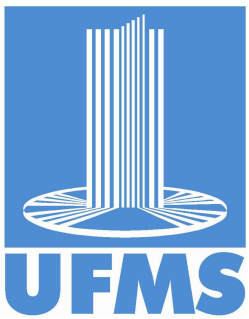
Trabalho de Conclusão de Curso

Monitoramento de servidores Linux por web sites.

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

Monitoramento de servidores Linux por web sites.

Coxim, 15 de dezembro de 2017.

Banca Examinadora:

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

Resumo

Abstract

Agradecimentos

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Quadros

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 (*Assymetrical 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 esse 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.

1.3 Organização da Proposta

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx No Capítulo 2 são apresentados os conceitos utilizados neste trabalho de acordo com a literatura estudada. Na Seção **??** explica-se os conceitos de falha, erro e defeito ou modelo de três universos. Na Seção **??** são descritas as principais fontes de radiação e seus efeitos nos circuitos eletrônicos. O conceito de "dependabilidade" é explicado na Seção **??** e na Seção **??** explica-se o conceito geral de tolerância a falhas e os atributos necessários para que uma falha seja definida. Na seção **??** são apresentadas as principais técnicas de tolerância a falhas e na seção **??** as principais técnicas de injeção de falhas.

As modificações realizadas nas bibliotecas e a criação da classe *TData* são exibidas no Capítulo 3, dividido em três seções. Na Seção **??** são apresentadas as implementações e as modificações realizadas na biblioteca *FaultInjector*. Na Seção **??** é exibida a extensão da biblioteca *FaultRecovery*. Na Seção **??** são exibidas as implementações realizadas para criação da classe *TData*, sua utilização é explicada mediante exemplos.

No Capítulo 5 são exibidos os resultados encontrados após os testes de tempo de execução e tolerância a falhas em que foram expostas as bibliotecas *FaultInjector*, *FaultRecovery* e a classe *TData*. No Capítulo 6 são exibidas as considerações finais deste trabalho. xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

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-

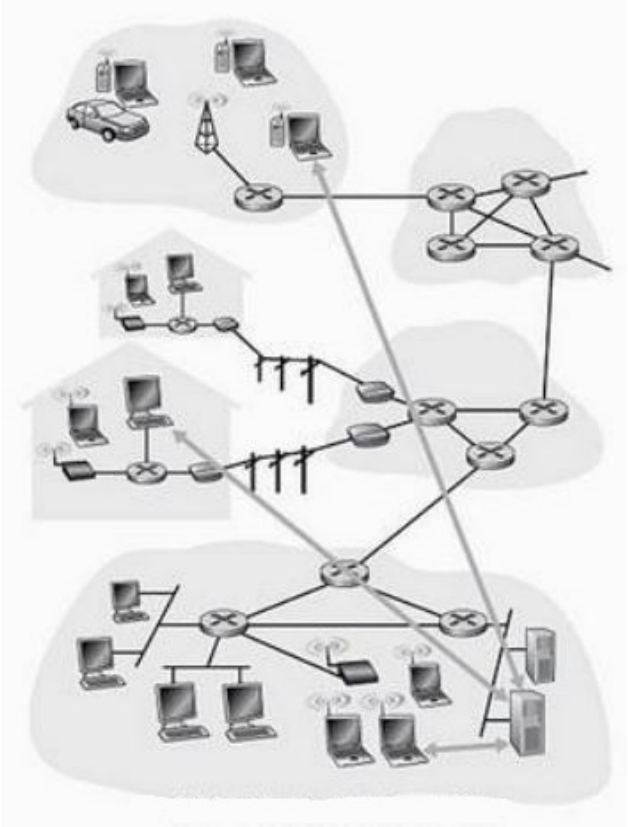


Figura 2.1: Representa a arquitetura clitente-Servidor, onde diversos tipos de cliente se comunicam com um mesmo servidor.

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 explicita 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 nenhum operação executada, mesmo que o sistema venha a falha.
- 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 salario de um funcionário e outro usuário tentar alterar o mesmo salario ao mesmo tempo uma transação só iniciara 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 *web* tem a forma de métodos GET.
HEAD	O método HEAD solicita apenas o cabeçalho da mensagem, sem a página propriamente dita. Essemétodo pode ser usado para se obter a data da última modificação feita na ágina, para reunirinformações 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 *web* 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 oservidor HTTP remoto esteja pronto para excluir a página, o arquivo subjacente pode ter um modoque 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. Essemétodo é útil quando as solicitações não estão sendo processadas corretamente e o cliente desejasaber 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 da 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 da 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 ira processar as informações, e retornara 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 modulos.

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, casso isso não ocorra, o teste deve indicar uma falha a ser reportada para o programado, 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 manira 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 minima 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 *donwload* 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
       projeto a dependência -->
11             <artifactId>spring-boot-starter</artifactId><!-- Nome da dependência
       -->
12             <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, uma desenvolvida em java que é um *web service* para a qual foi dado o nome de **MonitorWeb-Api**. A outra aplicação em C++ que ira rodar nos servidores linux cliente e tem o nome de **MonitorWeb-Cli**.

O MonitorWeb-Cli ira realizar leitura dos dados de seu hospedeiro tais como CPU(Central Processing Unit), memoria, banco dados e swap. Esse procedimento sera realizado de acordo com a configuração de tempo que o usuário desejar. Para cada leitura realizada o sistema realiza um 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 ele tenha acesso pela rede. Após efetuar um desses procedimento ele também envia uma mensagem para o MonitorWeb-Api que por sua vez armazena essas informações.

O MonitorWeb-Api é responsável por receber os dados de todos os MonitorWeb-Cli, e realizar a persistência no banco de dados. Ele também pode ser utilizado para disponibilizar esse dados para outras aplicações.

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 trafego na rede [13]. Para a persistência dos dados foi utilizado o banco de dados PostgreSQL.

O MonitorWeb-Api possui uma classe central chamado de Servidor. A partir do cadastro de um objeto dessa classe e o cadastro de um objeto da classe ServidorConfig e ambos estando relacionados, a aplicação já esta configurada para trabalhar com o MonitorWeb-Cli. Na Tabela 4.1 e na Tabela 4.2 estão as variáveis e descrição das classe Servidor e ServidorConfig respectivamente. Como realizar o cadastro desses recursos sera mostrado na

Subseção 4.1.5.

Variáveis	**Descrição**
id	Número único para cada objeto do tipo Servidor. (Gerado Automaticamente)
dominio	A qual domínio o servidor pertence (Não obrigatório)
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.

Variáveis	**Descrição**
id	Número único para cada objeto do tipo ServidorConfig. (Gerado Automaticamente)
servidor	Indica 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)
intervaloLeituraConfiguracoes	De quantos em quantos segundos o MonitorWeb-Cli deve ler esse arquivo de configurações, e se reconfigurar(Valor *default* 120 s).
intervaloLeituraConfiguracoesDb	De quantos em quantos segundos o MonitorWeb-Cli deve ler as configurações do banco de dados, e se reconfigurar(Valor *default* 120 s).
intervaloCpu	De quantos em quantos segundos o MonitorWeb-Cli deve enviar um registro da classe MonitoramentoCpu.(Valor *default* 1 s)
intervaloMemoria	De quantos em quantos segundos o MonitorWeb-Cli deve enviar um registro da classe MonitoramentoMemoria.(Valor *default* 1 s)
intervaloSwap	De quantos em quantos segundos o MonitorWeb-Cli deve enviar um registro da classe MonitoramentoSwap.(Valor *default* 1 s)
hostMonitoramento	IP do servidor principal do MonitorWeb-Api.
hostMonitoramento2	IP do servidor secundario do MonitorWeb-Api.
porta	Porta da aplicação no servidor principal do MonitorWeb-Api.
porta2	Porta da aplicação no servidor secundario do MonitorWeb-Api.

Tabela 4.2: Variáveis da classe ServidorConfig e suas descrições.

Um diagrama de classe resumido pode ser visto na Figura 4.1 contendo essas duas classes. A seguir uma listagem falando sobre cada bloco de cor do diagrama.

- No quadro amarelo (classes com prefixo Servidor) corresponde as classes que o usuário

tem que cadastrar para os recursos funcionarem. por exemplo para um monitoramento ocorrer um objeto da classe "ServidorConfig"deve ser criado e relacionado com o objeto da classe servidor em questão.

- No quadro rosa (classes com prefixo Informacoes) são classes que serão cadastradas automaticamente pela aplicação MonitorWeb-Cli e fazem referencia a os valores do servidor que são inseridos a cade vez que a aplicação é iniciada. Esses tipos de valores não tem como ser mudado sem que a maquina seja reiniciada. Um exemplos desses valor é o modelo do processador, ele nunca sera mudado sem que a maquina seja reiniciada.

- No quadro laranja (classes com prefixo Monitoramento) são classes que contem valores que precisam ser monitorados de tempos em tempo. Os intervalos de tempo do monitoramento das classe MonitoramentoCpu, MonitoramentoMemoria e MonitoramentoSwap são configurado através do objeto "ServidorConfig"como pode ser visto pela Tabela 4.2. Exemplos dos valores armazenados por objetos dessa classe são quanto de memoria esta sendo utilizado e quantos Mhz cada núcleo do processador esta utilizado.

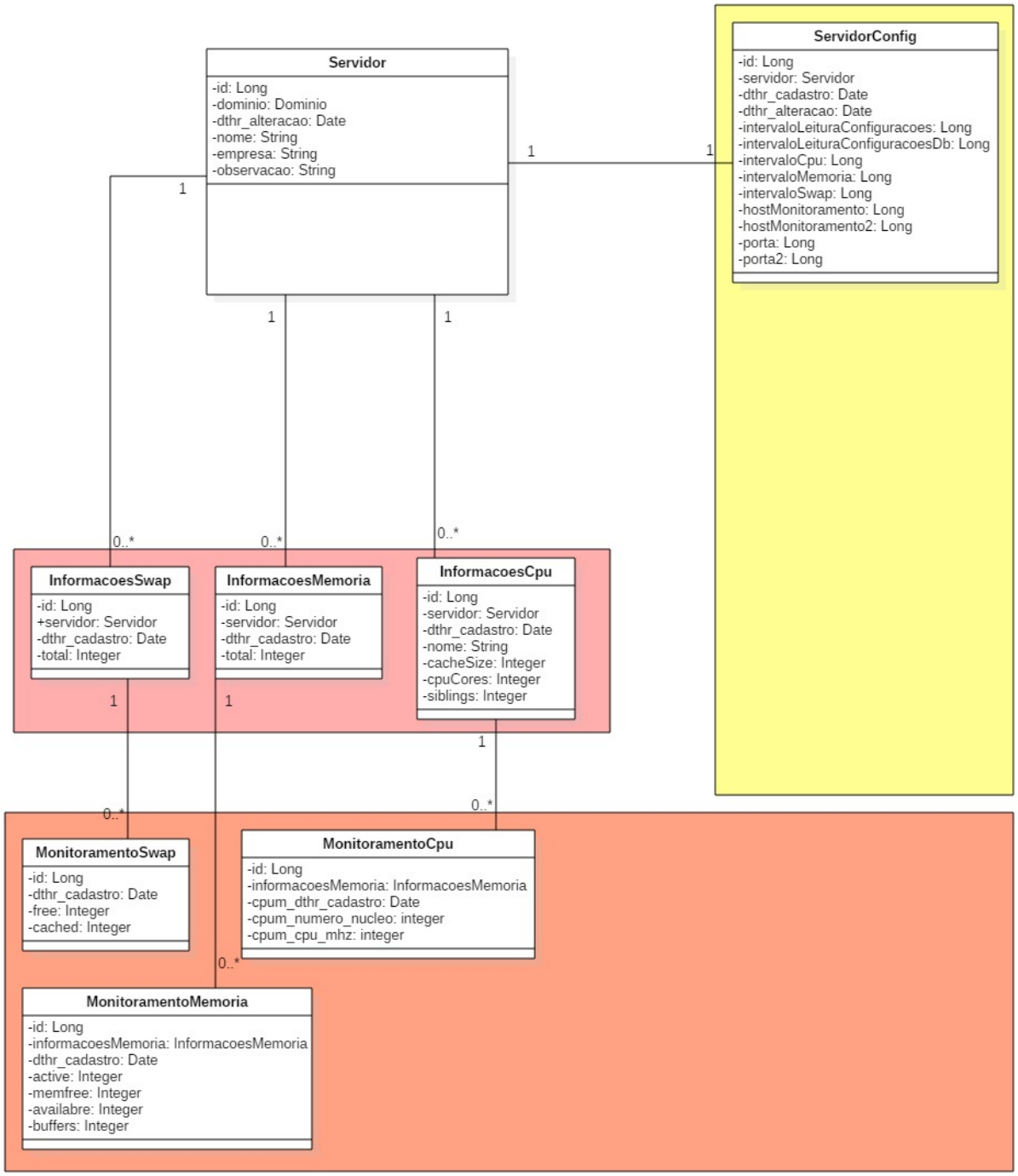


Figura 4.1: Diagrama entidade relacional bem resumido, separados pelas cores amarelo, rosa e laranja que correspondem respectivamente a Configurações que o usuário deve fazer, informações geradas pelo MonitorWeb-Api e monitoramentos gerados pelo MonitorWeb-Api

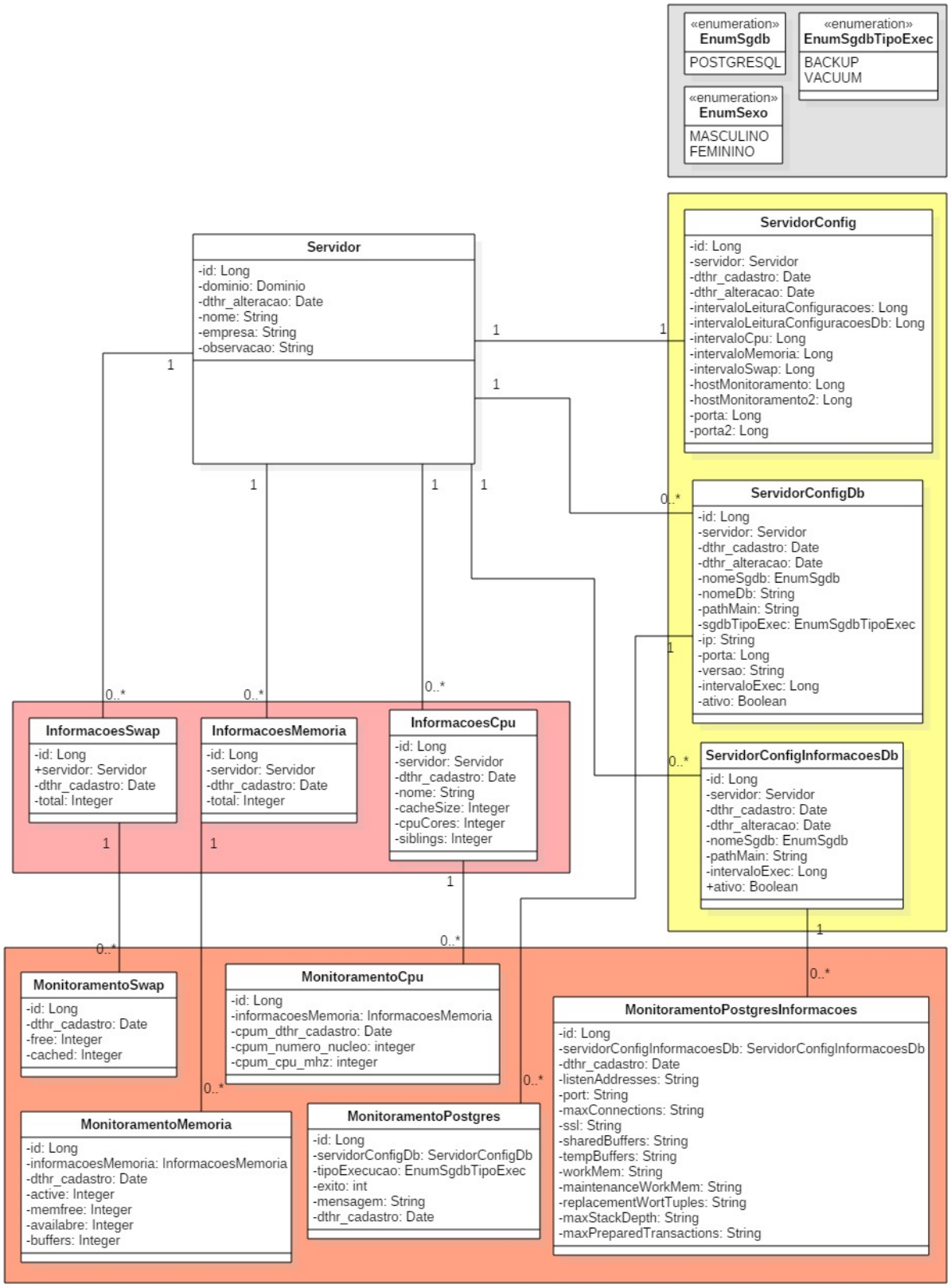


Figura 4.2: Pasta DiagramaDeClass.

4.1.1 Configurações Apache Maven

Como visto na Seção 3.2 o Apache Maven é um gerenciador de pacotes, que gerenciá 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 dependencias necessarias para o seu funcionamento [27].
- Quadro 6, linha 15 - spring-boot-starter-web, Essa dependência identifica que nossa aplicação sera uma aplicação *web*, rodara usando tomcat, utilizara 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.

4.1.2 Configurações application.properties

O spring 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].

- 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 nivel do spring tera 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.

4.1.3 Estrutura do Projeto

O projeto foi dividido em seis pastas que podem ser visto na Figura 4.4, e uma classe chama WebMonitorApp.class, a qual é responsável por iniciar a aplicação. A Tabela 4.3 temos o nome das pastas e suas descrição.

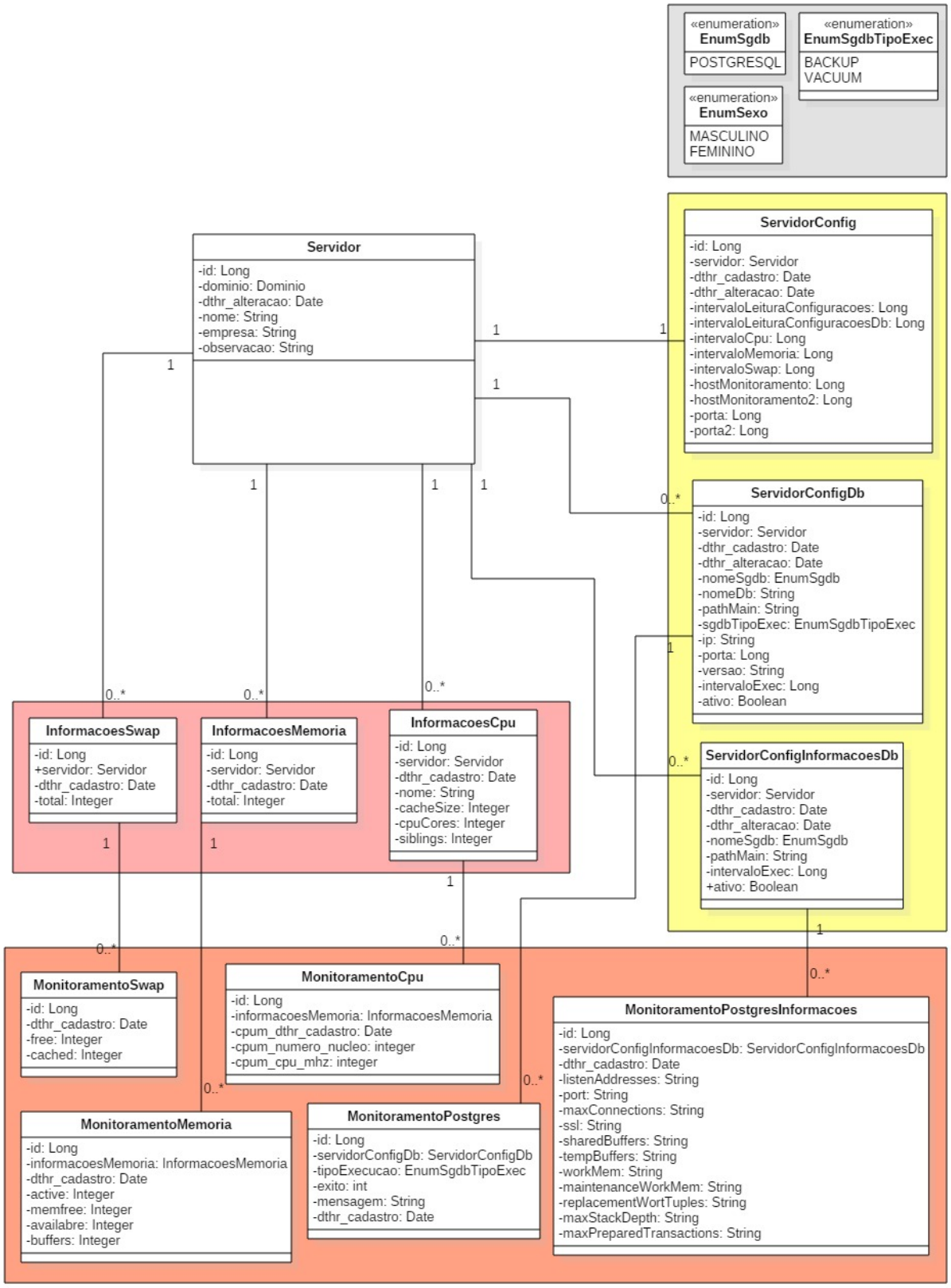


Figura 4.3: Pasta DiagramaDeClass.

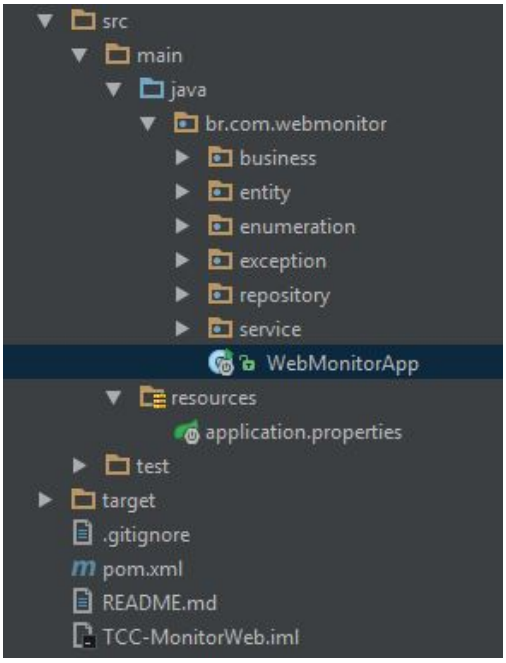


Figura 4.4: Estrutura de pastas do projeto.

Pasta	Descrição
business	Responsavel pelas regras de negocio.
entity	Entidades do sistema.
enumeration	Enum.
exception	Exception.
repository	Interfaces para gerar os acessos ao banco de dados.
service	Responsáveis pelos serviços da aplicação.

Tabela 4.3: Descrição das pastas do projeto.

4.1.4 Pasta entity

A paste entity é responsável pelas entidades da aplicação, e sua estrutura pode ser vistas na Figura 4.6. Os arquivos possuem três iniciais muito repetidas em seus nomes que são "Informacoes*", "Monitoramento*"e "ServidorConfig*"e cada uma delas possui um significado:

As classes que iniciam com a palavra "Informações", contem valores que só precisam ser armazenados uma única vez quando a aplicação é iniciada. Esses tipos de valores não tem como ser mudado sem que a maquina seja reiniciada e o sistema de monitoramento cliente reinicie. Exemplos desses valores são, modelo do processador, numero de núcleos do processador e quantidade de memoria.

As classes que iniciam com a palavra "Monitoramento*"contem valores que precisam ser armazenados de tempos em tempo, como quanto de memoria esta sendo utilizado, se o banco de dados executou o backup corretamente, quantos Mhz cada nucleo do processador esta endo utilizado.

As classes que iniciam com a palavra "ServidorConfig*"contem valores que o usuário deve configurar, como IP da aplicação, porta, tempo de intervalo de cada monitoramento.

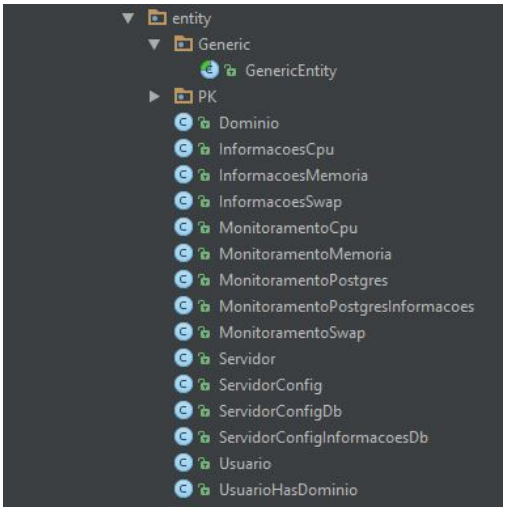


Figura 4.5: Pasta Entity.

classe GenericEntity

Todas as entidades da aplicação estendem da classe abstrata GenericEntity que por sua vez recebe uma 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 Quadro 8

```
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
        Serializable {
15
16     private static final long serialVersionUID = 1L;
17
18     public abstract T getId();
19
20     public abstract void setId(T id);
21
22     public abstract Date getDthr_cadastro();
```

```
23
24      public abstract void setDthr_cadastro(Date date);
25
26      /**
27       * Indica quando outro objeto é igual a este. Nesta implementação, qualquer
        objeto derivado de Bean é igual a este desde que seja exatamente da mesma
        classe e tenha o mesmo ID.
28       *
29       * @author Kleber Kruger
30       *
31       * @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.

4.1.5 Comsumindo recursos do MonitorWeb-Api

4.2 MonitorWeb-Cli

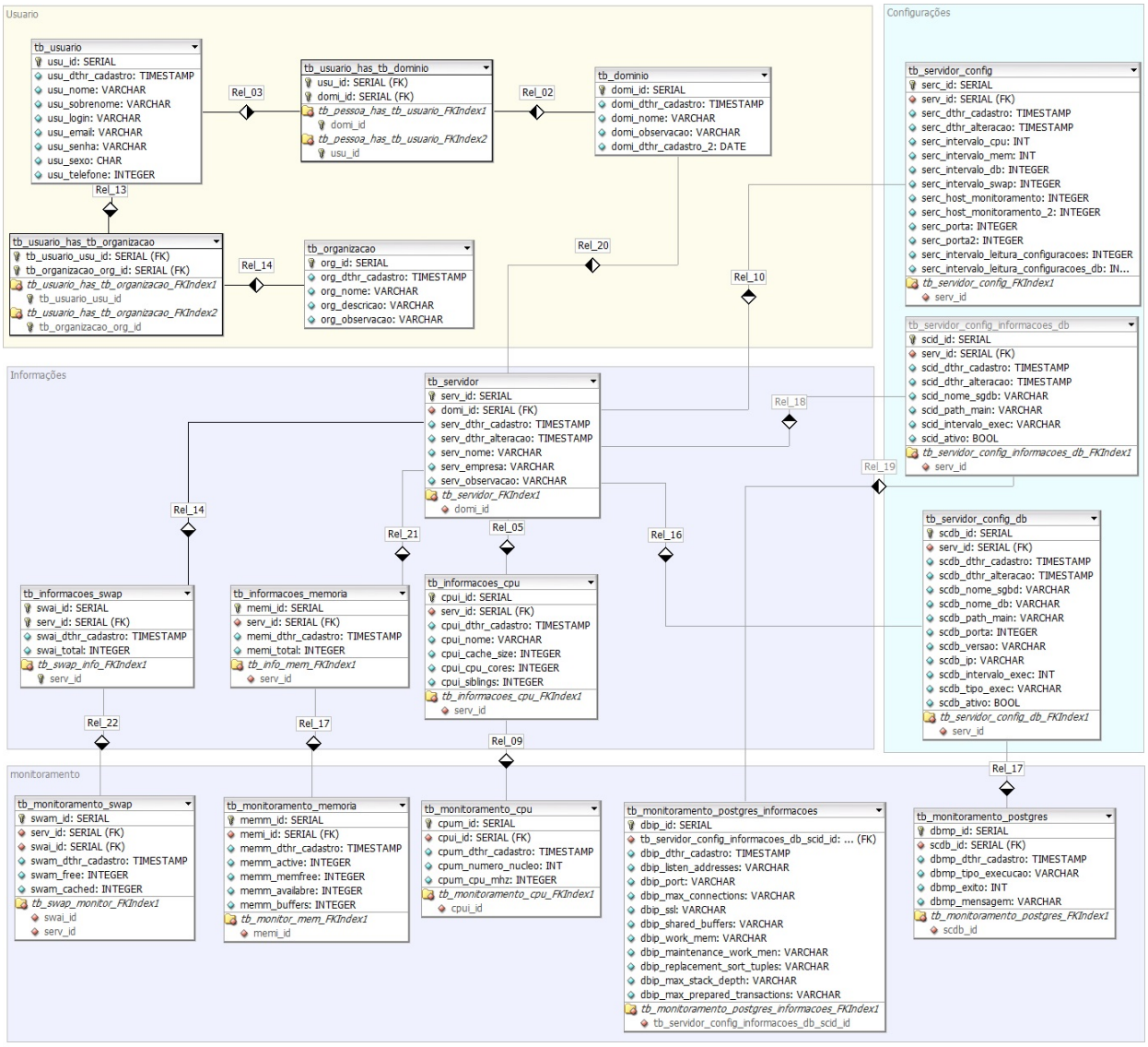


Figura 4.6: Pasta Entityaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.

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 5º edição.* Pearson Education do Brasil, 2010.

[2] TANENBAUM, A. S. *Redes de computadores 4º edição.* Campus, 2003.

[3] MARIMOTO, C. E. *Linux redes e servidores guia prático 2 º 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 3º 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 Multimidia 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 microsserviços.* 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