Ministério da Justiça

Projeto: SISCONARE

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	31/01/2020
1.1	Recomendações de segurança	Israel Branco	04/02/2020
1.2	Add referências para relatórios de testes	Israel Branco	18/02/2020

1 Sumário

2 Introdução

SISCONARE foi desenvolvido para automatizar o processo de entrada de refugiados no Brasil, sua utilização é feita tanto pelos interessados na solicitação quanto quanto agentes públicos. Este sistema esta classificado como crítico dado a sua notória relevância negocial.

O sistema possui funcionalidades com acessos públicos (não requer autenticação do agente) tais como cadastro de refugiados e verificação de documentos, possui também funcionalidades que necessitam de autenticação para a realização das solicitações de refúgios e a realização de plenárias. Estas funcionalidades compõe a base negocial da aplicação.

Motivado pelo baixo desempenho apresentado em ambiente produtivo, esta análise visa abordar os principais motivadores para o não atendimento dos requisitos não funcionais que permeiam a qualidade da aplicação, sendo eles: usabilidade, desempenho, confiabilidade, segurança, disponibilidade, manutenção de código, utilização de padrões para codificação e tecnologias envolvidas.

3 Apresentação do cenário atual

Esta sessão ira descrever a arquitetura, tecnologias, frameworks e dependências que compõe a base da aplicação.

O SISCONARE está construído para funcionar em ambiente WEB, utiliza tecnologia Java e está estruturada arquiteturalmente como uma aplicação monolítica (entende-se por este termo quando o sistema é composto por camadas de interface com usuário, camada de aplicação de regras negociais e camada de acesso a dados combinadas em uma única aplicação), utiliza banco de dados relacional *PostgreSQL* e se integra a ferramenta gestão de documentos/processos SEI – Sistema Eletrônico de Documentos.

O core da solução utiliza dois componentes, sendo eles: conare-entity e conare-web. O componente conare-entity é responsável por representar as entidades de negócio (*Value Objects - VO*) e faz o mapeamento Objeto/Relacional com o banco de dados. O componente conare-web é responsável por conter a representação da camada de acesso a dados (*Data Access Object – DAO*), camada de serviço responsável por operacionalizar as regras negociais (*Service*), camada de visão (iteração com usuário - *View)* e demais funcionalidades de aspecto de utilização geral do sistema (segurança, funcionalidades reutilizáveis, exceções , relatórios e etc).

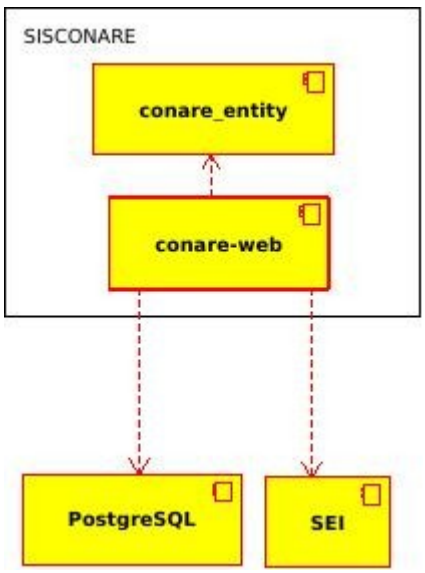

Figura 1: SISCONARE - Componentes

No ponto de vista de desenvolvimento de software nos aspectos corretivos e evolutivos, esta estrutura traz clareza ao implementador por utilizar padrões largamente utilizados no mercado.

Há boa estruturação dos pacotes é dada pela segregação lógica na organização dos mesmos.

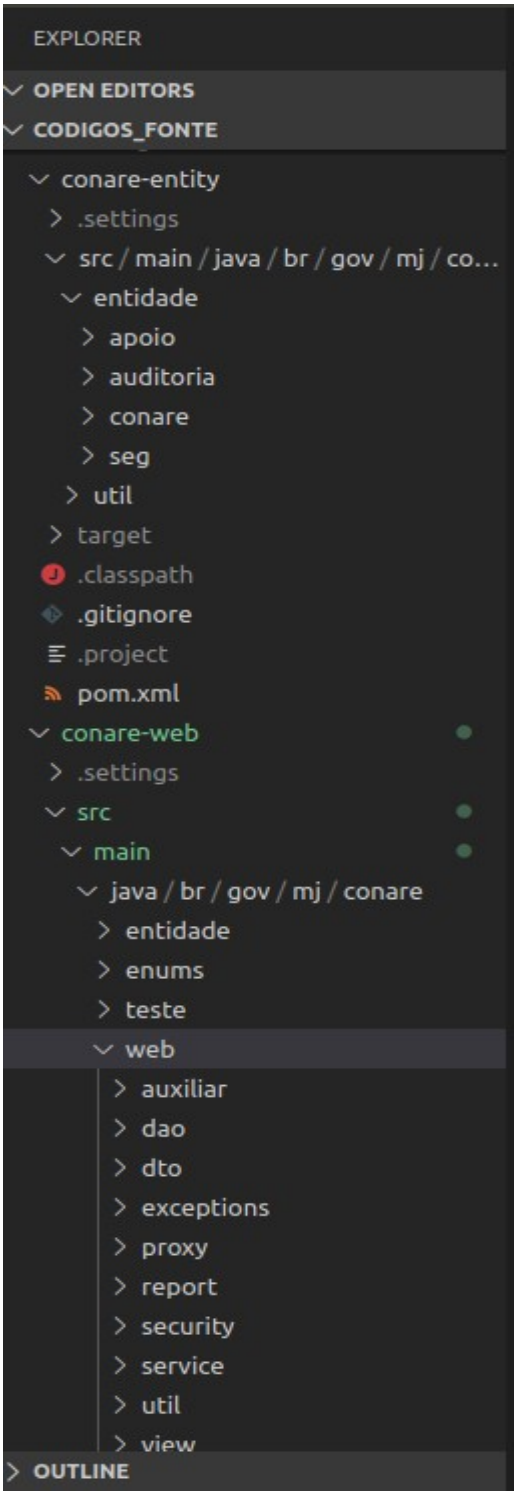

Figura 2: Estruturação hierárquica do projeto

3.2 Tecnologias utilizadas

Esta sessão descreve as tecnologias, frameworks e bibliotecas utilizadas na construção do projeto, descrevendo versões e propósitos de utilização.

Nome	Versão	Utilização	Observação
Java	1.8	Linguagem de programação.	
Hibernate	4.3.7	Framework ORM.	
Hibernate-Envers	4.3.7	Módulo Hibernate utilizado para auditoria na manipulação de dados.	
Wildfly	8.x	Servidor de aplicação JEE.	Utiliza containers EJB, CDI e Servlet.
Apache Wicket	6.19.0	Framework orientado a componentes utilizado no desenvolvimento de aplicações Java Web.	
Junit	4.11	Utilizado para a criação de testes automatizados.	
Jboss Arquillian	1.1.7/1.1.8	Framework para realização testes de integração.	Divergência de versões entre conare-web e parent
JasperReports	5.6.1	Componente utilizado para criação de relatórios.	
Apache POI	3.12	Componente utilizado para manipular documentos baseado em formato MS Office.	
Mockito	1.10.19	Componente utilizado para viabilizar a utilização mocks em testes unitários.	
Cucumber	4.2.0	Ferramenta para executar automação nos testes de aceitação.	Utiliza BDD – Behevior-Driven Development (Desenvolvimento orientado a

			comportamento) como base.
Selenium	3.9.1	Ferramenta para automação de testes funcionais.	
SEI-Client	0.3.7	Componente para integração com SEI	Componente desenvolvido pelo MJ.
PostgreSQL		Banco de dados relacional	

3.2 Implantação da ferramenta

Dado a utilização da arquitetura Java Enterprise Edition este projeto utiliza o servidor de aplicação Wildfly na versão 8.x. A disposição de sua utilização em ambiente produtivo está disposta conforme a ilustração a seguir.

Figura 3: Implantação SISCONARE

4 Análise técnica

Este tópico descreve a ferramenta do ponto de vista técnico, tanto nos aspectos de codificação quando de consumo de recursos utilizados pela ferramenta.

4.1 SonarQube

Ferramenta utilizada para verificação de qualidade de código. Para esta análise foram utilizadas as métricas de qualidade implantadas no SonarQube do Ministério da Justiça e os resultados foram os seguintes:

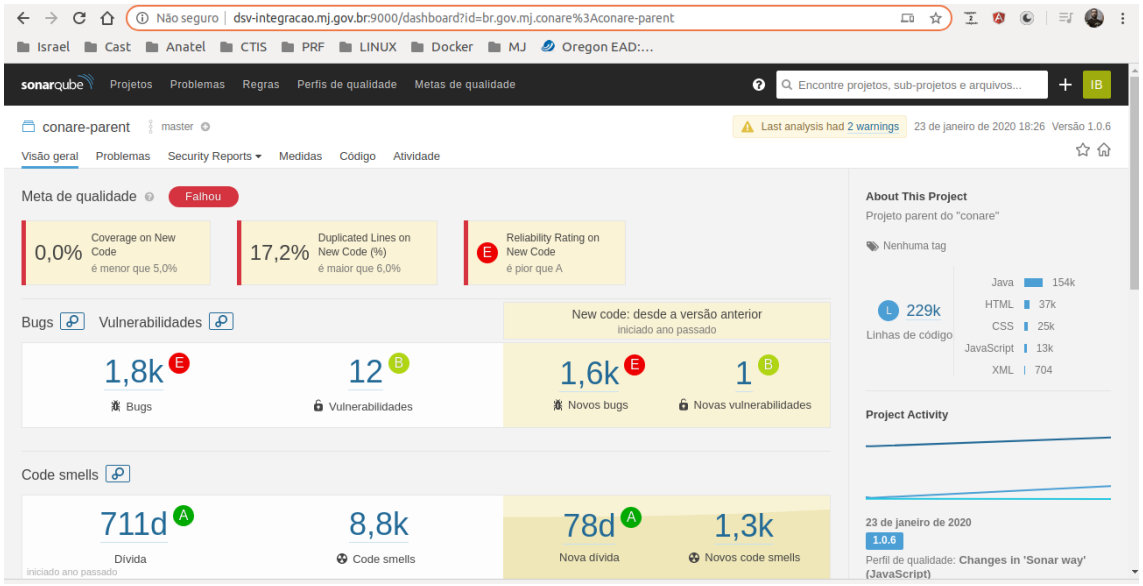

Figura 4: Análise de código

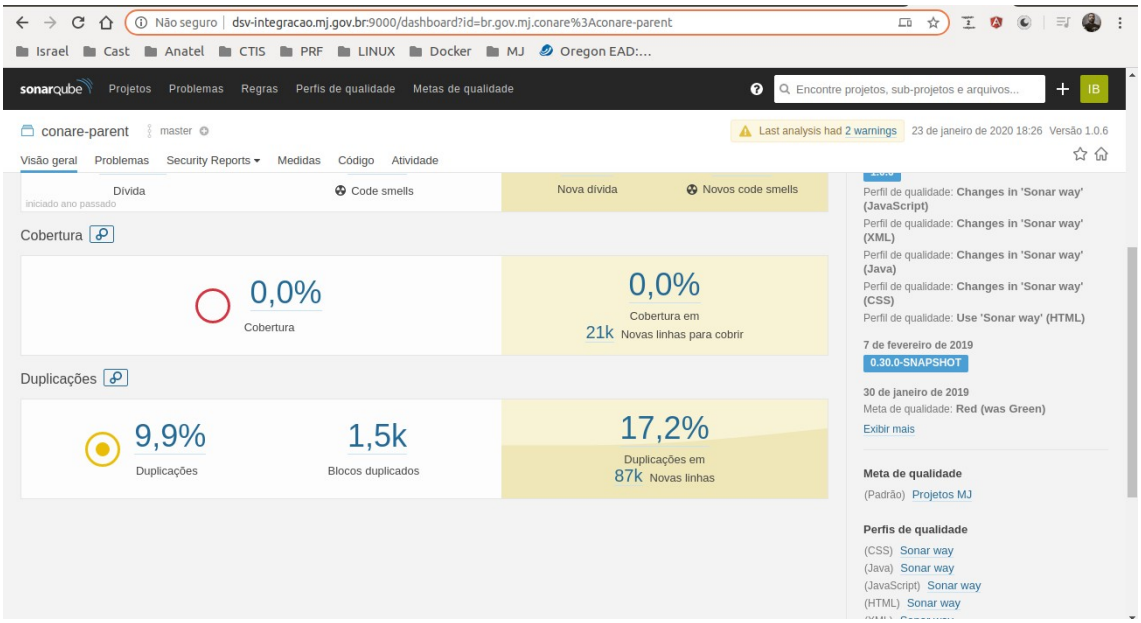

Figura 5: Análise de qualidade

Para a obtenção dos resultados, fora utilizado o código fonte do ambiente de produção (master branch), sendo que os resultados apresentados nesta análise demonstram a baixa qualidade no código produzido para este sismtema: 18 mil ocorrências de bugs, 12 ocorrências de vulnerabilidades, 711 ocorrências de débito técnico, 8.8 milhões de ocorrências de de código ruim, 0% de cobertura de testes e 17.2% no total de duplicação de código.

4.2 Kibana APM – App Monitor Tool

Kibana é um plugin de visualização de dados da ferramenta ElasticSearch, está ferramenta tem como propósito expor aspectos relativos a utilização de recursos de máquina (CPU e memória), utilização da ferramenta por quantitativos de requisições, duração de transações dentre outras características que demonstram o comportamento da solução no cenário a qual está submetida. Vale ressaltar que os resultados obtidos nesta análise se referem ao ambiente de produção do SISCONARE.

Os resultados apresentados refletem a utilização de ambos os nós do cluster. Esta análise leva em consideração a utilização da ferramenta em 1 semana e outra análise com duração referente ao pico de utilização da ferramenta em 1 dia.

4.2.1 Kibana: analise de 1 semana

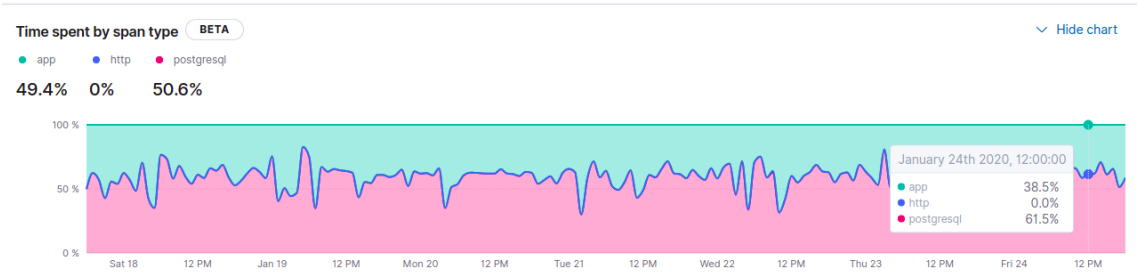

Figura 6: Tempo gasto por aplicação

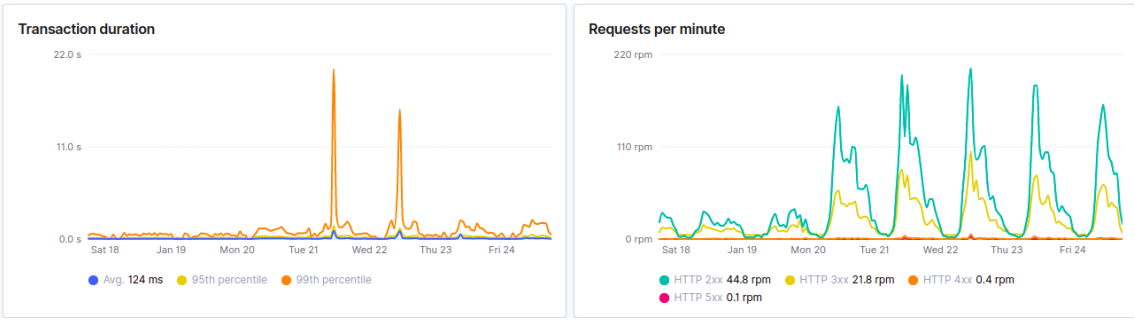

Figura 7: Duração de transação e quantidade de requisições por minuto

Figura 8: Ocorrência de erros

Figura 9: Classificação de erros

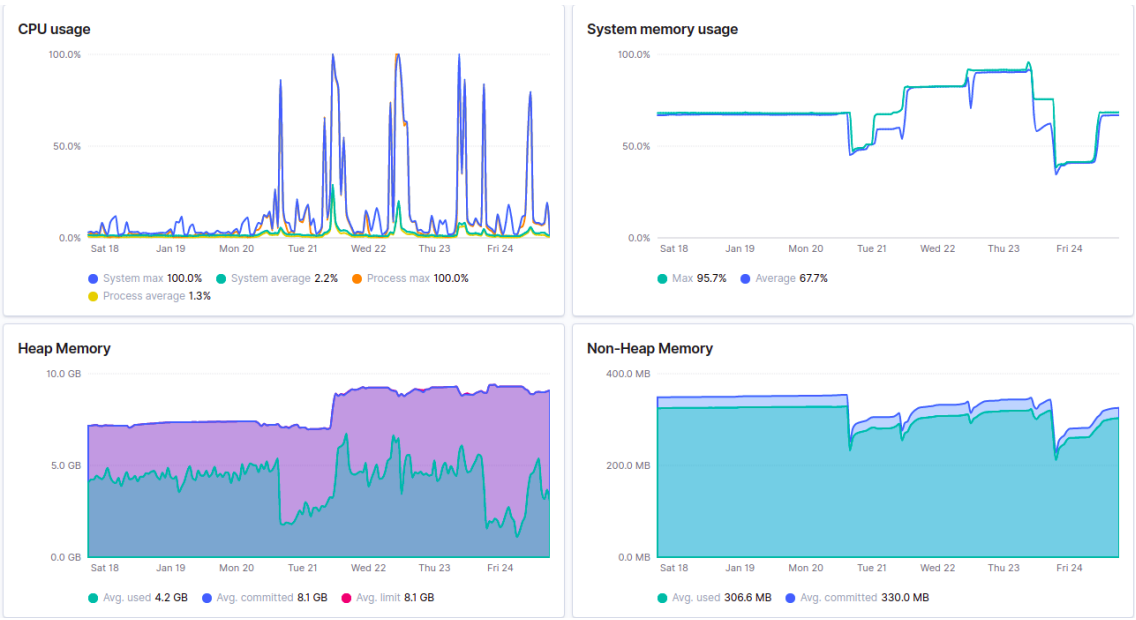

Figura 10: Utilização de CPU e memória

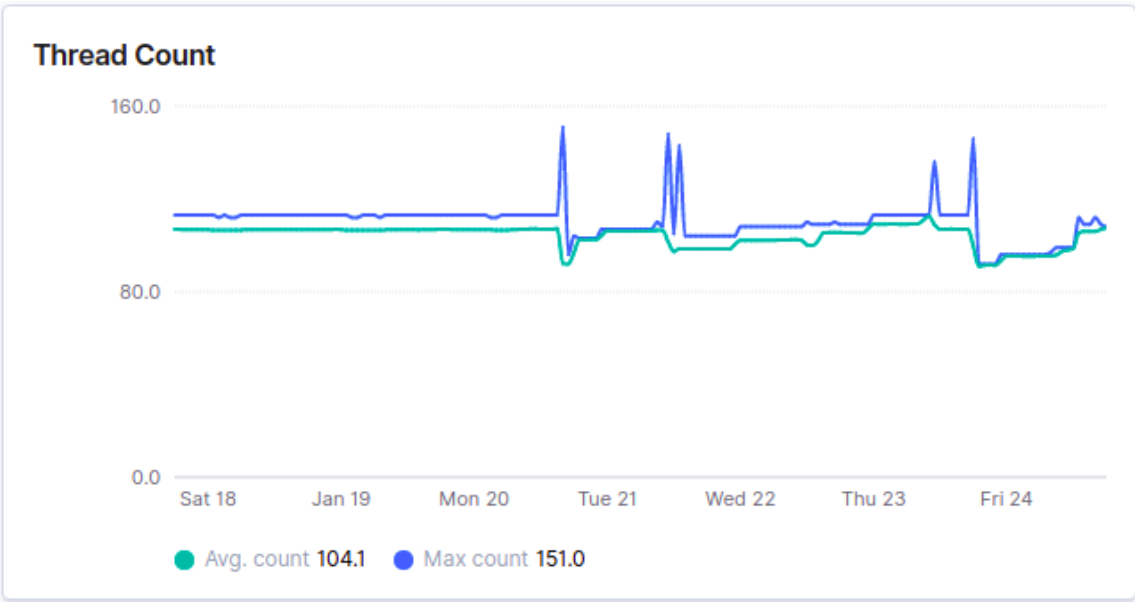

Figura 11: Quantidade de Threads

4.2.2 Kibana: análise de 1 dia

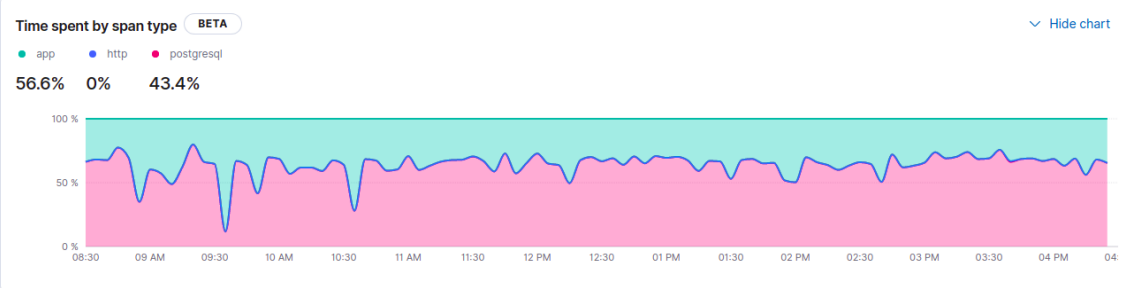

Figura 12: Tempo gasto por aplicação

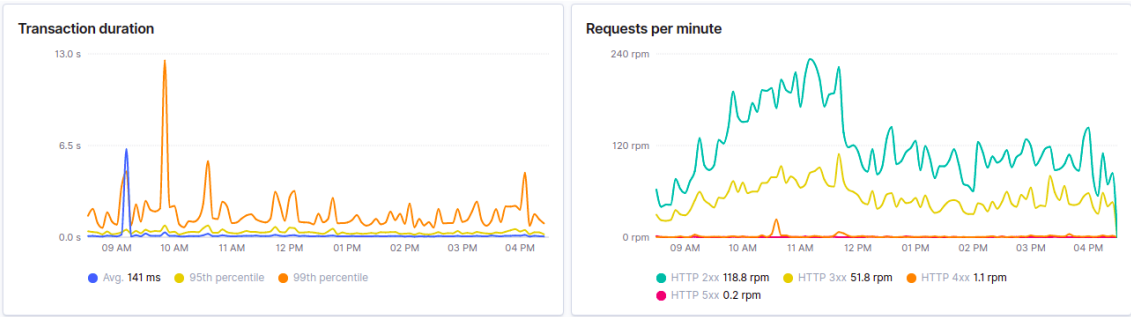

Figura 13: Duração de transação e quantidade de requisições por minuto

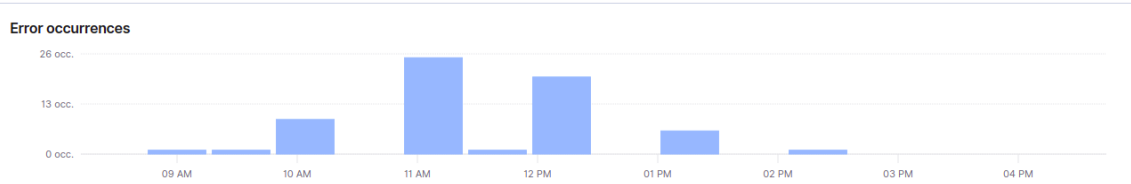

Figura 14: Ocorrência de erros

Errors

Group ID	Error message and culprit	Occurrences	Latest occurrence
b3781	ERROR: update or delete on table "tb_vsr_viagem_solicitacao_refu N/A	27	a day ago
cef39	ERROR: null value in column "src_ds_numero_processo_anterior" vi N/A	23	a day ago
f3d7d	ERROR: duplicate key value violates unique constraint "uk_vas_va N/A	8	a day ago
12bec	UT000010: Session not found VV5pOBqquR_LHsfRAa-bg9MQ N/A	3	a day ago
67861	Java heap space N/A	2	a day ago
b972a	ERROR: new row for relation "tb_scp_solicitacao_recadastro_princ N/A	1	a day ago

Rows per page: 25

Figura 15: Classificação de erros

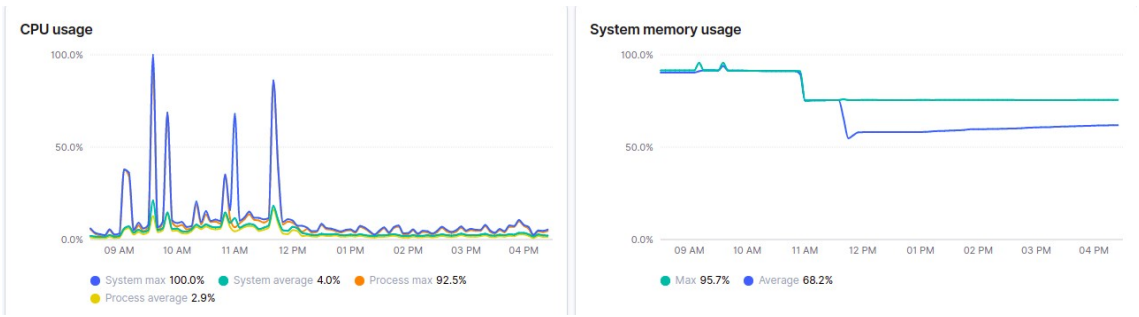

Figura 16: Utilização de CPU e memória

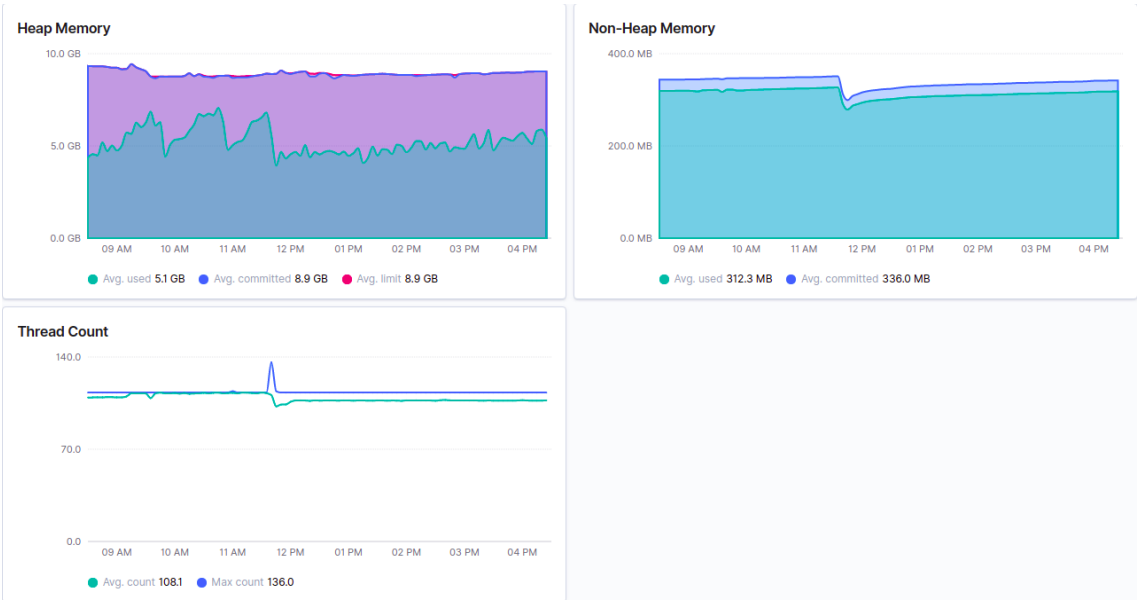

Figura 17: Utilização de memória e quantidade de Threads

4.2.3 Análise sobre os resultados

Este tópico tratará tecnicamente a análise baseado nos resultados obtidos pelas ferramentas citadas, juntamente com a análise amostral do código fonte, parecer da equipe técnica atuante no projeto e relatos da equipe de produção do Ministério da Justiça.

4.2.3.1 Manutenibilidade de código

Os relatórios apresentados pela ferramenta SonarQube demonstram uma série de vícios adotados durante o processo de construção do software e alinhado a estes vícios, o comportamento da ferramenta em ambiente produtivo demonstra instabilidade funcional.

A inexistência de cobertura de testes de unidade que trazem a dificuldade no processo de refactoring da aplicação, uma vez que não há condições de mensurar impactos durante o processo de manutenção corretiva/adaptativa. Alinhado a esta questão, a alta complexidade ciclomática e a falta de coesão dificultam este processo de refactoring, a ilustração abaixo demonstra ambos cenários apresentados em um único método (OBS: a característica apresentada é utilizada de forma recorrente em diversos momentos do código).

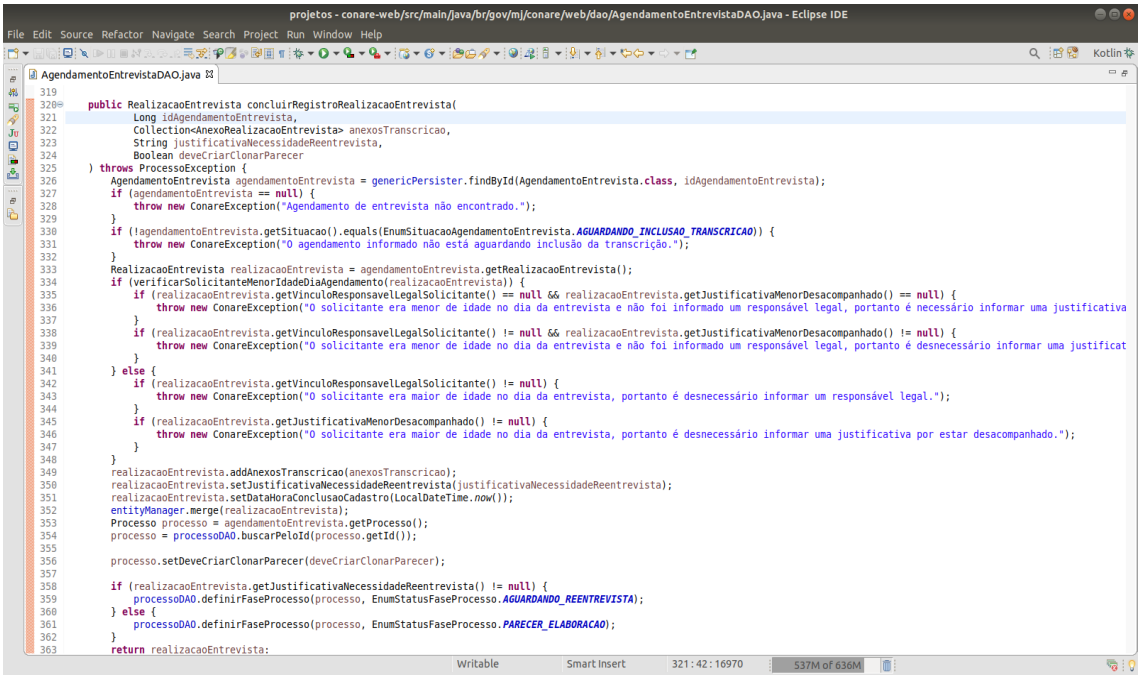

Figura 18: Camada de acesso a dados tratando regras negociais com alta complexidade ciclomática

4.2.3.2 Confiabilidade

Existe o tratamento de controle transacional na camada de serviço da aplicação, este é o tratamento segue boas práticas de desenvolvimento de aplicações, sendo esta camada responsável por orquestrar as execuções em banco de dados. Este controle transacional garante as propriedades ACID do SGBD.

Há erros por violação de chaves apresentado no relatório da ferramenta Kibana, estes erros não comprometem a integridade da informação dado a garantia imposta pelo SGBD, contudo, necessitam de revisão para evitar impedimentos de utilização da ferramenta.

Outra boa prática adotada neste projeto foi a utilização de datasource gerenciado pelo servidor de aplicação. Nela podemos gerenciar o pool de conexões e a disponibilidade dos recursos por ele gerenciado.

4.2.3.3 Performance e estabilidade

O erro relatado como *Heap Space* indica o estouro de memória heap na JVM, erro este causado quando não há possibilidade de expansão no uso da memória reservada para a JVM e a impossibilidade de limpar os objetos que ocupam o espaço de memória. Quando há o estouro da memória heap, a aplicação apresenta instabilidade e em alguns casos não consegue se recuperar, sendo necessário reiniciar a instância. Alinhado ao relatório apresentado pela ferramenta Kibana, a equipe de produção informa que alta frequência deste erro, se fazendo necessário reiniciar as instâncias de 3 a 5 vezes por semana (esta necessidade varia em conformidade com o fluxo de utilização da mesma).

Sendo o estouro de memória a principal causa de instabilidade e indisponibilidade da ferramenta, fatores como armazenamento de objetos compostos quando não utiliza o carregamento feito sob demanda (*Lazy Loading*), armazenamento de dados em sessão de usuário e armazenamento de objetos em estruturas de dados durante todo o ciclo de vida da aplicação são fatores que contribuem para este crescente consumo de recurso.

A má estruturação das consultas em banco de dados trazem consigo queda de performance da aplicação e também uma necessidade crescente no consumo de memória, sendo esta causa também responsável pelo estouro de memória. Estas consultas também trazem consigo a alta necessidade de processamento no servidor de banco de dados conforme demonstrado nas figuras 6 e 12 extraídas da ferramenta Kibana.

4.2.3.3 Escalabilidade

A arquitetura monolítica citada no tópico 3 deste documento prejudica a escalabilidade da ferramenta, os recursos empreendidos

para a escalabilidade vertical (aumento de recursos de processamento, disco, memória e demais) são limitados e atualmente são desproporcionais para a demanda atual de utilização da ferramenta (há mais recurso que o necessário, levando em consideração uma boa estrutura de código).

A escalabilidade vertical do monólito é possível levanto em consideração o aumento de nós no cluster, contudo esta escalabilidade é prejudicada tendo em vista que temos que escalar a aplicação como um todo, necessitando assim da mesma quantidade de recursos empreendidas nos demais nós existentes. Nesta arquitetura não há a possibilidade de escalar somente as funcionalidades/módulos que mais são demandados.

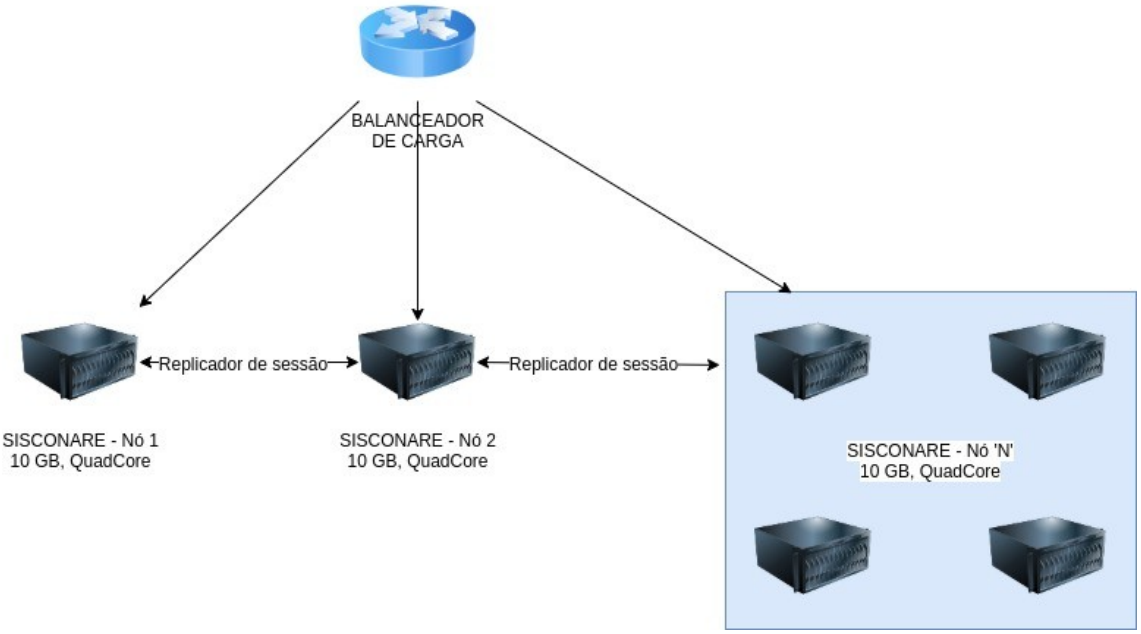

Figura 18: Server Farm SISCONARE

5 Recomendações

Toda a recomendação que tange refactoring de aplicação deve ser feita de forma investigativa e com auxílio de ferramentas de APM, afim de identificar gargalos e consumo de recurso. Por se tratar de um trabalho investigativo, este deve ser auxiliado pela área negocial do Ministério da Justiça, este trabalho é moroso dado a sua natureza investigativa.

De forma a minimizar os problemas enfrentados na manutenção da ferramenta em ambiente produtivo durante a escrita desta nota técnica, as recomendações que se seguem possuem foco na estabilidade da mesma. Estão fora destas recomendações as evidências apontadas pela ferramenta SonarQube (cobertura de testes, bugs, complexidade ciclomática e demais) juntamente com a coesão de código, não por falta de relevância e sim pela objetividade momentânea.

- Refactoring de código
    - Remover controle transacional das consultas na camada de persistência;
    - Refazer consultas com base nas necessidades de cada tela (existem múltiplos joins que trazem objetos que não estão sendo utilizados);
    - Remover a utilização das métodos adicionarObserver e removerObserver da classe br.gov.mj.conare.web.security.LocaleObservable. Estas implementações armazenam uma instância de cada classe Controller quando a mesma é instanciada, não há evidência de utilização destas instâncias por intermédio da classe LocaleObservable. Pela forma que está implementado, os objetos ali armazenados perduram por todo o ciclo de vida da aplicação, não havendo a destruição dos mesmos e consequentemente sendo um dos grandes causadores do grande consumo de memória;
    - Refatorar estratégia de utilização de armazenamento em sessão de usuário (após efetivado o login) os objetos que armazenam dados de perfis de acesso a ferramenta. Sugere-se a implementação de filtros interceptadores para realizar a atividade de autorização de acesso;
    - Utilização do container EJB para injeção de dependência por inversão de controle, tendo em vista que as classes dos pacotes das camadas de serviço e persistência são EJB’s e atualmente estão sendo injetadas pelo container CDI;
    - Utilizar o carregamento de arquivos (array de bytes) somente quando solicitado pelo usuário e ao término de sua

utilização (Lazy Loading), recomenda-se também que o mesmo seja disponibilizado ao Garbage Collector após a sua finalidade negocial para que haja a liberação de recursos de memória.

- Limitar tamanho de arquivos para upload, isso irá otimizar não somente os recursos de memória da instância mas também otimizará a utilização de recursos de rede. Vale ressaltar que com a implementação atual, um único usuário consegue indisponibilizar todo o serviço somente com a ação de upload de arquivo.

- Novos módulos/funcionalidades: não recomenda-se que a construção de novos módulos/funcionalidades seja feita em cima da arquitetura atual pelos seguintes fatores:
  - Baixa produtividade de implementação com a utilização do framework Apache Wicket;
  - Dificuldade de encontrar respostas aos problemas apresentados durante a fase de desenvolvimento com o framework Apache Wicket, fator este atribuído a sua baixa utilização comparado aos demais frameworks de mercado;
  - Dificuldade de escalonamento da aplicação (fatores expostos no item 4.2.3.3);
- Utilizar algoritmo Conc Mark Sweep GC nas configurações da JVM. Em testes realizados em ambiente de desenvolvimento local, este algoritmo demonstrou melhor performance na aplicação diminuindo a alocação de memória da JVM, porem aumentou a utilização de CPU;

- Utilizar camada de segurança SSL/TLS em ambiente produtivo (atualmente a ferramenta está operando com protocolo HTTP e não HTTPS);

- Atualizar versão do Apache Wicket 6.19.0 para versão 6.26.0 dado a existências de vulnerabilidades de ataques XSS (Cross-Site Scripting) e ataques Denial of Service (Negação de serviço);

- Pelo desconhecimento da disposição da infraestrutura atual, recomenda-se (caso não haja) a implantação de ferramentas de segurança que atuem na camada de 7 do modelo OSI (firewall de aplicação);

6 Conclusão

A implementação das recomendações realizadas nesta análise certamente trarão maior estabilidade ao funcionamento da ferramenta e uma melhora significativa da utilização de recursos computacionais e de redes. Sugere-se após a realização dos mesmos a execução de testes de carga/stress fim de aferir a capacidade de atendimento, aderência as expectativas de utilização e a necessidade de dimensionamento da infraestrutura.

Recomenda-se também que os novos módulos da ferramenta sejam construídos sob a utilização da arquitetura de referência proposta ao Ministério da Justiça. Esta arquitetura tem como preceito a segregação do front-end com o back-end, sendo facilitado por este cenário a criação de serviços especialistas, independentes e que possuem alta flexibilidade e facilidade de escalonamento.

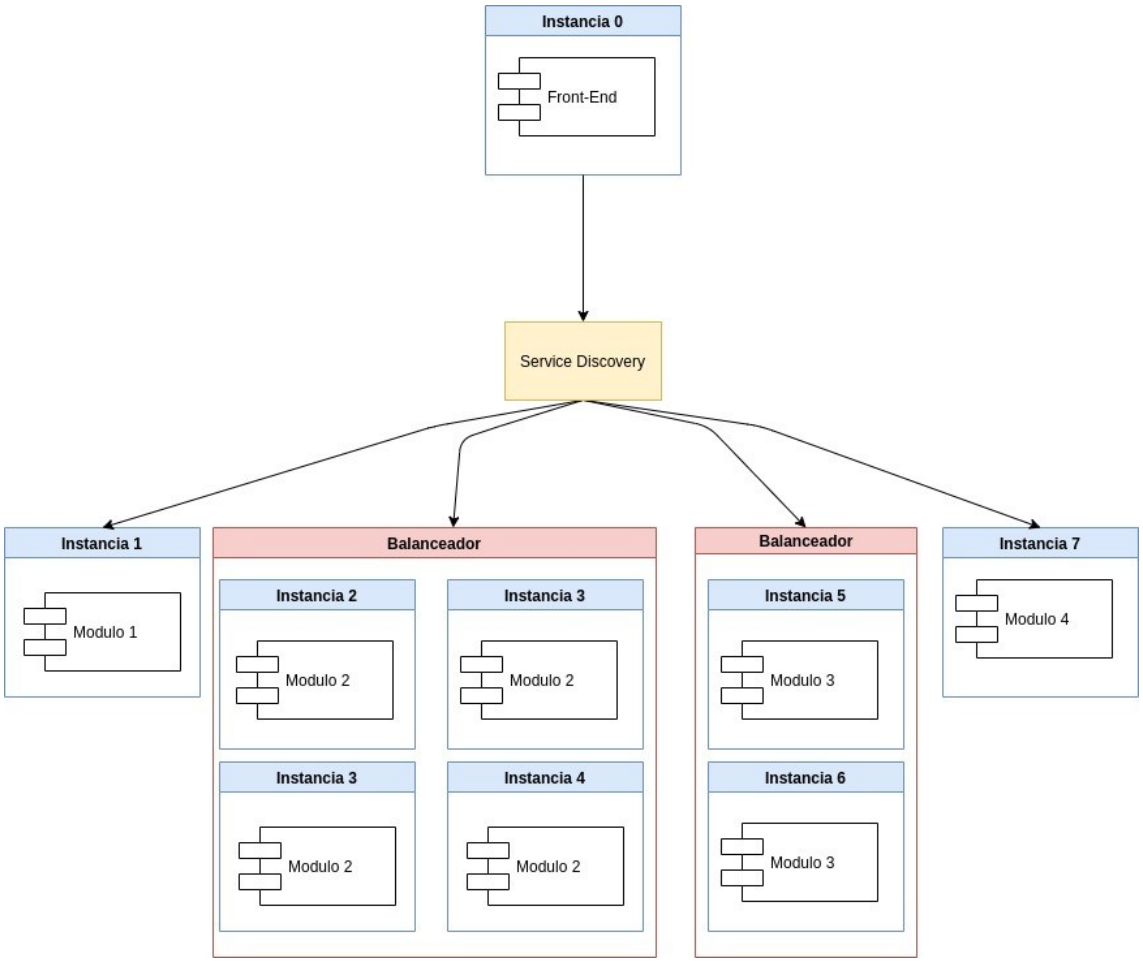

Figura 19: Escalonamento de módulos

A representação acima demonstra a facilidade de escalonar módulos em conformidade com a demanda negocial na aplicação. Com a aplicação da estratégia adequada, é possível a coexistência entre os módulos da plataforma antiga com a nova trazendo assim transparência no uso das funcionalidades para o usuário, manutenção de sessão/token, utilização dos mesmos mecanismos de autenticação/autorização, reaproveitamento de código do componente conare-entity demonstrado na figura 1 e o vislumbre da reconstrução do legado em detrimento da utilização da arquitetura proposta, isso tudo de forma escalar, modular e sob a ótica da necessidade negocial.

7 Relatório de execução de testes

Os relatórios das execuções dos testes de carga/stress e vulnerabilidade estão respectivamente disponíveis em:

- https://justicagovbr.sharepoint.com/:u:/r/sites/pwa/CONARE/Shared%20Documents/CTIS/Evidencia%20teste%20de%20Carga.7z?csf=1&e=zzEn9X
- https://justicagovbr.sharepoint.com/:u:/r/sites/pwa/CONARE/Shared%20Documents/CTIS/Analise%20de%20Vulnerabilidades%20CONARE.7z?csf=1&e=8PJLQk