UNIVERSIDADE FEDERAL DE SANTA CATARINA

PARALELISMO DA EXECUÇÃO DE TESTES AUTOMATIZADOS

Rodrigo Aguiar Costa

Florianópolis - SC
2019/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

PARALELISMO DA EXECUÇÃO DE TESTES AUTOMATIZADOS

Rodrigo Aguiar Costa

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Florianópolis - SC

2019/1

Rodrigo Aguiar Costa

PARALELISMO DA EXECUÇÃO DE TESTES AUTOMATIZADOS

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientador: Prof. Dr. Raul Sidnei Wazlawick

Banca examinadora

Profª. Drª. Patrícia Vilain

Fernando Luís Amorim Agostinho

Sumário

LISTA DE FIGURAS

RESUMO

Os testes automatizados fazem parte de uma das principais formas para realização dos testes de um *software*, com objetivo de avaliar a qualidade deste. Quando os testes são executados junto à interface gráfica, estas execuções, por vezes, tornam-se demoradas, aumentando o tempo que se dará o início da análise dos resultados após a execução dos testes. Realizar paralelismo da execução dos testes é uma forma de reduzir este tempo de resposta, entre o início da execução e o início da análise dos resultados. Este trabalho apresenta um estudo sobre o paralelismo da execução de testes automatizados baseado na estrutura dos testes automatizados realizados no laboratório Bridge. Será demonstrada uma possível alternativa para a execução de testes automatizados em paralelo, a fim de trazer maior agilidade e eficiência. Com a utilização de máquinas virtuais e *containers* foi possível reduzir em 66% o tempo de execução dos testes.

Palavras-chave: Testes, Qualidade de software, Automatização, Paralelismo.

1 INTRODUÇÃO

A metodologia de pesquisa deste trabalho baseia-se na metodologia pesquisa-ação, do inglês *research action*, apesar de ser uma metodologia pouco utilizada no contexto de Engenharia de Software, como afirma Medeiros (2009), possui potencial no estudo na área.

Medeiros (2009) define a pesquisa-ação como sendo uma metodologia com o objetivo de alinhar os conceitos educacionais junto a realidade operacional. Sendo realizado através de uma busca contínua pela melhoria, alinhado a um processo de estudo através das ações realizadas, ponderando constantemente estas ações. Também demonstrando uma análise das ações, mesmo que subjetivas, tomadas pelo pesquisador no contexto do cotidiano.

Assim, metodologia pesquisa-ação se mostrou mais adequada para a realização deste trabalho, após dificuldades encontradas pelo autor, na aplicação de outras metodologias mais convencionais. Com isso, foi utilizado um modelo adaptado, proposto por Medeiros (2009), como base estrutural deste trabalho.

Sendo assim o estudo foi realizado com base nos testes de sistema da aplicação CDS Offline, desenvolvido pelo laboratório Bridge. Podendo ser adaptado para outros tipos de testes.

Com isso, os capítulos a seguir encontram-se divididos da seguinte forma: no Capítulo 2, "Diagnóstico", é subdividido em "Descrição do Problema", "Contexto do Trabalho" e "Tema de Pesquisa". O Capítulo 3, "Planejamento", subdivide-se em "Revisão da Literatura", "Enfoque da Ação", "Hipóteses" e "Definições Operacionais". No Capítulo 4, "Ações", demonstra-se, de forma detalhada, as atividades realizadas durante a pesquisa. No Capítulo 5, "Avaliação e Análise", são

descritos os processos de análises e avaliações realizadas. E por fim no Capítulo 6, "Reflexões e Aprendizado", são realizadas as devidas reflexões, enfatizando o aprendizado obtido de acordo com as ações realizadas.

2 DIAGNÓSTICO

O diagnóstico está dividido em três seções: "Descrição do Problema", "Contexto do Trabalho" e "Tema de Pesquisa".

2.1 Descrição do Problema

Durante o desenvolvimento de aplicações *web* também são desenvolvidos testes automatizados. Entre esses testes, existem os que se utilizam da automação de navegadores, como o Selenium WebDriver, que auxiliam na realização da manutenção e das validações da aplicação que está sendo desenvolvida.

Bruns (2009) afirma que a realização de testes que interagem com a interface *web* no navegador tendem a ser mais lentos que os testes realizados diretamente no código da aplicação, como no caso dos testes de unidade. Com o aumento do número de casos de testes automatizados, o tempo da execução aumenta consideravelmente, podendo ser proporcional à quantidade de passos e verificações que são realizadas na interface da aplicação. Uma forma de melhorar o tempo de execução, como Bruns (2009) também apresenta, seria reduzir os testes em pequenos módulos, onde estes poderiam ser executados de forma paralela em máquinas diferentes.

Berner (2005) retrata que, além do foco na automação dos testes, deve também atentar-se a melhorar os processos de análises e reportes de falhas, e também dos procedimentos de instalação e configuração de ambientes para a realização dos testes. A redução do tempo da execução dos testes, e sua rápida análise, ajuda na rápida detecção de possíveis problemas na aplicação. Com constantes evoluções no código, os testes automatizados podem rapidamente realizar uma verificação de que

nada inesperado foi alterado no sistema desenvolvido, melhorando a qualidade da aplicação.

Para a rápida execução dos testes automatizados, podemos adotar diversas estratégias. Manualmente podemos separar os módulos que serão testados e realizar a execução de cada um deles em diferentes máquinas. Outra forma seria separar os testes considerados mais críticos e executá-los primeiramente, entregando o resultado dos testes de forma parcial. Porém, esta última opção se torna incompleta, deliberando ao analista de teste a responsabilidade por decidir quais testes são de maior impacto no sistema, prejudicando a cobertura completa da aplicação.

A realização da virtualização mostra-se um método interessante para testes de performance, como proposto por Kim (2013), ou, para realizar a verificação nos diferentes sistemas operacionais suportados pela aplicação.

Quando se deseja realizar execução de apenas uma aplicação, a virtualização completa de um sistema operacional pode gerar custos computacionais acima do esperado. Para minimizar o consumo de recursos, utiliza-se *containers*[1] para a execução da aplicação que, segundo Dua (2014), fornecem uma alternativa de isolar os processos da aplicação do restante do sistema. Apesar de não ser uma técnica nova, ela vem ganhando um destaque cada vez maior pela facilidade que se encontra, atualmente, em alguns projetos do mercado, como o projeto Docker.

Para o projeto de testes, onde este estudo foi conduzido, utiliza-se casos de testes automatizados com base na aplicação "CDS Offline". À medida que o número de testes do sistema cresce, o tempo para sua execução também aumenta consideravelmente. Para melhorar esta situação, esta pesquisa procura encontrar

1 https://www.docker.com/what-docker. Acessado em 14/06/2017.

uma melhor solução para execução dos testes de forma paralela, a fim de reduzir o tempo total da execução, com o menor consumo de recursos possíveis, fazendo assim, com que o processo da análise dos resultados se inicie o quanto antes.

2.2 Contexto do Trabalho

O contexto deste trabalho trata do estudo e aplicação de métodos existentes para a paralelização da execução dos testes automatizados da aplicação CDS Offline, a fim de reduzir o tempo total da execução de todos os testes automatizados existentes, que são executados a cada versão gerada. Este trabalho envolve o setor de qualidade do laboratório Bridge, mais especificamente, a área de testes automatizados do laboratório.

Como afirma Karhu (2009), a automatização dos testes aumenta a eficiência diminuindo o tempo de execução dos mesmos cenários, quando comparado com os testes manuais. Para o desenvolvimento de uma aplicação, o tempo é um dos fatores mais importantes dentro do seu planejamento. Com isso, reduzir o tempo consumindo o mínimo de recursos disponíveis, torna-se um desafio cada vez maior.

Para a implementação da aplicação é utilizada a tecnologia Java, com o uso das ferramentas JUnit e Selenium WebDriver.

A fim de não tornar este estudo muito vasto e complexo, algumas partes do processo de trabalho são omitidas e/ou alteradas para melhor análise e interpretação dos dados para o contexto geral deste trabalho.

Os testes automatizados são desenvolvidos de forma que suas execuções são realizadas em sequência. A execução dos testes depende da inicialização completa do sistema, sendo a aplicação disponibilizada no site do Portal do Departamento de

Atenção Básica, onde, durante a realização deste trabalho, encontra-se na versão 3.0.13. Os testes são construídos a partir da documentação do sistema.

Como limitação deste trabalho, e melhor fidelidade aos testes implementados no laboratório, as execuções dos casos de testes foram realizadas em um ambiente onde não existiu mais de uma instalação da aplicação CDS Offline no sistema operacional.

2.3 Tema de Pesquisa

De acordo com o que foi contextualizado nas seções anteriores, o tema de pesquisa deste trabalho é a implementação e avaliação da paralelização da execução de testes automatizados, com a utilização das ferramentas Selenium WebDriver, JUnit e Docker. Assim, este trabalho tem como objetivo reduzir o tempo da execução dos testes, utilizando técnicas de virtualização avaliando também o uso de recursos computacionais.

3 PLANEJAMENTO

Inicia-se nesta seção a fase do planejamento, onde é realizada a revisão da literatura relacionada ao tema de pesquisa proposto. Nas subseções seguintes são demonstrados o enfoque da ação, a definição das hipóteses e, por fim, as definições operacionais, onde serão explanadas as ferramentas utilizadas na realização deste trabalho, de acordo com a metodologia de pesquisa-ação.

3.1 Conceitos

Nas seções subsequentes serão detalhados alguns níveis de testes funcionais, como testes de sistema, testes de aceitação e testes de regressão.

3.1.1 Testes de sistema

Teste de sistema é um dos níveis de testes, onde seu objetivo está relacionado ao comportamento de uma aplicação como um todo, como é citado no SWEBOK (2004).

Segundo Rätzmann (2003), teste de sistema possui relação com questões relacionadas ao sistema por completo, onde todo sistema deve satisfazer as necessidades planejadas.

3.1.2 Testes de aceitação

Summervile (2011) afirma que antes da liberação de uma aplicação, como parte final do processo de testes, deve ser realizado os testes de aceitação. Em que o sistema deva ser testado com dados reais, podendo transparecer problemas de performance como também problemas de análise dos requisitos. Segundo Crispin (2006), teste de aceitação não interage com os objetos de programação de maneira direta, mas

com diferentes partes do código ao mesmo tempo. Ao realizar a alteração em apenas um objeto do sistema, esta mudança poderá impactar em diversos testes de aceitação, que não estão necessariamente relacionados de forma direta entre si. Este tipo de teste normalmente é utilizado por ferramentas específicas ou interfaces, para que possa ser testado e visualizado por um usuário.

Para a realização dos testes de aceitação é necessário que todo o código da aplicação esteja integrado e utilizável. Pois, segundo Crispin (2006), diferentemente dos testes de unidade, que utilizam pequenas partes do código para sua execução, os testes de aceitação necessitam que as diversas partes do código estejam trabalhando em conjunto.

3.1.3 Testes de regressão

Como é citado no SWEBOK (2004), o teste de regressão é uma forma de verificar que as modificações, realizadas na aplicação, não causaram mudanças inesperadas, ou seja, a repetição dos testes que haviam sido executados antes deve mostrar que não ocorreram mudanças de comportamentos na aplicação.

Para Rätzmann (2003), os testes de regressão, têm como principal característica, assegurar que todas as funcionalidades que estão corretas, não sofram com alterações no sistema, e que correções e novas funcionalidades realizadas na aplicação, não causem problemas com futuras mudanças no sistema.

3.2 Revisão da Literatura

Nesta seção será realizado uma revisão da literatura apontando conceitos sobre paralelismo de testes e testes automatizados.

3.2.1 Paralelismo

Segundo Xiao-ping (2005), testes paralelos têm como objetivo melhorar a estratégia utilizada para a execução dos testes. De modo geral, testes paralelos servem para que as disponibilidades de recursos estejam sempre ocupadas durante as execuções. Como a publicação em *National Instruments*[2] (2017) exemplifica, testes paralelos implicam testar, ao mesmo tempo, diversos produtos ou componentes.

Chillarege (1999) afirma, que os testes automatizados têm por objetivo diminuir o trabalho manual e melhorar a cobertura da aplicação com um maior número de testes. Entre alguns casos encontrados sobre paralelismo de testes está o Selenium Grid.

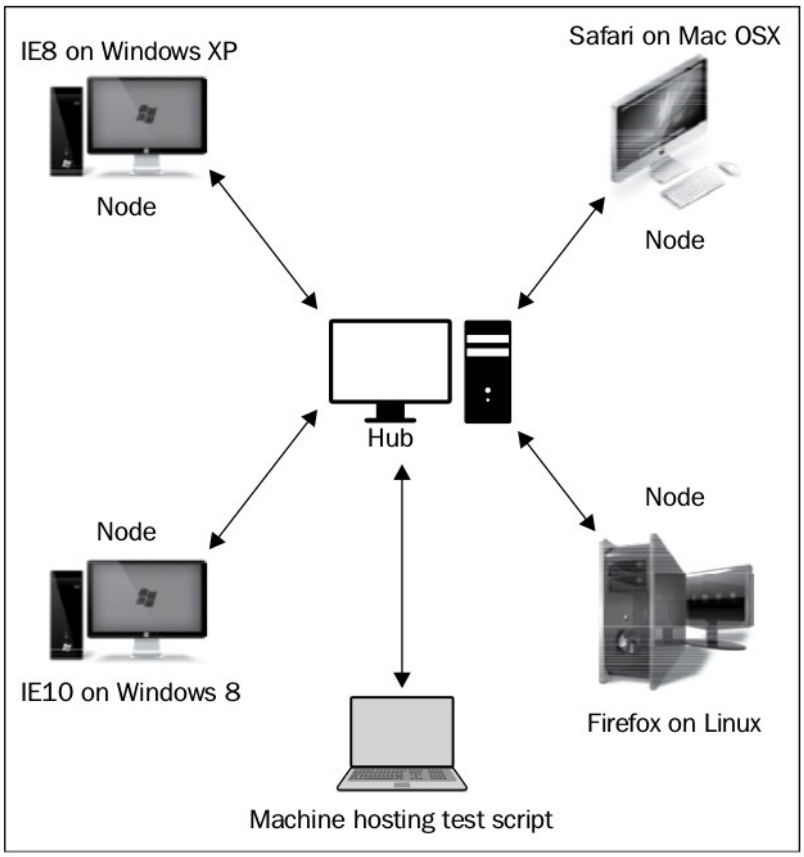

Figura 1 – Exemplificação de Selenium Grid (Avasarala, 2014)

Avasarala (2014) define que “Selenium Grid é uma infraestrutura de teste para

2 http://www.ni.com/white-paper/3043/en/ Benefits of Parallel Testing - National Instruments. Publish Date: Mar 21, 2017. Acessado em 27/06/2017.

diferentes plataformas onde seus testes podem ser executados, sendo gerenciados a partir de um ponto central". Como é demonstrado na Figura 1, os testes são designados a diferentes nodos, através de um *hub*, onde esses nodos podem ser de diferentes versões de sistemas operacionais e, ou, navegadores, fazendo assim com que os testes sejam distribuídos entre os sistemas desejados, retornando o resultado dos testes à máquina *host*.

3.2.2 Testes automatizados

Para Chillarege (1999), o princípio básico que envolve a execução dos testes automatizados, é que quanto maior o número de casos de testes, maior será a cobertura realizada com o menor esforço, considerando o mesmo procedimento dos testes quando realizado manualmente.

No que diz respeito a testes de software, para Torkar (2009), testes automatizados possuem a mesma finalidade de qualquer outro tipo de teste e que os processos deverão ser automatizados até certo ponto, para que não envolva esforço excessivo.

É observado, por Karhu (2009), que testes automatizados podem reduzir os custos e melhorar a qualidade, pois diferentes testes serão executados em um menor tempo com uma melhor cobertura. Entretanto isto gera novos custos, como tempo de implementação, manutenção de código do teste implementado e treinamento de pessoas para a criação dos testes automatizados.

Um outro fator que se pode levar em conta, segundo Torkar (2009), é que os recursos humanos para esta área têm custos relativamente altos quando comparados com custos dos recursos computacionais disponíveis hoje. Por este motivo, inúmeras tarefas tornam-se interessantes quando automatizadas.

Segundo Karhu (2009), algumas tarefas de testes são difíceis de automatizar, gerando um grande esforço, especialmente quando é exigido um conhecimento elevado por parte do desenvolvedor de teste.

Conforme Rätzmann e Young (2003), para a realização dos testes automatizados, não se faz necessário o desenvolvimento, ou aquisição, de ferramentas caras. Existe uma diversidade de ferramentas, sendo elas disponíveis tanto de forma comercial quanto de forma livre, seja paga ou gratuita[3]. Por vezes, de acordo com Rätzmann (2003), as ferramentas utilizadas possuem suas próprias linguagens de programação para criação dos testes, porém também é possível encontrar ferramentas que disponibilizam seu uso através das linguagens de programação já conhecidas e consolidadas no mercado.

Para Graham (2012), entretanto, a ferramenta a ser escolhida deve ser aquela que melhor se adapta à estrutura e necessidade do projeto que está sendo desenvolvido. A geração dos testes automatizados é de responsabilidade do programador, por este já possuir um conhecimento de programação.

Rätzmann e Young (2003) citam cinco características a respeito das quais se deve ficar atento quando se tem em mente a implementação de testes automatizados. São elas:

- *Testes automatizados devem ser aprendidos.* O tempo em que se usa para o aprendizado sobre como desenvolver testes automatizados deve ser levado em consideração, pois o esforço inicial é consideravelmente alto e custoso.
- *Ferramentas de testes não são substitutos de testadores.* Ferramentas possuem um papel importante no panorama da engenharia de software.

3 http://www.softwaretestingclass.com/software-testing-tools-list/ Acessado em 18/10/2016.

Porém, elas são apenas eficazes nas mãos de testadores treinados e experientes. Somente analistas de testes podem realizar avaliações de risco e planejamento dos testes de forma realmente eficiente bem como a análise de seus resultados.

- *Nem tudo pode ser testado, mesmo com automatização.* Para todo módulo de teste, deve-se realizar um planejamento adequado com a análise dos riscos e o custo. Por vezes, certos tipos de testes tornam-se muito custosos, ou inviáveis para sua realização. Com uma boa análise, pode-se evitar um esforço desnecessário, focando em pontos mais relevantes.
- *Testes automatizados devem estar em constante atualização.* O término de um módulo de casos de testes de uma aplicação, não significa que este módulo será deixado de lado. Os testes devem estar em constante atualização, acompanhando o desenvolvimento da aplicação que está sendo testada. Mesmo com a falha na execução posterior do teste, deve-se descobrir as reais causas para uma correta análise dos resultados, onde o tempo de análise pode ser consideravelmente alto, devido à necessidade de reconstruir o cenário que envolveu a falha e fazer a correta análise do ocorrido.
- *Testes automatizados custam tempo e dinheiro.* Mesmo que para a realização de um teste manual dure um tempo relativamente baixo, sua automatização pode levar um tempo muitas vezes maior que o teste realizado manualmente, devido sua complexidade, ou, necessidade de preparação de um ambiente adequado.

Para Graham (2012), o objetivo da automação de testes se difere dos objetivos de

testes, sendo a automação uma forma de como os testes serão executados. Por exemplo, caso esteja automatizando testes de regressão, mas se deseja encontrar novos erros, o objetivo dos testes pode estar errado e não sua automação pois, testes de regressão tem o objetivo de assegurar que nenhuma alteração recente no sistema trouxe novos problemas.

Segundo Rätzmann (2003), durante a realização dos primeiros casos de teste, geralmente encontram-se mais erros do que os relatados com as execuções posteriores. Isso se deve ao fato que, com a criação dos testes, são realizadas as análises das melhores alternativas e opções disponíveis para a execução dos testes.

Por isso, para Graham (2014), é importante existir clareza do objetivo da automatização dos testes. Por exemplo, se a execução dos testes não encontra novos problemas na aplicação, e o objetivo é o teste de regressão, isso significa que se atingiu o objetivo esperado.

Segundo Amaricai (2014), ao desenvolver um *framework* de testes automatizados, deve-se ter em mente que este tipo de aplicação se equipara ao desenvolvimento de um software, como outro qualquer. Esse deverá ser bem planejado e estruturado, contendo inúmeras funções que servirão de auxílio e poderão interagir com outras aplicações para atingir seu objetivo de realizar as verificações das aplicações que se pretende testar.

Para Berner (2015), a automatização dos processos também pode fazer parte da estratégia na realização dos testes automatizados. A utilização de ferramentas, tanto para a criação dos testes, como para os reportes de falhas, pode tornar todo processo mais eficiente.

Segundo observações realizadas por Berner, Weber e Keller (2015), à medida que

casos de testes automatizados são executados sem a intervenção humana, a relação entre custo e eficácia aumenta. Para eles, o início da realização dos testes automatizados deve ser pelo caminho que dará maior retorno, e quem cumpre essa função, geralmente, são testes de componentes, testes de integração e os *smoke tests*. Este último é definido como um grupo reduzido de testes que cobrem as principais funcionalidades de uma aplicação, ignorando aspectos mais específicos do sistema[4].

3.3 Enfoque da Ação

3.3.1 Objetivos

O objetivo geral deste trabalho é demonstrar uma forma de realizar o paralelismo da execução de testes automatizados, com a utilização de máquinas virtuais, *containers*, Selenium WebDriver, JUnit, entre outras ferramentas disponibilizadas no mercado.

Como objetivos específicos, são demonstrados alguns possíveis métodos para a realização do paralelismo de testes automatizados, como a criação de um cliente/servidor. Além de responder as questões de pesquisa detalhadas a seguir.

3.3.2 Questões de pesquisa

Para alcançar o objetivo proposto, deverão ser respondidas as seguintes questões:

- Questão 1: *A realização do paralelismo trouxe maior complexidade à inicialização da execução dos testes automatizados?* Esta questão tem por objetivo, responder se a realização de testes em paralelo não influencia a construção, não dificulta a inicialização da execução e posterior análise dos

4 https://glossary.istqb.org/en/search/smoke acessado em 04/06/2019

resultados dos testes executados.

- Questão 2: *Houve melhora na análise dos resultados dos testes automatizados?* Esta questão tem por objetivo responder se ocorreu uma melhora significativa na análise dos testes automatizados após sua execução, agilizando a posterior análise dos resultados obtidos para o reporte dos possíveis erros ocorridos na última versão disponibilizada do sistema.

Para realizar melhor a análise do resultado esperado, subdividiu-se as questões anteriores. Para a Questão 1 temos:

- Q1.1 Quais modificações foram realizadas para criação dos testes automatizados individualmente?
- Q1.2 Houve significativa mudança na inicialização entre uma execução linear e uma execução paralela?

Para responder estas questões se realizará a análise do código alterado e uma comparação entre a execução linear e paralela.

Para a Questão 2 temos:

- Q2.1 Qual a diferença do tempo entre uma execução linear e uma execução paralela?
- Q2.2 Quais os recursos utilizados na execução paralela?
- Q2.3 Houve mudança na forma da análise dos resultados dos testes?

Para responder estas questões, serão realizadas algumas execuções de forma linear e paralela, e serão capturados os dados relevantes.

3.4 Hipóteses

O paralelismo da execução dos testes automatizados deve tornar a execução mais rápida, sem alteração na forma que eles são codificados os testes, fazendo com que a análise dos resultados dos testes seja realizada mais cedo.

A inicialização do processo de execução dos testes poderá se tornar mais complexa devido à criação de mais de uma instância da aplicação e da necessidade de distribuição dos testes para cada execução paralela.

3.5 Tecnologias Utilizadas

Nesta seção serão explanadas as técnicas e ferramentas que auxiliarão no desenvolvimento deste trabalho. Definindo-as e mostrando sua aplicabilidade.

3.5.1 Virtualização

Uma definição disponibilizada pela *IBM Global Education* (2007) descreve que virtualização pode ser considerada uma técnica para mascarar características físicas dos recursos computacionais, que são utilizados por outros sistemas, aplicações ou usuários. Kim (2013) diz que a virtualização é a transformação de, por exemplo, uma plataforma de *hardware* ou sistema operacional, em uma versão virtual.

Ainda segundo Kim (2013), existem diferentes propósitos para a virtualização, que pode ser dividida em categorias diferentes como, por exemplo, virtualização de servidores, de ambiente de trabalho, de armazenamento e de aplicações específicas. Para uma unidade lógica, não faz diferença quais, ou quantos, são os dispositivos físicos, pois ela somente enxerga os recursos que foram disponibilizados. Entre as principais vantagens da utilização da virtualização está o aumento da eficiência, quando usada de forma consciente.

Para se evitar o desperdício dos recursos computacionais e melhorar o gerenciamento desses recursos, a virtualização é uma alternativa interessante. Porém, recursos humanos especializados podem fazer-se necessário para um bom aproveitamento e estruturação destas máquinas virtuais.

Kim (2013) ressalta que para se determinar a quantidade de recursos computacionais que serão necessários para uma determinada aplicação, existem algumas alternativas, entretanto, a forma mais simples é a observação. Em uma máquina real, por exemplo, é observado a quantidade de recursos que determinada aplicação utiliza, entre elas, espaço de armazenamento, memória consumida, consumo de processamento, entre outros.

Rätzmann (2003) afirma que, ao se levar em conta a execução dos inúmeros tipos de testes, desde funcionalidades básicas até comportamentos específicos, passando por diferentes ambientes, a utilização de máquinas virtuais são uma forma interessante para tais tipos de teste. Com elas, a praticidade e o controle tornam-se facilitadas.

Segundo Graham (2012), quando a aplicação é portável para vários ambientes, como sistemas operacionais e/ou navegadores de páginas *web*, por exemplo, os testes automatizados também devem ser. Alterando, em certos casos, apenas variáveis de ambiente do sistema. Porém, a essência do teste, que é testar o *software* se mantém, deve funcionar da mesma forma em qualquer outro ambiente. Para facilitar a execução e análise dos testes automatizados, ambientes virtuais são uma forma interessante de se explorar.

3.5.2 Containers

Para Dua (2014), um *container* é um sistema operacional minimalista, que é

executado em cima de um sistema hospedeiro. Xavier (2013) diz que diferentemente da virtualização padrão, no qual se tem um sistema convidado rodando por completo em cima do sistema hospedeiro, os *containers* operam em nível de sistema operacional, não realizando a virtualização de *hardware* ou emulação de instruções, rodando nativamente instruções do processador. *Containers,* também reduzem o consumo dos recursos em comparação à virtualização.

Segundo Xavier (2013), uma das formas mais populares têm sido a técnica de virtualização *hypervisor*, sendo este um sistema hospedeiro de monitoramento de máquinas virtuais. Como demonstrado na Figura 2, *containers* apenas isolam as dependências e processos dentro das suas imagens, acima de uma camada de virtualização em nível de sistema operacional. Enquanto máquinas virtuais consistem em uma completa virtualização de um sistema operacional convidado, onde são executados os processos.

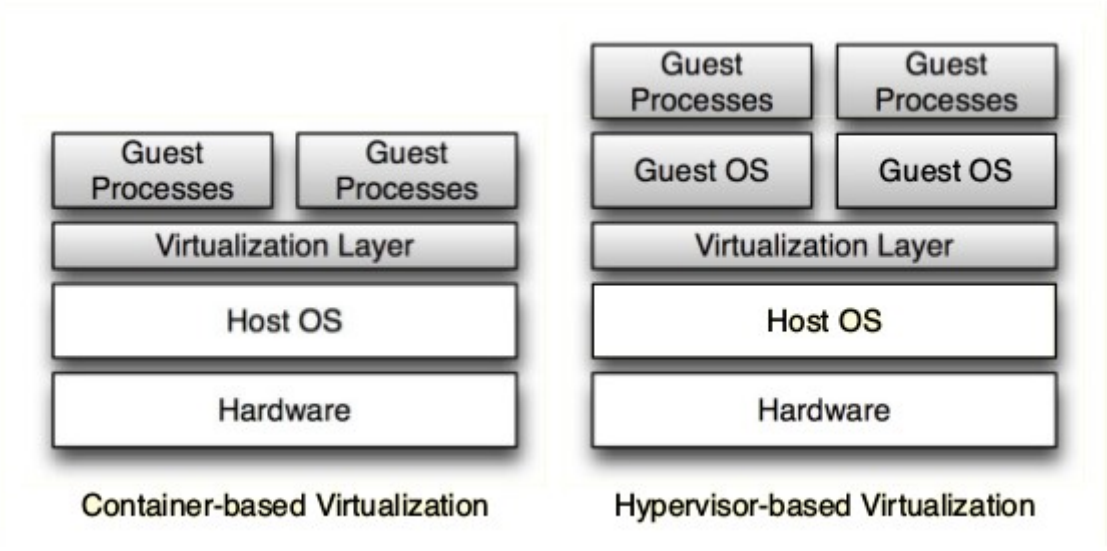

Figura 2 – *Container versus Hypervisor* (Xavier, 2013)

De acordo com Dua (2014), entre algumas das principais diferenças existentes entre as máquinas virtuais e os *containers,* está a performance. Enquanto, nas máquinas virtuais existem um excesso de consumo de recursos e suas instruções de

máquinas são traduzidas para o sistema hospedeiro, os *containers* possuem um desempenho mais próximo do sistema nativo. O espaço do armazenamento ocupado por uma máquina virtual é equivalente a um sistema operacional completo, somado aos programas que nele são instalados, enquanto um *container* possui uma parcela de ocupação extremamente menor, já que a base do sistema operacional é compartilhada.

3.5.3 Docker e c*ontainers*

Docker é uma plataforma de *container* de software, segundo a própria definição disponibilizada no site oficial do Docker[5].

Como Vitalino (2016) afirma, *container* pode ser definido como um conjunto de dependências com a aplicação que se deseja executar, confinados dentro de um pacote, onde este pacote contém somente as funcionalidades necessárias para execução desta aplicação. Apesar de serem similares às máquinas virtuais, os *containers* diferenciam-se por não possuir todo um sistema operacional dentro de sua imagem, além de compartilharem o *Kernel* do sistema operacional.

Entre as ferramentas disponibilizadas pelo Docker está o *compose*[6], que serve para criação e utilização de *containers.* Com esta ferramenta é possível configurar os serviços através de um arquivo de configuração YAML[7], um tipo de arquivo utilizado, essencialmente, para configurações de aplicações.

3.5.4 Selenium WebDriver

Conforme Leotta (2013), testes em ambientes *web*, tendem a possuir situações em execução em que ocorrem falhas com maior facilidade, o que ocasiona um atraso,

5 https://www.docker.com/what-docker. Acessado em 14/06/2017.
6 https://docs.docker.com/compose/overview/ Acessado em 30/04/2019.
7 https://yaml.org/about.html Acessado em 30/04/2019.

trabalho de manutenção ou reexecução, seja devido a mudanças de *layouts*, ou, até mesmo por causa do tempo de carregamento de uma página.

Para Leotta (2013), Selenium WebDriver é um *framework* que pode auxiliar na execução dos testes automatizados. Com ele podem ser implementados métodos para localizar os elementos da interface do usuário.

Para Angmo (2014), Selenium WebDriver torna mais fácil a ação de encontrar elementos na página *web*, o que reduz o esforço de manutenção quando há mudança de posição de elementos em uma página *web*.

Como consta na documentação[8] do Selenium WebDriver, este foi desenvolvido para dar melhor suporte às páginas dinâmicas, em que os elementos da página podem ser alterados sem que a mesma tenha sido atualizada, onde o WebDriver tem o objetivo de oferecer suporte a API's orientadas a objeto.

3.5.5 Java RMI

Java RMI, ou *Java Remote Method Invocation*, segundo a documentação Java[9], tem como objetivo permitir que um dado objeto que esteja sendo executado em uma determinada máquina virtual Java possa invocar outro objeto que esteja rodando em outra máquina virtual Java, utilizando-se de características cliente/servidor para carregar e executar processos pré-definidos em tempo de execução.

Aplicações RMI normalmente consistem em aplicações clientes/servidor onde, segundo definição da Oracle[9], o servidor cria um objeto remoto, tornando-o acessível, e espera por invocações deste objeto por algum cliente.

8 http://www.seleniumhq.org/docs/03_webdriver.jsp Acessado em 20/06/2017.
9 https://docs.oracle.com/javase/tutorial/rmi/. Acessado em 14/06/2017.

4 AÇÕES

Nesta seção, serão descritas as ações e atividades desempenhadas na construção do paralelismo da execução dos testes.

4.1 Primeiras Ações

As primeiras ações definiram o escopo do sistema no qual seriam realizados os testes automatizados. Inicialmente, foi cogitada a realização de testes sobre o sistema e-SUS AB, porém, devido à complexidade do sistema para realização deste trabalho, optou-se por realizar testes apenas no sistema de CDS Offline, considerando a simplicidade de instalação, inicialização do sistema e configuração inicial para realização dos testes.

Também, para efeitos de simplificação de configurações e estruturação do projeto, optou-se por definir alguns pré-requisitos para a realização dos testes, como a importação de um CNES (Cadastro Nacional de Estabelecimentos de Saúde) pré-definido para a realização dos testes, assim como a alteração no arquivo de configuração do JBoss, standalone.xml, permitindo mais de uma conexão no banco de dados, já que a utilização padrão do sistema somente uma conexão é permitida.

Como base para criação dos testes automatizados, foi utilizado uma simplificação do *framework* aplicada no laboratório Bridge. As classes de teste estendem uma classe abstrata, chamada CasoDeTeste, onde obrigatoriamente devem implementar o método *asserts.* Este método deve possuir somente verificações, ou ações que não alterem o estado atual do sistema. Os métodos *passos* e *passosAposAsserts* são opcionais, e possuem a finalidade de realizar ações no sistema. O método *passos* é executado antes do método *asserts*, para definir onde serão realizadas as

verificações. O método *passosAposAsserts* é executado depois do método *asserts*, realizando ações posteriores.

Os métodos *passos* e *passosAposAsserts* de um teste podem ser reutilizados como um fluxo de execução por outros testes. Assim, é possível reaproveitar as ações realizadas pelos testes com menor replicação de código.

4.2 Criação de Cliente/Servidor

Definido isto, foram utilizados aproximadamente 170 casos de testes para dar início à pesquisa. Após a realização de um estudo inicial, o modelo cliente/servidor mostrou-se mais apropriado ao objeto inicial, onde o servidor possui todos os testes e os distribui para os clientes realizarem requisições de execução. E ao final da execução, os clientes devolvem ao servidor as informações de execução do teste.

A solução encontrada para realizar esta tarefa, com a menor interferência no *framework* já estabelecido no laboratório, foi a ferramenta Java RMI, onde um objeto de uma máquina virtual Java específica possui a permissão de invocar métodos de um objeto que esteja rodando em outra máquina virtual, como especificado na documentação[10] disponibilizada pela Oracle. Mais informações foram descritas na seção Ferramentas.

Os serviços oferecidos pelo servidor são criados a partir de uma interface. Esta, por sua vez, deve estender a interface *Remote* do Java RMI. Como exemplificado no trecho de código demonstrado na Figura 3.

```
public interface RMIServices extends Remote {
    TestExecInfo getTesteParaExecucao() throws RemoteException;
    void setResultadoDaExecucao(TestExecInfo testResult) throws RemoteException;
```

10 https://docs.oracle.com/javase/tutorial/rmi/. Acessado em 14/06/2017.

```
}
```

Figura 3 – Interface implementada pelo servidor

Com a implementação desta interface, o servidor é criado. Ao instanciá-lo é definido o *hostname* e é criado o registro no *localhost,* definindo a porta por onde serão aceitas as requisições do cliente. Antes de iniciar o servidor, é preciso adicionar as suítes de testes que se deseja executar. Os testes inseridos ficam armazenados em um *buffer* no servidor, onde após a chamada do método *startServer*, demonstrado na Figura 4, o mesmo fica no aguardo à espera de uma requisição de um cliente.

```
public void addSuite(Class<?> suiteClass) {
    SuiteClasses suiteClasses = suiteClass.getAnnotation(SuiteClasses.class);
    for (Class<?> testClass : suiteClasses.value()) {
        if (testClass.isAnnotationPresent(SuiteClasses.class)) {
            this.addSuite(testClass);
            continue;
        }
        this.buffer.add(new TestExecInfo(suiteClass, testClass));
    }
}

public void startServer() {
    try {
        this.registry.bind(TEST_BUFFER, UnicastRemoteObject.exportObject(this, 0));
        this.totalTests = this.buffer.size();
       System.out.println(new Date().toString() + " - BUFFER INICIADO - Total de testes = " +
this.buffer.size());
        this.serverInitTimeMillis = System.currentTimeMillis();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (this.buffer.isEmpty()) {
            this.endServer();
        }
    }
}
```

Figura 4 – Trecho de código do servidor

Para a criação do cliente, é necessário passar como parâmetro o *host* do servidor para realizar o registro adequadamente. Através do *LocateRegistry*[11], é possível obter a referência do registro passando *host* e a porta, no caso é utilizada a porta padrão. Assim, o cliente pode obter a referência do servidor com os serviços

11 https://docs.oracle.com/javase/7/docs/api/java/rmi/registry/LocateRegistry.html Acessado em 23/04/2019.

disponíveis, como demonstrado na Figura 5.

```
public RMIExecClient(String host) {
    try {
            Registry registry = LocateRegistry.getRegistry(host);
            this.stub = (RMIServices) registry.lookup(RMIServices.TEST_BUFFER);
    } catch (NotBoundException | RemoteException e) {
        e.printStackTrace();
    }
}
```

Figura 5 – Instanciação do cliente

Ficou definido que, enquanto existirem testes a serem executados, o cliente ficará requisitando testes ao servidor. Ou seja, enquanto o servidor retornar testes ao cliente, este permanecerá executando, como demonstrado na Figura 6.

```
public void start() {
        while (true) {
                TestExecInfo rmiRequest = this.getTestExecInfo();
                if (rmiRequest == null) {
                        break;
                }
                this.executeSequence(rmiRequest);
        }
}
```

Figura 6 – Inicialização do cliente

Assim o método *getTestExecInfo* no cliente realiza uma chamada no servidor através do serviço *getTesteParaExecucao*. A implementação deste método no servidor, como mostrado na Figura 7, possuí um bloco de sincronização para controlar a concorrência caso exista mais de uma chamada simultaneamente, retornando o teste ao cliente. Caso o *buffer* contendo os testes esteja vazio o servidor é encerrado.

```
@Override
public TestExecInfo getTesteParaExecucao() throws RemoteException {
        synchronized (this) {
                if (!this.buffer.isEmpty()) {
                        System.out.println("BUFFER total = " + this.buffer.size());
                        return this.buffer.removeFirst();
                }
        }
```

```
        this.endServer();
        return null;
}
```

Figura 7 – Serviço implementado pelo servidor

Ao final da execução do teste, como mostrado na Figura 8, o cliente enviará o resultado da execução ao servidor em uma nova *thread*, não precisando esperar o envio dos resultados para requisitar novos testes ao servidor.

```
private void executeSequence(TestExecInfo rmiRequest) {

        if (rmiRequest.getTestCase() != null) {
                Result result = new ExecRequest(rmiRequest.getTestCase()).exec();
                rmiRequest.setResult(result);
                this.sendResult(rmiRequest);
        }
}

private void sendResult(final TestExecInfo testExecInfo) {
        this.storeData(testExecInfo);
        final RMIServices stubAux = this.stub;
        // criado nova thread para thread principal não esperar o envio e continuar
        new Thread(new Runnable() {
                @Override
                public void run() {
                        try {
                                stubAux.setResultadoDaExecucao(testExecInfo);
                        } catch (RemoteException e) {
                                this.run();
                                e.printStackTrace();
                        }
                }
        }, testExecInfo.getTestCase().getSimpleName()).start();
}
```

Figura 8 – Execução e envio dos testes pelo cliente

4.3 Criação dos *Containers*

Neste ponto, o servidor e o cliente estão implementados. Iniciou-se o estudo da execução da aplicação CDS Offline, tal como, da execução dos testes em *containers* docker. Visto que, se as aplicações rodam em *containers,* elas podem rodar em qualquer sistema que tenha suporte ao docker, por exemplo. Suportando,

evidentemente, as configurações mínimas de *hardware* da aplicação. Deste modo, o custo para criação de máquinas seria reduzido.

A partir deste ponto, a imagem é criada contendo a aplicação CDS Offline. Optou-se por realizar a instalação em um *container* docker a partir de uma máquina local. Após a imagem da instalação da aplicação ser criada, as demais máquinas utilizarão a mesma imagem. A aplicação CDS Offline tem como um dos pré-requisitos a utilização de Java 7. Então, foi decidido utilizar como base para utilização do sistema, a imagem *openjdk:7u181-jdk-jessie*, disponibilizada no site Docker Hub[12].

Para criação da imagem é utilizado o arquivo Dockerfile[13], onde é possível construir imagens automaticamente com o uso de instruções.

```
FROM openjdk:7u181-jdk-jessie

RUN apt-get update \
 && apt-get install -y xvfb \
 && rm -rf /var/lib/apt/lists/* \
 && rm -rf /src/*.deb

ENV HOME /home/cdsoffline
ENV ESUS_INSTALL Instalador-eSUS-AB-CDS-3.0.13-Linux_17102018
ENV DISPLAY :1

ADD ${ESUS_INSTALL}.zip ${HOME}/
ADD start.sh ${HOME}/

RUN groupadd -r cdsoffline \
 && useradd --no-log-init -r -g cdsoffline cdsoffline \
 && mkdir ${HOME}/${ESUS_INSTALL} \
 && chown -R cdsoffline:cdsoffline /home/cdsoffline \
 && chmod +x ${HOME}/start.sh

USER cdsoffline
WORKDIR ${HOME}

VOLUME ["${HOME}/${ESUS_INSTALL}"]

CMD ["/bin/bash", "-c", "${HOME}/start.sh"]
```

Figura 9 – Dockerfile da aplicação

12 https://hub.docker.com/_/openjdk Acessado em 20/04/2019.
13 https://docs.docker.com/engine/reference/builder/ Acessado em 23/04/2019.

Como demonstrado na Figura 9, na criação desta imagem será instalado o programa Xvfb. Com ele é possível iniciar uma tela virtual, para poder iniciar a aplicação CDS Offline. Também são adicionadas variáveis de ambientes, um arquivo compactado contendo a aplicação, um *script bash* que rodará ao iniciar o *container*, além da definição de um usuário, pois, a aplicação não deve ser rodada como superusuário.

Para realizar a execução dos testes, foi definido a criação de uma outra imagem, demonstrado na Figura 10, contendo uma instalação do navegador Chrome e uma aplicação *jar*, que executará os testes, bem como, um *script bash*, que será executado quando o *container* iniciar.

```
FROM openjdk:7u181-jdk-jessie

RUN apt-get update \
&& apt-get install -y \
  xvfb \
  apt-transport-https \
  ca-certificates \
  --no-install-recommends \
&& curl -sSL https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add - \
&& echo "deb [arch=amd64] https://dl.google.com/linux/chrome/deb/ stable main" > /etc/apt/
sources.list.d/google.list \
&& apt-get update \
&& apt-get install -y \
 google-chrome-stable \
 --no-install-recommends \
&& rm -rf /var/lib/apt/lists/* \
&& rm -rf /src/*.deb

ENV DISPLAY :1

ADD br.ufsc.meutcc-0.0.1-SNAPSHOT.jar /
ADD start.sh /

CMD ["/bin/bash", "/start.sh"]
```

Figura 10 – Dockerfile do cliente

Para realizar a execução em dois *containers* distintos, foi necessária a realização de mais uma alteração no arquivo standalone.xml do JBoss, que não permitia o acesso remoto à aplicação. E, também, foi adicionado um novo arquivo, *test.properties*

(Figura 11), no *resource* do projeto de testes, para definir a localização do banco de dados e ip do servidor de teste.

```
test.serverhost=192.168.0.10
test.databasepath=/home/cdsoffline/Instalador-eSUS-AB-CDS-3.0.13-Linux_17102018
```

Figura 11 – Aquivo *test.properties*

Para facilitar o início das imagens criadas da aplicação e execução dos testes, foi criado o arquivo *docker-compose.yml*, demonstrado na Figura 12. Para a utilização deste arquivo, também é necessário a instalação do *docker-compose*.

```
version: "2"

services:
  cdsoffline:
    image: cdsoffline
    volumes:
      - /home/cdsoffline/Instalador-eSUS-AB-CDS-3.0.13-Linux_17102018

  testes:
    image: chrome
    depends_on:
      - cdsoffline
    volumes_from:
      - cdsoffline
```

Figura 12 – Arquivo docker-compose

Outro motivo levado em consideração para a utilização do docker-compose foi a simplicidade da criação de um determinado volume de um *container* em outro. Assim, o acesso ao banco de dados foi contornado facilmente.

4.4 Execuções

Definido a criação dos *containers*, iniciou-se os testes em máquinas virtuais. Foram disponibilizadas um total de quatro máquinas de mesma configuração.

Após criadas, as imagens foram compartilhadas entre as máquinas que executarão os testes. Com isso, ao iniciar a execução destes *containers,* a aplicação CDS

Offline será inicializada e, em seguida, a execução dos testes. Isso agilizou o tempo de instalação inicial, desviando da possibilidade de ter qualquer diferença entre as instalações, sendo que todos os *containers* seguirão exatamente os mesmos passos iniciais.

Assim, a adição de mais máquinas se torna simples, pois, não é necessário ter muitos recursos, como recursos visuais para sua execução, porque todas as configurações necessárias estarão presentes dentro do *container*.

Por fim, realizou-se a execução dos testes nas máquinas virtuais disponíveis. A fim de melhor avaliar os resultados obtidos, foram realizados diferentes experimentos tais que, em cada um deles, um número diferente de máquinas foi utilizado para a execução paralela dos testes automatizados. Foram realizados testes em uma, duas e quatro máquinas.

Em seguida, foi aplicado o mesmo conceito no ambiente de produção. Foram executados 1326 testes da aplicação CDS Offline, coletando os dados para posterior análise.

5 AVALIAÇÃO E ANÁLISE

Nesta seção, será descrito o processo de análise dos dados obtidos através das execuções e procedimentos realizados.

5.1 Ambiente Controlado

Inicialmente, como ponto de partida para posterior análise dos resultados obtidos, foi executado, através da IDE eclipse, uma suíte contendo 173 casos de testes. O tempo de execução dos testes foi de 24 minutos e 26 segundos, demonstrado na Figura 13.

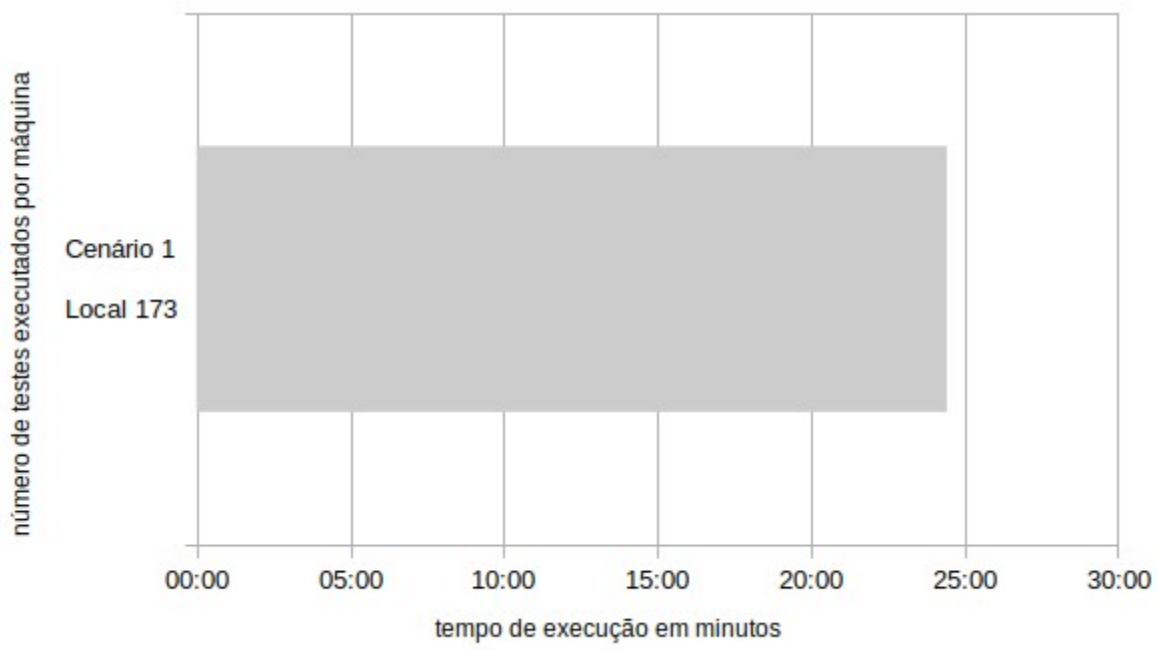

Figura 13 – Execução local

Como um dos questionamentos realizados no início deste trabalho Q1.1, não houve nenhuma modificação realizada na forma de criação dos testes automatizados. Como pode ser visualizado no Capítulo das Ações, o foco foi diretamente no modo de execução dos testes, não existindo a necessidade de realizar mudanças na forma

que são codificados os testes automatizados.

Entretanto, para responder à questão Q1.2, é necessário ressaltar que há uma maior complexidade na forma de executar os testes em paralelo. Visto que, existe no mínimo duas classes a serem executadas, um servidor e, pelo menos, um cliente. Porém, esta abordagem traz benefícios, como a facilidade de automatizar o processo de início da execução dos testes automatizados através de ferramentas existentes no mercado, como o Jenkins.

No experimento em que paralisamos a execução temos o servidor. Nele é realizado a separação dos testes que serão enviados aos clientes, quando os clientes solicitarem testes para execução.

Contudo, a inicialização de novas máquinas se mostra facilitada, visto que consiste apenas na necessidade de inicializar dois *containers*. Caso se faça necessário, o aumento de máquinas para execução em paralelo torna-se mais fácil.

Para efeitos de comparação e análise, foi realizado uma execução cliente/servidor com apenas um cliente, representado pelo Cenário 2. A Figura 14 contém o tempo de execução com os mesmos testes realizados no cenário inicial.

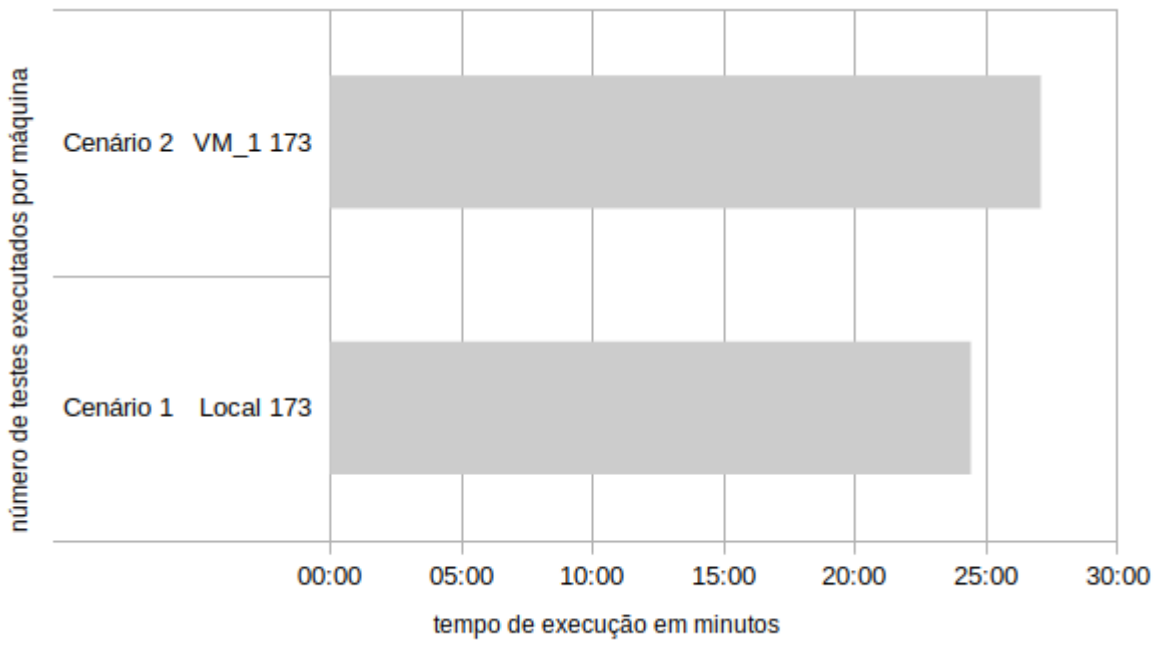

Figura 14 – Execução em uma máquina

O tempo total de execução foi de 27 minutos e 6 segundos, mostrando-se um pouco acima da execução local. Porém, esta análise é apenas para efeito de comparação, pois esta abordagem com apenas um cliente não é benéfico neste contexto.

A Figura 15 mostra a execução em duas máquinas, representado pelo Cenário 3. É possível observar que o número de testes executados por máquina difere neste cenário. Sendo a máquina VM_1 executando 13 testes a mais que a VM_2. Isso se reflete, principalmente, devido ao tempo de execução de cada teste. Porém, o tempo de resposta da rede também é um fator que se pode levar em consideração. O tempo de execução da máquina VM_1 foi de 12 minutos e 19 segundos, enquanto da máquina VM_2 foi de 12 minutos e 14 segundos. Com isso, o tempo total de execução foi igual ao da máquina VM_1, diminuindo aproximadamente 50% quando comparado com o valor da execução local, representado pelo Cenário 1.

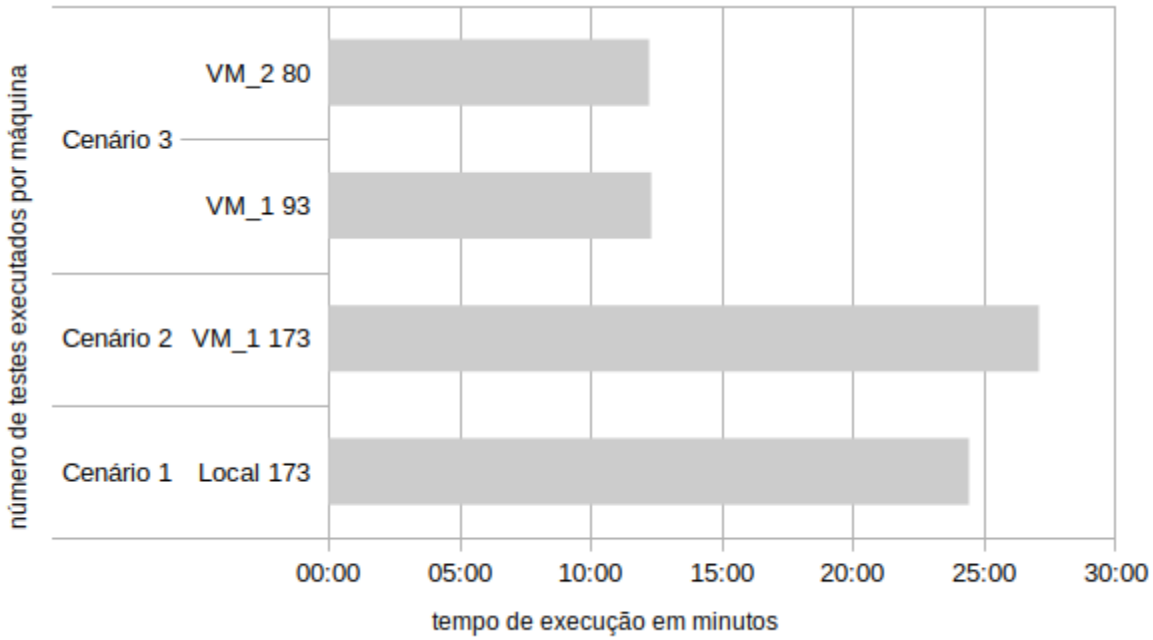

Figura 15 – Execução em duas máquinas

Este resultado era esperado para o tempo de execução. Vale ressaltar que, o resultado mostra apenas o tempo de execução, e não leva em consideração o tempo de configuração inicial e o de inicialização dos *containers*.

Para comparação posterior, foram realizados testes em quatro máquinas, representado pelo Cenário 4. Apesar do número de máquinas ter dobrado, em relação ao Cenário 3, o tempo não caiu pela metade, ficando aproximadamente 34% mais rápido. A máquina VM_1 levou 8 minutos e 10 segundos, a VM_2 levou 8 minutos, a VM_3 levou 8 minutos e 2 segundos e a VM_4 levou 6 minutos e 56 segundos, demonstrado na Figura 16.

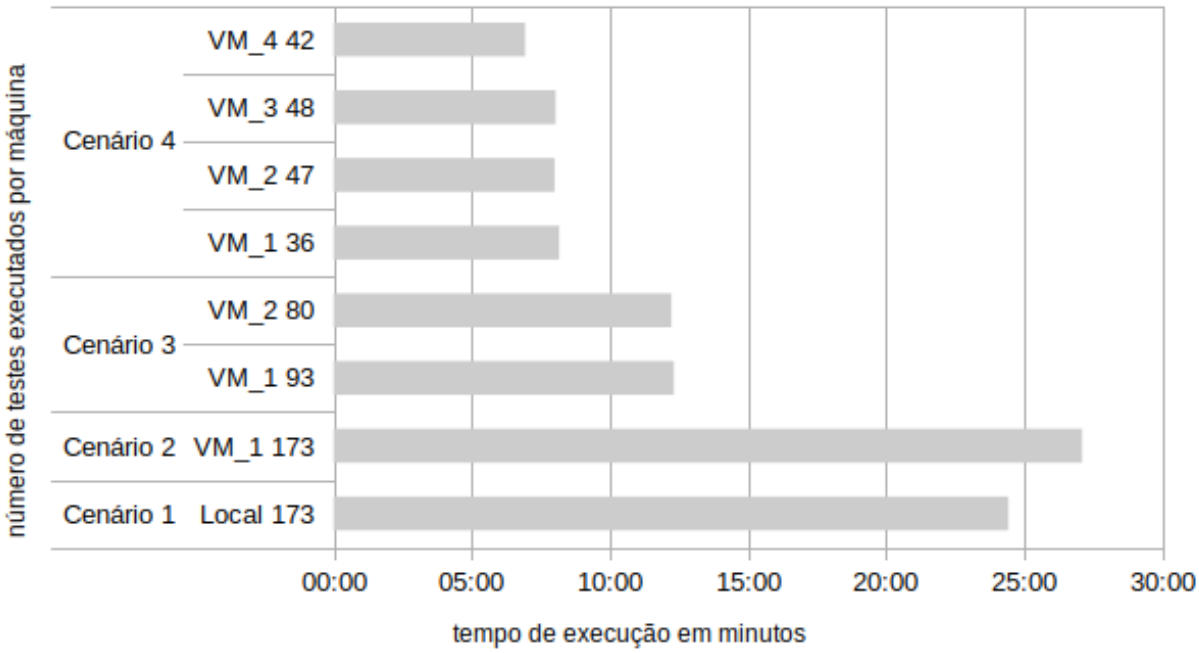

Figura 16 – Execução em quatro máquinas

É possível perceber também neste cenário, um desbalanceamento de testes executados por diferentes máquinas. Porém, apesar da máquina VM_1 executar menos testes que a VM_4, essa última levou menos tempo para finalizar todas as requisições.

Como foi possível perceber durante os testes, o tempo de execução foi diminuindo com o aumento do número de máquinas, no entanto, não foi uma redução proporcionalmente compatível. Somente aumentar o número de máquinas acaba gerando um aumento significativo da utilização de recursos para execução dos testes em paralelo.

Respondendo à questão Q2.3, a análise dos resultados teve uma melhora no tempo de resposta quando comparado com a execução local em uma IDE. O resultado da execução dos testes retorna ao servidor e é impresso na tela, o que não é uma forma amigável de se analisar os resultados quando há um grande número de testes falhando. Um modo interessante seria tratar esse retorno de forma que facilite a

análise dos resultados de maneira mais amigável, porém, isto foge do escopo deste trabalho.

5.2 Ambiente de Produção

Aplicando o conceito abordado anteriormente no contexto de execução real dos testes de CDS Offline, foi realizado testes no módulo das fichas, com 1326 testes. Estes testes foram localmente executados na IDE Eclipse e levou 3 horas e 3 minutos, representado na Figura 17.

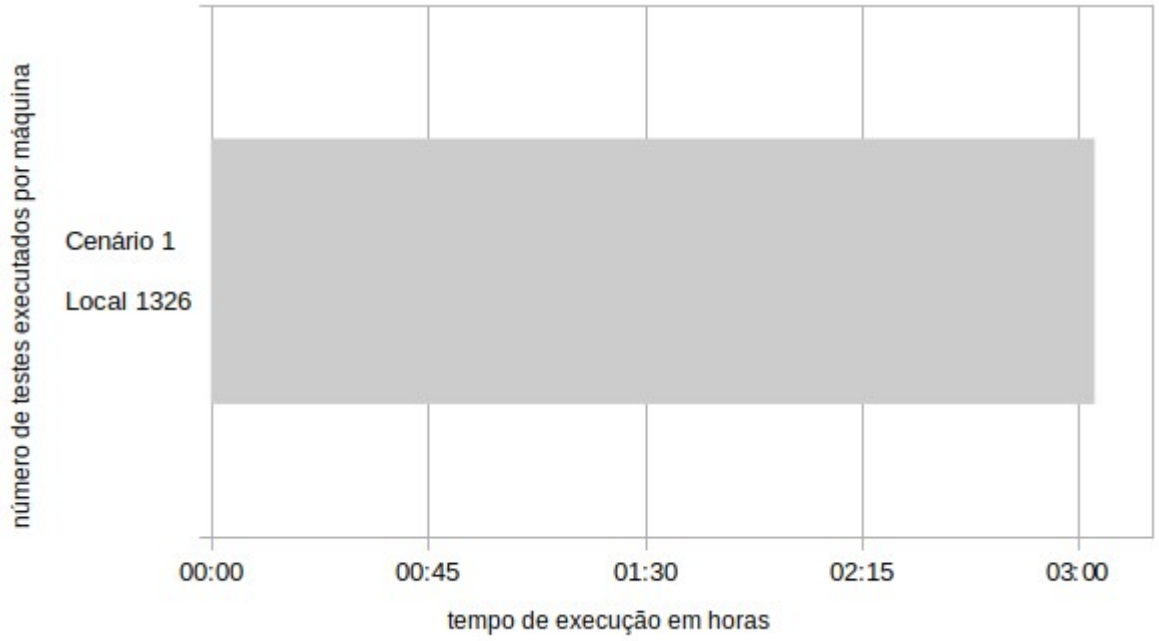

Figura 17 – Execução local

A execução em modo cliente/servidor, com apenas um cliente, levou 2 horas e 57 minutos. Um tempo bem próximo da execução local. A diferença de tempo mostra-se irrelevante, considerando o número elevado de testes. Como estes são executados no navegador, pode existir diferentes tempos de resposta para um mesmo teste quando ele é executado diversas vezes.

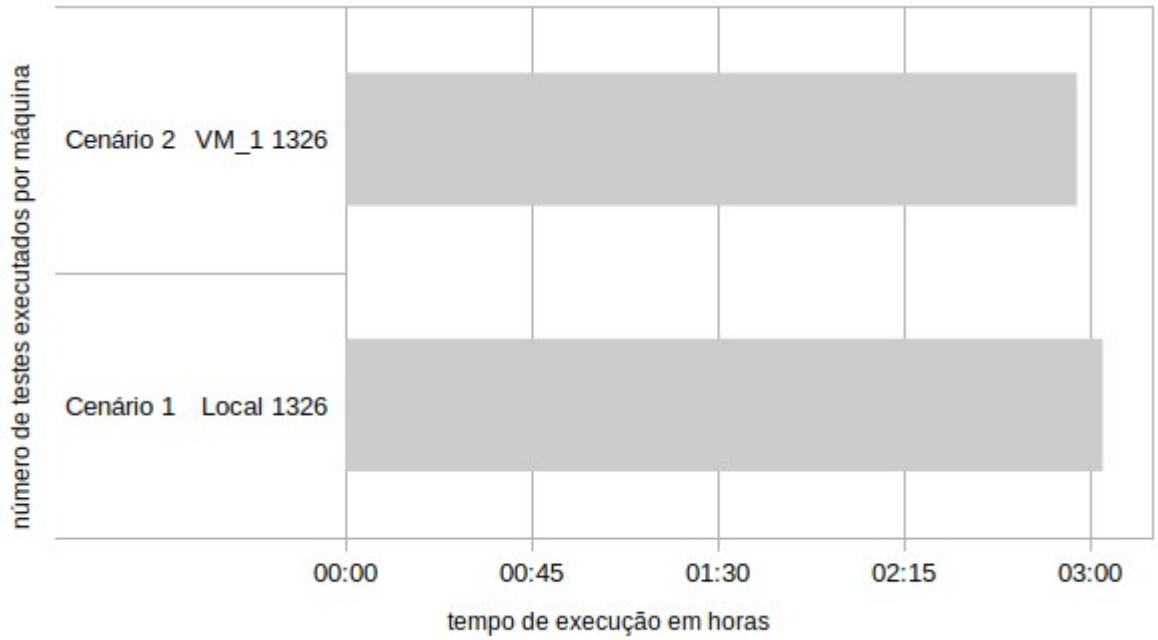

Figura 18 – Execução em uma máquina

Como é mostrado na Figura 19, os testes foram executados em duas máquinas virtuais, representado pelo Cenário 3. Sendo que, 613 testes foram executados na máquina VM_1, levando o tempo de 1 hora e 30 minutos, e 713 testes executados na máquina VM_2, com tempo de 1 hora e 29 minutos. Assim, o tempo que levou a execução em ambiente de produção se mostrou proporcional ao ambiente controlado, levando praticamente a metade do tempo de execução com duas máquinas.

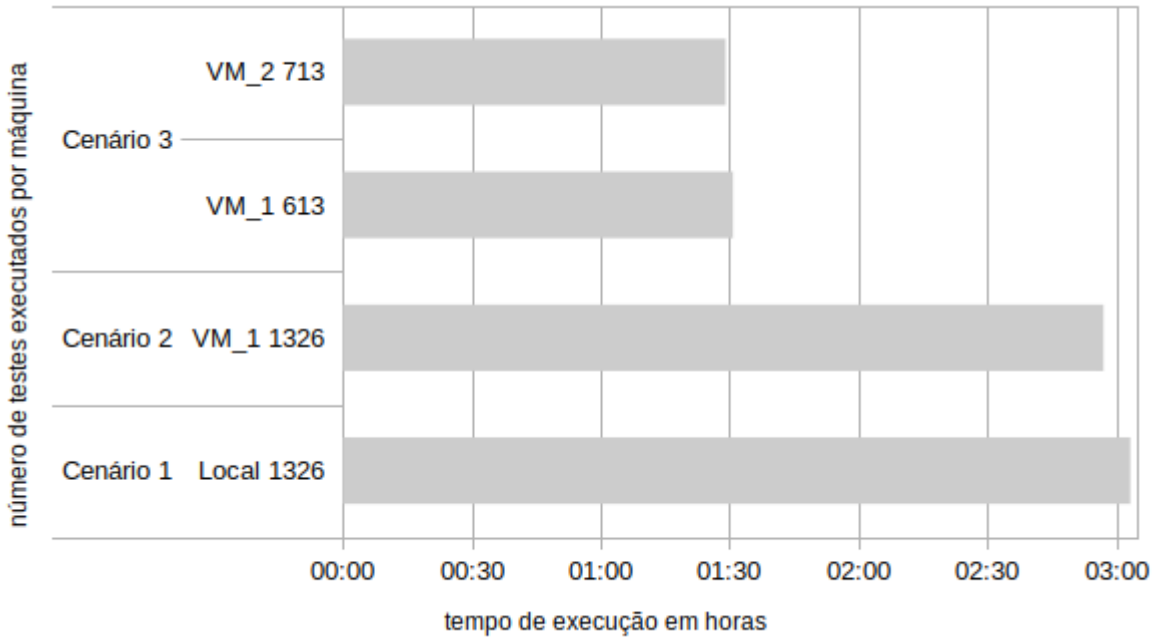

Figura 19 – Execução em duas máquinas

O Cenário 4, que conta com quatro máquinas virtuais, o tempo foi de aproximadamente 52 minutos para 253 testes na máquina VM_1, 52 minutos para 275 testes na máquina VM_2, 48 minutos para 415 testes na máquina VM_3 e 47 minutos para 383 testes na máquina VM_4, demonstrado na Figura 20.

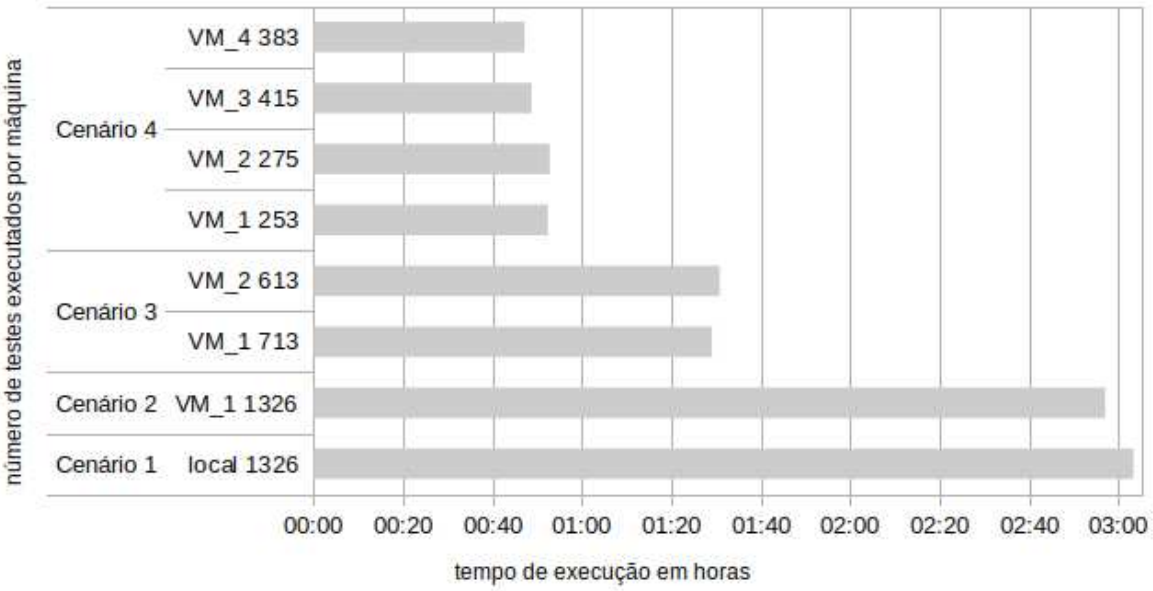

Figura 20 – Execução em quatro máquinas

Assim, como no ambiente controlado, o tempo total de execução no Cenário 4 não caiu pela metade como ocorreu no Cenário 2, apesar de ficar mais próximo. O número de testes executados mostra certo desequilíbrio de execuções por máquina. Apesar da máquina VM_3 ter executado aproximadamente 40% a mais de testes que a máquina VM_1, ela levou 5 minutos a menos para concluir sua execução.

6 REFLEXÕES E APRENDIZADO

Nesta seção, serão examinados os dados coletados, realizando reflexões sobre o conteúdo gerado. Será descrito os possíveis erros e acertos da realização deste trabalho.

Visto a dificuldade de encontrar estudos que tratassem especificamente deste tipo de problema, o início da pesquisa tornou-se de maior dificuldade do que o esperado. Necessitando uma maior discussão e estudo para avaliar a melhor prática a ser adotada, com o menor impacto possível na implementação dos testes automatizados do laboratório Bridge.

O aprendizado de novas tecnologias e a interação delas entre si foi um dos aspectos mais interessante deste trabalho. Pois a integração de aplicações diferentes para chegar a um objetivo é um desafio interessante. Neste caso o desafio de melhoria do tempo da execução de testes automatizados.

A implementação de um cliente/servidor com Java RMI alinhou-se adequadamente aos testes que já eram realizados com a linguagem Java, adquirindo um melhor conhecimento do funcionamento da ferramenta.

A utilização de *containers* mostrou-se um facilitador para escalar o número de máquinas e realizar a execução dos testes. Principalmente com a adição de mais de uma máquina virtual, evitando um esforço repetitivo de ambientação do sistema.

A premissa inicial de delimitar apenas uma instalação da aplicação e uma execução de testes por sistema operacional, devido a características dos testes realizados no laboratório, deverá ser revista. O escalonamento com *containers* permite diversas execuções em um mesmo sistema operacional, sem que exista interferência entre

eles. Alguns testes realizam chamadas de sistema, o que pode impactar diretamente nos *containers* em execução. Tais testes deverão ser tratados de forma diferenciada.

Em se tratando de pesquisa-ação, é possível analisar, com maior realidade, os impactos das escolhas realizadas. No decorrer do desenvolvimento, mais precisamente nas ações tomadas, é que se percebe o quanto decisões podem mudar o rumo da pesquisa. No transcorrer das atividades, teve-se a necessidade de adicionar novas ferramentas e realizar algumas alterações na aplicação que não havia sido previamente levado em consideração.

O objetivo principal foi alcançado. O tempo total da execução dos testes foi reduzido. Através do uso de máquinas virtuais para paralelizar a execução de testes automatizados, com quatro delas se obteve uma redução de até 66% no tempo de execução dos testes feitos para a aplicação CDS Offline. Esse ganho está relacionado diretamente com a segunda questão levantada, trazendo agilidade no tempo de resposta sobre a qualidade do produto.

A ação realizada em um ambiente real de produção, mostrou-se ainda mais eficiente dada as características intrínsecas do sistema de testes. Obteve-se um ganho de mais de 70% no tempo de execução. Dando maior agilidade de realização na análise dos resultados dos testes automatizados. A segurança por parte dos desenvolvedores mostrou-se maior, indicando também uma maior confiança no resultado final da aplicação a cada versão gerada.

A execução paralela dos testes trouxe uma maior complexidade no início da execução dos testes, como a Questão 1 abordou. Contudo, trouxe o benefício de se reduzir o tempo de execução. Por fim, o modelo de paralelismo apresentado neste

trabalho contribuiu no modo como as execuções dos testes são realizadas no Laboratório Bridge. A cada versão que é lançada, a execução de todos os testes é automaticamente iniciada e distribuída entre as diversas máquinas virtuais que estão disponíveis para esse propósito.

7 REFERÊNCIAS

Crispin, Lisa; House, Tip. **Testing Extreme Programming.** Ed. Addison Wesley, 2006.

Chillarege, Ram. **Software Testing Best Practices.** Center for Software Engineering - IBM Research, 1999.

Kim, Gwang-Hun; Kim, Yeon-Gyun; Chung, Kyung-Yong. **Towards virtualized and automated software performance test architecture.** Jun/2013.

Virtualization in Education. IBM Corporation, Whitepaper, Out/2007.

Torkar, Richard. **Towards Automated Software Testing.** Techniques, Classifications and Frameworks**.** 2006.

Karhu, Katja; Repo, Tiina; Taipale, Ossi; Smolander, Kari. **Empirical Observations on Software Testing Automation.** Lappeenranta University of Technology. 2009.

Rätzmann, Manfred; Young, Clinton De. **Software Testing and Internationalization.** Lemoine International, Inc. 2003.

Amaricai, Sabina; Constantinescu, Radu. **Designing a Software Test Automation Framework.** Informatica Economică, vol. 18, p. 152-161. Jan/2014.

Dua, Rajdeep; Raja, A Reddy; Kakadia, Dharmesh. **Virtualization vs Containerization to Support PaaS**. Cloud Engineering (IC2E). IEEE International Conference on. 2014.

Xavier, Miguel G.; Neves, Marcelo V.; Rossi, Fabio D.; Ferreto, Tiago C.; Lange, Timoteo; De Rose Cesar A. F.. **Performance Evaluation of Container-based Virtualization for High Performance Computing Environments.** 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. 2013.

Graham, Dorothy; Fewster,Mark. **Experiences of Test Automation: Case Studies of Software Test Automation**. Pearson Educantion, Inc. 2012.

Berner, Stefan; Weber, Roland; Keller, Rudolf K.. **Observations and Lessons Learned from Automated Testing**. Zühlke Engineering AG. 2005.

Leotta, Maurizio; Clerissi, Diego; Ricca, Filippo; Spadaro, Cristiano. **Comparing the Maintainability of Selenium WebDriver Test Suites Employing Different Locators: A Case Study.** DIBRIS - Università di Genova, Italy. 2013.

Angmo, Rigzin; Sharma, Monika. **Performance Evaluation of Web Based Automation Testing Tools.** University Institute of Engg. & Technology Panjab University (P.U.), India. 2014.

Bruns, Andreas; Kornstädt, Andreas; Wichmann, Dennis. **Web Application Tests with Selenium.** 2009, IEEE Computer Society.

Santos, Paulo Sérgio Medeiros. **Uma Análise da Utilização da Metodologia da Pesquisa-Ação em Engenharia de Software.** 176 f.. Dissertação (Mestrado). COPPE/UFRJ. Rio de Janeiro, 2009.

Avasarala, Satya. **Selenium WebDriver Practical Guide.** Packt Publishing Ltd.. Jan/ 2014.

Xiao-ping, Zhu; Ming-qing, Xiao. **Modeling on Parallel Test System Based on**

Object-Oriented. Air Force Engineering University. 2005.

Vitalino, Jeferson Fernando Noronha; Castro, Marcus André Nunes. **Descomplicando o Docker.** 2016. Editora Brasport.

Berner, Stefan; Weber, Roland; Keller, Rudolf K.. **Observations and Lessons Learned from Automated Testing.** 2005.

SWEBOK. **Guide to the Software Engineering Body of Knowledge**. A project of the IEEE Computer Society Professional Practices Committee, 2004.

Sommerville, Ian. **SOFTWARE ENGINEERING**. — 9th ed. 2010. Editora: Pearson.

APÊNDICE A - Código Fonte

```
 1: package br.ufsc.tcc.rmi.exec;
 2:
 3: import org.junit.runner.Result;
 4:
 5: import lombok.Getter;
 6: import lombok.Setter;
 7:
 8: @Getter
 9: public class TestExecInfo implements java.io.Serializable {
10:
11:  private static final long serialVersionUID = 1L;
12:
13:  private Class<?> suite;
14:  private Class<?> testCase;
15:
16:  @Setter
17:  private Result result;
18:
19:  public TestExecInfo(Class<?> suite, Class<?> testCase) {
20:        super();
21:        this.suite = suite;
22:        this.testCase = testCase;
23:  }
24:
25: }
```

```
 1: package br.ufsc.tcc.rmi.exec;
 2:
 3: import org.junit.runner.JUnitCore;
 4: import org.junit.runner.Request;
 5: import org.junit.runner.Result;
 6:
 7: import lombok.AllArgsConstructor;
 8:
 9: @AllArgsConstructor
10: public class ExecRequest {
11:
12:  private Class<?> testClass;
13:
14:  public Result exec() {
15:
16:        JUnitCore jUnitCore = new JUnitCore();
17:
18:        Request request = Request.aClass(this.testClass);
19:
20:        Result result = jUnitCore.run(request);
21:
22:        String executionPrint = result.wasSuccessful() ? "SUCESSO" : "FALHA";
23:
24:        System.out.println("ExecRequest.exec() - " + this.testClass.getSimpleName() + "   " + executionPrint);
25:
26:        return result;
27:   }
28:
29: }
```

```
 1: package br.ufsc.tcc.rmi;
 2:
 3: import br.ufsc.tcc.projecttest.tests.suite.SuiteFichasCdsOffline;
 4: import br.ufsc.tcc.rmi.parallel.RMIBufferServer;
 5:
 6: public class ScriptServer {
 7:
 8:  public static void main(String[] args) {
 9:
10:         RMIBufferServer bufferServer = new RMIBufferServer();
11:
12:         bufferServer.addSuite(SuiteFichasCdsOffline.class);
13:
14:         bufferServer.startServer();
15:
16:         bufferServer.waitServerFinish();
17:  }
18:
19: }
```

```
 1: package br.ufsc.tcc.rmi;
 2:
 3: import java.io.IOException;
 4: import java.io.InputStream;
 5: import java.util.Date;
 6: import java.util.Properties;
 7:
 8: import lombok.AllArgsConstructor;
 9: import lombok.ToString;
10:
11: import br.ufsc.tcc.rmi.parallel.RMIExecClient;
12:
13: @ToString
14: @AllArgsConstructor
15: public class ScriptClient {
16:
17:  private String serverhost;
18:
19:  private static String databasePath;
20:
21:  public ScriptClient() {
22:        this.readProperties();
23:  }
24:
25:  public static String getDatabasePath() {
26:        if (databasePath == null) {
27:        new ScriptClient();
28:        }
29:        return databasePath;
30:  }
31:
32:  public void doit() {
33:
34:        System.out.println(this.toString());
35:        RMIExecClient rmi = new RMIExecClient(this.serverhost);
36:        rmi.start();
37:  }
38:
39:  public static void main(String[] args) {
40:        System.out.println("ScriptClient.main() - " + new Date().toString());
41:        long initTime = System.currentTimeMillis();
42:        new ScriptClient().doit();
43:
44:        System.out.println("ScriptClient.main() - " + new Date().toString());
45:        long totalTimeMillis = System.currentTimeMillis() - initTime;
46:        System.out.println(
47:        "Fim dos testes. Tempo = " + totalTimeMillis + " ms ou " + totalTimeMillis / 60000.0 + " minutos");
48:  }
49:
50:  private void readProperties() {
51:        ClassLoader classLoader = ScriptClient.class.getClassLoader();
52:
53:        try (InputStream input = classLoader.getResourceAsStream("test.properties");) {
54:        Properties prop = new Properties();
55:        prop.load(input);
56:        this.serverhost = prop.getProperty("test.serverhost");
57:        ScriptClient.databasePath = prop.getProperty("test.databasepath");
58:        } catch (IOException ex) {
59:        System.err.println("Erro ao ler arquivo app.properties");
60:        }
61:  }
62: }
```

```
 1: package br.ufsc.tcc.rmi.parallel;
 2:
 3: import java.net.InetAddress;
 4: import java.net.NetworkInterface;
 5: import java.net.SocketException;
 6: import java.rmi.NotBoundException;
 7: import java.rmi.RemoteException;
 8: import java.rmi.registry.LocateRegistry;
 9: import java.rmi.registry.Registry;
10: import java.rmi.server.UnicastRemoteObject;
11: import java.util.Date;
12: import java.util.Enumeration;
13: import java.util.HashSet;
14: import java.util.LinkedList;
15: import java.util.Set;
16:
17: import org.junit.runners.Suite.SuiteClasses;
18:
19: import br.ufsc.tcc.rmi.exec.TestExecInfo;
20:
21: /*
22:  * implementaÃ§Ã£o do servidor Singleton class
23:  */
24: public class RMIBufferServer implements RMIServices {
25:
26:  private LinkedList<TestExecInfo> buffer;
27:
28:  private Registry registry;
29:
30:  private int totalTests;
31:
32:  private long serverInitTimeMillis;
33:
34:  public RMIBufferServer() {
35:        this.buffer = new LinkedList<>();
36:        System.setProperty("java.rmi.server.hostname", RMIConnectionAux.getHostname());
37:
38:        try {
39:        this.registry = LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
40:        } catch (RemoteException e) {
41:        throw new RuntimeException(e.getMessage(), e);
42:        }
43:  }
44:
45:  public void addSuite(Class<?> suiteClass) {
46:
47:        SuiteClasses suiteClasses = suiteClass.getAnnotation(SuiteClasses.class);
48:        for (Class<?> testClass : suiteClasses.value()) {
49:        if (testClass.isAnnotationPresent(SuiteClasses.class)) {
50:        this.addSuite(testClass);
51:        continue;
52:        }
53:        this.buffer.add(new TestExecInfo(suiteClass, testClass));
54:        }
55:  }
56:
57:  public void startServer() {
58:        try {
59:
60:        this.registry.bind(TEST_BUFFER, UnicastRemoteObject.exportObject(this, 0));
61:        this.totalTests = this.buffer.size();
62:        System.out.println(new Date().toString() + " - BUFFER INICIADO - Total de testes = " + this.buffer.size());
63:        this.serverInitTimeMillis = System.currentTimeMillis();
64:        } catch (Exception e) {
65:        e.printStackTrace();
66:        } finally {
67:        if (this.buffer.isEmpty()) {
68:        this.endServer();
```

```
    69:         }
    70:         }
    71:   }
    72:
    73:   public void endServer() {
    74:         while (this.totalTests != (this.sucesso + this.falhas)) {
    75:         // wait all clients send test result
    76:         }
    77:
    78:         try {
    79:
    80:         this.registry.unbind(TEST_BUFFER);
    81:
    82:         UnicastRemoteObject.unexportObject(this, true);
    83:
    84:         } catch (NotBoundException | RemoteException e) {
    85:         e.printStackTrace();
    86:         } finally {
    87:         if (this.waitServer != null) {
    88:         this.waitServer.interrupt();
    89:         }
    90:         this.printLog();
    91:         }
    92:   }
    93:
    94:   @Override
    95:   public TestExecInfo getTesteParaExecucao() throws RemoteException {
    96:         synchronized (this) {
    97:         if (!this.buffer.isEmpty()) {
    98:         System.out.println("BUFFER total = " + this.buffer.size());
    99:         return this.buffer.removeFirst();
   100:         }
   101:         }
   102:         this.endServer();
   103:         return null;
   104:   }
   105:
   106:   private int falhas = 0;
   107:   private int sucesso = 0;
   108:   private Set<TestExecInfo> errosTestSet = new HashSet<>();
   109:
   110:   @Override
   111:   public synchronized void setResultadoDaExecucao(TestExecInfo testResult) throws RemoteException {
   112:         String resultTest;
   113:         if (testResult.getResult().wasSuccessful()) {
   114:         resultTest = "SUCESSO";
   115:         this.sucesso++;
   116:         } else {
   117:         resultTest = "FALHA";
   118:         this.falhas++;
   119:         this.errosTestSet.add(testResult);
   120:         }
   121:
   122:         System.out.println(String.format("%s - %s : %s", testResult.getSuite().getSimpleName(),
   123:         testResult.getTestCase().getSimpleName(), resultTest));
   124:   }
   125:
   126:   /*
   127:    * Auxiliar mÃ©todos
   128:    */
   129:
   130:   private void printLog() {
   131:         System.out.println(new Date().toString() + " - END SERVER");
   132:         long totalTimeMillis = System.currentTimeMillis() - this.serverInitTimeMillis;
   133:         System.out.println(
   134:         "Fim dos testes. Tempo = " + totalTimeMillis + " ms ou " + (totalTimeMillis / 60000.0) + " minutos");
   135:         System.out.println();
   136:         System.out.println("TOTAL DE TESTES EXECUTADOS = " + this.totalTests);
```

```
137:            System.out.println("Total de testes sucesso    = " + this.sucesso);
138:            System.out.println("Total de testes com erro   = " + this.falhas);
139:            for (TestExecInfo testExecInfo : this.errosTestSet) {
140:            System.out.println(String.format("\t%s - %s", testExecInfo.getSuite().getSimpleName(),
141:            testExecInfo.getTestCase().getSimpleName()));
142:            }
143:    }
144:
145:    private Thread waitServer;
146:
147:    public void waitServerFinish() {
148:            this.waitServer = new Thread(new Runnable() {
149:
150:            @Override
151:            public void run() {
152:            try {
153:            while (true) {
154:            Thread.sleep(60000);
155:            }
156:            } catch (InterruptedException e) {
157:            Thread.currentThread().interrupt();
158:            }
159:            }
160:            });
161:            this.waitServer.start();
162:    }
163:
164:    /*
165:     * Auxiliar class
166:     */
167:
168:    private static class RMIConnectionAux {
169:            private static String getHostname() {
170:            try {
171:            Enumeration<NetworkInterface> networkInterfaces = NetworkInterface.getNetworkInterfaces();
172:            while (networkInterfaces.hasMoreElements()) {
173:            NetworkInterface nextElement = networkInterfaces.nextElement();
174:            Enumeration<InetAddress> inetAddresses = nextElement.getInetAddresses();
175:            String host = hostAddressValue(inetAddresses);
176:            if (host != null) {
177:            return host;
178:            }
179:            }
180:            } catch (SocketException e) {
181:            e.printStackTrace();
182:            }
183:            return null;
184:            }
185:
186:            private static String hostAddressValue(Enumeration<InetAddress> inetAddresses) {
187:            while (inetAddresses.hasMoreElements()) {
188:            InetAddress ia = inetAddresses.nextElement();
189:            String hostAddress = ia.getHostAddress();
190:            if (hostAddress.startsWith("192.168") || hostAddress.startsWith("150.162")) {
191:            return hostAddress;
192:            }
193:            }
194:            return null;
195:            }
196:    }
197:
198:  }
```

```
 1: package br.ufsc.tcc.rmi.parallel;
 2:
 3: import java.rmi.NotBoundException;
 4: import java.rmi.RemoteException;
 5: import java.rmi.registry.LocateRegistry;
 6: import java.rmi.registry.Registry;
 7: import java.util.HashSet;
 8: import java.util.Set;
 9:
10: import org.junit.runner.Result;
11:
12: import br.ufsc.tcc.rmi.exec.ExecRequest;
13: import br.ufsc.tcc.rmi.exec.TestExecInfo;
14:
15: public class RMIExecClient {
16:
17:  private RMIServices stub;
18:
19:  public RMIExecClient(String host) {
20:
21:        try {
22:        Registry registry = LocateRegistry.getRegistry(host);
23:        this.stub = (RMIServices) registry.lookup(RMIServices.TEST_BUFFER);
24:        } catch (NotBoundException | RemoteException e) {
25:        e.printStackTrace();
26:        }
27:  }
28:
29:  public void start() {
30:        while (true) {
31:        TestExecInfo rmiRequest = this.getTestExecInfo();
32:
33:        if (rmiRequest == null) {
34:        break;
35:        }
36:
37:        this.executeSequence(rmiRequest);
38:        }
39:        this.printLog();
40:  }
41:
42:  private void executeSequence(TestExecInfo rmiRequest) {
43:
44:        if (rmiRequest.getTestCase() != null) {
45:        Result result = new ExecRequest(rmiRequest.getTestCase()).exec();
46:        rmiRequest.setResult(result);
47:        this.sendResult(rmiRequest);
48:        }
49:
50:  }
51:
52:  private void sendResult(final TestExecInfo testExecInfo) {
53:        this.storeData(testExecInfo);
54:
55:        final RMIServices stubAux = this.stub;
56:        // criado nova thread para thread principal nÃ£o esperar o envio e continuar
57:        new Thread(new Runnable() {
58:        @Override
59:        public void run() {
60:        try {
61:        stubAux.setResultadoDaExecucao(testExecInfo);
62:        } catch (RemoteException e) {
63:        this.run();
64:        e.printStackTrace();
65:        }
66:        }
67:        }, testExecInfo.getTestCase().getSimpleName()).start();
68:  }
```

```
 69:
 70:   private int falhas = 0;
 71:   private int sucesso = 0;
 72:   private Set<TestExecInfo> falhasTestSet = new HashSet<>();
 73:
 74:   private void storeData(TestExecInfo testExecInfo) {
 75:         if (testExecInfo.getResult().wasSuccessful()) {
 76:         this.sucesso++;
 77:         } else {
 78:         this.falhas++;
 79:         this.falhasTestSet.add(testExecInfo);
 80:         }
 81:   }
 82:
 83:   private TestExecInfo getTestExecInfo() {
 84:
 85:         try {
 86:         return this.stub.getTesteParaExecucao();
 87:         } catch (RemoteException e) {
 88:         e.printStackTrace();
 89:         }
 90:
 91:         return null;
 92:   }
 93:
 94:   private void printLog() {
 95:         System.out.println("cliente finalizado");
 96:         System.out.println("Total testes executados = " + (this.falhas + this.sucesso));
 97:         System.out.println("Total testes sucesso    = " + this.sucesso);
 98:         System.out.println("Total testes com falhas = " + this.falhas);
 99:
100:         for (TestExecInfo testExecInfo : this.falhasTestSet) {
101:         System.out.println(String.format("%s - %s", testExecInfo.getSuite().getSimpleName(),
102:         testExecInfo.getTestCase().getSimpleName()));
103:         }
104:   }
105:
106:  }
```

```
 1: package br.ufsc.tcc.rmi.parallel;
 2:
 3: import java.rmi.Remote;
 4: import java.rmi.RemoteException;
 5:
 6: import br.ufsc.tcc.rmi.exec.TestExecInfo;
 7:
 8: /*
 9:  * serviÃ§os oferecidos pelo servidor
10:  */
11:
12: public interface RMIServices extends Remote {
13:
14:     String TEST_BUFFER = "testbuffer";
15:
16:     TestExecInfo getTesteParaExecucao() throws RemoteException;
17:
18:     void setResultadoDaExecucao(TestExecInfo testResult) throws RemoteException;
19:
20: }
```

```
 1:  package br.ufsc.tcc.framework.browser;
 2:
 3:  import org.openqa.selenium.WebDriver;
 4:
 5:  public interface WebDriverManager {
 6:
 7:   WebDriver createWebDriver();
 8:
 9:   void closeWebDriver();
10:
11:  }
```

```
 1: package br.ufsc.tcc.framework.testrunner.asserts;
 2:
 3: /*
 4:  * Classe abstrata que serÃ¡ modelo para os tipos de verificaÃ§Ãµes do sistema
 5:  */
 6: public abstract class Verificador {
 7:
 8:  private Verificador() {
 9:  }
10:
11:  public static Coletor getColetor() {
12:         return Coletor.getInstance();
13:  }
14:
15: }
```

```
 1: package br.ufsc.tcc.framework.testrunner.asserts;
 2:
 3: import java.util.LinkedList;
 4: import java.util.List;
 5:
 6: /*
 7:  * Singleton Class
 8:  */
 9: public class Coletor {
10:
11:  private static Coletor instance;
12:  private List<AssertionError> listaDeFalhas;
13:
14:  private Coletor() {
15:        this.listaDeFalhas = new LinkedList<>();
16:  }
17:
18:  public static Coletor getInstance() {
19:        if (instance == null) {
20:        instance = new Coletor();
21:        }
22:        return instance;
23:  }
24:
25:  public void addFalha(AssertionError item) {
26:        this.listaDeFalhas.add(item);
27:  }
28:
29:  public AssertionError[] getItens() {
30:        return this.listaDeFalhas.toArray(new AssertionError[this.listaDeFalhas.size()]);
31:  }
32:
33:  public void esvaziarListaDeFalhas() {
34:        this.listaDeFalhas = new LinkedList<>();
35:  }
36:
37: }
```

```
 1: package br.ufsc.tcc.framework.testrunner.annotation;
 2:
 3: import java.lang.annotation.ElementType;
 4: import java.lang.annotation.Retention;
 5: import java.lang.annotation.RetentionPolicy;
 6: import java.lang.annotation.Target;
 7:
 8: import br.ufsc.tcc.framework.testrunner.CasoDeTeste;
 9:
10: @Retention(RetentionPolicy.RUNTIME)
11: @Target(ElementType.TYPE)
12: public @interface PreRequisito {
13:
14:  Class<? extends CasoDeTeste> value();
15:
16: }
```

```
 1: package br.ufsc.tcc.framework.testrunner;
 2:
 3: import org.openqa.selenium.WebDriver;
 4:
 5: import br.ufsc.tcc.framework.browser.WebDriverManager;
 6: import br.ufsc.tcc.framework.exceptions.WebDriverManagerNotFoundException;
 7:
 8: public class EstadoCDT {
 9:
10:  private WebDriverManager webdriverManager;
11:  private WebDriver webdriver;
12:  private Class<?> cdtExecutado;
13:
14:  public EstadoCDT(WebDriverManager webdriverManager) {
15:        this.webdriverManager = webdriverManager;
16:  }
17:
18:  public void createWebDriver() {
19:        this.webdriver = this.webdriverManager.createWebDriver();
20:
21:        if (this.webdriver == null) {
22:        throw new WebDriverManagerNotFoundException();
23:        }
24:  }
25:
26:  public void closeWebDriver() {
27:        this.webdriverManager.closeWebDriver();
28:  }
29:
30:  public WebDriver getWebDriver() {
31:        return this.webdriver;
32:  }
33:
34:  public Class<?> getCdtExecutado() {
35:        return this.cdtExecutado;
36:  }
37:
38:  public void setCdtExecutado(Class<?> novoCDT) {
39:        this.cdtExecutado = novoCDT;
40:  }
41:
42: }
```

```
 1: package br.ufsc.tcc.framework.testrunner;
 2:
 3: import java.util.LinkedList;
 4:
 5: import br.ufsc.tcc.framework.exceptions.CycleTestException;
 6: import br.ufsc.tcc.framework.testrunner.annotation.PreRequisito;
 7: import br.ufsc.tcc.framework.testrunner.execution.ExecucaoCasoDeTeste;
 8: import br.ufsc.tcc.framework.testrunner.execution.ExecucaoCdt;
 9: import br.ufsc.tcc.framework.testrunner.execution.ExecucaoPreRequisito;
10:
11: public class ExecutionController {
12:
13:  private static ExecutionController instance;
14:  private TestRunnerController testRunnerController;
15:  private boolean prepararNovoEstadoCDT;
16:
17:  private ExecutionController() {
18:        this.testRunnerController = TestRunnerController.instance();
19:  }
20:
21:  public static ExecutionController getInstance() {
22:        if (instance == null) {
23:        instance = new ExecutionController();
24:        }
25:        return instance;
26:  }
27:
28:  public final void execucaoDoTeste(CasoDeTeste cdt) {
29:        try {
30:        this.executarPilhaDeExecucao(this.getPilhaDeExecucao(cdt));
31:        } catch (Exception exception) {
32:        throw exception;
33:        } finally {
34:        this.testRunnerController.finalizarEstadoAtual();
35:        }
36:  }
37:
38:  private void executarPilhaDeExecucao(LinkedList<ExecucaoCdt> pilha) {
39:
40:        EstadoCDT estadoCdt = this.testRunnerController.getEstadoAtual();
41:
42:        if (this.prepararNovoEstadoCDT) {
43:        this.testRunnerController.finalizarEstadoAtual();
44:        estadoCdt = new EstadoCDT(this.testRunnerController.getWebDriverManager());
45:        estadoCdt.createWebDriver();
46:        this.testRunnerController.setEstadoAtual(estadoCdt);
47:        }
48:
49:        while (!pilha.isEmpty()) {
50:        ExecucaoCdt cdt = pilha.pop();
51:        cdt.executar(estadoCdt.getWebDriver());
52:        }
53:
54:        this.testRunnerController.setEstadoAtual(estadoCdt);
55:  }
56:
57:  /*
58:   * MÃ©todos que montam a pilha de execuÃ§Ã£o
59:   */
60:
61:  private LinkedList<ExecucaoCdt> getPilhaDeExecucao(CasoDeTeste cdt) {
62:
63:        LinkedList<ExecucaoCdt> pilhaDeExecucao = new LinkedList<>();
64:        LinkedList<Class<?>> listaClassesDeTestes = new LinkedList<>(); // verificaÃ§Ã£o de ciclos
65:
66:        // adiciona o teste como primeiro da pilha de testes a ser executados
67:        pilhaDeExecucao.push(new ExecucaoCasoDeTeste(cdt));
68:        listaClassesDeTestes.add(cdt.getClass()); // verificaÃ§Ã£o de ciclos
```

```
69:
70:         CasoDeTeste cdtAuxiliarAtual = cdt;
71:
72:         // define pilha de execuÃ§Ã£o enquanto existir prÃ©-requisitos
73:         while (cdtAuxiliarAtual != null) {
74:
75:         this.definirPreRequisitoDoCasoDeTeste(cdtAuxiliarAtual);
76:
77:         Class<? extends CasoDeTeste> preRequisito = cdtAuxiliarAtual.getPreRequisitoDoCasoDeTeste();
78:
79:         if (preRequisito != null) {
80:
81:         this.verificarPresencaDeCiclos(listaClassesDeTestes, preRequisito); // verificaÃ§Ã£o de ciclos
82:
83:         CasoDeTeste cdtPreRequisito = criarInstancia(preRequisito);
84:
85:         // add cdt na pilha de execuÃ§Ã£o
86:         pilhaDeExecucao.push(new ExecucaoPreRequisito(cdtPreRequisito));
87:         listaClassesDeTestes.add(preRequisito); // verificaÃ§Ã£o de ciclos
88:
89:         cdtAuxiliarAtual = cdtPreRequisito;
90:         } else {
91:         this.prepararNovoEstadoCDT = true;
92:         return pilhaDeExecucao;
93:         }
94:         }
95:         return pilhaDeExecucao;
96:     }
97:
98:     private static CasoDeTeste criarInstancia(Class<? extends CasoDeTeste> cdtClass) {
99:         try {
100:        return cdtClass.newInstance();
101:        } catch (InstantiationException | IllegalAccessException e) {
102:        throw new RuntimeException(e);
103:        }
104:    }
105:
106:    private void verificarPresencaDeCiclos(LinkedList<Class<?>> listaClasses, Class<? extends CasoDeTeste> cdt) {
107:        if (listaClasses.contains(cdt)) {
108:        throw new CycleTestException(listaClasses, cdt);
109:        }
110:    }
111:
112:    /*
113:     * Adiciona na lista do caso de teste os prÃ©-requisitos
114:     */
115:    private void definirPreRequisitoDoCasoDeTeste(CasoDeTeste cdt) {
116:
117:        Class<?> classCasoDeTeste = cdt.getClass();
118:        while (classCasoDeTeste != CasoDeTeste.class) {
119:        if (classCasoDeTeste.isAnnotationPresent(PreRequisito.class)) {
120:        PreRequisito preRequisitoAnnotation = classCasoDeTeste.getAnnotation(PreRequisito.class);
121:        cdt.setPreRequistoDoCasoDeTeste(preRequisitoAnnotation.value());
122:        }
123:        classCasoDeTeste = classCasoDeTeste.getSuperclass();
124:        }
125:    }
126:
127: }
```

```
 1: package br.ufsc.tcc.framework.testrunner;
 2:
 3: import org.junit.runner.RunWith;
 4: import org.openqa.selenium.WebDriver;
 5:
 6: /*
 7:  * RunWith invocaÃ§Ã£o de um Custom Runner que estende Runner do JUnit
 8:  */
 9: @RunWith(TestRunner.class)
10: /*
11:  * Classe abstrata que serve de modelo para os Casos de testes a serem criados
12:  */
13: public abstract class CasoDeTeste {
14:  /*
15:   * PrÃ©-requisito explicito no Caso de teste
16:   */
17:  private Class<? extends CasoDeTeste> preRequisitoEspecificadoNaClasseDeTeste;
18:
19:  /*
20:   * Passos funciona como um fluxo "Before" os asserts
21:   */
22:  public void passos(WebDriver webdriver) {
23:  }
24:
25:  /*
26:   * Ãºnico mÃ©todo abstrato da classe onde Ã© realizado as verificaÃ§Ãµes dos testes
27:   */
28:  public abstract void asserts(WebDriver webdriver);
29:
30:  /*
31:   * passosAposAsserts funciona como um fluxo "After" apÃ³s os asserts
32:   */
33:  public void passosAposAsserts(WebDriver webdriver) {
34:  }
35:
36:  /*
37:   * adiciona o prÃ©-requisito especificado na classe com a
38:   * annotation @PreRequisito
39:   */
40:  public void setPreRequistoDoCasoDeTeste(Class<? extends CasoDeTeste> cdt) {
41:        this.preRequisitoEspecificadoNaClasseDeTeste = cdt;
42:  }
43:
44:  /*
45:   * Retorna prÃ©-requisitos
46:   */
47:  public Class<? extends CasoDeTeste> getPreRequisitoDoCasoDeTeste() {
48:        return this.preRequisitoEspecificadoNaClasseDeTeste;
49:  }
50:
51: }
```

```
 1: package br.ufsc.tcc.framework.testrunner.execution;
 2:
 3: import org.openqa.selenium.WebDriver;
 4:
 5: import br.ufsc.tcc.framework.testrunner.CasoDeTeste;
 6:
 7: public class ExecucaoPreRequisito implements ExecucaoCdt {
 8:
 9:  private final CasoDeTeste cdt;
10:
11:  public ExecucaoPreRequisito(CasoDeTeste cdt) {
12:        this.cdt = cdt;
13:  }
14:
15:  @Override
16:  public void executar(WebDriver webdriver) {
17:        this.cdt.passos(webdriver);
18:        this.cdt.passosAposAsserts(webdriver);
19:  }
20:
21: }
```

```
1:  package br.ufsc.tcc.framework.testrunner.execution;
2:
3:  import org.openqa.selenium.WebDriver;
4:
5:  public interface ExecucaoCdt {
6:
7:    void executar(WebDriver driver);
8:
9:  }
```

```
 1: package br.ufsc.tcc.framework.testrunner.execution;
 2:
 3: import org.openqa.selenium.WebDriver;
 4:
 5: import br.ufsc.tcc.framework.testrunner.CasoDeTeste;
 6:
 7: public class ExecucaoCasoDeTeste implements ExecucaoCdt {
 8:
 9:  private CasoDeTeste cdt;
10:
11:  public ExecucaoCasoDeTeste(CasoDeTeste cdt) {
12:         this.cdt = cdt;
13:  }
14:
15:  @Override
16:  public void executar(WebDriver webdriver) {
17:         this.cdt.passos(webdriver);
18:         this.cdt.asserts(webdriver);
19:         this.cdt.passosAposAsserts(webdriver);
20:  }
21:
22: }
```

```
 1: package br.ufsc.tcc.framework.testrunner;
 2:
 3: import java.util.LinkedList;
 4:
 5: import org.openqa.selenium.WebDriver;
 6:
 7: import br.ufsc.tcc.framework.browser.WebDriverManager;
 8: import br.ufsc.tcc.framework.exceptions.WebDriverManagerNotFoundException;
 9: import br.ufsc.tcc.framework.exceptions.WebDriverNotCreatedException;
10: import br.ufsc.tcc.testproject.config.driver.ChromeDriverCreator;
11:
12: public class TestRunnerController {
13:
14:  private boolean suiteEmAndamento;
15:  private EstadoCDT estadoAtual;
16:
17:  private WebDriverManager webDriverManager;
18:  private LinkedList<String> suitesAbertas;
19:
20:  private static TestRunnerController instance;
21:
22:  private TestRunnerController() {
23:        this.suiteEmAndamento = false;
24:        this.suitesAbertas = new LinkedList<>();
25:
26:        this.webDriverManager = this.instanciateWebDriverManagerByReflection();
27:  }
28:
29:  private WebDriverManager instanciateWebDriverManagerByReflection() {
30:
31:        try {
32:        String packageDefaultWebDriverCreator = ChromeDriverCreator.class.getCanonicalName();
33:        return (WebDriverManager) Class.forName(packageDefaultWebDriverCreator).newInstance();
34:        } catch (InstantiationException | IllegalAccessException | ClassNotFoundException e) {
35:        e.printStackTrace();
36:        }
37:        throw new WebDriverManagerNotFoundException();
38:  }
39:
40:  public static TestRunnerController instance() {
41:        if (instance == null) {
42:        instance = new TestRunnerController();
43:        }
44:        return instance;
45:  }
46:
47:  protected WebDriverManager getWebDriverManager() {
48:        return this.webDriverManager;
49:  }
50:
51:  // ----------------------------------------------------------------
52:  public WebDriver getDriver() {
53:        if ((this.estadoAtual == null) || (this.estadoAtual.getWebDriver() == null)) {
54:        throw new WebDriverNotCreatedException();
55:        }
56:        return this.estadoAtual.getWebDriver();
57:  }
58:
59:  protected void inicioDaSuite() {
60:        if (!this.suiteEmAndamento) {
61:        this.suitesAbertas.push("A");
62:        this.suiteEmAndamento = true;
63:        }
64:  }
65:
66:  protected void fimDaSuite() {
67:        if (this.suiteEmAndamento) {
68:        this.finalizarEstadoAtual();
```

```
69:
70:         this.suitesAbertas.pop();
71:
72:         if (this.suitesAbertas.isEmpty()) {
73:         this.suiteEmAndamento = false;
74:         }
75:         }
76:   }
77:
78:   protected EstadoCDT getEstadoAtual() {
79:         return this.estadoAtual;
80:   }
81:
82:   protected void setEstadoAtual(EstadoCDT novoEstado) {
83:         this.estadoAtual = novoEstado;
84:   }
85:
86:   protected void finalizarEstadoAtual() {
87:
88:         if (this.estadoAtual != null) {
89:         try {
90:         if (this.estadoAtual.getWebDriver() != null) {
91:         this.estadoAtual.closeWebDriver();
92:         }
93:         } finally {
94:         this.estadoAtual = null;
95:         }
96:         }
97:
98:   }
99:
100:  }
```

```
 1: package br.ufsc.tcc.framework.testrunner;
 2:
 3: import org.junit.runner.Description;
 4: import org.junit.runner.Runner;
 5: import org.junit.runner.notification.Failure;
 6: import org.junit.runner.notification.RunNotifier;
 7:
 8: import br.ufsc.tcc.framework.testrunner.asserts.Coletor;
 9:
10: /*
11:  * Custom Runner
12:  */
13: public class TestRunner extends Runner {
14:
15:  private CasoDeTeste cdt;
16:  private Description descriptionCdt;
17:
18:  /*
19:   * Constructor que tem como parÃ¢metro a classe do CasoDeTeste
20:   */
21:  public TestRunner(Class<?> cdtClass) {
22:
23:        // define Description para posterior execuÃ§Ã£o do teste
24:        this.descriptionCdt = Description.createTestDescription(cdtClass, cdtClass.getSimpleName());
25:
26:        try {
27:        // cria uma nova instacia de cada caso de teste via reflection
28:        this.cdt = (CasoDeTeste) Class.forName(cdtClass.getName()).newInstance();
29:        } catch (InstantiationException | IllegalAccessException | ClassNotFoundException e) {
30:        e.printStackTrace();
31:        }
32:  }
33:
34:  @Override
35:  public Description getDescription() {
36:        return this.descriptionCdt;
37:  }
38:
39:  @Override
40:  public void run(RunNotifier rn) {
41:        rn.fireTestRunStarted(this.descriptionCdt);
42:        rn.fireTestStarted(this.descriptionCdt);
43:
44:        // zera coletor a cada execuÃ§Ã£o
45:        Coletor coletor = Coletor.getInstance();
46:        coletor.esvaziarListaDeFalhas();
47:
48:        try {
49:        // inicia a execuÃ§Ã£o do teste
50:        ExecutionController.getInstance().execucaoDoTeste(this.cdt);
51:
52:        } catch (Exception exception) {
53:        // qualquer excessÃ£o resulta numa falha do teste
54:        Failure fail = new Failure(this.descriptionCdt, exception);
55:        rn.fireTestFailure(fail);
56:        exception.printStackTrace();
57:
58:        } finally {
59:        // Todas as falhas coletadas resultam numa falha do teste
60:        AssertionError[] falhas = coletor.getItens();
61:        for (AssertionError falha : falhas) {
62:        Failure fail = new Failure(this.descriptionCdt, falha);
63:        rn.fireTestFailure(fail);
64:        }
65:        rn.fireTestFinished(this.descriptionCdt);
66:        }
67:  }
68:
```

```
69: }
```

```
 1: package br.ufsc.tcc.framework.exceptions;
 2:
 3: public class LoadPageException extends RuntimeException {
 4:
 5:  private static final long serialVersionUID = 1L;
 6:
 7:  public LoadPageException() {
 8:        super("A pÃ¡gina inicial nÃ£o foi carregada.");
 9:  }
10:
11:  public LoadPageException(Throwable t) {
12:        super("A pÃ¡gina inicial nÃ£o foi carregada.", t);
13:  }
14:
15: }
```

```
 1: package br.ufsc.tcc.framework.exceptions;
 2:
 3: public class WebDriverNotCreatedException extends RuntimeException {
 4:
 5:  private static final long serialVersionUID = 1L;
 6:
 7:  public WebDriverNotCreatedException() {
 8:        super("O webdriver nÃ£o foi criado.");
 9:  }
10:
11: }
```

```
 1: package br.ufsc.tcc.framework.exceptions;
 2:
 3: public class WebDriverManagerNotFoundException extends RuntimeException {
 4:
 5:  private static final long serialVersionUID = 1L;
 6:
 7:  public WebDriverManagerNotFoundException() {
 8:        super("O WebDriverCreator nÃ£o foi encontrado.");
 9:  }
10:
11: }
```

```
 1: package br.ufsc.tcc.framework.exceptions;
 2:
 3: import java.util.LinkedList;
 4:
 5: import br.ufsc.tcc.framework.testrunner.CasoDeTeste;
 6:
 7: public class CycleTestException extends RuntimeException {
 8:  private static final long serialVersionUID = 1L;
 9:
10:  public CycleTestException(Class<?> clazz) {
11:        super("O teste " + clazz.getSimpleName() + " possui ciclo entre seus prÃ©-requisitos: \n");
12:  }
13:
14:  public CycleTestException(String msg) {
15:        super(msg);
16:  }
17:
18:  public CycleTestException(LinkedList<Class<?>> listaClasses, Class<? extends CasoDeTeste> classCdtAnterior) {
19:        super(getMensagemErroCiclo(listaClasses, classCdtAnterior));
20:  }
21:
22:  private static String getMensagemErroCiclo(LinkedList<Class<?>> listaClasses, Class<?> classAtual) {
23:        StringBuilder msg = new StringBuilder(
24:        "O teste " + classAtual.getClass().getSimpleName() + " possui ciclo entre seus prÃ©-requisitos: \n");
25:        for (Class<?> clazz : listaClasses) {
26:
27:        if (clazz == classAtual) {
28:        msg.append(" > ");
29:        } else {
30:        msg.append("    ");
31:        }
32:
33:        msg.append(clazz.getName() + "\n");
34:        }
35:
36:        msg.append(" > " + classAtual.getName() + "");
37:
38:        return msg.toString();
39:  }
40:
41: }
```

```
 1: package br.ufsc.tcc.testproject.config.driver;
 2:
 3: import java.util.concurrent.TimeUnit;
 4:
 5: import org.openqa.selenium.WebDriver;
 6:
 7: import br.ufsc.tcc.framework.testrunner.TestRunnerController;
 8:
 9: public class DriverFactory {
10:
11:  public static Driver getDriver() {
12:
13:         TestRunnerController controle = TestRunnerController.instance();
14:         WebDriver webdriver = controle.getDriver();
15:         Driver driver = new Driver();
16:         driver.wd = webdriver;
17:         driver.wd.manage().timeouts().implicitlyWait(1, TimeUnit.SECONDS);
18:         driver.idHandleMaster = driver.wd.getWindowHandle();
19:
20:         return driver;
21:  }
22:
23: }
```

```
 1: package br.ufsc.tcc.testproject.config.driver;
 2:
 3: import java.util.List;
 4:
 5: import org.openqa.selenium.By;
 6: import org.openqa.selenium.TimeoutException;
 7: import org.openqa.selenium.WebDriver;
 8: import org.openqa.selenium.WebElement;
 9: import org.openqa.selenium.interactions.Actions;
10: import org.openqa.selenium.support.ui.ExpectedCondition;
11: import org.openqa.selenium.support.ui.ExpectedConditions;
12: import org.openqa.selenium.support.ui.WebDriverWait;
13:
14: import br.ufsc.tcc.framework.exceptions.LoadPageException;
15:
16: public class Driver {
17:
18:  protected WebDriver wd;
19:
20:  protected String idHandleMaster;
21:  private boolean isClosed;
22:
23:  private static final int TIMEOUT_DEFAULT = 5;
24:
25:  public WebDriver getWebDriver() {
26:        return this.wd;
27:  }
28:
29:  public void acessar(String pag) {
30:        try {
31:        this.wd.get(pag);
32:        } catch (TimeoutException e) {
33:        this.wd.navigate().refresh();
34:        } catch (Exception e) {
35:        throw new LoadPageException(e);
36:        }
37:  }
38:
39:  public void atualizarPagina(String pag) {
40:        this.acessar(pag);
41:  }
42:
43:  public String getUrlAtual() {
44:        return this.wd.getCurrentUrl();
45:  }
46:
47:  public WebElement getClickableElement(By by) {
48:        return this.getClickableElement(by, Driver.TIMEOUT_DEFAULT);
49:  }
50:
51:  public WebElement getClickableElement(By by, int tempoMaximo) {
52:        WebDriverWait wait = new WebDriverWait(this.wd, tempoMaximo);
53:        wait.until(ExpectedConditions.elementToBeClickable(by));
54:        return this.wd.findElement(by);
55:  }
56:
57:  public WebElement getElementoComEspera(By by) {
58:        return this.getElementoComEspera(by, Driver.TIMEOUT_DEFAULT);
59:  }
60:
61:  public WebElement getElementoComEspera(By by, int tempoMaximo) {
62:        WebDriverWait wait = new WebDriverWait(this.wd, tempoMaximo);
63:        wait.until(ExpectedConditions.visibilityOfElementLocated(by));
64:        return this.wd.findElement(by);
65:  }
66:
67:  public WebElement[] getElementosComEspera(By by) {
68:        WebDriverWait wait = new WebDriverWait(this.wd, Driver.TIMEOUT_DEFAULT);
```

```
69:         wait.until(ExpectedConditions.visibilityOfElementLocated(by));
70:         List<WebElement> listaWebElements = this.wd.findElements(by);
71:         WebElement[] elementos = new WebElement[listaWebElements.size()];
72:         int i = 0;
73:         for (WebElement we : listaWebElements) {
74:         elementos[i] = we;
75:         i++;
76:         }
77:         return elementos;
78:     }
79:
80:     public WebElement getElementoComEsperaPorPresenca(By by, int tempoMaximo) {
81:         WebDriverWait wait = new WebDriverWait(this.wd, tempoMaximo);
82:         wait.until(ExpectedConditions.presenceOfElementLocated(by));
83:         return this.wd.findElement(by);
84:     }
85:
86:     public WebElement getElementoSemEspera(By by) {
87:         WebElement we = this.wd.findElement(by);
88:         return we;
89:     }
90:
91:     public WebElement getElementoSemEspera(WebElement elemento, By by) {
92:         WebElement we = elemento.findElement(by);
93:         return we;
94:     }
95:
96:     public boolean waitTextInElement(By by, String texto) {
97:         WebDriverWait wait = new WebDriverWait(this.wd, Driver.TIMEOUT_DEFAULT);
98:         wait.until(ExpectedConditions.textToBePresentInElementLocated(by, texto));
99:         return true;
100:    }
101:
102:    public WebElement[] getElementosSemEspera(By by) {
103:        List<WebElement> listaWebElements = this.wd.findElements(by);
104:        WebElement[] arr = new WebElement[listaWebElements.size()];
105:        int i = 0;
106:        for (WebElement we : listaWebElements) {
107:        arr[i] = we;
108:        i++;
109:        }
110:        return arr;
111:    }
112:
113:    public boolean existeElemento(By by) {
114:        return this.existeElemento(by, Driver.TIMEOUT_DEFAULT);
115:    }
116:
117:    public boolean existeElemento(By by, int tempoMaximo) {
118:        try {
119:        this.getElementoComEspera(by, tempoMaximo);
120:        } catch (Exception e) {
121:        return false;
122:        }
123:        return true;
124:    }
125:
126:    public void esperarPorPagina(final String url) {
127:        this.esperarPorPagina(url, Driver.TIMEOUT_DEFAULT);
128:    }
129:
130:    public void esperarPorPagina(final String url, int tempoMaximo) {
131:        WebDriverWait wait = new WebDriverWait(this.wd, tempoMaximo);
132:        ExpectedCondition<Boolean> expected = new ExpectedCondition<Boolean>() {
133:        @Override
134:        public Boolean apply(WebDriver w) {
135:        return w.getCurrentUrl().startsWith(url);
136:        }
```

```
137:         };
138:         wait.until(expected);
139:    }
140:
141:    public void esperarPelaSaidaDoElemento(final By by, int tempoMaximo) {
142:
143:         WebDriverWait wait = new WebDriverWait(this.wd, tempoMaximo);
144:         wait.until(ExpectedConditions.invisibilityOfElementLocated(by));
145:
146:    }
147:
148:    public void esperarPorTextoEmCampoTexto(final By by, final String texto) {
149:         WebDriverWait wait = new WebDriverWait(this.wd, Driver.TIMEOUT_DEFAULT);
150:
151:         ExpectedCondition<Boolean> expected = new ExpectedCondition<Boolean>() {
152:         @Override
153:         public Boolean apply(WebDriver w) {
154:         return w.findElement(by).getAttribute("value").equals(texto);
155:         }
156:         };
157:         wait.until(expected);
158:         wait.until(ExpectedConditions.visibilityOfElementLocated(by));
159:    }
160:
161:    public void setFocus(WebElement element) {
162:         new Actions(this.wd).moveToElement(element).perform();
163:    }
164:
165:    public WebElement getElementoComFoco() {
166:         return this.wd.switchTo().activeElement();
167:    }
168:
169:    public void moveMouseToElement(By by) {
170:         Actions builder = new Actions(this.wd);
171:         WebElement we = this.getElementoComEspera(by);
172:         builder.moveToElement(we).build().perform();
173:    }
174:
175:    public boolean isClosed() {
176:         return this.isClosed;
177:    }
178:
179:   }
```

```
  1: package br.ufsc.tcc.testproject.config.driver;
  2:
  3: import java.io.File;
  4: import java.io.FileOutputStream;
  5: import java.io.IOException;
  6: import java.io.InputStream;
  7: import java.io.OutputStream;
  8: import java.util.HashMap;
  9: import java.util.Map;
 10: import java.util.concurrent.TimeUnit;
 11:
 12: import org.openqa.selenium.WebDriver;
 13: import org.openqa.selenium.chrome.ChromeDriver;
 14: import org.openqa.selenium.chrome.ChromeDriverService;
 15: import org.openqa.selenium.chrome.ChromeOptions;
 16:
 17: import br.ufsc.tcc.framework.browser.WebDriverManager;
 18: import br.ufsc.tcc.projecttest.utils.comum.funcoes.FuncoesFile;
 19: import br.ufsc.tcc.projecttest.utils.comum.funcoes.FuncoesLinux;
 20: import br.ufsc.tcc.projecttest.utils.comum.funcoes.SOFactory;
 21:
 22: public class ChromeDriverCreator implements WebDriverManager {
 23:
 24:  private static final String DRIVER_PATH = System.getProperty("user.home") + "/tcc-files/chromedriver";
 25:
 26:  private WebDriver webdriver;
 27:
 28:  @Override
 29:  public WebDriver createWebDriver() {
 30:      System.out.println();
 31:      this.prepareAmbiente();
 32:      System.out.println("ChromeDriverCreator.createWebDriver()");
 33:      ChromeDriverService chromeDriverService = new ChromeDriverService.Builder()
 34:      .usingDriverExecutable(new File(DRIVER_PATH)).usingAnyFreePort().build();
 35:
 36:      ChromeOptions options = new ChromeOptions();
 37:      options.addArguments("--start-maximized");
 38:      options.addArguments("--incognito");
 39:      options.addArguments("--no-sandbox");
 40:
 41: //     options.addArguments("--headless");
 42:
 43:      options.addArguments("--user-data-dir=" + System.getProperty("user.home") + "/tcc-files/chromefiles/");
 44:
 45:      Map<String, Object> preferences = new HashMap<>();
 46:      options.setExperimentalOption("prefs", preferences);
 47:      preferences.put("plugins.always_open_pdf_externally", Boolean.TRUE);
 48:      preferences.put("partition.per_host_zoom_levels.x", "{}");
 49: //     preferences.put("useAutomationExtension", Boolean.FALSE);
 50:
 51:      this.webdriver = new ChromeDriver(chromeDriverService, options);
 52:      this.webdriver.manage().timeouts().pageLoadTimeout(15, TimeUnit.SECONDS);
 53:      return this.webdriver;
 54:
 55:  }
 56:
 57:  @Override
 58:  public void closeWebDriver() {
 59:      this.webdriver.close();
 60:      this.webdriver.quit();
 61:  }
 62:
 63:  private void prepareAmbiente() {
 64:      SOFactory.getInstance().killWebDriver();
 65:      FuncoesFile.createFolder(System.getProperty("user.home") + "/tcc-files/");
 66:      System.out.println("ChromeDriverCreator.prepareAmbiente()");
 67:      String extract = ChromeDriverCreator.extract("/chromedriver");
 68:      FuncoesLinux.tornarArquivoExecutavel(ChromeDriverCreator.DRIVER_PATH);
```

```
69:         System.out.println("ChromeDriverCreator.prepareAmbiente() - extracted = " + extract);
70:     }
71:
72:     public static String extract(String jarFilePath) {
73:   //    https://coderwall.com/p/d0kssw/resource-extraction-from-jar-to-local-file-system
74:         if (jarFilePath == null) {
75:         return null;
76:         }
77:
78:         try (InputStream fileStream = ChromeDriverCreator.class.getResourceAsStream(jarFilePath);
79:         OutputStream out = new FileOutputStream(DRIVER_PATH);) {
80:
81:         if (fileStream == null) {
82:         System.err.println("NÃ£o foi encontrado o arquivo");
83:         return null;
84:         }
85:
86:         // Create an output stream to barf to the temp file
87:
88:         // Write the file to the temp file
89:         byte[] buffer = new byte[1024];
90:         int len = fileStream.read(buffer);
91:         while (len != -1) {
92:         out.write(buffer, 0, len);
93:         len = fileStream.read(buffer);
94:         }
95:
96:         // Return the path of this sweet new file
97:         return DRIVER_PATH;
98:
99:         } catch (IOException e) {
100:        e.printStackTrace();
101:        return null;
102:        }
103:    }
104:
105:  }
```

```
 1: package br.ufsc.tcc.testproject.config.database;
 2:
 3: import java.sql.Connection;
 4: import java.sql.SQLException;
 5: import java.sql.Statement;
 6: import java.util.Set;
 7:
 8: import br.ufsc.tcc.testproject.config.database.generator.TableInfo;
 9: import br.ufsc.tcc.testproject.utils.database.jdbc.ConnectionDB;
10:
11: public class CleanDatabaseTestService {
12:
13:  public void deleteTablesH2(Set<TableInfo> tablesInfo) {
14:
15:         try (Connection connection = ConnectionDB.getConnection();
16:         Statement statement = connection.createStatement();) {
17:
18:         statement.addBatch("SET REFERENTIAL_INTEGRITY FALSE");
19:         for (TableInfo tableInfo : tablesInfo) {
20:         statement.addBatch(String.format("DELETE FROM %s WHERE %s > '%s'", tableInfo.getTableName(),
21:         tableInfo.getPkColumnName(), tableInfo.getMaxValue()));
22:         }
23:
24:         statement.addBatch("SET REFERENTIAL_INTEGRITY TRUE");
25:         statement.executeBatch();
26:         } catch (SQLException e) {
27:         throw new RuntimeException("erro na deleÃ§Ã£o das tabelas", e);
28:         }
29:  }
30:
31: }
```

```
 1: package br.ufsc.tcc.testproject.config.database;
 2:
 3: import br.ufsc.tcc.rmi.ScriptClient;
 4: import lombok.Getter;
 5:
 6: @Getter
 7: public class DatabaseValues {
 8:
 9:  private DatabaseConfig dbConfig;
10:
11:  public static final DatabaseConfig DB_Postgres = new DatabaseConfig() {
12:        {
13:        this.setConnectionUrl("jdbc:postgresql://localhost:5432/esus");
14:        this.setLogin("esus");
15:        this.setPassword("esus");
16:        }
17:  };
18:
19:  // info about AUTO_SERVER h2 :
20:  // http://www.h2database.com/html/features.html#auto_mixed_mode
21:  public static final DatabaseConfig DB_H2 = new DatabaseConfig() {
22:        {
23:        this.setConnectionUrl("jdbc:h2:"
24: //       + System.getProperty("user.home") + "/tcc-files/cdsoffline_container"
25:        + ScriptClient.getDatabasePath()
26:        + "/bin/database/esus-h2;"
27: //       + "CACHE_TYPE=SOFT_LRU;"
28: //       + "LOCK_TIMEOUT=10000;"
29: //       + "MVCC=TRUE;"
30: //       + "ifexists=true;"
31:        + "AUTO_SERVER=TRUE");// added on standalone.xml
32:        this.setLogin("sa");
33:        this.setPassword("sa");
34:        }
35:  };
36: }
```

```
 1: package br.ufsc.tcc.testproject.config.database.generator;
 2:
 3: import java.io.File;
 4: import java.io.FileNotFoundException;
 5: import java.io.PrintStream;
 6: import java.io.StringWriter;
 7: import java.io.UnsupportedEncodingException;
 8: import java.security.MessageDigest;
 9: import java.security.NoSuchAlgorithmException;
10: import java.util.ArrayList;
11: import java.util.Arrays;
12: import java.util.Collections;
13: import java.util.HashSet;
14: import java.util.List;
15: import java.util.Set;
16:
17: import javax.swing.UIManager;
18: import javax.swing.UnsupportedLookAndFeelException;
19:
20: import org.apache.commons.lang3.StringUtils;
21:
22: public abstract class BaseGenerator {
23:
24:  private static final char[] IDENTATION;
25:
26:  static {
27:         try {
28:         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
29:         } catch (ClassNotFoundException e) {
30:         } catch (InstantiationException e) {
31:         } catch (IllegalAccessException e) {
32:         } catch (UnsupportedLookAndFeelException e) {
33:         }
34:
35:         char[] ident = new char[100];
36:         for (int i = 0; i < 100; i++) {
37:         ident[i] = '\t';
38:         }
39:         IDENTATION = ident;
40:
41:  }
42:
43:  public BaseGenerator() {
44:         this.buffer = new StringWriter();
45:         this.imports = new HashSet<>();
46:         this.identation = 0;
47:         this.overwrite = true;
48:  }
49:
50:  /*
51:   * Fields
52:   */
53:
54:  protected File folder;
55:
56:  protected boolean overwrite;
57:
58:  private StringWriter buffer;
59:
60:  private Set<String> imports;
61:
62:  private int importPosition;
63:
64:  private int identation;
65:
66:  private boolean commentedCodeBlock = false;
67:
68:  /*
```

```
69:  * Setters
70:  */
71:
72:  public void setFolder(File folder) {
73:      this.folder = folder;
74:  }
75:
76:  /*
77:   * Public API
78:   */
79:
80:  public void execute() throws Exception {
81:      if (this.folder == null) {
82:      throw new AssertionError("folder Ã© obrigatÃ³rio");
83:      }
84:
85:      if (!this.folder.exists()) {
86:      if (!this.folder.mkdirs()) {
87:      throw new AssertionError("Nao foi possivel criar a pasta");
88:      }
89:      }
90:  }
91:
92:  /*
93:   * Utility API
94:   */
95:
96:  protected void markImport() {
97:      this.importPosition = this.buffer.getBuffer().length();
98:  }
99:
100: private String buildImport() {
101:     List<String> entries = new ArrayList<>();
102:
103:     {
104:     List<String> javaEntries = new ArrayList<>();
105:     List<String> otherEntries = new ArrayList<>();
106:     for (String oneImport : this.imports) {
107:     String prefix = oneImport.split("\\.")[0];
108:     if ("java".equals(prefix) || "javax".equals(prefix)) {
109:     javaEntries.add(oneImport);
110:     } else {
111:     otherEntries.add(oneImport);
112:     }
113:     }
114:
115:     Collections.sort(javaEntries);
116:     Collections.sort(otherEntries);
117:
118:     String prev = null;
119:     for (List<String> group : Arrays.asList(javaEntries, otherEntries)) {
120:     for (String oneImport : group) {
121:     String cur = oneImport.split("\\.")[0];
122:     if (prev != null && !cur.equals(prev)) {
123:     entries.add(null);
124:     }
125:     entries.add("import " + oneImport + ";");
126:     prev = cur;
127:     }
128:     }
129:     }
130:
131:     return StringUtils.join(entries, "\n");
132: }
133:
134: public String importType(Class<?> type) {
135:     if (type.isPrimitive() || type.getPackage() == String.class.getPackage()) {
136:     return type.getSimpleName();
```

```
137:        }
138:
139:        this.imports.add(type.getName());
140:
141:        return type.getSimpleName();
142:    }
143:
144:    public void importType(String typeName) {
145:        this.imports.add(typeName);
146:    }
147:
148:    public void ident() {
149:        this.identation = Math.min(this.identation + 1, IDENTATION.length - 1);
150:    }
151:
152:    public void unident() {
153:        this.identation = Math.max(this.identation - 1, 0);
154:    }
155:
156:    public void writeln() {
157:        this.buffer.write('\n');
158:    }
159:
160:    public void writelnSingleIdentation(String text) {
161:        text = this.processComments(text);
162:        if (this.identation > 0) {
163:        this.buffer.write(IDENTATION, 0, this.identation + 1);
164:        }
165:        this.buffer.write(text);
166:        this.buffer.write('\n');
167:    }
168:
169:    public void writelnSingleIdentation(String text, Object... args) {
170:        this.writelnSingleIdentation(String.format(text, args));
171:    }
172:
173:    public void writeln(String text) {
174:        text = this.processComments(text);
175:        if (this.identation > 0) {
176:        this.buffer.write(IDENTATION, 0, this.identation);
177:        }
178:        this.buffer.write(text);
179:        this.buffer.write('\n');
180:    }
181:
182:    public void identToWrite() {
183:        if (this.identation > 0) {
184:        this.buffer.write(IDENTATION, 0, this.identation);
185:        }
186:    }
187:
188:    public void write(String text) {
189:        text = this.processComments(text);
190:        this.buffer.write(text);
191:    }
192:
193:    public void writeln(String text, Object... args) {
194:        this.writeln(String.format(text, args));
195:    }
196:
197:    private long generateSerialVersionID() throws NoSuchAlgorithmException, UnsupportedEncodingException {
198:        long result = 0L;
199:
200:        byte[] hashBytes = MessageDigest.getInstance("SHA")
201:        .digest(this.buffer.getBuffer().toString().getBytes("UTF-8"));
202:
203:        /*
204:         * 9. The hash value is assembled from the first and second 32-bit values of the
```

```
205:          * SHA-1 message digest. If the result of the message digest, the five 32-bit
206:          * words H0 H1 H2 H3 H4, is in an array of five int values named sha, the hash
207:          * value would be computed as follows: long hash = ((sha[0] >>> 24) & 0xFF) |
208:          * ((sha[0] >>> 16) & 0xFF) << 8 | ((sha[0] >>> 8) & 0xFF) << 16 | ((sha[0] >>>
209:          * 0) & 0xFF) << 24 | ((sha[1] >>> 24) & 0xFF) << 32 | ((sha[1] >>> 16) & 0xFF)
210:          * << 40 | ((sha[1] >>> 8) & 0xFF) << 48 | ((sha[1] >>> 0) & 0xFF) << 56;
211:          */
212:         for (int i = Math.min(hashBytes.length, 8) - 1; i >= 0; i--) {
213:         result = result << 8 | hashBytes[i] & 0xFF;
214:         }
215:
216:         return result;
217:     }
218:
219:     protected abstract File getOutputFile();
220:
221:     protected void save() throws FileNotFoundException, UnsupportedEncodingException, NoSuchAlgorithmException
{
222:         String imports = this.buildImport();
223:         if (imports != null && imports.trim().length() > 0) {
224:         this.buffer.getBuffer().insert(this.importPosition, imports + "\n\n");
225:         }
226:         int versionPosition = this.buffer.getBuffer().indexOf("$$VERSION$$");
227:         if (versionPosition != -1) {
228:         this.buffer.getBuffer().replace(versionPosition, versionPosition + 11,
229:         String.valueOf(this.generateSerialVersionID()) + "L");
230:         }
231:
232:         File file = this.getOutputFile();
233:         if (this.canWriteFile(file, this.buffer.toString())) {
234:         PrintStream out = new PrintStream(file, "UTF-8");
235:         try {
236:         out.print(this.buffer.toString());
237:         out.flush();
238:         this.onFileSavedSuccessfully(file);
239:         } finally {
240:         out.close();
241:         }
242:         } else {
243:         logDebug("Buffer nÃ£o gravado: jÃ¡ existe o arquivo %s (override: false)", file);
244:         }
245:     }
246:
247:     protected void onFileSavedSuccessfully(File file) {
248:         logDebug("Buffer (%s chars) gravado em %s", this.buffer.toString().length(), file.getAbsolutePath());
249:     }
250:
251:     protected boolean canWriteFile(File file, String content) {
252:         return this.overwrite || !file.exists();
253:     }
254:
255:     protected String getBooleanString(boolean value) {
256:         return value ? "TRUE" : "FALSE";
257:     }
258:
259:     public static void logDebug(String message, Object... params) {
260:         if (logDebug) {
261:         log("DEBUG", message, params);
262:         }
263:     }
264:
265:     public static void logInfo(String message, Object... params) {
266:         log("INFO ", message, params);
267:     }
268:
269:     private static boolean logDebug = true;
270:
271:     protected static void log(String level, String message, Object... params) {
```

```
272:            if (params != null && params.length > 0) {
273:            message = String.format(message, params);
274:            }
275:            logOut.writeln(String.format("[%s] %s", level, message));
276:    }
277:
278:    private String processComments(String input) {
279:            if (this.commentedCodeBlock) {
280:            input = input.trim();
281:            if (input.startsWith("//")) {
282:            this.buffer.write("//");
283:            input = input.substring(2).trim();
284:            }
285:            }
286:            return input;
287:    }
288:
289:    public static LogWriter logOut = new LogWriter() {
290:
291:            @Override
292:            public void write(String s) {
293:            System.out.print(s);
294:            }
295:
296:            @Override
297:            public void writeln(String s) {
298:            System.out.println(s);
299:            }
300:
301:    };
302:
303:    public interface LogWriter {
304:            void write(String s);
305:
306:            void writeln(String s);
307:    }
308:
309:  }
```

```
 1: package br.ufsc.tcc.testproject.config.database.generator;
 2:
 3: import java.io.File;
 4: import java.sql.Connection;
 5: import java.sql.DriverManager;
 6: import java.sql.ResultSet;
 7: import java.sql.SQLException;
 8: import java.util.LinkedList;
 9: import java.util.List;
10:
11: import com.google.common.collect.ImmutableSet;
12:
13: import br.ufsc.tcc.testproject.config.database.CdsSequencesInfo;
14: import br.ufsc.tcc.testproject.config.database.DatabaseConfig;
15: import br.ufsc.tcc.testproject.config.database.DatabaseValues;
16:
17: public class GeradorSequences extends BaseGenerator {
18:
19:  private List<SequencesInfo> sequencesInfo;
20:
21:  private DatabaseConfig conn;
22:
23:  public GeradorSequences() {
24:        this.sequencesInfo = new LinkedList<>();
25:
26:        this.conn = DatabaseValues.DB_H2;
27:        System.out.println(this.conn.toString());
28:  }
29:
30:  public void calculate() {
31:        try (Connection connection = DriverManager.getConnection(this.conn.getConnectionUrl(), this.conn.getLogin(),
32:        this.conn.getPassword());) {
33:        try (ResultSet executeQuery = connection.createStatement().executeQuery("select * from ALL_SEQUENCES;")
;) {
34: //      .executeQuery("SELECT sequence_name from \"information_schema\".\"sequences\";");) {
35:
36:        while (executeQuery.next()) {
37:        this.sequencesInfo.add(new SequencesInfo(executeQuery.getString(1)));
38:        }
39:        }
40:        } catch (SQLException e) {
41:        e.printStackTrace();
42:        }
43:        // sort list
44: //      this.sequencesInfo = this.sequencesInfo.stream().sorted(Comparator.comparing(SequencesInfo::getSequence))
45: //      .collect(Collectors.toList());
46:  }
47:
48:  @Override
49:  protected File getOutputFile() {
50:        String projectFolder = this.getClass().getProtectionDomain().getCodeSource().getLocation().getPath()
51:        .replace("target/classes/", "");
52:        return new File(projectFolder + "/src/main/java/br/ufsc/tcc/testproject/config/database/",
53:        "CdsSequencesInfo.java");
54:  }
55:
56:  @Override
57:  public void execute() throws Exception {
58:        File outputFile = this.getOutputFile();
59:        this.setFolder(outputFile.getParentFile());
60:        super.execute();
61:
62:        this.writeln(CdsSequencesInfo.class.getPackage() + ";");
63:        this.writeln();
64:        this.writeln("import " + SequencesInfo.class.getCanonicalName() + ";");
65:        this.writeln();
66:        this.writeln("import " + ImmutableSet.class.getCanonicalName() + ";");
67:        this.writeln();
```

```
68:         this.writeln("//@formatter:off");
69:         this.writeln(String.format("public class %s {", outputFile.getName().replace(".java", "")));
70:         this.writeln();
71:         this.ident();
72:
73:         this.writeln();
74:         this.writeln("public static final ImmutableSet<SequencesInfo> getSequencesInfo() {");
75:         this.ident();
76:         this.writeln("return ImmutableSet.<SequencesInfo> builder()");
77:         this.ident();
78:         for (SequencesInfo cInfo : this.sequencesInfo) {
79:         this.writeln(String.format(".add(new SequencesInfo(\"%s\")", cInfo.getSequence()));
80:         }
81:         this.unident();
82:         this.writeln(".build();");
83:         this.unident();
84:         this.writeln("}");
85:         this.unident();
86:         this.writeln();
87:         this.unident();
88:         this.writeln("}");
89:
90:         super.save();
91:     }
92:
93:     public static void main(String[] args) {
94:         GeradorSequences gerador = new GeradorSequences();
95:         gerador.calculate();
96:
97:         try {
98:         gerador.execute();
99:         } catch (Exception e) {
100:        e.printStackTrace();
101:        }
102:    }
103:
104:    }
```

```
1:  package br.ufsc.tcc.testproject.config.database.generator;
2:
3:  import lombok.AllArgsConstructor;
4:  import lombok.Getter;
5:
6:  @Getter
7:  @AllArgsConstructor
8:  public class SequencesInfo {
9:
10:   private String sequence;
11:
12:  }
```

```
 1: package br.ufsc.tcc.testproject.config.database.generator;
 2:
 3: import lombok.AllArgsConstructor;
 4: import lombok.Getter;
 5: import lombok.NoArgsConstructor;
 6: import lombok.Setter;
 7: import lombok.ToString;
 8:
 9: @Getter
10: @Setter
11: @ToString
12: @NoArgsConstructor
13: @AllArgsConstructor
14: public class ContraintInfo {
15:
16:  private String constraintName;
17:
18:  private String tableName;
19:
20:  private String fkColumnName;
21:
22:  private String pkTableName;
23:
24:  private String pkColumnName;
25:
26: }
```

```
 1: package br.ufsc.tcc.testproject.config.database.generator;
 2:
 3: import lombok.AllArgsConstructor;
 4: import lombok.Getter;
 5: import lombok.NoArgsConstructor;
 6: import lombok.Setter;
 7: import lombok.ToString;
 8:
 9: @Getter
10: @Setter
11: @ToString
12: @NoArgsConstructor
13: @AllArgsConstructor
14: public class TableInfo {
15:
16:  private String tableName;
17:
18:  private String pkColumnName;
19:
20:  private Long maxValue;
21:
22: }
```

```
 1: package br.ufsc.tcc.testproject.config.database.generator;
 2:
 3: import java.io.File;
 4: import java.sql.Connection;
 5: import java.sql.DatabaseMetaData;
 6: import java.sql.DriverManager;
 7: import java.sql.ResultSet;
 8: import java.sql.SQLException;
 9: import java.util.HashSet;
10: import java.util.LinkedList;
11: import java.util.List;
12: import java.util.Set;
13:
14: import br.ufsc.tcc.testproject.config.database.CdsTables;
15: import br.ufsc.tcc.testproject.config.database.DatabaseConfig;
16: import br.ufsc.tcc.testproject.config.database.DatabaseValues;
17:
18: public class GeradorTabelas extends BaseGenerator {
19:
20:  private List<ContraintInfo> contraintsInfos;
21:  private List<TableInfo> tableInfos;
22:
23:  private DatabaseConfig conn;
24:
25:  public GeradorTabelas() {
26:      this.contraintsInfos = new LinkedList<>();
27:      this.tableInfos = new LinkedList<>();
28:
29:      this.conn = DatabaseValues.DB_H2;
30:      System.out.println(this.conn.toString());
31:  }
32:
33:  public void calculate() {
34:      try (Connection connection = DriverManager.getConnection(this.conn.getConnectionUrl(), this.conn.getLogin(),
35:      this.conn.getPassword());) {
36:      DatabaseMetaData metaData = connection.getMetaData();
37:      try (ResultSet table = metaData.getTables(null, null, "%", new String[] { "TABLE" })) {
38:
39:      while (table.next()) {
40:      this.contraintInfoData(metaData, table);
41:
42:      this.tableInfoData(connection, metaData, table);
43:
44:      }
45:      }
46:      } catch (SQLException e) {
47:      e.printStackTrace();
48:      }
49:
50:      // sort lists
51: //   this.contraintsInfos = this.contraintsInfos.stream()
52: //   .sorted(Comparator.comparing(ContraintInfo::getConstraintName)).collect(Collectors.toList());
53: //   this.tableInfos = this.tableInfos.stream().sorted(Comparator.comparing(TableInfo::getTableName))
54: //   .collect(Collectors.toList());
55:  }
56:
57:  private void tableInfoData(Connection connection, DatabaseMetaData metaData, ResultSet table) throws SQLExcepti
on {
58:      String tableName = table.getString("TABLE_NAME");
59:      try (ResultSet pk = metaData.getPrimaryKeys(null, null, tableName);) {
60:
61:      while (pk.next()) {
62:      String columnName = pk.getString("COLUMN_NAME");
63:      try (ResultSet executeQuery = connection.createStatement()
64:      .executeQuery(String.format("SELECT MAX(%s) FROM %s", columnName, tableName));) {
65:      if (executeQuery.next()) {
66:      TableInfo tableInfo = new TableInfo();
67:      tableInfo.setTableName(tableName);
```

```
68:             tableInfo.setPkColumnName(columnName);
69:             tableInfo.setMaxValue(executeQuery.getLong(1));
70:             this.tableInfos.add(tableInfo);
71:             }
72:             } catch (Exception e) {
73:             System.err.println("PK - " + columnName + " isn't long - " + tableName);
74:             }
75:             }
76:             }
77:     }
78:
79:     private void contraintInfoData(DatabaseMetaData metaData, ResultSet tables) throws SQLException {
80:             String tableName = tables.getString("TABLE_NAME");
81:
82:             try (ResultSet fk = metaData.getImportedKeys(null, null, tableName);) {
83:
84:             while (fk.next()) {
85:             if (fk.getString("FK_NAME") != null) {
86:             ContraintInfo contraintsInfo = new ContraintInfo();
87:             contraintsInfo.setTableName(tableName);
88:             contraintsInfo.setConstraintName(fk.getString("FK_NAME"));
89:             contraintsInfo.setFkColumnName(fk.getString("FKCOLUMN_NAME"));
90:             contraintsInfo.setPkTableName(fk.getString("PKTABLE_NAME"));
91:             contraintsInfo.setPkColumnName(fk.getString("PKCOLUMN_NAME"));
92:             this.contraintsInfos.add(contraintsInfo);
93:             }
94:             }
95:             } catch (Exception e) {
96:             // do nothing
97:             }
98:     }
99:
100:    @Override
101:    protected File getOutputFile() {
102:            String projectFolder = this.getClass().getProtectionDomain().getCodeSource().getLocation().getPath()
103:            .replace("target/classes/", "");
104:            String parent = projectFolder + "/src/main/java/"
105:            + CdsTables.class.getPackage().getName().replaceAll("\\.", "/");
106:            return new File(parent, "CdsTables.java");
107:    }
108:
109:    @Override
110:    public void execute() throws Exception {
111:            File outputFile = this.getOutputFile();
112:            this.setFolder(outputFile.getParentFile());
113:            super.execute();
114:
115:            this.writeln(CdsTables.class.getPackage() + ";");
116:            this.writeln();
117:            this.writeln("import " + HashSet.class.getCanonicalName() + ";");
118:            this.writeln("import " + Set.class.getCanonicalName() + ";");
119:            this.writeln();
120:    //      this.writeln("import " + ContraintInfo.class.getCanonicalName() + ";");
121:            this.writeln("import " + TableInfo.class.getCanonicalName() + ";");
122:            this.writeln();
123:            this.writeln("//@formatter:off");
124:            this.writeln(String.format("public class %s {", outputFile.getName().replace(".java", "")));
125:            this.writeln();
126:            this.ident();
127:
128:            this.writeln("public static final Set<TableInfo> getTablesInfo() {");
129:            this.ident();
130:            this.writeln("Set<TableInfo> tableSet = new HashSet<>();");
131:            for (TableInfo tableInfo : this.tableInfos) {
132:            this.writeln(String.format("tableSet.add(new TableInfo(\"%s\", \"%s\", %sL));", tableInfo.getTableName(),
133:            tableInfo.getPkColumnName(), tableInfo.getMaxValue()));
134:            }
135:            this.writeln("return tableSet;");
```

```
136:            this.unident();
137:            this.writeln("}");
138:
139:    //      this.writeln();
140:    //      this.writeln("public static final Set<ContraintInfo> getConstraintsInfo() {");
141:    //      this.ident();
142:    //      this.writeln("Set<ContraintInfo> cInfoSet = new HashSet<>();");
143:    //      for (ContraintInfo cInfo : this.contraintsInfos) {
144:    //      this.writeln(String.format("cInfoSet.add(new ContraintInfo(\"%s\", \"%s\", \"%s\", \"%s\", \"%s\"));",
145:    //      cInfo.getConstraintName(), cInfo.getTableName(), cInfo.getFkColumnName(), cInfo.getPkTableName(),
146:    //      cInfo.getPkColumnName()));
147:    //      }
148:    //      this.writeln("return cInfoSet;");
149:    //      this.unident();
150:    //      this.writeln("}");
151:            this.unident();
152:            this.writeln();
153:            this.unident();
154:            this.writeln("}");
155:
156:            super.save();
157:     }
158:
159:     public static void main(String[] args) {
160:
161:            GeradorTabelas gerador = new GeradorTabelas();
162:            gerador.calculate();
163:
164:            try {
165:            gerador.execute();
166:            } catch (Exception e) {
167:            e.printStackTrace();
168:            }
169:     }
170:
171:    }
```

```
 1: package br.ufsc.tcc.testproject.config.database;
 2:
 3: import java.util.HashSet;
 4: import java.util.Set;
 5:
 6: import br.ufsc.tcc.testproject.config.database.generator.TableInfo;
 7:
 8: //@formatter:off
 9: public class CdsTablesSemImportarCnes {
10:
11:  public static final Set<TableInfo> getTablesInfo() {
12:         Set<TableInfo> tableSet = new HashSet<>();
13:         tableSet.add(new TableInfo("RL_ANTECEDENTE_CIAP", "CO_CIAP", 0L));
14:         tableSet.add(new TableInfo("RL_ANTECEDENTE_CIAP", "CO_PRONTUARIO", 0L));
15:         tableSet.add(new TableInfo("RL_ATEND_PROCED", "CO_SEQ_ATEND_PROCED", 0L));
16:         tableSet.add(new TableInfo("RL_ATEND_PROF_AD_TIPO_INELEGVL", "CO_ATEND_PROF_AD", 0L));
17:         tableSet.add(new TableInfo("RL_ATEND_PROF_AD_TIPO_INELEGVL", "TP_AD_INELEGIVEL", 0L));
18:         tableSet.add(new TableInfo("RL_ATEND_PROF_CONDUTA", "CO_ATEND_PROF", 0L));
19:         tableSet.add(new TableInfo("RL_ATEND_PROF_CONDUTA", "TP_CDS_CONDUTA", 0L));
20:         tableSet.add(new TableInfo("RL_ATEND_PROF_ODONTO_TIPO_ENCM", "CO_ATEND_PROF_ODONTO
", 0L));
21:         tableSet.add(new TableInfo("RL_ATEND_PROF_ODONTO_TIPO_ENCM", "TP_ENCAM_ODONTO", 0L));
22:         tableSet.add(new TableInfo("RL_ATEND_PROF_ODONTO_TIPO_FRNC", "CO_ATEND_PROF_ODONTO"
, 0L));
23:         tableSet.add(new TableInfo("RL_ATEND_PROF_ODONTO_TIPO_FRNC", "TP_FORNECIMENTO_ODONT
O", 0L));
24:         tableSet.add(new TableInfo("RL_ATEND_PROF_PIC", "CO_ATEND_PROF", 0L));
25:         tableSet.add(new TableInfo("RL_ATEND_PROF_PIC", "CO_CDS_PIC", 0L));
26:         tableSet.add(new TableInfo("RL_ATEND_TIPO_SERVICO", "CO_ATEND", 0L));
27:         tableSet.add(new TableInfo("RL_ATEND_TIPO_SERVICO", "TP_SERVICO", 0L));
28:         tableSet.add(new TableInfo("RL_ATOR_PAPEL_PERFIL", "CO_ATOR_PAPEL", 0L));
29:         tableSet.add(new TableInfo("RL_ATOR_PAPEL_PERFIL", "CO_PERFIL", 0L));
30:         tableSet.add(new TableInfo("RL_CDS_ATEND_DOM_PROCED", "CO_CDS_ATEND_DOMICILIAR", 0L));
31:         tableSet.add(new TableInfo("RL_CDS_ATEND_DOM_PROCED", "CO_PROCED", 0L));
32:         tableSet.add(new TableInfo("RL_CDS_ATEND_DOM_SITUACAO_PRES", "CO_CDS_ATEND_DOMICILI
AR", 0L));
33:         tableSet.add(new TableInfo("RL_CDS_ATEND_DOM_SITUACAO_PRES", "CO_CDS_SITUACAO_PRESE
NTE", 0L));
34:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_CIAP", "CO_CDS_ATEND_INDIVIDUAL",
0L));
35:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_CIAP", "CO_CIAP", 0L));
36:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_CONDUT", "CO_CDS_ATEND_INDIVIDUA
L", 0L));
37:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_CONDUT", "TP_CDS_CONDUTA", 0L));
38:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_EXAME", "CO_CDS_ATEND_INDIVIDUAL
", 0L));
39:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_EXAME", "CO_EXAME", 0L));
40:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_EXAME", "ST_SOLICITADO_AVALIADO",
 0L));
41:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_NASF", "CO_CDS_ATEND_INDIVIDUAL",
0L));
42:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_NASF", "TP_CDS_ATEND_NASF", 0L));
43:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_PROCED", "CO_CDS_ATEND_ODONTO", 0L));
44:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_PROCED", "CO_PROCED", 0L));
45:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_CNSLT", "CO_CDS_ATEND_ODONTO",
0L));
46:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_CNSLT", "TP_CONSULTA_ODONTO", 0
L));
47:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_ENCAM", "CO_CDS_ATEND_ODONTO",
 0L));
48:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_ENCAM", "TP_ENCAM_ODONTO", 0L));
49:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_FORNC", "CO_CDS_ATEND_ODONTO",
0L));
50:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_FORNC", "TP_FORNECIMENTO_ODON
TO", 0L));
51:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONT_TIP_VIG_BUC", "CO_CDS_ATEND_ODONTO", 0
L));
52:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONT_TIP_VIG_BUC", "TP_CDS_VIG_SAUDE_BUCAL
```

```
", 0L));
    53:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATEND_INDIVDL_PRF", "CO_CDS_FICHA_ATEND_INDI
VIDUAL", 0L));
    54:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATEND_INDIVDL_PRF", "CO_CDS_PROF", 0L));
    55:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATEND_ODONTO_PROF", "CO_CDS_FICHA_ATEND_OD
ONTO", 0L));
    56:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATEND_ODONTO_PROF", "CO_CDS_PROF", 0L));
    57:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PRATICA", "CO_CDS_ATIV_COL_PRATICA",
0L));
    58:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PRATICA", "CO_CDS_FICHA_ATIV_COL", 0L
));
    59:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PROF", "CO_CDS_FICHA_ATIV_COL", 0L));
    60:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PROF", "CO_CDS_PROF", 0L));
    61:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PUB_ALVO", "CO_CDS_ATIV_COL_PUBLIC
O_ALVO", 0L));
    62:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PUB_ALVO", "CO_CDS_FICHA_ATIV_COL",
0L));
    63:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_TEMA", "CO_CDS_ATIV_COL_TEMA", 0L));
    64:         tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_TEMA", "CO_CDS_FICHA_ATIV_COL", 0L));
    65:         tableSet.add(new TableInfo("RL_CDS_PROCED_ADM_MEDICAMENTO", "CO_CDS_ADM_MEDICAME
NTO", 0L));
    66:         tableSet.add(new TableInfo("RL_CDS_PROCED_ADM_MEDICAMENTO", "CO_CDS_PROCED", 0L));
    67:         tableSet.add(new TableInfo("RL_CDS_PROCED_ADM_MEDICAMENTO", "CO_PROCED", 0L));
    68:         tableSet.add(new TableInfo("RL_CDS_PRONTUARIO_UNIDADE_SAUD", "CO_ATOR_PAPEL", 0L));
    69:         tableSet.add(new TableInfo("RL_CDS_PRONTUARIO_UNIDADE_SAUD", "CO_PRONTUARIO", 0L));
    70:         tableSet.add(new TableInfo("RL_CDS_VISITA_DOM_MOTIVO", "CO_CDS_VISITA_DOMICILIAR", 0L));
    71:         tableSet.add(new TableInfo("RL_CDS_VISITA_DOM_MOTIVO", "CO_CDS_VISITA_DOM_MOTIVO", 0L)
);
    72:         tableSet.add(new TableInfo("RL_CIAP_CID10", "CO_CIAP", 751L));
    73:         tableSet.add(new TableInfo("RL_CIAP_CID10", "CO_CID10", 14230L));
    74:         tableSet.add(new TableInfo("RL_EVOLUCAO_AVALIACAO_CIAP_CID", "CO_SEQ_EVOLUCAO_AVAL_
CIAP_CID", 0L));
    75:         tableSet.add(new TableInfo("RL_EVOLUCAO_OBJETIVO_MEDICAO", "CO_ATEND_PROF", 0L));
    76:         tableSet.add(new TableInfo("RL_EVOLUCAO_OBJETIVO_MEDICAO", "CO_MEDICAO", 0L));
    77:         tableSet.add(new TableInfo("RL_EVOLUCAO_ODONTO_PARTE_BUCAL", "CO_EVOLUCAO_ODONTO"
, 0L));
    78:         tableSet.add(new TableInfo("RL_EVOLUCAO_ODONTO_PARTE_BUCAL", "CO_PARTE_BUCAL", 0L));
    79:         tableSet.add(new TableInfo("RL_EVOLUCAO_ODONTO_PROCED", "CO_EVOLUCAO_ODONTO", 0L));
    80:         tableSet.add(new TableInfo("RL_EVOLUCAO_ODONTO_PROCED", "CO_PROCED", 0L));
    81:         tableSet.add(new TableInfo("RL_EVOLUCAO_PLANO_CIAP", "CO_SEQ_EVOLUCAO_PLANO_CIAP", 0L
));
    82:         tableSet.add(new TableInfo("RL_EVOLUCAO_SUBJETIVO_CIAP", "CO_SEQ_EVOLUCAO_SUBJETIVO_
CIAP", 0L));
    83:         tableSet.add(new TableInfo("RL_EVOL_AVAL_TP_VIG_SAUDE_BUCL", "CO_EVOLUCAO_AVALIACA
O", 0L));
    84:         tableSet.add(new TableInfo("RL_EVOL_AVAL_TP_VIG_SAUDE_BUCL", "CO_TIPO_VIG_SAUDE_BUCA
L", 0L));
    85:         tableSet.add(new TableInfo("RL_MEDICAO_TIPO_MEDICAO", "CO_SEQ_MEDICAO_TIPO_MEDICAO",
0L));
    86:         tableSet.add(new TableInfo("RL_PERFIL_CBO_PADRAO", "CO_CBO", 2609L));
    87:         tableSet.add(new TableInfo("RL_PESSOA_ENDERECO", "CO_ATOR", 0L));
    88:         tableSet.add(new TableInfo("RL_PESSOA_ENDERECO", "CO_ENDERECO", 0L));
    89:         tableSet.add(new TableInfo("RL_PROCED_ATRIBUTO_COMPLEM", "CO_ATRIBUTO_COMPLEM", 41L))
;
    90:         tableSet.add(new TableInfo("RL_PROCED_ATRIBUTO_COMPLEM", "CO_PROCED", 4746L));
    91:         tableSet.add(new TableInfo("RL_PROCED_CBO", "CO_CBO", 2608L));
    92:         tableSet.add(new TableInfo("RL_PROCED_CBO", "CO_PROCED", 4744L));
    93:         tableSet.add(new TableInfo("RL_PROCED_CDS_PROCED", "CO_CDS_PROCED", 0L));
    94:         tableSet.add(new TableInfo("RL_PROCED_CDS_PROCED", "CO_PROCED", 0L));
    95:         tableSet.add(new TableInfo("RL_PROCED_CID10", "CO_CID10", 12444L));
    96:         tableSet.add(new TableInfo("RL_PROCED_CID10", "CO_PROCED", 4746L));
    97:         tableSet.add(new TableInfo("RL_PROCED_EXAME_DETALHE", "CO_SEQ_PROCED_EXAME_DETALHE
", 1L));
    98:         tableSet.add(new TableInfo("RL_PROCED_REGRA_CONDICIONADA", "CO_PROCED", 4633L));
    99:         tableSet.add(new TableInfo("RL_PROCED_REGRA_CONDICIONADA", "CO_REGRA_CONDICIONADA",
 9L));
   100:         tableSet.add(new TableInfo("RL_PROCED_TIPO_FILTRO_PROCED", "CO_PROCED", 4529L));
   101:         tableSet.add(new TableInfo("RL_PROCED_TIPO_FILTRO_PROCED", "TP_FILTRO_PROCED", 4L));
```

```
    102:        tableSet.add(new TableInfo("RL_PROCED_TIPO_REGISTRO", "CO_PROCED", 4746L));
    103:        tableSet.add(new TableInfo("RL_PROCED_TIPO_REGISTRO", "TP_REGISTRO", 9L));
    104:        tableSet.add(new TableInfo("RL_PROF_MUNICIPIO", "CO_ATOR_PAPEL", 0L));
    105:        tableSet.add(new TableInfo("RL_PROF_MUNICIPIO", "CO_LOCALIDADE", 0L));
    106:        tableSet.add(new TableInfo("RL_QST_ASSOCIACAO_OPCAO_PERGNT", "CO_QST_ASSOCIACAO_PER
GUNTA", 3L));
    107:        tableSet.add(new TableInfo("RL_QST_ASSOCIACAO_OPCAO_PERGNT", "CO_QST_OPCAO_PERGUNT
A", 70L));
    108:        tableSet.add(new TableInfo("RL_QST_RESPOSTA_OPCAO_PERGUNTA", "CO_QST_OPCAO_PERGUNT
A", 0L));
    109:        tableSet.add(new TableInfo("RL_QST_RESPOSTA_OPCAO_PERGUNTA", "CO_QST_RESPOSTA", 0L));
    110:        tableSet.add(new TableInfo("RL_RECURSO_ACAO", "CO_RECURSO", 0L));
    111:        tableSet.add(new TableInfo("RL_RECURSO_ACAO", "NU_ACAO", 0L));
    112:        tableSet.add(new TableInfo("RL_RECURSO_ACAO", "NU_OBRIGACAO", 0L));
    113:        tableSet.add(new TableInfo("RL_RECURSO_TIPO_PERFIL", "CO_RECURSO", 0L));
    114:        tableSet.add(new TableInfo("RL_RECURSO_TIPO_PERFIL", "TP_PERFIL", 0L));
    115:        tableSet.add(new TableInfo("RL_REL_OP_CRIANCA_CID10", "CO_SEQ_REL_OP_CRIANCA_CID10", 0L)
);
    116:        tableSet.add(new TableInfo("RL_TIPO_ATEND_PROCED_AUTOMATIC", "CO_PROCED_AUTOMATICO"
, 132L));
    117:        tableSet.add(new TableInfo("RL_TIPO_ATEND_PROCED_AUTOMATIC", "TP_ATEND", 6L));
    118:        tableSet.add(new TableInfo("RL_TIPO_CONSULTA_ODNT_PRC_AUTO", "CO_PROCED_AUTOMATICO"
, 151L));
    119:        tableSet.add(new TableInfo("RL_TIPO_CONSULTA_ODNT_PRC_AUTO", "TP_CONSULTA_ODONTO", 4
L));
    120:        tableSet.add(new TableInfo("RL_TIPO_ENCAM_ODONTO_PROCD_AUT", "CO_PROCED_AUTOMATICO
", 52L));
    121:        tableSet.add(new TableInfo("RL_TIPO_ENCAM_ODONTO_PROCD_AUT", "TP_ENCAM_ODONTO", 15L)
);
    122:        tableSet.add(new TableInfo("RL_TIPO_PERFIL_TIPO_INSTALACAO", "NO_TIPO_INSTALACAO", 0L));
    123:        tableSet.add(new TableInfo("RL_TIPO_PERFIL_TIPO_INSTALACAO", "TP_PERFIL", 0L));
    124:        tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_COMPLEXIDADE", "CO_ATOR_PAPEL", 0L));
    125:        tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_COMPLEXIDADE", "CO_COMPLEXIDADE", 0L));
    126:        tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_HORUS", "CO_UNIDADE_SAUDE", 0L));
    127:        tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_HORUS", "CO_UNIDADE_SAUDE_HORUS", 0L));
    128:        tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_LOCALIDADE", "CO_ATOR_PAPEL", 0L));
    129:        tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_LOCALIDADE", "CO_LOCALIDADE", 0L));
    130:        tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_TIPO_SERVICO", "CO_ATOR_PAPEL", 0L));
    131:        tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_TIPO_SERVICO", "TP_SERVICO", 0L));
    132:        tableSet.add(new TableInfo("RL_VIA_ADM_LOCAL_APL_VACINA", "CO_LOCAL_APL_VACINA", 21L))
;
    133:        tableSet.add(new TableInfo("RL_VIA_ADM_LOCAL_APL_VACINA", "CO_VIA_ADM_VACINA", 6L));
    134:        tableSet.add(new TableInfo("TB_ACESSO", "CO_PERFIL", 0L));
    135:        tableSet.add(new TableInfo("TB_ACESSO", "CO_RECURSO", 0L));
    136:        tableSet.add(new TableInfo("TB_ACESSO", "NU_ACAO", 0L));
    137:        tableSet.add(new TableInfo("TB_ACESSO", "NU_OBRIGACAO", 0L));
    138:        tableSet.add(new TableInfo("TB_ADM_GERAL", "CO_ATOR_PAPEL", 0L));
    139:        tableSet.add(new TableInfo("TB_ADM_MUNICIPAL", "CO_ATOR_PAPEL", 0L));
    140:        tableSet.add(new TableInfo("TB_ADM_MUNICIPAL_CENTRALIZADOR", "CO_ATOR_PAPEL", 0L));
    141:        tableSet.add(new TableInfo("TB_AD_CIDADAO", "CO_SEQ_AD_CIDADAO", 0L));
    142:        tableSet.add(new TableInfo("TB_AD_CIDADAO_HISTORICO", "CO_SEQ_AD_CIDADAO_HISTORICO", 0
L));
    143:        tableSet.add(new TableInfo("TB_AD_DESTINO", "CO_SEQ_AD_DESTINO", 9L));
    144:        tableSet.add(new TableInfo("TB_AD_MODALIDADE", "CO_SEQ_AD_MODALIDADE", 5L));
    145:        tableSet.add(new TableInfo("TB_AD_ORIGEM", "CO_AD_ORIGEM", 16L));
    146:        tableSet.add(new TableInfo("TB_AD_QUESTAO", "CO_SEQ_AD_QUESTAO", 0L));
    147:        tableSet.add(new TableInfo("TB_AD_RESPOSTA", "CO_SEQ_AD_RESPOSTA", 0L));
    148:        tableSet.add(new TableInfo("TB_AD_SYNC_ENTITY", "CO_SEQ_AD_SYNC_ENTITY", 0L));
    149:        tableSet.add(new TableInfo("TB_AD_TIPO_ELEGIVEL", "CO_AD_TIPO_ELEGIVEL", 4L));
    150:        tableSet.add(new TableInfo("TB_AD_TIPO_INELEGIVEL", "CO_AD_TIPO_INELEGIVEL", 5L));
    151:        tableSet.add(new TableInfo("TB_AD_TRANSMISSAO_SESSAO", "CO_UNICO_TRANSMISSAO_SESSAO"
, 0L));
    152:        tableSet.add(new TableInfo("TB_AGENDADO", "CO_SEQ_AGENDADO", 0L));
    153:        tableSet.add(new TableInfo("TB_ALERGIA", "CO_SEQ_ALERGIA", 0L));
    154:        tableSet.add(new TableInfo("TB_ALERGIA_EVOLUCAO", "CO_SEQ_ALERGIA_EVOLUCAO", 0L));
    155:        tableSet.add(new TableInfo("TB_ALIM_BEBIDA", "CO_ALIM_BEBIDA", 10L));
    156:        tableSet.add(new TableInfo("TB_ANTECEDENTE", "CO_PRONTUARIO", 0L));
    157:        tableSet.add(new TableInfo("TB_ANTECEDENTE_HISTORICO", "CO_SEQ_ANTECEDENTE_HISTORICO
```

```
", 0L));
   158:         tableSet.add(new TableInfo("TB_ANTECEDENTE_ITEM", "CO_SEQ_ANTECEDENTE_ITEM", 0L));
   159:         tableSet.add(new TableInfo("TB_ANTECEDENTE_TIPO_ITEM", "CO_ANTECEDENTE_TIPO_ITEM", 28L)
);
   160:         tableSet.add(new TableInfo("TB_APLICACAO_MEDICAMENTO", "CO_APLICACAO_MEDICAMENTO",
27L));
   161:         tableSet.add(new TableInfo("TB_ARCADA", "CO_PARTE_BUCAL", 1L));
   162:         tableSet.add(new TableInfo("TB_ATEND", "CO_SEQ_ATEND", 0L));
   163:         tableSet.add(new TableInfo("TB_ATEND_PROF", "CO_SEQ_ATEND_PROF", 0L));
   164:         tableSet.add(new TableInfo("TB_ATEND_PROF_AD", "CO_ATEND_PROF_AD", 0L));
   165:         tableSet.add(new TableInfo("TB_ATEND_PROF_AD_QST", "CO_SEQ_ATEND_PROF_AD_QST", 0L));
   166:         tableSet.add(new TableInfo("TB_ATEND_PROF_ODONTO", "CO_ATEND_PROF_ODONTO", 0L));
   167:         tableSet.add(new TableInfo("TB_ATEND_PROF_PRE_NATAL", "CO_ATEND_PROF_PRE_NATAL", 0L));
   168:         tableSet.add(new TableInfo("TB_ATEND_PROF_PUERICULTURA", "CO_ATEND_PROF_PUERICULTUR
A", 0L));
   169:         tableSet.add(new TableInfo("TB_ATEND_PROF_TIPO_ENCAM_INTRN", "CO_SEQ_ATEND_PROF_TIPO
_ENC_INT", 0L));
   170:         tableSet.add(new TableInfo("TB_ATEND_PROF_VACINACAO", "CO_ATEND_PROF_VACINACAO", 0L))
;
   171:         tableSet.add(new TableInfo("TB_ATESTADO", "CO_SEQ_ATESTADO", 0L));
   172:         tableSet.add(new TableInfo("TB_ATIVACAO_AGENDAMENTO_ONLINE", "CO_SEQ_ATIVACAO_AGEN
DAMENTO_ON", 0L));
   173:         tableSet.add(new TableInfo("TB_ATOR", "CO_SEQ_ATOR", 0L));
   174:         tableSet.add(new TableInfo("TB_ATOR_PAPEL", "CO_SEQ_ATOR_PAPEL", 0L));
   175:         tableSet.add(new TableInfo("TB_ATRIBUTO_COMPLEM", "CO_ATRIBUTO_COMPLEM", 41L));
   176:         tableSet.add(new TableInfo("TB_BAIRRO", "CO_BAIRRO", 50685L));
   177:         tableSet.add(new TableInfo("TB_CATEGORIA_AGENTE_CAUSADOR", "CO_CATEGORIA_AGENTE_CA
USADOR", 6L));
   178:         tableSet.add(new TableInfo("TB_CBO", "CO_CBO", 2609L));
   179:         tableSet.add(new TableInfo("TB_CBO_ATEND", "CO_SEQ_CBO_ATEND", 0L));
   180:         tableSet.add(new TableInfo("TB_CDS_ADM_MEDICAMENTO", "CO_CDS_ADM_MEDICAMENTO", 4426
L));
   181:         tableSet.add(new TableInfo("TB_CDS_ALEITAMENTO_MATERNO", "CO_CDS_ALEITAMENTO_MATE
RNO", 4L));
   182:         tableSet.add(new TableInfo("TB_CDS_ATEND_DOMICILIAR", "CO_SEQ_CDS_ATEND_DOMICILIAR", 0
L));
   183:         tableSet.add(new TableInfo("TB_CDS_ATEND_INDIVIDUAL", "CO_SEQ_CDS_ATEND_INDIVIDUAL", 0
L));
   184:         tableSet.add(new TableInfo("TB_CDS_ATEND_ODONTO", "CO_SEQ_CDS_ATEND_ODONTO", 0L));
   185:         tableSet.add(new TableInfo("TB_CDS_ATIV_COL_PARTICIPANTE", "CO_SEQ_CDS_ATIV_COL_PARTI
CIPNT", 0L));
   186:         tableSet.add(new TableInfo("TB_CDS_ATIV_COL_PRATICA", "CO_CDS_ATIV_COL_PRATICA", 30L));
   187:         tableSet.add(new TableInfo("TB_CDS_ATIV_COL_PUBLICO_ALVO", "CO_CDS_ATIV_COL_PUBLICO_
ALVO", 18L));
   188:         tableSet.add(new TableInfo("TB_CDS_ATIV_COL_TEMA", "CO_CDS_ATIV_COL_TEMA", 7L));
   189:         tableSet.add(new TableInfo("TB_CDS_AVAL_ELEGIBILIDADE", "CO_SEQ_CDS_AVAL_ELEGIBILIDAD
E", 0L));
   190:         tableSet.add(new TableInfo("TB_CDS_CAD_DOMICILIAR", "CO_SEQ_CDS_CAD_DOMICILIAR", 0L));
   191:         tableSet.add(new TableInfo("TB_CDS_CAD_INDIVIDUAL", "CO_SEQ_CDS_CAD_INDIVIDUAL", 0L));
   192:         tableSet.add(new TableInfo("TB_CDS_CIDADAO_RESPOSTA", "CO_SEQ_CDS_CIDADAO_RESPOSTA",
0L));
   193:         tableSet.add(new TableInfo("TB_CDS_DOMICILIO", "CO_SEQ_CDS_DOMICILIO", 0L));
   194:         tableSet.add(new TableInfo("TB_CDS_DOMICILIO_FAMILIA", "CO_SEQ_CDS_DOMICILIO_FAMILIA",
0L));
   195:         tableSet.add(new TableInfo("TB_CDS_DOMICILIO_RESPOSTA", "CO_SEQ_CDS_DOMICILIO_RESPOST
A", 0L));
   196:         tableSet.add(new TableInfo("TB_CDS_FICHA_ATEND_DOMICILIAR", "CO_SEQ_CDS_FICHA_ATEND_
DOM", 0L));
   197:         tableSet.add(new TableInfo("TB_CDS_FICHA_ATEND_INDIVIDUAL", "CO_SEQ_CDS_FICHA_ATEND_I
NDIVDL", 0L));
   198:         tableSet.add(new TableInfo("TB_CDS_FICHA_ATEND_ODONTO", "CO_SEQ_CDS_FICHA_ATEND_ODO
NTO", 0L));
   199:         tableSet.add(new TableInfo("TB_CDS_FICHA_ATIV_COL", "CO_SEQ_CDS_FICHA_ATIV_COL", 0L));
   200:         tableSet.add(new TableInfo("TB_CDS_FICHA_CONSUMO_ALIMENTAR", "CO_SEQ_CDS_FICHA_CONS
UMO_ALIM", 0L));
   201:         tableSet.add(new TableInfo("TB_CDS_FICHA_PROCED", "CO_SEQ_CDS_FICHA_PROCED", 0L));
   202:         tableSet.add(new TableInfo("TB_CDS_FICHA_VACINACAO", "CO_SEQ_CDS_FICHA_VACINACAO", 0L)
);
```

```
   203:         tableSet.add(new TableInfo("TB_CDS_FICHA_VISITA_DOMICILIAR", "CO_SEQ_CDS_FICHA_VISITA_D
OM", 0L));
   204:         tableSet.add(new TableInfo("TB_CDS_FICHA_ZIKA_MICROCEFALIA", "CO_SEQ_CDS_FICHA_ZICA_M
ICRCFL", 0L));
   205:         tableSet.add(new TableInfo("TB_CDS_PIC", "CO_CDS_PIC", 8L));
   206:         tableSet.add(new TableInfo("TB_CDS_PROCED", "CO_SEQ_CDS_PROCED", 0L));
   207:         tableSet.add(new TableInfo("TB_CDS_PROF", "CO_SEQ_CDS_PROF", 0L));
   208:         tableSet.add(new TableInfo("TB_CDS_TIPO_ATEND_NASF", "CO_CDS_TIPO_ATEND_NASF", 3L));
   209:         tableSet.add(new TableInfo("TB_CDS_TIPO_ATIV_COL", "CO_CDS_TIPO_ATIV_COL", 7L));
   210:         tableSet.add(new TableInfo("TB_CDS_TIPO_CONDUTA", "CO_CDS_TIPO_CONDUTA", 12L));
   211:         tableSet.add(new TableInfo("TB_CDS_TIPO_CUIDADOR", "CO_CDS_TIPO_CUIDADOR", 8L));
   212:         tableSet.add(new TableInfo("TB_CDS_TIPO_IMOVEL", "CO_CDS_TIPO_IMOVEL", 99L));
   213:         tableSet.add(new TableInfo("TB_CDS_TIPO_ORIGEM", "CO_CDS_TIPO_ORIGEM", 4L));
   214:         tableSet.add(new TableInfo("TB_CDS_TIPO_SITUACAO_PRESENTE", "CO_CDS_TIPO_SITUACAO_PRE
SENTE", 24L));
   215:         tableSet.add(new TableInfo("TB_CDS_TIPO_VIG_SAUDE_BUCAL", "CO_CDS_TIPO_VIG_SAUDE_BUC
AL", 99L));
   216:         tableSet.add(new TableInfo("TB_CDS_TURNO", "CO_CDS_TURNO", 3L));
   217:         tableSet.add(new TableInfo("TB_CDS_VACINA", "CO_SEQ_CDS_VACINA", 0L));
   218:         tableSet.add(new TableInfo("TB_CDS_VACINACAO", "CO_SEQ_CDS_VACINACAO", 0L));
   219:         tableSet.add(new TableInfo("TB_CDS_VISITA_DOMICILIAR", "CO_SEQ_CDS_VISITA_DOMICILIAR", 0
L));
   220:         tableSet.add(new TableInfo("TB_CDS_VISITA_DOM_DESFECHO", "CO_CDS_VISITA_DOM_DESFECHO
", 3L));
   221:         tableSet.add(new TableInfo("TB_CDS_VISITA_DOM_MOTIVO", "CO_CDS_VISITA_DOM_MOTIVO", 37L
));
   222:         tableSet.add(new TableInfo("TB_CIAP", "CO_SEQ_CIAP", 751L));
   223:         tableSet.add(new TableInfo("TB_CIAP_CAPITULO", "CO_SEQ_CIAP_CAPITULO", 18L));
   224:         tableSet.add(new TableInfo("TB_CIAP_COMPONENTE", "CO_CIAP_COMPONENTE", 7L));
   225:         tableSet.add(new TableInfo("TB_CIAP_DAB", "CO_CIAP", 751L));
   226:         tableSet.add(new TableInfo("TB_CIAP_MS", "CO_CIAP", 726L));
   227:         tableSet.add(new TableInfo("TB_CID10", "CO_CID10", 14232L));
   228:         tableSet.add(new TableInfo("TB_CIDADAO", "CO_ATOR_PAPEL", 0L));
   229:         tableSet.add(new TableInfo("TB_CIDADAO_NUCLEO_FAMILIAR", "CO_SEQ_CIDADAO_NUCLEO_FA
MILIAR", 0L));
   230:         tableSet.add(new TableInfo("TB_CLASSE_IMUNOBIOLOGICO", "CO_CLASSE_IMUNOBIOLOGICO", 3L)
);
   231:         tableSet.add(new TableInfo("TB_CLASSIFICACAO_RISCO", "CO_CLASSIFICACAO_RISCO", 9L));
   232:         tableSet.add(new TableInfo("TB_CLASSIFICACAO_RISCO_ENCAM", "CO_CLASSIFICACAO_RISCO_EN
CAM", 3L));
   233:         tableSet.add(new TableInfo("TB_CNS", "CO_SEQ_CNS", 0L));
   234:         tableSet.add(new TableInfo("TB_COMPARTILHAMENTO_PRONTUARIO", "CO_SEQ_COMPARTILHA_P
RONTUARIO", 0L));
   235:         tableSet.add(new TableInfo("TB_COMPLEXIDADE", "CO_COMPLEXIDADE", 6L));
   236:         tableSet.add(new TableInfo("TB_CONFIGURACAO_HORARIO", "CO_SEQ_CONFIG_PERIODO", 0L));
   237:         tableSet.add(new TableInfo("TB_CONFIG_AGENDA_DETALHE", "CO_SEQ_CONFIG_AGENDA_DETAL
HE", 0L));
   238:         tableSet.add(new TableInfo("TB_CONFIG_AGENDA_FECHAMENTO", "CO_SEQ_CONFIG_AGENDA_FE
CHAMENT", 0L));
   239:         tableSet.add(new TableInfo("TB_CONFIG_ATENCAO_DOMICILIAR", "CO_SEQ_CONFIG_ATENCAO_D
OMICILR", 0L));
   240:         tableSet.add(new TableInfo("TB_CONFIG_ATEND_DOMICILIAR", "CO_SEQ_CONFIG_ATEND_DOMICI
LIAR", 0L));
   241:         tableSet.add(new TableInfo("TB_CONSELHO_CLASSE", "CO_CONSELHO_CLASSE", 83L));
   242:         tableSet.add(new TableInfo("TB_CONTATO", "CO_SEQ_CONTATO", 0L));
   243:         tableSet.add(new TableInfo("TB_CONTEXTO_PERGUNTA", "CO_CONTEXTO_PERGUNTA", 2L));
   244:         tableSet.add(new TableInfo("TB_CRITICIDADE_ALERGIA", "CO_CRITICIDADE_ALERGIA", 2L));
   245:         tableSet.add(new TableInfo("TB_CRONICIDADE", "CO_CRONICIDADE", 2L));
   246:         tableSet.add(new TableInfo("TB_DADO_RECEBIDO_COMPETENCIA", "CO_SEQ_DADO_RECEBIDO_C
OMPETENC", 0L));
   247:         tableSet.add(new TableInfo("TB_DADO_RECEBIDO_INFO_INSTALAC", "CO_SEQ_DADO_RECEBIDO_I
NFO_INST", 0L));
   248:         tableSet.add(new TableInfo("TB_DADO_REL_PROCESSAMENTO", "CO_SEQ_DADO_REL_PROCESSAM
ENTO", 0L));
   249:         tableSet.add(new TableInfo("TB_DADO_TRANSP", "CO_SEQ_DADO_TRANSP", 0L));
   250:         tableSet.add(new TableInfo("TB_DADO_TRANSP_RECEBIDO", "CO_SEQ_DADO_TRANSP_RECEBIDO",
 0L));
   251:         tableSet.add(new TableInfo("TB_DADO_TRANSP_RECEBIDO_ONLINE", "CO_SEQ_DADO_TRANSP_RE
```

```
CEB_ONLIN", 0L));
    252:        tableSet.add(new TableInfo("TB_DENTE", "CO_PARTE_BUCAL", 59L));
    253:        tableSet.add(new TableInfo("TB_DIA_SEMANA", "CO_DIA_SEMANA", 6L));
    254:        tableSet.add(new TableInfo("TB_DIM_AGRUPADOR_FILTRO", "CO_SEQ_DIM_AGRUPADOR_FILTRO",
0L));
    255:        tableSet.add(new TableInfo("TB_DIM_ALEITAMENTO", "CO_SEQ_DIM_ALEITAMENTO", 5L));
    256:        tableSet.add(new TableInfo("TB_DIM_CBO", "CO_SEQ_DIM_CBO", 0L));
    257:        tableSet.add(new TableInfo("TB_DIM_CIAP", "CO_SEQ_DIM_CIAP", 0L));
    258:        tableSet.add(new TableInfo("TB_DIM_CID", "CO_SEQ_DIM_CID", 0L));
    259:        tableSet.add(new TableInfo("TB_DIM_CUIDADOR", "CO_SEQ_DIM_CUIDADOR", 8L));
    260:        tableSet.add(new TableInfo("TB_DIM_DESFECHO_VISITA", "CO_SEQ_DIM_DESFECHO_VISITA", 4L));
    261:        tableSet.add(new TableInfo("TB_DIM_DOSE_IMUNOBIOLOGICO", "CO_SEQ_DIM_DOSE_IMUNOBIOL
OGICO", 38L));
    262:        tableSet.add(new TableInfo("TB_DIM_EQUIPE", "CO_SEQ_DIM_EQUIPE", 0L));
    263:        tableSet.add(new TableInfo("TB_DIM_ESTRATEGIA_VACINACAO", "CO_SEQ_DIM_ESTRATEGIA_VA
CNACAO", 10L));
    264:        tableSet.add(new TableInfo("TB_DIM_ETNIA", "CO_SEQ_DIM_ETNIA", 265L));
    265:        tableSet.add(new TableInfo("TB_DIM_FAIXA_ETARIA", "CO_SEQ_DIM_FAIXA_ETARIA", 22L));
    266:        tableSet.add(new TableInfo("TB_DIM_FREQUENCIA_ALIMENTACAO", "CO_SEQ_DIM_FREQUENCIA_
ALIMENT", 4L));
    267:        tableSet.add(new TableInfo("TB_DIM_GRUPO_CBO", "CO_SEQ_DIM_GRUPO_CBO", 0L));
    268:        tableSet.add(new TableInfo("TB_DIM_IDENTIDADE_GENERO", "CO_SEQ_DIM_IDENTIDADE_GENERO
", 5L));
    269:        tableSet.add(new TableInfo("TB_DIM_IMUNOBIOLOGICO", "CO_SEQ_DIM_IMUNOBIOLOGICO", 0L));
    270:        tableSet.add(new TableInfo("TB_DIM_LOCAL_ATENDIMENTO", "CO_SEQ_DIM_LOCAL_ATENDIMEN
TO", 16L));
    271:        tableSet.add(new TableInfo("TB_DIM_MODALIDADE_AD", "CO_SEQ_DIM_MODALIDADE_AD", 5L));
    272:        tableSet.add(new TableInfo("TB_DIM_MUNICIPIO", "CO_SEQ_DIM_MUNICIPIO", 0L));
    273:        tableSet.add(new TableInfo("TB_DIM_NACIONALIDADE", "CO_SEQ_DIM_NACIONALIDADE", 4L));
    274:        tableSet.add(new TableInfo("TB_DIM_PAIS", "CO_SEQ_DIM_PAIS", 239L));
    275:        tableSet.add(new TableInfo("TB_DIM_PIC", "CO_SEQ_DIM_PIC", 9L));
    276:        tableSet.add(new TableInfo("TB_DIM_PROCEDENCIA_ORIGEM", "CO_SEQ_DIM_PROCEDENCIA_ORI
GEM", 6L));
    277:        tableSet.add(new TableInfo("TB_DIM_PROCEDIMENTO", "CO_SEQ_DIM_PROCEDIMENTO", 0L));
    278:        tableSet.add(new TableInfo("TB_DIM_PROFISSIONAL", "CO_SEQ_DIM_PROFISSIONAL", 0L));
    279:        tableSet.add(new TableInfo("TB_DIM_RACA_COR", "CO_SEQ_DIM_RACA_COR", 6L));
    280:        tableSet.add(new TableInfo("TB_DIM_RACIONALIDADE_SAUDE", "CO_SEQ_DIM_RACIONALIDADE_
SAUDE", 7L));
    281:        tableSet.add(new TableInfo("TB_DIM_SEXO", "CO_SEQ_DIM_SEXO", 4L));
    282:        tableSet.add(new TableInfo("TB_DIM_SITUACAO_TRABALHO", "CO_SEQ_DIM_SITUACAO_TRABALH
O", 12L));
    283:        tableSet.add(new TableInfo("TB_DIM_TEMPO", "CO_SEQ_DIM_TEMPO", 0L));
    284:        tableSet.add(new TableInfo("TB_DIM_TEMPO_MORADOR_RUA", "CO_SEQ_DIM_TEMPO_MORADOR_
RUA", 5L));
    285:        tableSet.add(new TableInfo("TB_DIM_TIPO_ABASTECIMENTO_AGUA", "CO_SEQ_DIM_TIPO_ABASTE
C_AGUA", 6L));
    286:        tableSet.add(new TableInfo("TB_DIM_TIPO_ACESSO_DOMICILIO", "CO_SEQ_DIM_TIPO_ACESSO_DO
MICIL", 5L));
    287:        tableSet.add(new TableInfo("TB_DIM_TIPO_ATENDIMENTO", "CO_SEQ_DIM_TIPO_ATENDIMENTO",
11L));
    288:        tableSet.add(new TableInfo("TB_DIM_TIPO_ATIVIDADE", "CO_SEQ_DIM_TIPO_ATIVIDADE", 8L));
    289:        tableSet.add(new TableInfo("TB_DIM_TIPO_CONDICAO_PESO", "CO_SEQ_DIM_TIPO_CONDICAO_PES
O", 4L));
    290:        tableSet.add(new TableInfo("TB_DIM_TIPO_CONSULTA", "CO_SEQ_DIM_TIPO_CONSULTA", 6L));
    291:        tableSet.add(new TableInfo("TB_DIM_TIPO_DESTINO_LIXO", "CO_SEQ_DIM_TIPO_DESTINO_LIXO", 5
L));
    292:        tableSet.add(new TableInfo("TB_DIM_TIPO_DOMICILIO", "CO_SEQ_DIM_TIPO_DOMICILIO", 5L));
    293:        tableSet.add(new TableInfo("TB_DIM_TIPO_ELEGIBILIDADE", "CO_SEQ_DIM_TIPO_ELEGIBILIDADE",
 4L));
    294:        tableSet.add(new TableInfo("TB_DIM_TIPO_ESCOAMENTO_SANITAR", "CO_SEQ_DIM_TIPO_ESCOAM
ENTO_SNT", 7L));
    295:        tableSet.add(new TableInfo("TB_DIM_TIPO_ESCOLARIDADE", "CO_SEQ_DIM_TIPO_ESCOLARIDADE"
, 16L));
    296:        tableSet.add(new TableInfo("TB_DIM_TIPO_FICHA", "CO_SEQ_DIM_TIPO_FICHA", 15L));
    297:        tableSet.add(new TableInfo("TB_DIM_TIPO_IMOVEL", "CO_SEQ_DIM_TIPO_IMOVEL", 13L));
    298:        tableSet.add(new TableInfo("TB_DIM_TIPO_LOCALIZACAO", "CO_SEQ_DIM_TIPO_LOCALIZACAO", 3
L));
    299:        tableSet.add(new TableInfo("TB_DIM_TIPO_LOGRADOURO", "CO_SEQ_DIM_TIPO_LOGRADOURO", 28
```

```
2L));
   300:         tableSet.add(new TableInfo("TB_DIM_TIPO_MATERIAL_PAREDE", "CO_SEQ_DIM_TIPO_MATERIAL_P
ARED", 9L));
   301:         tableSet.add(new TableInfo("TB_DIM_TIPO_ORIENTACAO_SEXUAL", "CO_SEQ_DIM_TIPO_ORIENTA
CAO_SXL", 5L));
   302:         tableSet.add(new TableInfo("TB_DIM_TIPO_ORIGEM_DADO_TRANSP", "CO_SEQ_DIM_TP_ORGM_DA
DO_TRANSP", 6L));
   303:         tableSet.add(new TableInfo("TB_DIM_TIPO_PARENTESCO", "CO_SEQ_DIM_TIPO_PARENTESCO", 11L)
);
   304:         tableSet.add(new TableInfo("TB_DIM_TIPO_POSSE_TERRA", "CO_SEQ_DIM_TIPO_POSSE_TERRA", 9L)
);
   305:         tableSet.add(new TableInfo("TB_DIM_TIPO_RENDA_FAMILIAR", "CO_SEQ_DIM_TIPO_RENDA_FAMIL
IAR", 8L));
   306:         tableSet.add(new TableInfo("TB_DIM_TIPO_SAIDA_CADASTRO", "CO_SEQ_DIM_TIPO_SAIDA_CADAS
TRO", 3L));
   307:         tableSet.add(new TableInfo("TB_DIM_TIPO_SITUACAO_MORADIA", "CO_SEQ_DIM_TIPO_SITUACAO_
MORAD", 9L));
   308:         tableSet.add(new TableInfo("TB_DIM_TIPO_TRATAMENTO_AGUA", "CO_SEQ_DIM_TIPO_TRATAMEN
T_AGUA", 6L));
   309:         tableSet.add(new TableInfo("TB_DIM_TURNO", "CO_SEQ_DIM_TURNO", 4L));
   310:         tableSet.add(new TableInfo("TB_DIM_UF", "CO_SEQ_DIM_UF", 0L));
   311:         tableSet.add(new TableInfo("TB_DIM_UNIDADE_SAUDE", "CO_SEQ_DIM_UNIDADE_SAUDE", 0L));
   312:         tableSet.add(new TableInfo("TB_DOSE_IMUNOBIOLOGICO", "CO_DOSE_IMUNOBIOLOGICO", 38L));
   313:         tableSet.add(new TableInfo("TB_ENCAMINHAMENTO", "CO_SEQ_ENCAMINHAMENTO", 0L));
   314:         tableSet.add(new TableInfo("TB_ENDERECO", "CO_SEQ_ENDERECO", 0L));
   315:         tableSet.add(new TableInfo("TB_ENDERECO_EXTERIOR", "CO_SEQ_ENDERECO_EXTERIOR", 0L));
   316:         tableSet.add(new TableInfo("TB_ENVIO_LOG", "CO_SEQ_ENVIO_LOG", 0L));
   317:         tableSet.add(new TableInfo("TB_EQUIPE", "CO_SEQ_EQUIPE", 0L));
   318:         tableSet.add(new TableInfo("TB_ESCOLARIDADE", "CO_ESCOLARIDADE", 15L));
   319:         tableSet.add(new TableInfo("TB_ESPECIALIDADE_SISREG", "CO_ESPECIALIDADE_SISREG", 76L));
   320:         tableSet.add(new TableInfo("TB_ESTADO_CIVIL", "CO_ESTADO_CIVIL", 5L));
   321:         tableSet.add(new TableInfo("TB_ESTRATEGIA_VACINACAO", "CO_ESTRATEGIA_VACINACAO", 10L))
;
   322:         tableSet.add(new TableInfo("TB_ETNIA", "CO_ETNIA", 264L));
   323:         tableSet.add(new TableInfo("TB_EVOLUCAO_AVALIACAO", "CO_ATEND_PROF", 0L));
   324:         tableSet.add(new TableInfo("TB_EVOLUCAO_DENTE", "CO_SEQ_EVOLUCAO_DENTE", 0L));
   325:         tableSet.add(new TableInfo("TB_EVOLUCAO_OBJETIVO", "CO_ATEND_PROF", 0L));
   326:         tableSet.add(new TableInfo("TB_EVOLUCAO_ODONTO", "CO_SEQ_EVOLUCAO_ODONTO", 0L));
   327:         tableSet.add(new TableInfo("TB_EVOLUCAO_PLANO", "CO_ATEND_PROF", 0L));
   328:         tableSet.add(new TableInfo("TB_EVOLUCAO_SUBJETIVO", "CO_ATEND_PROF", 0L));
   329:         tableSet.add(new TableInfo("TB_EXAME_DETALHE", "CO_SEQ_EXAME_DETALHE", 1L));
   330:         tableSet.add(new TableInfo("TB_EXAME_DETALHE_RESULTADO", "CO_SEQ_EXAME_DETALHE_RES
ULTADO", 0L));
   331:         tableSet.add(new TableInfo("TB_EXAME_HEMOGLOBINA_GLICADA", "CO_SEQ_EXAME_HEMOGLOB
INA_GLICD", 0L));
   332:         tableSet.add(new TableInfo("TB_EXAME_PRENATAL", "CO_SEQ_EXAME_PRENATAL", 0L));
   333:         tableSet.add(new TableInfo("TB_EXAME_PUERICULTURA", "CO_SEQ_EXAME", 0L));
   334:         tableSet.add(new TableInfo("TB_EXAME_REQUISITADO", "CO_SEQ_EXAME_REQUISITADO", 0L));
   335:         tableSet.add(new TableInfo("TB_FAIXA_ETARIA_VACINACAO", "CO_FAIXA_ETARIA_VACINACAO",
184L));
   336:         tableSet.add(new TableInfo("TB_FAMILIA", "CO_SEQ_FAMILIA", 0L));
   337:         tableSet.add(new TableInfo("TB_FAT_ATD_IND_PROBLEMAS", "CO_SEQ_FAT_ATEND_IND_PROBLE
MAS", 0L));
   338:         tableSet.add(new TableInfo("TB_FAT_ATD_IND_PROCEDIMENTOS", "CO_SEQ_FAT_ATEND_IND_PRO
CED", 0L));
   339:         tableSet.add(new TableInfo("TB_FAT_ATENDIMENTO_DOMICILIAR", "CO_SEQ_FAT_ATEND_DOMICI
LIAR", 0L));
   340:         tableSet.add(new TableInfo("TB_FAT_ATENDIMENTO_INDIVIDUAL", "CO_SEQ_FAT_ATD_IND", 0L));
   341:         tableSet.add(new TableInfo("TB_FAT_ATENDIMENTO_ODONTO", "CO_SEQ_FAT_ATD_ODNT", 0L));
   342:         tableSet.add(new TableInfo("TB_FAT_ATEND_DOM_PROBLEMA", "CO_SEQ_FAT_ATEND_DOM_PRO
BLEMA", 0L));
   343:         tableSet.add(new TableInfo("TB_FAT_ATEND_DOM_PROCED", "CO_SEQ_FAT_ATEND_DOM_PROCED
", 0L));
   344:         tableSet.add(new TableInfo("TB_FAT_ATEND_ODONTO_PROBLEMAS", "CO_SEQ_FAT_ATND_ODONT
O_PROBL", 0L));
   345:         tableSet.add(new TableInfo("TB_FAT_ATEND_ODONTO_PROCED", "CO_SEQ_FAT_ATEND_ODONTO_
PROCED", 0L));
   346:         tableSet.add(new TableInfo("TB_FAT_ATIVIDADE_COLETIVA", "CO_SEQ_FAT_ATIVIDADE_COLETIV
```

```
A", 0L));
   347:         tableSet.add(new TableInfo("TB_FAT_ATVDD_COLETIVA_EXT", "CO_SEQ_FAT_ATVDD_CLTV_EXT",
0L));
   348:         tableSet.add(new TableInfo("TB_FAT_ATVDD_COLETIVA_INT", "CO_SEQ_FAT_ATVDD_CLTV_INT", 0
L));
   349:         tableSet.add(new TableInfo("TB_FAT_AVALIACAO_ELEGIBILIDADE", "CO_SEQ_FAT_AVALIACAO_E
LEGIBLDD", 0L));
   350:         tableSet.add(new TableInfo("TB_FAT_CAD_DOMICILIAR", "CO_SEQ_FAT_CAD_DOMICILIAR", 0L));
   351:         tableSet.add(new TableInfo("TB_FAT_CAD_DOM_FAMILIA", "CO_SEQ_FAT_CAD_DOM_FAMILIA", 0L
));
   352:         tableSet.add(new TableInfo("TB_FAT_CAD_INDIVIDUAL", "CO_SEQ_FAT_CAD_INDIVIDUAL", 0L));
   353:         tableSet.add(new TableInfo("TB_FAT_CIDADAO", "CO_SEQ_FAT_CIDADAO", 0L));
   354:         tableSet.add(new TableInfo("TB_FAT_COMPLEMENTAR", "CO_SEQ_FAT_COMPLEMENTAR", 0L));
   355:         tableSet.add(new TableInfo("TB_FAT_FAMILIA", "CO_SEQ_FAT_FAMILIA", 0L));
   356:         tableSet.add(new TableInfo("TB_FAT_FICHAS", "CO_SEQ_FAT_FICHAS", 0L));
   357:         tableSet.add(new TableInfo("TB_FAT_MARCA_CONSUMO_ALIMNT", "CO_SEQ_FAT_MARCA_CON_A
LMNT", 0L));
   358:         tableSet.add(new TableInfo("TB_FAT_PROCEDIMENTO", "CO_SEQ_FAT_PROCEDIMENTO", 0L));
   359:         tableSet.add(new TableInfo("TB_FAT_PROCED_ATEND", "CO_SEQ_FAT_PROCED_ATEND", 0L));
   360:         tableSet.add(new TableInfo("TB_FAT_PROCED_ATEND_PROCED", "CO_SEQ_FAT_PROCED_ATEND_P
ROCED", 0L));
   361:         tableSet.add(new TableInfo("TB_FAT_VACINACAO", "CO_SEQ_FAT_VACINACAO", 0L));
   362:         tableSet.add(new TableInfo("TB_FAT_VACINACAO_VACINA", "CO_SEQ_FAT_VACINACAO_VACINA",
 0L));
   363:         tableSet.add(new TableInfo("TB_FAT_VISITA_DOMICILIAR", "CO_SEQ_FAT_VISITA_DOMICILIAR", 0
L));
   364:         tableSet.add(new TableInfo("TB_FICHA_ZIKA_TIPO_EXAME", "CO_FICHA_ZIKA_TIPO_EXAME", 18L))
;
   365:         tableSet.add(new TableInfo("TB_FORMA_FARMACEUTICA", "CO_FORMA_FARMACEUTICA", 115L));
   366:         tableSet.add(new TableInfo("TB_GESTOR_ESTADUAL", "CO_ATOR_PAPEL", 0L));
   367:         tableSet.add(new TableInfo("TB_GESTOR_MUNICIPAL", "CO_ATOR_PAPEL", 0L));
   368:         tableSet.add(new TableInfo("TB_GRAVIDADE", "CO_GRAVIDADE", 2L));
   369:         tableSet.add(new TableInfo("TB_GRUPO_ESPECIALIDADE", "CO_GRUPO_ESPECIALIDADE", 3L));
   370:         tableSet.add(new TableInfo("TB_HISTORICO_RELATORIO", "CO_SEQ_HISTORICO_RELATORIO", 0L));
   371:         tableSet.add(new TableInfo("TB_IMUNOBIOLOGICO", "CO_IMUNOBIOLOGICO", 84L));
   372:         tableSet.add(new TableInfo("TB_INTEGRACAO_HORUS", "CO_UNIDADE_SAUDE", 0L));
   373:         tableSet.add(new TableInfo("TB_JUSTIFICATIVA_AGENDA", "CO_SEQ_JUSTIFICATIVA_AGENDAMNT
", 0L));
   374:         tableSet.add(new TableInfo("TB_JUSTIFICATIVA_PRONTUARIO", "CO_SEQ_JUSTIFICATIVA_PRONTU
AR", 0L));
   375:         tableSet.add(new TableInfo("TB_JUSTIFICATIVA_STATUS_CIDDAO", "CO_SEQ_JUSTIFICA_INATIV_C
IDADA", 0L));
   376:         tableSet.add(new TableInfo("TB_LEMBRETE", "CO_SEQ_LEMBRETE", 0L));
   377:         tableSet.add(new TableInfo("TB_LEMBRETE_EVOLUCAO", "CO_SEQ_LEMBRETE_EVOLUCAO", 0L));
   378:         tableSet.add(new TableInfo("TB_LISTA_MEDICAMENTO", "CO_LISTA_MEDICAMENTO", 14L));
   379:         tableSet.add(new TableInfo("TB_LOCALIDADE", "CO_LOCALIDADE", 10828L));
   380:         tableSet.add(new TableInfo("TB_LOCAL_APL_VACINA", "CO_LOCAL_APL_VACINA", 21L));
   381:         tableSet.add(new TableInfo("TB_LOCAL_ATEND", "CO_LOCAL_ATEND", 15L));
   382:         tableSet.add(new TableInfo("TB_LOGRADOURO", "CO_LOGRADOURO", 1013205L));
   383:         tableSet.add(new TableInfo("TB_LOTACAO", "CO_ATOR_PAPEL", 0L));
   384:         tableSet.add(new TableInfo("TB_LOTE_TRANSP", "CO_SEQ_LOTE_TRANSP", 0L));
   385:         tableSet.add(new TableInfo("TB_LOTE_TRANSP_HISTORICO_EXPRT", "CO_SEQ_LOTE_TRANSP_HIST
ORC_EXP", 0L));
   386:         tableSet.add(new TableInfo("TB_LOTE_TRANSP_ITEM", "CO_SEQ_LOTE_TRANSP_ITEM", 0L));
   387:         tableSet.add(new TableInfo("TB_LOTE_TRANSP_ITEM_NODO", "CO_SEQ_LOTE_TRANSP_ITEM_NOD
O", 0L));
   388:         tableSet.add(new TableInfo("TB_LOTE_TRANSP_NODO", "CO_SEQ_LOTE_TRANSP_NODO", 0L));
   389:         tableSet.add(new TableInfo("TB_MEDICAMENTO", "CO_SEQ_MEDICAMENTO", 3338L));
   390:         tableSet.add(new TableInfo("TB_MEDICAMENTO_CATMAT", "CO_MEDICAMENTO_CATMAT", 3338L))
;
   391:         tableSet.add(new TableInfo("TB_MEDICAMENTO_USO_CONTINUO", "CO_SEQ_MEDICAMENT_USO_C
ONTINUO", 0L));
   392:         tableSet.add(new TableInfo("TB_MEDICAO", "CO_SEQ_MEDICAO", 0L));
   393:         tableSet.add(new TableInfo("TB_MEMORIA", "CO_MEMORIA", 0L));
   394:         tableSet.add(new TableInfo("TB_MES", "CO_MES", 11L));
   395:         tableSet.add(new TableInfo("TB_MIGRACAO_ESTRUTURA_TRAVA", "CO_MIGRACAO_ESTRUTURA_
TRAVA", 1L));
   396:         tableSet.add(new TableInfo("TB_MOTIVO_RESERVA", "CO_MOTIVO_RESERVA", 15L));
```

```
   397:         tableSet.add(new TableInfo("TB_NACIONALIDADE", "CO_NACIONALIDADE", 3L));
   398:         tableSet.add(new TableInfo("TB_NODO", "CO_NODO", 0L));
   399:         tableSet.add(new TableInfo("TB_ODONTOGRAMA", "CO_SEQ_ODONTOGRAMA", 0L));
   400:         tableSet.add(new TableInfo("TB_ORIENTACAO", "CO_SEQ_ORIENTACAO", 0L));
   401:         tableSet.add(new TableInfo("TB_ORIGEM", "CO_ORIGEM", 1L));
   402:         tableSet.add(new TableInfo("TB_PAIS", "CO_PAIS", 243L));
   403:         tableSet.add(new TableInfo("TB_PAPEL", "CO_SEQ_PAPEL", 0L));
   404:         tableSet.add(new TableInfo("TB_PARTE_BUCAL", "CO_PARTE_BUCAL", 59L));
   405:         tableSet.add(new TableInfo("TB_PARTE_BUCAL_PROCED", "CO_PARTE_BUCAL_PROCED", 283L));
   406:         tableSet.add(new TableInfo("TB_PERFIL", "CO_SEQ_PERFIL", 23L));
   407:         tableSet.add(new TableInfo("TB_PERGUNTA", "CO_SEQ_PERGUNTA", 1001L));
   408:         tableSet.add(new TableInfo("TB_PERGUNTA_DETALHE", "CO_PERGUNTA_DETALHE", 156L));
   409:         tableSet.add(new TableInfo("TB_PERIODO", "CO_PERIODO", 3L));
   410:         tableSet.add(new TableInfo("TB_PESSOA_FISICA", "CO_ATOR", 0L));
   411:         tableSet.add(new TableInfo("TB_PESSOA_FISICA_IMAGEM", "CO_SEQ_PESSOA_FISICA_IMAGEM", 0L
));
   412:         tableSet.add(new TableInfo("TB_PESSOA_JURIDICA", "CO_ATOR", 0L));
   413:         tableSet.add(new TableInfo("TB_POVO_COMUNIDADE_TRADICIONAL", "CO_POVO_COMUNIDADE_T
RADICIONAL", 22L));
   414:         tableSet.add(new TableInfo("TB_PRE_NATAL", "CO_SEQ_PRE_NATAL", 0L));
   415:         tableSet.add(new TableInfo("TB_PRINCIPIO_ATIVO", "CO_PRINCIPIO_ATIVO", 1482L));
   416:         tableSet.add(new TableInfo("TB_PROBLEMA", "CO_SEQ_PROBLEMA", 0L));
   417:         tableSet.add(new TableInfo("TB_PROBLEMA_EVOLUCAO", "CO_SEQ_PROBLEMA_EVOLUCAO", 0L));
   418:         tableSet.add(new TableInfo("TB_PROCED", "CO_SEQ_PROCED", 4746L));
   419:         tableSet.add(new TableInfo("TB_PROCED_AUTOMATICO", "CO_SEQ_PROCED_AUTOMATICO", 244L))
;
   420:         tableSet.add(new TableInfo("TB_PROCED_EXAME_ESPECIFICO", "CO_PROCED_EXAME_ESPECIFICO"
, 9L));
   421:         tableSet.add(new TableInfo("TB_PROCED_FILTRO", "CO_PROCED", 3250L));
   422:         tableSet.add(new TableInfo("TB_PROCED_FORMA_ORGANIZACIONAL", "CO_PROCED_FORMA_ORG
ANIZACIONAL", 384L));
   423:         tableSet.add(new TableInfo("TB_PROCED_GRUPO", "CO_PROCED_GRUPO", 8L));
   424:         tableSet.add(new TableInfo("TB_PROCED_NOME_POPULAR", "CO_SEQ_PROCED_NOME_POPULAR",
0L));
   425:         tableSet.add(new TableInfo("TB_PROCED_SOLICITADO", "CO_SEQ_PROCED_SOLICITADO", 0L));
   426:         tableSet.add(new TableInfo("TB_PROCED_SUBGRUPO", "CO_PROCED_SUBGRUPO", 59L));
   427:         tableSet.add(new TableInfo("TB_PROCESSAMENTO_DOM_CID", "CO_SEQ_PROCESSAMENTO_DOM_
CID", 0L));
   428:         tableSet.add(new TableInfo("TB_PROCESSO", "CO_SEQ_PROCESSO", 0L));
   429:         tableSet.add(new TableInfo("TB_PROF", "CO_ATOR_PAPEL", 0L));
   430:         tableSet.add(new TableInfo("TB_PRONTUARIO", "CO_SEQ_PRONTUARIO", 0L));
   431:         tableSet.add(new TableInfo("TB_PRONTUARIO_UNIDADE_SAUDE", "CO_SEQ_PRONTUARIO_UNIDAD
E_SAUD", 0L));
   432:         tableSet.add(new TableInfo("TB_QST_ASSOCIACAO_PERGUNTA", "CO_QST_ASSOCIACAO_PERGUNT
A", 3L));
   433:         tableSet.add(new TableInfo("TB_QST_OPCAO_PERGUNTA", "CO_QST_OPCAO_PERGUNTA", 122L));
   434:         tableSet.add(new TableInfo("TB_QST_OPCAO_TIPO_PERGUNTA", "CO_QST_OPCAO_TIPO_PERGUNT
A", 17L));
   435:         tableSet.add(new TableInfo("TB_QST_ORIENTACAO_PROF", "CO_QST_ORIENTACAO_PROF", 27L));
   436:         tableSet.add(new TableInfo("TB_QST_PERGUNTA", "CO_QST_PERGUNTA", 41L));
   437:         tableSet.add(new TableInfo("TB_QST_QUESTIONARIO", "CO_QST_QUESTIONARIO", 3L));
   438:         tableSet.add(new TableInfo("TB_QST_QUESTIONARIO_PERGUNTA", "CO_QST_QUESTIONARIO_PERG
UNTA", 41L));
   439:         tableSet.add(new TableInfo("TB_QST_QUESTIONARIO_RESPONDIDO", "CO_SEQ_QST_QST_RESPOND
IDO", 0L));
   440:         tableSet.add(new TableInfo("TB_QST_RESPOSTA", "CO_SEQ_QST_RESPOSTA", 0L));
   441:         tableSet.add(new TableInfo("TB_QST_TIPO_QUESTIONARIO", "CO_QST_TIPO_QUESTIONARIO", 0L));
   442:         tableSet.add(new TableInfo("TB_QST_TIPO_RESPOSTA", "CO_QST_TIPO_RESPOSTA", 3L));
   443:         tableSet.add(new TableInfo("TB_RACA_COR", "CO_RACA_COR", 6L));
   444:         tableSet.add(new TableInfo("TB_RACIONALIDADE_SAUDE", "CO_RACIONALIDADE_SAUDE", 6L));
   445:         tableSet.add(new TableInfo("TB_RECEBIMENTO_ITEM", "CO_SEQ_RECEB_ITEM", 0L));
   446:         tableSet.add(new TableInfo("TB_RECEBIMENTO_LOTE", "CO_SEQ_RECEB_LOTE", 0L));
   447:         tableSet.add(new TableInfo("TB_RECEBIMENTO_VALIDACAO_ERROS", "CO_RECEB_ITEM", 0L));
   448:         tableSet.add(new TableInfo("TB_RECEITA", "CO_SEQ_RECEITA", 0L));
   449:         tableSet.add(new TableInfo("TB_RECEITA_MEDICAMENTO", "CO_SEQ_RECEITA_MEDICAMENTO", 0
L));
   450:         tableSet.add(new TableInfo("TB_RECEITA_TIPO_FREQUENCIA", "CO_RECEITA_TIPO_FREQUENCIA",
3L));
```

```
   451:         tableSet.add(new TableInfo("TB_RECURSO", "CO_SEQ_RECURSO", 0L));
   452:         tableSet.add(new TableInfo("TB_REGRA_CONDICIONADA", "CO_REGRA_CONDICIONADA", 10L));
   453:         tableSet.add(new TableInfo("TB_REGRA_VACINAL_DOSE", "CO_REGRA_VACINAL_DOSE", 437L));
   454:         tableSet.add(new TableInfo("TB_REGRA_VACINAL_ESTRATEGIA", "CO_REGRA_VACINAL_ESTRATE
GIA", 876L));
   455:         tableSet.add(new TableInfo("TB_RELATORIO_PROCESSAMENTO", "CO_SEQ_RELATORIO_PROCESSA
MENTO", 0L));
   456:         tableSet.add(new TableInfo("TB_REL_FICHAS_CONFIG", "CO_RELATORIO_FICHA", 0L));
   457:         tableSet.add(new TableInfo("TB_REL_OP_CIDADAO", "CO_SEQ_REL_OP_CIDADAO", 0L));
   458:         tableSet.add(new TableInfo("TB_REL_OP_CODIGO_UNICO_CIDADAO", "CO_SEQ_REL_OP_CODIGO_U
NICO", 0L));
   459:         tableSet.add(new TableInfo("TB_REL_OP_CRIANCA", "CO_SEQ_REL_OP_CRIANCA", 0L));
   460:         tableSet.add(new TableInfo("TB_REL_OP_FAMILIA", "CO_SEQ_REL_OP_FAMILIA", 0L));
   461:         tableSet.add(new TableInfo("TB_REL_OP_GESTANTE", "CO_SEQ_REL_OP_GESTANTE", 0L));
   462:         tableSet.add(new TableInfo("TB_REL_OP_IMOVEL", "CO_SEQ_REL_OP_IMOVEL", 0L));
   463:         tableSet.add(new TableInfo("TB_REL_OP_NUCLEO_FAMILIAR", "CO_SEQ_REL_OP_NUCLEO_FAMILI
AR", 0L));
   464:         tableSet.add(new TableInfo("TB_REL_OP_RISCO_CARDIO", "CO_SEQ_REL_OP_RISCO_CARDIO", 0L));
   465:         tableSet.add(new TableInfo("TB_REL_OP_SITUACAO_CLINICA", "CO_SITUACAO_CLINICA", 4L));
   466:         tableSet.add(new TableInfo("TB_RENDA_FAMILIAR", "CO_RENDA_FAMILIAR", 7L));
   467:         tableSet.add(new TableInfo("TB_REQUISICAO_EXAME", "CO_SEQ_REQUISICAO_EXAME", 0L));
   468:         tableSet.add(new TableInfo("TB_RES_ENVIO", "CO_SEQ_RES_ENVIO", 0L));
   469:         tableSet.add(new TableInfo("TB_RES_ENVIO_ERROS", "CO_RES_ENVIO", 0L));
   470:         tableSet.add(new TableInfo("TB_RES_ENVIO_ESTADO", "CO_RES_ENVIO_ESTADO", 4L));
   471:         tableSet.add(new TableInfo("TB_REVISAO", "CO_SEQ_REVISAO", 0L));
   472:         tableSet.add(new TableInfo("TB_SESSAO_SINCRONIZACAO", "CO_UNICO_SESSAO", 0L));
   473:         tableSet.add(new TableInfo("TB_SEXO", "CO_SEXO", 4L));
   474:         tableSet.add(new TableInfo("TB_SEXTANTE", "CO_PARTE_BUCAL", 7L));
   475:         tableSet.add(new TableInfo("TB_SITUACAO_AGENDADO", "CO_SITUACAO_AGENDADO", 7L));
   476:         tableSet.add(new TableInfo("TB_SITUACAO_COROA", "CO_SITUACAO_COROA", 2L));
   477:         tableSet.add(new TableInfo("TB_SITUACAO_DADO_RECEBIDO", "CO_SITUACAO_DADO_RECEBIDO",
 7L));
   478:         tableSet.add(new TableInfo("TB_SITUACAO_FACE", "CO_SITUACAO_FACE", 22L));
   479:         tableSet.add(new TableInfo("TB_SITUACAO_LOCALIDADE", "CO_SITUACAO_LOCALIDADE", 3L));
   480:         tableSet.add(new TableInfo("TB_SITUACAO_LOTE_TRANSP_NODO", "CO_SITUACAO_LOTE_TRANSP
_NODO", 4L));
   481:         tableSet.add(new TableInfo("TB_SITUACAO_PROBLEMA", "CO_SITUACAO_PROBLEMA", 2L));
   482:         tableSet.add(new TableInfo("TB_SITUACAO_RAIZ", "CO_SITUACAO_RAIZ", 15L));
   483:         tableSet.add(new TableInfo("TB_STATUS_ATEND", "CO_STATUS_ATEND", 5L));
   484:         tableSet.add(new TableInfo("TB_STATUS_ATEND_PROF", "CO_STATUS_ATEND_PROF", 2L));
   485:         tableSet.add(new TableInfo("TB_TIPO_AGENDAMENTO", "CO_TIPO_AGENDAMENTO", 1L));
   486:         tableSet.add(new TableInfo("TB_TIPO_AGRAVO", "CO_TIPO_AGRAVO", 2L));
   487:         tableSet.add(new TableInfo("TB_TIPO_ATEND", "CO_TIPO_ATEND", 9L));
   488:         tableSet.add(new TableInfo("TB_TIPO_ATEND_PROF", "CO_TIPO_ATEND_PROF", 12L));
   489:         tableSet.add(new TableInfo("TB_TIPO_CIAP", "CO_TIPO_CIAP", 7L));
   490:         tableSet.add(new TableInfo("TB_TIPO_CONFIG_ATEND_DOMICILIR", "CO_TIPO_CONFIG_ATEND_DO
M", 1L));
   491:         tableSet.add(new TableInfo("TB_TIPO_CONSULTA_ODONTO", "CO_TIPO_CONSULTA_ODONTO", 99L)
);
   492:         tableSet.add(new TableInfo("TB_TIPO_CONTATO", "CO_TIPO_CONTATO", 6L));
   493:         tableSet.add(new TableInfo("TB_TIPO_DADO_TRANSP", "CO_TIPO_DADO_TRANSP", 14L));
   494:         tableSet.add(new TableInfo("TB_TIPO_EDEMA", "CO_TIPO_EDEMA", 4L));
   495:         tableSet.add(new TableInfo("TB_TIPO_ENCAM_INTERNO", "CO_TIPO_ENCAM_INTERNO", 3L));
   496:         tableSet.add(new TableInfo("TB_TIPO_ENCAM_ODONTO", "CO_TIPO_ENCAM_ODONTO", 99L));
   497:         tableSet.add(new TableInfo("TB_TIPO_EQUIPE", "CO_SEQ_TIPO_EQUIPE", 54L));
   498:         tableSet.add(new TableInfo("TB_TIPO_EXAME", "CO_TIPO_EXAME", 1L));
   499:         tableSet.add(new TableInfo("TB_TIPO_FILTRO_PROCED", "CO_TIPO_FILTRO_PROCED", 4L));
   500:         tableSet.add(new TableInfo("TB_TIPO_FORNEC_ODONTO", "CO_TIPO_FORNEC_ODONTO", 3L));
   501:         tableSet.add(new TableInfo("TB_TIPO_GLICEMIA", "CO_TIPO_GLICEMIA", 3L));
   502:         tableSet.add(new TableInfo("TB_TIPO_GRAVIDEZ", "CO_TIPO_GRAVIDEZ", 4L));
   503:         tableSet.add(new TableInfo("TB_TIPO_LOCALIDADE", "CO_TIPO_LOCALIDADE", 4L));
   504:         tableSet.add(new TableInfo("TB_TIPO_LOGRADOURO", "CO_TIPO_LOGRADOURO", 281L));
   505:         tableSet.add(new TableInfo("TB_TIPO_MEDICAO", "CO_SEQ_TIPO_MEDICAO", 13L));
   506:         tableSet.add(new TableInfo("TB_TIPO_OPCAO", "CO_TIPO_OPCAO", 4L));
   507:         tableSet.add(new TableInfo("TB_TIPO_ORIGEM_DADO_TRANSP", "CO_SEQ_TIPO_ORIGEM_DADO_TR
ANSP", 5L));
   508:         tableSet.add(new TableInfo("TB_TIPO_PARIDADE", "CO_TIPO_PARIDADE", 5L));
   509:         tableSet.add(new TableInfo("TB_TIPO_PARTE_BUCAL", "CO_TIPO_PARTE_BUCAL", 3L));
```

```
510:            tableSet.add(new TableInfo("TB_TIPO_PERFIL", "CO_TIPO_PERFIL", 8L));
511:            tableSet.add(new TableInfo("TB_TIPO_PERGUNTA", "CO_TIPO_PERGUNTA", 4L));
512:            tableSet.add(new TableInfo("TB_TIPO_PESSOA_FISICA_IMAGEM", "CO_TIPO_PESSOA_FISICA_IMAG
EM", 0L));
513:            tableSet.add(new TableInfo("TB_TIPO_RECEITA", "CO_TIPO_RECEITA", 4L));
514:            tableSet.add(new TableInfo("TB_TIPO_REGISTRO", "CO_TIPO_REGISTRO", 9L));
515:            tableSet.add(new TableInfo("TB_TIPO_SANGUINEO", "CO_TIPO_SANGUINEO", 7L));
516:            tableSet.add(new TableInfo("TB_TIPO_SERVICO", "CO_TIPO_SERVICO", 10L));
517:            tableSet.add(new TableInfo("TB_TIPO_UNIDADE_SAUDE", "CO_SEQ_TIPO_UNIDADE_SAUDE", 37L));
518:            tableSet.add(new TableInfo("TB_TITULO_PATENTE", "CO_TITULO_PATENTE", 741L));
519:            tableSet.add(new TableInfo("TB_UF", "CO_UF", 27L));
520:            tableSet.add(new TableInfo("TB_UNIDADE_MEDIDA", "CO_UNIDADE_MEDIDA", 40L));
521:            tableSet.add(new TableInfo("TB_UNIDADE_MEDIDA_TEMPO", "CO_UNIDADE_MEDIDA_TEMPO", 5L))
;
522:            tableSet.add(new TableInfo("TB_UNIDADE_SAUDE", "CO_ATOR_PAPEL", 0L));
523:            tableSet.add(new TableInfo("TB_USUARIO", "CO_SEQ_USUARIO", 0L));
524:            tableSet.add(new TableInfo("TB_VACINA", "CO_SEQ_VACINA", 0L));
525:            tableSet.add(new TableInfo("TB_VIA_ADM_VACINA", "CO_VIA_ADM_VACINA", 6L));
526:            tableSet.add(new TableInfo("TB_VISIBILIDADE_LEMBRETE", "CO_VISIBILIDADE_LEMBRETE", 1L));
527:            return tableSet;
528:    }
529:
530:  }
```

```
 1: package br.ufsc.tcc.testproject.config.database;
 2:
 3: import br.ufsc.tcc.testproject.config.database.generator.SequencesInfo;
 4:
 5: import com.google.common.collect.ImmutableSet;
 6:
 7: //@formatter:off
 8: public class CdsSequencesInfo {
 9:
10:
11:  public static final ImmutableSet<SequencesInfo> getSequencesInfo() {
12:         return ImmutableSet.<SequencesInfo> builder()
13:         .build();
14:  }
15:
16: }
```

```
 1:  package br.ufsc.tcc.testproject.config.database;
 2:
 3:  import lombok.Getter;
 4:  import lombok.Setter;
 5:  import lombok.ToString;
 6:
 7:  @Setter
 8:  @Getter
 9:  @ToString
10:  public class DatabaseConfig {
11:   private String connectionUrl;
12:   private String login;
13:   private String password;
14:
15:  }
```

```
 1: package br.ufsc.tcc.testproject.config.database;
 2:
 3: import java.util.HashSet;
 4: import java.util.Set;
 5:
 6: import br.ufsc.tcc.testproject.config.database.generator.TableInfo;
 7:
 8: //@formatter:off
 9: public class CdsTables {
10:
11:  public static final Set<TableInfo> getTablesInfo() {
12:         Set<TableInfo> tableSet = new HashSet<>();
13:         tableSet.add(new TableInfo("RL_ANTECEDENTE_CIAP", "CO_CIAP", 0L));
14:         tableSet.add(new TableInfo("RL_ANTECEDENTE_CIAP", "CO_PRONTUARIO", 0L));
15:         tableSet.add(new TableInfo("RL_ATEND_PROCED", "CO_SEQ_ATEND_PROCED", 0L));
16:         tableSet.add(new TableInfo("RL_ATEND_PROF_AD_TIPO_INELEGVL", "CO_ATEND_PROF_AD", 0L));
17:         tableSet.add(new TableInfo("RL_ATEND_PROF_AD_TIPO_INELEGVL", "TP_AD_INELEGIVEL", 0L));
18:         tableSet.add(new TableInfo("RL_ATEND_PROF_CONDUTA", "CO_ATEND_PROF", 0L));
19:         tableSet.add(new TableInfo("RL_ATEND_PROF_CONDUTA", "TP_CDS_CONDUTA", 0L));
20:         tableSet.add(new TableInfo("RL_ATEND_PROF_ODONTO_TIPO_ENCM", "CO_ATEND_PROF_ODONTO
", 0L));
21:         tableSet.add(new TableInfo("RL_ATEND_PROF_ODONTO_TIPO_ENCM", "TP_ENCAM_ODONTO", 0L));
22:         tableSet.add(new TableInfo("RL_ATEND_PROF_ODONTO_TIPO_FRNC", "CO_ATEND_PROF_ODONTO"
, 0L));
23:         tableSet.add(new TableInfo("RL_ATEND_PROF_ODONTO_TIPO_FRNC", "TP_FORNECIMENTO_ODONT
O", 0L));
24:         tableSet.add(new TableInfo("RL_ATEND_PROF_PIC", "CO_ATEND_PROF", 0L));
25:         tableSet.add(new TableInfo("RL_ATEND_PROF_PIC", "CO_CDS_PIC", 0L));
26:         tableSet.add(new TableInfo("RL_ATEND_TIPO_SERVICO", "CO_ATEND", 0L));
27:         tableSet.add(new TableInfo("RL_ATEND_TIPO_SERVICO", "TP_SERVICO", 0L));
28:         tableSet.add(new TableInfo("RL_ATOR_PAPEL_PERFIL", "CO_ATOR_PAPEL", 0L));
29:         tableSet.add(new TableInfo("RL_ATOR_PAPEL_PERFIL", "CO_PERFIL", 0L));
30:         tableSet.add(new TableInfo("RL_CDS_ATEND_DOM_PROCED", "CO_CDS_ATEND_DOMICILIAR", 0L));
31:         tableSet.add(new TableInfo("RL_CDS_ATEND_DOM_PROCED", "CO_PROCED", 0L));
32:         tableSet.add(new TableInfo("RL_CDS_ATEND_DOM_SITUACAO_PRES", "CO_CDS_ATEND_DOMICILI
AR", 0L));
33:         tableSet.add(new TableInfo("RL_CDS_ATEND_DOM_SITUACAO_PRES", "CO_CDS_SITUACAO_PRESE
NTE", 0L));
34:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_CIAP", "CO_CDS_ATEND_INDIVIDUAL",
0L));
35:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_CIAP", "CO_CIAP", 0L));
36:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_CONDUT", "CO_CDS_ATEND_INDIVIDUA
L", 0L));
37:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_CONDUT", "TP_CDS_CONDUTA", 0L));
38:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_EXAME", "CO_CDS_ATEND_INDIVIDUAL
", 0L));
39:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_EXAME", "CO_EXAME", 0L));
40:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_EXAME", "ST_SOLICITADO_AVALIADO",
 0L));
41:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_NASF", "CO_CDS_ATEND_INDIVIDUAL",
0L));
42:         tableSet.add(new TableInfo("RL_CDS_ATEND_INDIVIDUAL_NASF", "TP_CDS_ATEND_NASF", 0L));
43:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_PROCED", "CO_CDS_ATEND_ODONTO", 0L));
44:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_PROCED", "CO_PROCED", 0L));
45:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_CNSLT", "CO_CDS_ATEND_ODONTO",
0L));
46:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_CNSLT", "TP_CONSULTA_ODONTO", 0
L));
47:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_ENCAM", "CO_CDS_ATEND_ODONTO",
 0L));
48:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_ENCAM", "TP_ENCAM_ODONTO", 0L));
49:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_FORNC", "CO_CDS_ATEND_ODONTO",
0L));
50:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONTO_TIPO_FORNC", "TP_FORNECIMENTO_ODON
TO", 0L));
51:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONT_TIP_VIG_BUC", "CO_CDS_ATEND_ODONTO", 0
L));
52:         tableSet.add(new TableInfo("RL_CDS_ATEND_ODONT_TIP_VIG_BUC", "TP_CDS_VIG_SAUDE_BUCAL
```

```
", 0L));
     53:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATEND_INDIVDL_PRF", "CO_CDS_FICHA_ATEND_INDI
VIDUAL", 0L));
     54:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATEND_INDIVDL_PRF", "CO_CDS_PROF", 0L));
     55:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATEND_ODONTO_PROF", "CO_CDS_FICHA_ATEND_OD
ONTO", 0L));
     56:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATEND_ODONTO_PROF", "CO_CDS_PROF", 0L));
     57:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PRATICA", "CO_CDS_ATIV_COL_PRATICA",
0L));
     58:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PRATICA", "CO_CDS_FICHA_ATIV_COL", 0L
));
     59:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PROF", "CO_CDS_FICHA_ATIV_COL", 0L));
     60:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PROF", "CO_CDS_PROF", 0L));
     61:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PUB_ALVO", "CO_CDS_ATIV_COL_PUBLIC
O_ALVO", 0L));
     62:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_PUB_ALVO", "CO_CDS_FICHA_ATIV_COL",
0L));
     63:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_TEMA", "CO_CDS_ATIV_COL_TEMA", 0L));
     64:          tableSet.add(new TableInfo("RL_CDS_FICHA_ATIV_COL_TEMA", "CO_CDS_FICHA_ATIV_COL", 0L));
     65:          tableSet.add(new TableInfo("RL_CDS_PROCED_ADM_MEDICAMENTO", "CO_CDS_ADM_MEDICAME
NTO", 0L));
     66:          tableSet.add(new TableInfo("RL_CDS_PROCED_ADM_MEDICAMENTO", "CO_CDS_PROCED", 0L));
     67:          tableSet.add(new TableInfo("RL_CDS_PROCED_ADM_MEDICAMENTO", "CO_PROCED", 0L));
     68:          tableSet.add(new TableInfo("RL_CDS_PRONTUARIO_UNIDADE_SAUD", "CO_ATOR_PAPEL", 0L));
     69:          tableSet.add(new TableInfo("RL_CDS_PRONTUARIO_UNIDADE_SAUD", "CO_PRONTUARIO", 0L));
     70:          tableSet.add(new TableInfo("RL_CDS_VISITA_DOM_MOTIVO", "CO_CDS_VISITA_DOMICILIAR", 0L));
     71:          tableSet.add(new TableInfo("RL_CDS_VISITA_DOM_MOTIVO", "CO_CDS_VISITA_DOM_MOTIVO", 0L)
);
     72:          tableSet.add(new TableInfo("RL_CIAP_CID10", "CO_CIAP", 751L));
     73:          tableSet.add(new TableInfo("RL_CIAP_CID10", "CO_CID10", 14230L));
     74:          tableSet.add(new TableInfo("RL_EVOLUCAO_AVALIACAO_CIAP_CID", "CO_SEQ_EVOLUCAO_AVAL_
CIAP_CID", 0L));
     75:          tableSet.add(new TableInfo("RL_EVOLUCAO_OBJETIVO_MEDICAO", "CO_ATEND_PROF", 0L));
     76:          tableSet.add(new TableInfo("RL_EVOLUCAO_OBJETIVO_MEDICAO", "CO_MEDICAO", 0L));
     77:          tableSet.add(new TableInfo("RL_EVOLUCAO_ODONTO_PARTE_BUCAL", "CO_EVOLUCAO_ODONTO"
, 0L));
     78:          tableSet.add(new TableInfo("RL_EVOLUCAO_ODONTO_PARTE_BUCAL", "CO_PARTE_BUCAL", 0L));
     79:          tableSet.add(new TableInfo("RL_EVOLUCAO_ODONTO_PROCED", "CO_EVOLUCAO_ODONTO", 0L));
     80:          tableSet.add(new TableInfo("RL_EVOLUCAO_ODONTO_PROCED", "CO_PROCED", 0L));
     81:          tableSet.add(new TableInfo("RL_EVOLUCAO_PLANO_CIAP", "CO_SEQ_EVOLUCAO_PLANO_CIAP", 0L
));
     82:          tableSet.add(new TableInfo("RL_EVOLUCAO_SUBJETIVO_CIAP", "CO_SEQ_EVOLUCAO_SUBJETIVO_
CIAP", 0L));
     83:          tableSet.add(new TableInfo("RL_EVOL_AVAL_TP_VIG_SAUDE_BUCL", "CO_EVOLUCAO_AVALIACA
O", 0L));
     84:          tableSet.add(new TableInfo("RL_EVOL_AVAL_TP_VIG_SAUDE_BUCL", "CO_TIPO_VIG_SAUDE_BUCA
L", 0L));
     85:          tableSet.add(new TableInfo("RL_MEDICAO_TIPO_MEDICAO", "CO_SEQ_MEDICAO_TIPO_MEDICAO",
0L));
     86:          tableSet.add(new TableInfo("RL_PERFIL_CBO_PADRAO", "CO_CBO", 2609L));
     87:          tableSet.add(new TableInfo("RL_PESSOA_ENDERECO", "CO_ATOR", 937L));
     88:          tableSet.add(new TableInfo("RL_PESSOA_ENDERECO", "CO_ENDERECO", 927L));
     89:          tableSet.add(new TableInfo("RL_PROCED_ATRIBUTO_COMPLEM", "CO_ATRIBUTO_COMPLEM", 41L))
;
     90:          tableSet.add(new TableInfo("RL_PROCED_ATRIBUTO_COMPLEM", "CO_PROCED", 4746L));
     91:          tableSet.add(new TableInfo("RL_PROCED_CBO", "CO_CBO", 2608L));
     92:          tableSet.add(new TableInfo("RL_PROCED_CBO", "CO_PROCED", 4744L));
     93:          tableSet.add(new TableInfo("RL_PROCED_CDS_PROCED", "CO_CDS_PROCED", 0L));
     94:          tableSet.add(new TableInfo("RL_PROCED_CDS_PROCED", "CO_PROCED", 0L));
     95:          tableSet.add(new TableInfo("RL_PROCED_CID10", "CO_CID10", 12444L));
     96:          tableSet.add(new TableInfo("RL_PROCED_CID10", "CO_PROCED", 4746L));
     97:          tableSet.add(new TableInfo("RL_PROCED_EXAME_DETALHE", "CO_SEQ_PROCED_EXAME_DETALHE
", 1L));
     98:          tableSet.add(new TableInfo("RL_PROCED_REGRA_CONDICIONADA", "CO_PROCED", 4633L));
     99:          tableSet.add(new TableInfo("RL_PROCED_REGRA_CONDICIONADA", "CO_REGRA_CONDICIONADA",
 9L));
    100:          tableSet.add(new TableInfo("RL_PROCED_TIPO_FILTRO_PROCED", "CO_PROCED", 4529L));
    101:          tableSet.add(new TableInfo("RL_PROCED_TIPO_FILTRO_PROCED", "TP_FILTRO_PROCED", 4L));
```

```
   102:         tableSet.add(new TableInfo("RL_PROCED_TIPO_REGISTRO", "CO_PROCED", 4746L));
   103:         tableSet.add(new TableInfo("RL_PROCED_TIPO_REGISTRO", "TP_REGISTRO", 9L));
   104:         tableSet.add(new TableInfo("RL_PROF_MUNICIPIO", "CO_ATOR_PAPEL", 33351L));
   105:         tableSet.add(new TableInfo("RL_PROF_MUNICIPIO", "CO_LOCALIDADE", 8866L));
   106:         tableSet.add(new TableInfo("RL_QST_ASSOCIACAO_OPCAO_PERGNT", "CO_QST_ASSOCIACAO_PER
GUNTA", 3L));
   107:         tableSet.add(new TableInfo("RL_QST_ASSOCIACAO_OPCAO_PERGNT", "CO_QST_OPCAO_PERGUNT
A", 70L));
   108:         tableSet.add(new TableInfo("RL_QST_RESPOSTA_OPCAO_PERGUNTA", "CO_QST_OPCAO_PERGUNT
A", 0L));
   109:         tableSet.add(new TableInfo("RL_QST_RESPOSTA_OPCAO_PERGUNTA", "CO_QST_RESPOSTA", 0L));
   110:         tableSet.add(new TableInfo("RL_RECURSO_ACAO", "CO_RECURSO", 0L));
   111:         tableSet.add(new TableInfo("RL_RECURSO_ACAO", "NU_ACAO", 0L));
   112:         tableSet.add(new TableInfo("RL_RECURSO_ACAO", "NU_OBRIGACAO", 0L));
   113:         tableSet.add(new TableInfo("RL_RECURSO_TIPO_PERFIL", "CO_RECURSO", 0L));
   114:         tableSet.add(new TableInfo("RL_RECURSO_TIPO_PERFIL", "TP_PERFIL", 0L));
   115:         tableSet.add(new TableInfo("RL_REL_OP_CRIANCA_CID10", "CO_SEQ_REL_OP_CRIANCA_CID10", 0L)
);
   116:         tableSet.add(new TableInfo("RL_TIPO_ATEND_PROCED_AUTOMATIC", "CO_PROCED_AUTOMATICO"
, 132L));
   117:         tableSet.add(new TableInfo("RL_TIPO_ATEND_PROCED_AUTOMATIC", "TP_ATEND", 6L));
   118:         tableSet.add(new TableInfo("RL_TIPO_CONSULTA_ODNT_PRC_AUTO", "CO_PROCED_AUTOMATICO"
, 151L));
   119:         tableSet.add(new TableInfo("RL_TIPO_CONSULTA_ODNT_PRC_AUTO", "TP_CONSULTA_ODONTO", 4
L));
   120:         tableSet.add(new TableInfo("RL_TIPO_ENCAM_ODONTO_PROCD_AUT", "CO_PROCED_AUTOMATICO
", 52L));
   121:         tableSet.add(new TableInfo("RL_TIPO_ENCAM_ODONTO_PROCD_AUT", "TP_ENCAM_ODONTO", 15L)
);
   122:         tableSet.add(new TableInfo("RL_TIPO_PERFIL_TIPO_INSTALACAO", "NO_TIPO_INSTALACAO", 0L));
   123:         tableSet.add(new TableInfo("RL_TIPO_PERFIL_TIPO_INSTALACAO", "TP_PERFIL", 0L));
   124:         tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_COMPLEXIDADE", "CO_ATOR_PAPEL", 33346L));
   125:         tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_COMPLEXIDADE", "CO_COMPLEXIDADE", 1L));
   126:         tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_HORUS", "CO_UNIDADE_SAUDE", 0L));
   127:         tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_HORUS", "CO_UNIDADE_SAUDE_HORUS", 0L));
   128:         tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_LOCALIDADE", "CO_ATOR_PAPEL", 33346L));
   129:         tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_LOCALIDADE", "CO_LOCALIDADE", 8866L));
   130:         tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_TIPO_SERVICO", "CO_ATOR_PAPEL", 33346L));
   131:         tableSet.add(new TableInfo("RL_UNIDADE_SAUDE_TIPO_SERVICO", "TP_SERVICO", 10L));
   132:         tableSet.add(new TableInfo("RL_VIA_ADM_LOCAL_APL_VACINA", "CO_LOCAL_APL_VACINA", 21L))
;
   133:         tableSet.add(new TableInfo("RL_VIA_ADM_LOCAL_APL_VACINA", "CO_VIA_ADM_VACINA", 6L));
   134:         tableSet.add(new TableInfo("TB_ACESSO", "CO_PERFIL", 0L));
   135:         tableSet.add(new TableInfo("TB_ACESSO", "CO_RECURSO", 0L));
   136:         tableSet.add(new TableInfo("TB_ACESSO", "NU_ACAO", 0L));
   137:         tableSet.add(new TableInfo("TB_ACESSO", "NU_OBRIGACAO", 0L));
   138:         tableSet.add(new TableInfo("TB_ADM_GERAL", "CO_ATOR_PAPEL", 0L));
   139:         tableSet.add(new TableInfo("TB_ADM_MUNICIPAL", "CO_ATOR_PAPEL", 0L));
   140:         tableSet.add(new TableInfo("TB_ADM_MUNICIPAL_CENTRALIZADOR", "CO_ATOR_PAPEL", 0L));
   141:         tableSet.add(new TableInfo("TB_AD_CIDADAO", "CO_SEQ_AD_CIDADAO", 0L));
   142:         tableSet.add(new TableInfo("TB_AD_CIDADAO_HISTORICO", "CO_SEQ_AD_CIDADAO_HISTORICO", 0
L));
   143:         tableSet.add(new TableInfo("TB_AD_DESTINO", "CO_SEQ_AD_DESTINO", 9L));
   144:         tableSet.add(new TableInfo("TB_AD_MODALIDADE", "CO_SEQ_AD_MODALIDADE", 5L));
   145:         tableSet.add(new TableInfo("TB_AD_ORIGEM", "CO_AD_ORIGEM", 16L));
   146:         tableSet.add(new TableInfo("TB_AD_QUESTAO", "CO_SEQ_AD_QUESTAO", 0L));
   147:         tableSet.add(new TableInfo("TB_AD_RESPOSTA", "CO_SEQ_AD_RESPOSTA", 0L));
   148:         tableSet.add(new TableInfo("TB_AD_SYNC_ENTITY", "CO_SEQ_AD_SYNC_ENTITY", 0L));
   149:         tableSet.add(new TableInfo("TB_AD_TIPO_ELEGIVEL", "CO_AD_TIPO_ELEGIVEL", 4L));
   150:         tableSet.add(new TableInfo("TB_AD_TIPO_INELEGIVEL", "CO_AD_TIPO_INELEGIVEL", 5L));
   151:         tableSet.add(new TableInfo("TB_AD_TRANSMISSAO_SESSAO", "CO_UNICO_TRANSMISSAO_SESSAO"
, 0L));
   152:         tableSet.add(new TableInfo("TB_AGENDADO", "CO_SEQ_AGENDADO", 0L));
   153:         tableSet.add(new TableInfo("TB_ALERGIA", "CO_SEQ_ALERGIA", 0L));
   154:         tableSet.add(new TableInfo("TB_ALERGIA_EVOLUCAO", "CO_SEQ_ALERGIA_EVOLUCAO", 0L));
   155:         tableSet.add(new TableInfo("TB_ALIM_BEBIDA", "CO_ALIM_BEBIDA", 10L));
   156:         tableSet.add(new TableInfo("TB_ANTECEDENTE", "CO_PRONTUARIO", 0L));
   157:         tableSet.add(new TableInfo("TB_ANTECEDENTE_HISTORICO", "CO_SEQ_ANTECEDENTE_HISTORICO
```

```
", 0L));
   158:         tableSet.add(new TableInfo("TB_ANTECEDENTE_ITEM", "CO_SEQ_ANTECEDENTE_ITEM", 0L));
   159:         tableSet.add(new TableInfo("TB_ANTECEDENTE_TIPO_ITEM", "CO_ANTECEDENTE_TIPO_ITEM", 28L)
);
   160:         tableSet.add(new TableInfo("TB_APLICACAO_MEDICAMENTO", "CO_APLICACAO_MEDICAMENTO",
27L));
   161:         tableSet.add(new TableInfo("TB_ARCADA", "CO_PARTE_BUCAL", 1L));
   162:         tableSet.add(new TableInfo("TB_ATEND", "CO_SEQ_ATEND", 0L));
   163:         tableSet.add(new TableInfo("TB_ATEND_PROF", "CO_SEQ_ATEND_PROF", 0L));
   164:         tableSet.add(new TableInfo("TB_ATEND_PROF_AD", "CO_ATEND_PROF_AD", 0L));
   165:         tableSet.add(new TableInfo("TB_ATEND_PROF_AD_QST", "CO_SEQ_ATEND_PROF_AD_QST", 0L));
   166:         tableSet.add(new TableInfo("TB_ATEND_PROF_ODONTO", "CO_ATEND_PROF_ODONTO", 0L));
   167:         tableSet.add(new TableInfo("TB_ATEND_PROF_PRE_NATAL", "CO_ATEND_PROF_PRE_NATAL", 0L));
   168:         tableSet.add(new TableInfo("TB_ATEND_PROF_PUERICULTURA", "CO_ATEND_PROF_PUERICULTUR
A", 0L));
   169:         tableSet.add(new TableInfo("TB_ATEND_PROF_TIPO_ENCAM_INTRN", "CO_SEQ_ATEND_PROF_TIPO
_ENC_INT", 0L));
   170:         tableSet.add(new TableInfo("TB_ATEND_PROF_VACINACAO", "CO_ATEND_PROF_VACINACAO", 0L))
;
   171:         tableSet.add(new TableInfo("TB_ATESTADO", "CO_SEQ_ATESTADO", 0L));
   172:         tableSet.add(new TableInfo("TB_ATIVACAO_AGENDAMENTO_ONLINE", "CO_SEQ_ATIVACAO_AGEN
DAMENTO_ON", 0L));
   173:         tableSet.add(new TableInfo("TB_ATOR", "CO_SEQ_ATOR", 940L));
   174:         tableSet.add(new TableInfo("TB_ATOR_PAPEL", "CO_SEQ_ATOR_PAPEL", 33352L));
   175:         tableSet.add(new TableInfo("TB_ATRIBUTO_COMPLEM", "CO_ATRIBUTO_COMPLEM", 41L));
   176:         tableSet.add(new TableInfo("TB_BAIRRO", "CO_BAIRRO", 50685L));
   177:         tableSet.add(new TableInfo("TB_CATEGORIA_AGENTE_CAUSADOR", "CO_CATEGORIA_AGENTE_CA
USADOR", 6L));
   178:         tableSet.add(new TableInfo("TB_CBO", "CO_CBO", 2609L));
   179:         tableSet.add(new TableInfo("TB_CBO_ATEND", "CO_SEQ_CBO_ATEND", 0L));
   180:         tableSet.add(new TableInfo("TB_CDS_ADM_MEDICAMENTO", "CO_CDS_ADM_MEDICAMENTO", 4426
L));
   181:         tableSet.add(new TableInfo("TB_CDS_ALEITAMENTO_MATERNO", "CO_CDS_ALEITAMENTO_MATE
RNO", 4L));
   182:         tableSet.add(new TableInfo("TB_CDS_ATEND_DOMICILIAR", "CO_SEQ_CDS_ATEND_DOMICILIAR", 0
L));
   183:         tableSet.add(new TableInfo("TB_CDS_ATEND_INDIVIDUAL", "CO_SEQ_CDS_ATEND_INDIVIDUAL", 0
L));
   184:         tableSet.add(new TableInfo("TB_CDS_ATEND_ODONTO", "CO_SEQ_CDS_ATEND_ODONTO", 0L));
   185:         tableSet.add(new TableInfo("TB_CDS_ATIV_COL_PARTICIPANTE", "CO_SEQ_CDS_ATIV_COL_PARTI
CIPNT", 0L));
   186:         tableSet.add(new TableInfo("TB_CDS_ATIV_COL_PRATICA", "CO_CDS_ATIV_COL_PRATICA", 30L));
   187:         tableSet.add(new TableInfo("TB_CDS_ATIV_COL_PUBLICO_ALVO", "CO_CDS_ATIV_COL_PUBLICO_
ALVO", 18L));
   188:         tableSet.add(new TableInfo("TB_CDS_ATIV_COL_TEMA", "CO_CDS_ATIV_COL_TEMA", 7L));
   189:         tableSet.add(new TableInfo("TB_CDS_AVAL_ELEGIBILIDADE", "CO_SEQ_CDS_AVAL_ELEGIBILIDAD
E", 0L));
   190:         tableSet.add(new TableInfo("TB_CDS_CAD_DOMICILIAR", "CO_SEQ_CDS_CAD_DOMICILIAR", 0L));
   191:         tableSet.add(new TableInfo("TB_CDS_CAD_INDIVIDUAL", "CO_SEQ_CDS_CAD_INDIVIDUAL", 0L));
   192:         tableSet.add(new TableInfo("TB_CDS_CIDADAO_RESPOSTA", "CO_SEQ_CDS_CIDADAO_RESPOSTA",
0L));
   193:         tableSet.add(new TableInfo("TB_CDS_DOMICILIO", "CO_SEQ_CDS_DOMICILIO", 0L));
   194:         tableSet.add(new TableInfo("TB_CDS_DOMICILIO_FAMILIA", "CO_SEQ_CDS_DOMICILIO_FAMILIA",
0L));
   195:         tableSet.add(new TableInfo("TB_CDS_DOMICILIO_RESPOSTA", "CO_SEQ_CDS_DOMICILIO_RESPOST
A", 0L));
   196:         tableSet.add(new TableInfo("TB_CDS_FICHA_ATEND_DOMICILIAR", "CO_SEQ_CDS_FICHA_ATEND_
DOM", 0L));
   197:         tableSet.add(new TableInfo("TB_CDS_FICHA_ATEND_INDIVIDUAL", "CO_SEQ_CDS_FICHA_ATEND_I
NDIVDL", 0L));
   198:         tableSet.add(new TableInfo("TB_CDS_FICHA_ATEND_ODONTO", "CO_SEQ_CDS_FICHA_ATEND_ODO
NTO", 0L));
   199:         tableSet.add(new TableInfo("TB_CDS_FICHA_ATIV_COL", "CO_SEQ_CDS_FICHA_ATIV_COL", 0L));
   200:         tableSet.add(new TableInfo("TB_CDS_FICHA_CONSUMO_ALIMENTAR", "CO_SEQ_CDS_FICHA_CONS
UMO_ALIM", 0L));
   201:         tableSet.add(new TableInfo("TB_CDS_FICHA_PROCED", "CO_SEQ_CDS_FICHA_PROCED", 0L));
   202:         tableSet.add(new TableInfo("TB_CDS_FICHA_VACINACAO", "CO_SEQ_CDS_FICHA_VACINACAO", 0L)
);
```

```
   203:         tableSet.add(new TableInfo("TB_CDS_FICHA_VISITA_DOMICILIAR", "CO_SEQ_CDS_FICHA_VISITA_D
OM", 0L));
   204:         tableSet.add(new TableInfo("TB_CDS_FICHA_ZIKA_MICROCEFALIA", "CO_SEQ_CDS_FICHA_ZICA_M
ICRCFL", 0L));
   205:         tableSet.add(new TableInfo("TB_CDS_PIC", "CO_CDS_PIC", 8L));
   206:         tableSet.add(new TableInfo("TB_CDS_PROCED", "CO_SEQ_CDS_PROCED", 0L));
   207:         tableSet.add(new TableInfo("TB_CDS_PROF", "CO_SEQ_CDS_PROF", 0L));
   208:         tableSet.add(new TableInfo("TB_CDS_TIPO_ATEND_NASF", "CO_CDS_TIPO_ATEND_NASF", 3L));
   209:         tableSet.add(new TableInfo("TB_CDS_TIPO_ATIV_COL", "CO_CDS_TIPO_ATIV_COL", 7L));
   210:         tableSet.add(new TableInfo("TB_CDS_TIPO_CONDUTA", "CO_CDS_TIPO_CONDUTA", 12L));
   211:         tableSet.add(new TableInfo("TB_CDS_TIPO_CUIDADOR", "CO_CDS_TIPO_CUIDADOR", 8L));
   212:         tableSet.add(new TableInfo("TB_CDS_TIPO_IMOVEL", "CO_CDS_TIPO_IMOVEL", 99L));
   213:         tableSet.add(new TableInfo("TB_CDS_TIPO_ORIGEM", "CO_CDS_TIPO_ORIGEM", 4L));
   214:         tableSet.add(new TableInfo("TB_CDS_TIPO_SITUACAO_PRESENTE", "CO_CDS_TIPO_SITUACAO_PRE
SENTE", 24L));
   215:         tableSet.add(new TableInfo("TB_CDS_TIPO_VIG_SAUDE_BUCAL", "CO_CDS_TIPO_VIG_SAUDE_BUC
AL", 99L));
   216:         tableSet.add(new TableInfo("TB_CDS_TURNO", "CO_CDS_TURNO", 3L));
   217:         tableSet.add(new TableInfo("TB_CDS_VACINA", "CO_SEQ_CDS_VACINA", 0L));
   218:         tableSet.add(new TableInfo("TB_CDS_VACINACAO", "CO_SEQ_CDS_VACINACAO", 0L));
   219:         tableSet.add(new TableInfo("TB_CDS_VISITA_DOMICILIAR", "CO_SEQ_CDS_VISITA_DOMICILIAR", 0
L));
   220:         tableSet.add(new TableInfo("TB_CDS_VISITA_DOM_DESFECHO", "CO_CDS_VISITA_DOM_DESFECHO
", 3L));
   221:         tableSet.add(new TableInfo("TB_CDS_VISITA_DOM_MOTIVO", "CO_CDS_VISITA_DOM_MOTIVO", 37L
));
   222:         tableSet.add(new TableInfo("TB_CIAP", "CO_SEQ_CIAP", 751L));
   223:         tableSet.add(new TableInfo("TB_CIAP_CAPITULO", "CO_SEQ_CIAP_CAPITULO", 18L));
   224:         tableSet.add(new TableInfo("TB_CIAP_COMPONENTE", "CO_CIAP_COMPONENTE", 7L));
   225:         tableSet.add(new TableInfo("TB_CIAP_DAB", "CO_CIAP", 751L));
   226:         tableSet.add(new TableInfo("TB_CIAP_MS", "CO_CIAP", 726L));
   227:         tableSet.add(new TableInfo("TB_CID10", "CO_CID10", 14232L));
   228:         tableSet.add(new TableInfo("TB_CIDADAO", "CO_ATOR_PAPEL", 0L));
   229:         tableSet.add(new TableInfo("TB_CIDADAO_NUCLEO_FAMILIAR", "CO_SEQ_CIDADAO_NUCLEO_FA
MILIAR", 0L));
   230:         tableSet.add(new TableInfo("TB_CLASSE_IMUNOBIOLOGICO", "CO_CLASSE_IMUNOBIOLOGICO", 3L)
);
   231:         tableSet.add(new TableInfo("TB_CLASSIFICACAO_RISCO", "CO_CLASSIFICACAO_RISCO", 9L));
   232:         tableSet.add(new TableInfo("TB_CLASSIFICACAO_RISCO_ENCAM", "CO_CLASSIFICACAO_RISCO_EN
CAM", 3L));
   233:         tableSet.add(new TableInfo("TB_CNS", "CO_SEQ_CNS", 0L));
   234:         tableSet.add(new TableInfo("TB_COMPARTILHAMENTO_PRONTUARIO", "CO_SEQ_COMPARTILHA_P
RONTUARIO", 0L));
   235:         tableSet.add(new TableInfo("TB_COMPLEXIDADE", "CO_COMPLEXIDADE", 6L));
   236:         tableSet.add(new TableInfo("TB_CONFIGURACAO_HORARIO", "CO_SEQ_CONFIG_PERIODO", 0L));
   237:         tableSet.add(new TableInfo("TB_CONFIG_AGENDA_DETALHE", "CO_SEQ_CONFIG_AGENDA_DETAL
HE", 0L));
   238:         tableSet.add(new TableInfo("TB_CONFIG_AGENDA_FECHAMENTO", "CO_SEQ_CONFIG_AGENDA_FE
CHAMENT", 0L));
   239:         tableSet.add(new TableInfo("TB_CONFIG_ATENCAO_DOMICILIAR", "CO_SEQ_CONFIG_ATENCAO_D
OMICILR", 0L));
   240:         tableSet.add(new TableInfo("TB_CONFIG_ATEND_DOMICILIAR", "CO_SEQ_CONFIG_ATEND_DOMICI
LIAR", 0L));
   241:         tableSet.add(new TableInfo("TB_CONSELHO_CLASSE", "CO_CONSELHO_CLASSE", 83L));
   242:         tableSet.add(new TableInfo("TB_CONTATO", "CO_SEQ_CONTATO", 0L));
   243:         tableSet.add(new TableInfo("TB_CONTEXTO_PERGUNTA", "CO_CONTEXTO_PERGUNTA", 2L));
   244:         tableSet.add(new TableInfo("TB_CRITICIDADE_ALERGIA", "CO_CRITICIDADE_ALERGIA", 2L));
   245:         tableSet.add(new TableInfo("TB_CRONICIDADE", "CO_CRONICIDADE", 2L));
   246:         tableSet.add(new TableInfo("TB_DADO_RECEBIDO_COMPETENCIA", "CO_SEQ_DADO_RECEBIDO_C
OMPETENC", 0L));
   247:         tableSet.add(new TableInfo("TB_DADO_RECEBIDO_INFO_INSTALAC", "CO_SEQ_DADO_RECEBIDO_I
NFO_INST", 0L));
   248:         tableSet.add(new TableInfo("TB_DADO_REL_PROCESSAMENTO", "CO_SEQ_DADO_REL_PROCESSAM
ENTO", 0L));
   249:         tableSet.add(new TableInfo("TB_DADO_TRANSP", "CO_SEQ_DADO_TRANSP", 0L));
   250:         tableSet.add(new TableInfo("TB_DADO_TRANSP_RECEBIDO", "CO_SEQ_DADO_TRANSP_RECEBIDO",
 0L));
   251:         tableSet.add(new TableInfo("TB_DADO_TRANSP_RECEBIDO_ONLINE", "CO_SEQ_DADO_TRANSP_RE
```

```
CEB_ONLIN", 0L));
    252:        tableSet.add(new TableInfo("TB_DENTE", "CO_PARTE_BUCAL", 59L));
    253:        tableSet.add(new TableInfo("TB_DIA_SEMANA", "CO_DIA_SEMANA", 6L));
    254:        tableSet.add(new TableInfo("TB_DIM_AGRUPADOR_FILTRO", "CO_SEQ_DIM_AGRUPADOR_FILTRO",
0L));
    255:        tableSet.add(new TableInfo("TB_DIM_ALEITAMENTO", "CO_SEQ_DIM_ALEITAMENTO", 5L));
    256:        tableSet.add(new TableInfo("TB_DIM_CBO", "CO_SEQ_DIM_CBO", 0L));
    257:        tableSet.add(new TableInfo("TB_DIM_CIAP", "CO_SEQ_DIM_CIAP", 0L));
    258:        tableSet.add(new TableInfo("TB_DIM_CID", "CO_SEQ_DIM_CID", 0L));
    259:        tableSet.add(new TableInfo("TB_DIM_CUIDADOR", "CO_SEQ_DIM_CUIDADOR", 8L));
    260:        tableSet.add(new TableInfo("TB_DIM_DESFECHO_VISITA", "CO_SEQ_DIM_DESFECHO_VISITA", 4L));
    261:        tableSet.add(new TableInfo("TB_DIM_DOSE_IMUNOBIOLOGICO", "CO_SEQ_DIM_DOSE_IMUNOBIOL
OGICO", 38L));
    262:        tableSet.add(new TableInfo("TB_DIM_EQUIPE", "CO_SEQ_DIM_EQUIPE", 0L));
    263:        tableSet.add(new TableInfo("TB_DIM_ESTRATEGIA_VACINACAO", "CO_SEQ_DIM_ESTRATEGIA_VA
CNACAO", 10L));
    264:        tableSet.add(new TableInfo("TB_DIM_ETNIA", "CO_SEQ_DIM_ETNIA", 265L));
    265:        tableSet.add(new TableInfo("TB_DIM_FAIXA_ETARIA", "CO_SEQ_DIM_FAIXA_ETARIA", 22L));
    266:        tableSet.add(new TableInfo("TB_DIM_FREQUENCIA_ALIMENTACAO", "CO_SEQ_DIM_FREQUENCIA_
ALIMENT", 4L));
    267:        tableSet.add(new TableInfo("TB_DIM_GRUPO_CBO", "CO_SEQ_DIM_GRUPO_CBO", 0L));
    268:        tableSet.add(new TableInfo("TB_DIM_IDENTIDADE_GENERO", "CO_SEQ_DIM_IDENTIDADE_GENERO
", 5L));
    269:        tableSet.add(new TableInfo("TB_DIM_IMUNOBIOLOGICO", "CO_SEQ_DIM_IMUNOBIOLOGICO", 0L));
    270:        tableSet.add(new TableInfo("TB_DIM_LOCAL_ATENDIMENTO", "CO_SEQ_DIM_LOCAL_ATENDIMEN
TO", 16L));
    271:        tableSet.add(new TableInfo("TB_DIM_MODALIDADE_AD", "CO_SEQ_DIM_MODALIDADE_AD", 5L));
    272:        tableSet.add(new TableInfo("TB_DIM_MUNICIPIO", "CO_SEQ_DIM_MUNICIPIO", 0L));
    273:        tableSet.add(new TableInfo("TB_DIM_NACIONALIDADE", "CO_SEQ_DIM_NACIONALIDADE", 4L));
    274:        tableSet.add(new TableInfo("TB_DIM_PAIS", "CO_SEQ_DIM_PAIS", 239L));
    275:        tableSet.add(new TableInfo("TB_DIM_PIC", "CO_SEQ_DIM_PIC", 9L));
    276:        tableSet.add(new TableInfo("TB_DIM_PROCEDENCIA_ORIGEM", "CO_SEQ_DIM_PROCEDENCIA_ORI
GEM", 6L));
    277:        tableSet.add(new TableInfo("TB_DIM_PROCEDIMENTO", "CO_SEQ_DIM_PROCEDIMENTO", 0L));
    278:        tableSet.add(new TableInfo("TB_DIM_PROFISSIONAL", "CO_SEQ_DIM_PROFISSIONAL", 0L));
    279:        tableSet.add(new TableInfo("TB_DIM_RACA_COR", "CO_SEQ_DIM_RACA_COR", 6L));
    280:        tableSet.add(new TableInfo("TB_DIM_RACIONALIDADE_SAUDE", "CO_SEQ_DIM_RACIONALIDADE_
SAUDE", 7L));
    281:        tableSet.add(new TableInfo("TB_DIM_SEXO", "CO_SEQ_DIM_SEXO", 4L));
    282:        tableSet.add(new TableInfo("TB_DIM_SITUACAO_TRABALHO", "CO_SEQ_DIM_SITUACAO_TRABALH
O", 12L));
    283:        tableSet.add(new TableInfo("TB_DIM_TEMPO", "CO_SEQ_DIM_TEMPO", 0L));
    284:        tableSet.add(new TableInfo("TB_DIM_TEMPO_MORADOR_RUA", "CO_SEQ_DIM_TEMPO_MORADOR_
RUA", 5L));
    285:        tableSet.add(new TableInfo("TB_DIM_TIPO_ABASTECIMENTO_AGUA", "CO_SEQ_DIM_TIPO_ABASTE
C_AGUA", 6L));
    286:        tableSet.add(new TableInfo("TB_DIM_TIPO_ACESSO_DOMICILIO", "CO_SEQ_DIM_TIPO_ACESSO_DO
MICIL", 5L));
    287:        tableSet.add(new TableInfo("TB_DIM_TIPO_ATENDIMENTO", "CO_SEQ_DIM_TIPO_ATENDIMENTO",
11L));
    288:        tableSet.add(new TableInfo("TB_DIM_TIPO_ATIVIDADE", "CO_SEQ_DIM_TIPO_ATIVIDADE", 8L));
    289:        tableSet.add(new TableInfo("TB_DIM_TIPO_CONDICAO_PESO", "CO_SEQ_DIM_TIPO_CONDICAO_PES
O", 4L));
    290:        tableSet.add(new TableInfo("TB_DIM_TIPO_CONSULTA", "CO_SEQ_DIM_TIPO_CONSULTA", 6L));
    291:        tableSet.add(new TableInfo("TB_DIM_TIPO_DESTINO_LIXO", "CO_SEQ_DIM_TIPO_DESTINO_LIXO", 5
L));
    292:        tableSet.add(new TableInfo("TB_DIM_TIPO_DOMICILIO", "CO_SEQ_DIM_TIPO_DOMICILIO", 5L));
    293:        tableSet.add(new TableInfo("TB_DIM_TIPO_ELEGIBILIDADE", "CO_SEQ_DIM_TIPO_ELEGIBILIDADE",
 4L));
    294:        tableSet.add(new TableInfo("TB_DIM_TIPO_ESCOAMENTO_SANITAR", "CO_SEQ_DIM_TIPO_ESCOAM
ENTO_SNT", 7L));
    295:        tableSet.add(new TableInfo("TB_DIM_TIPO_ESCOLARIDADE", "CO_SEQ_DIM_TIPO_ESCOLARIDADE"
, 16L));
    296:        tableSet.add(new TableInfo("TB_DIM_TIPO_FICHA", "CO_SEQ_DIM_TIPO_FICHA", 15L));
    297:        tableSet.add(new TableInfo("TB_DIM_TIPO_IMOVEL", "CO_SEQ_DIM_TIPO_IMOVEL", 13L));
    298:        tableSet.add(new TableInfo("TB_DIM_TIPO_LOCALIZACAO", "CO_SEQ_DIM_TIPO_LOCALIZACAO", 3
L));
    299:        tableSet.add(new TableInfo("TB_DIM_TIPO_LOGRADOURO", "CO_SEQ_DIM_TIPO_LOGRADOURO", 28
```

```
2L));
   300:         tableSet.add(new TableInfo("TB_DIM_TIPO_MATERIAL_PAREDE", "CO_SEQ_DIM_TIPO_MATERIAL_P
ARED", 9L));
   301:         tableSet.add(new TableInfo("TB_DIM_TIPO_ORIENTACAO_SEXUAL", "CO_SEQ_DIM_TIPO_ORIENTA
CAO_SXL", 5L));
   302:         tableSet.add(new TableInfo("TB_DIM_TIPO_ORIGEM_DADO_TRANSP", "CO_SEQ_DIM_TP_ORGM_DA
DO_TRANSP", 6L));
   303:         tableSet.add(new TableInfo("TB_DIM_TIPO_PARENTESCO", "CO_SEQ_DIM_TIPO_PARENTESCO", 11L)
);
   304:         tableSet.add(new TableInfo("TB_DIM_TIPO_POSSE_TERRA", "CO_SEQ_DIM_TIPO_POSSE_TERRA", 9L)
);
   305:         tableSet.add(new TableInfo("TB_DIM_TIPO_RENDA_FAMILIAR", "CO_SEQ_DIM_TIPO_RENDA_FAMIL
IAR", 8L));
   306:         tableSet.add(new TableInfo("TB_DIM_TIPO_SAIDA_CADASTRO", "CO_SEQ_DIM_TIPO_SAIDA_CADAS
TRO", 3L));
   307:         tableSet.add(new TableInfo("TB_DIM_TIPO_SITUACAO_MORADIA", "CO_SEQ_DIM_TIPO_SITUACAO_
MORAD", 9L));
   308:         tableSet.add(new TableInfo("TB_DIM_TIPO_TRATAMENTO_AGUA", "CO_SEQ_DIM_TIPO_TRATAMEN
T_AGUA", 6L));
   309:         tableSet.add(new TableInfo("TB_DIM_TURNO", "CO_SEQ_DIM_TURNO", 4L));
   310:         tableSet.add(new TableInfo("TB_DIM_UF", "CO_SEQ_DIM_UF", 0L));
   311:         tableSet.add(new TableInfo("TB_DIM_UNIDADE_SAUDE", "CO_SEQ_DIM_UNIDADE_SAUDE", 0L));
   312:         tableSet.add(new TableInfo("TB_DOSE_IMUNOBIOLOGICO", "CO_DOSE_IMUNOBIOLOGICO", 38L));
   313:         tableSet.add(new TableInfo("TB_ENCAMINHAMENTO", "CO_SEQ_ENCAMINHAMENTO", 0L));
   314:         tableSet.add(new TableInfo("TB_ENDERECO", "CO_SEQ_ENDERECO", 927L));
   315:         tableSet.add(new TableInfo("TB_ENDERECO_EXTERIOR", "CO_SEQ_ENDERECO_EXTERIOR", 0L));
   316:         tableSet.add(new TableInfo("TB_ENVIO_LOG", "CO_SEQ_ENVIO_LOG", 0L));
   317:         tableSet.add(new TableInfo("TB_EQUIPE", "CO_SEQ_EQUIPE", 58L));
   318:         tableSet.add(new TableInfo("TB_ESCOLARIDADE", "CO_ESCOLARIDADE", 15L));
   319:         tableSet.add(new TableInfo("TB_ESPECIALIDADE_SISREG", "CO_ESPECIALIDADE_SISREG", 76L));
   320:         tableSet.add(new TableInfo("TB_ESTADO_CIVIL", "CO_ESTADO_CIVIL", 5L));
   321:         tableSet.add(new TableInfo("TB_ESTRATEGIA_VACINACAO", "CO_ESTRATEGIA_VACINACAO", 10L))
;
   322:         tableSet.add(new TableInfo("TB_ETNIA", "CO_ETNIA", 264L));
   323:         tableSet.add(new TableInfo("TB_EVOLUCAO_AVALIACAO", "CO_ATEND_PROF", 0L));
   324:         tableSet.add(new TableInfo("TB_EVOLUCAO_DENTE", "CO_SEQ_EVOLUCAO_DENTE", 0L));
   325:         tableSet.add(new TableInfo("TB_EVOLUCAO_OBJETIVO", "CO_ATEND_PROF", 0L));
   326:         tableSet.add(new TableInfo("TB_EVOLUCAO_ODONTO", "CO_SEQ_EVOLUCAO_ODONTO", 0L));
   327:         tableSet.add(new TableInfo("TB_EVOLUCAO_PLANO", "CO_ATEND_PROF", 0L));
   328:         tableSet.add(new TableInfo("TB_EVOLUCAO_SUBJETIVO", "CO_ATEND_PROF", 0L));
   329:         tableSet.add(new TableInfo("TB_EXAME_DETALHE", "CO_SEQ_EXAME_DETALHE", 1L));
   330:         tableSet.add(new TableInfo("TB_EXAME_DETALHE_RESULTADO", "CO_SEQ_EXAME_DETALHE_RES
ULTADO", 0L));
   331:         tableSet.add(new TableInfo("TB_EXAME_HEMOGLOBINA_GLICADA", "CO_SEQ_EXAME_HEMOGLOB
INA_GLICD", 0L));
   332:         tableSet.add(new TableInfo("TB_EXAME_PRENATAL", "CO_SEQ_EXAME_PRENATAL", 0L));
   333:         tableSet.add(new TableInfo("TB_EXAME_PUERICULTURA", "CO_SEQ_EXAME", 0L));
   334:         tableSet.add(new TableInfo("TB_EXAME_REQUISITADO", "CO_SEQ_EXAME_REQUISITADO", 0L));
   335:         tableSet.add(new TableInfo("TB_FAIXA_ETARIA_VACINACAO", "CO_FAIXA_ETARIA_VACINACAO",
184L));
   336:         tableSet.add(new TableInfo("TB_FAMILIA", "CO_SEQ_FAMILIA", 0L));
   337:         tableSet.add(new TableInfo("TB_FAT_ATD_IND_PROBLEMAS", "CO_SEQ_FAT_ATEND_IND_PROBLE
MAS", 0L));
   338:         tableSet.add(new TableInfo("TB_FAT_ATD_IND_PROCEDIMENTOS", "CO_SEQ_FAT_ATEND_IND_PRO
CED", 0L));
   339:         tableSet.add(new TableInfo("TB_FAT_ATENDIMENTO_DOMICILIAR", "CO_SEQ_FAT_ATEND_DOMICI
LIAR", 0L));
   340:         tableSet.add(new TableInfo("TB_FAT_ATENDIMENTO_INDIVIDUAL", "CO_SEQ_FAT_ATD_IND", 0L));
   341:         tableSet.add(new TableInfo("TB_FAT_ATENDIMENTO_ODONTO", "CO_SEQ_FAT_ATD_ODNT", 0L));
   342:         tableSet.add(new TableInfo("TB_FAT_ATEND_DOM_PROBLEMA", "CO_SEQ_FAT_ATEND_DOM_PRO
BLEMA", 0L));
   343:         tableSet.add(new TableInfo("TB_FAT_ATEND_DOM_PROCED", "CO_SEQ_FAT_ATEND_DOM_PROCED
", 0L));
   344:         tableSet.add(new TableInfo("TB_FAT_ATEND_ODONTO_PROBLEMAS", "CO_SEQ_FAT_ATND_ODONT
O_PROBL", 0L));
   345:         tableSet.add(new TableInfo("TB_FAT_ATEND_ODONTO_PROCED", "CO_SEQ_FAT_ATEND_ODONTO_
PROCED", 0L));
   346:         tableSet.add(new TableInfo("TB_FAT_ATIVIDADE_COLETIVA", "CO_SEQ_FAT_ATIVIDADE_COLETIV
```

```
A", 0L));
   347:         tableSet.add(new TableInfo("TB_FAT_ATVDD_COLETIVA_EXT", "CO_SEQ_FAT_ATVDD_CLTV_EXT",
0L));
   348:         tableSet.add(new TableInfo("TB_FAT_ATVDD_COLETIVA_INT", "CO_SEQ_FAT_ATVDD_CLTV_INT", 0
L));
   349:         tableSet.add(new TableInfo("TB_FAT_AVALIACAO_ELEGIBILIDADE", "CO_SEQ_FAT_AVALIACAO_E
LEGIBLDD", 0L));
   350:         tableSet.add(new TableInfo("TB_FAT_CAD_DOMICILIAR", "CO_SEQ_FAT_CAD_DOMICILIAR", 0L));
   351:         tableSet.add(new TableInfo("TB_FAT_CAD_DOM_FAMILIA", "CO_SEQ_FAT_CAD_DOM_FAMILIA", 0L
));
   352:         tableSet.add(new TableInfo("TB_FAT_CAD_INDIVIDUAL", "CO_SEQ_FAT_CAD_INDIVIDUAL", 0L));
   353:         tableSet.add(new TableInfo("TB_FAT_CIDADAO", "CO_SEQ_FAT_CIDADAO", 0L));
   354:         tableSet.add(new TableInfo("TB_FAT_COMPLEMENTAR", "CO_SEQ_FAT_COMPLEMENTAR", 0L));
   355:         tableSet.add(new TableInfo("TB_FAT_FAMILIA", "CO_SEQ_FAT_FAMILIA", 0L));
   356:         tableSet.add(new TableInfo("TB_FAT_FICHAS", "CO_SEQ_FAT_FICHAS", 0L));
   357:         tableSet.add(new TableInfo("TB_FAT_MARCA_CONSUMO_ALIMNT", "CO_SEQ_FAT_MARCA_CON_A
LMNT", 0L));
   358:         tableSet.add(new TableInfo("TB_FAT_PROCEDIMENTO", "CO_SEQ_FAT_PROCEDIMENTO", 0L));
   359:         tableSet.add(new TableInfo("TB_FAT_PROCED_ATEND", "CO_SEQ_FAT_PROCED_ATEND", 0L));
   360:         tableSet.add(new TableInfo("TB_FAT_PROCED_ATEND_PROCED", "CO_SEQ_FAT_PROCED_ATEND_P
ROCED", 0L));
   361:         tableSet.add(new TableInfo("TB_FAT_VACINACAO", "CO_SEQ_FAT_VACINACAO", 0L));
   362:         tableSet.add(new TableInfo("TB_FAT_VACINACAO_VACINA", "CO_SEQ_FAT_VACINACAO_VACINA",
 0L));
   363:         tableSet.add(new TableInfo("TB_FAT_VISITA_DOMICILIAR", "CO_SEQ_FAT_VISITA_DOMICILIAR", 0
L));
   364:         tableSet.add(new TableInfo("TB_FICHA_ZIKA_TIPO_EXAME", "CO_FICHA_ZIKA_TIPO_EXAME", 18L))
;
   365:         tableSet.add(new TableInfo("TB_FORMA_FARMACEUTICA", "CO_FORMA_FARMACEUTICA", 115L));
   366:         tableSet.add(new TableInfo("TB_GESTOR_ESTADUAL", "CO_ATOR_PAPEL", 0L));
   367:         tableSet.add(new TableInfo("TB_GESTOR_MUNICIPAL", "CO_ATOR_PAPEL", 0L));
   368:         tableSet.add(new TableInfo("TB_GRAVIDADE", "CO_GRAVIDADE", 2L));
   369:         tableSet.add(new TableInfo("TB_GRUPO_ESPECIALIDADE", "CO_GRUPO_ESPECIALIDADE", 3L));
   370:         tableSet.add(new TableInfo("TB_HISTORICO_RELATORIO", "CO_SEQ_HISTORICO_RELATORIO", 9L));
   371:         tableSet.add(new TableInfo("TB_IMUNOBIOLOGICO", "CO_IMUNOBIOLOGICO", 84L));
   372:         tableSet.add(new TableInfo("TB_INTEGRACAO_HORUS", "CO_UNIDADE_SAUDE", 0L));
   373:         tableSet.add(new TableInfo("TB_JUSTIFICATIVA_AGENDA", "CO_SEQ_JUSTIFICATIVA_AGENDAMNT
", 0L));
   374:         tableSet.add(new TableInfo("TB_JUSTIFICATIVA_PRONTUARIO", "CO_SEQ_JUSTIFICATIVA_PRONTU
AR", 0L));
   375:         tableSet.add(new TableInfo("TB_JUSTIFICATIVA_STATUS_CIDDAO", "CO_SEQ_JUSTIFICA_INATIV_C
IDADA", 0L));
   376:         tableSet.add(new TableInfo("TB_LEMBRETE", "CO_SEQ_LEMBRETE", 0L));
   377:         tableSet.add(new TableInfo("TB_LEMBRETE_EVOLUCAO", "CO_SEQ_LEMBRETE_EVOLUCAO", 0L));
   378:         tableSet.add(new TableInfo("TB_LISTA_MEDICAMENTO", "CO_LISTA_MEDICAMENTO", 14L));
   379:         tableSet.add(new TableInfo("TB_LOCALIDADE", "CO_LOCALIDADE", 10828L));
   380:         tableSet.add(new TableInfo("TB_LOCAL_APL_VACINA", "CO_LOCAL_APL_VACINA", 21L));
   381:         tableSet.add(new TableInfo("TB_LOCAL_ATEND", "CO_LOCAL_ATEND", 15L));
   382:         tableSet.add(new TableInfo("TB_LOGRADOURO", "CO_LOGRADOURO", 1013205L));
   383:         tableSet.add(new TableInfo("TB_LOTACAO", "CO_ATOR_PAPEL", 33352L));
   384:         tableSet.add(new TableInfo("TB_LOTE_TRANSP", "CO_SEQ_LOTE_TRANSP", 0L));
   385:         tableSet.add(new TableInfo("TB_LOTE_TRANSP_HISTORICO_EXPRT", "CO_SEQ_LOTE_TRANSP_HIST
ORC_EXP", 0L));
   386:         tableSet.add(new TableInfo("TB_LOTE_TRANSP_ITEM", "CO_SEQ_LOTE_TRANSP_ITEM", 0L));
   387:         tableSet.add(new TableInfo("TB_LOTE_TRANSP_ITEM_NODO", "CO_SEQ_LOTE_TRANSP_ITEM_NOD
O", 0L));
   388:         tableSet.add(new TableInfo("TB_LOTE_TRANSP_NODO", "CO_SEQ_LOTE_TRANSP_NODO", 0L));
   389:         tableSet.add(new TableInfo("TB_MEDICAMENTO", "CO_SEQ_MEDICAMENTO", 3338L));
   390:         tableSet.add(new TableInfo("TB_MEDICAMENTO_CATMAT", "CO_MEDICAMENTO_CATMAT", 3338L))
;
   391:         tableSet.add(new TableInfo("TB_MEDICAMENTO_USO_CONTINUO", "CO_SEQ_MEDICAMENT_USO_C
ONTINUO", 0L));
   392:         tableSet.add(new TableInfo("TB_MEDICAO", "CO_SEQ_MEDICAO", 0L));
   393:         tableSet.add(new TableInfo("TB_MEMORIA", "CO_MEMORIA", 0L));
   394:         tableSet.add(new TableInfo("TB_MES", "CO_MES", 11L));
   395:         tableSet.add(new TableInfo("TB_MIGRACAO_ESTRUTURA_TRAVA", "CO_MIGRACAO_ESTRUTURA_
TRAVA", 1L));
   396:         tableSet.add(new TableInfo("TB_MOTIVO_RESERVA", "CO_MOTIVO_RESERVA", 15L));
```

```
   397:         tableSet.add(new TableInfo("TB_NACIONALIDADE", "CO_NACIONALIDADE", 3L));
   398:         tableSet.add(new TableInfo("TB_NODO", "CO_NODO", 0L));
   399:         tableSet.add(new TableInfo("TB_ODONTOGRAMA", "CO_SEQ_ODONTOGRAMA", 0L));
   400:         tableSet.add(new TableInfo("TB_ORIENTACAO", "CO_SEQ_ORIENTACAO", 0L));
   401:         tableSet.add(new TableInfo("TB_ORIGEM", "CO_ORIGEM", 1L));
   402:         tableSet.add(new TableInfo("TB_PAIS", "CO_PAIS", 243L));
   403:         tableSet.add(new TableInfo("TB_PAPEL", "CO_SEQ_PAPEL", 0L));
   404:         tableSet.add(new TableInfo("TB_PARTE_BUCAL", "CO_PARTE_BUCAL", 59L));
   405:         tableSet.add(new TableInfo("TB_PARTE_BUCAL_PROCED", "CO_PARTE_BUCAL_PROCED", 283L));
   406:         tableSet.add(new TableInfo("TB_PERFIL", "CO_SEQ_PERFIL", 23L));
   407:         tableSet.add(new TableInfo("TB_PERGUNTA", "CO_SEQ_PERGUNTA", 1001L));
   408:         tableSet.add(new TableInfo("TB_PERGUNTA_DETALHE", "CO_PERGUNTA_DETALHE", 156L));
   409:         tableSet.add(new TableInfo("TB_PERIODO", "CO_PERIODO", 3L));
   410:         tableSet.add(new TableInfo("TB_PESSOA_FISICA", "CO_ATOR", 940L));
   411:         tableSet.add(new TableInfo("TB_PESSOA_FISICA_IMAGEM", "CO_SEQ_PESSOA_FISICA_IMAGEM", 0L
));
   412:         tableSet.add(new TableInfo("TB_PESSOA_JURIDICA", "CO_ATOR", 937L));
   413:         tableSet.add(new TableInfo("TB_POVO_COMUNIDADE_TRADICIONAL", "CO_POVO_COMUNIDADE_T
RADICIONAL", 22L));
   414:         tableSet.add(new TableInfo("TB_PRE_NATAL", "CO_SEQ_PRE_NATAL", 0L));
   415:         tableSet.add(new TableInfo("TB_PRINCIPIO_ATIVO", "CO_PRINCIPIO_ATIVO", 1482L));
   416:         tableSet.add(new TableInfo("TB_PROBLEMA", "CO_SEQ_PROBLEMA", 0L));
   417:         tableSet.add(new TableInfo("TB_PROBLEMA_EVOLUCAO", "CO_SEQ_PROBLEMA_EVOLUCAO", 0L));
   418:         tableSet.add(new TableInfo("TB_PROCED", "CO_SEQ_PROCED", 4746L));
   419:         tableSet.add(new TableInfo("TB_PROCED_AUTOMATICO", "CO_SEQ_PROCED_AUTOMATICO", 244L))
;
   420:         tableSet.add(new TableInfo("TB_PROCED_EXAME_ESPECIFICO", "CO_PROCED_EXAME_ESPECIFICO"
, 9L));
   421:         tableSet.add(new TableInfo("TB_PROCED_FILTRO", "CO_PROCED", 3250L));
   422:         tableSet.add(new TableInfo("TB_PROCED_FORMA_ORGANIZACIONAL", "CO_PROCED_FORMA_ORG
ANIZACIONAL", 384L));
   423:         tableSet.add(new TableInfo("TB_PROCED_GRUPO", "CO_PROCED_GRUPO", 8L));
   424:         tableSet.add(new TableInfo("TB_PROCED_NOME_POPULAR", "CO_SEQ_PROCED_NOME_POPULAR",
0L));
   425:         tableSet.add(new TableInfo("TB_PROCED_SOLICITADO", "CO_SEQ_PROCED_SOLICITADO", 0L));
   426:         tableSet.add(new TableInfo("TB_PROCED_SUBGRUPO", "CO_PROCED_SUBGRUPO", 59L));
   427:         tableSet.add(new TableInfo("TB_PROCESSAMENTO_DOM_CID", "CO_SEQ_PROCESSAMENTO_DOM_
CID", 0L));
   428:         tableSet.add(new TableInfo("TB_PROCESSO", "CO_SEQ_PROCESSO", 0L));
   429:         tableSet.add(new TableInfo("TB_PROF", "CO_ATOR_PAPEL", 33351L));
   430:         tableSet.add(new TableInfo("TB_PRONTUARIO", "CO_SEQ_PRONTUARIO", 0L));
   431:         tableSet.add(new TableInfo("TB_PRONTUARIO_UNIDADE_SAUDE", "CO_SEQ_PRONTUARIO_UNIDAD
E_SAUD", 0L));
   432:         tableSet.add(new TableInfo("TB_QST_ASSOCIACAO_PERGUNTA", "CO_QST_ASSOCIACAO_PERGUNT
A", 3L));
   433:         tableSet.add(new TableInfo("TB_QST_OPCAO_PERGUNTA", "CO_QST_OPCAO_PERGUNTA", 122L));
   434:         tableSet.add(new TableInfo("TB_QST_OPCAO_TIPO_PERGUNTA", "CO_QST_OPCAO_TIPO_PERGUNT
A", 17L));
   435:         tableSet.add(new TableInfo("TB_QST_ORIENTACAO_PROF", "CO_QST_ORIENTACAO_PROF", 27L));
   436:         tableSet.add(new TableInfo("TB_QST_PERGUNTA", "CO_QST_PERGUNTA", 41L));
   437:         tableSet.add(new TableInfo("TB_QST_QUESTIONARIO", "CO_QST_QUESTIONARIO", 3L));
   438:         tableSet.add(new TableInfo("TB_QST_QUESTIONARIO_PERGUNTA", "CO_QST_QUESTIONARIO_PERG
UNTA", 41L));
   439:         tableSet.add(new TableInfo("TB_QST_QUESTIONARIO_RESPONDIDO", "CO_SEQ_QST_QST_RESPOND
IDO", 0L));
   440:         tableSet.add(new TableInfo("TB_QST_RESPOSTA", "CO_SEQ_QST_RESPOSTA", 0L));
   441:         tableSet.add(new TableInfo("TB_QST_TIPO_QUESTIONARIO", "CO_QST_TIPO_QUESTIONARIO", 0L));
   442:         tableSet.add(new TableInfo("TB_QST_TIPO_RESPOSTA", "CO_QST_TIPO_RESPOSTA", 3L));
   443:         tableSet.add(new TableInfo("TB_RACA_COR", "CO_RACA_COR", 6L));
   444:         tableSet.add(new TableInfo("TB_RACIONALIDADE_SAUDE", "CO_RACIONALIDADE_SAUDE", 6L));
   445:         tableSet.add(new TableInfo("TB_RECEBIMENTO_ITEM", "CO_SEQ_RECEB_ITEM", 0L));
   446:         tableSet.add(new TableInfo("TB_RECEBIMENTO_LOTE", "CO_SEQ_RECEB_LOTE", 0L));
   447:         tableSet.add(new TableInfo("TB_RECEBIMENTO_VALIDACAO_ERROS", "CO_RECEB_ITEM", 0L));
   448:         tableSet.add(new TableInfo("TB_RECEITA", "CO_SEQ_RECEITA", 0L));
   449:         tableSet.add(new TableInfo("TB_RECEITA_MEDICAMENTO", "CO_SEQ_RECEITA_MEDICAMENTO", 0
L));
   450:         tableSet.add(new TableInfo("TB_RECEITA_TIPO_FREQUENCIA", "CO_RECEITA_TIPO_FREQUENCIA",
3L));
```

```
   451:         tableSet.add(new TableInfo("TB_RECURSO", "CO_SEQ_RECURSO", 0L));
   452:         tableSet.add(new TableInfo("TB_REGRA_CONDICIONADA", "CO_REGRA_CONDICIONADA", 10L));
   453:         tableSet.add(new TableInfo("TB_REGRA_VACINAL_DOSE", "CO_REGRA_VACINAL_DOSE", 437L));
   454:         tableSet.add(new TableInfo("TB_REGRA_VACINAL_ESTRATEGIA", "CO_REGRA_VACINAL_ESTRATE
GIA", 876L));
   455:         tableSet.add(new TableInfo("TB_RELATORIO_PROCESSAMENTO", "CO_SEQ_RELATORIO_PROCESSA
MENTO", 0L));
   456:         tableSet.add(new TableInfo("TB_REL_FICHAS_CONFIG", "CO_RELATORIO_FICHA", 0L));
   457:         tableSet.add(new TableInfo("TB_REL_OP_CIDADAO", "CO_SEQ_REL_OP_CIDADAO", 0L));
   458:         tableSet.add(new TableInfo("TB_REL_OP_CODIGO_UNICO_CIDADAO", "CO_SEQ_REL_OP_CODIGO_U
NICO", 0L));
   459:         tableSet.add(new TableInfo("TB_REL_OP_CRIANCA", "CO_SEQ_REL_OP_CRIANCA", 0L));
   460:         tableSet.add(new TableInfo("TB_REL_OP_FAMILIA", "CO_SEQ_REL_OP_FAMILIA", 0L));
   461:         tableSet.add(new TableInfo("TB_REL_OP_GESTANTE", "CO_SEQ_REL_OP_GESTANTE", 0L));
   462:         tableSet.add(new TableInfo("TB_REL_OP_IMOVEL", "CO_SEQ_REL_OP_IMOVEL", 0L));
   463:         tableSet.add(new TableInfo("TB_REL_OP_NUCLEO_FAMILIAR", "CO_SEQ_REL_OP_NUCLEO_FAMILI
AR", 0L));
   464:         tableSet.add(new TableInfo("TB_REL_OP_RISCO_CARDIO", "CO_SEQ_REL_OP_RISCO_CARDIO", 0L));
   465:         tableSet.add(new TableInfo("TB_REL_OP_SITUACAO_CLINICA", "CO_SITUACAO_CLINICA", 4L));
   466:         tableSet.add(new TableInfo("TB_RENDA_FAMILIAR", "CO_RENDA_FAMILIAR", 7L));
   467:         tableSet.add(new TableInfo("TB_REQUISICAO_EXAME", "CO_SEQ_REQUISICAO_EXAME", 0L));
   468:         tableSet.add(new TableInfo("TB_RES_ENVIO", "CO_SEQ_RES_ENVIO", 0L));
   469:         tableSet.add(new TableInfo("TB_RES_ENVIO_ERROS", "CO_RES_ENVIO", 0L));
   470:         tableSet.add(new TableInfo("TB_RES_ENVIO_ESTADO", "CO_RES_ENVIO_ESTADO", 4L));
   471:         tableSet.add(new TableInfo("TB_REVISAO", "CO_SEQ_REVISAO", 33490L));
   472:         tableSet.add(new TableInfo("TB_SESSAO_SINCRONIZACAO", "CO_UNICO_SESSAO", 0L));
   473:         tableSet.add(new TableInfo("TB_SEXO", "CO_SEXO", 4L));
   474:         tableSet.add(new TableInfo("TB_SEXTANTE", "CO_PARTE_BUCAL", 7L));
   475:         tableSet.add(new TableInfo("TB_SITUACAO_AGENDADO", "CO_SITUACAO_AGENDADO", 7L));
   476:         tableSet.add(new TableInfo("TB_SITUACAO_COROA", "CO_SITUACAO_COROA", 2L));
   477:         tableSet.add(new TableInfo("TB_SITUACAO_DADO_RECEBIDO", "CO_SITUACAO_DADO_RECEBIDO",
 7L));
   478:         tableSet.add(new TableInfo("TB_SITUACAO_FACE", "CO_SITUACAO_FACE", 22L));
   479:         tableSet.add(new TableInfo("TB_SITUACAO_LOCALIDADE", "CO_SITUACAO_LOCALIDADE", 3L));
   480:         tableSet.add(new TableInfo("TB_SITUACAO_LOTE_TRANSP_NODO", "CO_SITUACAO_LOTE_TRANSP
_NODO", 4L));
   481:         tableSet.add(new TableInfo("TB_SITUACAO_PROBLEMA", "CO_SITUACAO_PROBLEMA", 2L));
   482:         tableSet.add(new TableInfo("TB_SITUACAO_RAIZ", "CO_SITUACAO_RAIZ", 15L));
   483:         tableSet.add(new TableInfo("TB_STATUS_ATEND", "CO_STATUS_ATEND", 5L));
   484:         tableSet.add(new TableInfo("TB_STATUS_ATEND_PROF", "CO_STATUS_ATEND_PROF", 2L));
   485:         tableSet.add(new TableInfo("TB_TIPO_AGENDAMENTO", "CO_TIPO_AGENDAMENTO", 1L));
   486:         tableSet.add(new TableInfo("TB_TIPO_AGRAVO", "CO_TIPO_AGRAVO", 2L));
   487:         tableSet.add(new TableInfo("TB_TIPO_ATEND", "CO_TIPO_ATEND", 9L));
   488:         tableSet.add(new TableInfo("TB_TIPO_ATEND_PROF", "CO_TIPO_ATEND_PROF", 12L));
   489:         tableSet.add(new TableInfo("TB_TIPO_CIAP", "CO_TIPO_CIAP", 7L));
   490:         tableSet.add(new TableInfo("TB_TIPO_CONFIG_ATEND_DOMICILIR", "CO_TIPO_CONFIG_ATEND_DO
M", 1L));
   491:         tableSet.add(new TableInfo("TB_TIPO_CONSULTA_ODONTO", "CO_TIPO_CONSULTA_ODONTO", 99L)
);
   492:         tableSet.add(new TableInfo("TB_TIPO_CONTATO", "CO_TIPO_CONTATO", 6L));
   493:         tableSet.add(new TableInfo("TB_TIPO_DADO_TRANSP", "CO_TIPO_DADO_TRANSP", 14L));
   494:         tableSet.add(new TableInfo("TB_TIPO_EDEMA", "CO_TIPO_EDEMA", 4L));
   495:         tableSet.add(new TableInfo("TB_TIPO_ENCAM_INTERNO", "CO_TIPO_ENCAM_INTERNO", 3L));
   496:         tableSet.add(new TableInfo("TB_TIPO_ENCAM_ODONTO", "CO_TIPO_ENCAM_ODONTO", 99L));
   497:         tableSet.add(new TableInfo("TB_TIPO_EQUIPE", "CO_SEQ_TIPO_EQUIPE", 54L));
   498:         tableSet.add(new TableInfo("TB_TIPO_EXAME", "CO_TIPO_EXAME", 1L));
   499:         tableSet.add(new TableInfo("TB_TIPO_FILTRO_PROCED", "CO_TIPO_FILTRO_PROCED", 4L));
   500:         tableSet.add(new TableInfo("TB_TIPO_FORNEC_ODONTO", "CO_TIPO_FORNEC_ODONTO", 3L));
   501:         tableSet.add(new TableInfo("TB_TIPO_GLICEMIA", "CO_TIPO_GLICEMIA", 3L));
   502:         tableSet.add(new TableInfo("TB_TIPO_GRAVIDEZ", "CO_TIPO_GRAVIDEZ", 4L));
   503:         tableSet.add(new TableInfo("TB_TIPO_LOCALIDADE", "CO_TIPO_LOCALIDADE", 4L));
   504:         tableSet.add(new TableInfo("TB_TIPO_LOGRADOURO", "CO_TIPO_LOGRADOURO", 281L));
   505:         tableSet.add(new TableInfo("TB_TIPO_MEDICAO", "CO_SEQ_TIPO_MEDICAO", 13L));
   506:         tableSet.add(new TableInfo("TB_TIPO_OPCAO", "CO_TIPO_OPCAO", 4L));
   507:         tableSet.add(new TableInfo("TB_TIPO_ORIGEM_DADO_TRANSP", "CO_SEQ_TIPO_ORIGEM_DADO_TR
ANSP", 5L));
   508:         tableSet.add(new TableInfo("TB_TIPO_PARIDADE", "CO_TIPO_PARIDADE", 5L));
   509:         tableSet.add(new TableInfo("TB_TIPO_PARTE_BUCAL", "CO_TIPO_PARTE_BUCAL", 3L));
```

```
   510:         tableSet.add(new TableInfo("TB_TIPO_PERFIL", "CO_TIPO_PERFIL", 8L));
   511:         tableSet.add(new TableInfo("TB_TIPO_PERGUNTA", "CO_TIPO_PERGUNTA", 4L));
   512:         tableSet.add(new TableInfo("TB_TIPO_PESSOA_FISICA_IMAGEM", "CO_TIPO_PESSOA_FISICA_IMAG
EM", 0L));
   513:         tableSet.add(new TableInfo("TB_TIPO_RECEITA", "CO_TIPO_RECEITA", 4L));
   514:         tableSet.add(new TableInfo("TB_TIPO_REGISTRO", "CO_TIPO_REGISTRO", 9L));
   515:         tableSet.add(new TableInfo("TB_TIPO_SANGUINEO", "CO_TIPO_SANGUINEO", 7L));
   516:         tableSet.add(new TableInfo("TB_TIPO_SERVICO", "CO_TIPO_SERVICO", 10L));
   517:         tableSet.add(new TableInfo("TB_TIPO_UNIDADE_SAUDE", "CO_SEQ_TIPO_UNIDADE_SAUDE", 37L));
   518:         tableSet.add(new TableInfo("TB_TITULO_PATENTE", "CO_TITULO_PATENTE", 741L));
   519:         tableSet.add(new TableInfo("TB_UF", "CO_UF", 27L));
   520:         tableSet.add(new TableInfo("TB_UNIDADE_MEDIDA", "CO_UNIDADE_MEDIDA", 40L));
   521:         tableSet.add(new TableInfo("TB_UNIDADE_MEDIDA_TEMPO", "CO_UNIDADE_MEDIDA_TEMPO", 5L))
;
   522:         tableSet.add(new TableInfo("TB_UNIDADE_SAUDE", "CO_ATOR_PAPEL", 33346L));
   523:         tableSet.add(new TableInfo("TB_USUARIO", "CO_SEQ_USUARIO", 913L));
   524:         tableSet.add(new TableInfo("TB_VACINA", "CO_SEQ_VACINA", 0L));
   525:         tableSet.add(new TableInfo("TB_VIA_ADM_VACINA", "CO_VIA_ADM_VACINA", 6L));
   526:         tableSet.add(new TableInfo("TB_VISIBILIDADE_LEMBRETE", "CO_VISIBILIDADE_LEMBRETE", 1L));
   527:         return tableSet;
   528:     }
   529:
   530: }
```

```
 1: package br.ufsc.tcc.testproject.tests.estados.padrao;
 2:
 3: import br.ufsc.tcc.testproject.config.database.CdsTables;
 4: import br.ufsc.tcc.testproject.utils.database.truncate.TruncateAll;
 5:
 6: public class EstadoPadraoCDSOffline {
 7:
 8:  public void estadoPadrao() {
 9:         TruncateAll.doit(CdsTables.getTablesInfo());
10:  }
11:
12:  public static void main(String[] args) {
13:         TruncateAll.doit(CdsTables.getTablesInfo());
14:  }
15: }
```

```
 1: package br.ufsc.tcc.testproject.utils.database.truncate;
 2:
 3: import java.util.Set;
 4:
 5: import br.ufsc.tcc.testproject.config.database.CdsTables;
 6: import br.ufsc.tcc.testproject.config.database.CleanDatabaseTestService;
 7: import br.ufsc.tcc.testproject.config.database.generator.TableInfo;
 8:
 9: public class TruncateAll {
10:
11:  public static void doit(Set<TableInfo> tables) {
12:        long tempo0 = System.currentTimeMillis();
13:
14:        new CleanDatabaseTestService().deleteTablesH2(tables);
15:
16:        long tempo3 = System.currentTimeMillis();
17:        System.out.println("Tempo de truncateAll: " + (tempo3 - tempo0));
18:  }
19:
20:  public static void main(String[] args) {
21: //      TruncateAll.doit(CdsTablesSemImportarCnes.getTablesInfo());
22:        TruncateAll.doit(CdsTables.getTablesInfo());
23:  }
24:
25: }
```

```
 1: package br.ufsc.tcc.testproject.utils.database.jdbc;
 2:
 3: import java.sql.Connection;
 4: import java.sql.DriverManager;
 5: import java.sql.SQLException;
 6:
 7: import br.ufsc.tcc.testproject.config.database.DatabaseConfig;
 8: import br.ufsc.tcc.testproject.config.database.DatabaseValues;
 9:
10: public class ConnectionDB {
11:
12:  public static Connection getConnection() {
13:
14:         DatabaseConfig dbConfig = DatabaseValues.DB_H2;
15:         System.out.println(dbConfig.toString());
16:         try {
17:         return DriverManager.getConnection(dbConfig.getConnectionUrl(), dbConfig.getLogin(),
18:         dbConfig.getPassword());
19:         } catch (SQLException e) {
20:         System.err.println("Problemas com o BD" + e);
21:         }
22:
23:         throw new RuntimeException("nÃ£o foi possÃ-vel iniciar uma nova conexÃ£o");
24:   }
25:
26: }
```

```
 1: FROM openjdk:7u181-jdk-jessie
 2:
 3: RUN apt-get update \
 4:  && apt-get install -y \
 5:   xvfb \
 6:  && rm -rf /var/lib/apt/lists/* \
 7:  && rm -rf /src/*.deb
 8:
 9: ENV HOME /home/cdsoffline
10: ENV ESUS_INSTALL Instalador-eSUS-AB-CDS-3.0.13-Linux_17102018
11: ENV DISPLAY :1
12:
13: ADD ${ESUS_INSTALL}.zip ${HOME}/
14: ADD start.sh ${HOME}/
15:
16: # https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#user
17: RUN groupadd -r cdsoffline \
18:  && useradd --no-log-init -r -g cdsoffline cdsoffline \
19:  && mkdir ${HOME}/${ESUS_INSTALL} \
20:  && chown -R cdsoffline:cdsoffline /home/cdsoffline \
21:  && chmod +x ${HOME}/start.sh
22:
23: USER cdsoffline
24: WORKDIR ${HOME}
25:
26: VOLUME ["${HOME}/${ESUS_INSTALL}"]
27: # java -DjbossHome=./ -DLD_LIBRARY_PATH=./ -Xmx1024m -Xms1024m -XX:PermSize=256m -XX:MaxPermSi
ze=256m -jar cds-app.jar
28: CMD ["/bin/bash", "-c", "${HOME}/start.sh"]
29:
30: # path database ${HOME}/${ESUS_INSTALL}/bin/database/esus-h2
```

```
FROM openjdk:7u181-jdk-jessie

RUN apt-get update \
&& apt-get install -y \
  sudo \
  xvfb \
  procps \
  apt-transport-https \
  ca-certificates \
  --no-install-recommends \
&& curl -sSL https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add - \
&& echo "deb [arch=amd64] https://dl.google.com/linux/chrome/deb/ stable main" > /etc/apt/sources.list.d/google.list
\
&& apt-get update \
&& apt-get install -y \
 google-chrome-stable \
 --no-install-recommends \
&& rm -rf /var/lib/apt/lists/* \
&& rm -rf /src/*.deb

ENV DISPLAY :1

ADD br.ufsc.meutcc-0.0.1-SNAPSHOT.jar /
ADD start.sh /

CMD ["/bin/bash", "/start.sh"]
```

```
#!/bin/bash

if [[ -f /tmp/.X1-lock ]]; then
  rm -f /tmp/.X1-lock
fi

Xvfb :1 -screen 0 1360x1020x16 &

#shared folder
rm -rf ${ESUS_INSTALL}/*

unzip -oq ${ESUS_INSTALL}.zip -d ${ESUS_INSTALL}

cd ${ESUS_INSTALL}

./cds-app.sh
```

```
 1: #!/bin/bash
 2: echo "started chrome container"
 3: if [[ -f /tmp/.X1-lock ]]; then
 4:   rm -f /tmp/.X1-lock
 5: fi
 6:
 7: Xvfb :1 -screen 0 1360x1020x16 &
 8:
 9: URL_ESUS="http://cdsoffline:8090/esus"
10:
11: attempt=0
12:
13: # espera pelo carregamento completo do sistema
14: until [[ $(wget -S --spider $URL_ESUS  2>&1 | grep 'HTTP/1.* 200 OK') ]]; do
15:   printf "."
16:   sleep 1
17:     if [[ attempt=$(( $attempt + 1 )) -gt 300 ]] ; then
18:         exit 1
19:     fi
20: done
21:
22: java -cp br.ufsc.meutcc-0.0.1-SNAPSHOT.jar br.ufsc.tcc.rmi.ScriptClient
```

APÊNDICE B - Artigo

Paralelismo da Execução de Testes Automatizados

Rodrigo A. Costa

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil
rodrigo.aguiar@inf.ufsc.br

Abstract. *The automated tests are part of the main ways to carry out the tests of a software, in order to evaluate the quality. When the tests are executed on the graphical interface, these executions sometimes become time consuming, increasing the time between the beginning of tests execution and the results analysis. Parallelism of the tests execution is a way to reduce this response time. This research presents a study on the parallelism of the tests execution based on the structure of the automated tests in the Bridge laboratory. It will be demonstrated a possible alternative for the execution of automated tests in parallel, in order to bring greater agility and efficiency. With the use of virtual machines and containers it was possible to reduce the execution time of the tests by 66%.*

Resumo. *Os testes automatizados fazem parte de uma das principais formas para realização dos testes de um software, com objetivo de avaliar a qualidade deste. Quando os testes são executados junto à interface gráfica, estas execuções, por vezes, tornam-se demoradas, aumentando o tempo que se dará o início da análise dos resultados após a execução dos testes. Realizar paralelismo da execução dos testes é uma forma de reduzir este tempo de resposta, entre o início da execução e o início da análise dos resultados. Este trabalho apresenta um estudo sobre o paralelismo da execução de testes automatizados baseado na estrutura dos testes automatizados realizados no laboratório Bridge. Será demonstrada uma possível alternativa para a execução de testes automatizados em paralelo, a fim de trazer maior agilidade e eficiência. Com a utilização de máquinas virtuais e containers foi possível reduzir em 66% o tempo de execução dos testes.*

1. Introdução

A metodologia de pesquisa deste trabalho baseia-se na metodologia pesquisa-ação, do inglês *research action*. Medeiros (2009) define a pesquisa-ação como sendo uma metodologia com o objetivo de alinhar os conceitos educacionais junto a realidade operacional. Sendo realizado através de uma busca contínua pela melhoria, alinhado a um processo de estudo através das ações realizadas, ponderando constantemente estas ações. Também demonstrando uma análise das ações, mesmo que subjetivas, tomadas pelo pesquisador no contexto do cotidiano.

Sendo assim o estudo foi realizado com base nos testes de sistema da aplicação CDS Offline, desenvolvido pelo laboratório Bridge. Podendo ser adaptado para outros tipos de testes.

2. Descrição do problema

Durante o desenvolvimento de aplicações web também são desenvolvidos testes automatizados. Entre esses testes, existem os que se utilizam da automação de navegadores, como o Selenium WebDriver, que auxiliam na realização da manutenção e das validações da aplicação que está sendo desenvolvida.

Bruns (2009) afirma que a realização de testes que interagem com a interface web no navegador tendem a ser mais lentos que os testes realizados diretamente no código da aplicação, como no caso dos testes de unidade. Com o aumento do número de casos de testes automatizados, o tempo da execução aumenta consideravelmente, podendo ser proporcional à quantidade de passos e verificações que são realizadas na interface da aplicação. Uma forma de melhorar o tempo de execução, como Bruns (2009) também apresenta, seria reduzir os testes em pequenos módulos, onde estes poderiam ser executados de forma paralela em máquinas diferentes.

Para o projeto de testes, onde este estudo foi conduzido, utiliza-se casos de testes automatizados com base na aplicação "CDS Offline". À medida que o número de testes do sistema cresce, o tempo para sua execução também aumenta consideravelmente. Para melhorar esta situação, esta pesquisa procura encontrar uma melhor solução para execução dos testes de forma paralela, a fim de reduzir o tempo total da execução, com o menor consumo de recursos possíveis, fazendo assim, com que o processo da análise dos resultados se inicie o quanto antes.

O contexto deste trabalho trata do estudo e aplicação de métodos existentes para a paralelização da execução dos testes automatizados da aplicação CDS Offline, a fim de reduzir o tempo total da execução de todos os testes automatizados existentes, que são executados a cada versão gerada. Este trabalho envolve o setor de qualidade do laboratório Bridge, mais especificamente, a área de testes automatizados do laboratório.

Os testes automatizados são desenvolvidos de forma que suas execuções são realizadas em sequência. A execução dos testes depende da inicialização completa do sistema, sendo a aplicação disponibilizada no site do Portal do Departamento de Atenção Básica, onde, durante a realização deste trabalho, encontra-se na versão 3.0.13. Os testes são construídos a partir da documentação do sistema.

Como limitação, e melhor fidelidade aos testes implementados no laboratório, as execuções dos casos de testes foram realizadas em um ambiente onde não existiu mais de uma instalação da aplicação CDS Offline no sistema operacional.

Portanto, o tema de pesquisa é a implementação e avaliação da paralelização da execução de testes automatizados, com a utilização das ferramentas disponibilizadas no mercado.

3. Enfoque da ação

3.1. Objetivos

O objetivo geral deste trabalho é demonstrar uma forma de realizar o paralelismo da execução de testes automatizados, com a utilização de máquinas virtuais, *containers*, Selenium WebDriver, JUnit, entre outras ferramentas disponibilizadas no mercado.

Como objetivos específicos, são demonstrados alguns possíveis métodos para a realização do paralelismo de testes automatizados, como a criação de um cliente/servidor. Além de responder as questões de pesquisa detalhadas a seguir.

3.2. Questões de pesquisa

Para alcançar o objetivo proposto, deverão ser respondidas as seguintes questões:

- Questão 1: *A realização do paralelismo trouxe maior complexidade à inicialização da execução dos testes automatizados?* Esta questão tem por objetivo, responder se a realização de testes em paralelo não influencia a construção, não dificulta a inicialização da execução e posterior análise dos resultados dos testes executados.
- Questão 2: *Houve melhora na análise dos resultados dos testes automatizados?* Esta questão tem por objetivo responder se ocorreu uma melhora significativa na análise dos testes automatizados após sua execução, agilizando a posterior análise dos resultados obtidos para o reporte dos possíveis erros ocorridos na última versão disponibilizada do sistema.

Para realizar melhor a análise do resultado esperado, subdividiu-se as questões anteriores. Para a Questão 1 temos:

- Q1.1 *Quais modificações foram realizadas para criação dos testes automatizados individualmente?*
- Q1.2 *Houve significativa mudança na inicialização entre uma execução linear e uma execução paralela?*

Para responder estas questões se realizará a análise do código alterado e uma comparação entre a execução linear e paralela.

Para a Questão 2 temos:

- Q2.1 *Qual a diferença do tempo entre uma execução linear e uma execução paralela?*
- Q2.2 *Quais os recursos utilizados na execução paralela?*
- Q2.3 *Houve mudança na forma da análise dos resultados dos testes?*

Para responder estas questões, serão realizadas algumas execuções de forma linear e paralela, e serão capturados os dados relevantes.

4. Ações

4.1. Ações iniciais

As primeiras ações definiram o escopo do sistema no qual seriam realizados os testes automatizados. Optou-se por realizar testes no sistema de CDS Offline, considerando a simplicidade de instalação, inicialização do sistema e configuração inicial para realização dos testes.

Também, para efeitos de simplificação de configurações e estruturação do projeto, optou-se por definir alguns pré-requisitos para a realização dos testes, como a importação de entidades pré-definidas para a realização dos testes, assim como a

alteração no arquivo de configuração do JBoss, permitindo mais de uma conexão no banco de dados, já que a utilização padrão do sistema somente uma conexão é permitida.

Como base para criação dos testes automatizados, foi utilizado uma simplificação do *framework* aplicada no laboratório Bridge. As classes de teste estendem uma classe abstrata, chamada CasoDeTeste, onde obrigatoriamente devem implementar o método *asserts*. Este método deve possuir somente verificações, ou ações que não alterem o estado atual do sistema. Os métodos *passos* e *passosAposAsserts* são opcionais, e possuem a finalidade de realizar ações no sistema. O método *passos* é executado antes do método *asserts*, para definir onde serão realizadas as verificações. O método *passosAposAsserts* é executado depois do método *asserts*, realizando ações posteriores.

Os métodos *passos* e *passosAposAsserts* de um teste podem ser reutilizados como um fluxo de execução por outros testes. Assim, é possível reaproveitar as ações realizadas pelos testes com menor replicação de código.

4.2. Cliente/Servidor

Foram utilizados aproximadamente 170 casos de testes para dar início à pesquisa. O modelo cliente/servidor mostrou-se mais apropriado ao objeto inicial, onde o servidor possui todos os testes e os distribui para os clientes realizarem requisições de execução. E ao final da execução, os clientes devolvem ao servidor as informações de execução do teste.

A solução encontrada para realizar esta tarefa, com a menor interferência no *framework* já estabelecido no laboratório, foi a ferramenta Java RMI, onde um objeto de uma máquina virtual Java específica possui a permissão de invocar métodos de um objeto que esteja rodando em outra máquina virtual, como especificado na documentação disponibilizada pela Oracle.

Os serviços oferecidos pelo servidor são criados a partir de uma interface. Esta, por sua vez, deve estender a interface *Remote* do Java RMI. Como exemplificado no trecho de código demonstrado na Figura 1.

```
public interface RMIServices extends Remote {
   TestExecInfo getTesteParaExecucao() throws RemoteException;
   void setResultadoDaExecucao(TestExecInfo testResult) throws RemoteException;
}
```

Figura 1. Interface implementada pelo servidor

Com a implementação desta interface, o servidor é criado. Ao instanciá-lo é definido o *hostname* e é criado o registro no *localhost*, definindo a porta por onde serão aceitas as requisições do cliente. Antes de iniciar o servidor, é preciso adicionar as suítes de testes que se deseja executar. Os testes inseridos ficam armazenados em um *buffer* no servidor, onde após a chamada do método *startServer*, o mesmo fica no aguardo à espera de uma requisição de um cliente.

Para a criação do cliente, é necessário passar como parâmetro o *host* do servidor para realizar o registro adequadamente. Através do *LocateRegistry*, é possível obter a referência do registro passando *host* e a porta, no caso é utilizada a porta padrão. Assim, o cliente pode obter a referência do servidor com os serviços disponíveis.

Ficou definido que, enquanto existirem testes a serem executados, o cliente ficará requisitando testes ao servidor. Ou seja, enquanto o servidor retornar testes ao cliente, este permanecerá executando, como demonstrado na Figura 2.

```
public void start() {
	while (true) {
		TestExecInfo rmiRequest = this.getTestExecInfo();
		if (rmiRequest == null) {
			break;
		}
		this.executeSequence(rmiRequest);
	}
}
```

Figura 2. Inicialização do cliente

Assim o método *getTestExecInfo* no cliente realiza uma chamada no servidor através do serviço *getTesteParaExecucao*. A implementação deste método no servidor, como mostrado na Figura 3, possuí um bloco de sincronização para controlar a concorrência caso exista mais de uma chamada simultaneamente, retornando o teste ao cliente. Caso o *buffer* contendo os testes esteja vazio o servidor é encerrado.

```
@Override
public TestExecInfo getTesteParaExecucao() throws RemoteException {
	synchronized (this) {
		if (!this.buffer.isEmpty()) {
			System.out.println("BUFFER total = " + this.buffer.size());
			return this.buffer.removeFirst();
		}
	}
	this.endServer();
	return null;
}
```

Figura 3. Serviço implementado pelo servidor

Ao final da execução do teste, como mostrado na Figura 4, o cliente enviará o resultado da execução ao servidor em uma nova *thread*, não precisando esperar o envio dos resultados para requisitar novos testes ao servidor.

```
private void executeSequence(TestExecInfo rmiRequest) {

	if (rmiRequest.getTestCase() != null) {
		Result result = new ExecRequest(rmiRequest.getTestCase()).exec();
		rmiRequest.setResult(result);
		this.sendResult(rmiRequest);
	}
}

private void sendResult(final TestExecInfo testExecInfo) {
```

```
        this.storeData(testExecInfo);
        final RMIServices stubAux = this.stub;
        // criado nova thread para thread principal não esperar o envio e continuar
        new Thread(new Runnable() {
                @Override
                public void run() {
                        try {
                                stubAux.setResultadoDaExecucao(testExecInfo);
                        } catch (RemoteException e) {
                                this.run();
                                e.printStackTrace();
                        }
                }
        }, testExecInfo.getTestCase().getSimpleName()).start();
}
```

Figura 4. Execução e envio dos testes pelo cliente

4.3. Containers

Neste ponto, o servidor e o cliente estão implementados. Iniciou-se o estudo da execução da aplicação CDS Offline, tal como, da execução dos testes em *containers* docker. Visto que, se as aplicações rodam em *containers*, elas podem rodar em qualquer sistema que tenha suporte ao docker, por exemplo. Suportando, evidentemente, as configurações mínimas de hardware da aplicação. Deste modo, o custo para criação de máquinas seria reduzido.

A partir deste ponto, a imagem é criada contendo a aplicação CDS Offline. Optou-se por realizar a instalação em um *container* docker a partir de uma máquina local. Após a imagem da instalação da aplicação ser criada, as demais máquinas utilizarão a mesma imagem. A aplicação CDS Offline tem como um dos pré-requisitos a utilização de Java 7. Então, foi decidido utilizar como base para utilização do sistema, a imagem *openjdk:7u181-jdk-jessie*, disponibilizada no site Docker Hub.

Para criação da imagem é utilizado o arquivo Dockerfile, onde é possível construir imagens automaticamente com o uso de instruções.

```
FROM openjdk:7u181-jdk-jessie

RUN apt-get update \
 && apt-get install -y xvfb \
 && rm -rf /var/lib/apt/lists/* \
 && rm -rf /src/*.deb

ENV HOME /home/cdsoffline
ENV ESUS_INSTALL Instalador-eSUS-AB-CDS-3.0.13-Linux_17102018
ENV DISPLAY :1

ADD ${ESUS_INSTALL}.zip ${HOME}/
ADD start.sh ${HOME}/

RUN groupadd -r cdsoffline \
 && useradd --no-log-init -r -g cdsoffline cdsoffline \
 && mkdir ${HOME}/${ESUS_INSTALL} \
 && chown -R cdsoffline:cdsoffline /home/cdsoffline \
 && chmod +x ${HOME}/start.sh

USER cdsoffline
WORKDIR ${HOME}

VOLUME ["${HOME}/${ESUS_INSTALL}"]

CMD ["/bin/bash", "-c", "${HOME}/start.sh"]
```

Figura 5. Dockerfile da aplicação

Como demonstrado na Figura 5, na criação desta imagem será instalado o programa Xvfb. Com ele é possível iniciar uma tela virtual, para poder iniciar a aplicação CDS Offline. Também são adicionadas variáveis de ambientes, um arquivo compactado contendo a aplicação, um script bash que rodará ao iniciar o *container*, além da definição de um usuário, pois, a aplicação não deve ser rodada como superusuário.

Para realizar a execução dos testes, foi definido a criação de uma outra imagem, demonstrado na Figura 6, contendo uma instalação do navegador Chrome e uma aplicação jar, que executará os testes, bem como, um *script bash*, que será executado quando o *container* iniciar.

```
FROM openjdk:7u181-jdk-jessie

RUN apt-get update \
&& apt-get install -y \
  xvfb \
  apt-transport-https \
  ca-certificates \
  --no-install-recommends \
&& curl -sSL https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add - \
&& echo "deb [arch=amd64] https://dl.google.com/linux/chrome/deb/ stable main" > /etc/apt/
sources.list.d/google.list \
&& apt-get update \
&& apt-get install -y \
```

```
 google-chrome-stable \
 --no-install-recommends \
&& rm -rf /var/lib/apt/lists/* \
&& rm -rf /src/*.deb

ENV DISPLAY :1

ADD br.ufsc.meutcc-0.0.1-SNAPSHOT.jar /
ADD start.sh /

CMD ["/bin/bash", "/start.sh"]
```

Figura 6. Dockerfile do cliente

Para realizar a execução em dois *containers* distintos, foi necessária a realização de mais uma alteração no arquivo *standalone.xml* do JBoss, que não permitia o acesso remoto à aplicação. E, também, foi adicionado um novo arquivo, *test.properties*, no *resource* do projeto de testes, para definir a localização do banco de dados e ip do servidor de teste.

Para facilitar o início das imagens criadas da aplicação e execução dos testes, foi criado o arquivo *docker-compose.yml*, demonstrado na Figura 7. Para a utilização deste arquivo, também é necessário a instalação do *docker-compose*.

```
version: "2"

services:
  cdsoffline:
    image: cdsoffline
    volumes:
      - /home/cdsoffline/Instalador-eSUS-AB-CDS-3.0.13-Linux_17102018

  testes:
    image: chrome
    depends_on:
      - cdsoffline
    volumes_from:
      - cdsoffline
```

Figura 7. Arquivo docker-compose

Outro motivo levado em consideração para a utilização do docker-compose foi a simplicidade da criação de um determinado volume de um *container* em outro. Assim, o acesso ao banco de dados foi contornado facilmente.

4.4. Execuções

Definido a criação dos *containers*, iniciou-se os testes em máquinas virtuais. Foram disponibilizadas um total de quatro máquinas de mesma configuração.

Após criadas, as imagens foram compartilhadas entre as máquinas que executarão os testes. Com isso, ao iniciar a execução destes *containers*, a aplicação CDS Offline será inicializada e, em seguida, a execução dos testes. Isso agilizou o tempo de instalação

inicial, desviando da possibilidade de ter qualquer diferença entre as instalações, sendo que todos os *containers* seguirão exatamente os mesmos passos iniciais.

Assim, a adição de mais máquinas se torna simples, pois, não é necessário ter muitos recursos, como recursos visuais para sua execução, porque todas as configurações necessárias estarão presentes dentro do *container*.

Por fim, realizou-se a execução dos testes nas máquinas virtuais disponíveis. A fim de melhor avaliar os resultados obtidos, foram realizados diferentes experimentos tais que, em cada um deles, um número diferente de máquinas foi utilizado para a execução paralela dos testes automatizados. Foram realizados testes em uma, duas e quatro máquinas.

Em seguida, foi aplicado o mesmo conceito no ambiente de produção. Foram executados 1326 testes da aplicação CDS Offline, coletando os dados para posterior análise.

5. Avaliação e Análise

5.1. Ambiente controlado

Inicialmente, como ponto de partida para posterior análise dos resultados obtidos, foi executado, através da IDE eclipse, uma suíte contendo 173 casos de testes. O tempo de execução dos testes foi de 24 minutos e 26 segundos, demonstrado na Figura 8.

Como um dos questionamentos realizados no início deste trabalho Q1.1, não houve nenhuma modificação realizada na forma de criação dos testes automatizados. Como pode ser visualizado no Capítulo das Ações, o foco foi diretamente no modo de execução dos testes, não existindo a necessidade de realizar mudanças na forma que são codificados os testes automatizados.

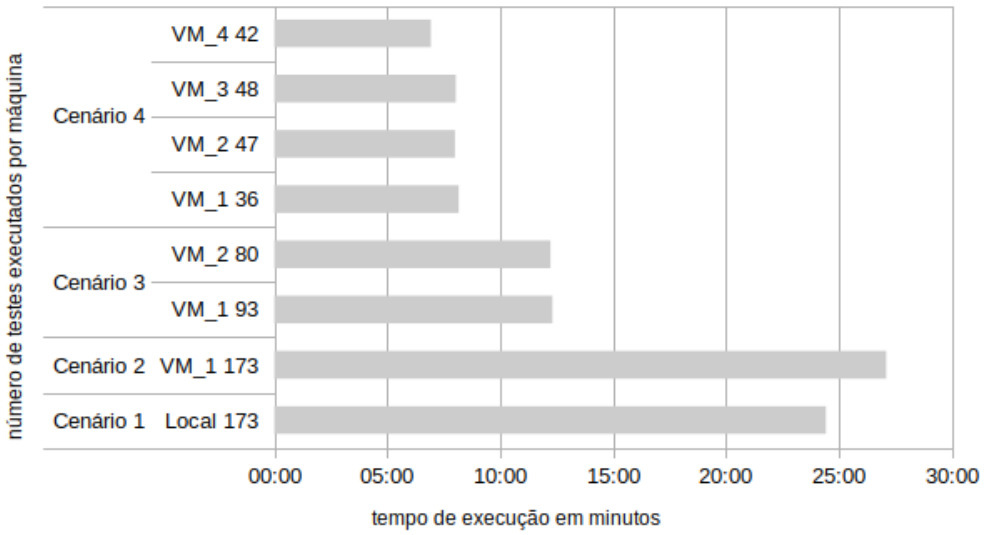

Figura 8. Execução em quatro máquinas

Entretanto, para responder à questão Q1.2, é necessário ressaltar que há uma maior complexidade na forma de executar os testes em paralelo. Visto que, existe no mínimo duas classes a serem executadas, um servidor e, pelo menos, um cliente. Porém, esta

abordagem traz benefícios, como a facilidade de automatizar o processo de início da execução dos testes automatizados através de ferramentas existentes no mercado, como o Jenkins.

No experimento em que paralisamos a execução temos o servidor. Nele é realizado a separação dos testes que serão enviados aos clientes, quando os clientes solicitarem testes para execução.

Contudo, a inicialização de novas máquinas se mostra facilitada, visto que consiste apenas na necessidade de inicializar dois *containers*. Caso se faça necessário, o aumento de máquinas para execução em paralelo torna-se mais fácil.

Para efeitos de comparação e análise, foi realizado uma execução cliente/servidor com apenas um cliente, representado pelo Cenário 2.

O tempo total de execução foi de 27 minutos e 6 segundos, mostrando-se um pouco acima da execução local. Porém, esta análise é apenas para efeito de comparação, pois esta abordagem com apenas um cliente não é benéfico neste contexto.

O Cenário 3 mostra a execução em duas máquinas. É possível observar que o número de testes executados por máquina difere neste cenário. Sendo a máquina VM_1 executando 13 testes a mais que a VM_2. Isso se reflete, principalmente, devido ao tempo de execução de cada teste. Porém, o tempo de resposta da rede também é um fator que se pode levar em consideração. O tempo de execução da máquina VM_1 foi de 12 minutos e 19 segundos, enquanto da máquina VM_2 foi de 12 minutos e 14 segundos. Com isso, o tempo total de execução foi igual ao da máquina VM_1, diminuindo aproximadamente 50% quando comparado com o valor da execução local, representado pelo Cenário 1.

Este resultado era esperado para o tempo de execução. Vale ressaltar que, o resultado mostra apenas o tempo de execução, e não leva em consideração o tempo de configuração inicial e o de inicialização dos *containers*.

Para comparação posterior, foram realizados testes em quatro máquinas, representado pelo Cenário 4. Apesar do número de máquinas ter dobrado, em relação ao Cenário 3, o tempo não caiu pela metade, ficando aproximadamente 34% mais rápido. A máquina VM_1 levou 8 minutos e 10 segundos, a VM_2 levou 8 minutos, a VM_3 levou 8 minutos e 2 segundos e a VM_4 levou 6 minutos e 56 segundos.

É possível perceber também neste cenário, um desbalanceamento de testes executados por diferentes máquinas. Porém, apesar da máquina VM_1 executar menos testes que a VM_4, essa última levou menos tempo para finalizar todas as requisições.

Como foi possível perceber durante os testes, o tempo de execução foi diminuindo com o aumento do número de máquinas, no entanto, não foi uma redução proporcionalmente compatível. Somente aumentar o número de máquinas acaba gerando um aumento significativo da utilização de recursos para execução dos testes em paralelo.

Respondendo à questão Q2.3, a análise dos resultados teve uma melhora no tempo de resposta quando comparado com a execução local em uma IDE. O resultado da execução dos testes retorna ao servidor e é impresso na tela, o que não é uma forma amigável de se analisar os resultados quando há um grande número de testes falhando.

Um modo interessante seria tratar esse retorno de forma que facilite a análise dos resultados de maneira mais amigável, porém, isto foge do escopo deste trabalho.

5.2. Ambiente produção

Aplicando o conceito abordado anteriormente no contexto de execução real dos testes de CDS Offline, foi realizado testes no módulo das fichas, com 1326 testes. Estes testes foram localmente executados na IDE Eclipse e levou 3 horas e 3 minutos, representado na Figura 9.

A execução em modo cliente/servidor, com apenas um cliente, levou 2 horas e 57 minutos. Um tempo bem próximo da execução local. A diferença de tempo mostra-se irrelevante, considerando o número elevado de testes. Como estes são executados no navegador, pode existir diferentes tempos de resposta para um mesmo teste quando ele é executado diversas vezes.

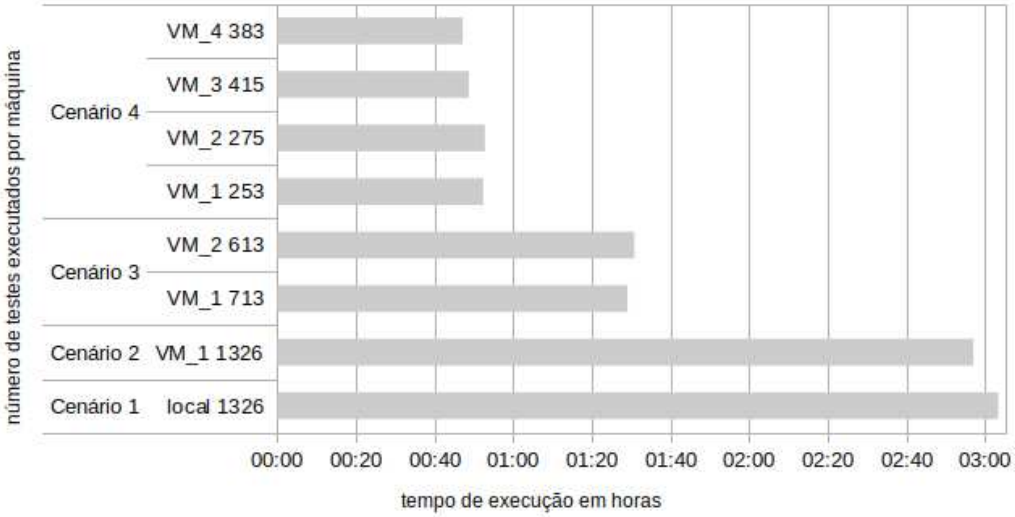

Figura 9. Execução em quatro máquinas

Como é mostrado no Cenário 3, os testes foram executados em duas máquinas virtuais. Sendo que, 613 testes foram executados na máquina VM_1, levando o tempo de 1 hora e 30 minutos, e 713 testes executados na máquina VM_2, com tempo de 1 hora e 29 minutos. Assim, o tempo que levou a execução em ambiente de produção se mostrou proporcional ao ambiente controlado, levando praticamente a metade do tempo de execução com duas máquinas.

O Cenário 4, que conta com quatro máquinas virtuais, o tempo foi de aproximadamente 52 minutos para 253 testes na máquina VM_1, 52 minutos para 275 testes na máquina VM_2, 48 minutos para 415 testes na máquina VM_3 e 47 minutos para 383 testes na máquina VM_4.

Assim, como no ambiente controlado, o tempo total de execução no Cenário 4 não caiu pela metade como ocorreu no Cenário 2, apesar de ficar mais próximo. O número de testes executados mostra certo desequilíbrio de execuções por máquina. Apesar da máquina VM_3 ter executado aproximadamente 40% a mais de testes que a máquina VM_1, ela levou 5 minutos a menos para concluir sua execução.

6. Conclusões

O aprendizado de novas tecnologias e a interação delas entre si foi um dos aspectos mais interessante deste trabalho. Pois a integração de aplicações diferentes para chegar a um objetivo é um desafio interessante. Neste caso o desafio de melhoria do tempo da execução de testes automatizados.

A implementação de um cliente/servidor com Java RMI alinhou-se adequadamente aos testes que já eram realizados com a linguagem Java, adquirindo um melhor conhecimento do funcionamento da ferramenta.

A utilização de *containers* mostrou-se um facilitador para escalar o número de máquinas e realizar a execução dos testes. Principalmente com a adição de mais de uma máquina virtual, evitando um esforço repetitivo de ambientação do sistema.

A premissa inicial de delimitar apenas uma instalação da aplicação e uma execução de testes por sistema operacional, devido a características dos testes realizados no laboratório, deverá ser revista. O escalonamento com *containers* permite diversas execuções em um mesmo sistema operacional, sem que exista interferência entre eles. Alguns testes realizam chamadas de sistema, o que pode impactar diretamente nos *containers* em execução. Tais testes deverão ser tratados de forma diferenciada.

Em se tratando de pesquisa-ação, é possível analisar, com maior realidade, os impactos das escolhas realizadas. No decorrer do desenvolvimento, mais precisamente nas ações tomadas, é que se percebe o quanto decisões podem mudar o rumo da pesquisa. No transcorrer das atividades, teve-se a necessidade de adicionar novas ferramentas e realizar algumas alterações na aplicação que não havia sido previamente levado em consideração.

O objetivo principal foi alcançado. O tempo total da execução dos testes foi reduzido. Através do uso de máquinas virtuais para paralelizar a execução de testes automatizados, com quatro delas se obteve uma redução de até 66% no tempo de execução dos testes feitos para a aplicação CDS Offline. Esse ganho está relacionado diretamente com a segunda questão levantada, trazendo agilidade no tempo de resposta sobre a qualidade do produto.

A ação realizada em um ambiente real de produção, mostrou-se ainda mais eficiente dada as características intrínsecas do sistema de testes. Obteve-se um ganho de mais de 70% no tempo de execução. Dando maior agilidade de realização na análise dos resultados dos testes automatizados. A segurança por parte dos desenvolvedores mostrou-se maior, indicando também uma maior confiança no resultado final da aplicação a cada versão gerada.

A execução paralela dos testes trouxe uma maior complexidade no início da execução dos testes, como a Questão 1 abordou. Contudo, trouxe o benefício de se reduzir o tempo de execução. Por fim, o modelo de paralelismo apresentado neste trabalho contribuiu no modo como as execuções dos testes são realizadas no Laboratório Bridge. A cada versão que é lançada, a execução de todos os testes é automaticamente iniciada e distribuída entre as diversas máquinas virtuais que estão disponíveis para esse propósito.

7. Referências

Santos, Paulo Sérgio Medeiros. “Uma Análise da Utilização da Metodologia da Pesquisa-Ação em Engenharia de Software”. 176 f.. Dissertação (Mestrado). COPPE/UFRJ. Rio de Janeiro, 2009.

Bruns, Andreas; Kornstädt, Andreas; Wichmann, Dennis. “Web Application Tests with Selenium”. 2009, IEEE Computer Society.

https://docs.oracle.com/javase/tutorial/rmi/ Acessado em 14/06/2017.

https://docs.oracle.com/javase/7/docs/api/java/rmi/registry/LocateRegistry.html Acessado em 23/04/2019.

https://hub.docker.com/_/openjdk Acessado em 20/04/2019.

https://docs.docker.com/engine/reference/builder/ Acessado em 23/04/2019.