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

Desenvolvimento de uma Aplicação Web Progressiva para o registro de ocorrências de um grupo de bombeiros voluntários

Gabriel Oliveira Silva

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Euler Horta Marinho

Setembro, 2021
João Monlevade–MG

Gabriel Oliveira Silva

Desenvolvimento de uma Aplicação Web Progressiva para o registro de ocorrências de um grupo de bombeiros voluntários

Orientador: Euler Horta Marinho

Monografia apresentada ao curso de Sistemas de Informação 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

Setembro de 2021

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S586d Silva, Gabriel Oliveira .
Desenvolvimento de uma aplicação web progressiva para o registro de ocorrências de um grupo de bombeiros voluntários. [manuscrito] / Gabriel Oliveira Silva. - 2021.
52 f.: il.: color..

Orientador: Prof. Me. Euler Horta Marinho.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Sistemas de Informação .

1. Aplicações Web. 2. Aplicativos móveis. 3. Engenharia de software. 4. Software de aplicação - Desenvolvimento. I. Marinho, Euler Horta. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.41

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

Gabriel Oliveira Silva

Desenvolvimento de uma Aplicação Web Progressiva para o registro de ocorrências de um grupo de bombeiros voluntários

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

Aprovada em 01 de setembro de 2021

Membros da banca

Mestre - Euler Horta Marinho - Orientador (Universidade Federal de Ouro Preto)
Doutor - Diego Zuquim Guimarães Garcia - (Universidade Federal de Ouro Preto)
Mestra - Daniela Rodrigues Dias - (Doutoranda em Educação - Universidade Federal de Ouro Preto)

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 01/10/2021

Documento assinado eletronicamente por **Euler Horta Marinho**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 01/10/2021, às 13:35, 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 **0227934** e o código CRC **2566D9C6**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: - www.ufop.br

Este trabalho é dedicado especialmente a minha família, amigos e a todos que fizeram parte do meu desenvolvimento pessoal e acadêmico.

Agradecimentos

Agradeço primeiramente a Deus pela saúde e aos meus pais João e Geane pelos sacrifícios realizados e por todo apoio e amor que sempre me deram. A minha irmã Isabela e a toda minha família pelo carinho e incentivo sempre oferecidos.

Aos professores e colegas que tive a oportunidade de conviver e aprender durante os anos de graduação.

Ao meu orientador Euler por acreditar no meu potencial e me guiar durante o desenvolvimento deste trabalho.

Aos percursores do SisGera, Vinícius e Silvandro e ao coordenador dos Bombeiros Voluntários de São Domingos do Prata, Bráulio, pela confiança, cooperação e por me permitir fazer parte deste projeto.

"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

As aplicações *Web* progressivas se mostram como uma evolução dos sistemas *Web* tradicionais. Suas tecnologias permitem que uma aplicação *Web* se comporte similarmente a uma aplicação nativa de dispositivos móveis, incluindo permitir o uso de funcionalidades *offline*. Dessa forma, é possível utilizar essa arquitetura para melhorar a usabilidade de um *site* no âmbito de mobilidade e conexões instáveis. Neste contexto, este trabalho tem como objetivo o desenvolvimento de uma aplicação *Web* progressiva baseada no SisGera, para o apoio ao registro de boletins de ocorrências atendidas por grupos de bombeiros voluntários que utilizam esse sistema e disponibilizar o seu uso sem acesso à *Internet*. Durante o desenvolvimento deste trabalho, foram analisados os principais conceitos envolvidos com esta tecnologia, além da caracterização da arquitetura *Web* tradicional e do desenvolvimento de aplicações móveis. Utilizando a aplicação desenvolvida é possível utilizar o SisGera para o registro de ocorrências mesmo sem conexão a rede, impactando positivamente no trabalho voluntário exercido pelas corporações e por consequência nas comunidades abrangidas por eles.

Palavras-chaves: aplicações *web*. aplicações *web* progressivas. aplicações móveis.

Abstract

Progressive Web applications are shown as an evolution of traditional Web systems. Its technologies allow a Web application to behave similarly to a native app of mobile devices, including allowing the offline use of functionalities. In this way, it is possible to use this architecture to improve the usability of a site in the scope of mobility and unstable connections. In this context, this work has the goal of develop a Progressive Web App based on SisGera, to support the registration of occurrences attended by groups of volunteer firefighters who use this system and make its use available without Internet access. During the development of this work, the main concepts involved with this technology were analyzed, in addition to the characterization of the traditional Web architecture and the mobile application development . Through the use of the application, it is possible to use SisGera to record occurrences even without network connections, positively impacting the voluntary work performed by these groups and by consequence, its communities abranged.

Key-words: web applications. progressive web apps. mobile applications.

Lista de ilustrações

Lista de abreviaturas e siglas

AW Aplicação *Web*

AWP Aplicação *Web* Progressiva

SW *Service Worker*

SisGera Sistema de Gerenciamento e Registro de Atividades

JSON *JavaScript Object Notation*

AJAX *Asynchronous JavaScript e XML*

CSRF *Cross-site Request Forgery*

Sumário

1 Introdução

O SisGera é uma Aplicação *Web* (AW) desenvolvida durante o trabalho de Arantes (2018) e Oliveira (2018) para atender as necessidades observadas das organizações de Bombeiros Voluntários de São Domingos do Prata e Bombeiros Voluntários de Barão de Cocais.

O trabalho de Arantes (2018), abordou a demanda dos bombeiros em fazer o registro das ocorrências atendidas de forma rápida e eficiente, em consideração que, anteriormente, faziam os registros de forma manual e escrita, tornando o processo mais lento, inconsistente e suscetível a erros. Assim, durante seu trabalho, foi desenvolvido um sistema para apoio ao registro de boletins de ocorrências. Para unificar os modelos de boletins utilizados nas corporações, Arantes (2018) criou três modelos distintos de boletins para diferentes ocasiões atendidas: Resgate e Salvamento, Combate a Incêndios e Simplificado.

Em sequência, Oliveira (2018) deu ênfase nas deficiências observadas em relação a área administrativa das corporações e a possibilidade do desenvolvimento de novas funcionalidades, como o controle do estoque de materiais e de doações.

Atualmente, este sistema está em operação, atendendo às principais necessidades do grupo de bombeiros, conseguindo assim, facilitar suas tarefas administrativas e operacionais, tornando as rotinas diárias mais eficientes. Porém, as deficiências originadas da arquitetura das aplicações Web, em particular, sua necessidade de uma conexão confiável à *Internet*, motivaram o desenvolvimento deste trabalho que consiste na criação de uma Aplicação *Web* Progressiva (AWP) para o registro de ocorrências *offline* e sua posterior sincronização com o servidor.

1.1 Problema

O SisGera é utilizado por grupos de bombeiros voluntários que, previamente a sua implantação, realizavam os registros de forma escrita pois não contam com os recursos necessários para adquirir um Sistema de Informação comercial. Isso ocasionava lentidão nos registros e erros dos dados, o que levava a um comprometimento do exercício das funções administrativas e operacionais como um todo. Esta deficiência motivou o trabalho de Arantes (2018), que acabou observando novas necessidades relacionado a área administrativa, o que instigou o desenvolvimento do trabalho de Oliveira (2018).

Com a implantação do Sistema de Informação criado, foi levantado um novo questionamento por Arantes (2018). Durante o exercício das funções, especialmente durante o atendimento de ocorrências, os bombeiros podem precisar se deslocar até áreas sem

cobertura de *Internet*, ou no máximo com conexão ruim, e com isso, pelo SisGera se tratar de uma AW, ficariam impossibilitados de usar o sistema para o registro das ocorrências atendidas.

Uma AWP é um modelo de sistema que permite o uso de funções de um sistema *Web* sem ligação a rede, se apresentando como uma solução para o problema encontrado. Dessa forma, torna-se possível o registro de ocorrências e armazenamento temporário no dispositivo do bombeiro e posteriormente o envio dos dados para o banco de dados do servidor quando voltar a conexão.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver uma versão *Web* progressiva para apoio ao registro de ocorrências do sistema SisGera.

1.2.2 Objetivos específicos

Este trabalho possui aos seguintes objetivos específicos:

- Descrever as aplicações *Web* e algumas abordagens de desenvolvimento para dispositivos móveis.
- Introduzir o conceito e as tecnologias envolvidas nas AWPs.
- Utilizar ferramentas de teste para a validação da aplicação desenvolvida.

1.3 Organização do trabalho

O restante deste trabalho é organizado como se segue. O Capítulo 2 apresenta a revisão bibliográfica do trabalho, introduzindo os principais conceitos presentes. O Capítulo 3 demonstra o desenvolvimento da versão AWP para o SisGera. Ao decorrer do Capítulo 4 é apresentado os resultados obtidos ao decorrer do desenvolvimento do trabalho. Finalmente, o Capítulo 5 traz as considerações finais e a conclusão deste trabalho.

2 Revisão bibliográfica

Este capítulo apresenta uma revisão bibliográfica e das tecnologias envolvidas no contexto deste trabalho, com ênfase nas tecnologias e práticas ligadas as AWPs.

2.1 Aplicações *Web*

As aplicações *Web* são sistemas projetados para serem utilizados por meio da Internet. Nos dias atuais, é possível a utilização de sistemas *Web* cada vez mais poderosos, já que a ligação entre o cliente e servidor é cada vez mais confiável devido ao avanço da rede de computadores e a evolução da computação como um todo. Consoante a Pressman e Maxim (2016, p: 9), essa arquitetura de sistema "evoluiu para sofisticadas ferramentas computacionais que não apenas oferecem funções especializadas ao usuário, como também foram integradas aos bancos de dados corporativos e as aplicações de negócio".

Esse modelo de sistema, tem como a principal característica a alta acessibilidade e mobilidade, tendo em consideração que, por não requerer nenhum tipo de instalação no dispositivo do usuário, é possível o acesso a qualquer momento e de qualquer lugar desde que esteja conectado a *Internet*, além da possibilidade do uso em dispositivos heterogêneos ou até mesmo ultrapassados. Outra característica relevante é o baixo custo de desenvolvimento e manutenção ao se comparar com um sistema nativo.

As principais tecnologias envolvidas em um ambiente Web são:

- **HTML** - *HyperText Markup Language* : Linguagem de marcação utilizada para a construção da estrutura de uma pagina Web.
- **CSS** - *Cascading Style Sheets* é uma linguagem usada para definir o estilo e a apresentação de um documento escrito em HTML.[1]
- ***JavaScript*** - O *JavaScript* é uma linguagem de programação que permite a implementação de itens complexos em páginas Web. Ele interage com o restante da página para gerenciar processos e dados, fornecendo uma experiência mais dinâmica e completa ao usuário. [2]

[1] Disponível em: https://www.developer.mozilla.org/pt-BR/docs/Web/CSS. Acesso em 20 de mai. de 2021

[2] Disponível em: https://www.developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em 20 de mai. de 2021

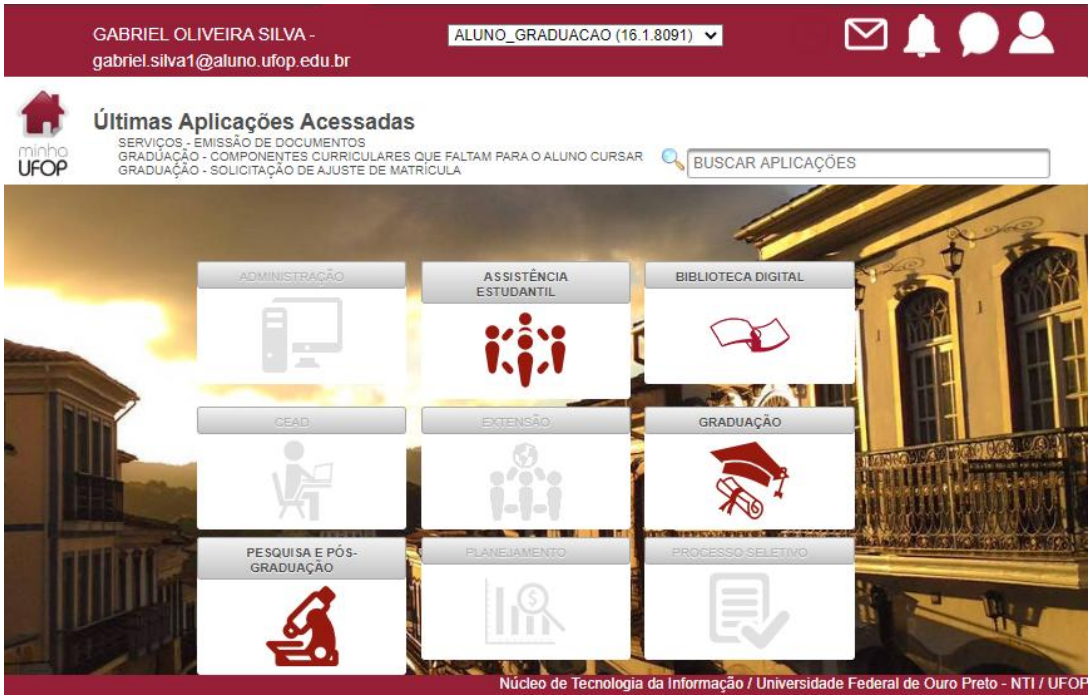
- **PHP** - *PHP: Hypertext Preprocessor* é uma linguagem de *script open source* de uso geral, especialmente adequada para o desenvolvimento *Web*. É focada principalmente na gestão dos *scripts* no lado do servidor. [3]

- **HTTP** - *HyperText Transfer Protocol* é o protocolo de comunicação que permite a obtenção de recursos, como documentos HTML. É a base de qualquer troca de dados na Web e um protocolo cliente-servidor.[4]

- **HTTPS** - *HyperText Transfer Protocol Secure* é uma implementação do HTTP que utiliza uma camada TLS - *Transport Layer Security* que fornece a encriptação e integridade dos dados garantindo que não será modificado ou acessado por um terceiro, além de autenticar a comunicação entre as duas partes, protegendo contra um ataque.

A Figura 1 exibe a tela inicial do portal minha UFOP, um exemplo de AW.

Figura 1 – Portal Minha UFOP, exemplo de aplicação *Web*.



Fonte: https://www.zeppelin10.ufop.br/minhaUfop/desktop/principal.xhtml. Acesso em: 10 de mai. de 2021

[3] Disponível em: https://www.php.net/. Acesso em 20 de mai. de 2021
[4] Disponível em: https://www.developer.mozilla.org/pt-BR/docs/Web/HTTP. Acesso em: 10 de mai. de 2021

2.1.1 Navegadores

Sobre os navegadores *Web* é possível afirmar que

> São ferramentas de *software* de fácil uso, utilizadas para apresentar páginas da *Web*, para acessá-la e acessar outros recursos da Internet. Os navegadores da *Web* tornaram-se a principal interface de acesso à Internet ou de utilização de sistemas em rede baseados nessa tecnologia. (LAUDON; LAUDON, 2014, p: 163).

Dessa forma, essas ferramentas são a base de toda AW, e algumas delas possuem funcionalidades especiais que proporcionam uma evolução dessa categoria de sistema. Os navegadores que possuem o uso mais difundidos nos dias de hoje são o `Google Chrome`, `Mozilla Firefox`, `Microsoft Edge` e o `Safari`.

2.2 Desenvolvimento para dispositivos móveis

Os *smartphones* e outros dispositivos móveis possuem limitações e particularidades que tornam o desenvolvimento de aplicações um desafio maior do que para os computadores. Os desafios mais relevantes estão na necessidade de gerenciamento da bateria, os limites relacionados ao espaço menor de armazenamento interno que forçam o uso eficiente do espaço disponível e a convivência com a *Internet* móvel, geralmente bem menos confiável que outras conexões.

Além disso, o desenvolvimento esbarra em diferentes tipos de sistema operacional, como *Android* e *iOS* e diferenças grandes de *hardware*, o que ocasiona em muitas vezes no uso de diferentes linguagens de programação em uma única aplicação.

Todavia, qualquer complicação encontrada e o alto custo ocasionado durante a fase de desenvolvimento são justificáveis, pois esse modelo de dispositivo simboliza uma parcela considerável do mercado da tecnologia. De acordo com uma pesquisa do site Gartner, quase 380 milhões de *smartphones* foram vendidos para o usuário final durante o primeiro quarto de 2021[5]. Com isso, fornecer a melhor experiência de uso para o usuário se mostra crucial para que uma empresa permaneça competitiva frente aos concorrentes.

Nesse contexto, existem diferentes tipos de abordagens e de arquiteturas disponíveis para os programadores escolherem ao criar aplicações para dispositivos móveis, onde alguns deles estão descritos a seguir.

2.2.1 Aplicações Nativas

O principal modelo de aplicações desenvolvidas para dispositivos móveis atualmente é conhecido como aplicativos nativos. Consoante a Jobe (2013, p: 2), "aplicações nativas

[5] Disponível em: https://www.gartner.com/en/newsroom/press-releases/2021-06-07-1q21-smartphone-market-share. Acesso em: 30 de jul de 2021

referem-se a programas que são particularmente escritos e desenvolvidos para um sistema operacional especifico".

Conforme Hume (2017, p. 4), "as aplicações nativas móveis conseguem oferecer uma experiência geral muito melhor ao usuário. Você instala a aplicação e ela carrega imediatamente. Se não houver conexão à rede não será o fim do mundo. Você já tem a maioria dos recursos necessários para atender seu cliente instalado no dispositivo". Ainda conforme Hume (2017, p. 4), "a Web não oferece esses mesmos benefícios, tais como o uso de recursos offline, tempos de carregamento baixos ou até instantâneos e confiabilidade aprimorada".

Dessa maneira, ao utilizar essa arquitetura de aplicação, o sistema resultante estará completamente adaptado a interagir melhor com o usuário. Funcionalidades como notificações, acesso rápido por um ícone, uso com conexão ruim ou até sem nenhuma e a utilização eficiente dos módulos do dispositivo proporcionam uma experiência de uso superior.

Essas aplicações podem ser criados para as plataformas móveis como *iOS, Android* e outros sistemas operacionais e requerem um desenvolvimento focado especificamente para cada um desses sistemas, demandando assim mais tempo e desenvolvedores. Outro problema é que essas aplicações estão sujeitas a aprovação da loja de aplicativos dos sistemas móveis, em específico, da Play Store[6] nos dispositivos *Android* e da App Store[7], nos aparelhos com o sistema *iOS*. Esses dois pontos somados acabam, por consequência, gerando um custo mais elevado de desenvolvimento e de manutenção ao se comparar com uma AW.

2.2.2 Desenvolvimento Multiplataforma

A heterogeneidade dos dispositivos móveis, tanto no contexto de sistemas operacionais quanto nos muitos fabricantes de aparelhos e *hardware* nos dias de hoje, tem criado dificuldades para os desenvolvedores de aplicações conseguirem entregar um bom produto. Em busca de usabilidade e desempenho para seu usuário acaba levando a utilização de linguagens diferentes para cada ambiente. Para Xanthopoulos e Xinogalos (2013), o objetivo final do desenvolvimento multiplataforma, é alcançar a performance das aplicações nativas e executar em tantas plataformas quanto for possível.

Esse modelo de aplicação gira ao redor de uma única implementação e do reaproveitamento de código para a geração de aplicações aptas a funcionarem em plataformas distintas, gerando códigos nativos para os diferentes modelos de sistema. Exemplos de *frameworks* utilizados por desenvolvedores dessa arquitetura são o `React Native, Ionic e o Flutter.`

[6] Disponível em: https://www.play.google.com/store/. Acesso em: 01 de jul. de 2021
[7] Disponível em: https://www.apple.com/br/app-store/. Acesso em: 01 de jul. de 2021

2.2.3 Aplicações Web Progressivas

Finalmente, utilizando-se do poder da Web e dos navegadores modernos, é possível aprimorar a experiência de uso de uma AW em um dispositivo móvel e agregar valor para a qualidade do sistema quando esse modelo é o escolhido. Essa evolução da Web é conhecida como AWP.

Segundo Wargo (2020, p: 3), "uma AWP é uma AW que utiliza recursos especiais do navegador permitindo-a agir como uma aplicação nativo quando executado em navegadores capazes".

Baseando-se em Sheppard (2017, p: 6), As características principais das AWP são :

Rapidez: Consegue carregar/executar no dispositivo em poucos segundos.

Confiável: Funciona sem uma conexão a Internet sólida e pode oferecer algumas funcionalidades *offline*.

Engajante: Assim como uma aplicação nativa, pode ser instalada na tela inicial de um *smartphone*, possuindo ícone e tela de abertura e também é capaz de enviar notificações ao usuário, mesmo sem o navegador estar aberto e usar os módulos de um dispositivo.

Deste modo, as AWPs são confiáveis e possuem uma boa experiência de uso assim como as aplicações nativas, ao mesmo tempo em que possuem a alta acessibilidade e relativo baixo custo de desenvolvimento e manutenção, característicos das aplicações *Web*, agindo como um meio termo entre esses dois modelos tradicionais. Por fim, essa arquitetura pode ser facilmente integrada com um sistema *Web* já em operação, se mostrando assim uma alternativa ideal para desenvolvimento para dispositivos móveis baseados em uma aplicação já pronta.

A Figura 2 demonstra uma AWP inicializada a partir de um ícone na tela inicial de um dispositivo onde não é possível perceber alguma diferença significativa para a aplicação padrão, nem que está sendo executado pelo navegador.

No entanto, essa arquitetura, sofre limitações. Uma delas é a dependência do navegador que por consequência, não estarão disponibilizados para acesso nas lojas de aplicativos dos dispositivos ocasionando em perda de visibilidade ao ser comparada com aplicações nativas. Outra dificuldade que vem sendo minimizada com o amadurecimento das AWPs, está no acesso aos módulos e recursos de *hardware* dos dispositivos.

As AWPs utilizam-se das seguintes tecnologias para prover essas funcionalidades e se destacarem de uma AW convencional.

Figura 2 – Tela principal da versão AWP do *Twitter*.

Fonte: https://www.twitter.com/. Acesso em: 01 de nov. de 2020

2.2.3.1 Web App Manifest

Um Web app manifest é um arquivo *JavaScript Object Notation* (JSON) que, apesar de ser simples, é crucial para o bom funcionamento de uma AWP. Conforme Hume (2017, p: 67), esse arquivo fornece informações sobre a aplicação como o nome, autor, ícone e descrição. Também possibilita o usuário a instalação da aplicação na tela inicial e permite a customização do ícone, tela de abertura, cores de fundo.

2.2.3.2 Service Worker

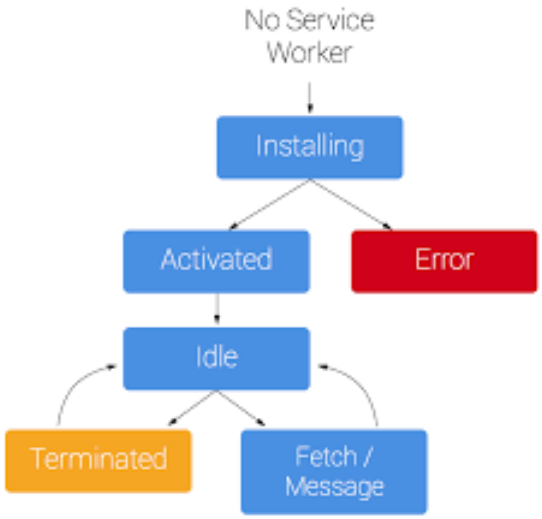
O *Service Worker* (SW) é a principal tecnologia das AWPs. É ele quem possibilita o uso constante sem ter uma conexão boa, a sincronização *offline* e o envio das notificações.

Segundo informações do *Google* [8], um SW é um *script* que o navegador executa em segundo plano, separado da página da *Web*. Isso possibilita recursos que não precisam de uma página da *Web* ou de interação do usuário. Essa tecnologia surgiu como o sucessora do *AppCache*, uma tentativa anterior para sanar o principal problema das AWs [9].

Além de requisitar um navegador compatível, só é possível registrar um SW em páginas disponibilizadas em HTTPS, para garantir que o SW não seja adulterado, tendo em consideração que, como permite o controle sobre a conexão, bem como fabricar e filtrar respostas, poderia ser utilizada por um terceiro não autorizado para sequestrar a conexão do usuário. O uso exclusivo do HTTPS também garante uma alta segurança para as AWPs.[10]

O SW, antes de ser usado pela primeira vez, deve ser registrado durante o primeiro acesso do usuário. Após registrado, o seu *script* é instalado no dispositivo do usuário e ativado. Em sequência, ele entra em uma fase ociosa onde aguardará por eventos ou requisições, tais como a perda de conexão, e então responderá conforme programado pelo desenvolvedor. Esse processo é demonstrado na Figura 3.

Figura 3 – Ciclo de vida do *Service Worker*.



Fonte: https://www.developers.google.com/Web/fundamentals/primers/service-workers?hl=pt-br. Acesso em: 23 de mai. de 2021

Tudo isso ocorre em segundo plano. Assim, o usuário não percebe as operações acontecendo e pode continuar a usar a aplicação normalmente.

[8] Disponível em: https://www.developers.google.com/Web/fundamentals/primers/service-workers?hl=pt-br. Acesso em: 18 de mai. de 2021

[9] Disponível em: https://www.developer.mozilla.org/pt-BR/docs/Web/API/Service_Worker_API. Acesso em: 18 de mai. de 2021

[10] Disponível em: https://www.developers.google.com/Web/fundamentals/primers/service-workers?hl=pt-br. Acesso em: 18 de mai. de 2021

2.2.3.3 *Cache*

A principal função do SW é permitir que a aplicação possa executar e oferecer algumas funcionalidades mesmo com uma conexão ruim ou inexistente. Para isso, é necessário armazenar partes da aplicação no dispositivo do usuário, de forma que seja fácil e rápida a recuperação dos arquivos requisitados pelo navegador. A interface *Cache*, presente nos principais navegadores, aborda essa necessidade.

Para Wargo (2020, p: 139), "a *Cache* é um repositório local de pares *request/response*. A *request* é um objeto representando a solicitação de um determinado recurso da AW, e a *response* é um objeto que representa a resposta do servidor para esse recurso específico."

A API *Cache* proporciona um mecanismo para armazenamento diretamente no navegador. Pode-se guardar pequenas partes da aplicação como imagens e mensagens, ou também o todo, desde que esteja no limite de tamanho de armazenamento que o navegador fornece.

Assim, é necessário o gerenciamento adequado dos recursos de uma AW, tanto para armazenar quanto para fornecer ao usuário. As principais estratégias para o armazenamento dos recursos são:[11]

Na Instalação: Durante a instalação do SW é indicado guardar os arquivos HTML, CSS, JS e outros estáticos que compõem o chamado *app shell*, a parte crucial da aplicação que sem ela tudo é interrompido e nada mais pode executar.

Na Interação do Usuário: Partes nas quais o usuário pode se interessar em acessar *offline* como vídeos, artigos e imagens. Um método comum é fornecer um botão "Ler Depois" para o usuário escolher o que quer salvar que, quando clicado, traz da rede e adiciona à cache.

Na resposta da rede: Se a requisição não está na cache, é interessante obter da rede e enviar para a *Cache*. É indicado para recursos atualizados com frequência. Porém, pode sobrecarregar o armazenamento. Então, deve-se eliminar os itens que não são mais necessários que ainda estejam armazenados na cache.

Logo, é importante destacar as seguintes técnicas para servir os recursos da cache ao utilizador:

Somente *Cache*: Usa apenas o que estiver na *Cache*, e é indicado para servir materiais estáticos.

[11] Disponível em: https://www.developers.google.com/Web/ilt/pwa/caching-files-with-service-worker. Acesso em: 04 de jul. de 2021

Somente Rede: Método para funcionalidades da aplicação que não podem ser feitas *off-line*. Por exemplo, as transações monetárias.

Cache, com *fallback* para a rede: Recomendada para aplicações focadas em uso *off-line*. Utiliza-se da cache quando ela existe e da rede quando não estiver disponível. Assim tem, o comportamento de "Somente *Cache*" para itens armazenados e "Somente Rede" para recursos que não podem ser guardados.

Rede, com *fallback* para a Cache: Uma abordagem indicada para recursos que são atualizados com frequência, por exemplo, placares de uma partida de futebol. Oferece aos usuários *on-line* o conteúdo atualizado, e os usuários *off-line* recebem uma versão anterior armazenada em cache. No entanto, esse método tem falhas. Um usuário com conexão lenta terá de esperar por uma falha de rede até receber o conteúdo da cache.

Primeiro a Cache: Abordagem também recomendada para recursos que atualizam com frequência. Fornece algum conteúdo o mais rápido possível e atualiza assim que a resposta atualizada da rede chega. No entanto, é preciso tomar o cuidado de não substituir algo que o usuário esteja interagindo.

***Fallback* Genérico:** Quando ocorre algum problema ao servir um recurso da cache e não é possível obter este recurso da rede, é interessante implementar um plano de contingência. Essa técnica se baseia em substituir o conteúdo esperado por uma alternativa que mantenha o usuário no contexto da aplicação. Um exemplo do uso dessa técnica seria fornecer uma página informando sobre problemas relacionados a conexão do dispositivo.

Já que é comum uma aplicação empregar mais de uma estratégia, cabe ao desenvolvedor utilizá-las da melhor maneira possível, escolhendo o momento certo de implementar cada uma delas.

2.2.3.4 Notificações

O envio de notificações é uma ferramenta muito eficiente para manter a aplicação engajante para o usuário. Isso torna a experiência de uso melhor e incentiva o utilizador a retornar frequentemente ao aplicativo.

Baseando-se em Sheppard (2017, p: 109), existe uma responsabilidade por parte do desenvolvedor gerenciar como e quando as notificações serão utilizadas, enviar requisições para notificações muito cedo pode afastar o cliente, também é importante controlar a quantidade de notificações para que mantenha o usuário atraído, porém que não o faça sentir-se assediado. A Figura 4 exibe uma solicitação de permissão do *Twitter* para o envio de notificações *push*.

Figura 4 – Solicitação de permisão para o envio de *push notifications* da versão AWP do *Twitter*.

Disponível em: https://www.twitter.com/, Acesso em: 01 de nov. de 2020

Existem duas categorias de notificações em um contexto de AWP:

- ***Web Notifications*** : são as notificações enviadas diretamente pelo navegador. Seu uso é limitado a somente quando o navegador estiver aberto, dessa forma, não se mostra tão eficaz para o engajamento do utilizador.
- ***Push Notifications*** : uma ferramenta poderosa para as AWPs, permite o envio das notificações mesmo sem o navegador aberto por meio de um servidor de notificações, que controla os envios. Isso mantêm o usuário engajado com o sistema mesmo ele não estando em execução.

2.2.3.5 `IndexedDB`

Devido à natureza do problema abordado neste trabalho e das AWPs, é vital a utilização de um banco de dados que permita o uso com conexões ruins ou inexistentes. Dessa forma, é necessário utilizar um método de armazenamento que seja capaz de gerenciar grandes quantidades de dados de maneira eficaz no lado do cliente, para que ele consiga utilizar funções dinâmicas da aplicação mesmo desconectado, como exemplo registrar um novo boletim de ocorrência mesmo na ausência de conexão. Uma solução para esse problema é a utilização do `IndexedDB`.

Conforme Hume (2017, p. 123), "`IndexedDB` é uma API para armazenamento de quantidades significativas de dados no lado do cliente e faz a utilização de índices para conseguir uma alta performance nas buscas."

É utilizado para o armazenamento de dados em um formato de objetos JSON. Portanto, se trata de um banco de dados não relacional, orientado a documentos. Através do `IndexedDB` é possível armazenar dados da ocorrência temporariamente, em caso de

perda de conexão com a *Internet*. Quando a conexão com o servidor voltar, esse dado pode ser enviado. Dessa forma, é sincronizado o lado do cliente com o servidor. O contrário também pode ser feito, ou seja, obter um dado da rede e guardar para estar disponível posteriormente para interação, mesmo quando estiver *offline*.

2.3 Teste de *Software*

A evolução do ambiente *Web* e o avanço de sua capacidade de oferecer aplicações cada vez mais robustas, somado ao aumento da dependência global por serviços oferecidos via rede, tem forçado uma necessidade de sistemas com alta confiabilidade e qualidade. Isso decorre do fato de que qualquer erro, ou queda na disponibilidade das funcionalidades de um sistema *Web* pode gerar prejuízos enormes para qualquer entidade dependente dele.

Nesse contexto, existe uma necessidade grande da elaboração de testes de *software* eficazes, a fim de descobrir falhas, validar funcionalidades e no geral, garantir o bom funcionamento esperado do *software* implementado.

Para Sommerville (2011, p. 144), o processo de teste de *software* tem dois objetivos distintos:

- Demonstrar ao desenvolvedor e ao cliente que o *software* atende aos requisitos.
- Descobrir situações em que o software se comporta de maneira incorreta, indesejável ou de forma diferente do especificado.

Desta maneira, as AWPs representam um desafio singular ao se projetar métodos de teste para a aplicação desenvolvida. Independentemente de se tratar de uma AW, sua ênfase é muito clara em executar em dispositivos móveis. Logo, as particularidades existentes nesses aparelhos, em especial as descritas no decorrer da seção 2.2, devem ser consideradas durante a etapa de testes e validação do sistema.

2.3.1 Teste Funcional

Consoante a Delamaro, Maldonado e Jino (2016), teste funcional é uma técnica utilizada para projetar casos de teste onde o programa é submetido a todas as possíveis entradas durante a utilização. Em seguida, é feita uma avaliação das saídas geradas e se estão conforme o esperado e ainda atendendo os requisitos. Nessa técnica, os detalhes de implementação não são considerados e o software é avaliado de acordo com a visão do usuário.

Uma opção bastante eficiente para a realização desses testes é utilização de uma ferramenta de testes automatizados baseado na interface gráfica de usuário. Com base

em Alegroth, Feldt e Ryrholm (2014), a técnica mais comum presente nessa categoria de ferramentas é a de Capturar e Repetir, onde as entradas do usuário ao utilizar uma funcionalidade são imitadas e capturadas, como a navegação e inserção de dados. Na sequência, os passos gravados são repetidos automaticamente, permitindo maior eficiência e agilidade para o processo de validação das funcionalidades do sistema.

Alegroth, Feldt e Ryrholm (2014) ressalta que essas ferramentas dependem dos componentes da interface gráfica não serem alterados, ou seja, os casos de teste projetados para uma aplicação podem deixar de funcionar se uma nova versão do sistema que altera um componente envolvido nos passos gravados for implementada.

2.3.2 Teste de Desempenho

Conforme descrito por Delamaro, Maldonado e Jino (2016), os testes de desempenho são focados em identificar e eliminar possíveis gargalos que possam provocar lentidão ou funcionamento de forma inesperada da aplicação.

Em um contexto de desenvolvimento para dispositivos móveis, muitas métricas usadas em testes de aplicações tradicionais acabam não se aplicando em consequência das particularidades descritas anteriormente, o que eleva a demanda por maior planejamento e individualidade na hora de criar esse tipo de teste.

Delamaro, Maldonado e Jino (2016) ainda relata que o desempenho deve considerar a harmonia entre o funcionamento da aplicação e as necessidades de seus usuários e a produtividade para o usuário final.

3 Desenvolvimento

Este capítulo descreve o processo de desenvolvimento do sistema. A seção 3.1 demonstra o SisGera. Na sequência, a seção 3.2 aborda as escolhas tecnológicas para o trabalho. Durante a seção 3.3, é apresentado o processo de desenvolvimento da aplicação. Finalmente, na seção 3.4 é exibido os testes de sistemas realizados.

3.1 SisGera

Conforme o conteúdo abordado durante a introdução, o SisGera é uma AW desenvolvida durante o trabalho de Arantes (2018) e Oliveira (2018) para atender as necessidades observadas das organizações de Bombeiros Voluntários de São Domingos do Prata e Bombeiros Voluntários de Barão de Cocais. A Figura 5 exibe o logo do SisGera.

Figura 5 – Logo do SisGera

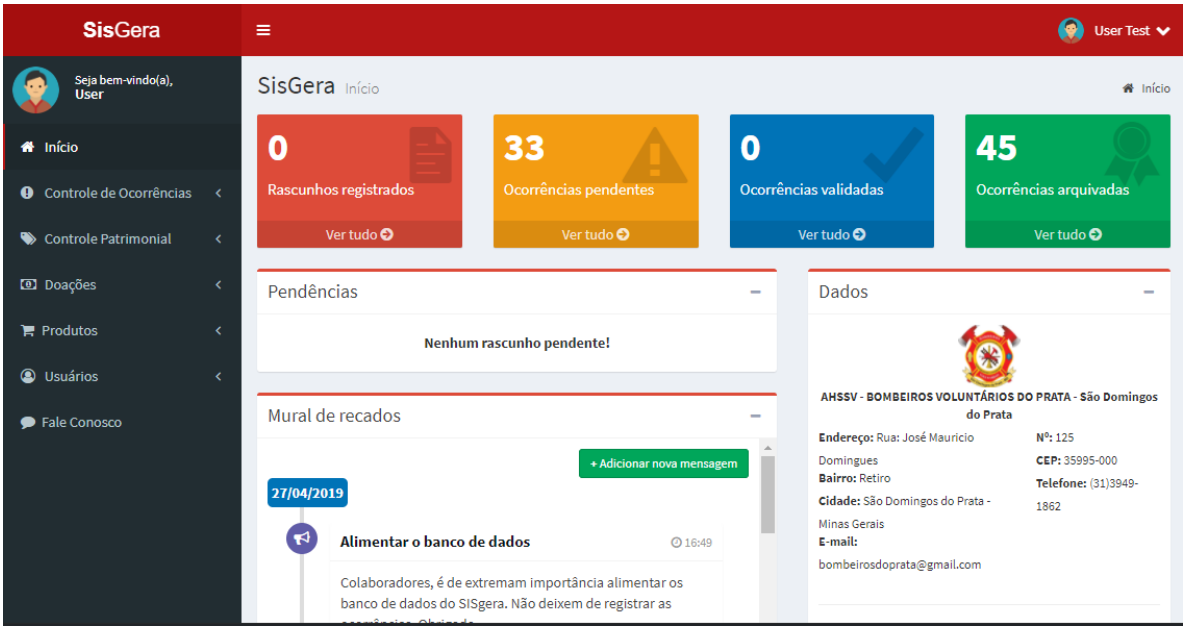
Fonte: https://www.sisgera.com.br/. Acesso em: 01 de jul. de 2021.

Esse sistema se encontra em uso pelos bombeiros, atendendo suas principais necessidades, conseguindo assim, facilitar suas tarefas administrativas e operacionais, tornando as rotinas diárias mais eficientes. A Figura 6, apresenta a tela principal do SisGera exibido em um computador.

3.2 Escolhas tecnológicas

Esta seção aborda as escolhas das ferramentas e tecnologias que foram utilizadas durante o desenvolvimento deste trabalho.

Figura 6 – Tela principal do SisGera.



Fonte: https://www.sisgera.com.br/. Acesso em: 01 de jul. de 2021.

3.2.1 *Frameworks*

Atualmente, é exigido dos desenvolvedores de softwares uma alta padronização e qualidade do programa. Além disso, prazos de entrega mais curtos demandam uma maior velocidade no desenvolvimento. Isso pode trazer problemas e erros futuros que poderão ser de difícil resolução, devido a baixa qualidade do código. Com isso, se mostra necessário a utilização de auxílios para sanar esses defeitos.

Baseando-se em Schmidt, Gokhale e Natarajan (2004), um *framework* é um conjunto integrado de artefatos de software (como classes, objetos, e componentes) que colaboram para fornecer um arquitetura reusável para uma família de aplicativos relacionados.

Os *frameworks* surgem para poupar tempo e organizar códigos durante o desenvolvimento. Seu uso pode facilitar a correção de erros e a manutenção do software, além de padronizar a codificação, facilitando que vários programadores trabalhem em um mesmo código, economizando ainda mais tempo.

3.2.2 `Bootstrap`

Com esses benefícios exemplificados, se mostra essencial a utilização de *frameworks* que auxiliem na programação do estilo e a usabilidade do sistema para o desenvolvedor. O `Bootstrap` possui o código aberto, e por meio do HTML, CSS e JS, fornece componentes pré-construídos, elementos de diversos tamanhos, interfaces responsivas e ferramentas JS

para facilitar o desenvolvimento de qualquer AW.[1] Desenvolvido por Mark Otto e Jacob Thornton, engenheiros do *Twitter* e lançado em 2011, é a *framework* mais utilizada para a criação de uma AW.

3.2.2.1 `AdminLTE`

O `AdminLTE` é um *template* de código aberto utilizado para a elaboração de painéis de administração e controle em aplicações Web, construido embasado no `Bootstrap`.[2] Possui uma *interface* gráfica responsiva e atraente para o usuário e pode ser customizado, agregando durante o processo de desenvolvimento da aplicação.

3.2.3 `Dexie.JS`

O `IndexedDB` nativo possui uma alta complexidade de seus códigos, dificultando criar os comandos para as consultas e transições no banco de dados necessárias para a aplicação. Criado por David Fahlander, em 2014, o `Dexie.JS` é um empacotamento minimalista para o `IndexedDB`. Ao utilizar o `Dexie.JS`, é possível economizar tempo de codificação e, ao mesmo tempo, manipular o banco de dados local de maneira mais eficiente. Além disso, consegue fornecer um melhor manuseio das respostas e de possíveis erros originados de suas operações. [3]

3.2.4 Laravel

O Laravel é um *framework* PHP com o objetivo de facilitar e agilizar a implementação das funcionalidades e tarefas comuns nos sistemas *Web*, tais como a autenticação, construção das páginas *Web*, e um sistema eficiente de roteamento dessas páginas.

Finalmente, o Laravel inclui o `Eloquent`, um mapeamento objeto-relacional que fornece uma melhor interação com o banco de dados, simplificando as operações e o manuseio das tabelas, como a inserção, consulta e a remoção de dados, promovendo uma maior eficiência para o processo de desenvolvimento em qualquer AW. [4]

3.3 Desenvolvimento da Aplicação

Durante o decorrer desta seção, será apresentado o processo de desenvolvimento da versão AWP do SisGera, contendo os requisitos iniciais, o mapeamento do banco de dados e a implementação das funcionalidades.

[1] Disponível em: https://www.getbootstrap.com/. Acesso em 04 de jul. de 2021
[2] Disponível em: https://www.adminlte.io/. Acesso em 04 de jul. de 2021
[3] Disponível em: https://www.dexie.org/. Acesso em 02 de jul. de 2021
[4] Disponível em: https://www.laravel.com/docs/8.x/. Acesso em 04 de jul. de 2021

3.3.1 Requisitos

De acordo com Pressman e Maxim (2016, p: 132), o processo de entendimento dos requisitos de um sistema é chamado de Engenharia de Requisitos, a qual "é uma ação da Engenharia de Software importante que se inicia durante a atividade de comunicação e continua na de modelagem. Ela deve ser adaptada às necessidades do processo, do projeto, do produto e das pessoas que estão o realizando o trabalho."

Para o desenvolvimento da versão AWP do SisGera, com o objetivo de compreender rapidamente os requisitos dos bombeiros, o modelo história de usuários, conforme descrito por Longo e Silva (2014) foi utilizado. Esse modelo é estruturado como se segue:

- **Como um ...** (Quem? – Papel/ator)
- **eu quero ...** (O que? – Funcionalidade planejada)
- **de modo que ...** (Porque? – Motivação/Valor que pode trazer)

As histórias de usuário para as novas funcionalidades do SisGera, desenvolvidas neste trabalho estão apresentadas abaixo:

Registrar ocorrência offline

Como um usuário

eu quero registrar uma ocorrência quando estiver sem conexão à internet

de modo que ela fique salva no meu dispositivo.

Ocorrência não sincronizada

Como um usuário

eu quero visualizar se alguma ocorrência precisa ser sincronizada

de modo que eu fique ciente da necessidade de me conectar a internet.

Sincronizar ocorrência quando online

Como um usuário

eu quero enviar uma ocorrência quando estiver conectado a internet

de modo que ela seja sincronizada com o banco de dados do servidor.

Histórico de Sincronias

Como um usuário

eu quero visualizar meu histórico de sincronias

de modo que eu possa ter um controle das ocorrências que foram enviadas para o servidor.

Notificação de Sincronia

Como um usuário

eu quero receber um aviso quando a ocorrência for sincronizada com o servidor

de modo que eu fique ciente que a sincronização da ocorrência com o banco de dados foi concluída.

3.3.2 Banco de Dados

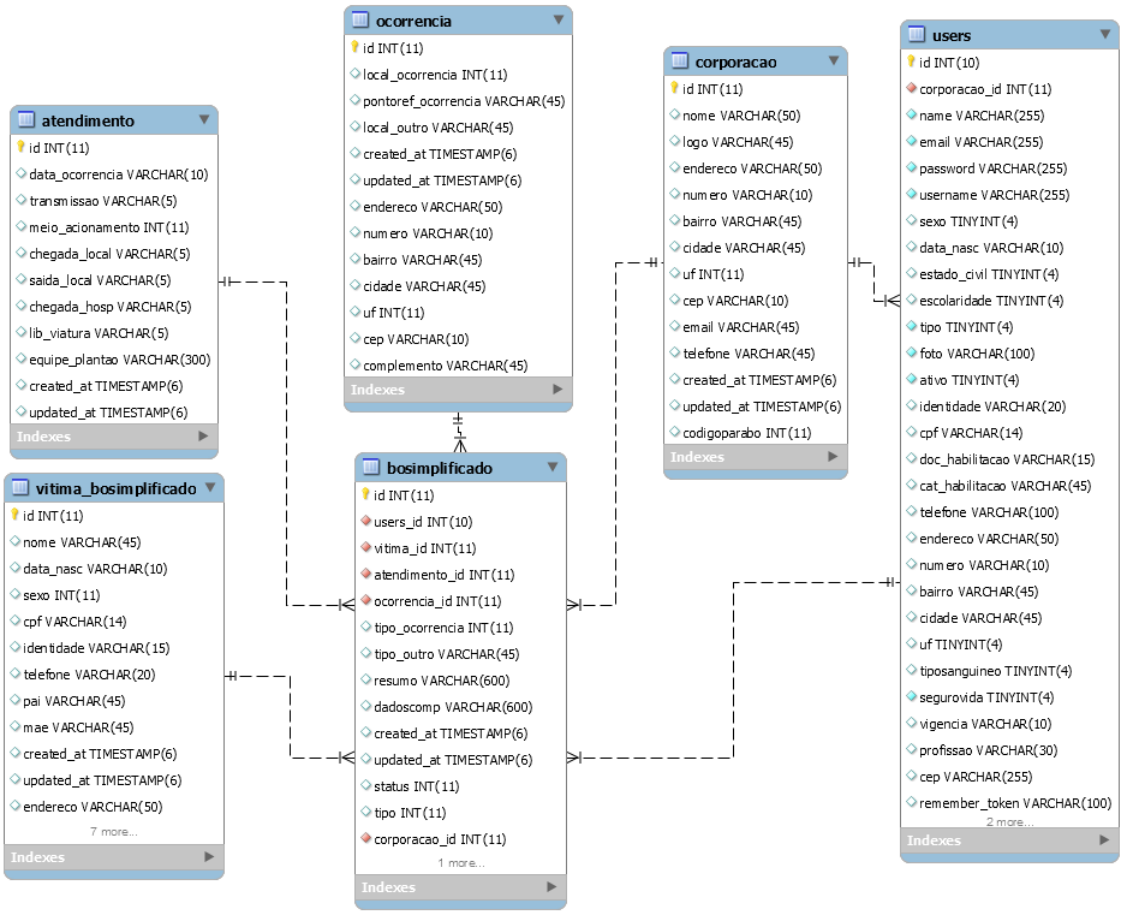
Durante o desenvolvimento de seu trabalho, Arantes (2018) optou por utilizar o banco de dados relacional `MySQL`[5] para o armazenamento dos dados estruturados originados dos boletins de ocorrências registrados pela aplicação. A Figura 7 ilustra o diagrama Entidade-Relacionamento referente a versão adaptada do boletim de ocorrências modelo simplificado do SisGera atual.

Conforme informado no capítulo 1, devido a necessidade do registro de ocorrências sem conexão à Internet, torna crucial o uso de um banco de dados para o armazenamento temporário dos dados da ocorrência no dispositivo do usuário e que possa ser facilmente acessado pelo navegador. Uma opção para a solução desse problema seria o uso da `Web SQL`, uma interface de programação para o gerenciamento de dados relacionais integrada aos navegadores. Contudo, essa tecnologia foi depreciada e não recebe mais nenhum tipo de manutenção, significando que a qualquer momento poderia parar de funcionar parcial ou totalmente. Isso significaria o seu uso conflitante com a definição das AWPs de fornecer suporte e funcionalidades de modo progressivo, sem comprometer a utilização, mesmo em dispositivos ultrapassados.[6]

Consequentemente, a alternativa escolhida para o gerenciamento de dados localmente foi o `IndexedDB`. Ressaltando o que foi introduzido durante o capítulo 2, se trata de uma interface de programação integrada aos navegadores que funciona como um banco de dados orientado a objetos JSON para o armazenamento de dados estruturados. Por serem dois modelos conflitantes de banco de dados, foi necessário realizar um mapeamento das tabelas relacionais para objetos, de forma que os dados salvos temporariamente no `IndexedDB` sejam posteriormente enviados para o banco de dados relacional do servidor. Com isso, as mudanças mais relevantes foram as seguintes:

[5] Disponível em: https://www.mysql.com/. Acesso em 03 de jun. de 2021
[6] Disponível em: https://www.w3.org/TR/webdatabase/. Acesso em 20 de jul. de 2021

Figura 7 – Diagrama Entidade-Relacionamento para o boletim modelo simplificado.



Fonte : Elaborado pelo autor (2021) e adaptado de Arantes (2018)

- União das tabelas diretamente ligadas a cada modelo de boletim em um único objeto, por exemplo, o boletim de ocorrência simplificado relacional possui tabelas para o atendimento, ocorrência, vítima e do boletim contendo o restante das informações e as chaves estrangeiras relacionadas. Todos os campos foram transformados em um objeto denominado "BOSimplificado".

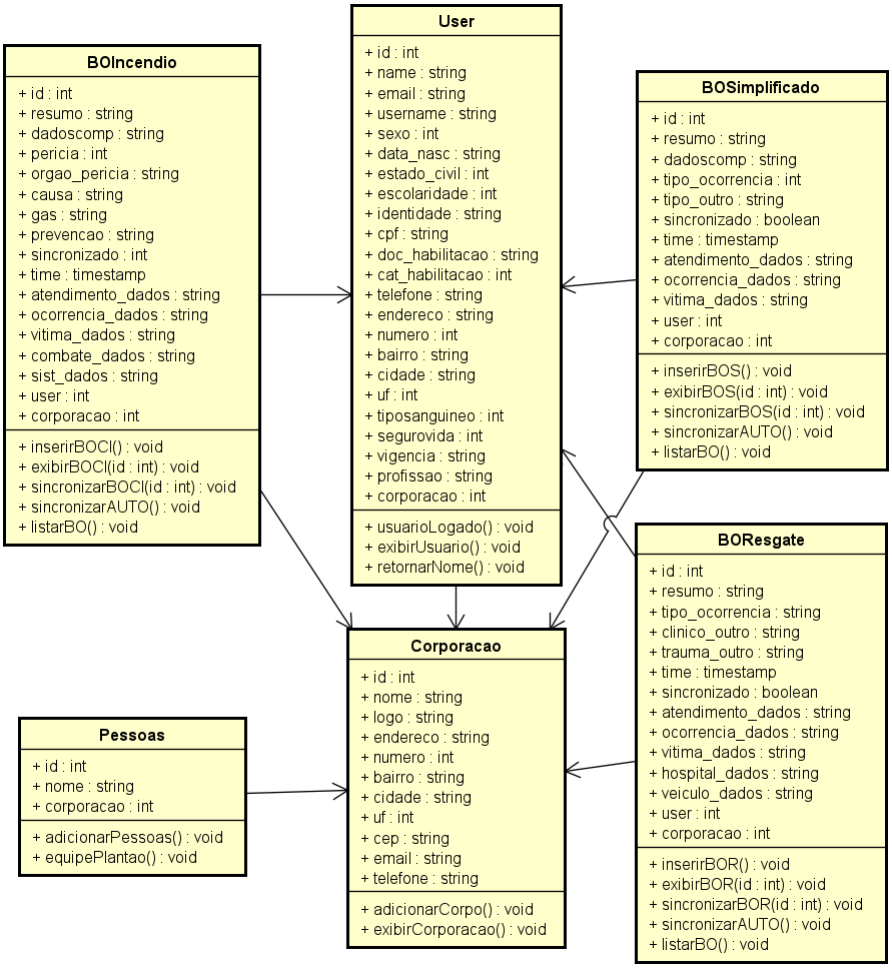
- Remoção de dados desnecessários para o funcionamento da AWP, como os relacionados a autenticação de usuário.

- Adição de um objeto "Pessoas", referente ao nome e id dos bombeiros da mesma corporação do usuário logado, para ser utilizado no campo "Equipe de Plantão"no momento do registro da ocorrência.

- O objeto "Corporação" e "User", em contramão do banco de dados do servidor que armazena todos os registrados, apenas contém as informações relacionadas ao bombeiro logado e sua corporação.

Essas adaptações e o `IndexedDB` resultante utilizada pela aplicação desenvolvida neste trabalho estão representados em uma adaptação do modelo de diagrama de classes exibido na Figura 8.

Figura 8 – Diagrama de Classes para o IndexedDB da versão AWP do SisGera.



Fonte : Elaborado pelo autor (2021)

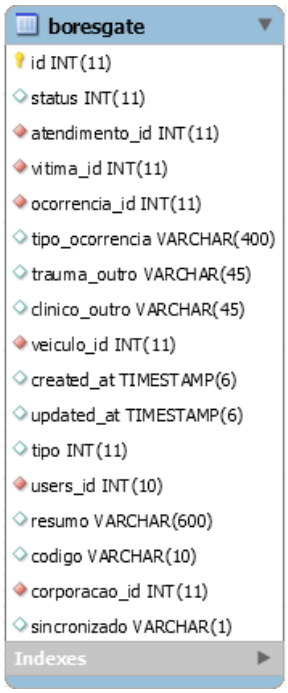
3.3.2.1 Histórico de Sincronias

Uma única alteração foi realizada no banco de dados relacional existente. Com o objetivo de atender o requisito "Histórico de Sincronias", foi feita a adição de um campo, denominado "sincronizado", servindo para sinalizar que o boletim foi originado a partir da AWP. Esse campo foi inserido nas três tabelas centrais de cada tipo de boletim. A Figura 9 ilustra a tabela resultante do modelo de boletim Resgate e Salvamento por meio de um diagrama Entidade-Relacionamento.

Logo, para exibir o histórico de sincronias, a aplicação busca por boletins sinalizados como sincronizados e utiliza do campo `created_at`, padrão do Laravel, para fornecer a

data e hora de sincronia para o usuário.

Figura 9 – Diagrama Entidade-Relacionamento da tabela `boresgate` resultante.



Fonte : Elaborado pelo autor (2021) e adaptado de Arantes (2018)

3.3.3 Funcionalidades

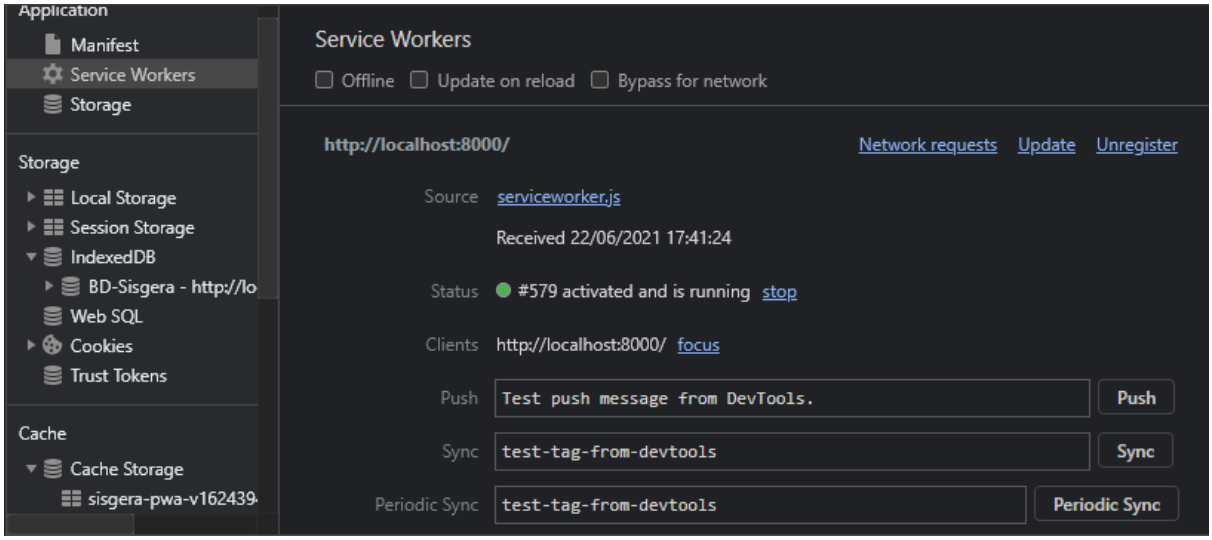
Nesta subseção, serão apresentadas as funcionalidades implementadas e diagramas ilustrativos que formam a base da aplicação desenvolvida neste trabalho.

3.3.3.1 Versão AWP do SisGera

Com o objetivo de facilitar a integração com o restante do sistema e a minimização do impacto causado, o desenvolvimento da versão AWP do SisGera foi visado em criar uma aplicação em paralelo ao sistema já em uso.

Devido ao motivo de que a página principal do SisGera pode ser utilizada por usuários não pertencentes aos bombeiros voluntários, toda AWP desenvolvida necessita de que o bombeiro realize a autenticação no sistema. Isso ainda adiciona uma camada extra de segurança para a AWP implementada. Após a confirmação, o SW é registrado no dispositivo e inicia o processo descrito na seção 2.2.3.2, onde os arquivos necessários para o funcionamento, especificamente os JS, CSS e de interface são armazenados na `Cache`. A Figura 10 exibe o SW implementado registrado e ativado no navegador Google Chrome, ainda é possível observar o `IndexedDB` contendo o "BD-Sisgera" e a interface `Cache` em uso.

Figura 10 – SW do SisGera ativado no navegador.



Fonte : Elaborado pelo autor (2021)

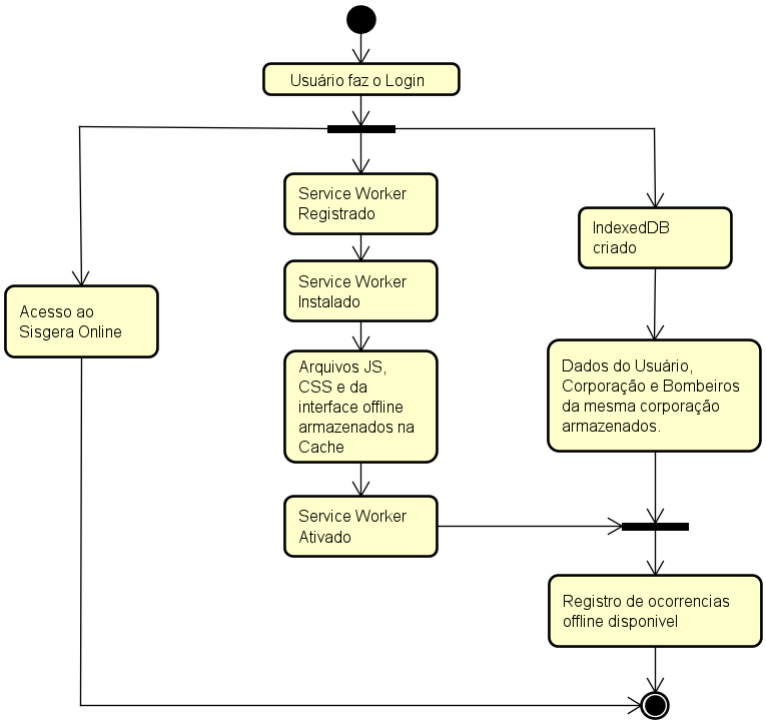
Enquanto isso, o `IndexedDB` é criado e os dados relevantes como o usuário, corporação e bombeiros da mesma corporação são salvos. Após a conclusão desses processos, na versão AWP, o registro de ocorrências *offline* está disponível e, em paralelo, o sistema SisGera convencional pode ser utilizado da mesma maneira. Esse processo é ilustrado na Figura 11.

3.3.3.2 `Cache` implementada

Conforme apresentado anteriormente, a `Cache` desenvolvida para o SisGera contém os arquivos necessários para a funcionalidade "Registro de ocorrências *offline*" estar habilitada corretamente. Logo, foram armazenados os recursos que pertencem a interface gráfica da aplicação, como os arquivos HTML, CSS, e imagens, além dos JS desenvolvidos para a validação dos formulários e principalmente, o banco de dados desenhado.

As estratégias escolhidas neste trabalho foram a de "Na Instalação" para o armazenamento dos recursos, e *Fallback* Genérico para servir ao usuário. Essa escolha se deu principalmente para criar um ambiente separado para o SisGera AWP, e facilitar sua integração com o sistema existente, possibilitando o uso primário do SisGera diretamente pelo servidor, enquanto existe um plano de contingência, caso seja necessário registrar ocorrências sem a conexão. A Figura 12 ilustra a `Cache` criada contendo os arquivos necessários para o funcionamento dos requisitos implementados neste trabalho.

Figura 11 – Diagrama de atividades da versão AWP do SisGera.



Fonte : Elaborado pelo autor (2021)

3.3.3.3 Registro de ocorrências *offline*

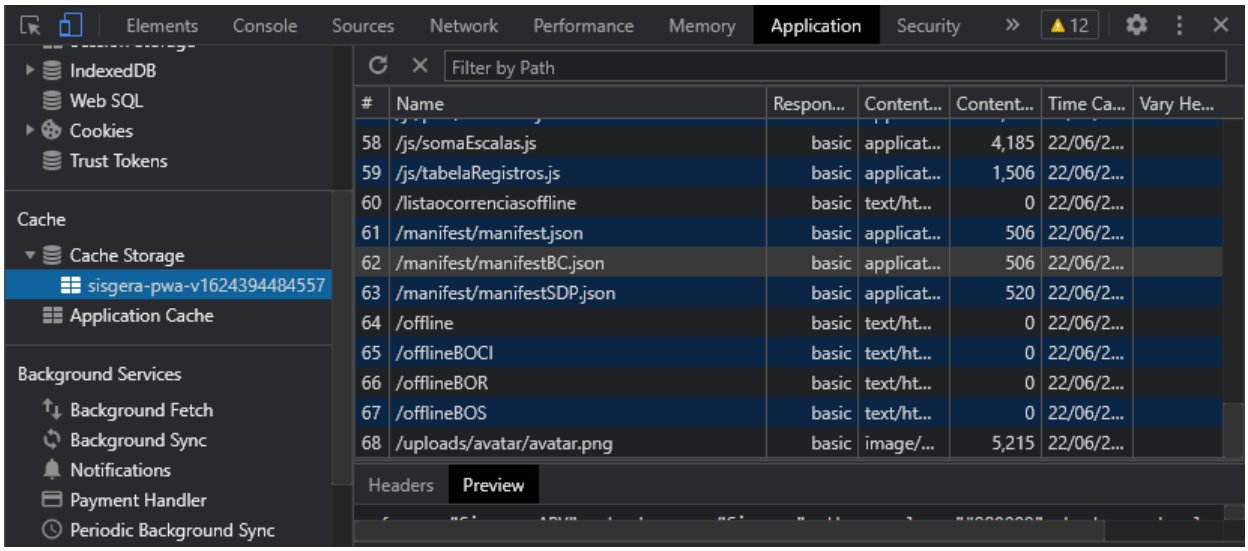
A motivação principal deste trabalho é justamente possibilitar o uso do sistema SisGera, especificamente a funcionalidade de registro de ocorrências na ausência de *Internet*. Dessa forma, foram desenvolvidos formulários baseados naqueles criados por Arantes (2018), já que, como os dados trabalhados seriam os mesmos e o sistema já se encontra em uso como parte do convívio diário dos bombeiros, não seria adequado mudar a interface que se encontra adaptada.

A Figura 13 ilustra o processo do registro quando conectado à *Internet*, desenvolvido por Arantes (2018) e a alternativa implementada onde os dados são salvos temporariamente no `IndexedDB` e aguardam o processo de sincronia.

3.3.3.4 Sincronização de ocorrências

A funcionalidade apresentada anteriormente não teria muita utilidade se os dados registrados não pudessem ser enviados para o servidor, onde eles seguiriam os procedimentos atuais, como a validação do chefe de equipe e sua arquivação posterior. Com isso, é necessário enviar os boletins para o banco de dados do servidor quando a conexão estiver disponível, e esta funcionalidade foi chamada de sincronia de ocorrências.

Figura 12 – Cache do SisGera e arquivos armazenados.



Fonte : Elaborado pelo autor (2021)

Esse processo foi desenvolvido a partir da utilização do *Asynchronous JavaScript e XML* (AJAX) para passar os dados salvos do `IndexedDB` para o Laravel usando o método de requisição POST para inserção dos dados nas tabelas do servidor, assim como desenvolvido por Arantes (2018). Esse processo é considerado seguro devido ao uso do HTTPS e o *token* de proteção *Cross-site Request Forgery* (CSRF) do Laravel.

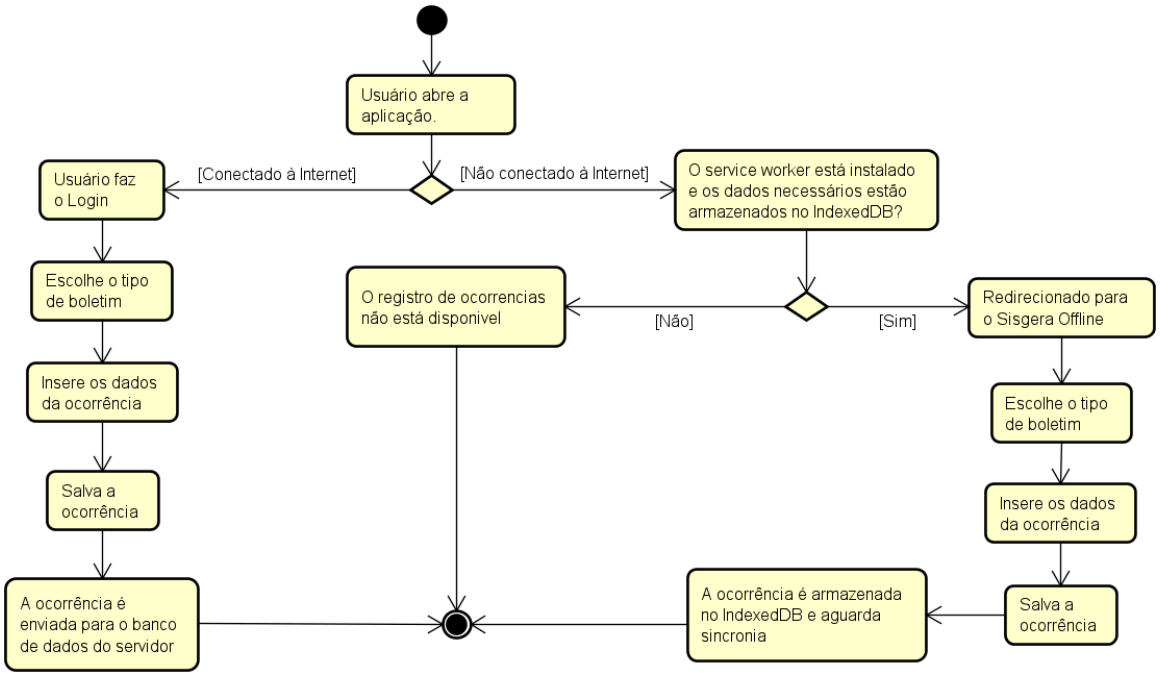
O *token* CSRF é utilizado para combater falsificações de solicitações entre sites, uma ameaça onde comandos não autorizados são executados por um terceiro malicioso que utiliza um usuário autenticado, ou seja, que o sistema confia. Esse *token*, o qual uma aplicação maliciosa não tem acesso, é verificado pelo Laravel em cada uma das requisições, garantindo a integridade e confiabilidade das operações e solicitações realizadas.[7]

Assim, foram implementados dois modelos possíveis para essa funcionalidade onde, após o bombeiro abrir a aplicação, enquanto conectado à *Internet*, existe a possibilidade da sincronia ser feita automaticamente pelo sistema. Por outro lado, por opção do bombeiro, ele pode manualmente abrir a tela de sincronia e selecionar os boletins para serem sincronizados. A Figura 14 ilustra essas duas alternativas para o processo de sincronização resultantes.

Houve ainda, uma tentativa não sucedida para uma nova alternativa onde a aplicação não necessitaria ser aberta e os dados seriam enviados assim que uma conexão boa fosse detectada. No entanto, como as AWs dependem do navegador estar aberto para o funcionamento, a única opção seria o uso de um *worker* como o SW. Como abordado

[7] Disponível em: https://www.laravel.com/docs/8.x/csrf. Acesso em 04 de jul. de 2021.

Figura 13 – Diagrama de atividades para o processo de registro de ocorrências *offline*



Fonte : Elaborado pelo autor (2021)

no capítulo 2, o SW funciona em segundo plano no dispositivo, em especial, a interface de programação *Background-sync*, criada originalmente para permitir a atualização de recursos da `Cache` ao registrar um evento no SW que, ao detectar a volta de conexão, realiza essa atualização. [8] Porém, esses *scripts* não têm acesso direto ao restante da aplicação, o que impossibilita a comunicação com o Laravel protegida por um *token* de proteção CSRF, e como sua remoção traria os riscos abordados previamente, essa alternativa acabou sendo descartada neste trabalho.

3.4 Testes do Sistema

No decorrer desta seção, serão demonstradas as abordagens de testes utilizados neste trabalho.

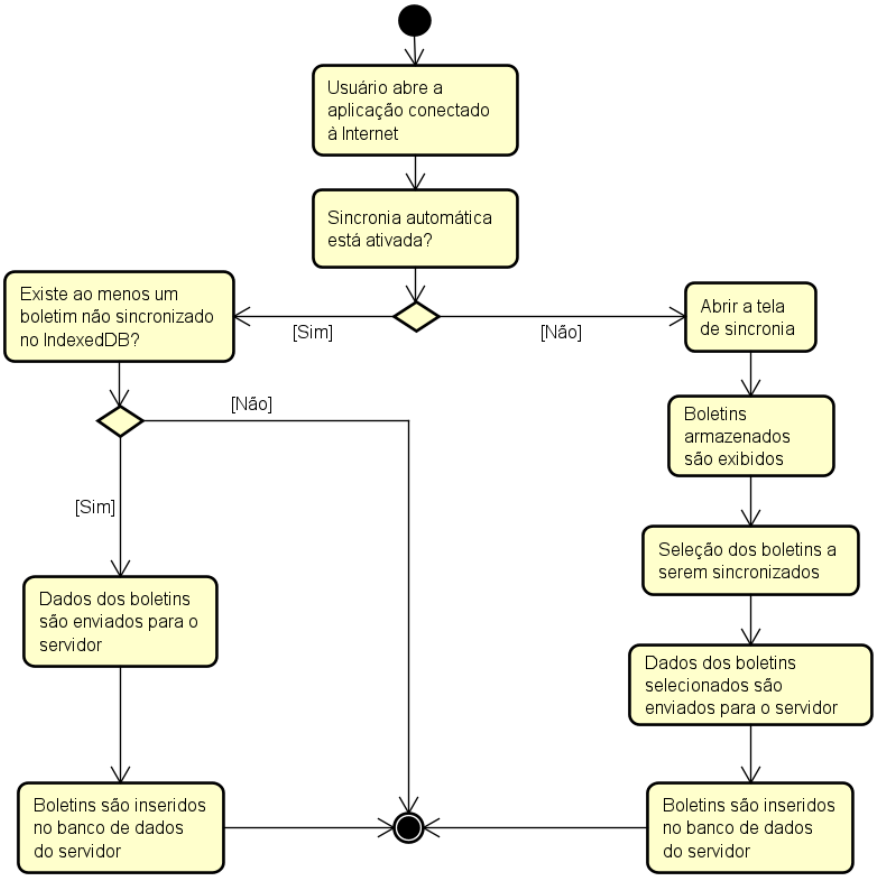
3.4.1 Testes Funcionais

Os testes funcionais da aplicação foram realizados utilizando a ferramenta `Katalon Recorder`, uma extensão disponibilizada para os navegadores Chrome, Firefox e Edge que permite a automatização dos testes funcionais em aplicações *Web*.[9]

[8] Disponível em: https://www.developers.google.com/web/updates/2015/12/background-sync. Acesso em 04 de jul. de 2021

[9] Disponível em: https://www.katalon.com/katalon-recorder-ide/; Acesso em 19 de jul de 2021

Figura 14 – Diagrama de atividades para o processo de sincronização de ocorrências.



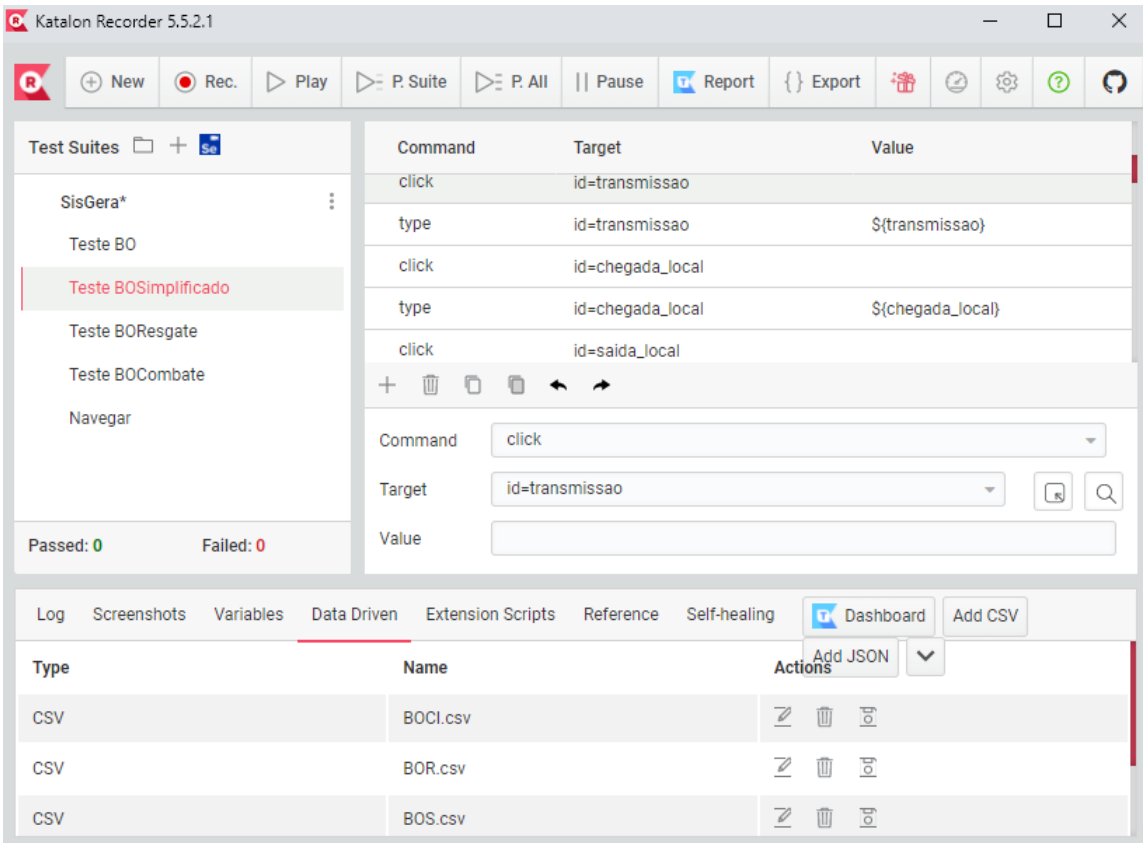
Fonte : Elaborado pelo autor (2021)

No contexto deste trabalho, os testes foram gerados por meio de uma gravação dos passos realizados pelo usuário, incluindo entrada de dados, ao acessar as funcionalidades implementadas. Logo, o `Katalon Recorder` permite a geração de um caso de teste automatizado que registra esse passo a passo para a execução da funcionalidade, facilitando essa etapa de validação do sistema. Essa ferramenta permite ainda o uso de arquivos externos `.CSV` ou JSON para agilizar a avaliação de entrada de dados em ampla escala. Para avaliar as funcionalidades desenvolvidas, a ferramenta `Mockaroo` foi utilizada para criar arquivos `.CSV`. Essa ferramenta gera grandes quantidades de dados aleatórios, porém realistas, pois se baseiam em regras pré-definidas pelo desenvolvedor.[10]

A Figura 15 exibe o caso de teste utilizado para avaliar o processo de registro de ocorrências *offline*.

[10] Disponível em: https://www.mockaroo.com/. Acesso em 22 de jul de 2021.

Figura 15 – Testes criados pelo `Katalon Recorder`.



Fonte : Elaborado pelo autor (2021)

4 Resultados

Esse capítulo apresenta os resultados alcançados neste trabalho, como as telas e funcionalidades implementadas e a avaliação das características e conceitos de AWP.

4.1 Apresentação da aplicação

No decorrer desta seção, apresenta-se as telas e as descrições das funcionalidades desenvolvidas neste trabalho. Como se trata de um sistema existente, onde os usuários estão adaptados com a interface, o mínimo de alterações relacionados ao *layout* das páginas foi realizado. Com o objetivo de demonstrar o potencial da arquitetura no contexto de desenvolvimento para dispositivos móveis, entre as seções 4.1.1 e 4.1.3 as imagens das telas serão originadas de ambientes `Android` e `iOS`, exemplificando assim o seu uso nos dois sistemas de *smartphones* com o uso mais difundido atualmente.

4.1.1 Instalação da AWP

Assim como nas aplicações nativas, é possível a instalação de uma AWP. Conforme indicado na seção 2.2.1, as aplicações nativas oferecem essa funcionalidade majoritariamente por meio das lojas de aplicativos de seu sistema operacional. Por dependerem do navegador, as AWP não possuem esse recurso. Assim, para instalar a aplicação, é necessário, no primeiro acesso, abrir o site no navegador e acessar a opção de "Adicionar o *app* á tela inicial".

Para promover o uso da AWP, o `Chrome` alerta o usuário dessa possibilidade durante a primeira visita. Esse alerta no entanto, demanda que a aplicação possua as seguintes características:[1]

- Ser servida através do HTTPS.
- Possui um arquivo *manifest.json* com as informações da aplicação.
- Possui ícones de tamanho apropriado a ser exibido na área de trabalho.
- O SW registrado para fornecer funcionalidades *offline*.

Como a aplicação desenvolvida cumpre esses requisitos, o navegador permite a sua instalação, como é exemplificado na Figura 16, onde a Figura 16a exibe o alerta do

[1] Disponível em: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Add_to_home_screen

Figura 16 – Abas para a instalação da AWP na tela inicial

(a) Alerta em um dispositivo `Android` aberto no navegador Chrome

(b) Janela do navegador Safari em um dispositivo `iOS`

Fonte : Elaborado pelo autor (2021)

Chrome para a possibilidade de instalação e a Figura 16b demonstra a janela do navegador Safari de mesmo objetivo.

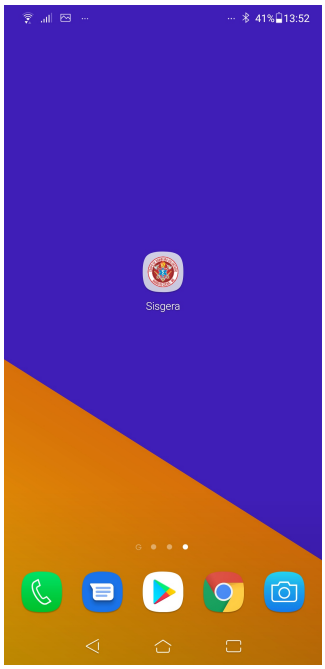
4.1.2 Ícone na área de trabalho

Para se equiparar as aplicações nativas, é importante permitir o acesso rápido por meio de um ícone na área de trabalho do dispositivo. A Figura 17a exibe o ícone do SisGera AWP em um aparelho `Android`, e a Figura 17b em um `iOS`, demonstrando que a aplicação desenvolvida cumpre esse requisito exatamente como uma aplicação nativa.

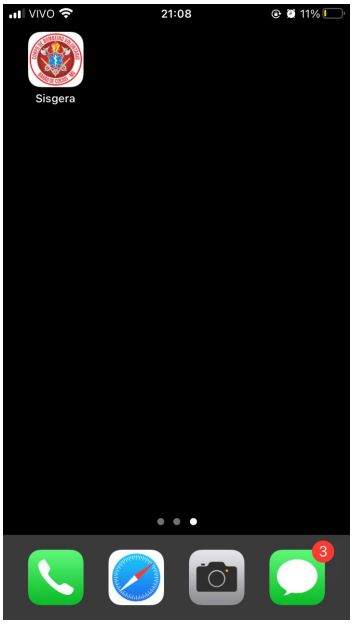
4.1.3 Tela inicial

Ao abrir a aplicação sem conexão à *Internet*, a tela inicial exibe primeiramente um aviso da falta de rede. Em sequência, caso exista algum boletim não sincronizado ainda salvo localmente, é exibido um contêiner indicando quantos boletins e uma lista resumida das informações deles. Finalmente, os dados do último usuário logado, relacionados ao bombeiro e a sua corporação pertencente são apresentados. A Figura 18a exibe a tela de início e suas informações em um ambiente `Android` e a Figura 18b exibe sua versão em

Figura 17 – Ícone do SisGera AWP



(a) Ícone do SisGera na área de trabalho de um dispositivo `Android`



(b) Ícone do SisGera na área de trabalho de um dispositivo `iOS`

Fonte : Elaborado pelo autor (2021)

um dispositivo `iOS`.

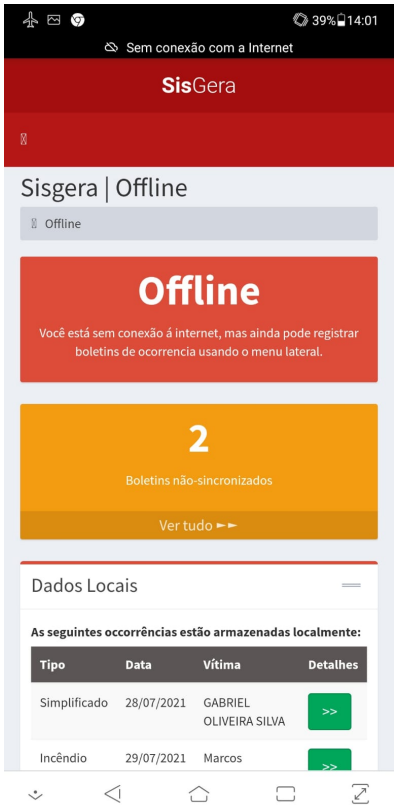
4.1.4 Registro de ocorrências

Essa é principal funcionalidade da aplicação implementada. Novamente, o *layout* existente é aproveitado e o bombeiro pode preencher o formulário com as informações necessárias, da mesma maneira que ele já está familiarizado. Após a conclusão da entrada dos dados pelo bombeiro, a ocorrência é armazenada no *IndexedDB* e aguarda o processo de envio para o servidor. A Figura 19 exibe os três modelos de boletins utilizados pelo SisGera, ordenados respectivamente em Combate a Incêndio, Resgate e Salvamento e Simplificado.

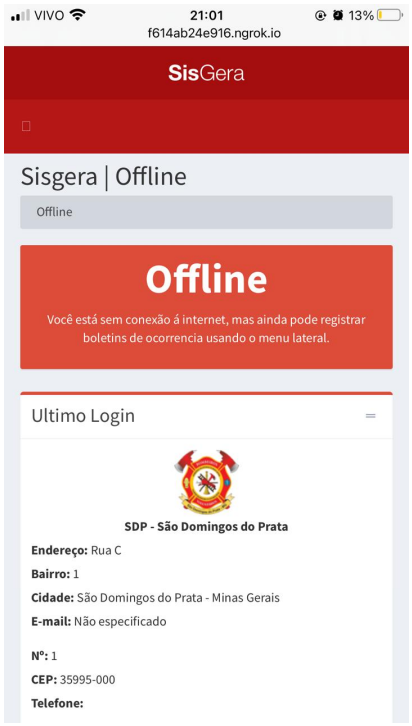
4.1.5 Ocorrências locais

Para melhorar a interação com o usuário, as ocorrências armazenadas no dispositivo do bombeiro podem ser facilmente acessadas por meio da tela de lista de ocorrências locais. Essa tela contém as informações principais e o acesso para o detalhe do boletim, para que o usuário possa consultar os dados posteriormente. A Figura 20 apresenta a lista de ocorrências salvas localmente.

Figura 18 – Tela inicial SisGera AWP.



(a) Tela inicial em um dispositivo `Android`



(b) Tela inicial em um dispositivo `iOS`

Fonte : Elaborado pelo autor (2021)

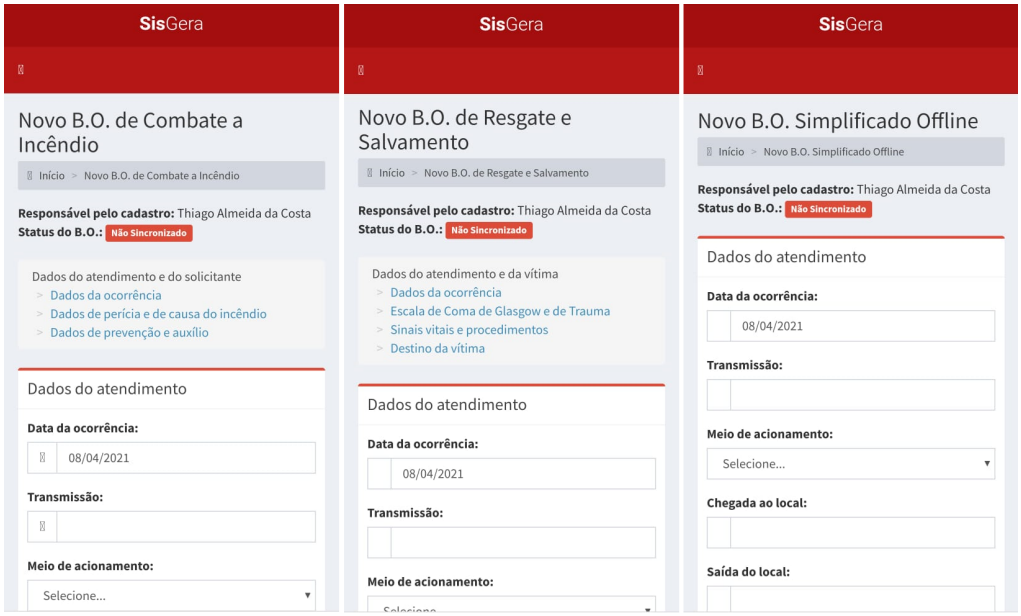
4.1.6 Sincronia e histórico

A página de sincronia tem o objetivo de controlar as operações envolvendo o envio dos dados para o servidor e o controle do histórico das sincronias. Essa tela só pode ser acessada diretamente pelo servidor. Caso exista algum boletim disponível para sincronia, o usuário é informado e pode selecionar os que deseja enviar. Em seguida, é exibida uma lista de histórico das sincronias, com a data e hora do envio, conforme descrito na seção 3.3.2.1. Assim, o bombeiro pode ter esse controle e organizar melhor suas ações. Por fim, é possível marcar a opção de sincronia automática para que não seja necessário selecionar e enviar manualmente os boletins que o usuário deseje sincronizar.

O histórico das sincronias foi criado para facilitar o controle dos boletins originados pela AWP, principalmente nessa fase inicial de utilização, onde problemas imprevistos podem acontecer.

A Figura 21 demonstra a tela de controle de sincronias de ocorrências da aplicação.

Figura 19 – Formulários dos três modelos de boletim de ocorrência do SisGera.



Fonte : Elaborado pelo autor (2021)

4.1.7 Alterações

Para integrar a aplicação desenvolvida com a existente, foram realizadas pequenas alterações de *layout* das páginas. Especificamente, foi proposta a adição de uma nova opção no menu de acesso lateral que encaminha o usuário para a tela de controle de sincronias, com a denominação de "Sincronizar ocorrências". Outra alteração proposta foi a adição de uma caixa de alerta na tela inicial informando o bombeiro, caso exista um boletim não sincronizado. Essas alterações estão exemplificadas na Figura 22.

4.2 Lighthouse

O `Lighthouse` é uma ferramenta desenvolvida pela Google para auditar a qualidade de uma AW. É executado a partir de uma extensão do Chrome e pode ser usado para identificar as falhas e as possíveis ações indicadas para melhoria da aplicação desenvolvida. Seu funcionamento se baseia em executar automaticamente uma série de testes na página e gerar um relatório sobre o desempenho observado.[2]

O relatório gerado pelo `Lighthouse` é baseado nas métricas de Performance, Acessibilidade, Melhores Práticas e Otimização para Motores de Busca (SEO), onde a avaliação é resumida em notas de 0 a 100 em cada um desses quesitos. A Figura 23 exibe o resumo

[2] Disponível em: https://www.developers.google.com/web/tools/lighthouse?hl=pt-br. Acesso em 21 de jul. de 2021

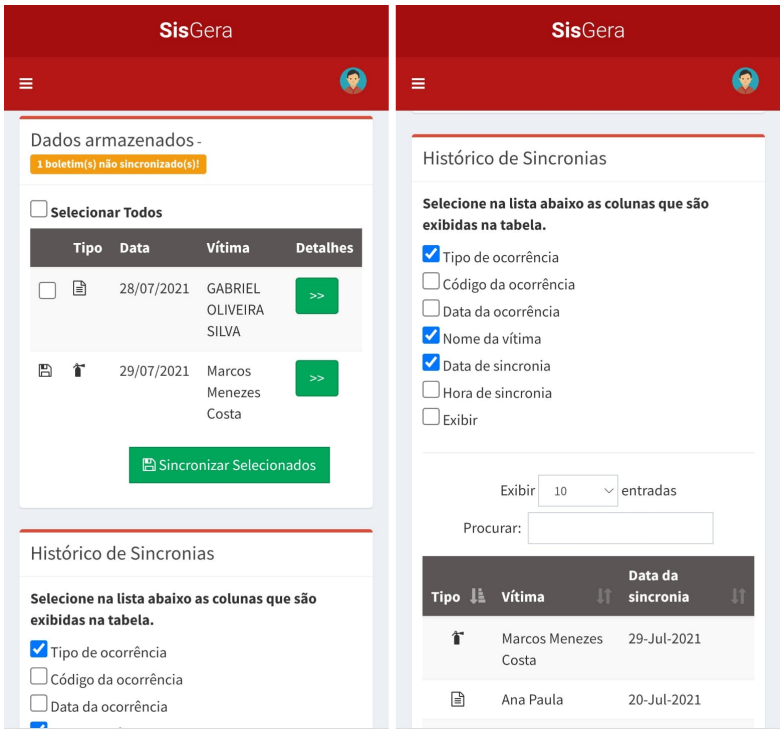
Figura 20 – Lista de ocorrências armazenadas.

Fonte : Elaborado pelo autor (2021)

da análise efetuada para o SisGera AWP.

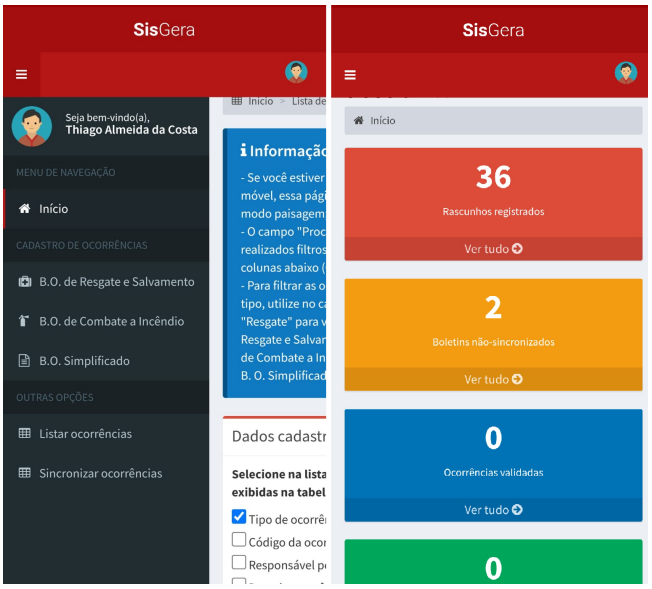
Essa ferramenta se mostra particularmente interessante para o contexto de uma AWP, pois, em uma métrica extra, fornece uma validação dos aspectos envolvidos no contexto dessa arquitetura, como, por exemplo, se a aplicação é instalável e otimizada para o uso em dispositivos móveis distintos. A Figura 24 contém a validação dos aspectos do SisGera AWP, onde é possível notar que foi obtido todos os requisitos, exceto o de "Redirecionar tráfego de HTTP para HTTPS". Porém, como se trata de uma configuração do servidor, esse requisito não pode ser abordado diretamente neste trabalho.

Figura 21 – Tela de controle de sincronias.



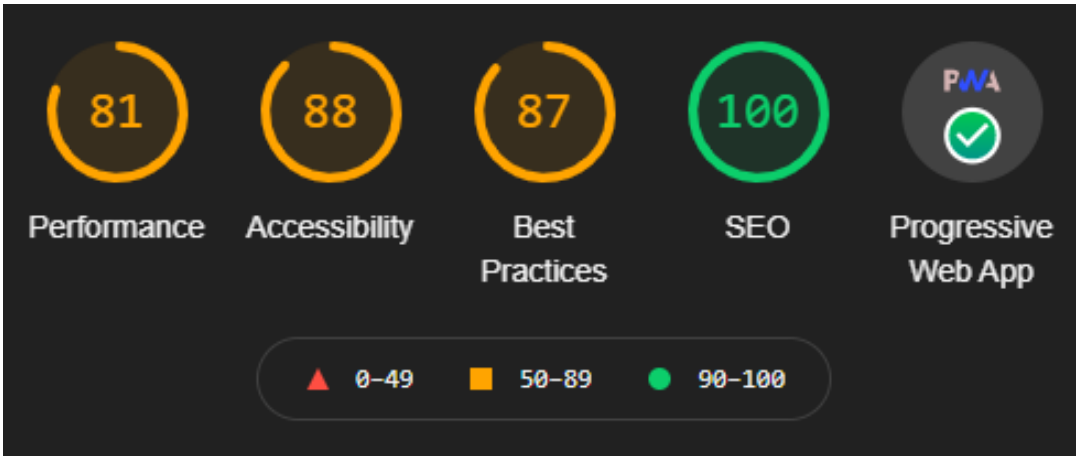
Fonte : Elaborado pelo autor (2021)

Figura 22 – Alterações visuais para o SisGera.



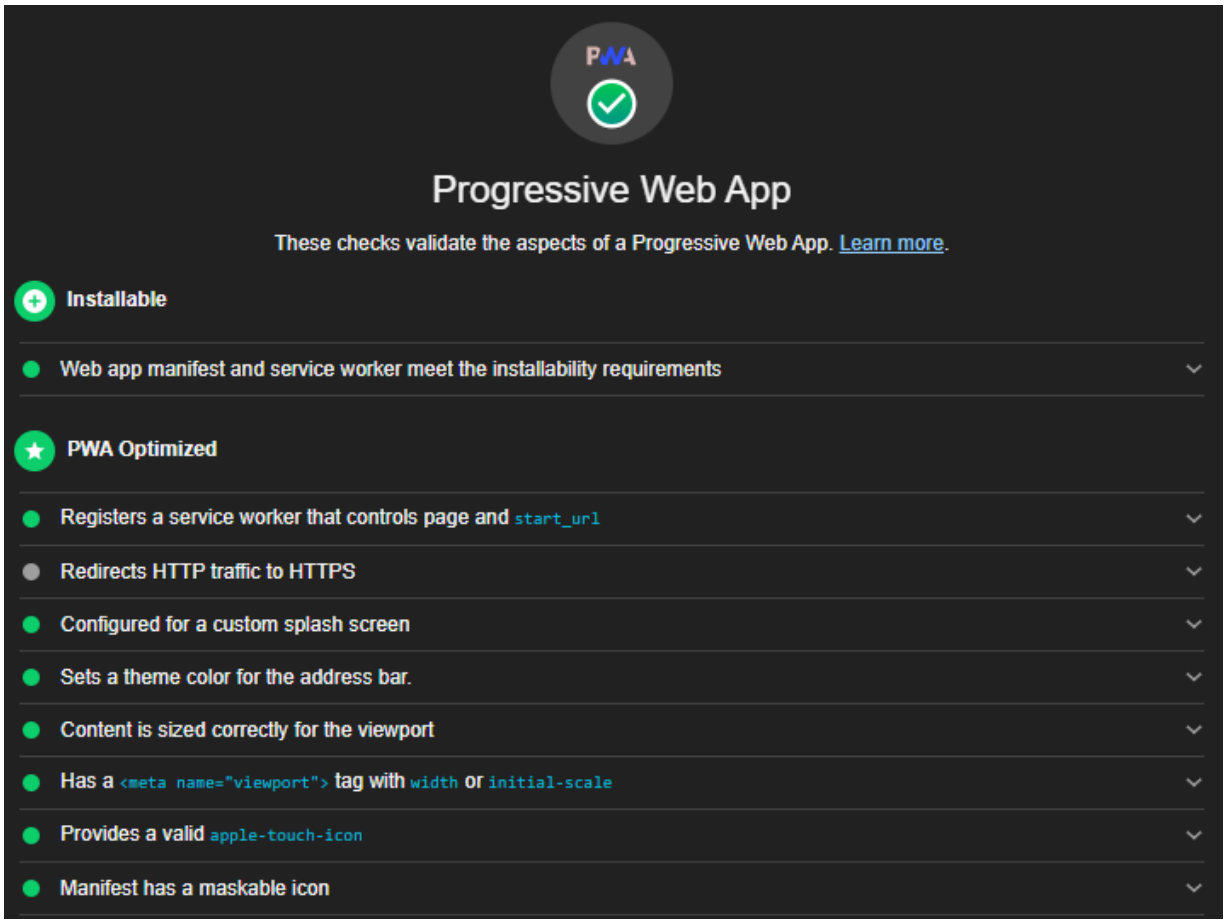
Fonte : Elaborado pelo autor (2021)

Figura 23 – Resumo do relatório gerado pelo `Lighthouse`.



Fonte : Elaborado pelo autor (2021)

Figura 24 – Aspectos de AWP da aplicação desenvolvida validados pelo `Lighthouse`.



Fonte : Elaborado pelo autor (2021)

5 Considerações Finais

Este trabalho apresentou o desenvolvimento de uma Aplicação *Web* Progressiva para o SisGera. A ênfase da aplicação desenvolvida está em apoiar o processo de registro de ocorrências atendidas por grupos de bombeiros voluntários e a necessidade de utilização do sistema sem conexão à *Internet*. Por meio de técnicas e ferramentas relacionadas as AWPs é possível o seu uso *offline*. A aplicação foi baseada no sistema já em uso para facilitar a operação dos usuários acostumados com a interface do sistema.

Primeiramente, foi realizada uma revisão da literatura envolvida neste trabalho. Foram levantadas informações sobre as aplicações *Web* e algumas abordagens para o desenvolvimento para dispositivos móveis, onde o termo e as tecnologias ligadas às AWPs foram esclarecidos. Também foram apresentados o SisGera, desenvolvido por Arantes (2018) e Oliveira (2018) e conceitos sobre o teste de *software*.

Na etapa seguinte, foi apresentada a escolha das ferramentas utilizadas para o desenvolvimento da aplicação. Na sequência, foi demonstrado os requisitos observados por meio de um modelo de história de usuário. Também foi descrito todo o processo de desenvolvimento da aplicação, incluindo o mapeamento do banco de dados, o desenvolvimento das funcionalidades da aplicação observadas nos requisitos e por fim, os testes funcionais realizados.

Finalmente, foram apresentados os resultados obtidos neste trabalho, por meio das telas e descrições das funcionalidades produzidas e a avaliação das métricas ligadas as AWPs utilizando a ferramenta `Lighthouse`.

Conforme apresentado ao decorrer do trabalho, o objetivo de tornar os registros de ocorrências atendidas pelos bombeiros voluntários por meio do SisGera disponível para o uso *offline* por meio da arquitetura AWP foi alcançado com sucesso. Consequentemente, espera-se que a aplicação desenvolvida se mostre uma adição útil para os grupos de bombeiros que utilizam o SisGera e com isso, impacte positivamente as comunidades apoiadas pelo trabalho voluntário das corporações.

5.1 Trabalhos Futuros

Com base em situações experienciadas no decorrer do desenvolvimento da aplicação e deste trabalho, foram identificadas algumas concepções e propostas para trabalhos futuros apresentadas a seguir:

- Avaliar e testar as funcionalidades e a usabilidade da aplicação desenvolvida, junta-

mente com os usuários finais da aplicação.

- Realizar testes de integração da aplicação desenvolvida com o sistema existente.
- Analisar a aplicação na prática para desenvolver formas de simplificar o *layout* das páginas e o seu código para melhorias de performance e a minimização do tamanho da aplicação.
- Desenvolver uma maior integração entre o SisGera AWP e o sistema existente para tornar mais eficiente o processo de registro de ocorrências *offline*.
- Introduzir novas funcionalidades para o SisGera AWP como a edição de boletins *offline*.

Referências

ALEGROTH, E.; FELDT, R.; RYRHOLM, L. Visual GUI testing in practice: challenges, problems and limitations. *Empirical Software Engineering*, v. 20, 01 2014. Citado na página 26.

ARANTES, V. M. *Um Sistema de Informação para apoio ao registro de ocorrências atendidas por grupos de bombeiros voluntários.* 2018. 80 f. Monografia (Graduação em Sistemas de Informação), Instituto de Ciências Exatas e Aplicadas, Universidade Federal de Ouro Preto, João Monlevade. Citado 8 vezes nas páginas 13, 27, 31, 32, 34, 36, 37 e 49.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao teste de software*. 2. ed. Rio de Janeiro: Elsevier, 2016. Citado 2 vezes nas páginas 25 e 26.

HUME, D. A. *Progressive Web App*. EUA: Manning Publications, 2017. Citado 3 vezes nas páginas 18, 20 e 24.

JOBE, W. Native apps vs. mobile web apps. *International Journal of Interactive Mobile Technologies (iJIM)*, v. 7, n. 4, p. 27–32, 2013. ISSN 1865-7923. Disponível em: <https://onlinejour.journals.publicknowledgeproject.org/index.php/i-jim/article/view/3226>. Citado na página 17.

LAUDON, K.; LAUDON, J. *Sistemas de Informação Gerenciais*. 11. ed. EUA: Pearson Education, 2014. Citado na página 17.

LONGO, H. E. R.; SILVA, M. P. da. A utilização de histórias de usuários no levantamento de requisitos Ágeis. International Journal of Knowledge Engineering and Management (IJKEM), p. 1–30, 07 2014. Citado na página 30.

OLIVEIRA, S. S. M. *Sistema de Informação para controle de materiais e doações aos bombeiros voluntários.* 2018. 91 f. Monografia (Graduação em Sistemas de Informação), Instituto de Ciências Exatas e Aplicadas, Universidade Federal de Ouro Preto, João Monlevade. Citado 3 vezes nas páginas 13, 27 e 49.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software*. 8. ed. [S.l.]: McGraw-Hill, 2016. Citado 2 vezes nas páginas 15 e 30.

SCHMIDT, D. C.; GOKHALE, A.; NATARAJAN, B. Leveraging application frameworks: Why frameworks are important and how to apply them effectively. *Queue*, Association for Computing Machinery, New York, NY, USA, v. 2, n. 5, p. 66–75, jul. 2004. ISSN 1542-7730. Disponível em: <https://doi.org/10.1145/1016998.1017005>. Citado na página 28.

SHEPPARD, D. *Beginning Progressive Web App Development: Creating Native App Experience on the Web*. EUA: Apress, 2017. Citado 2 vezes nas páginas 19 e 23.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. [S.l.]: Pearson Education, 2011. Citado na página 25.

WARGO, J. *Learning Progressive Web Apps*. EUA: Addison-Wesley, 2020. Citado 2 vezes nas páginas 19 e 22.

XANTHOPOULOS, S.; XINOGALOS, S. A comparative analysis of cross-platform development approaches for mobile applications. Association for Computing Machinery, New York, NY, USA, p. 213–220, 2013. Disponível em: <https://doi.org/10.1145/2490257.2490292>. Citado na página 18.