Ministério da Justiça

Projeto: GESTÃO DE RISCO

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	06/03/2020

1 Sumário

2 Introdução

Este documento visa reportar o resultado da análise efetuada na aplicação GESTÃO DE RISCO. Para este estudo foram desconsiderados todo o contexto negocial ao qual a ferramenta está inserida, também foram desconsideradas o ambiente ao qual a ferramenta esta operando sendo analisado puramente questões que tangem a qualidade de código, padrões de codificação, vulnerabilidades de dependências, modelo relacional de banco de dados e concepção arquitetural.

3 Apresentação do cenário atual

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

O sistema GESTÃO DE RISCO está construído para funcionar em ambiente WEB com uma segregação entre as camadas de front-end e back-end, sua arquitetura esta projetada para trabalhar de forma desacoplada e distribuída.

O backend da aplicação está construído sobre a stack Java Enterprise Edition, já a aplicação front-end está construída para trabalhar como uma SPA – Single Page Aplication com o framework Angular.

O diagrama a seguir representa o modelo de componentes ao qual a aplicação está construída, suas dependências e seu modelo de comunicação.

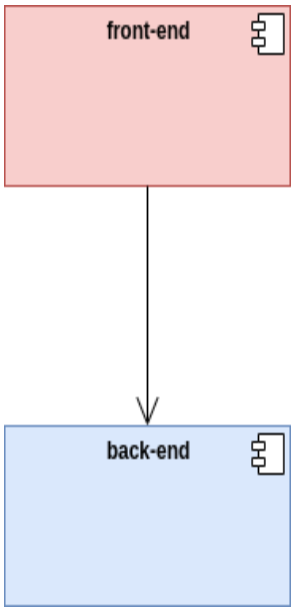

Figura 1: Diagrama de componentes

Diferentemente dos sistemas dotados com a mesma arquitetura de referência (como exemplo o sistema Orcrim), o sistema GESTÃO DE RISCO não possui estrutura componentizada em seu back-end. Há um único projeto que contempla todas as camadas e responsabilidades do back-end organizadas por pacotes.

3.1 Front-End

Esta camada representa a interface com o usuário da aplicação e está organizada de forma lógica por segregação de módulos/funcionalidades, proporciona boa capacidade produtiva para manutenções corretivas e evolutivas.

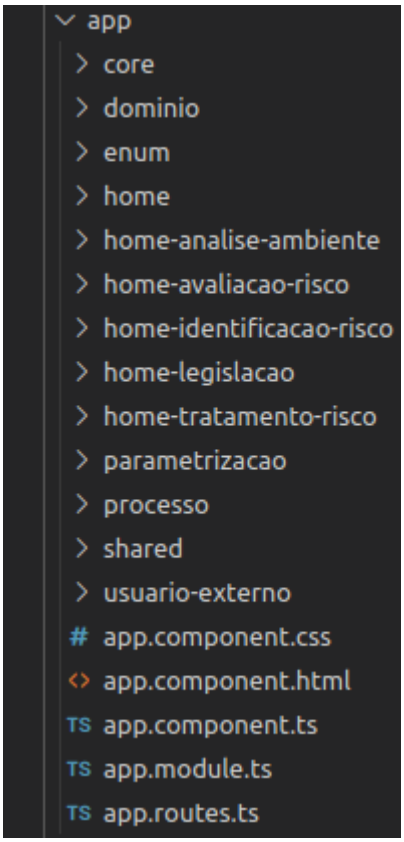

Figura 2: Organização do projeto front-end

3.2 Back-end

Este componente representa toda a camada de back-end da aplicação, contendo em sua composição:

- Classes utilitárias para iteração com SGBD (Sistema Gerenciador de Banco de Dados).
- Classes POJO (Plain Old Java Object) que possuem objetivo de trafegar estado entre as camadas da aplicação (VO (Value Object) e DTO (Data Transfer Object)).
- Classes que disponibilizam API Rest (Representation Transfer State) para consumo da aplicação front-end com a utilização do Design Pattern Facade.
- Classes de serviço que gerenciam as regras de negócio e efetuam requisições para a camada de acesso a dados.

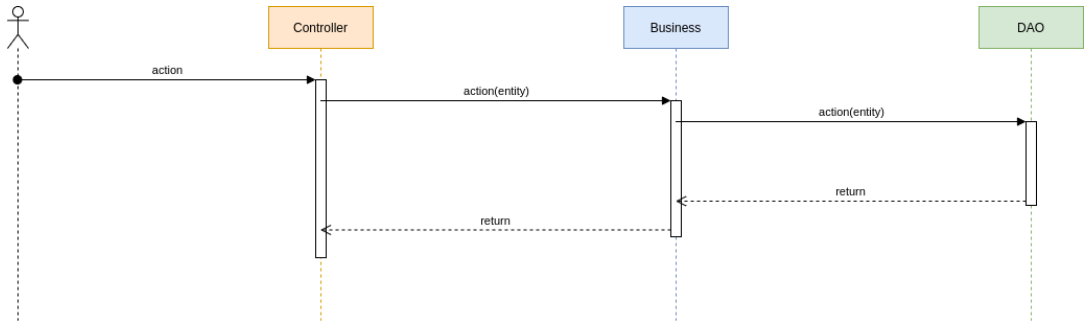

Figura 3: Sequência padrão de ações

Figura 4: Orgranização do projeto back-end

3.2 Tecnologias utilizadas

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

Nome	Versão	Utilização	Observação
Java	1.8	Linguagem de programação.	
Javaee-api	7.0	Stack Java Enterprise Edition	
Hibernate	4.3.7	Framework ORM	
Wildfly	8.x	Servidor de aplicação JEE.	Utiliza containers EJB, CDI e Servlet.
Jackson-Annotations	2.4.1	Biblioteca para serializar/ deserializar quando se trabalha com Pojos/Json	
Junit	4.11	Utilizado para a criação de testes automatizados.	
Keycloak	3.4.0	Framework para gerenciamento de identidades e acesso.	
PostgreSQL		Banco de dados relacional	

3.3 Modelagem de dados

A estrutura de banco de dados esta composta pela utilização de 4 schemas (apoio, auditoria, seg e gestao_riscos), estas estruturas não demonstram ausência de normatização em sua composição.

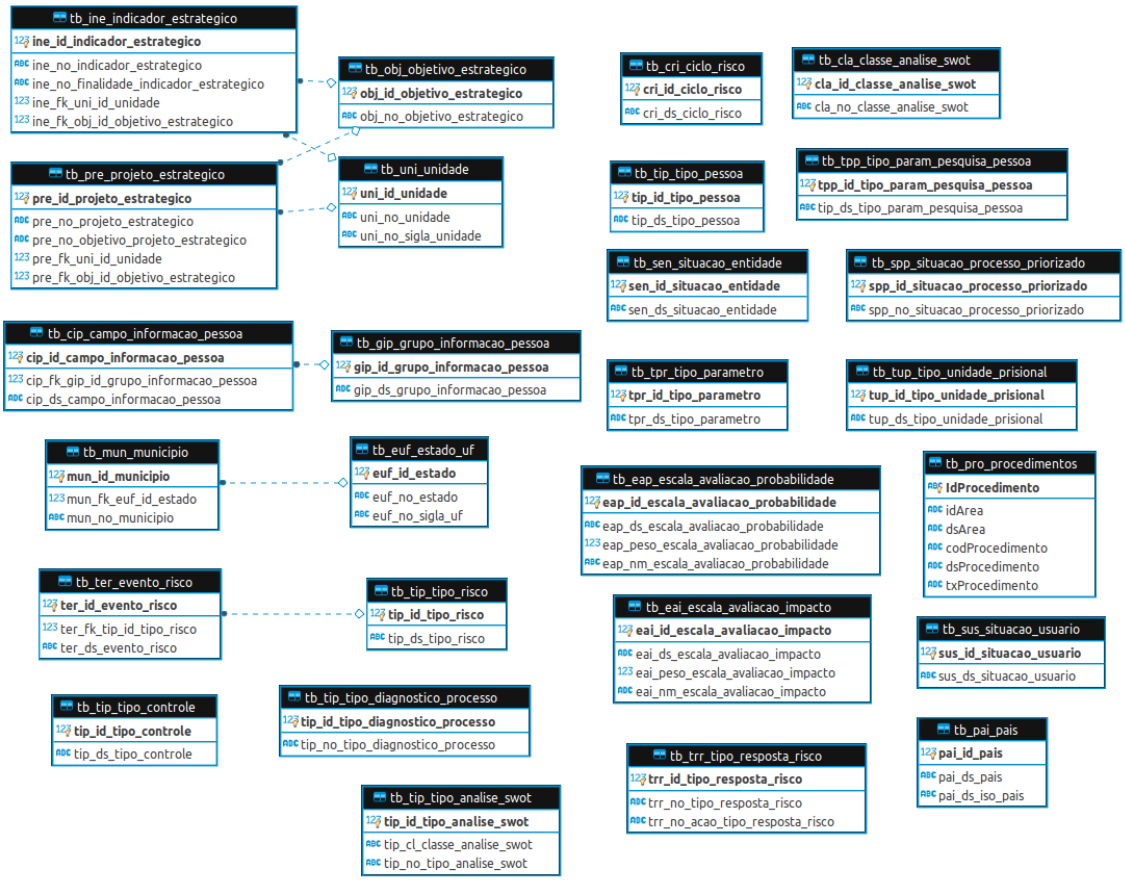

Figura 5: Apoio

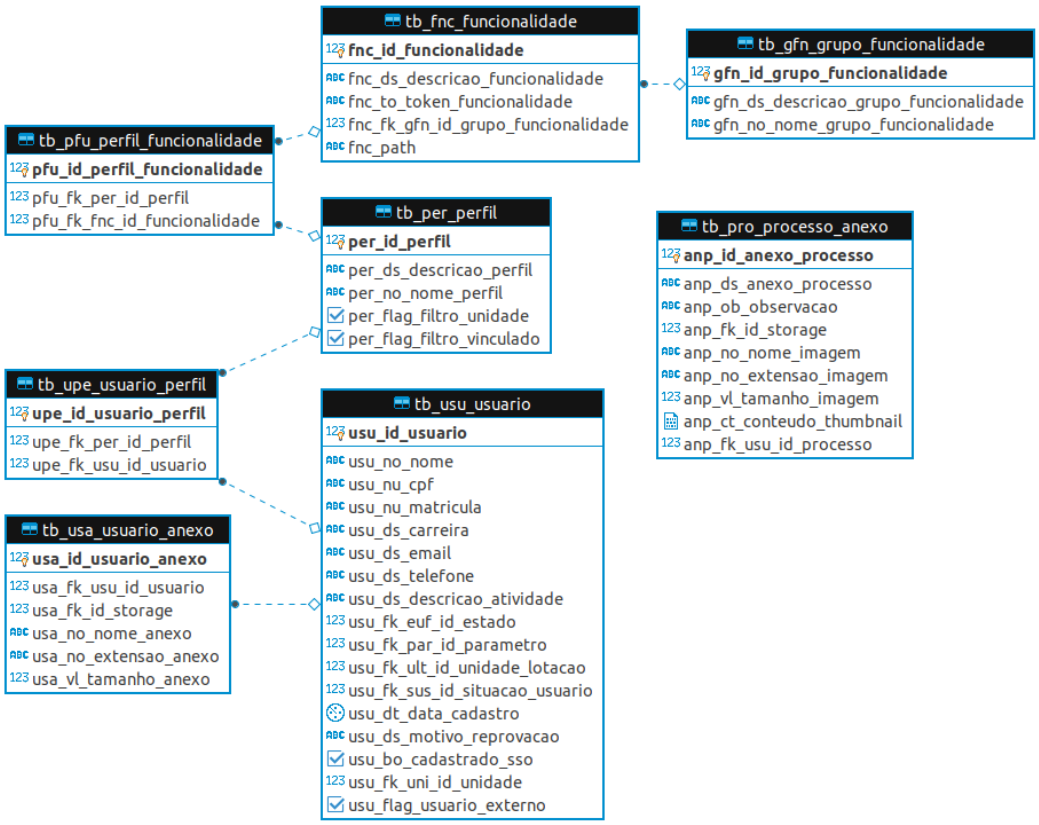

Figura 6: Schema Seg

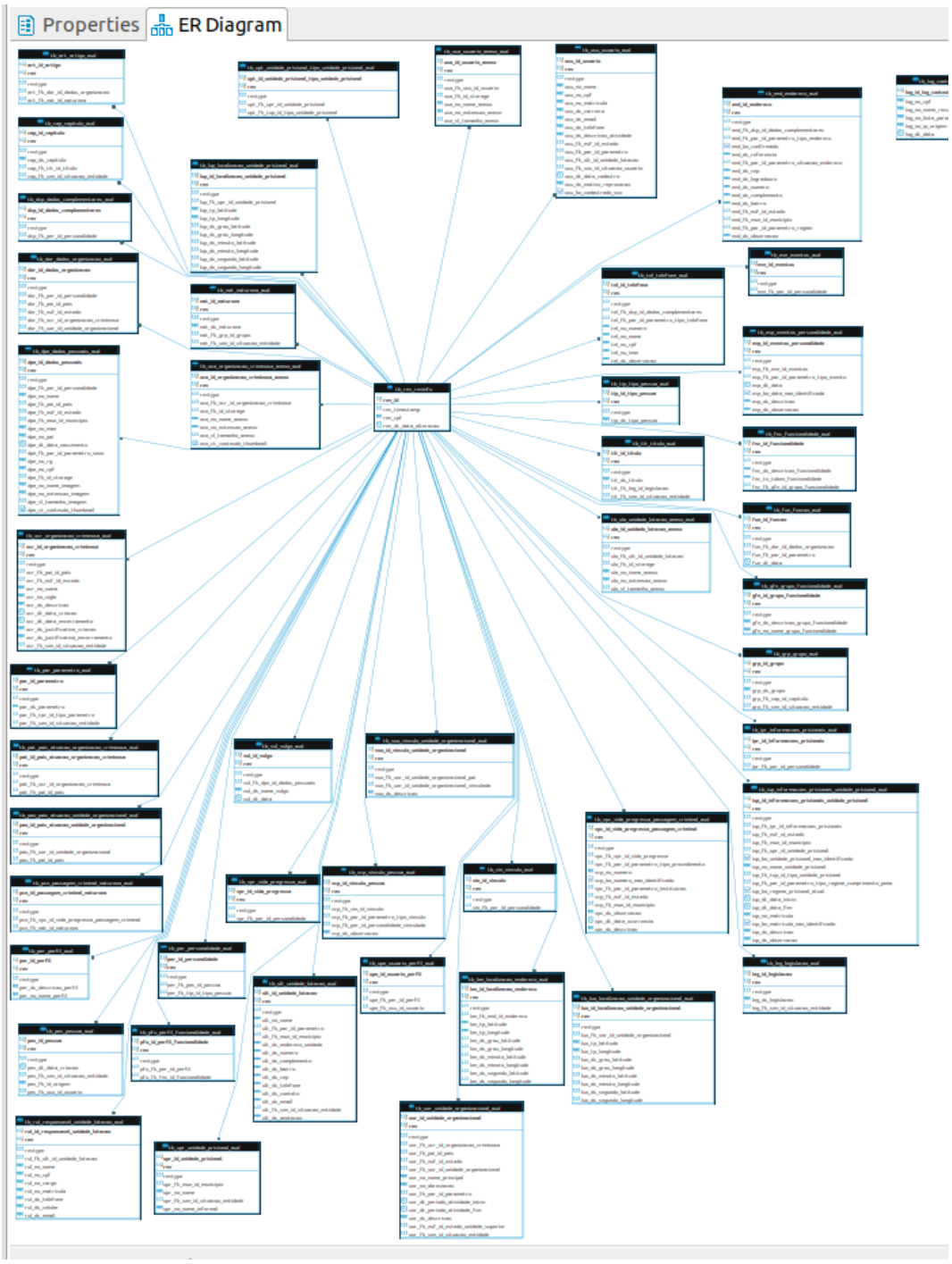

Figura 7: Schema auditoria

Figura 8: Schema gestao_riscos

4 Análise técnica

Este tópico descreve a ferramenta do ponto de vista técnico, tanto nos aspectos de codificação, análise estática de código, análise de vulnerabilidade de dependências e particularidades de implementação.

4.1 SonarQube

Ferramenta utilizada para verificação de estática de código. Para esta análise não foram utilizadas as métricas de qualidade implantadas no SonarQube do Ministério da Justiça, contudo foram utilizadas as regras padrões de análise da ferramenta. Os resultados foram os seguintes para a aplicação back-end:

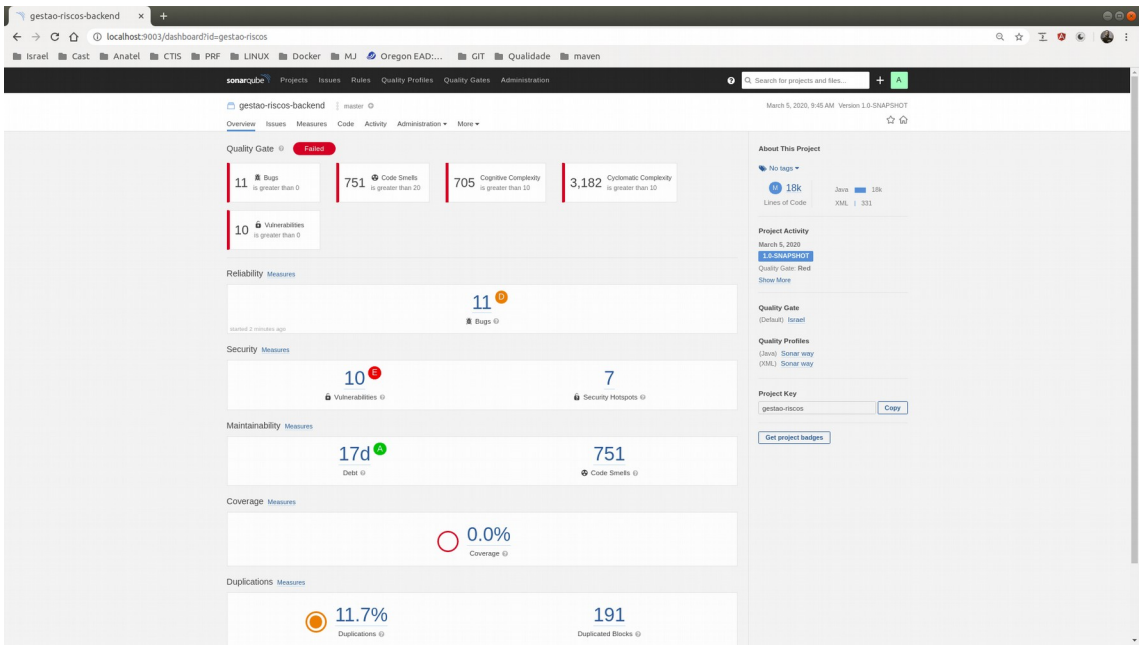

Figura 9: Análise estática de código

Para a obtenção dos resultados, fora utilizado o código fonte com último histórico de commit (correão_issues_capgemini):

- 11 bugs;
- 751 violações de más práticas;
- 705 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 3182 violações de complexidade ciclomática (complexidade de código);
- 10 vulnerabilidades;
- 11.7% de duplicação de código;

A ferramenta apresenta 0% de cobertura de testes, o código fonte não apresenta artefatos de testes de unidade.

4.2 OWASP Dependency Check

A utilização de bibliotecas de terceiros aumenta substancialmente a produtividade na construção de um software, contudo estas podem trazer consigo vulnerabilidades que afetam diretamente a segurança da aplicação. A ferramenta Dependency Check tem como propósito efetuar análise de vulnerabilidade de dependências utilizadas no projeto back-end, a seguir temos as principais informações extraídas desta análise.

Dependency	Highest Severity	CVE Count	Confidence	Evidence Count
jackson-databind-2.4.1.jar	CRITICAL	24	Highest	38
resteasy-jaxrs-3.0.9.Final.jar	HIGH	3		21
keycloak-core-3.4.0.Final.jar	CRITICAL	12	Highest	35
keycloak-adapter-core-3.4.0.Final.jar	HIGH	2	Highest	35
bcprov-jdk15on-1.56.jar	CRITICAL	3	Low	47

A planilha acima apresenta as vulnerabilidades encontradas nas dependências do componente back-end, o detalhamento encontra-se no Anexo I deste documento.

4.3 NPM Audit

Após realizar a instalação dos módulos de dependências gerenciados pelo NodeJS (npm install), foram reportados existências de 45 vulnerabilidades na aplicação/dependências e foram classificadas como: 15 são de baixo risco, 11 de riscos moderados, 18 são alto risco e 1 são críticos.

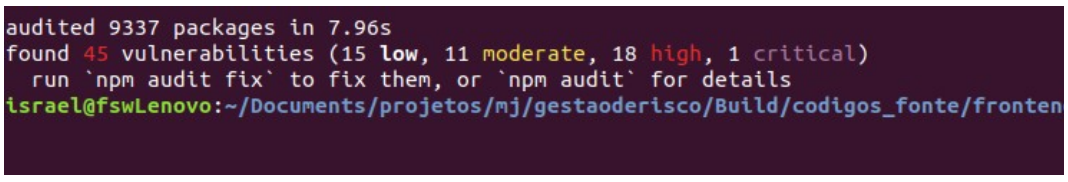

Figura 10: Execução do comando npm install

A partir da versão 6 do gerenciador de pacotes NPM, fora disponibilizado a ferramenta “Audit” que além de auxiliar na descoberta dos itens vulneráveis auxilia através do comando “fix” a atualização/resolução de parte dos problemas. Após a execução do mesmo restaram 6 vulnerabilidades que requerem intervenção manual.

As vulnerabilidades listadas são oriundas das versões de dependências utilizadas neste projeto, a atualização das mesmas deve ser feito de forma cautelosa e baseada em testes para que não haja instabilidade em seu uso.

O relatório completo pode ser acessado através da execução do comando “npm audit”.

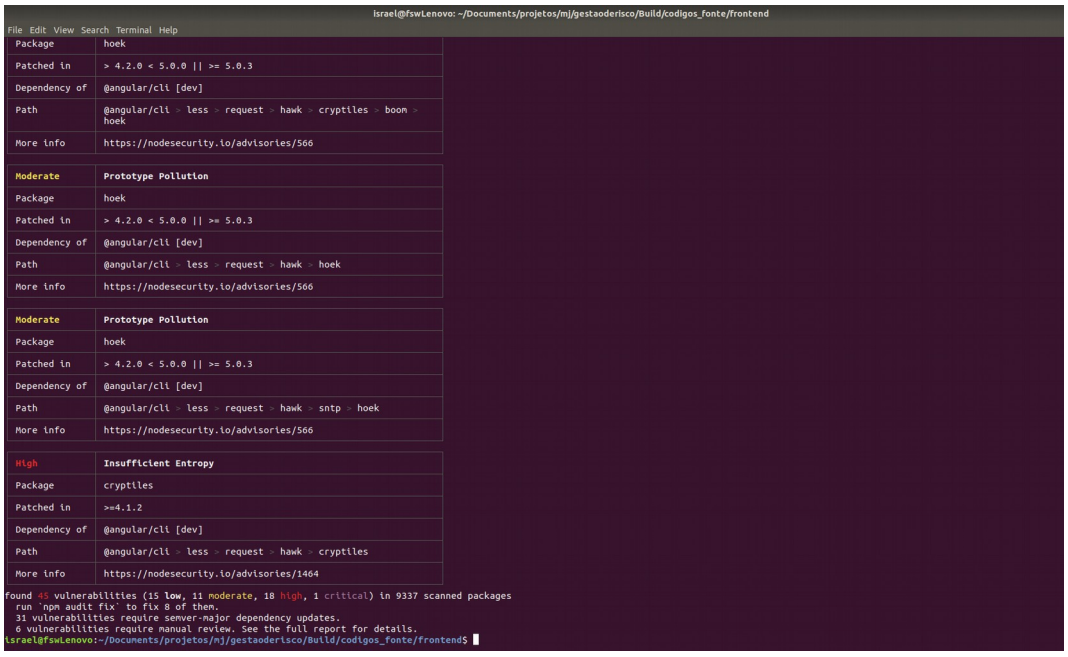

Figura 11: Execução do comando npm audit

4.4 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.

4.4.1 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.

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. A alta complexidade ciclomática dificulta o processo de refactoring, a ilustração abaixo demonstra o cenário apresentados(OBS: a característica apresentada é utilizada de forma recorrente em diversos momentos do código).

```
private void atualizaUsuarioSso(Usuario entidadeAtual) {

    String primeiroNome = entidadeAtual.getNome().split(" ")[0];
    String sobrenome = entidadeAtual.getNome().substring(primeiroNome.length(), entidadeAtual.getNome().le

    if (entidadeAtual.getSituacao().getId().equals(situacaoUsuarioService.recuperarAtivo().getId()) && !en

        try {
            keyCloakDAO.criarUsuario(entidadeAtual.getCpf(), primeiroNome, sobrenome, entidadeAtual.getEma
            entidadeAtual.setCadastradoSSO(Boolean.TRUE);
        } catch (IOException e) {
            e.printStackTrace();
            throw new NegocioException("msg.usuario.keycloack.erro");

        }
        entidadeAtual.setCadastradoSSO(Boolean.TRUE);
    } else if (entidadeAtual.getSituacao().getId().equals(situacaoUsuarioService.recuperarInativo().getId
        try {
            keyCloakDAO.desativarUsuario(entidadeAtual.getCpf());
        } catch (IOException e) {
            e.printStackTrace();
            throw new NegocioException("msg.usuario.keycloack.erro");

        }
    } else if (entidadeAtual.getSituacao().getId().equals(situacaoUsuarioService.recuperarAtivo().getId()
        try {
            keyCloakDAO.atualizarUsuario(entidadeAtual.getCpf(), primeiroNome, sobrenome, entidadeAtual.ge
            entidadeAtual.setCadastradoSSO(Boolean.TRUE);
        } catch (IOException e) {
            e.printStackTrace();
            throw new NegocioException("msg.usuario.keycloack.erro");

        }
    }

}
```

Figura 12: Alta complexidade ciclomática – UsuarioService.java

4.4.2 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 a camada responsável por orquestrar as execuções em banco de dados. Este controle transacional garante as propriedades ACID do SGBD.

4.4.3 Performance e estabilidade

Não foi analisado a aplicação em funcionamento para avaliar demais requisitos não funcionais contudo por análise de código fora identificada a presença de arrays de bytes nas entidades que transitam valor. A presença de funcionalidades de download/upload de arquivos na solução deve ser bem analisada para que a infraestrutura seja bem dimensionada evitando assim instabilidades em seu comportamento por estouro de memória.

4.4.3 Escalabilidade

A arquitetura baseada em micro serviços e o comportamento sem estado (stateless) da aplicação back-end promove a esta uma boa capacidade de escalonamento na horizontal com a utilização de cluster e balanceadores de carga.

Esta arquitetura além de promover ambiente escalonável, favorece a utilização de ambientes redundantes com maior probabilidade de tolerância a falhas.

4.4.3 UX – User experience

Durante a análise de dependência do projeto front-end foi notado

a presença da dependência @angular/material versão 2 beta e da dependência PrimeNG versão 4.3. Ambas dependências objetivam trazer ao projeto a utilização de componentes visuais reutilizáveis, contudo a utilização de ambas em um mesmo projeto pode trazer (quando não devidamente tratado) dualidade nos padrões de componentes de interface, trazendo ao usuário uma experiência negativa dado a falta de padronização de telas e de seus componentes.

5 Recomendações

É altamente recomendado que seja efetuado refactoring de código dos bugs e vulnerabilidades de código apontadas pelo SonarQube , estas atividades certamente trarão maior confiabilidade a ferramenta e estabilidade em seu uso. Para os demais itens apontados pela ferramenta SonarQube durante o processo de análise de código são altamente desejáveis, contudo este processo de ajuste de código é moroso e trás consigo risco em potencial e está diretamente aliado a falta de cobertura de testes de unidade.

Ajustar as dependências que trazem maior risco para a aplicação é altamente recomendável, contudo este trabalho deve ser feito de forma analítica e cautelosa afim de não prejudicar a estabilidade da ferramenta. Sugere-se a utilização de ferramentas de pentest (análise de vulnerabilidade) tais como OWASP ZAP (https://owasp.org/www-project-zap/) para que seja analisado as principais vulnerabilidades da aplicação e associá-las as dependências que oferecem tais riscos para os devidos ajustes. Esta recomendação esta embasada na interseção de resultados das ferramentas utilizadas e na otimização e na assertividade do trabalho de refactoring.

O código fonte apresenta referências de classes no pacote Builder que fogem ao contexto da aplicação e não são utilizadas em nenhuma parte do sistema (Ex: OrganizacaoCriminosaBuilder, LocalizacaoUnidadePrisionalBuilder, LocalizacaoUnidadeOrganizacionalBuilder dentre outras), apresenta tambem no arquivo messages_pt referencias de textos com a mesma característica citada. Recomenda-se que seja efetuado uma higienização neste código afim de enxugar a aplicação e remover as partes desnecessárias.

Recomenda-se também que seja instalado o agente da

ferramenta de APM do Ministério da Justiça nos ambientes de homologação e produção, criar métricas e alarmes auxiliam na continuidade do serviço (monitoramento de processamento e memória por exemplo) tendo em vista que esta ferramenta fornece mecanismos para determinarmos o comportamento da solução (auxiliam no refactoring de código) também subsidia para o correto dimensionamento da infraestrutura.

Não há evidências que comprovem instabilidade na versão 4 do framework Angular, contudo caso exista o interesse em atualizá-lo recomenda-se a execução de estudo de viabilidade técnica e dimensionamento de esforço para a atualização da mesma para a versão mais estável mais atual (a versão 8 do framework é versão mais recente e estável no momento da escrita desta nota técnica). Dentre as vantagens da atualização da versão temos: suporte a versão 3.2 do TypesScript, otimização do processo de build, suporte a nova versão da biblioteca RxJS, lazy loading modules, differencial loading module e novas features do Angular Material.

Independentemente da versão utilizada no framework Angular, é altamente recomendado a manutenção de somente uma extensão de componentes seja angular material (versão superior a 2 Beta) ou primeng.

Não há evidencia de utilização do Schema auditoria na aplicação, segure-se a remoção do mesmo.

6 Conclusão

A aplicação apresenta uma boa estruturação em sua construção o que facilita a sua manutenção corretiva/evolutiva e aliado a necessidade de utilização dos recursos ofertados na versão 8 (ou superior) do framework Angular para a aplicação front-end, sugere-se o estudo de viabilidade técnica para o devido upgrade

Em relação aos ajustes de código da aplicação, sugestiona-se que os mesmos sejam efetuados com o auxilio/suporte de testes unitários tendo em vista que este caminho trará maior assertividade e menor risco para o não comprometimento da estabilidade da ferramenta.