Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Elaboração de testes para o Sistema de Gerenciamento e Registro de Atividades (SisGera)

Bryan Lucas Tavares Pinto

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:

Euler Horta Marinho

COORIENTAÇÃO:

Silvandro Sergio Martins Oliveira

Novembro, 2022
João Monlevade–MG

Bryan Lucas Tavares Pinto

Elaboração de testes para o Sistema de Gerenciamento e Registro de Atividades (SisGera)

Orientador: Euler Horta Marinho

Coorientador: Silvandro Sergio Martins Oliveira

Monografia apresentada ao curso de Engenharia da Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Novembro de 2022

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

P659e Pinto, Bryan Lucas Tavares.
Elaboração de testes para o Sistema de Gerenciamento e Registro de Atividades (SisGera). [manuscrito] / Bryan Lucas Tavares Pinto. - 2022.
48 f.: il.: color., gráf., tab..

Orientador: Prof. Me. Euler Marinho Marinho.
Coorientador: Esp. Silvandro Sergio Martins Oliveira.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia de Computação .

1. Engenharia de software. 2. Segurança de sistemas. 3. Software - Testes. I. Marinho, Euler Marinho. II. Oliveira, Silvandro Sergio Martins. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 004.415.5

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Bryan Lucas Tavares Pinto

Elaboração de testes para o Sistema de Gerenciamento e Registro de Atividades (SisGera)

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação

Aprovada em 03 de novembro de 2022

Membros da banca

Mestre - Euler Horta Marinho - Orientador (Universidade Federal de Ouro Preto)
Silvandro Sergio Martins Oliveira - Coorientador - Clube FII
Doutor - Diego Zuquim Guimarães Garcia - (Universidade Federal de Ouro Preto)
Doutora - Kattiana Fernandes Constantino - (DELCOM/CEFET-MG)

Euler Horta Marinho, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 29/11/2022

Documento assinado eletronicamente por **Euler Horta Marinho**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 29/11/2022, às 12:38, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0433883** e o código CRC **4C4C4204**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Este trabalho é dedicado a minha mãe e meu pai, por sempre acreditarem em mim e me apoiarem, ao meu irmão e a todos os meus amigos.

Agradecimentos

Agradeço primeiramente aos meus pais Edigar e Mary que sempre fizeram de tudo para que eu pudesse realizar meus sonhos e que sempre me forçaram e motivaram a continuar seguindo em frente. Ao meu irmão Iago que sempre esteve presente e me auxiliou quando precisei.

Aos meus amigos Victor Duarte, Victor Hugo, Hellrison, Lucas Rocha, Lucas Lírio, Lucas Pereira, Natália, Thuane, Débora, Pedro, Tiezi, Caio, Rodrigo, Natan e muitos outros que fizeram parte da minha vida ao longo da graduação.

Aos meus amigos da rede social Discord que me ajudaram a passar por situações difíceis e me apoiar, sem eles nada disso seria possível.

Ao Euler e Silvandro, pelo apoio no desenvolvimento deste projeto, por acreditarem em mim e sempre estarem disponíveis.

Aos professores e professoras que tive durante a graduação, pelos conhecimentos e ensinamentos passados.

E a todos que diretamente e indiretamente, fizeram parte da minha jornada nessa graduação e que sempre me apoiaram e não me deixaram desistir, muito obrigado.

"Science is more than a body of knowledge; it is a way of thinking."

— Carl Sagan (1934 – 1996),
in: The Demon-Haunted World: Science as a Candle in the Dark.

Resumo

O desenvolvimento tecnológico possibilitou as organizações o desenvolvimento de sistemas que ajudam a auxiliar na execução de suas tarefas e deixar todo o processo mais dinâmico e eficiente. Contudo, esses sistemas precisam garantir um bom desempenho e a segurança das informações presentes. Neste contexto, este trabalho tem como finalidade o desenvolvimento e execução de testes de software de maneira sistemática no SisGera. Durante o desenvolvimento deste trabalho, foram selecionados e analisados os principais testes a serem utilizados, além de uma forma de garantir a preservação dos dados para posteriores manutenções no sistema. Com os resultados obtidos, é possível melhorar o SisGera e garantir que ele tenha mais segurança e desempenho em seu funcionamento.

Palavras-chaves: teste de software. teste funcional. teste de segurança. teste de carga.

Abstract

Technological development has made it possible for organizations to develop systems that help to assist in the execution of their tasks and make the whole process more dynamic and efficient. However, these systems need to ensure good performance and security of the present information. In this context, this work aims to the development and execution of software tests in a systematic way in SisGera. During the development of this work, the main tests to be used were selected and analyzed, as well as a way to ensure data preservation for subsequent system maintenance. With the results obtained, it is possible to improve the SisGera and ensure that it has more security and performance in it's operation.

Key-words: software testing. functional testing. security testing. load testing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SisGera Sistema de Gerenciamento e Registro de Atividades

VVT Verificação, Validação e Teste

AWP Aplicação Web Progressiva

OWASP Open Web Application Security Project

ZAP Zed Attack Proxy

DoS *Denial Of Service*

SQL *Structured Query Language*

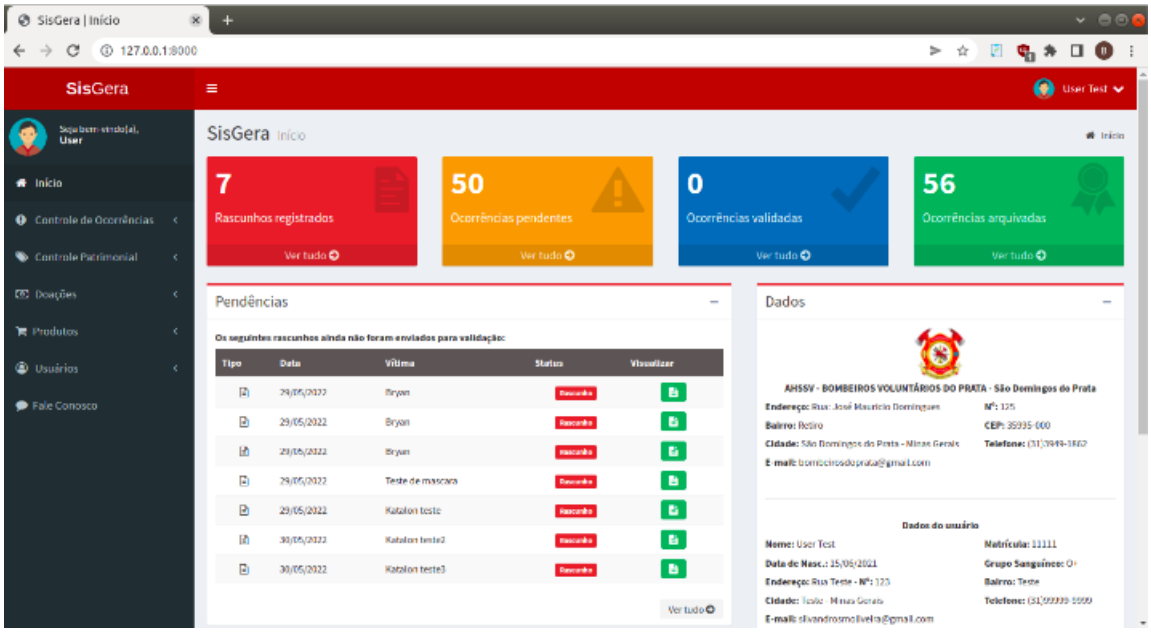
Sumário

1 Introdução

A Associação de Bombeiros Voluntários de São Domingos do Prata é uma organização que de forma voluntária presta serviços de resgate. Dentre esses serviços, é possível destacar situações como atendimento a vítimas com trauma, busca e salvamento, entre outros.

Conforme a associação cresceu, ela realizava vários processos de maneira manual. Nesse contexto, a aplicação SisGera foi criada. O SisGera é uma Aplicação Web que foi desenvolvida ao longo dos trabalhos de (ARANTES, 2018) e (OLIVEIRA, 2018), com o objetivo de suprir as necessidades das organizações de Bombeiros Voluntários de São Domingos do Prata e Barão dos Cocais. No processo antigo, a organização utiliza de meios manuais para registrar as ocorrências, isso implicava em erros e processo mais lento de preenchimento. Com o sistema desenvolvido, o processo passaria a ser mais rápido e eficiente, não teria papeis e registros físicos. Na imagem a seguir, mostra como ficou a tela inicial do sistema desenvolvido.

Figura 1 – Página inicial do SisGera



Fonte: Elaborada pelo Autor(2022)

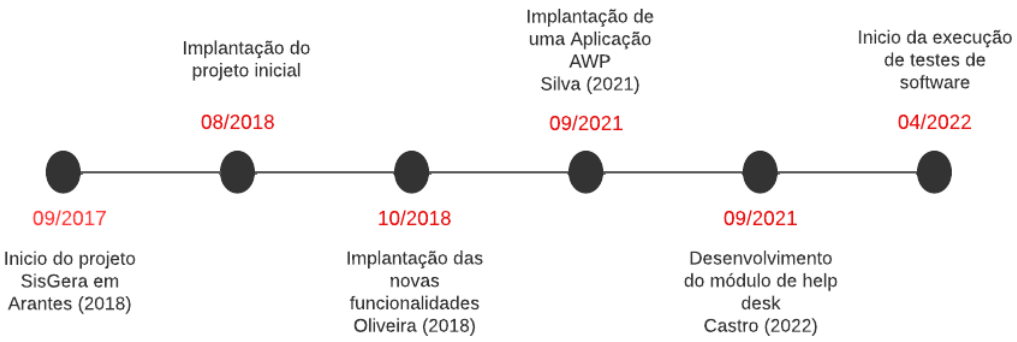
O trabalho desenvolvido por (ARANTES, 2018) foi o que deu origem ao sistema. Nele, foi elaborada a etapa de realização do registro das ocorrências. Até então, os registros eram realizados de forma manual, fazendo com que o processo fosse mais lento e passivo de erro humano. Dessa forma, seu trabalho foi desenvolvido para suprir essa necessidade e tornar todo registro mais dinâmico. Logo em seguida, (ARANTES, 2018) foi responsável pela melhoria do sistema e criou três modelos de boletins para diferentes ocasiões e melhor separação.

O próximo trabalho foi o de (OLIVEIRA, 2018), que teve como objetivo adicionar novas funcionalidades. As funcionalidades foram focadas na parte administrativa da organização, incluindo o controle de materiais do estoque e as doações recebidas.

Na sequência, (SILVA, 2021) foi responsável por criar uma Aplicação Web Progressiva (AWP), como o objetivo de possibilitar o registro de ocorrências de forma *offline*. Até então, o sistema era dependente da *Internet* para seu funcionamento. Então, essa melhoria deixaria o sistema funcionar com a ausência de rede e, posteriormente, a sincronização seria realizada. Logo após, (CASTRO, 2022) foi o responsável pela última implementação de funcionalidade até então, que seria o sistema de *help desk* que auxilia na resolução de problemas do usuário.

Atualmente, o sistema está em funcionamento em 5 cidades, sendo elas, Barão de Cocais, São Domingos do Prata, Nova Era, Santa Bárbara do Leste e Cláudio. Contudo, como descrito anteriormente, todos os trabalhos envolveram melhoria ou desenvolvimento do sistema. Contudo, não houve a execução de testes na aplicação como um todo. A linha do tempo da ferramenta até o momento e mostrada segundo a imagem a seguir.

Figura 2 – Linha do tempo SisGera



Fonte: Elaborada pelo Autor (2022)

1.1 Problema

O SisGera foi desenvolvido de forma voluntária inicialmente por (ARANTES, 2018) e, logo em seguida, aprimorado e desenvolvido por (OLIVEIRA, 2018), (SILVA, 2021). O corpo de bombeiros voluntários executava as tarefas de forma manuscrita e isso trazia risco de erros ao executar o preenchimento. Sendo assim, esse problema foi solucionado pela implantação da ferramenta que otimizou a forma como eram tratadas as tarefas na organização.

Contudo, durante o desenvolvimento do SisGera e suas posteriores melhorias, não houve uma forma precisa de validar a ferramenta. Foram verificados e tratados problemas superficiais da ferramenta. Entretanto, para o desenvolvimento de um software, é preciso executar testes de forma sistemática.

O teste de software é capaz de revelar diversos tipos de falhas. Porém, a ausência de testes elaborados de maneira criteriosa pode ocasionar a descoberta de problemas graves pelos usuários. Dessa forma, torna-se necessária a validação de todas as funcionalidades já implementadas do SisGera.

1.2 Objetivos

Nesta seção, serão descritos o objetivo geral do trabalho e os objetivos específicos pelos quais serão apresentadas as práticas desempenhadas para atingir o objetivo geral do trabalho.

1.2.1 Objetivo Geral

O presente trabalho tem como objetivo dar continuidade ao projeto SisGera, criado inicialmente no trabalho de (ARANTES, 2018) e posteriormente aprimorado nos trabalhos de (OLIVEIRA, 2018) e (SILVA, 2021), no qual serão executados três tipos de testes de software de forma sistemática englobando todo o sistema e cobrindo os pontos mais importantes. Desde a criação, nunca houve a aplicação exclusiva de testes de software na ferramenta, então a expectativa e identificar a catalogar os problemas para soluções futuras. Vale ressaltar, que os testes não passaram pelas atualizações do trabalho de (SILVA, 2021), pois no momento da inclusão de sua atualização no sistema, os testes já estavam no processo de conclusão.

1.2.2 Objetivos Específicos

Este trabalho tem como objetivos específicos os seguintes itens:

- Revisar a literatura acerca de Testes de Software, Qualidade de Software, Engenharia de Software e similares.
- Planejar e executar testes.
- Gerar relatórios dos testes e apontar possíveis soluções para as falhas identificadas.

1.3 Justificativa

Inicialmente, o SisGera foi utilizado pelas corporações de Barão de Cocais e São Domingos do Prata e, posteriormente, por corporações de mais três cidades, sendo elas, Santa Barbara do Leste/MG, Cláudio/MG e Nova Era/MG. Contudo, eles executam esse serviço de forma voluntária e não recebem qualquer tipo de remuneração. Isso implica que não possuem recursos para contratar um profissional especializado para a manutenção e avaliação de qualidade. Como tudo que foi realizado em cima do SisGera, foi de maneira voluntária e não houve a realização de testes de maneira sistemática em cima de tudo que foi criado, surgiu a oportunidade de aplicação de testes para apontar possíveis pontos de melhoria para o sistema e colaborar com a organização na manutenção da qualidade do software.

Com o SisGera crescendo tanto como aplicação, como em número de corporações usuárias, é preciso testar e avaliar o sistema, já que ela tem um grande impacto nas operações dessas corporações e, por consequência, uma relevância social. Caso o sistema não passe por essa etapa, pode colocar em risco a organização, os dados e até as pessoas que utilizam essa ferramenta por causa de suas informações armazenadas.

Desta forma, este trabalho visa contribuir para o aprimoramento do SisGera, possibilitando o incremento da qualidade do sistema.

1.4 Organização do trabalho

Este trabalho está organizado de tal forma que o Capítulo 2 apresenta a revisão bibliográfica, onde são mencionados temas relevantes para desenvolvimento do estudo. O Capítulo 3 descreve as ferramentas e abordagens utilizadas na execução dos testes de software e os resultados obtidos. O Capítulo 4 apresenta as considerações finais e propostas para trabalhos futuros.

2 Revisão bibliográfica

Este capítulo apresenta a revisão bibliográfica dos principais conceitos utilizados neste trabalho.

2.1 Teste de Software

O avanço tecnológico possibilitou a sociedade o progresso em diversas áreas do conhecimento, dentre elas, o desenvolvimento de software. Apesar dos benefícios e facilidades gerados por esse desenvolvimento, houve o surgimento de novas demandas devido ao avanço da complexidade dos softwares, a qualidade e confiabilidade do software a partir de testes.

Segundo (HIEATT; MEE, 2002), a difusão da Internet produziu uma demanda mais rigorosa pelas aplicações Web. Contudo, a pressão do mercado por tempos de lançamento curtos, leva aos testes das Aplicações Web ser negligenciado pelos desenvolvedores, por ser demorado e sem retorno imediato. Segundo (RIOS; MOREIRA, 2006), testes mal aplicados podem significar um caminho livre para problemas graves como vulnerabilidades, fraudes e incorreções que geram prejuízos e descontentamentos por parte dos usuários.

Segundo (SOMMERVILLE, 2015), os processos de verificação, validação e testes (VV&T) tem como objetivo avaliar se o software satisfaz o que era esperado, bem como o que está sendo pago para o mesmo desempenhar, juntamente com a correção de falhas/*bugs*. Apesar de verificação e validação serem recorrentemente confundidos, não são a mesma coisa. Para melhor entender a diferença, duas perguntas podem ser feitas conforme descreve (BOEHM, 1984):

- Validação: estamos construindo o produto certo?
- Verificação: estamos construindo o produto da forma certa?

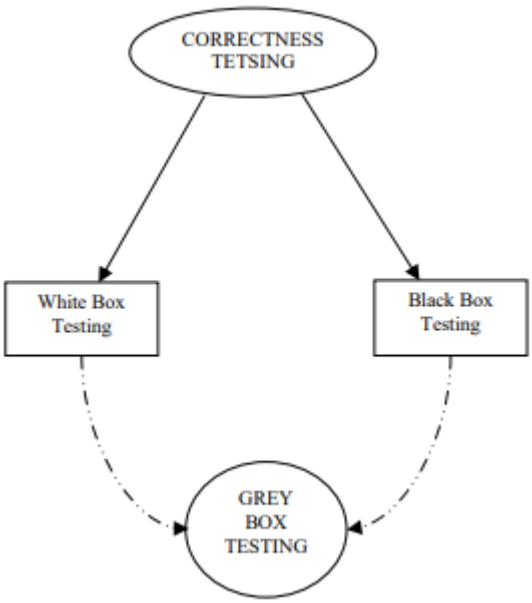
Para a execução de teste de software, é preciso respeitar todo um processo para o desenvolvimento dessa atividade, que consiste em algumas etapas: planejamento, análise e modelagem, execução e avaliação. Conforme (MORAIS; SOARES; ZANIN, 2020):

- Planejamento: É a etapa inicial do processo de teste, ela tem por finalidade especificar e definir os objetivos dos testes. Etapa que também ocorre a criação de um documento que organiza todas as pessoas envolvidas, bem como tipos de teste, ferramentas, riscos, entre outros.

- Análise e modelagem: O principal objetivo dessa etapa é a criação de casos de teste, scripts de teste e verificação dos ambientes de teste.
- Execução: Como o próprio nome já diz, é a etapa onde os teste definido nas etapas anteriores serão executados. As execuções seguem por etapas, como tipo de teste, funcionalidade ou prioridade.
- Avaliação: Etapa onde são coletado as saídas de execuções dos teste e avaliar os resultados obtidos.

Dentro da área de teste, existem as mais diversas técnicas de teste de software. Segundo (JORGENSEN, 2013), as técnicas de teste podem ser classificadas como testes caixa-branca e testes caixa-preta. Sendo o primeiro referente a teste estrutural baseado em informações do código fonte, enquanto o segundo é referente a técnicas de testes funcionais baseado em especificações da documentação, e ambas técnicas são complementares entre si. Alguns autores defendem a existência da técnica de teste caixa cinza, definido por (KHAN, 2010) como a combinação dos testes caixa-branca e caixa-preta, quando são testadas partes do software com base nas especificações e tendo conhecimento do código.

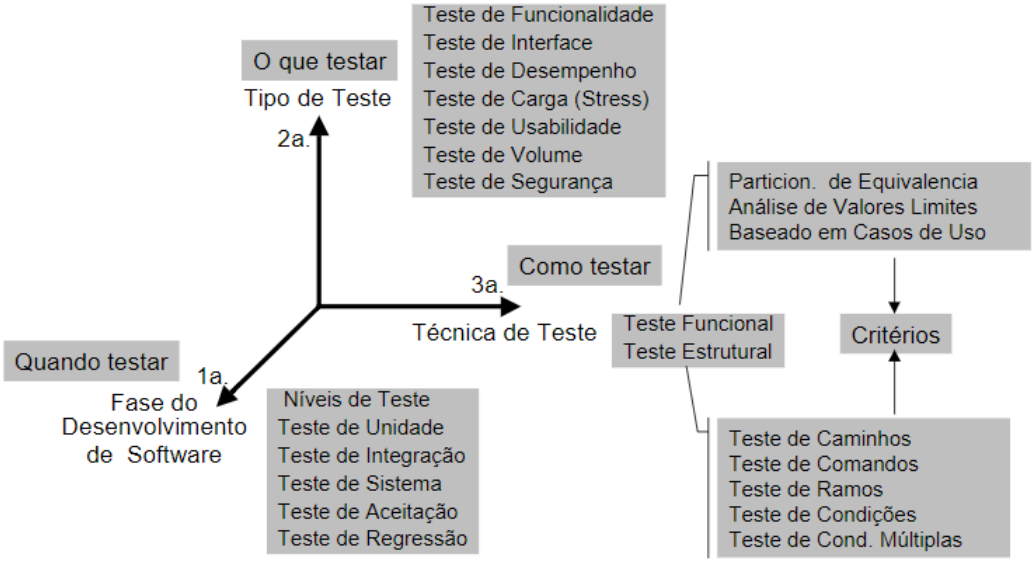
Figura 3 – Representação dos tipos de teste



Fonte: (KHAN, 2010) ACESSO DIA 24/06 10:41

Segundo (CRESPO et al., 2004), as estratégias de teste podem ser fundamentadas em diferentes dimensões. A figura 3 representa a relação entre os níveis, as técnicas, os tipos e os critérios que podem ser adotados durante a estratégia de teste de software.

Figura 4 – Estratégia de teste de software



Fonte: (CRESPO et al., 2004) ACESSO DIA 24/06 11:00

Segundo (JORGENSEN, 2013), os testes de software podem ser aplicados em vários estágios do ciclo de vida de um software. Existem diferentes tipos de testes, com técnicas diversas, como exemplifica a figura a seguir:

Tabela 1 – Relação de tipo por técnica de teste

Tipo de teste	**Técnica**
Unidade	Teste Caixa-Branca
Integração	Teste Caixa-Preta Teste Caixa-Branca
Funcional	Teste Caixa-Preta
Sistema	Teste Caixa-Preta
Aceitação	Teste Caixa-Preta
Regressão	Teste Caixa-Preta Teste Caixa-Branca

Fonte: Elaborado pelo autor (2022)

2.2 Teste Funcional

Como foi apresentado na tabela da seção anterior, o teste funcional se trata de um teste de caixa preta. Segundo (BARBOSA et al., 2000), se trata de um teste onde é apenas conhecido o lado externo da aplicação, desta forma, os dados fornecidos na entrada

serão responsáveis por produzir uma saída. Assim, o teste funcional se preocupa com as funções do sistema, ignorando os detalhes de implementação por trás do software.

Segundo (PRESSMAN; MAXIM, 2021), o teste funcional pode ser dividido em dois principais passos, sendo eles, identificação das funções que o software é capaz de realizar e a criação de teste capaz de verificar se essas funções estão sendo realizadas. Além disso, o teste funcional pode ser separado em alguns critérios, sendo eles:

- Particionamento em classes de equivalência: se baseando nas condições de entrada de dados, é dividido em entrada de dados válidos ou inválidos. Após isso, é selecionado pequenos grupos de casos de teste para representar toda a classe, e para os casos inválidos, um teste distinto.
- Análise do Valor Limite: considerado um complemento ao item anterior, os casos de teste são escolhidos nas fronteiras das classes para se pegar o maior número de erros possíveis.
- Grafo de Causa-Efeito: este critério explora a combinação de condições de entrada do sistema. Aqui, as combinações de entradas são combinadas com possíveis ações que trará um efeito no programa e por fim é executado. Então, ocorre a criação de uma tabela de decisão a partir dos casos de teste.

Segundo (DELAMARO; JINO; MALDONADO, 2013), uma das vantagens da utilização desses critérios é que requer somente a especificação do produto para o teste. Assim, é possível aplicar a qualquer sistema de qual natureza ele seja. Contudo, apresentam algumas limitações, como por exemplo a existência de muitas possibilidades de saídas e casos. Apesar disso, é preciso entender que as técnicas sejam entendidas como complementares para que o software seja explorado a partir de diversos pontos de vista diferente. Apesar disso, entender que não podem assegurar que partes críticas ou essenciais tenham sido contempladas na execução.

Dentro das técnicas presente no teste funcional, existe o chamado teste exploratório. Segundo (ABREU; MARTINO; SCHIAVONI, 2016), é um tipo de teste onde não se segue um roteiro rígido, não se elaboram casos de teste com antecedência e são baseados em pensamento estruturado e exploração livre. Diferente de outros tipos de teste, o exploratório diz respeito a uma investigação no sistema e por meio disso ocorre a avalização do software.

Segundo (KANER; BACH; PETTICHORD, 2008), o teste exploratório pode ser utilizado em situações especificas, dentre elas:

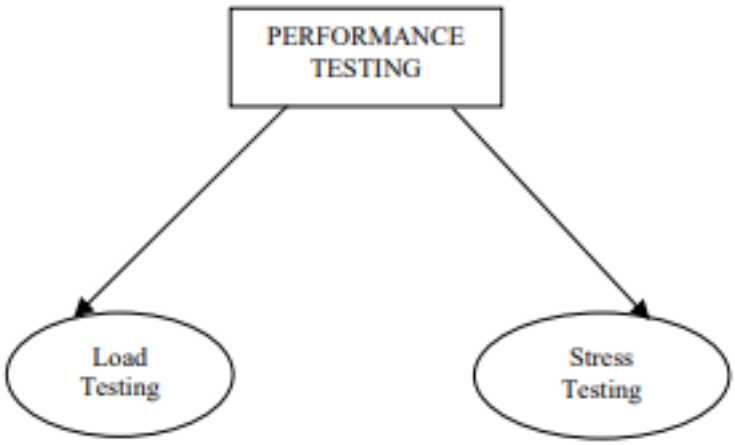
- Entregar um feedback rápido;
- Aprender sobre o produto rapidamente;

- Encontrar os problemas mais importantes e visíveis num curto espaço de tempo;
- Investigar a situação de risco, e avaliar a necessidade de aplicar testes com script sobre aquele determinado problema;
- Falta de documentação.

2.3 Teste de Carga

Segundo (KHAN, 2010), o teste de carga se encontra dentro da categoria de teste de performance, que envolve todo o ciclo de vida do software e envolve estratégias como por exemplo, plano de execução, design, execução, análise e relatórios. Seu objetivo é avaliar o desempenho de um sistema que inclui o uso de recursos, rendimento, estímulo e tempo de resposta. Em Aplicações Web, sua importância é manter uma latência baixa de um site com um alto rendimento. Os testes de performance podem ser divididos em dois tipos: o teste de carga e o teste de estresse.

Figura 5 – Tipos de teste de performance



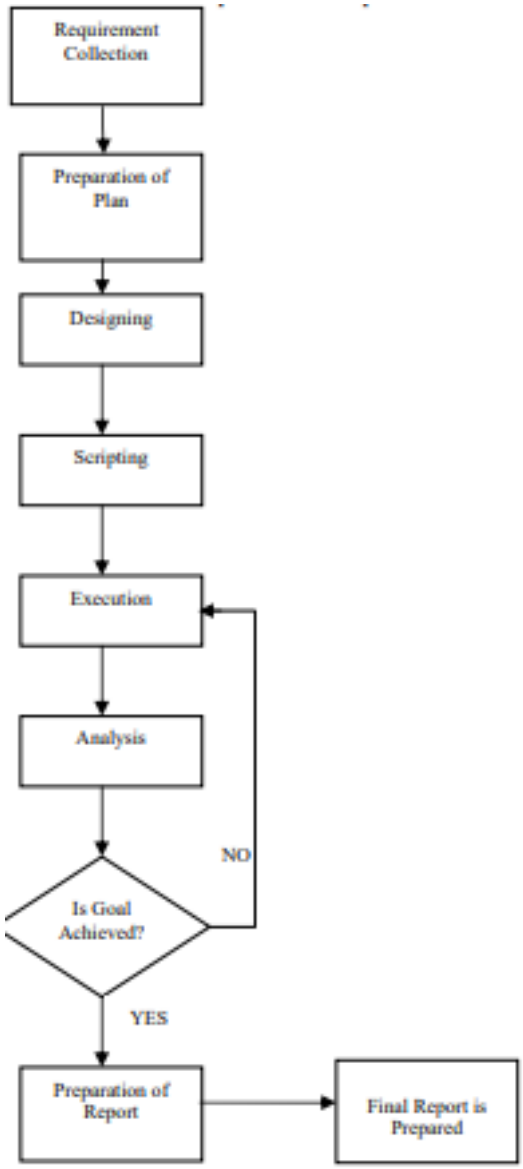
Fonte: (KHAN, 2010) ACESSO DIA 24/06 11:00

De acordo ainda com (KHAN, 2010), os testes de performance podem ser divididos em diferentes fases:

- Fase 1 - Estudo
- Fase 2 - Plano de teste
- Fase 3 - Design do teste
- Fase 4 - Script do teste
- Fase 5 - Execução do teste

- Fase 6 - Analise de resultado
- Fase 7 - Preparação do relatório

Figura 6 – Processo de teste de performance

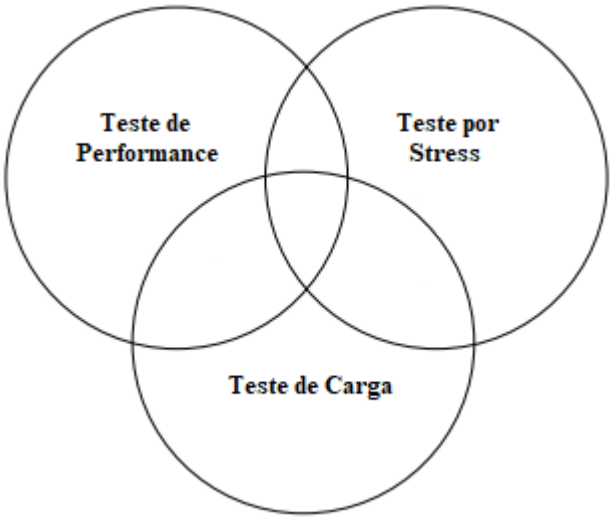


Fonte:(KHAN, 2010)5 ACESSO DIA 24/06 11:00

Segundo (KHAN, 2010), apesar de os dois tipos de teste seguirem o mesmo tipo de estrutura, eles apresentam objetivos diferentes apesar de serem fundamentalmente parecidos. O teste de carga pretende descobrir a capacidade do sistema, isso é, sua capacidade de processamento e acessos. Enquanto o teste de estresse seria descobrir em que ponto o sistema começa a não funcionar mais, ou seja, verificar a robustez do software.

Segundo (JIANG; HASSAN, 2015), tanto o teste de carga, quanto o teste de estresse têm suas próprias finalidades. Contudo, os dois testes são complementares entre si, uma vez que, a aplicação dos dois avalia um cenário totalmente diferente de quando executados separadamente. Este cenário está ligado ao balanço perfeito do sistema, onde existe a robustez por aguentar inúmeras requisições e um ótimo tempo de resposta.

Figura 7 – Relação dos testes de performance



Fonte: Adaptado pelo Autor

2.4 Teste de Segurança

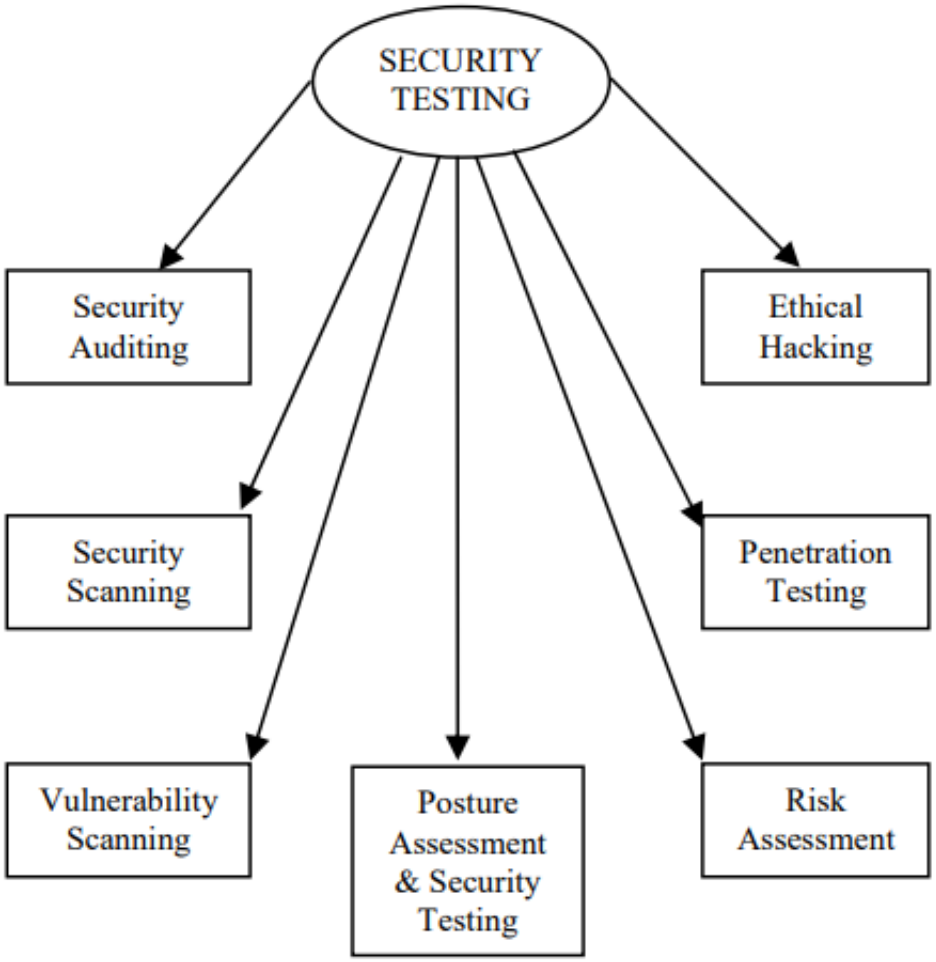
Segundo (SCHIEFERDECKER; GROSSMANN; SCHNEIDER, 2012), o teste de segurança é um recurso para verificar as funcionalidades, eficiência e avaliar as funcionalidades de segurança e/ou segurança do sistema, com o objetivo de garantir confiabilidade e integridade.

De acordo com (FELDERER et al., 2016), o teste de segurança é utilizado para identificar as propriedades de segurança especificadas ou pretendidas. Este teste mostra se o que se encontra nos requisitos está acontecendo ou tenta identificar vulnerabilidades. Para isso, entra com dados maliciosos e outros recursos com o objetivo de explorar todas as possíveis vulnerabilidades do sistema.

De acordo ainda com o autor, o requisitos podem ser positivos e funcional, isto é, a funcionalidade de segurança esperada de um mecanismo de segurança é explicito. Ou pode ser negativa e não funcional, especifica o que o sistema não deve fazer. Essas duas classificações influenciam diretamente na hora de executar os testes.

Segundo (KHAN, 2010), existem diversos tipos de teste de segurança como mostrado na imagem a seguir.

Figura 8 – Tipos de Teste de Segurança



Fonte: (KHAN, 2010)

- Verificação de Segurança: inspeção direta do sistema operacional que o software é desenvolvido. O objetivo é descobrir falhas operacionais e de rede.
- Verificação de Vulnerabilidades: descobrir todas as vulnerabilidades existentes.
- Avaliação de Riscos: avaliar todos os riscos que envolvem o sistema na ocorrência de algum problema.
- Avaliação de postura e teste de segurança: avaliar o contexto de segurança e combinar recursos de varredura e avaliação de riscos.
- Teste de Penetração: objetivo de descobrir as falhas de um sistema, feita por um testador que entra a força no sistema por meio das brechas que o sistema apresenta.
- Hacking Ético: envolve interromper a entrada forçada de qualquer elemento de um sistema que está sob um teste de segurança.

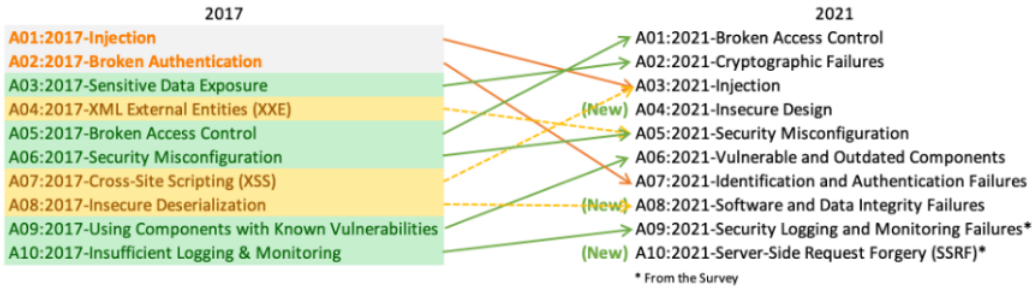
Segundo (PRESSMAN; MAXIM, 2021), dentro do contexto das aplicações móveis, o teste de segurança verifica se não compromete os requisitos de privacidade ou segurança

de seus usuários. Enquanto em aplicações *Desktop* e Web, é garantir que não existam nenhum acesso indevido e falhas que prejudiquem o usuário. Apesar de serem parecidos, seus objetivos diferem, mas no final o objetivo real é garantir a integridade do sistema e seus usuários.

Em meio a tantos tipos de vulnerabilidades é preciso saber como planejar os testes. Segundo (MORENO, 2019), a Open Web Application Security Project (OWASP) é uma organização sem fins lucrativos com o objetivo de melhorar a segurança de software, e nela existe a OWASP Top 10, que foca exclusivamente em mostrar as 10 falhas de Aplicações Web mais perigosas dos últimos anos e também sobre as lista de ferramentas para auxiliar a correção das mesmas. E por esse motivo que é classificada como referência mundial de guia de descrição de vulnerabilidades.

Atualmente eles disponibilizam a tabela mostrando a mudança nas vulnerabilidades mais perigosas de 2017 em relação a 2021. Como estamos em 2022 e o ano ainda não chegou ao seu fim, só existe a referência do ano anterior. A imagem a seguir ilustra a relação das vulnerabilidades.

Figura 9 – Relação de Vulnerabilidades de 2017 a 2021



Fonte: https://owasp.org/www-project-top-ten/

Como descrito anteriormente, a figura 9 demonstra a comparação entre 2017 e 2021. Essa diferença de posições nas imagem se diz respeito a evolução das vulnerabilidades. Enquanto em 2017 a listagem era uma, em 2021 ela passou a ser outra. Assim como as técnicas de segurança evoluem todos os anos, as vulnerabilidades também. Sendo assim, essas mudanças de posições na tabela sempre podem ocorrer.

3 Desenvolvimento

Este capítulo descreve o processo e as decisões tomadas na execução dos testes no sistema. A seção 3.1 abordas as escolhas de ferramentas tecnológicas para o trabalho. A seção 3.2 apresenta o processo de desenvolvimento dos testes.

3.1 Escolhas tecnológicas

Esta seção apresenta as tecnologias escolhidas para o desenvolvimento dos testes no projeto SisGera.

3.1.1 Katalon Studio

Geralmente para testes funcionais do tipo exploratório, não é preciso gerar *script*. Pois, é uma metodologia que permite conhecer o software sem precisar de documentação, e a execução desse tipo de teste segue os seguintes passos:

- Entrar com um conjunto de dados.
- Projetar qual deve ser a saída esperada.
- Verificar qual foi a saída.

Como descrito nos passos anteriores, caso a saída dos dados seja igual a esperada sem nenhum problema no programa, o teste foi bem sucedido. Caso contrário, ocorreu a descoberta de um erro. Testes do tipo exploratório não precisa de códigos, justamente porque são simples de executar. Além disso, permite ao testador conhecer a ferramenta sem ter a documentação em mãos, que foi o caso do SisGera.

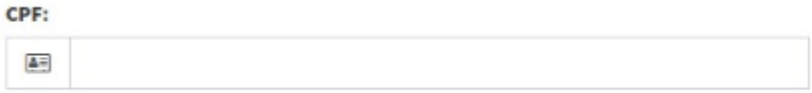
Contudo, foi optado a geração dos *scripts* para registrar o caminho e as entradas até o problema observado em questão, pois vão ser de suma importância futuramente para o rastreio dos problemas apontado. Então é necessário a presença de um software capaz de registrar os passos do testes e, posteriormente, permitir a localização para a execução das melhorias. O Katalon Studio foi selecionado. Ele é uma ferramenta de software para a automatização de testes *open source* criada em 2015. Dentro do Katalon, existe uma funcionalidade de *recorder* onde é possível registrar todas as ações que o testador vai tomando, ao longo do tempo da execução.

No processo de escolha, foram consideradas a Katalon e a Selenium IDE ambas com o mesmo objetivo. Contudo, o diferencial para a seleção do Katalon foi graças a

sua simplicidade de utilização, bem como a capacidade de realizar a captura de campos tratados, pois, dentro do SisGera existem vários campos com máscara. Essa máscara faz com que o campo tenha um tipo especifico de entrada e um tamanho especifico, por exemplo, o CPF é campo que aceita apenas números e obrigatoriamente só aceita a entrada se tiver os 11 dígitos preenchidos.

Na criação do *script* na Selenium, esses campos não conseguia ser capturados, o que gerava *scripts* defeituosos já que esses lugares ficava vazio. Diferente do Katalon, que permitiu a captura e configuração desse tipo de campo. Então, o Katalon se sobressaiu com sua facilidade de permitir a configuração e tratamento desse tipo de campo. A imagem a seguir mostra um campo com máscara presenta na aplicação SisGera para melhor ilustração.

Figura 10 – Demonstrativo de um campo com máscara



Fonte: Autor

3.1.2 Exploratory Testing

A Exploratory Testing é uma extensão disponível na maioria dos navegadores, mas principalmente ligada ao navegador Chrome. Sua seleção foi feita pela praticidade de utilização e por não precisar fazer instalações direcionadas ou outros softwares. Sua escolha se fez pela necessidade de organizar e salvar os resultados obtidos na execução do teste exploratório. A ideia é organizar e documentar os testes de forma fácil e clara. Essa extensão se destaca ainda por:

- Conseguir salvar uma *print* da página facilmente
- Gerar um relatório facilmente
- Manuseabilidade simples

3.1.3 JMeter

O JMeter é uma ferramenta que faz a execução de teste de carga. Talvez, seja a maior referência para esse tipo de teste. Sua escolha foi a mais simples dentre os outros testes, devido a sua fama, materiais disponíveis e também a sua qualidade de teste. O software oferece muitos recursos para a execução dos teste, visto que, é focado exclusivamente na tarefa de testar a capacidade da aplicação.

3.1.4 Zed Attack Proxy

Zed Attack Proxy (ZAP) é um software livre e aberto, responsável pela execução de teste de segurança, focado em teste de penetração. Ele consegue extrair as mais diversas falhas dos sites. Como descrito no Capítulo 2, a OWASP classifica as vulnerabilidades mais perigosas e sugere algumas ferramentas. O ZAP é uma dessas sugestões, sua funcionalidade que mais se destaca é a capacidade de interceptar e adulterar qualquer tipo de solicitação de entrada e saída de dados. Sendo assim, ele consegue obter uma gama de informações e retornar ao testador pontos que poderiam causar falhas graves ou não.

A escolha da ferramenta de segurança é a que gerou mais atenção, pois existiam alguns pontos muito importantes que precisavam levar em consideração, tais como:

- Quais falhas queremos capturar?
- Essas falhas estão na lista da OWASP?
- O software de teste, está atualizado?
- Qual o melhor software que satisfaz os itens anteriores?

Com base nas questões anteriores, a seleção e a pesquisa dessas ferramentas prosseguiu e foi criada uma relação das possíveis ferramentas na imagem a seguir, utilizando como referência a tabela disponibilizada pelo OWASP e o livro de Pentest em aplicações web de (MORENO, 2019):

Tabela 2 – Relação de ferramenta do livro Pentest com a tabela OWASP

Nome	Tipo	Pentest	OWASP
SQLMap	SQL Injection	Sim	SQL Injection
jSQL	Detecção de Vulnerabilidades de SQL Injection	Sim	-
Nikto	Scanner de Vulnerabilidades	Sim	XSS
Vega	Scanner de Vulnerabilidades	Sim	cross-site scripting
Skipfish	Scanner de Vulnerabilidades	Sim	-
Wapiti	Scanner de Vulnerabilidades	Sim	cross-site scripting
Zed Attack Proxy	Scanner de Vulnerabilidades	Não	Broken access, SQL Injection

Fonte: Elaborado pelo autor (2022)

3.2 Desenvolvimento dos Testes

Durante o decorrer desta seção, será apresentado o processo de desenvolvimento dos três tipos de testes no sistema.

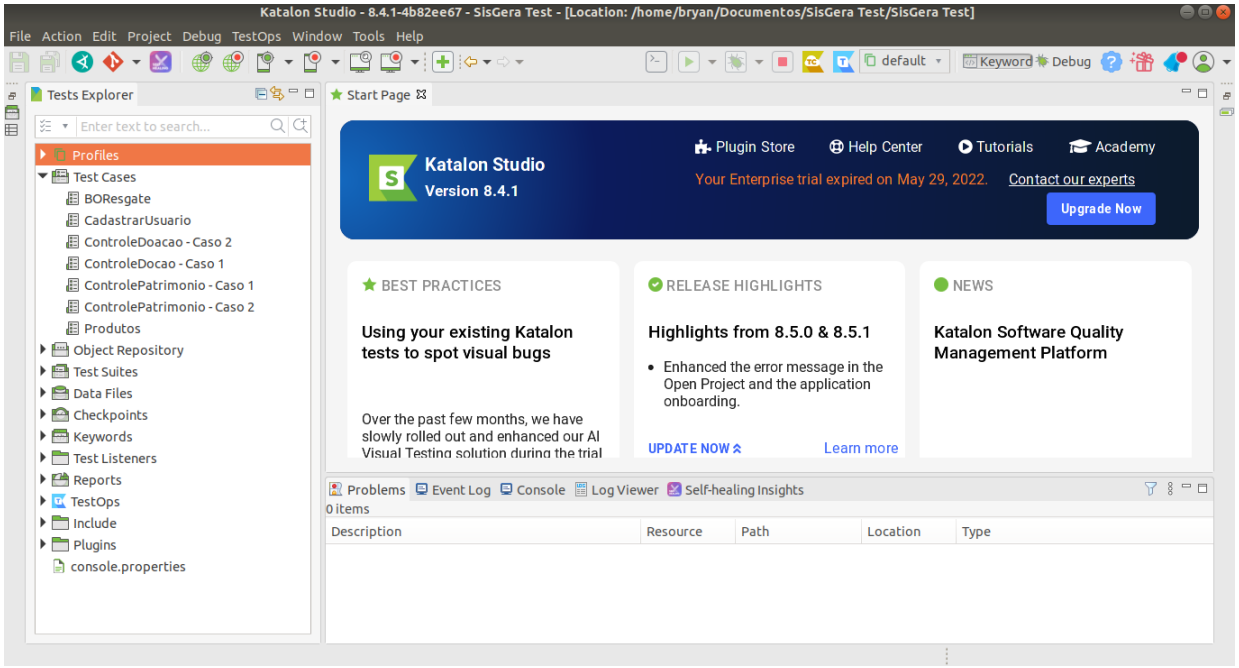
3.2.1 Teste Funcional

Os trabalhos de (ARANTES, 2018), (OLIVEIRA, 2018) e (SILVA, 2021) foram focados no desenvolvimento do sistema. Sendo assim, mesmo que tenha tido algum tipo de teste no sistema, não foi realizada de forma sistemática, e isso pode implicar na existência de determinados tipos de problemas.

Desta forma, foi selecionada a execução de Teste Funcional do tipo Exploratório. Assim como explicado na sessão 2, esta é uma metodologia em que o testador é livre para traçar o caminho que ele quiser. Contudo, nosso objetivo, além de achar essas falhas, é, também, documentar e armazena-las para que posteriormente a pessoa a cargo de corrigi-las tenha acesso a estes resultados.

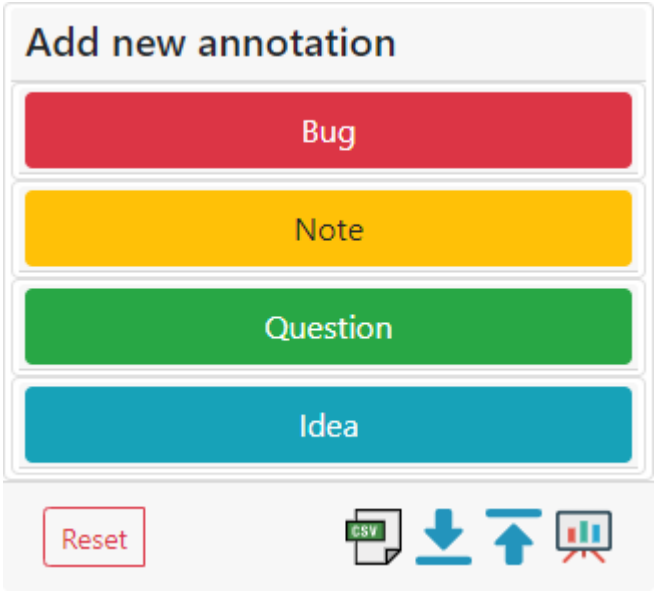
Assim, a ferramenta Katalon foi utilizada para salvar os *scripts* de execução e, junto do Exploratory Testing, para relacionar a descrição com uma *print* da página em questão. As imagens, a seguir, ilustram a janela do Katalon e do Exploratory Testing respectivamente.

Figura 11 – Tela inicial Katalon



Fonte: Autor

Figura 12 – Tela da aba do Exploratory Testing



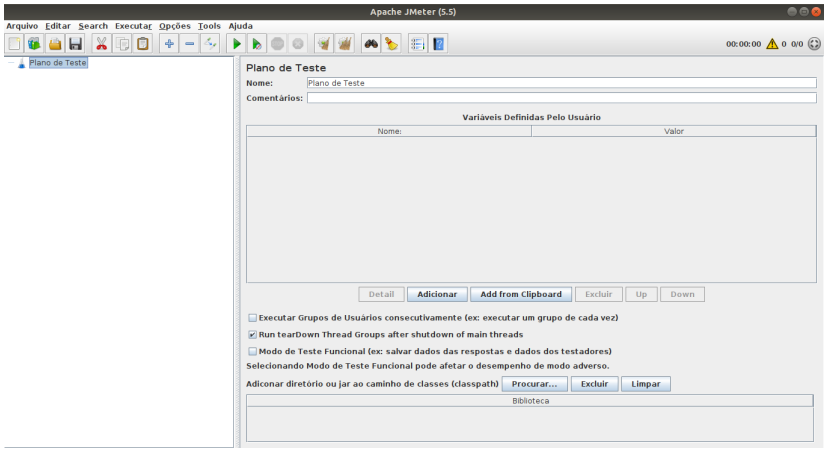
Fonte: Autor

Com as duas ferramentas em mãos, prosseguiu-se com o teste exploratório, observando todas entradas de dados e sua respectivas saídas. Caso um problema fosse identificado, seria gerado um *script* na Katalon. Logo após, seria registrado o ocorrido na Exploratory Testing, juntamente com o nome do *script* disponível. Desta forma, foi seguida a execução do teste exploratório para todo o sistema.

3.2.2 Teste de Carga

O desenvolvimento do Teste de Carga exigiu um estudo aprofundado da literatura Pro Apache JMeter: web application performance testing de (MATAM; JAIN, 2017). Esse estudo foi necessário, não porque o JMeter seja complexo, mas para retornar um resultado satisfatório, de modo a identificar qual a melhor forma de execução e os dados esperados. O JMeter é um software com uma interface bem simples, mas com várias possibilidades. Isso quer dizer que é possível avaliar diferentes tipos de resultados, com diferentes maneiras de executar e colher dados. A imagem a seguir mostra a janela do JMeter.

Figura 13 – Tela inicial JMeter



Fonte: Autor

Assim, foi necessário configurar determinados pontos no JMeter, passando a URL do sistema e a forma como seria executado o teste de carga. É importante ressaltar que os três desenvolvimentos de testes foram realizados no sistema executando localmente.

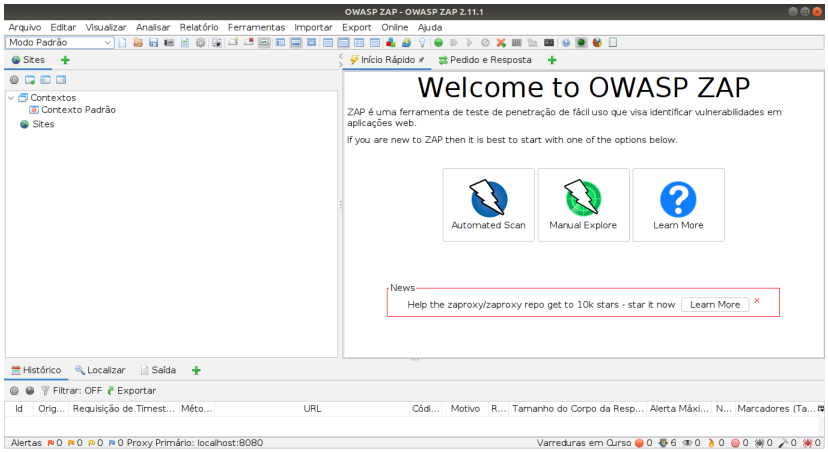
Atualmente, o sistema do SisGera está executando em um servidor e existe o sistema de homologação que também está funcionando em um servidor. Assim, para executar um teste de carga seria necessário pedir autorização do provedor de serviços e isso geraria uma grande burocracia. Então, optou-se pela execução em ambiente local, já que, obteria o resultado desejado da mesma forma e sem correr o risco de comprometer um servidor inteiro. Lembrando que testes do tipo carga, simulam a entrada de um grande volume de dados e usuários, ou seja, não é recomendado fazer sem autorização. Uma vez que o servidor poderia configurar isso como um ataque DoS (Denial Of Service) e isso acarretaria problemas não só ao servidor, quanto ao usuário operando o teste. Segundo (LAUFER et al., 2005), é um tipo de ataque que consome recursos do servidor e roteamento, e impede que usuários legítimos tenham acesso ao serviço.

3.2.3 Teste de Segurança

Dentre todos os testes executados, o Teste de Segurança é o mais importante. Pois, aqui estamos falando sobre manter a integridade de dados e do próprio sistema.

Sendo assim, foi selecionado a ferramenta de teste de maneira sistemática, com base em ensaios, documentações e resultados. Assim, o ZAP foi selecionado e usado para o desenvolvimento desta etapa. A imagem a seguir ilustra a tela do ZAP.

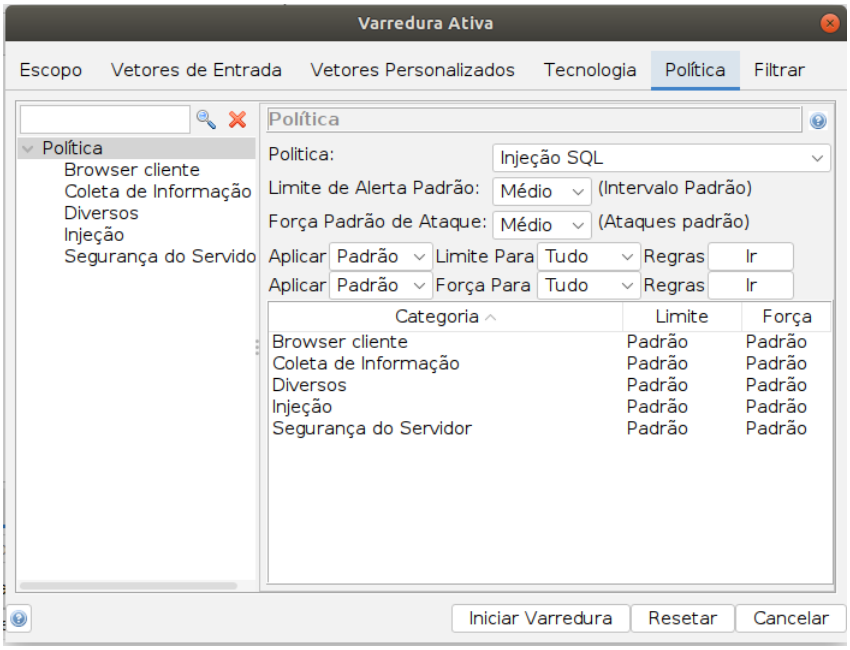
Figura 14 – Tela inicial Zed Attack Proxy



Fonte: Autor

Na primeira execução da ferramenta em cima do sistema SisGera, foi obtido uma resposta pouco satisfatória sobre o quanto era esperado. Contudo, como esse software diz respeito ao método de teste por penetração, foi necessário adentrar as configurações e alterar a amplitude, visto que, por padrão o software inicia por modo padrão. A imagem a seguir ilustra a página de configuração da força de penetração do software.

Figura 15 – Configurações da força do teste



Fonte: Autor

Após esse ajuste e a execução novamente do teste, foi obtido o resultado final e esperado da ferramenta que será descrito no Capítulo 4.

4 Resultados

Este capítulo apresenta os resultados obtidos neste trabalho, como as respostas geradas a partir da execução dos testes propostos.

4.1 Teste Funcional

Com a execução do Teste Funcional pelo método Exploratório, os resultados foram sendo listados e classificados na ferramenta Exploratory Testing. A base de registro foi bem simples, na página onde se encontra o problema foi tirado um *print* e descrito o problema, bem como em qual *script* se encontra a demonstração para tal ocorrência. Além disso, foi proposto algumas ideias e questões sobre determinados tópicos, por exemplo, o formato de um determinado campo apresenta um padrão em todos os casos? Se confirmado, poderia ser tratado, contudo, até o momento não era um ponto a se causar erro. A imagem a seguir, mostra os *scripts* de testes separados por funcionalidades do site.

Figura 16 – Aba de testes do Katalon

Fonte: Autor

Na imagem 16 é possível observar que apresenta algumas variações para a mesma funcionalidade do sistema, esse motivo se deve a demonstrações diferentes. Porém, algumas execuções apresenta apenas um *script* para mais de um registro no Exploratory Testing, isso se deve ao fato que na mesma tela foi observado problemas diferentes, e ao invés de gerar um *script* redundante. Desta forma, como o objetivo e catalogar os problemas, é feito um único registro para cada falha.

Figura 17 – Resultados do teste exploratório catalogados

Fonte: Autor

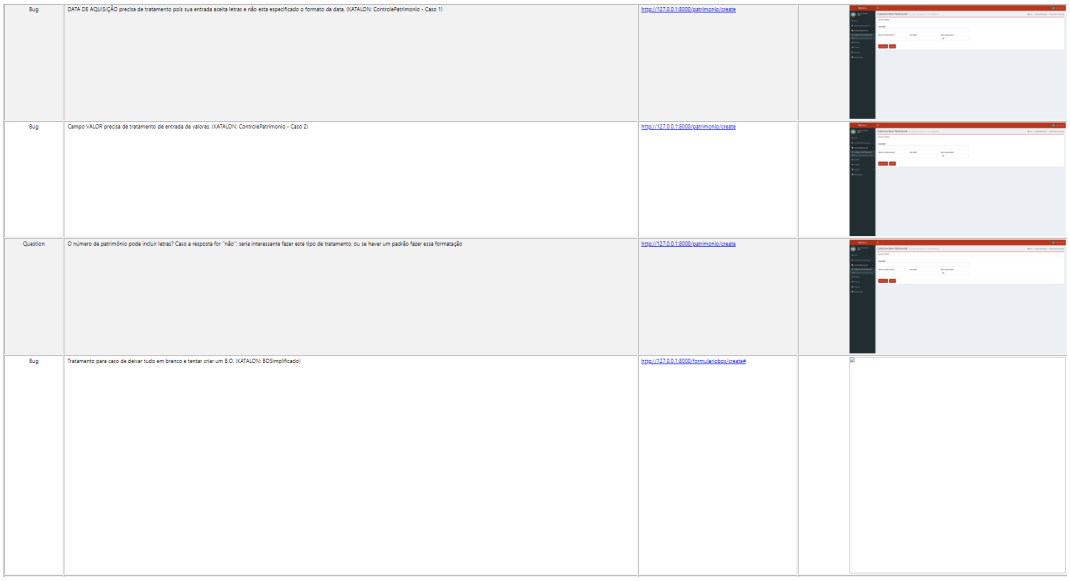
Figura 18 – Resultados do teste exploratório catalogados

Fonte: Autor

Figura 19 – Resultados do teste exploratório catalogados

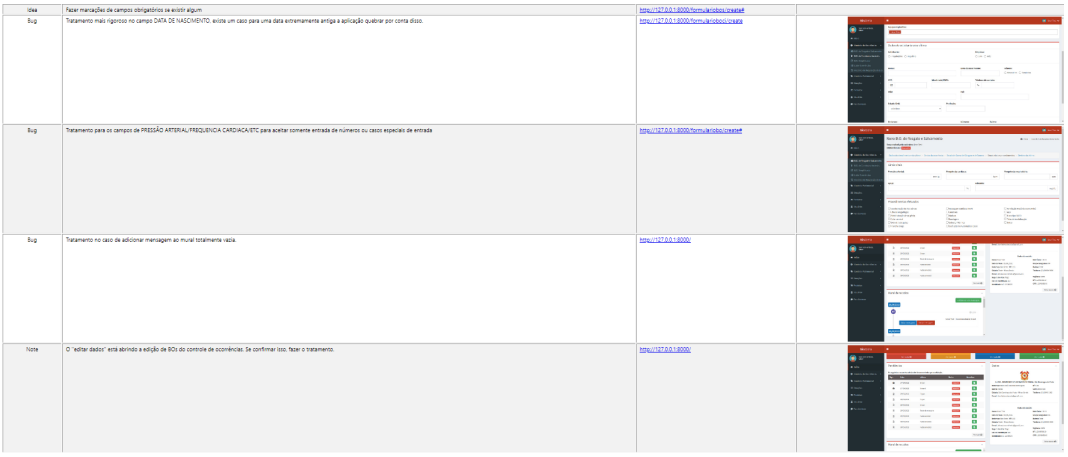
Fonte: Autor

Figura 20 – Resultados do teste exploratório catalogados



Fonte: Autor

Figura 21 – Resultados do teste exploratório catalogados



Fonte: Autor

Sendo assim, esses foi os dados registrados na ferramenta Exploratory Testing após a execução dos testes. Foram apontados ao todo, 21 sugestões de melhorias que poderiam estar afetando a ferramenta, sejam elas negativamente, ou como pontos de melhoria a qualidade do sistema. Nas imagens anteriores, é possível observar que foram descrito os problemas, e entre parenteses indica a referência a qual execução de *script* mostra o acontecimento, além disso, está acompanhado do *link* do *localhost*, bem como um *print* da página que está o ocorrido.

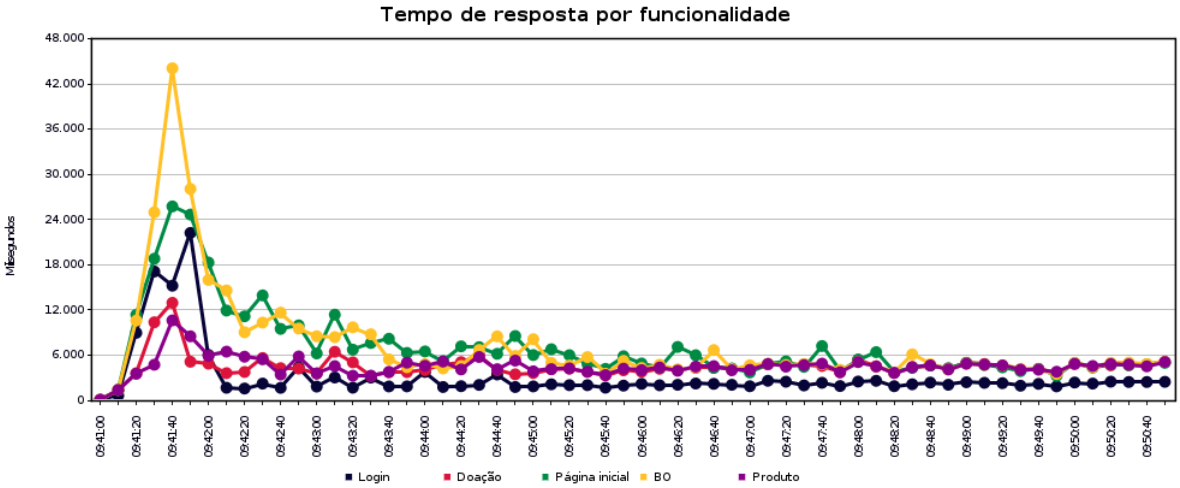
Assim, é possível afirmar que o teste foi positivo, já que ele conseguiu apontar problemas e melhorias ao sistema. Então, a pessoa que tiver esses dados em mãos no futuro, estará bem orientada na hora de seguir com as melhorias apontadas por este trabalho.

4.2 Teste Carga

Dentre os testes executados no SisGera ao longo deste trabalho, o teste de carga é sem duvidas aquele com resultados mais complexos a serem analisados. Isso se deve ao fato que a análise é feita com base em tempo de resposta, conexão, requisições, entre outros.

Para este teste, foi selecionado os principais páginas do sistema e realizado uma análise baseado em tempo de resposta por funcionalidade sob uma quantidade base de 500 usuários por tempo de observação de 10 minutos. Com essas configurações, foi possível obter o seguinte gráfico:

Figura 22 – Gráfico de tempo de resposta



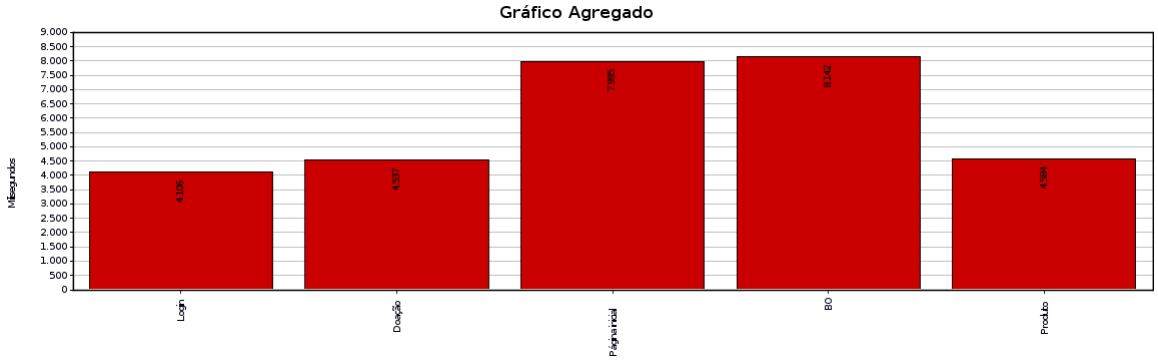
Fonte: Autor

No gráfico 22, é possível observar que ele mede o tempo de resposta em cada página ao longo do tempo, tempo esse dado por milissegundos. Sendo assim, quanto maior a linha do gráfico em relação ao eixo Y pior é o desempenho do gráfico naqueles instante. Ou seja, no começo da execução onde haviam cerca de 500 usuários fazendo requisições as páginas, passou-se a ter tempo de resposta muito alto, e isso implica em lentidão da aplicação.

Além disso, uma configuração foi imposta ao teste de que a cada falha de requisição do usuário, ele seria automaticamente retirado da amostra. Sendo assim, o inicio de 500 usuários passou a ser de 136 usuários passado os 10 minutos de execução. Isso mostra que o desempenho do sistema tende a se degradar rapidamente com essa quantidade de usuário, e mostra ainda que a estabilidade do sistema é atingida por volta de 136 usuários ativos.

Na imagem a seguir, foi pega a relação de requisição de cada usuário e condensado em um gráfico agregado que representa o tempo de resposta de todas requisições juntas nesse período de tempo observado.

Figura 23 – Gráfico agregado de tempo de resposta



Fonte: Autor

Além disso, foi gerado uma tabela do sistema onde faz a relação de quantidade de requisições em cada página no período de tempo observado e outras métricas, como por exemplo vazão e porcentagem de erro.

Figura 24 – Tabela relacionando os dados

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Sent KB/sec	Média de Bytes
Login	4829	4106	15	171421	15119,14	1,53%	7,9/sec	31,88	1,47	4135,5
Doação	4387	4537	55	149497	8620,35	0,46%	7,2/sec	38,43	2,73	5493,8
Página inicial	4744	7995	23	171994	20172,55	2,72%	7,7/sec	41,05	2,84	5426,0
BO	4583	8142	34	171742	20404,57	2,73%	7,5/sec	39,67	2,88	5426,8
Produto	4434	4584	43	172443	8872,91	0,50%	7,2/sec	38,85	2,76	5492,5
TOTAL	22977	5889	15	172443	15740,36	1,61%	37,5/sec	189,81	12,68	5180,7

Fonte: Autor

Essa tabela gerada pela ferramenta do JMeter devolve muitos dados aos testadores, e por ela é possível concluir muitas informações. As duas principais conclusões que podem ser obtidas ao observar os dados, está na coluna de Vazão e de Desvio Padrão. Quanto maior vazão de uma página, mais rápido ela consegue responder a grandes quantidades de volume de dados, ou seja, se for um número baixo, a página não consegue lidar com muitas informações. E o desvio padrão é uma unidade que quanto menor, melhor é o desempenho da página, já que representa o grau de variação.

Nos dados obtidos, é possível afirmar que o sistema sofre com alguns problemas de desempenho, ou seja, muitos usuários podem fazer o sistema apresentar instabilidade e sua resposta pode ficar lenta. Por outro lado, apresenta uma porcentagem de erro por requisição baixo, mostrando que o sistema é capaz de completar tarefas mesmo sobrecarregado.

Vale deixar registrado que os acessos as páginas é feito de forma aleatória pelo próprio JMeter, sendo assim, existe essas diferenças de quantidade de requisição por página.

4.3 Teste Segurança

Como foi descrito no 3, foi necessário executar algumas modificações nas configurações do ZAP, para que pudéssemos melhorar o obtenção dos resultados esperados. O resultado de sua execução é mostrado na figura abaixo que é possível perceber que a ferramenta lista os problemas de seguranças observados por prioridade.

Figura 25 – Relatório de execução do ZAP

Summary of Alerts

NÃvel de Risco	Number of Alerts
Alto	2
MÃ©dio	4
Baixo	6
Informativo	1

Alertas

Nome	NÃvel de Risco	Number of Instances
Injeção SQL	Alto	5
Redirecionamento Externo	Alto	1
Content Security Policy (CSP) Header Not Set	MÃ©dio	7
Missing Anti-clickjacking Header	MÃ©dio	4
Vazamento de informações .htaccess	MÃ©dio	1
Vulnerable JS Library	MÃ©dio	1
Application Error Disclosure	Baixo	1
Cookie No HttpOnly Flag	Baixo	8
Cookie without SameSite Attribute	Baixo	16
Divulgação de Data e Hora - Unix	Baixo	61
O servidor vaza informações por meio dos campos de cabeçalho de resposta HTTP "X-Powered-By"	Baixo	10
X-Content-Type-Options Header Missing	Baixo	13
Divulgação de Informações - Comentários Suspeitos	Informativo	12

Fonte: Autor

Além disso, a ferramenta trás analises ainda melhores quando se expande cada estrutura e analisa os dados. Porém, para a melhor observação, é necessário gerar um relatório, onde vai estar organizado todas essas informações. Após acionada a opção de gerar relatório e o abrindo-o, o sistema organiza cada falha capturada durante o teste por categoria, onde foi capturado o problema, como resolver esse tipo de problema e ainda sugere referenciais bibliográficas sobre o tema.

Vale descrever, que o relatório da ferramenta em questão apresenta algumas inconsistências como por exemplo não fazer a leitura de caracteres como o "ç"e acentuações. Então é possível ver que ele fica desformatado graças a esse problema, contudo, não é uma falha preocupante já que é possível entender a informação. A seguir é apresentado alguns dos problemas presentes no relatório da ferramenta, como por exemplo Injeção SQL (Structured Query Language) e entre outros.

Figura 26 – Resultado da captura de problemas de segurança por Injeção SQL

Alto	Injeção SQL
DescriÃ§Ã£o	Injeção SQL pode ser possível.
URL	http://127.0.0.1:8000/gerarBO
MÃ©todo	GET
Parameter	laravel_session
Ataque	eyJpdiI6ImtDU3FDUEQ2XC9GSjVWcU9FN3h4OWJ3PT0iLCJ2YWx1ZSI6ImpDVTlwaDFXTVk1SnplbEFxWHN6OHc1Q2xPc0F2QzBzYU1WTUVxejMrb1BmTXVYNStKSFBMMU1pQm5HOGFMbkoiLCJtYWMiOiIzYWQyMDAxZmY5ZWI5OGNkMGI4YzE3MTdmZjZmNTY0NGExNmU4ZGU4MzRiYWRkNzA5NzlkNTUxOWNlNDg2M2E0In0=%
Evidence	
URL	http://127.0.0.1:8000/login
MÃ©todo	GET
Parameter	laravel_session
Ataque	eyJpdiI6IkZnU1BZdmd1dGRRNVB6cmlnQkJxXC9nPT0iLCJ2YWx1ZSI6IlFwMlQ1a1NJdnRTNGt3ek83SzRcL1NpSXVTNnN1aDVCaEFDclR5b2JXRVpCVjBMa2NTeUFVQTdyVzV0b24rQVdyIiwibWFjIjoiNTY2M2Q3ZmIzZmI1ZGU2OTFkMjNiNTFlYzAyMmMxM2U1ZjgxZmZhMzhjZTA2ZDMzZmE0ZTY4OTRjMWRiNTE2YyJ9%
Evidence	
URL	http://127.0.0.1:8000/validarBO
MÃ©todo	GET
Parameter	laravel_session
Ataque	eyJpdiI6IlId2Z1hhSHphaII2aDNNbVJ5QXVVMFE9PSIsInZhbHVlIjoiWTI4aHFmVkwyTytnaHFtNHVuMktcL3BWbHJjdFR3VVpkcnlEU01FOCtMclhWWkZ1eVFROUE5K09HamphdnJaeUYiLCJtYWMiOiJkZGEwOGNhOWJiN2FkODhhY2M3NjQ5Y2U0YjQ3MWU0ZmQ3ZjAzZTA2Yzc5YTQ3ZjIxNTM5YmIwODc5MTMwNTUwIn0=%
Evidence	
URL	http://127.0.0.1:8000/confirmarBO
MÃ©todo	POST
Parameter	laravel_session
Ataque	eyJpdiI6IkRSQ2pUUFhQZ3I2a0Z2ejZ0TGJNT0E9PSIsInZhbHVlIjoiNWZvbExYbkEwTGNxeFR4S3ZxUFd2Vmg0QTAwVzAyWUpVUDBXVWxoMDg3dzc4V3NaOE80b2plb2xQQWNKWHZjUCIsIm1hYyI6ImIzZjkxOWUxYzE1MzA4OTA1YTQzMjlyY2VlMjcxMDllMDZmNDNhZTlhYjE5OGFhNjBhOWQxM2E4MTIyNGQ1NmUifQ==%
URL	http://127.0.0.1:8000/login
MÃ©todo	POST
Parameter	laravel_session
Ataque	eyJpdiI6IlRKcnNDQ1FlRno3V2l4TlFQbkE0V1E9PSIsInZhbHVlIjoiZDhwdjhCT3RyVjdUSmI3V1JxWFJ1ZytyQ2NWdzhNOWZcL0FRM0dGVHRTVW1haTk1SmRsdURvS3BxVkdEUHNiUlAiLCJtYWMiOiIzNzQ3ZWJkODk0ZTdlOTM3YmRlNTkwNGNlOGMxMThiYjQ1ODYzMDBlNjU4MmFlMDM2ODY2M2MxZTFiNjEwMDQ3In0=%
Evidence	
Instances	5
Solution	Não confie na entrada do lado do cliente, mesmo se houver validação do lado do cliente em vigor. Em geral, digite verificar todos os dados no lado do servidor. Se o aplicativo usar JDBC, use Declaração preparada ou Declaração exigível, com parâmetros passados por '?' Se o aplicativo usa ASP, use objetos de comando ADO com verificação de tipo forte e consultas parametrizadas. Se os procedimentos armazenados do banco de dados puderem ser usados, use-os. * Não * concatene strings em consultas no procedimento armazenado ou use 'exec', 'exec imediato' ou funcionalidade equivalente! Não crie consultas SQL dinâmicas usando concatenação de string simples. Escape de todos os dados recebidos do cliente. Aplique uma 'lista de permissão' de caracteres permitidos ou uma 'lista de negação' de caracteres não permitidos na entrada do usuário. Aplique o princípio do menor privilégio usando o usuário de banco de dados com o menor privilégio possível. Em particular, evite usar os usuários do banco de dados 'sa' ou 'db-owner'. Isso não elimina a injeção de SQL, mas minimiza seu impacto. Conceda o mínimo de acesso ao banco de dados necessário para o aplicativo.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Fonte: Autor

Figura 27 – Resultado da captura de problemas de segurança de Redirecionamento Externo

Alto	Redirecionamento Externo
DescriÃ§Ã£o	Os redirecionadores de URL representam uma funcionalidade comum empregada por sites para encaminhar uma solicitação de entrada a um recurso alternativo. Isso pode ser feito por vários motivos e geralmente é feito para pe funcionalidade para usuários que solicitam o recurso em seu local anterior. Os redirecionadores de URL também podem ser usados para implementar balanceamento de carga, aproveitando URLs abreviadas ou registrando links exemplo abaixo. Os redirecionadores de URL não representam necessariamente uma vulnerabilidade de segurança direta, mas podem ser violados por invasores que tentam fazer com que as vítimas acreditem que estão naveg
URL	http://127.0.0.1:8000/login
MÃ©todo	POST
Parameter	Referer
Ataque	2614381746526730018.owasp.org
Evidence	2614381746526730018.owasp.org
Instances	1
Solution	Presuma que toda a entrada de dados é maliciosa. Use uma estratégia de validação de entrada "aceita como boa", ou seja, use uma lista de permissões de entradas aceitáveis que estejam estritamente em conformidade com as transforme-as em algo que esteja. Não confie exclusivamente na procura de entradas maliciosas ou malformadas (ou seja, não confie em uma lista de negação). No entanto, as listas de negação podem ser úteis para detectar at imediatamente. Ao executar a validação de entradas de dados, considere todas as propriedades potencialmente relevantes, incluindo comprimento, tipo de entrada, a gama completa de valores aceitáveis, entradas ausentes ou extras, sintaxe, c lógica de regra de negócios. "barco" pode ser sintaticamente válido porque contém apenas caracteres alfanuméricos, mas não é válido se você estiver esperando cores como "vermelho" ou "azul". Use uma lista de permissão de URLs ou domínios aprovados a serem usados para redirecionamento. Use uma página de isenção de responsabilidade intermediária que forneça ao usuário um aviso claro de que ele está deixando seu site. Implemente um longo tempo limite (timeout) antes que ocorra o redirecionamento ou force responsabilidade. Havendo um conjunto de objetos aceitáveis, limitados ou conhecidos como nomes de arquivos ou URLs, crie um mapeamento de um conjunto de valores de entrada fixos (como IDs numéricos) para os nomes de arquivos ou UR Por exemplo, o ID 1 pode ser mapeado para "/login.asp" e o ID 2 pode ser mapeado para "http://www.example.com/". Recursos como o ESAPI AccessReferenceMap fornecem esse recurso. Compreenda todas as potenciais áreas onde entradas de dados não confiáveis podem ser inseridas em seu software: parâmetros ou argumentos, cookies, qualquer coisa lida a partir da rede, variáveis de ambiente, pesquisas de bancos de dados e quaisquer sistemas externos que forneçam dados para a aplicação. Lembre-se de que essas entradas podem ser obtidas indiretamente por meio de chamadas de API. Muitos problemas de redirecionamento aberto ocorrem porque o desenvolvedor presumiu que certas entradas não poderiam ser modificadas, como cookies e campos de formulário ocultos.
Reference	http://projects.webappsec.org/URL-Redirector-Abuse http://cwe.mitre.org/data/definitions/601.html

Fonte: Autor

Figura 28 – Resultado da captura de problemas de segurança de nível médio

MÃ©dio	Content Security Policy (CSP) Header Not Set
DescriÃ§Ã£o	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, Act
URL	http://127.0.0.1:8000/
MÃ©todo	GET
Parameter	
Ataque	
Evidence	
URL	http://127.0.0.1:8000/exibirBO
MÃ©todo	GET
Parameter	
Ataque	
Evidence	
URL	http://127.0.0.1:8000/gerarBO
MÃ©todo	GET
Parameter	
Ataque	
Evidence	
URL	http://127.0.0.1:8000/login
MÃ©todo	GET
Parameter	
Ataque	
Evidence	
URL	http://127.0.0.1:8000/sitemap.xml
MÃ©todo	GET
Parameter	
Ataque	
Evidence	
URL	http://127.0.0.1:8000/validarBO
MÃ©todo	GET
Parameter	
Ataque	
Evidence	
URL	http://127.0.0.1:8000/exibirBO
MÃ©todo	POST
Parameter	
Ataque	
Evidence	
Instances	7
Solution	Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header, to achieve optimal browser support: "Content-Security-Policy" for Chrome 25+, Firefox 23+ and Safari 7+, Safari 6+.
Reference	https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html http://www.w3.org/TR/CSP/ http://w3c.github.io/webappsec/specs/content-security-policy/csp-specification.dev.html http://www.html5rocks.com/en/tutorials/security/content-security-policy/ http://caniuse.com/#feat=contentsecuritypolicy http://content-security-policy.com/

Fonte: Autor

Figura 29 – Resultado da captura de problemas de segurança de nível médio

MÃ©dio	Missing Anti-clickjacking Header
DescriÃ§Ã£o	The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
URL	http://127.0.0.1:8000/
MÃ©todo	GET
Parameter	X-Frame-Options
Ataque	
Evidence	
URL	http://127.0.0.1:8000/gerarBO
MÃ©todo	GET
Parameter	X-Frame-Options
Ataque	
Evidence	
URL	http://127.0.0.1:8000/login
MÃ©todo	GET
Parameter	X-Frame-Options
Ataque	
Evidence	
URL	http://127.0.0.1:8000/validarBO
MÃ©todo	GET
Parameter	X-Frame-Options
Ataque	
Evidence	
Instances	4
Solution	Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively cons
Reference	https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

Fonte: Autor

Figura 30 – Resultado da captura de problemas de segurança de nível médio

MÃ©dio	Vazamento de informações .htaccess
DescriÃ§Ã£o	Os arquivos htaccess podem ser usados para alterar a configuração do software Apache Web Servidor para habilitar/desabilitar funcionalidades e recursos adicionais que o software Apache Web Servidor tem a oferecer.
URL	http://127.0.0.1:8000/.htaccess
MÃ©todo	GET
Parameter	
Ataque	
Evidence	HTTP/1.1 200 OK
Instances	1
Solution	Certifique-se de que o arquivo .htaccess não esteja acessível.
Reference	http://www.htaccess-guide.com/

Fonte: Autor

Figura 31 – Resultado da captura de problemas de segurança de nível médio

MÃ©dio	Vulnerable JS Library
DescriÃ§Ã£o	A biblioteca identificadajquery, versÃ£o 3.3.1 Ã© vulnerÃ¡vel.
URL	http://127.0.0.1:8000/css/bower_components/jquery/dist/jquery.min.js
MÃ©todo	GET
Parameter	
Ataque	
Evidence	/*! jQuery v3.3.1
Instances	1
Solution	Atualize para a versÃ£o mais recente do jquery.
Reference	https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/ https://nvd.nist.gov/vuln/detail/CVE-2019-11358 https://github.com/jquery/jquery/commit/753d591aea698e57d6db58c9f722cd0808619b1b https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/

Fonte: Autor

O perigo de um caso de Injeção SQL como mostrado na imagem , possibilita ao invasor interferir diretamente nos dados e consultas do banco de dados do sistema. Isso permitiria que ele alterasse, apagasse ou capturar qualquer dado que queira. Já um perigo como o Redirecionamento Externo pode fazer a vítima acreditar estar navegando o site que sempre navega, porém é um site falso que rouba suas informações. Outros tipos de problemas reportados pela ferramenta são perigos como vazamento de dados, alteração de dados, entre outros.

Por isso, o maior diferencial desta ferramenta, foi sem dúvidas a precisão e a qualidade de retorno das informações de cada problema capturado durante sua execução. Ao apontar o local onde foi observado a falha de segurança, a gravidade e ainda retornar a resolução, isso mostra muito valor da ferramenta, já que potencialmente em trabalhos posteriores facilita a resolução dos problemas identificados.

4.4 Desafios

Foram três tipos diferentes de testes executados ao longo do projeto e cada um com seus respectivos desafios. As primeiras dificuldades apareceram logo no primeiro teste, quando houve o problema com os campos de máscara comentados anteriormente na escolha da ferramenta do teste funcional. Além disso, a ferramenta *Exploratory Testing* apresentou uma limitação que, após o fechamento da ferramenta, os arquivos salvos não podiam mais ser editados. Isso implicou que a catalogação dos resultados do testes fossem executados completamente em uma única execução. Apesar do prosseguimento normal do projeto, isso poderia implicar em problemas futuros caso fosse executado de forma pausada os testes,

mas como foi executado completamente antes de passar ao próximo, não foi um problema que implicaria na troca da ferramenta.

Passando para os desafios na execução do teste de carga, um deles foi comentado anteriormente, o fato de só poder executar localmente, por precisar de liberação do servidor que está hospedado a aplicação. Esse problema influência diretamente em outro, o fato de executar localmente um teste de carga, ele não pode ser declarado 100% preciso. Isso acontece porque, como está sendo medido a capacidade de uma aplicação rodando localmente, os recursos disponíveis no computador como memória e processamento influenciam diretamente nos resultados obtidos. Contudo, eles ainda são válidos, já que mostram o comportamento do sistema em um ambiente controlado e nos dá uma estimativa real de como o sistema responderia em suas normais condições.

Já o teste de segurança mostrou poucos desafios, o principal foi em relação as configurações da ferramenta, já que elas indicavam como falhas seriam capturadas. Tirando esse detalhe apresentado, só existe um ponto no relatório final das falhas que deveria mostrar a prova da falha encontrada, que vêm vazio em alguns casos. Contudo, isso não mostrou um grande empecilho que, devido ao conhecimento prévio e a outras descrições, é possível saber que realmente aquele local apresenta o ponto da falha.

5 Conclusão

Este trabalho apresentou o desenvolvimento e a aplicação de testes de software para o sistema SisGera, sendo que tais testes servirão de base para melhorias futuras. O trabalho se baseou em três tipos de teste, sendo eles o teste funcional, teste de carga e o teste de segurança.

Em primeiro momento, foi realizado uma revisão da literatura envolvida neste trabalho. Foram levantadas informações sobre cada tipo de teste e suas subdivisões, além disso, foi mostrado como cada teste impacta em um sistema e sua importância geral.

Na etapa seguinte, foi mostrado como se deu as escolhas de ferramentas que seriam utilizadas no decorrer do projeto. No processo de escolha ainda foi falado alguns pontos importantes que foram cruciais em cada escolha.

E finalmente foi apresentado os resultados obtidos pela execução de cada um dos três tipos de teste. Eles foram separados por tipo e apresentado de forma singular, mostrando uma análise sobre os resultados.

Conforme apresentado no decorrer do trabalho, o objetivo de aplicar testes de software de maneira sistemática no sistema, com a garantia de deixar registrado para posteriores melhorias, foi alcançado com sucesso. Consequentemente, espera-se que os dados registrados possam garantir aos trabalhos posteriores uma orientação clara das melhorias necessárias, a fim de que o SisGera continue impactando positivamente as comunidades apoiadas pelo trabalho voluntário das corporações.

5.0.1 Trabalhos Futuros

Com base nas experiências adquiridas no decorrer do desenvolvimento deste trabalho, foram identificados algumas concepções e propostas para trabalhos futuros que serão apresentados a seguir:

- Execução das melhorias apontadas pela execução dos testes deste trabalho
- Teste da Aplicação Web progressiva, com o objetivo de mensurar consumo de recurso, como bateria e memória.
- Outros tipos de testes para expandir o espectro de testes abordados, tais como o teste de usabilidade.

Referências

ABREU, J. C. de; MARTINO, F. A.; SCHIAVONI, M. A. Testes exploratórios em software. *Revista Eletrônica Científica do CRA-PR-RECC*, v. 3, n. 1, p. 62–75, 2016. Citado na página 20.

ARANTES, V. M. Um sistema de informação para apoio ao registro de ocorrências atendidas por grupos de bombeiros voluntários. 2018. Disponível em: <http://www.monografias.ufop.br/handle/35400000/1188>. Citado 4 vezes nas páginas 13, 14, 15 e 30.

BARBOSA, E. F. et al. Introdução ao teste de software. *Minicurso apresentado no XIV Simpósio Brasileiro de Engenharia de Software (SBES 2000)*, 2000. Citado na página 19.

BOEHM, B. W. Software engineering economics. *IEEE Transactions on Software Engineering*, SE-10, n. 1, p. 4–21, 1984. Citado na página 17.

CASTRO, L. H. M. Desenvolvimento de um módulo de help desk para o sistema sisgera. 2022. Disponível em: <http://www.monografias.ufop.br/handle/35400000/4515>. Citado na página 14.

CRESPO, A. et al. Uma metodologia para teste de software no contexto da melhoria de processo. In: *Anais do III Simpósio Brasileiro de Qualidade de Software*. Porto Alegre, RS, Brasil: SBC, 2004. p. 204–218. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/sbqs/article/view/16194>. Citado na página 19.

DELAMARO, M.; JINO, M.; MALDONADO, J. *Introdução ao teste de software*. [S.l.]: Elsevier Brasil, 2013. Citado na página 20.

FELDERER, M. et al. Security testing: A survey. In: _____. [S.l.: s.n.], 2016. p. 1–51. ISBN 9780128051580. Citado na página 24.

HIEATT, E.; MEE, R. Going faster: testing the web application. *IEEE Software*, v. 19, n. 2, p. 60–65, 2002. Citado na página 17.

JIANG, Z.; HASSAN, A. E. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, v. 41, p. 1–1, 11 2015. Citado na página 23.

JORGENSEN, P. C. *Software testing: a craftsman's approach*. [S.l.]: Auerbach Publications, 2013. Citado 2 vezes nas páginas 18 e 19.

KANER, C.; BACH, J.; PETTICHORD, B. *Lessons learned in software testing*. [S.l.]: John Wiley & Sons, 2008. Citado na página 20.

KHAN, M. E. Different forms of software testing techniques for finding errors. *International Journal of Computer Science Issues (IJCSI)*, Citeseer, v. 7, n. 3, p. 24, 2010. Citado 6 vezes nas páginas 18, 21, 22, 23, 24 e 25.

LAUFER, R. P. et al. Negação de serviço: Ataques e contramedidas. *Sociedade Brasileira de Computação*, 2005. Citado na página 32.

MATAM, S.; JAIN, J. *Pro Apache JMeter: web application performance testing*. [S.l.]: Apress, 2017. Citado na página 31.

MORAIS, I.; SOARES, D.; ZANIN, A. *Engenharia de software. Disponível em: Minha Biblioteca*. [S.l.: s.n.], 2020. Citado na página 17.

MORENO, D. *Introdução ao Pentest*. [S.l.]: Novatec Editora, 2019. Citado 2 vezes nas páginas 26 e 29.

OLIVEIRA, S. S. M. Sistema de informação para controle de materiais e doações aos bombeiros voluntários. 2018. Disponível em: <http://www.monografias.ufop.br/handle/35400000/1621>. Citado 4 vezes nas páginas 13, 14, 15 e 30.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software-9*. [S.l.]: McGraw Hill Brasil, 2021. Citado 2 vezes nas páginas 20 e 25.

RIOS, E.; MOREIRA, T. *Teste de software*. [S.l.]: Alta Books Editora, 2006. Citado na página 17.

SCHIEFERDECKER, I.; GROSSMANN, J.; SCHNEIDER, M. Model-based security testing. *arXiv preprint arXiv:1202.6118*, 2012. Citado na página 24.

SILVA, G. O. Desenvolvimento de uma aplicação web progressiva para o registro de ocorrências de um grupo de bombeiros voluntários. 2021. Disponível em: <http://www.monografias.ufop.br/handle/35400000/3503>. Citado 3 vezes nas páginas 14, 15 e 30.

SOMMERVILLE, I. Engenharia de software. *10a. edição, Addison-Wesley/Pearson*, 2015. Citado na página 17.