Departamento de Polícia Rodoviária Federal

Projeto: PRF MÓVEL

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	20/11/2020

1 Sumário

2 Considerações iniciais

O presente documento tem por objetivo reportar a análise efetuada no sistema de PRF-Móvel. 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.

Como insumo para esta análise as branches https://git.prf/romulo.teixeira/sistemasmoveis-api, https://git.prf/jean.santiago/PRFMovel e https://git.prf/prf/silver/tree/prfmovel. Estas são respectivamente correlatas aos sistemas SistemasMoveisAPI, PRFMovel e Silver.

3 Apresentação do cenário atual

Esta sessão ira descrever a arquitetura, tecnologias, frameworks e dependências que compõe a base da solução PRF Móvel ao qual está construído para funcionar em ambiente MOBILE com requisições diretas a APIs Rest construídas especificamente para atendimento das requisições do mesmo.

A solução é composta por módulo mobile construída em com linguagem de programação Kotlin projetada para operar em dispositivos dotados do sistema operacional Android, conjuntamente com 4 módulos que expõem APIs Rest, sendo 3 destes construídos em linguagem Java com stack Spring Boot e um módulo escrito em linguagem Kotlin com stack Spring Boot.

Os diagramas que seguem esta sessão representam o modelo de componentes ao qual a aplicação está construída, suas dependências, fluxo sequencial e seu modelo de comunicação entre os módulos.

3.1 Documentação existente

Não há documentos disponíveis tais como, documento de arquitetura, documento negocial, manual do usuário, manual do desenvolvedor e manual de implantação.

3.2 Diagrama de componentes

O diagrama a baixo representa a composição da solução dividida em três divisões estruturais sendo elas uma camada de serviços internos/externos da PRF, uma camada específica do projeto composto por 4 projetos (SISCOM, EFM, Main e Comandos) onde cada um expõe um conjunto de endpoints utilizados pelo projeto PRF

Móvel, ao qual compõe a última camada deste diagrama.

A camada REST denominada daqui por diante como PRF Móvel API não utiliza composição de componentes em sua construção, todos os componentes são independentes e trabalham de forma isolada.

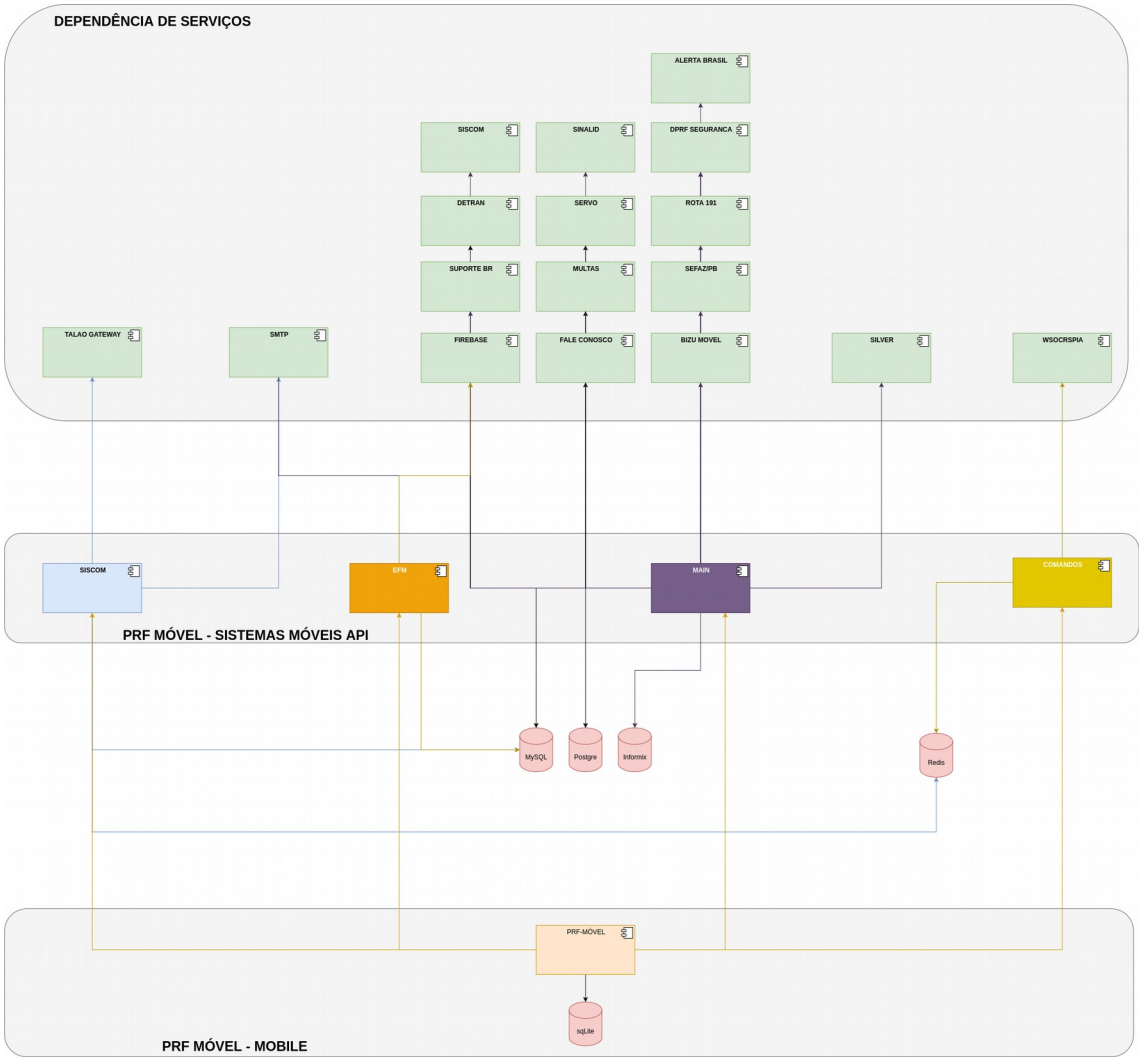

Figura 1: Figura 1: Diagrama de componentes

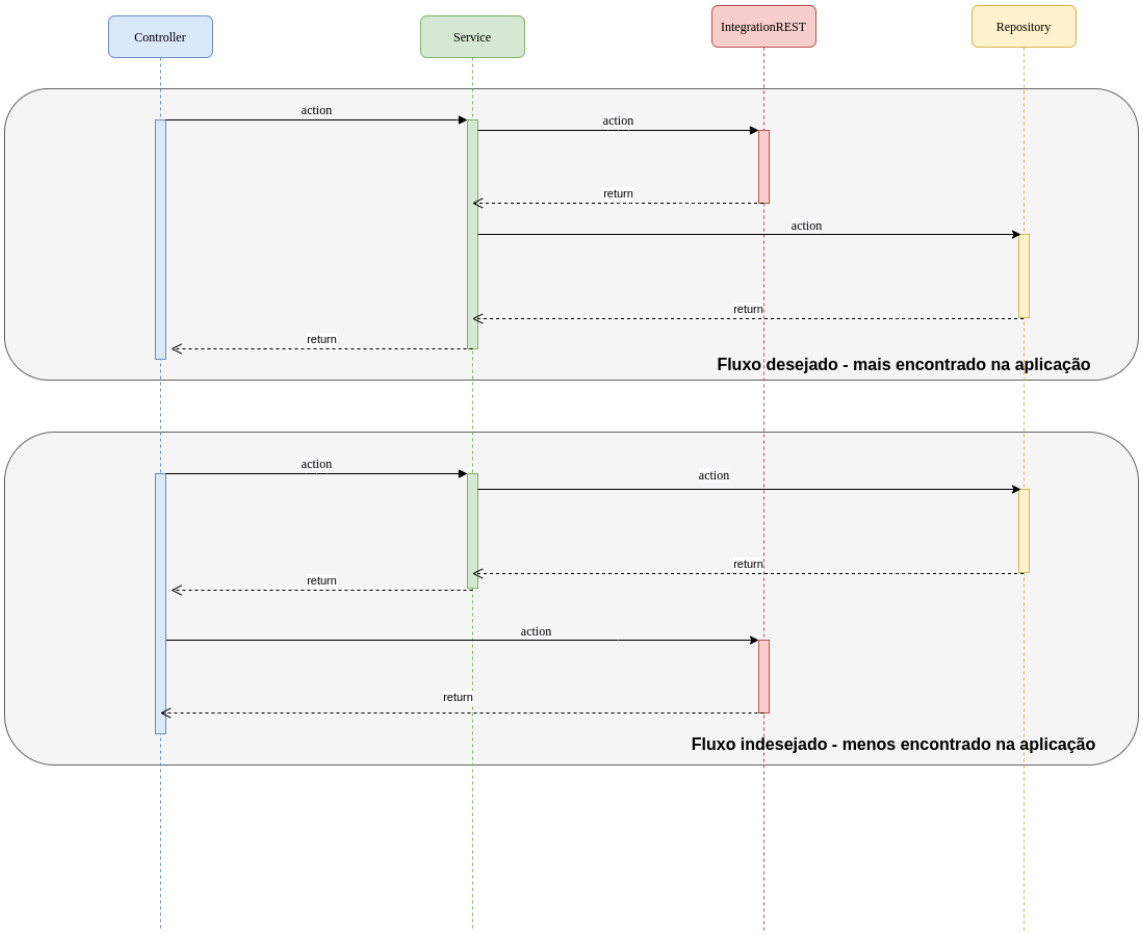

Figura 2: Diagrama de sequcias - API's Rest

4 Ambiente

A montagem de ambiente da solução consiste na preparação de de três cenários, sendo eles sistemas móveis API, silver e prf móvel.

Para o primeiro cenário de compilação e execução dos módulos que compõe o conjunto sistemas móveis API’s, se faz necessário a compilação dos módulos utilizando o ambiente Java JDK 8 com o gerenciador de build/dependências Apache Maven 3.3.x ou superior. Os módulos que compõe este projeto podem ser executados de forma isolada utilizando o Tomcat embarcado no Spring Boot ou em conjunto com a execução do módulo composer do Docker.

O projeto Silver (legado) necessita de compilação similar aos módulos citados acima. O projeto pertencente a branch especificada no inicio do documento possui descritor Dockerfile para sua execução em ambiente de container.

O ambiente móvel necessita de compilação com Gradle versão 6.7 ou superior. Para sua execução em ambiente de desenvolvimento também será necessário a utilização do Android Studio com a utilização do ambiente de simulação Android.

5 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.

5.1 SonarQube

Ferramenta utilizada para verificação de estática de código. Para esta análise não foram utilizadas as métricas de qualidade implantadas no SonarQube da DPRF, contudo foram utilizadas as regras padrões de análise da ferramenta.

Esta análise se restringe aos projetos que compõe a camada PRF Móvel API.

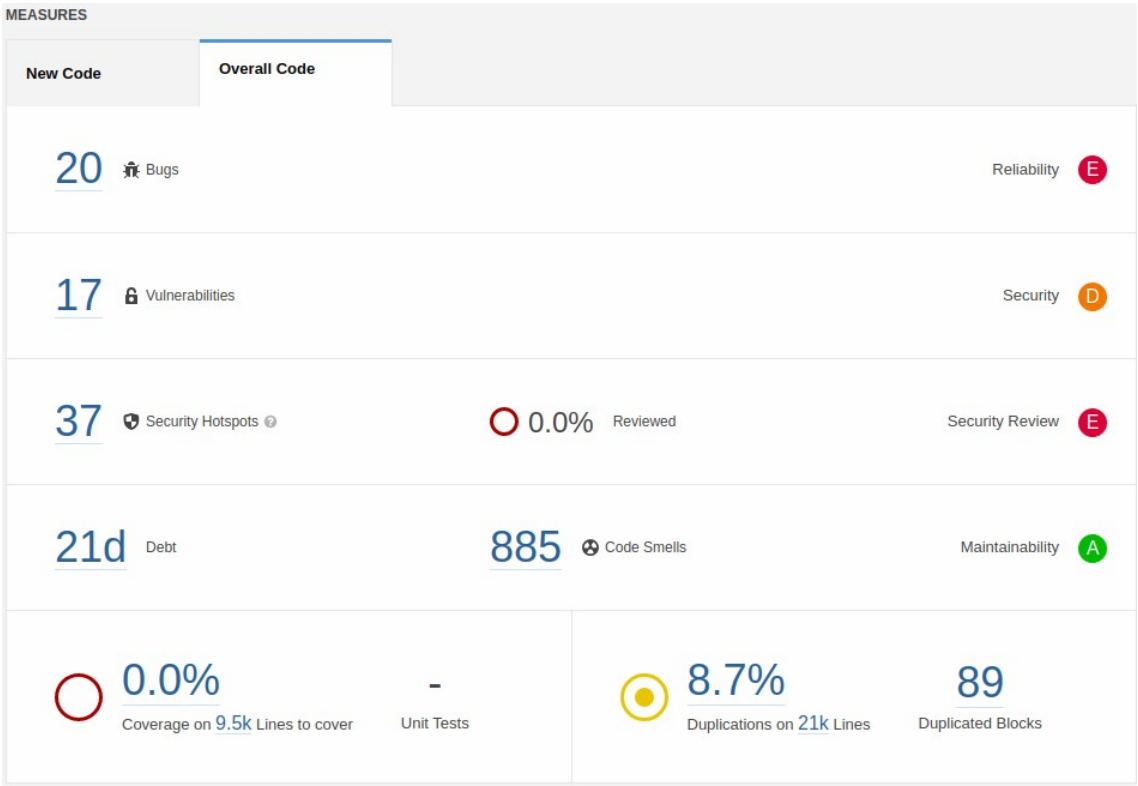

Figura 3: Análise estática de código

Nesta análise obtivemos os seguintes resultados:

- 20 bugs;
- 17 vulnerabilidades de código;
- 37 violações de segurança;
- 885 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 8,7% de duplicidade de código;

5.2 Banco de dados

Os Sistemas gerenciadores de banco de dados utilizado neste projeto foram MySQL para a base principal do PRF Móvel, Informix, PostgreSQL para a utilização de bases legadas e Redis

como uma camada de cache para as consultas das APIs. O banco de dados Sqlite foi utilizado de forma embarcada na aplicação PRF Móvel (mobile) para armazenamento local.

Por falta de acesso aos SGBDs citados acima, não foi possível extrair o modelo entidade relacional para esta análise.

5.3 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.

COMANDOS				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.12.1.jar	LOW	1	Highest	46
snakeyaml-1.25.jar	HIGH	1	Highest	28
tomcat-embed-core-9.0.31.jar	HIGH	6	Highest	39
spring-core-5.2.4.RELEASE.jar	MEDIUM	1	Highest	30
EVF				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
hibernate-core-5.4.12.Final.jar	MEDIUM	1	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
log4j-api-2.12.1.jar	LOW	1	Highest	46
snakeyaml-1.25.jar	HIGH	1	Highest	28
tomcat-embed-core-9.0.31.jar	HIGH	6	Highest	39
mysql-connector-java-5.1.46.jar	HIGH	5	Highest	38
spring-core-5.2.4.RELEASE.jar	MEDIUM	1	Highest	30
MAIN				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.12.1.jar	LOW	1	Highest	46
snakeyaml-1.25.jar	HIGH	1	Highest	28
tomcat-embed-core-9.0.31.jar	HIGH	6	Highest	39
spring-core-5.2.4.RELEASE.jar	MEDIUM	1	Highest	30
hibernate-core-5.4.12.Final.jar	MEDIUM	1	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
mysql-connector-java-5.1.46.jar	HIGH	5	Highest	38
postgresql-42.2.10.jar	HIGH	1	Highest	52
xstream-1.3.jar	CRITICAL	3	Highest	24
commons-email-1.3.1.jar	HIGH	2	Highest	38
google-oauth-client-1.30.1.jar	CRITICAL	1	Low	37
netty-transport-4.1.45.Final.jar	CRITICAL	1	Highest	31
grpc-netty-shaded-1.25.0.jar (shaded: io.netty:n	CRITICAL	3	Highest	9
SISCOM				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.12.1.jar	LOW	1	Highest	46
snakeyaml-1.25.jar	HIGH	1	Highest	28
tomcat-embed-core-9.0.31.jar	HIGH	6	Highest	39
spring-core-5.2.4.RELEASE.jar	MEDIUM	1	Highest	30
hibernate-core-5.4.12.Final.jar	MEDIUM	1	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
mysql-connector-java-5.1.46.jar	HIGH	5	Highest	38
xstream-1.2.2.jar	CRITICAL	3	Highest	24
commons-email-1.3.1.jar	HIGH	2	Highest	38

5.4 Estrutura do projeto

Os módulos que expõe o sistema móvel API’s possuem concepções similares e sua estrutura possui segmentação lógica entre os pacotes da solução.

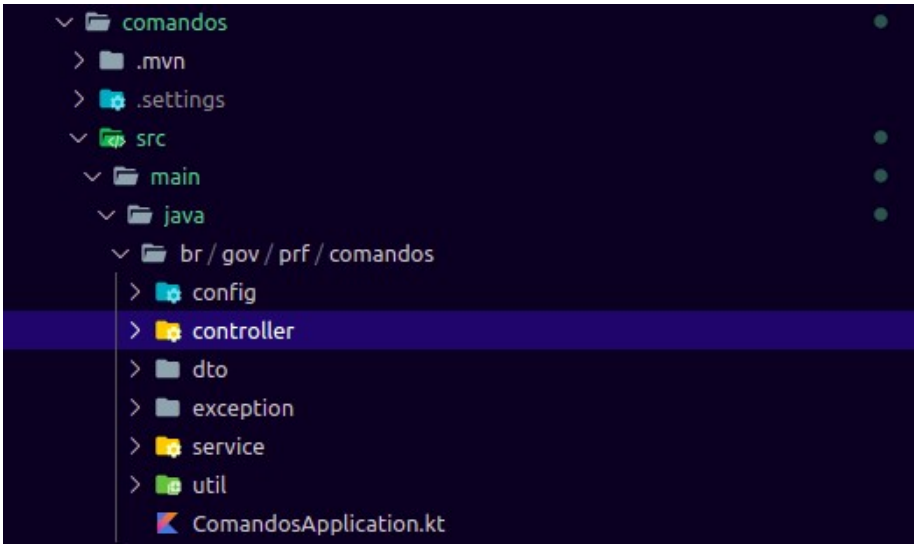

Figura 4: Módulo - Comandos

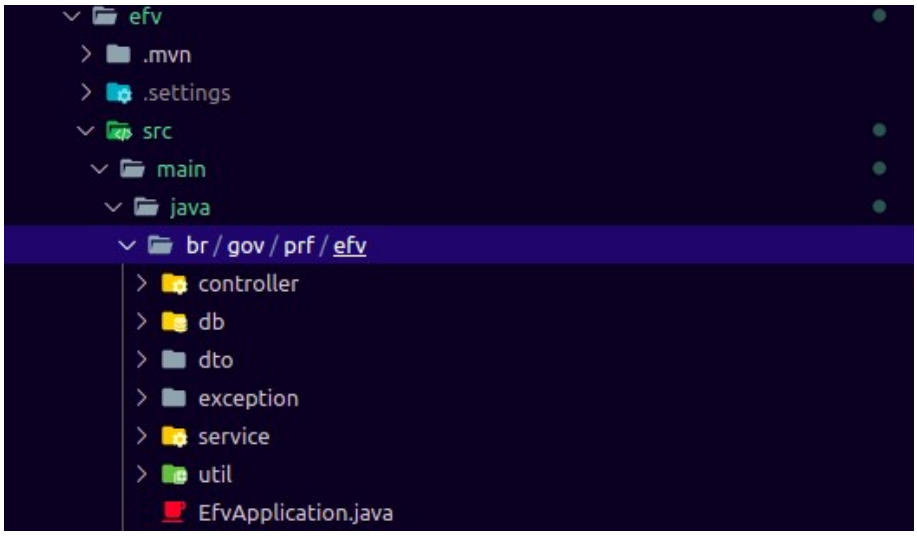

Figura 5: Módulo - EFV

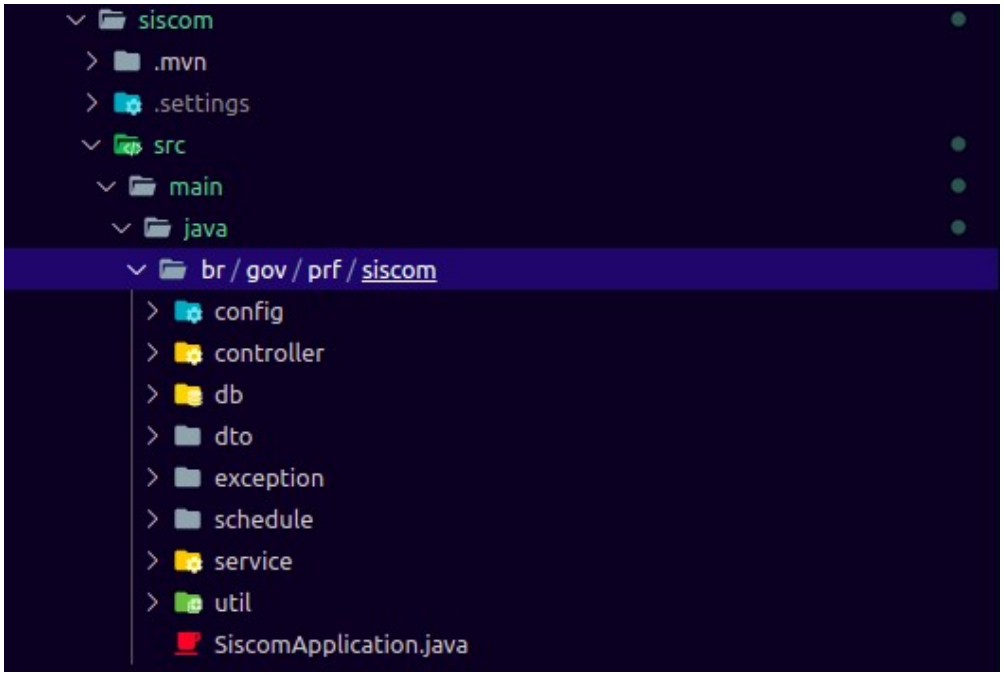

Figura 6: Módulo - Siscom

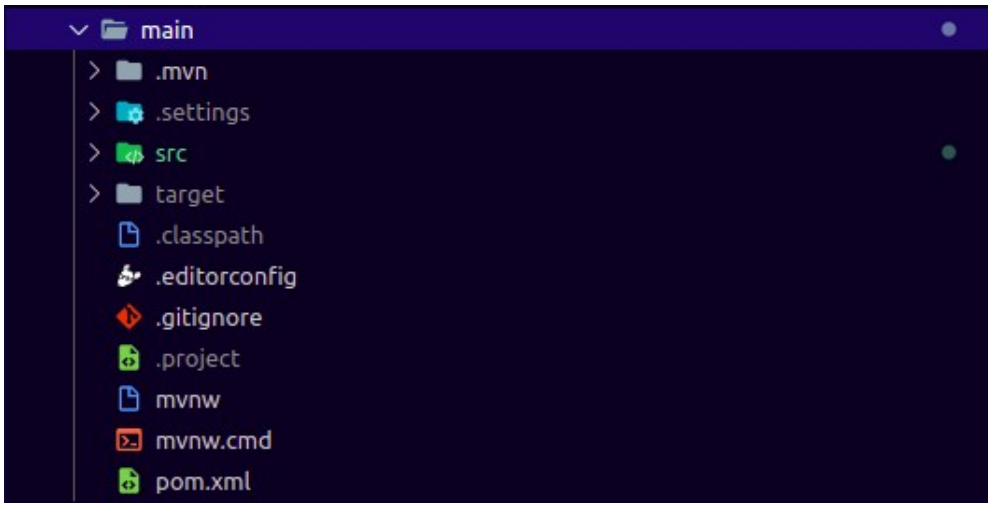

Figura 7: Módulo - Main

O módulo mobile da solução também possui boa estrutura organizacional, cada componente possui estrutura própria e segmentada dos demais.

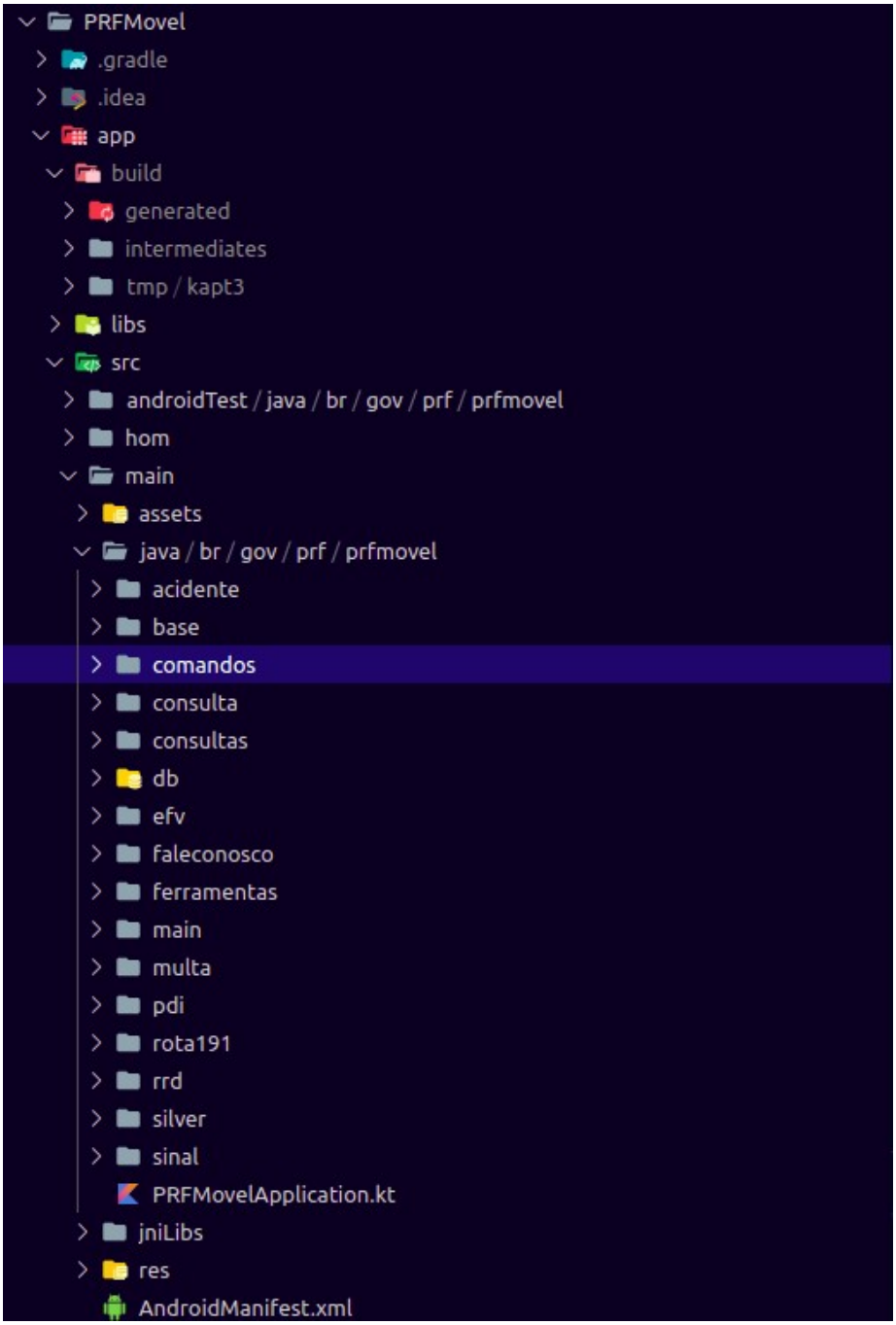

Figura 8: Módulo - Mobile

5.5 Manutenibilidade de código

Os relatórios apresentados pela ferramenta SonarQube demonstram poucos vícios adotados durante o processo de construção do software, embora tenhamos boa qualidade no código produzido, os pontos levantados nesta análise necessitam de revisão e atribuem certa vulnerabilidade e instabilidade de comportamento da solução.

A inexistência de cobertura de testes de unidade na camada de serviço da aplicação traz consigo a dificuldade no processo de refactoring e manutenção da aplicação, uma vez que há dificuldade em mensurar impactos durante o processo de manutenção corretiva/ adaptativa/evolutiva.

A aplicação apresenta boa estrutura arquitetural e padronização de código aliado a baixa complexidade ciclomática e a boa coesão, fatores estes que facilitam a manutenção do código.

5.6 Confiabilidade

Não há evidências que demonstre a existência de tratativas de controle transacional na aplicação, este tratamento segue as boas práticas de desenvolvimento de aplicações. A falta deste controle transacional impede a garantia das propriedades ACID do SGBD. Também não há uma modelagem de dados que seja consistente tendo em vista há inexistência de relacionamentos entre as tabelas.

A manutenção da consistência de dados é algo fortemente desejado, contudo esta não garante toda a confiabilidade da solução.

É importante ressaltar a necessidade de revisão de dependências apresentada nos resultados no relatório da ferramentas de análise de dependências.

5.7 Performance e estabilidade

Não foi analisado o funcionamento da aplicação para avaliar demais requisitos não funcionais. O alto grau de dependência das API’s que compõe a solução com os diversos serviços internos e externos a PRF conforme demonstrado no diagrama de componentes, aumentam a capacidade de instabilidade no funcionamento da solução, sejam com a utilização da camada mobile quanto das rotinas agendadas.

A utilização do banco de dados em memória Redis certamente aumenta a capacidade performática da solução, aumenta também o nível de resiliência quanto a necessidade de consulta nos serviços internos/externos a PRF.

A arquitetura baseada em serviços e o comportamento sem estado (stateless) das API’s Rest da aplicação, promovem 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. Para melhor promoção da escalabilidade horizontal, recomenda-se que sejam efetuadas as alterações sugeridas na sessão de Recomendações.

6 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, contudo, minimizado pela boa cobertura de testes de unidade.

Ajustar as dependências que trazem maior risco para a aplicação é altamente recomendável, recomenda-se que este trabalho deve ser feito de forma analítica e cautelosa afim de não prejudicar a estabilidade da ferramenta.

Recomenda-se também que seja instalado o agente da ferramenta de APM 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. Esta análise também possibilitará melhor utilização do banco de dados em memória, direcionando as consultas mais demandadas na aplicação para este banco.

Recomenda-se a remoção das funcionalidades que trabalham com agendamentos do core das API’s Rest. Esta prática reduz a capacidade de escalabilidade da solução e compromete o compartilhamento de recursos computacionais, uma vez que quando escalado horizontalmente um modulo, as rotinas promoverão concorrências em suas execuções.

Por sua vez, recomenda-se também que sejam removidos as

constantes estáticas que apontam para URL’s externas de dentro do código da aplicação para o arquivo de configuração do módulo ao qual seja pertinente. Esta prática evita necessidade de recompilação de código quando houver necessidade de mudança das mesmas.

```
@Autowired
public SuporteBrService(FirestoreService firestoreService,
                        RestTemplateBuilder restTemplateBuilder) {
    this.firestoreService = firestoreService;
    restTemplate = restTemplateBuilder
        .rootUri("https://api-siga-caminhoneiro.prf.gov.br/api/v1/")
        .setConnectTimeout(Duration.ofMillis(30000))
        .setReadTimeout(Duration.ofMillis(30000))
        .build();
}
```

Figura 9: Módulo Main - SuporteBrService.java

Ajustar o fluxo sequencial da solução ajudará a manter a coesão de cada camada quanto ao seu propósito. Os fluxos demonstrados na Figura 2 deste documento representam dois cenários, sendo que o segundo representa o cenário não desejado sendo este encontrado em menor escala nos módulos que compõe o projeto de API’s.

Para fins de manutenção centralizada, recomenda-se que as funcionalidades criadas para o projeto Silver (legado) sejam incorporadas ao core do sistema Silver em ambiente produtivo. Isso evitará a descentralização das manutenções das soluções.