodos genéricos de inserção e exclusão do banco de dados.

A GenericBO recebe um tipo genérico Entity que estender de GenericEntity informando que essa classe é uma classe da aplicação e um tipo genérico Repository que estende de JpaRepository o que informa que essa classe possui um repositório de acesso ao banco de dados controlado pelo *Spring Boot*. A classe GenericBO pode ser vista no Apêndice B.

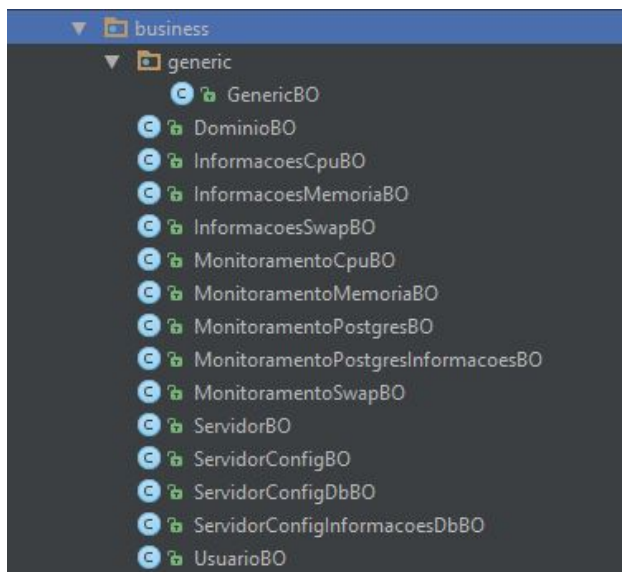


Figura 4.8: Pasta Business.

Service

A pasta Service é responsável por coordenar o serviço da aplicação. São elas que recebem as requisições HTTP e dizem o que será feito com cada requisição. Cada entidade da aplicação contém uma classe Service correspondente que segue o padrão de nomenclatura nome da classe mais Service, como pode ser visto na Figura 4.9.

Cada classe possui uma URL destinada a gerenciar seu recurso. As URLs de cada recurso podem ser vistas na ??

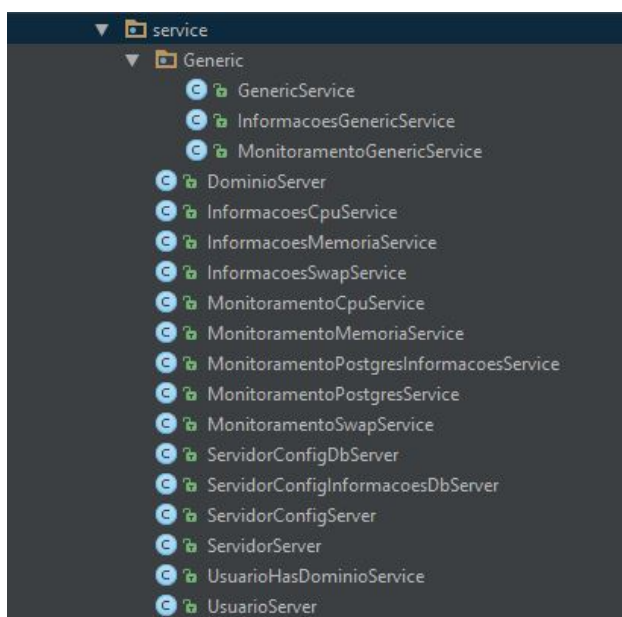


Figura 4.9: Pasta Service.

Classe	URL
Usuario	/usuario
Dominio	/dominio
Servidor	/servidor
InformacoesCpu	/servidor/{idServidor}/informacoescpu
InformacoesMemoria	/servidor/{idServidor}/informacoesModuleoria
InformacoesSwap	/servidor/{idServidor}/informacoesswap
MonitoramentoCpu	/servidor/informacoes/{idInformacoesCpu}/mon-itoramentocpu
MonitoramentoMemoria	/servidor/informacoes/{idInformacoesMemoria}-/monitoramentomemoria
MonitoramentoPostgresInformacoes	/servidor/informacoes/{idInformacoesPostgresIn- formacoes}monitoramentopostgresinformacoes
MonitoramentoPostgres	/servidor/informacoes/{idInformacoesPostgres}/- monitoramentopostgres
MonitoramentoSwap	/servidor/informacoes/{idInformacoesSwap}/mo- nitoramentoswap
ServidorConfigDb	/servidor/{idServidor}/servidorconfiguracoesdb
ServidorConfigInformacoesDb	/servidor/{idServidor}/servidorconfiguracoesinfo- rmacoesdb
ServidorConfig	/servidor/{idServidor}/servidorconfiguracoes

Tabela 4.6: Variáveis da classe ServidorConfigInformacoesDb e suas descrições.

4.1.5 Consumindo serviços do MonitorWeb-API

4.2 MonitorWeb-Cli

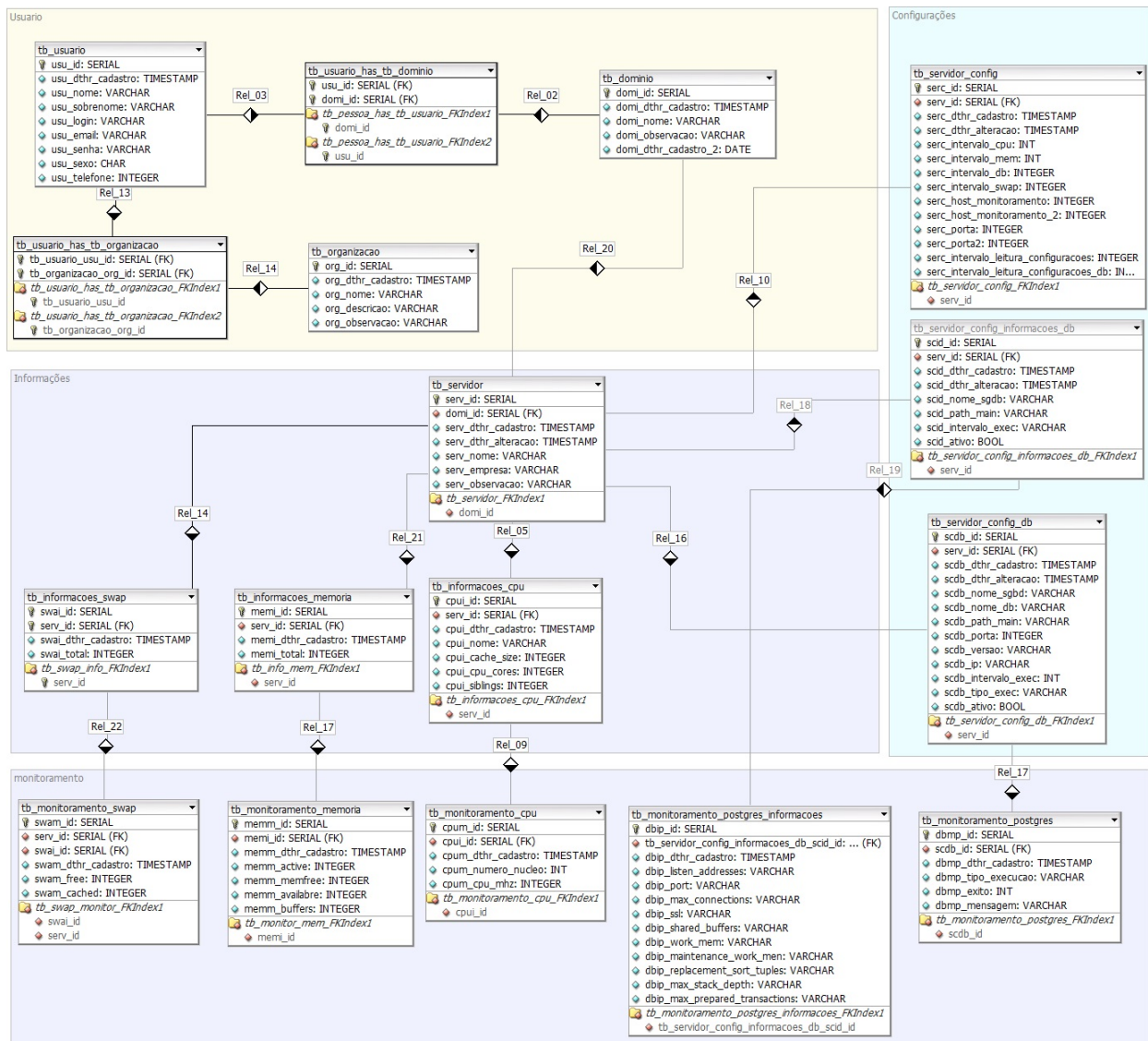


Figura 4.10: Pasta Entity

Capítulo 5

Resultados

Neste capítulo são apresentados os resultados dos testes realizados.

5.1 Desempenho da Biblioteca *FaultRecovery*

5.2 Desempenho e Eficiência da Classe TData

Capítulo 6

Conclusão

Bibliografia

- [1] ROSS, K. *Redes de computadores e a internet, uma abordagem top-down 5ª edição*. Pearson Education do Brasil, 2010.
- [2] TANENBAUM, A. S. *Redes de computadores 4ª edição*. Campus, 2003.
- [3] MARIMOTO, C. E. *Linux redes e servidores guia prático 2ª edição*. Sul Editores, 2011.
- [4] SAMPAIO, C. *Soa e web service em java*. Brasport Livros e Multimídia Ltda, 2003.
- [5] WEBER, T. S. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. Publicação online, 2002. Disponível em: < <http://www.inf.ufrgs.br/taisy/disciplinas/textos/Dependabilidade.pdf> > Acesso em: 29/03/2017.
- [6] TANENBAUM, A. S. *Sistemas operacionais modernos 3ª edição*. Pearson Education do Brasil, 2010.
- [7] GALVIN, A. S. G. G. P. B. *Sistemas operacionais conceitos e aplicações*. Campus, 2000.
- [8] LAUDO. Laudo aponta que incêndio no prédio do inss foi acidental. Publicação online, 2006. Disponível em: < <http://www1.folha.uol.com.br/folha/cotidiano/ult95u117284.shtml> > Acesso em: 29/03/2017.
- [9] MACHADO, N. Incêndio no inss destrói processos de fraudes. Publicação online, 2005. Disponível em: < <http://noticias.terra.com.br/brasil/interna/0,,OI810789-EI306,00.html> > Acesso em: 29/03/2017.
- [10] FUTEMA, F. Incêndio destruiu processos administrativos e logística do inss, diz deputado. Publicação online, 2005. Disponível em: < <http://www1.folha.uol.com.br/folha/cotidiano/ult95u116752.shtml> > Acesso em: 29/03/2017.
- [11] JORGE ABÍLIO ABINADER, R. D. L. *Web service em java*. Brasport livros e Multimídia Ltda, 2006.
- [12] W3C. Webservices framework e assertion exchange using saml. Publicação online, 2001. Disponível em: < <https://www.w3.org/2001/03/WSWS-popa/paper23> > Acesso em: 29/03/2017.

- [13] SAUDATE, A. *Rest - construa api's inteligentes de maneira simples*. Casa do Código, 2014.
- [14] BOAGRIO, F. *Spring boot - acelere o desenvolvimento de microserviços*. Casa do Código, 2017.
- [15] SAUDATE, A. *Soa - aplicado integrando com web services e além*. Casa do Código, 2012.
- [16] ANICHE, M. *Testes automatizados de software - um guia prático*. Casa do Código, 2015.
- [17] ANICHE, M. *Test-driven development - teste e design no mundo real*. Casa do Código, 2012.
- [18] SPRING.IO. Spring framework overview. Publicação online, 2017. Disponível em: < <https://docs.spring.io/spring/docs/5.0.2.BUILD-SNAPSHOT/spring-framework-reference/overview.html> > Acesso em: 23/11/2017.
- [19] SPRING.IO. Webservices framework e assertion exchange using saml. Publicação online, 2017. Disponível em: < <https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle> > Acesso em: 23/11/2017.
- [20] SPRING.IO. Spring boot test. Publicação online, 2017. Disponível em: < <https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html> > Acesso em: 25/11/2017.
- [21] JUNIT. Junit. Publicação online, 2017. Disponível em: < <http://junit.org/junit5/docs/current/user-guide> > Acesso em: 25/11/2017.
- [22] HAMCREST. hamcrest - tutorial.wiki. Publicação online, 2017. Disponível em: < <https://code.google.com/archive/p/hamcrest/wikis/Tutorial.wiki> > Acesso em: 25/11/2017.
- [23] MOCKITO. Tasty mocking framework for unit tests in java. Publicação online, 2017. Disponível em: < <http://site.mockito.org> > Acesso em: 25/11/2017.
- [24] JSONPATH. Jayway jsonpath. Publicação online, 2017. Disponível em: < <https://github.com/json-path/JsonPath> > Acesso em: 25/11/2017.
- [25] SPRING.IO. Spring data. Publicação online, 2017. Disponível em: < <https://projects.spring.io/spring-data> > Acesso em: 24/11/2017.
- [26] WEISSMANN, H. L. *Vire o jogo com spring framework*. Casa do Código, 2012.
- [27] SPRING.IO. Spring data jpa. Publicação online, 2017. Disponível em: < <https://projects.spring.io/spring-data-jpa> > Acesso em: 24/11/2017.
- [28] APACHE. Feature summary. Publicação online, 2017. Disponível em: < <https://maven.apache.org/maven-features.html> > Acesso em: 27/11/2017.
- [29] GROUP, T. P. G. D. Chapter 1. introduction. Publicação online, 2017. Disponível em: < <https://jdbc.postgresql.org/documentation/94/intro.html> > Acesso em: 27/11/2017.

Apêndice A

Anexos

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfig. (Gerado Automaticamente)
servidor	Indica com qual objeto Servidor esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
nome	Nome do processador.
cacheSize	Tamanho do cache do processador.
cpuCores	Quantos núcleos tem o processador.
siblings	Quantos núcleos virtuais tem o processador.

Tabela A.1: Variáveis da classe InformacoesCpu e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfig. (Gerado Automaticamente)
servidor	Indica com qual objeto Servidor esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
total	Quanto de memoria existe no servidor.

Tabela A.2: Variáveis da classe InformacoesMemoria e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfig. (Gerado Automaticamente)
servidor	Indica com qual objeto Servidor esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
total	Quanto de memoria swap existe no servidor.

Tabela A.3: Variáveis da classe InformacoesSwap e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfig. (Gerado Automaticamente)
informacoesCpu	Indica com qual objeto InformacoesCpu esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
coreId	Id do núcleo que esta sendo monitorado.
cpuMhz	Quantos Mhz (MegaHertz) esta sendo utilizado.

Tabela A.4: Variáveis da classe MonitoramentoCpu e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfig. (Gerado Automaticamente)
informacoesMemoria	Indica com qual objeto InformacoesMemoria esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
active	A quantidade total de buffer (Memoria Temporaria) ou memória cache que foi utilizada recentemente e não foi liberada.
memfree	A quantidade de memoria física não utilizada.
avaiabre	A quantidade de memoria que pode ser acessada pelo servidor.
buffers	A quantidade de memoria física utilizada para buffers de arquivos.

Tabela A.5: Variáveis da classe MonitoramentoMemoria e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfig. (Gerado Automaticamente)
InformacoesSwap	Indica com qual objeto InformacoesSwap esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
free	O montante total de swap livre.
cached	A quantidade de swap que esta sendo utilizada.

Tabela A.6: Variáveis da classe MonitoramentoSwap e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfigDb. (Gerado Automaticamente)
servidorConfigDb	Indica com qual objeto ServidorConfigDb esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
tipoExecucao	É uma enum do tipo EnumSgdbTipoExec o qual informa se o procedimento é um backup ou um vacuum.
exito	Informa se a procedimento foi realizado com sucesso.
mensagem	Mensagem gerada pelo procedimento.

Tabela A.7: Variáveis da classe MonitoramentoPostgres e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfigDb. (Gerado Automaticamente)
ServidorConfigInformacoesDb	Indica com qual objeto ServidorConfigInformacoesDb esse registro esta relacionado.
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
listenAddresses	De qual ip o postgresQL ira aceitar conexão.
port	Porta que o postgresQL esta rodando.
maxConnections	Numero máximo de conexão que o postgresQL ira aceitar simultaneamente.
ssl	Se a conexão com o banco possui ssl (Secure Socket Layer).
sharedBuffers	Quantidade de memória compartilhada utilizada pelo postgresQL.
tempBuffers	Quantidade de memória utilizada por sessão do postgresQL.
workMem	Quantidade de memória utilizada por operações de classificação interna e tabelas de hash antes de gravar em arquivos de disco.
maintenanceWorkMem	Quantidade máxima de memória a ser usada por operações de manutenção, como VACUUM , CREATE INDEX e ALTER TABLE ADD FOREIGN KEY.
maxStackDepth	Especifica a profundidade máxima da pilha de execução do postgresQL no servidor.
maxPreparedTransactions	Define o número máximo de transações que podem estar no estado "prepared".

Tabela A.8: Variáveis da classe ServidorConfigInformacoesDb e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfigDb. (Gerado Automaticamente)
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
nome	Nome do usuário.
sobrenome	Sobrenome do usuário.
login	Login do usuário.
email	Email do usuário.
senha	Senha de acesso do usuário.
sexo	Enum do tipo EnumSexo para informar o sexo do usuário.
telefone	Telefone para contato com o usuário.

Tabela A.9: Variáveis da classe Usuario e suas descrições.

Variável	Descrição
id	Número único para cada objeto do tipo ServidorConfigDb. (Gerado Automaticamente)
dthr_cadastro	Data e hora do cadastro. (Gerado Automaticamente)
nome	Nome do Dominio.
observacao	Observação para o dominio.

Tabela A.10: Variáveis da classe Dominio e suas descrições.

Apêndice B

Anexos

```
1 package br.com.webmonitor.entity.Generic;
2
3 import java.io.Serializable;
4 import java.util.Date;
5 import java.util.Objects;
6
7 /**
8  * Classe base para qualquer objeto serializável.
9  *
10 * @author Eduardo Balan
11 *
12 * @param <T> o tipo do atributo id
13 */
14 public abstract class GenericEntity<T extends Serializable> implements
15     Serializable {
16
17     private static final long serialVersionUID = 1L;
18
19     public abstract T getId();
20
21     public abstract void setId(T id);
22
23     public abstract Date getDthr_cadastro();
24
25     public abstract void setDthr_cadastro(Date date);
26
27     /**
28      * Indica quando outro objeto é igual a este. Nesta implementação, qualquer
29      * objeto derivado de Bean é igual a este desde que seja exatamente da mesma
30      * classe e tenha o mesmo ID.
31      *
32      * @author Kleber Kruger
33      *
34      * @param obj o objeto a comparar com este
```

```

32     * @return {@code true} se este objeto é igual ao do argumento; {@code false}
    caso contrário.
33     */
34     @Override
35     public boolean equals(Object obj) {
36         if (getId() != null && obj instanceof GenericEntity) {
37             GenericEntity x = (GenericEntity) obj;
38             return getClass() == x.getClass() && getId().equals(x.getId());
39         }
40         return super.equals(obj);
41     }
42
43     /**
44     * Retorna um valor de hash code para este objeto. Nesta implementação, este
    valor é gerado por
45     * uma combinação do hash code da classe (getClass().hashCode()) somado ao
    hash code do atributo
46     * id (id.hashCode()).
47     *
48     * @author Kleber Kruger
49     *
50     * @return um valor de hash code para este objeto
51     */
52     @Override
53     public int hashCode() {
54         if (getId() != null) {
55             return 43 * 7 + Objects.hashCode(getClass().hashCode() + getId().
    hashCode());
56         }
57         return super.hashCode();
58     }
59
60 }

```

Quadro 8: Entidade genérica da aplicação GenericEntity e suas funcionalidades.

```

1  package br.com.webmonitor.business.generic;
2
3  import br.com.webmonitor.entity.Generic.GenericEntity;
4  import br.com.webmonitor.exception.SqlGenericRuntimeException;
5  import br.com.webmonitor.exception.SqlInexistenteRuntimeException;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.data.jpa.repository.JpaRepository;
8
9
10 import javax.persistence.MappedSuperclass;
11 import java.util.Date;
12
13
14 /**

```



```
15  * Class GenericBO é a classe responsável por regras de negocio, genéricos e
    * simples como:
16  * Inserção em uma base de dados, remoção da base de dados, e update.
17  *
18  * @author Eduardo Balan
19  *
20  * @param Entity Entidade a qual ela ira prestar o servico.
21  * @param Repository Repositorio responsavel pela Entity que vc esta utilizando.
22  *
23  * @throws SqlInexistenteRuntimeException
24  * @throws SqlGenericRuntimeException
25  */
26  @MappedSuperclass
27  public class GenericBO <Entity extends GenericEntity, Repository extends
    JpaRepository<Entity, Long>> {
28
29      /* Repositorio responsavel pela Entity */
30      @Autowired
31      private Repository repository;
32
33      /**
34       * Metodo responsável pelas regras de negocio genéricas da inserção.
35       *
36       * @author Eduardo Balan
37       *
38       * @param Entity Entidade que sera persistida no banco de dados.
39       *
40       * @throws SqlGenericRuntimeException
41       *
42       * return Entity persistida no banco de dados.
43       */
44      public Entity inserir(Entity entityNova){
45          try{
46              entityNova.setDthr_cadastro(new Date());
47              return repository.save(entityNova);
48          }catch (Exception e){
49              throw new SqlGenericRuntimeException(e);
50          }
51      }
52
53      /**
54       * Metodo responsável pelas regras de negocio genéricas da exclusão.
55       *
56       * @author Eduardo Balan
57       *
58       * @param Long id da entidade que sera removida do banco de dados.
59       *
60       * @throws SqlInexistenteRuntimeException
61       * @throws SqlGenericRuntimeException
62       *
```

```
63     * return void.  
64     */  
65     public void excluir(Long idEntity){  
66         Entity entityPersistidaNoDB = repository.findOne(idEntity);  
67         if(entityPersistidaNoDB == null){  
68             throw new SqlInexistenteRuntimeException("Registro não encontrado na  
base de dados.", null);  
69         }  
70         try{  
71             repository.delete(idEntity);  
72         }catch (Exception e){  
73             throw new SqlGenericRuntimeException(e);  
74         }  
75     }  
76  
77 }  
78  
79 }
```

Quadro 9: Entidade genérica GenericBO e suas funcionalidades.