UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I - CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

THIAGO PABLICIO CABRAL NOGUEIRA

SISTEMA MULTIMÍDIA DE INFORMAÇÃO AO PASSAGEIRO

CAMPINA GRANDE - PB
2022

UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I - CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA

THIAGO PABLICIO CABRAL NOGUEIRA

SISTEMA MULTIMÍDIA DE INFORMAÇÃO AO PASSAGEIRO

Relatório de conclusão apresentado ao Curso de Ciências da Computação do Centro de Ciências e Tecnologia da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Computação.

Orientador: Profa. Dra. Sabrina de Figueirêdo Souto.

CAMPINA GRANDE - PB
2022

THIAGO PABLICIO CABRAL NOGUEIRA

SISTEMA MULTIMÍDIA DE INFORMAÇÃO AO PASSAGEIRO

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 02 de Dezembro de 2022.

Profa. Dra. Sabrina de Figueirêdo Souto (DC - UEPB)
Orientador(a)

Profa. Dra. Kézia de Vasconcelos Oliveira Dantas (DC - UEPB)
Examinador(a)

Prof. Dr. Paulo Eduardo e Silva Barbosa (DC - UEPB)
Examinador(a)

N778s Nogueira, Thiago Pablicio Cabral.
Sistema multimídia de informação ao passageiro [manuscrito] / Thiago Pablicio Cabral Nogueira. - 2022.
63 p. : il. colorido.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2022.
"Orientação : Profa. Dra. Sabrina de Figueirêdo Souto , Coordenação do Curso de Computação - CCT."

1. Transporte público. 2. Sistema de informação. 3. Multimídia. I. Título

21. ed. CDD 005.43

Elaborada por Deivid A. M. Neves - CRB - 15/899 BC/UEPB

Dedico esse trabalho a quem projetou e desejou esse momento desde o dia que nasci. Para você que está aí no céu minha mãe Elídia (*in memoriam*).

AGRADECIMENTOS

À minha mãe Elídia (*in memoriam*) que lutou contra todas as dificuldades que a vida pode impor para me oferecer educação, proteção e vida.

À meus familiares que estão sempre me ajudando e me incentivando a não parar nunca de evoluir.

À meus amigos e minha namorada Shara que nas horas boas e ruins me apoiam para que eu possa continuar a jornada, sem a ajuda deles não conseguiria avançar até aqui.

Aos meus colegas de curso que me ajudaram em todas as etapas da minha jornada não só acadêmica mas na vida.

Aos professores e funcionários da Universidade estadual da Paraíba por todos os ensinamentos e apoio prestado durante a minha formação.

Às professoras Sabrina Figueiredo Souto, Cheyenne Ribeiro Guedes e Luciana de Queiroz Leal pela paciência, incentivo e dedicação em me orientar neste trabalho.

Aos professores Misael Morais, Paulo Barbosa e Thiago Batista pela mentoria e inspiração que me passaram para evoluir como pessoa e profissionalmente.

Agradecimento especial aos amigos e colegas de curso Marcus Vinícius e Taynar Souza pela ajuda direta em vários momentos, tornando menos complicada minha jornada durante este curso.

E ao meu Pai que me deixa estar aqui para escrever mais um capítulo de minha vida, obrigado Deus.

“As pessoas não sabem o que querem até mostrarmos a elas” Steve Jobs

RESUMO

Tendo em vista a necessidade de informar um número cada vez maior de usuários de transporte público sobre o uso dos serviços, publicidade, entretenimento e informações de emergência em tempo real em caso de uma eventual catástrofe. E sabendo da necessidade de que essas informações sejam disponibilizadas de forma fácil e automatizada, desenvolvemos um Sistema Multimídia de Informação ao Passageiro (SMM). Buscamos ferramentas similares no mercado, como o Gerenciamento Remoto de Mensagens (GRM) que é uma ferramenta utilizada para o mesmo fim que envia mensagens a painéis de LED e a autofalantes, porém não atendia de forma satisfatória aos requisitos dos clientes que necessitavam de uma solução que fosse personalizável e que também permitisse a exibição de conteúdo em TVs e que permitisse troca de mensagens em tempo real sem a necessidade de estar conectado à internet. Com isso precisamos desenvolver um sistema de informação aos passageiros que atendessem a esse propósito. Desenvolvemos um aplicativo de exibição de conteúdos que foi instalado em terminais de TVs espalhados por uma estação ferroviária, e um aplicativo para gerenciar esses conteúdos. Descrevemos as etapas necessárias para a implementação da funcionalidade de agendamento de tarefas mostrando como criamos as tabelas de banco de dados, a codificação das telas e os testes manuais.

Palavras-Chave: transporte público; informação; multimídia; tempo real.

ABSTRACT

Given the need to inform a growing number of public transport users about the use of services, advertising, entertainment and emergency information in real time in the event of a possible catastrophe. And knowing the need for this information to be made available in an easy and automated way, we have developed a Multimedia Passenger Information System (SMM). We looked for similar tools on the market, such as Remote Message Management (GRM), which is a tool used for the same purpose that sends messages to LED panels and speakers, but it did not satisfactorily meet the requirements of customers who needed a solution. that was customizable and that also allowed the display of content on TVs and that allowed the exchange of messages in real time without the need to be connected to the internet. Therefore, we needed to develop a passenger information system that served this purpose. We developed a content display application that was installed on TV terminals spread across a railway station and an application to manage this content. We describe the necessary steps to implement the task scheduling functionality, showing how we created the database tables, coding the screens and testing manually.

Keywords: public transport; information; multimedia; real time.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTO

Segundo a Associação Nacional das Empresas de Transportes Urbanos (NTU)[1], em 2019, foram realizadas 33,2 milhões de viagens por passageiros de transporte público pagantes no país. Esse número dá uma dimensão da quantidade de pessoas indo e vindo nas ruas, paradas de ônibus, aeroportos, rodoviárias, e demais locais de grande circulação, daí surge a oportunidade de informar as pessoas sobre os mais variados assuntos de interesse, desde a hora, previsão do tempo, até informações mais específicas como, qual o próximo ônibus, onde estão os banheiros, qual o horário de partida e chegada do trem, até mesmo um bom dia.

Com os sistemas de informação podemos garantir aos usuários uma melhor utilização dos serviços, prestando informações atualizadas e precisas de forma clara, com a maior facilidade e velocidade possível. Scariot; Lanzoni; Spinillo (2011). Junto a essas mensagens informativas é possível ainda transmitir mensagens audiovisuais para pessoas com necessidades especiais, mensagens emergências em caso de alguma eventual catástrofe e mensagens publicitárias.

Segundo Schein (2003) Sistemas de Informação ao Usuário: combina tecnologias computacionais e de comunicação para prover informação aos passageiros em casa, no trabalho, na rua ou na parada de ônibus ou estação de trem/metrô Essa distribuição de informação abrange por vezes grandes áreas a serem cobertas com centenas de milhares de pessoas circulando e é necessário portanto que consigamos prender a atenção delas e transmitir a mensagem que desejamos. Uma das formas de realizar tal comunicação é por meios eletrônicos audiovisuais, conhecidos como sistemas multimídia, que são compostos por telas e alto-falantes dispostos em locais estratégicos para atingir o maior número possível de pessoas.

Mansur et al (2019) afirma que essa informação é importante para que o usuário tenha maior controle sobre seu tempo, como por exemplo tomar a decisão do melhor momento para aguardar no terminal ou até mesmo consumir bens e serviços, enquanto aguarda seu meio de transporte. Um dos maiores desafios desses sistemas, além de sua instalação, é o monitoramento e atualização dos conteúdos sendo apresentados. A solução mais eficiente é

[1] https://www.ntu.org.br/

que tal tarefa seja realizada de forma remota e automatizada, removendo a necessidade de um funcionário ir pessoalmente em cada monitor alimentar esses dispositivos com as mídias. Desse modo economiza-se tempo e dinheiro, além de permitir criar programações que serão reproduzidas em tempos pré determinados, criando rotinas, além de possibilitar o envio de mensagens personalizadas a qualquer momento para locais - ou grupos de locais - específicos sem afetar os demais.

No mercado existem diversas empresas especializadas em mídia e propaganda. A maioria delas trabalha com veiculação de anúncios em *outdoors* e *banners* espalhados pelas ruas, mercados, rodoviárias, etc. Uma parte dessas empresas se especializou na veiculação digital dessas propagandas e mensagens pois é mais atrativo para os usuários finais, é mais versátil pois permite a troca rápida dos conteúdos, gera mais receita pois em um mesmo ponto é possível colocar um número bem maior de propagandas, como podemos observar, por exemplo nas partidas de futebol, onde antes existiam placas estáticas com publicidade, e foram substituídas por painéis de eletrônicos que permitem maior volume, gerando, consequentemente mais lucro.

1.2 PROBLEMA

A veiculação dessas mensagens é difícil dependendo da mídia que se deseja utilizar, sendo comumente utilizado conteúdo estático que precisa de pouco ou nenhum monitoramento, porém esses conteúdos estáticos não são facilmente personalizáveis e sempre exigem que alguém vá ao local para trocar o conteúdo. Podemos citar o exemplo dos *outdoors* que tem uma impressão que é fixada por determinado período, precisando depois que funcionários se dirijam até o local para remoção e troca por outro conteúdo. Na mesma linha existem painéis de LED[2] ou LCD[3] que têm quase a mesma dinâmica, sendo necessário que alguém responsável de tempos em tempos dirija-se ao local para trocar o conteúdo através de um pendrive, cd ou baixando de forma manual conteúdos de determinados repositórios.

A empresa já possuía em seu portfólio um sistema de gerenciamento de mensagens chamado Gerenciador Remoto de Mensagens (GRM) que era capaz de enviar mensagens previamente gravadas a painéis de LED e auto falantes, porém não tinha integração com TVs e nem permitia o envio nem a troca de conteúdos de forma rápida porque não dava margem para o carregamento de novos conteúdos possibilitava enviar textos nos painéis e mensagens

[2] LED = Light Emiting Diode, do inglês.

[3] LCD = Liquid Crystal Display, do inglês.

de áudio pelos auto falantes, porém apenas através de agendamento, não possibilitando que fossem enviadas em tempo real,diante disso surgiu a necessidade de criar um sistema multimídia que fosse facilmente personalizável, tivesse mensagens em tempo real, fácil gestão de conteúdos e fosse de operação simples porém não encontrou um que atendesse a todas as suas exigências de forma satisfatória. Analisou, por exemplo, o DSplay[4] que era um sistema de TV corporativa, porém esbarrava na necessidade de ser conectado à internet e de não possuir mensagens em tempo real.

1.3 SOLUÇÃO

Como tínhamos em mãos os requisitos e já existia o GRM com parte da modelagem da solução implementada, partimos dele e evoluímos a modelagem para solucionar a falta de flexibilidade de criação de conteúdos, bem como a impossibilidade de envio de mensagens em tempo real. Para isso criamos o SMM que possuía além dos requisitos conhecidos no GRM, ainda as funcionalidades de criação de templates, playlists, inclusão de biblioteca de mídias, mensagens em tempo real, agendamento de tarefas, e configuração de terminais remotamente.

O trabalho consistiu em produzir uma plataforma de monitoramento e configuração remota de terminais multimídias - tais como TVs - que estarão espalhados por uma estação rodoviária. O sistema deve ser capaz de se comunicar com os diversos terminais bem como oferecer ao operador um painel de gerenciamento desses terminais. A interface nos terminais será implementada utilizando navegadores web, porém a comunicação será via rede interna, pois nem sempre será possível garantir internet nesses locais.

Dentre as funcionalidades planejadas, a plataforma deve permitir ao operador do sistema multimídia executar diversas tarefas de controle e gerenciamento dos conteúdos, a saber: criar layouts de exibição; criar, editar e salvar as mídias como vídeos, áudios, imagens, as playlists de reprodução, os terminais que exibirão essas mídias; e criar agendamentos em fila de reprodução, dando a possibilidade de interromper a programação, mandar mensagens ou conteúdos não programados, e após a exibição, retomar a programação normal estabelecida na fila original. Para tal será criado um painel em que os operadores terão acesso a suas respectivas áreas de conteúdo, onde poderão também cadastrar terminais de tv; criar agendamentos de conteúdo; checar o status, editar, e enviar mensagens personalizadas aos

[4] https://dsplay.tv/site/

passageiros como informativos, também será possível monitorar o tempo de chegada dos trens, a previsão do tempo, e o *feed* de notícias do dia.

A utilização do SMM trará benefícios para os operadores que terão maior de configuração e atualização dos conteúdos, para os passageiros que estão melhor informados sobre os serviços de transporte e demais informações úteis, e para as empresas que tornam a prestação do serviço mais eficiente, venderão mais publicidades, e diminuirão o tempo gasto com a configuração dos terminais de mídia.

1.4 OBJETIVOS

1.4.1 Objetivos Gerais

Este relatório técnico tem, portanto, o objetivo de desenvolver um Sistema Multimídia de Informação ao Passageiro (SMM), desenvolvido pelo autor como parte de suas atividades profissionais freelancer, e tendo como cliente uma empresa de aplicações eletrônicas multimídia.

1.4.2 Objetivos Específicos

1. Desenvolver um aplicativo web a ser instalado em cada terminal para exibir os conteúdos.
2. Diminuir o tempo de configuração e gestão de conteúdos a serem exibidos.
3. Melhorar a experiência de utilização dos usuários de transportes públicos.

1.5 ESTRUTURA DO DOCUMENTO

Buscaremos descrever o processo de desenvolvimento incremental de um sistema de monitoramento e controle de mídias inserido no contexto de uma rodoviária e na criação de uma interface amigável para os usuários desse sistema.

Para tanto, nos capítulos seguintes apresentaremos a Fundamentação Teórica onde traremos as tecnologias utilizadas (Capítulo 2), apresentaremos O Sistema Multimídia de Informação ao Passageiro, onde explicaremos as funcionalidade do software e a metodologia utilizada para desenvolver o projeto (Capítulo 3), a implementação de algumas das funcionalidades (Capítulo 4) e a conclusão do trabalho (Capítulo 5).

2 FUNDAMENTAÇÃO TEÓRICA

O desenvolvimento de um sistema web consiste na escolha de diversas ferramentas que trabalharão em conjunto para a implementação do banco de dados, servidor, páginas web e demais componentes. Dividimos este capítulo em três seções onde traremos tópicos relacionados a *backend*, *frontend* e sobre ferramentas e tecnologias utilizadas durante o projeto.

2.1 BACKEND

O *backend* compreende o lado do servidor onde fica a lógica da aplicação, o banco de dados, algoritmos, APIs, validações, integração com outros servidores, etc. Nesta seção apresentaremos cada uma das tecnologias do *backend* que foram utilizadas no trabalho.

2.1.1 PHP

Segundo a documentação oficial o PHP (um acrônimo recursivo para PHP: Hypertext Preprocessor) é uma linguagem de *script open source* de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do HTML (Php, c2022).

Por rodar tanto do lado do servidor, quanto do lado do cliente, é uma das linguagens mais versáteis da web, trabalha muito bem em conjunto com outras linguagens como o Javascript, e do lado do servidor também é amplamente utilizada pela fácil escrita e integração com o MySQL e diversas outras ferramentas.

Ao longo dos anos o PHP vem evoluindo bastante, muito devido a uma comunidade gigantesca e por ser a base de *frameworks* amplamente utilizados pelo mercado como o Wordpress que domina o mercado de sites e páginas web, além de *frameworks backend* como o Laravel, Zend, CodeIgniter, entre outros. Poucas linguagens tiveram uma evolução tão drástica e visível quanto o PHP nos últimos anos, podemos destacar a mudança para orientação a objetos que não existia nas primeiras versões, a criação do PDO[5] para facilitar a integração com bancos de dados, a introdução do Composer[6] como forma de gerenciar as

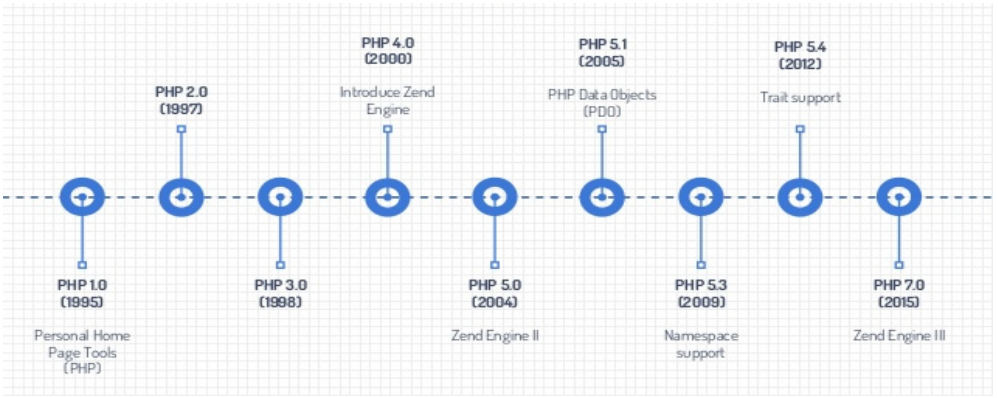
[5] PDO = PHP Data Objects, do inglês.

[6] https://getcomposer.org/

dependências, introdução da Zend[7] Engine que trouxe maior velocidade na interpretação e execução dos scripts PHP entre outras melhorias.

A Figura 1 mostra uma linha do tempo com algumas das evoluções que o PHP apresentou ao longo dos anos.

Figura 1 - Evolução do PHP



Fonte: Raphael Cozzi (c2022)

2.1.2 Node

Como um ambiente de execução JavaScript assíncrono orientado a eventos, o Node.js é projetado para desenvolvimento de aplicações escaláveis de rede (Opus Software, c2021).

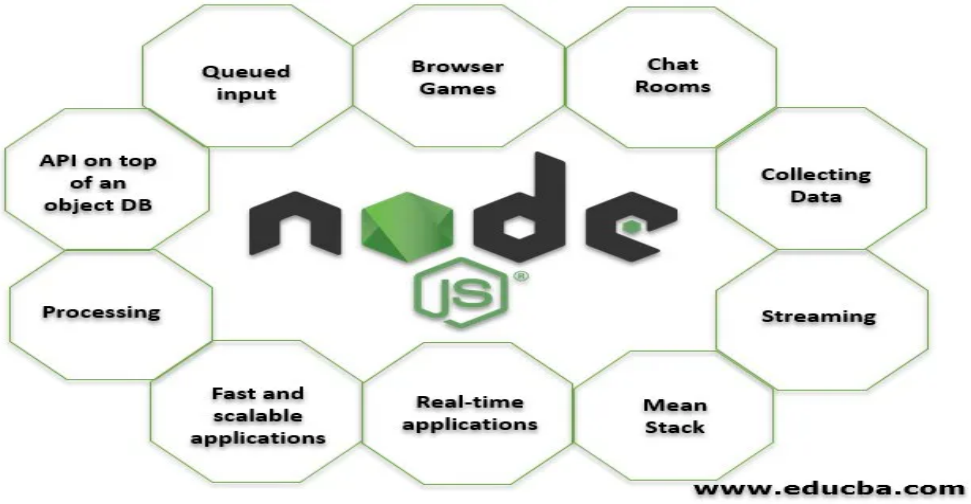
O Node é definido como um ambiente de execução de Javascript sem a necessidade de um browser, isso permite que possamos utilizar o Javascript também no *backend*, aumentando o leque de possibilidades de uso dessa linguagem.

Podemos citar como vantagens de utilização do Node, a leveza, simplicidade do código, alta produtividade, flexibilidade, suporte entre outros. Possui um poderoso gerenciador de pacotes chamado NPM que facilita na gestão de dependências, com ele é fácil manter o projeto organizado e também é possível executar diversas rotinas como rodar testes, executar servidores locais, rodar *builds* da aplicação, etc.

Dentre as diversas aplicações do Node podemos destacar: Criação de chats, APIs, streaming, jogos online, filas, proxies, aplicações real time, entre outras como vemos na Figura 4 abaixo.

[7] https://www.zend.com/

Figura 2 - Aplicações do Node



Fonte: Educba (c2022)

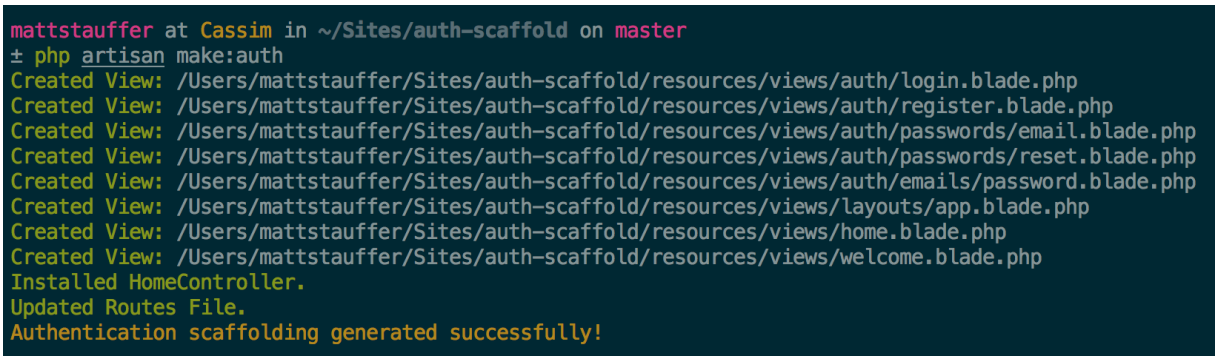
2.1.3 Laravel

Laravel é um *framework* PHP livre e *open-source* criado por Taylor B. Otwell para o desenvolvimento de sistemas web que utilizam o padrão MVC[8], com ele podemos criar sistemas web facilmente se preocupando apenas com a implementação, já que diversas configurações, bibliotecas e códigos já estão prontos (Laravel, c2022). Com uma gama de pacotes que compõem seu universo, o Laravel caiu nas graças da comunidade por simplificar o uso do PHP.

Através do Laravel a implementação de Filas, Eventos, *Websockets*, Emails, Notificações, *Jobs* e outras aplicações está a uma linha de comando, aliás, justamente é um dos pontos fortes, pois ele possuí um robô chamado Artisan que torna o desenvolvimento bastante simples e divertido. Com o comando: “php artisan make:auth” criamos de uma só vez, todo um sistema de login com todas as classes, tabelas e páginas HTML necessárias, a partir desse sistema de login podemos iniciar nosso desenvolvimento vendo os resultados imediatamente como vemos na Figura 5.

[8] MVC = Model View and Controller, do inglês.

Figura 3 - Comando artisan



Fonte: Matt Stauffer (c2022)

2.2 FRONT END

O *frontend* compreende a interface com o usuário, sites, páginas, documentos, aplicativos, etc. É por onde o usuário consegue inserir e ler informações do sistema. O trabalho do desenvolvedor *frontend* é tornar essa interação, simples e intuitiva utilizando diversas tecnologias existentes, a seguir apresentaremos as tecnologias que utilizamos no desenvolvimento do SMM.

2.2.1 Javascript

Javascript é uma linguagem de programação que permite a você criar conteúdo que se atualiza dinamicamente, controlar multimídias, imagens animadas (JavaScript, c2022).

É uma linguagem que foi pensada inicialmente para rodar no lado do cliente, posteriormente com a chegada do Node foi para o lado do servidor também. Para seu propósito inicial, é a principal linguagem de programação utilizada para manipular o HTML e criar as interações dinâmicas que vemos na tela, sem ela a web seria um emaranhado de páginas estáticas, pesadas e difíceis de manter.

É uma linguagem muito versátil, muito fácil de programar e como é executada diretamente em navegadores não precisa de muita configuração para que comecemos a escrever códigos.

No entanto, a partir do Javascript surgem diversos frameworks a todo tempo, entre os mais famosos podemos citar o JQuery que é uma biblioteca que simplifica a escrita do Javascript, além do VueJS, React e Angular que estão entre os frameworks mais usados

atualmente pelo mercado. Na Figura 2 vemos alguns dos principais frameworks e bibliotecas do universo Javascript, dentre os quais, estão os que citamos.

Figura 4 - Frameworks e bibliotecas Javascript

Fonte: Dizzy Coding (s.d.)

2.2.2 HTML

HTML é a linguagem de marcação que nós usamos para estruturar e dar significado para o nosso conteúdo web, por exemplo, definindo parágrafos, cabeçalhos, tabelas de conteúdo, ou inserindo imagens e vídeos na página (Mozilla, c2022).

Devido a sua popularidade, o HTML é atualmente a linguagem de marcação padrão da web, são utilizadas diversas tags para os mais variados usos. Dentre as mais populares destacamos a div que demarca blocos de separação de outras tags, fortemente utilizada na estruturação básica das páginas. As tags ul li usadas para a criação de listas, as tags table, tr, td, th usadas para estruturação de tabelas, a tag p para parágrafos, as tags h1, h2, h3, h4, h5, h6 usadas para títulos sendo a h6 para tamanhos menores de h1 para maiores, seguindo uma ordem decrescente.

Com o HTML apenas é possível criar páginas estáticas, porém na web moderna ele deve trabalhar em conjunto com o CSS e com linguagens de programação, o uso mais

comum para desenvolver uma página interativa atualmente é a tríade HTML, CSS e Javascript.

Abaixo vemos um exemplo de uma página HTML bem simples, onde é iniciada por uma tag de raiz chamada html, e possui um bloco delimitado pela tag head com as tags met, para configurações e definições, a tag title que é usada para identificar a página, e pela tag body onde realmente inserimos as demais tags e conteúdos da página. No exemplo o conteúdo é formado por uma tag de imagem.

```
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>Minha página de teste</title>
  </head>
  <body>
    <img src="images/firefox-icon.png" alt="minha página de teste">
  </body>
</html>
```

2.2.3 CSS

CSS é uma linguagem de marcação que nós usamos para aplicar estilo ao nosso conteúdo HTML. Por exemplo, definindo cores de fundo e fontes, e posicionando nosso conteúdo em múltiplas colunas (Mozilla, c2022). Com o surgimento do HTML, houve a necessidade de aplicar estilos e formatações cada vez mais complexas.

O CSS foi introduzido para permitir que a estilização fosse feita de maneira organizada, simplificada, permitindo a reutilização em diversas páginas e em arquivos apartados do HTML. Abaixo vemos um pequeno trecho de código CSS aplicado a uma tag de parágrafo que define a espessura da letra em 400 pixels e a cor vermelha.

```
p {
    font-weight: 400px;
    color: red;
}
```

2.2.4 Vue

Vue.js é um framework Javascript de código-aberto, focado no desenvolvimento de interfaces de usuário e aplicativos de página única. Ele funciona tanto para escrever todo um projeto ou apenas parte dele, sua popularidade se deve a facilidade de implementação, configuração e implantação, ele é leve sem deixar de ser robusto (Vue, c2022).

Com o vue é possível desenvolver todo um projeto, ou pequenas partes de projetos existentes. Atualmente é uma das bibliotecas mais utilizadas no mundo para criação de páginas web pois tem uma documentação extensa, uma comunidade ativa, uma grande gama de componentes criados, além de ter uma sintaxe simples, que permite uma curva de aprendizado relativamente baixa.

Os pacotes mais conhecidos que trabalham em conjunto com o Vue são o Vuex, para controle de estados da aplicação, o Vuetify[9] que é uma biblioteca de componentes gráficos, o Vue Router[10] que organiza os componentes através de rotas, permitindo uma melhor organização dos arquivos e o Axios[11], que é usado também por outros diversos frameworks Javascript.

2.3 CONCEITOS E FERRAMENTAS

2.3.1 MVC

Segundo Valente (2020) padrão arquitetural MVC (Model-View-Controller) foi proposto no final da década de 70 como uma forma de organizar as classes de um projeto em 3 camadas:

Visão: tudo que vemos na tela, botões, formulários, textos, etc. Nos sistemas modernos, essa camada é composta por Javascript, CSS e HTML e existem muitos

[9] https://vuetifyjs.com/en/

[10] https://router.vuejs.org/

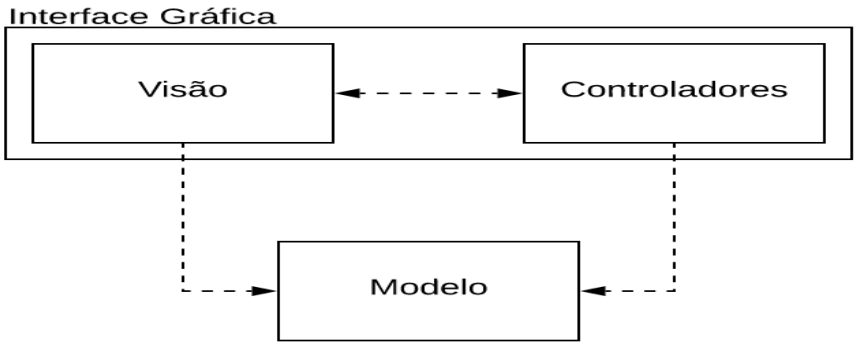
[11] https://axios-http.com/docs/intro

frameworks e bibliotecas que facilitam o desenvolvimento como Bootstrap[12], VueJS, Sass[13], Material UI[14], etc.

Controladoras: classes que tratam e interpretam eventos gerados por dispositivos de entrada, podemos enxergar essa camada como uma ponte entre a Visão e o Modelo, onde ela informa a mudança de estados de uma para a outra. Nos sistemas modernos, essa camada é acessada diretamente ou através de rotas que mapeiam os métodos contidos nessas controladoras.

Modelo: classes que contém a lógica de negócio, onde estão os métodos que manipulam o estado do domínio da aplicação, é nelas também que conversamos de perto com o banco de dados e enviamos as alterações para a camada controladora. Na Figura 9 está o diagrama MVC, onde a camada de visão troca mensagens diretamente com a camada controladora, e ambas estão conectadas ao modelo.

Figura 5 - Modelo MVC



Fonte: Valente (2020)

[12] https://getbootstrap.com/

[13] https://sass-lang.com/

[14] https://v4.mui.com/pt/

2.3.2 Websockets

WebSockets é uma tecnologia avançada que torna possível abrir uma sessão de comunicação interativa entre o navegador do usuário e um servidor (Mozilla, c2022). Através de websockets é possível criar uma comunicação em tempo real entre o cliente e o servidor removendo a necessidade de recarregar a página web para atualizar as informações.

Existem inúmeras aplicações para o uso desse tipo de comunicação como jogos online, onde a interação entre jogadores seria prejudicada se não houvesse a resposta instantânea, a experiência dos jogadores seria prejudicada. Outra aplicação que usa os websockets é o chat que a maioria das redes sociais utiliza.

A Figura 8 mostra como se dá a comunicação bidirecional entre cliente e servidor através de websockets, onde a informação é "empurrada" para o cliente, toda vez que há uma mudança no servidor.

Figura 6 - Comunicação bidirecional dos websockets

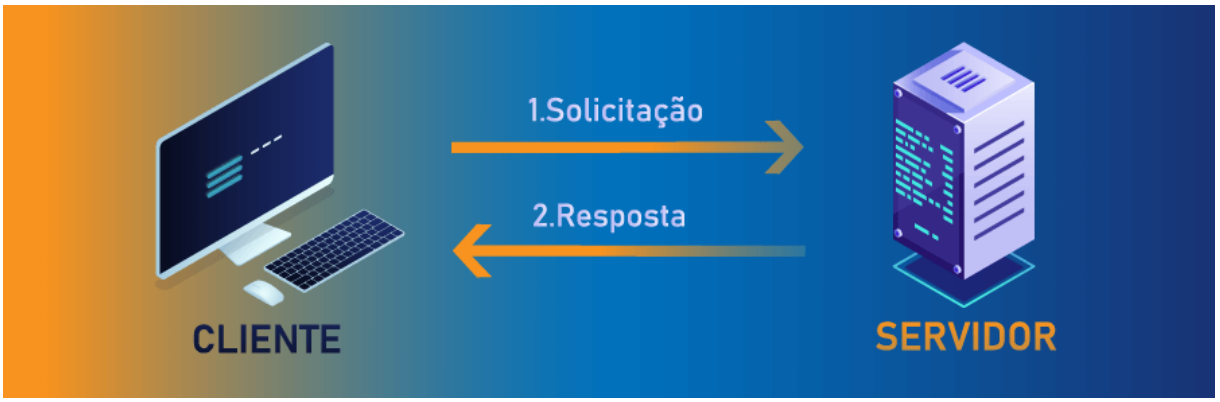
Fonte: Wallarm (c2022)

2.3.3 MySQL

MySQL é um sistema de gerenciamento de banco de dados (SGBD), gratuito e livre. Criado inicialmente em 1995, foi sofrendo evoluções com o tempo, e atualmente é a plataforma mais utilizada no mundo (Mysql, c2022).

O MySQL é um gerenciador de bancos de dados bastante popular que utiliza a linguagem SQL[15]. Atualmente é o segundo gerenciador mais utilizado do mundo e se destaca principalmente por: Ser de código aberto, gratuito e personalizável, ser bastante flexível e seguro, ter alto desempenho, é amplamente usado na indústria sendo inclusive um dos padrões para uso do SQL. Abaixo na Figura 3, vemos um esquema simples de cliente-servidor utilizado pelo MySQL.

Figura 7 - Esquema cliente-servidor utilizado pelo MySQL



Fonte: Hostinger (c2022)

2.3.3 Git

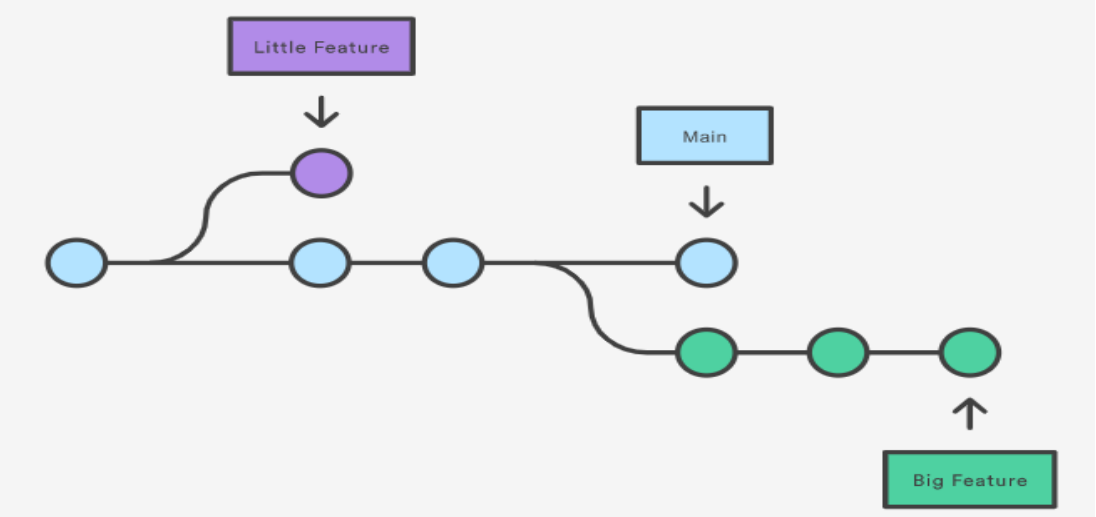
O Git é um dos sistemas de controle de versão mais utilizados no mundo de desenvolvimento de software. Ele é um projeto de código aberto desenvolvido em 2005 por Linus Torvalds, o criador do kernel do Linux[16]. É uma das formas mais inteligentes de manter o histórico do desenvolvimento, além de facilitar a colaboração de equipes, pois nele você tem várias versões do mesmo código possibilitando a integração de várias equipes sem a preocupação de conflitos impossíveis de resolver (Git, c2022).

[15] SQL = Structured Query Language, do inglês.

[16] https://4linux.com.br/

A partir da ramificação principal são criadas cópias onde de fato as pessoas fazem o trabalho de forma paralela, e após a conclusão, são mescladas novamente na principal. Abaixo na Figura 6 vemos dois exemplos de funcionalidades que foram desenvolvidas a partir da ramificação principal, que futuramente após serem validadas serão mescladas.

Figura 8 - Ramificações do git



Fonte: Atlassian (s.d.)

2.3.1 Visual Studio Code

O Visual Studio Code é um editor de código-fonte leve, mas poderoso, executado em sua área de trabalho e disponível para Windows[17], macOS[18] e Linux (Visual Studio Code, c2022).

Foi escrito utilizando o framework Electron[19], ferramenta criada pelo GitHub[20] que permite a criação de softwares Desktop com HTML, CSS e JavaScript. Não podemos confundí-lo com o Visual Studio, pois esse último é ambiente de desenvolvimento integrado completo, porém o VS Code não é menos poderoso pois devido a uma vasta gama de plugins ele permite uma alta produtividade, atualmente muitos desenvolvedores estão o adotando como única ferramenta para escrita de código (Microsoft, c2022).

[17] https://www.microsoft.com/

[18] https://www.techtudo.com.br/tudo-sobre/mac-os/

[19] https://www.electronjs.org/

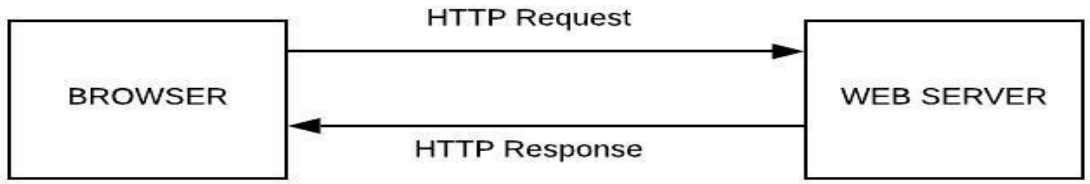
[20] https://github.com/

2.3.2 HTTP

HTTP é um protocolo que permite a obtenção de recursos, como documentos HTML. É a base de qualquer troca de dados na Web e um protocolo cliente-servidor, o que significa que as requisições são iniciadas pelo destinatário, geralmente um navegador da Web (Mozilla, c2022).

Na Figura 7, vemos como ocorre a interação entre cliente e servidor através do HTTP, onde o browser requisita dados do servidor e obtém a resposta.

Figura 9 - Protocolo HTTP



Fonte: Ashwin P. (2019)

No protocolo HTTP as trocas de mensagens são feitas através de métodos que são aplicados dependendo da necessidade, são por exemplo, os métodos GET, POST, DELETE, PUT, etc. Cada retorno desses métodos tem um status em forma de número para tipo de resposta, abaixo vemos alguns status e seu significado.

- 200 — Sucesso
- 301 — Movido permanentemente
- 302 — Encontrado, redirecionado para novo URL
- 304 — Não modificado
- 401 — Não autorizado
- 403 — Proibido
- 404 — Não encontrado
- 500 — Erro Interno do Servidor
- 504 — Tempo limite do gateway (Tempo limite de carregamento)

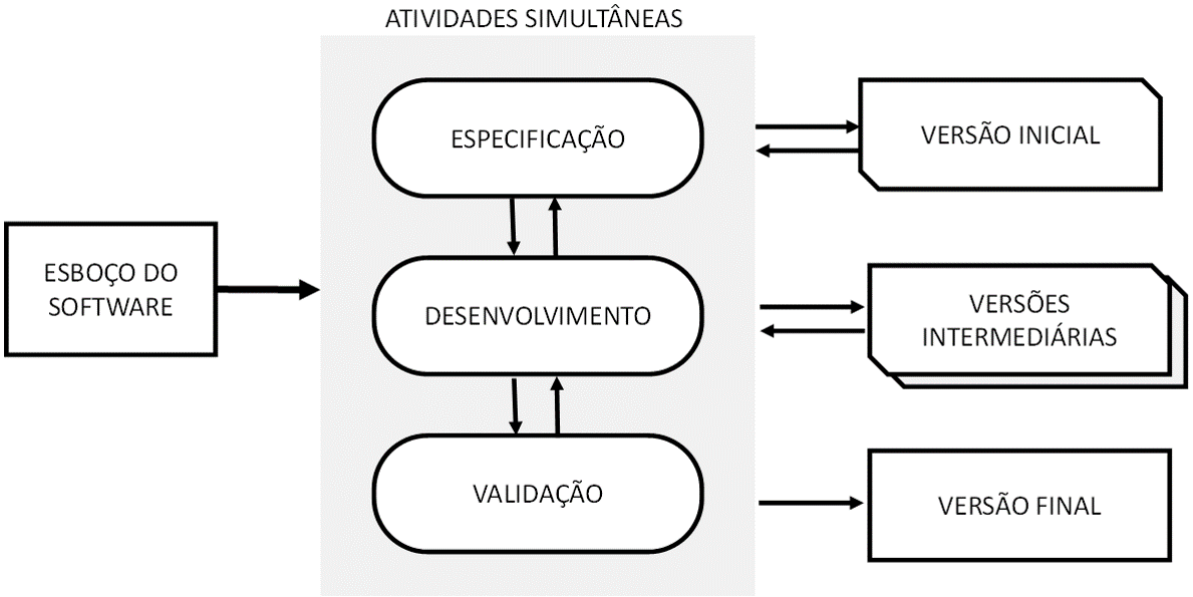
2.3.3 Modelo de Desenvolvimento Incremental

O Modelo de Desenvolvimento Incremental, diferente do Modelo em Cascata que especifica todos os requisitos de uma vez, baseia-se no desenvolvimento de pequenos pedaços que são entregues, para que o cliente dê mais rapidamente o feedback permitindo que as mudanças necessárias sejam aplicadas mais rapidamente, para Sommerville (2011) quando comparado ao modelo em cascata, o custo de acomodar mudanças é reduzido; é mais fácil obter feedback dos clientes; é possível realizar a entrega mais rápida.

Um dos problemas que ocorre nesse modelo é que os requisitos não são tão estáveis podem haver muitas iterações até que realmente sejam finalizados. Outro problema apontado por Sommerville (2011) é que o progresso não é visível e os gerentes precisam de entregas regulares para mensurar o progresso. Se os sistemas forem desenvolvidos com rapidez, não será economicamente viável produzir documentos que reflitam cada uma das versões do sistema, ainda ocorrem degradações do sistema ao longo dos incrementos que podem ser difíceis de serem corrigidas.

Abaixo na Figura 10 está uma representação das iterações do Modelo Incremental.

Figura 10 - Modelo Incremental



Fonte: Sommerville (2011, p. 22).

3 METODOLOGIA

Nesta seção apresentamos as etapas seguidas para a realização deste trabalho onde foram levantados os requisitos funcionais e não funcionais, a partir deles extraímos os casos de uso, definimos a arquitetura do sistema, escolhemos as tecnologias utilizadas e definimos as especificações para guiar o desenvolvimento.

Segundo Sommerville (2011, p.21) o desenvolvimento incremental consiste em escrever uma versão inicial, apresentar aos usuários e depois ir escrevendo outras versões até que tenha uma versão satisfatória desenvolvida. Foi escolhido o modelo incremental, onde foram desenvolvidos pequenos pedaços do sistema a cada iteração e enviados ao cliente para que ele nos desse feedbacks até a finalização do projeto.

Após conversas com o cliente bem como a análise de softwares similares, criamos os casos de uso e diagramas de entidade e relacionamentos para guiar o desenvolvimento, bem como um protótipo de interface. Além disso, escolhemos quais as tecnologias utilizadas e especificamos as funcionalidades.

O sistema em questão foi demandado por uma empresa de aplicações eletrônicas com foco em comunicação audiovisual, que buscava como principal solução um software que permitisse o gerenciamento e monitoramento remoto de todos os terminais de mídia disponíveis em uma determinada rodoviária.

3.1 REQUISITOS FUNCIONAIS

O cliente precisava de um Sistema Multimídia de Informação ao Passageiro que foi denominado SMM por eles para se referir a sistema multimídia. O SMM devia ser composto de um painel de controle das mídias que é uma aplicação web que seria operada por dois tipos de usuários, o OPERADOR que teria permissão de criar, alterar e excluir apenas conteúdos criados por ele mesmo. O ADMINISTRADOR que seria um usuário com permissões de criar, alterar e excluir dele próprio, bem como dos demais níveis de usuários, além de ter acesso ao painel de configurações avançadas que o OPERADOR não teria.

Esse painel teria as telas de dashboard onde podia ver as principais métricas do sistema, como número de tarefas em andamento ou de terminais cadastrados, a tela de monitoramento, as telas de criação, exclusão e edição de templates, mídias, playlists, terminais, zonas de conteúdo. As telas de agendamentos, de envios de mensagens em tempo real, telas de criação de backup e de gestão de usuários pois para a operação precisamos logar

no sistema. Outra parte do SMM seria uma aplicação web que seria instalada em cada terminal, ela seria a responsável por exibir os conteúdos, agendamentos e mensagens que foram criados pelo operador de forma remota utilizando o painel de controle de mídias. Essas duas partes se comunicam através de um backend comum a elas via HTTP e também via websockets no caso da troca de mensagens em tempo real.

De acordo com (SOMMERVILLE, 2011) requisitos funcionais são declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações.

É no levantamento e análise de requisitos que descrevemos o que um sistema vai fazer, que vai desde a descrição e objetivo geral do sistema, como por exemplo, um sistema de vendas, até objetivos específicos, como por exemplo o sistema terá dois tipos de usuários com perfis e permissões distintas.

O levantamento desses requisitos foi feito através de entrevistas ao cliente onde ele explicou como funciona a estação rodoviária e quais eram os maiores problemas a serem sesolvidos. Foram apresentados softwares similares que faziam uma tarefa parecida, mas não atendiam as especificações dos clientes de nosso cliente. Um dos softwares analisados foi o DSplay[21] que era um sistema de TV corporativa porém não atendia bem ao nível de personalização que o SMM iria atender, o principal deles seria a troca de mensagens em tempo real entre o painel de controle e os terminais. Em comum, identificamos que ambos teriam criação de playlists, templates, mídias que são inerentes a qualquer sistema multimídia existente. Outro sistema analisado foi o GRM que já era de propriedade da Apel, onde este tinha apenas algumas das funcionalidades porém elas foram aproveitadas no SMM como a gestão de mensagens remotamente e o agendamento de tarefas.

Com base nesse levantamento foi possível elencar e refinar os requisitos do sistema e deles extrair os casos de uso que veremos na Tabela 1:

Tabela 1- Requisitos funcionais do SMM

Requisitos Funcionais	**Caso de Uso**
RF01: O sistema deverá permitir aos operadores o monitoramento de terminais e painéis	**UC 01**

[21] https://dsplay.tv/site/

RF02: O sistema deverá permitir aos operadores a gestão de conteúdo em tempo real	**UC 02**
RF03: O sistema deverá permitir aos operadores o cadastro de templates	**UC 03**
RF04: O sistema deverá permitir aos operadores o cadastro de mídias	**UC04**
RF05: O sistema deverá permitir aos operadores criar listas de reprodução de mídias	**UC 05**
RF06: O sistema deverá permitir aos operadores o cadastro de terminais	**UC 06**
RF07: O sistema deverá permitir aos operadores o cadastro de zonas de conteúdo	**UC 07**
RF08: O sistema deverá permitir aos operadores o agendamento de reprodução de conteúdos	**UC 08**
RF9: O sistema deverá permitir aos operadores administrativos o cadastro de novos usuários	**UC 9**
RF10: O sistema deverá permitir aos operadores administrativos o cadastro de papéis no sistema	**UC 10**
RF11: O sistema deverá permitir aos operadores administrativos o cadastro de permissões	**UC 11**
RF12: O sistema deverá permitir aos operadores gerar e carregar backups das configurações feita	**UC 12**
RF13: O sistema deverá permitir aos operadores monitorar via dashboard as trocas de mensagem em tempo real	**UC 13**

Fonte: Próprio Autor

3.2 REQUISITOS NÃO-FUNCIONAIS

Entre os requisitos do cliente sobre o SMM estavam algumas regras e restrições para a implementação do sistema. Algumas delas foram, por exemplo, que não poderíamos contar com a rede de internet, apenas rede local para a operação pois nem todas as estações teriam acesso a internet. O SMM rodaria em um navegador web como o Google Chrome[22], seria feito em HTML, JAVASCRIPT e CSS. O Backend teria que ser escrito utilizando o PHP 7.2 ou superior, o banco de dados seria o MYSQL, deveriam haver trocas de mensagem em tempo real. Caso o terminal fosse desligado, ao reiniciar o aplicativo teria que ser reiniciado junto e já em tela cheia, sem a necessidade de ação humana, a troca de arquivos deveria ser usando HTTP porém deveria ser feito um cacheamento das mídias já baixadas para evitar uso excessivo da rede.

De acordo com (SOMMERVILLE, 2011) requisitos funcionais são restrições aos serviços ou funções oferecidos pelo sistema. Incluem restrições de timing, restrições no processo de desenvolvimento e restrições impostas pelas normas.

Nos requisitos não-funcionais temos um olhar mais crítico para o sistema, pois nele verificamos a confiabilidade, desempenho, proteção e demais parâmetros como por exemplo a obrigatoriedade de um campo de texto. No SMM, os requisitos não funcionais analisados e levantados na Tabela 2 a seguir:

Tabela 2 - Requisitos não funcionais do SMM

Requisitos Não Funcionais
RNF 01: O sistema será composto de três partes: *backend, frontend* de operação e interface de exibição instalada em terminais
RNF 02: O sistema deverá ter proteção de usuário e senha
RNF 03: O sistema deverá ter controle de acesso de usuários, separando por papéis e permissões
RNF 04: O sistema deverá ser executado em navegador Google Chrome ou compatível
RNF 05: O sistema deverá funcionar em tempo real

[22] https://www.google.com/intl/pt-BR/chrome/

RNF 06: O *back end* será feito em PHP 7.*
RNF 07: O *front end* será feito usando css, html e javascript
RNF 08: O banco de dados utilizado será o Mysql 8
RNF 09: O compartilhamento de arquivos será feito via HTTP
RNF 10: O ADMINISTRADOR poderá ter acesso aos dados produzidos pelos operadores OPERADOR
RNF 11: Os OPERADORES só poderão acessar seus dados em suas respectivas zonas de conteúdo
RNF 12: Os OPERADORES só poderão acessar seus dados em suas respectivas zonas de conteúdo

Fonte: Próprio Autor

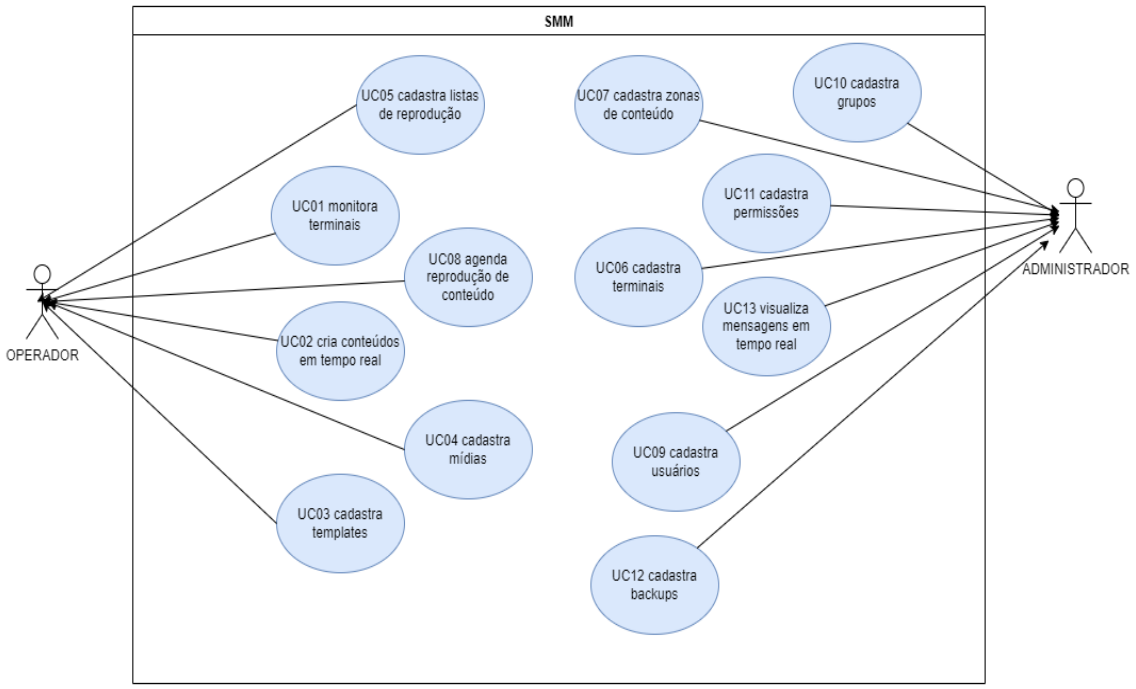
3.3 CASOS DE USO

Os casos de uso identificam quem são os atores e interações e a comunicação entre eles. De acordo com (SOMMERVILLE, 2011) O conjunto de casos de uso representa todas as possíveis interações que serão descritas nos requisitos de sistema.

A Figura 11 define a interação do SMM, identificando as interações entre usuário e sistema e os casos de uso do sistema e na Tabela 2 apresentamos uma breve descrição dos casos de uso.

Os atores do SMM são o OPERADOR que é responsável por manter mídias, templates, playlists, agendamentos e mensagens em tempo real, visualizar o dashboard e o monitoramento ADMINISTRADOR além de fazer tudo que os operadores fazem ainda são responsáveis pelo controle de usuários, permissões e papéis, adição de novos terminais e zonas de conteúdo, backups e monitoramento de mensagens em tempo real por um dashboard feito para esta finalidade.

Figura 11 - Casos de uso SMM



Fonte: Próprio Autor

Tabela 3 - Descrição resumida dos casos de uso

Caso de Uso	Resumo dos casos de uso
UC 01	O operador será capaz de monitorar os terminais online e offline, bem como ver o preview do conteúdo exibido em cada um deles.
UC 02	O operador poderá enviar mensagens e conteúdos em tempo real para os terminais de uma determinada zona de conteúdo.
UC 03	O operador poderá cadastrar templates com até cinco áreas de conteúdo.
UC04	O operador poderá cadastrar novas mídias como áudios, vídeos, imagens, widgets e feeds de notícias.
UC 05	O operador poderá criar listas de reprodução a partir das mídias cadastradas, incluindo o tempo de execução de cada mídia.
UC 06	O administrador poderá cadastrar novos terminais no SMM para conseguir interagir com eles via painel.

UC 07	O administrador poderá cadastrar novas zonas de conteúdo que formarão grupos de terminais
UC 08	O operador poderá cadastrar agendamentos de reprodução de conteúdos respeitando as regras de conflito em horários.
UC 9	O administrador poderá cadastrar novos usuários.
UC 10	O administrador poderá cadastrar novos grupos de usuários.
UC 11	O administrador poderá cadastrar novas permissões
UC 12	O administrador poderá criar novos backups tanto de arquivos quanto de banco de dados, ou de ambos.
UC 13	O administrador poderá monitorar via dashboard a troca de mensagens em tempo real de todos os usuários e terminais.

Fonte: Próprio Autor

3.4 ARQUITETURA

‘ Para (Rad Hat, c2022) API significa interface de programação de aplicações, um conjunto de definições e protocolos para criar e integrar softwares de aplicações, havia a necessidade de criar uma API que seria a fonte de dados tanto do aplicativo do terminal quanto do aplicativo de controle das mídias. Outra parte importante no backend foi a parte de websockets pois precisávamos mandar mensagens também em tempo real, nesses dois casos utilizamos um serviço escrito utilizando o framework Laravel para a implementação. O banco de dados escolhido foi o MySQL pela facilidade de utilização e por seu uso ser gratuito, além de ser um dos bancos de dados mais utilizados no mundo.

O frontend foi dividido entre o aplicativo que seria instalado nos terminais e também roda em um servidor nginx instalado em cada terminal, esse servidor também é responsável por cachear as mídias que já foram baixadas, evitando que haja uma sobrecarga no servidor principal.

Na área de computação, um cache é uma camada de armazenamento físico de dados de alta velocidade que guarda um subconjunto de dados, geralmente temporário por natureza,

para que futuras solicitações referentes a esses dados sejam atendidas de modo mais rápido do que é possível fazer ao acessar o local de armazenamento (Aws, c2022).

Em informática, os servidores web são basicamente programas de computador simples que entregam a página web solicitada sob demanda do cliente da Web (Vasconcelos, 2022), similarmente há o aplicativo de gerenciamento das mídias que também roda em um servidor web nginx como os dos terminais, porém instalados em PCs dependendo da necessidade dos clientes.

Para (Cloud Fare, c2022) uma LAN[23], ou rede local, é um grupo de dispositivos de computação conectados em uma área localizada que geralmente compartilha uma conexão centralizada com a internet. A comunicação será via rede local utilizando o protocolo HTTP para as comunicações síncronas e websocket para as comunicações real time. O diagrama da Figura 2 a seguir apresenta tais módulos e suas conexões.

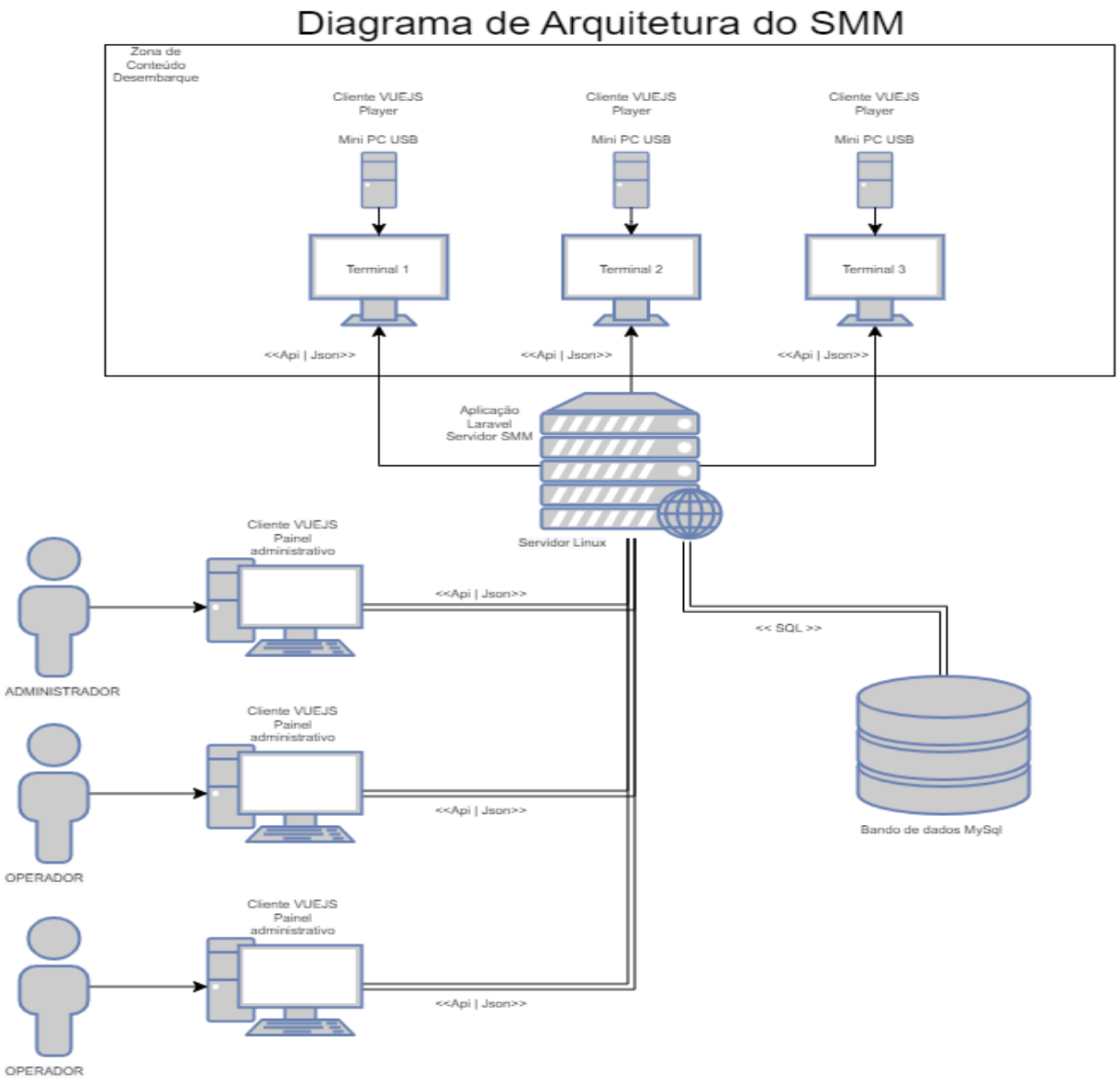
Como explicado acima há uma estratégia de cacheamento das requisições de mídias através de um proxy reverso, (Cloudfare, c2022) diz que um proxy reverso protege os servidores web contra ataques e pode oferecer benefícios de performance e confiabilidade do nginx que possibilita que cada terminal possa funcionar offline em caso de queda do servidor principal pois já teremos as mídias que foram baixadas anteriormente para exibir.

A Figura 12 mostra como o servidor se comunica com o banco de dados, e como cada aplicativo de gestão de mídias envia os conteúdos ao servidor e esses são buscados pelos aplicativos de exibição que estão em cada terminal tanto via HTTP como por websockets.

Os aplicativos de gestão de conteúdo estão instalados em PCs que contém o sistema operacional windows, similarmente os aplicativos dos terminais também rodam em windows que estão instalados em MINI PCs, já o servidor do backend está instalado em uma máquina que contém o Linux instalado. Nessa mesma máquina está rodando o MySQL que foi o banco de dados escolhido para o SMM.

[23] LAN = Local Area Networks, do inglês.

Figura 12 - Diagrama de arquitetura do SMM



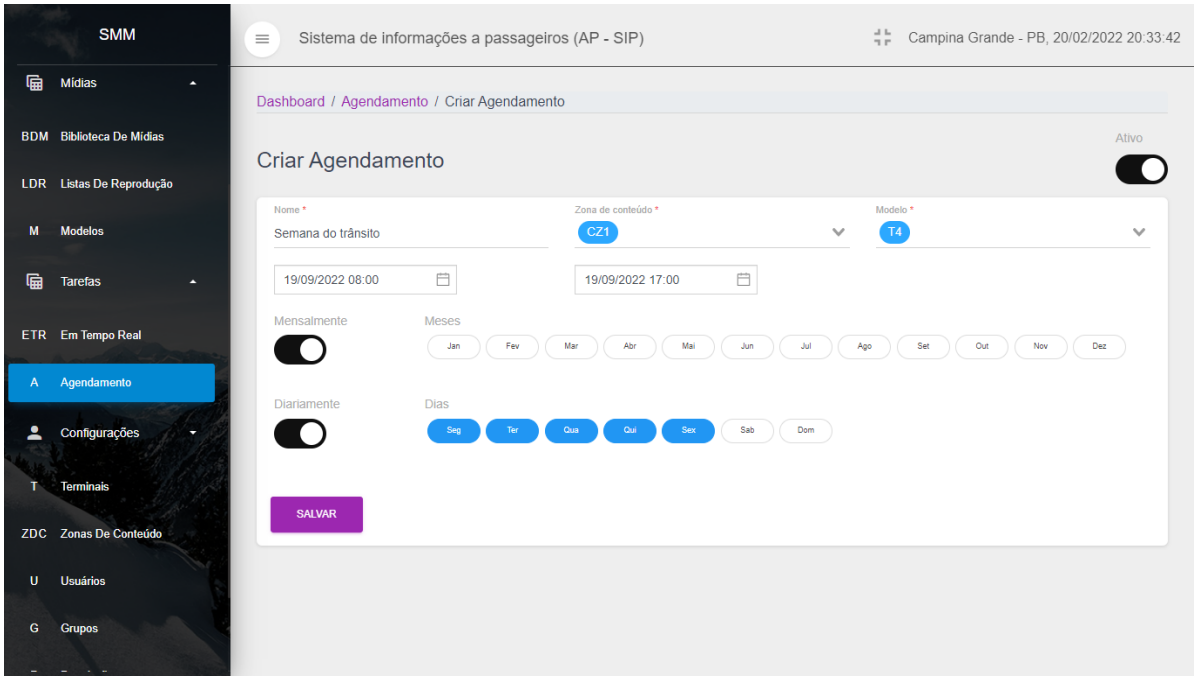
Fonte: Próprio Autor

3.5 PROTÓTIPO

A criação do protótipo foi guiada após o levantamento de requisitos e também com base na descrição que o cliente fez sobre as funcionalidades, também foi baseada na análise de softwares similares como o DSplay e o GRM que tinham o mesmo propósito. Não utilizamos ferramentas de prototipação, criamos um template básico usando apenas HTML, CSS e Javascript e organizamos as telas de acordo com o que definimos.

Na Figura 13 a funcionalidade de criação de agendamentos consiste em criarmos os agendamentos para as zonas de conteúdo. São as tarefas do sistema onde escolhemos o intervalo que queremos que determinado modelo seja exibido nas zonas de conteúdo. É feita uma validação para que não existam conflitos de datas, incluindo conflitos de datas em caso de agendamentos recorrentes, se houver o conflito é dada a possibilidade do usuário editar os agendamentos conflitantes manualmente ou simplesmente sobrescrever tudo permitindo que o agendamento seja feito. A única forma de haver um conflito é se existir algum agendamento com choque de horário e que seja ativo. Se eu escolher sobrescrever os conflitos, os que estiverem conflitando serão desativados para que eu possa cadastrar o agendamento atual.

Figura 13 - Protótipo da tela de criação de agendamentos



Fonte: Próprio Autor

.

3.6 SELEÇÃO DE FRAMEWORKS, BIBLIOTECAS E FERRAMENTAS PARA A CONSTRUÇÃO DO SMM

Nesta etapa foram escolhidas ferramentas e tecnologias que tornassem possível implementar o SMM da forma mais rápida e precisa possível.

Para a organização do trabalho foi utilizado o Notion[24] que é uma ferramenta de criação de listas e tabelas além de outras inúmeras outras funções, com ele foi possível

[24] https://www.notion.so/

colocar a descrição das tarefas permitindo uma rápida busca e organização dos itens a fazer. Para a construção dos diagramas utilizamos o Draw.io[25] que permitiu construir diagramas de entidades e relacionamentos, casos de uso, arquitetura, etc.

Para a codificação escolhemos o Visual Studio Code por ser simples e permitir a instalação de diversos plugins que facilitam a escrita de código em praticamente todas as linguagens de programação.

O banco de dados escolhido foi o MySQL que é amplamente utilizado e é um sistema de fácil manuseio e com documentação excelente, além de ser rápido, confiável, seguro e gratuito para a comunidade. Para facilitar o uso do MySQL utilizamos uma ferramenta chamada HeidSQL[26] que é open source e facilita na visualização de tabelas, configurações, consultas, etc.

ORM (Object Relational Mapper) é uma técnica de mapeamento objeto relacional que permite fazer uma relação dos objetos com os dados que os mesmos representam, (Devmedia, c2022). O backend foi escrito utilizando o Laravel que é um framework PHP amplamente utilizado pela comunidade, ele provê de forma fácil, modelos, controladores, conexão com bancos de dados usando ORMs, criação de migrações, etc.

Como no Laravel várias tarefas complexas já são fornecidas é preciso apenas se preocupar em utilizá-las para resolver seus problemas, com isso os projetos são escritos mais rapidamente diminuindo a chance de erros e os custos da implementação.

Segundo (Kenzie, c2022), backend é tudo aquilo que está por trás da interface de uma aplicação: seus sistemas, banco de dados, toda parte de segurança de dados, envio e recebimento de informações, armazenamento e etc. Já para (Totvs, c2022) essa forma de desenvolvimento se relaciona com o que está por trás das aplicações desenvolvidas na programação. Ou seja, tudo que dá estrutura e apoio às ações do usuário da máquina é chamado de backend.

Uma parte do backend consistiu em comunicação em tempo real, utilizamos websockets para isso onde após determinada ação enviamos a mensagem para um canal e esse canal poderia ser assinado por clientes que desejassem receber essas mensagens, os nossos clientes, neste caso, foram tanto o aplicativo dos terminais quanto da gestão de conteúdos. Utilizamos uma biblioteca dentro do próprio Laravel chamada Laravel Websockets que

[25] https://drawio-app.com/

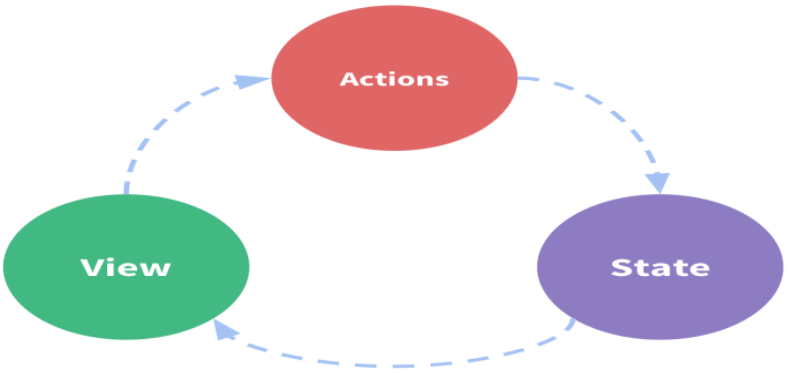
[26] https://www.heidisql.com/

abstrai o uso, bastando apenas a gente criar eventos que emitem essas mensagens para os canais que desejamos.

Para (Kenzie, c2022), frontend é toda parte da programação relativa à interface de uma aplicação (Xpeducacao, c2022) diz que um desenvolvedor front-end é o profissional responsável por criar a interface de utilização de um site ou aplicação web. A camada visual das aplicações web, na qual os usuários interagem, é então de responsabilidade deste desenvolvedor.

O frontend foi escrito utilizando o VueJs que é uma biblioteca javascript bastante popular usada para escrever pequenos pedaços de aplicações ou toda a aplicação de forma mais organizada e fácil. Usa o conceito de componentização e nela também é possível utilizar ciclos de vida de componentes que facilitam na resolução de problemas. Uma das bibliotecas que compõem o VueJs é o Vuex[27] que é uma biblioteca que implementa um padrão conhecido como Flux Pattern que é a gestão dos estados da aplicação que é dividido em **estado** que é a fonte da verdade de um dados, **visão** que é como a aplicação enxerga esse dado no momento atual é uma **ação** que modifica o estado atual dando uma nova verdade a ele como na Figura 14 abaixo.

Figura 14 - Representação do Vuex



Fonte: Vuex Blog (s.d.)

3.7 ESPECIFICAÇÃO

Nesta etapa foi feita uma descrição e organização de todas as funcionalidades a serem desenvolvidas com base no levantamento dos requisitos funcionais e não funcionais que colhemos junto ao cliente. O objetivo foi organizar cada tarefa em pequenas entregas de

[27] https://vuex.vuejs.org/

forma clara, delimitada e precisa. Mencionamos que nosso modelo de desenvolvimento foi incremental, portanto, mudanças inevitáveis ocorreram ao longo do desenvolvimento, porém com base nessa especificação conseguimos ter um guia para o desenvolvimento que diminuiu a chance de erros no desenvolvimento.

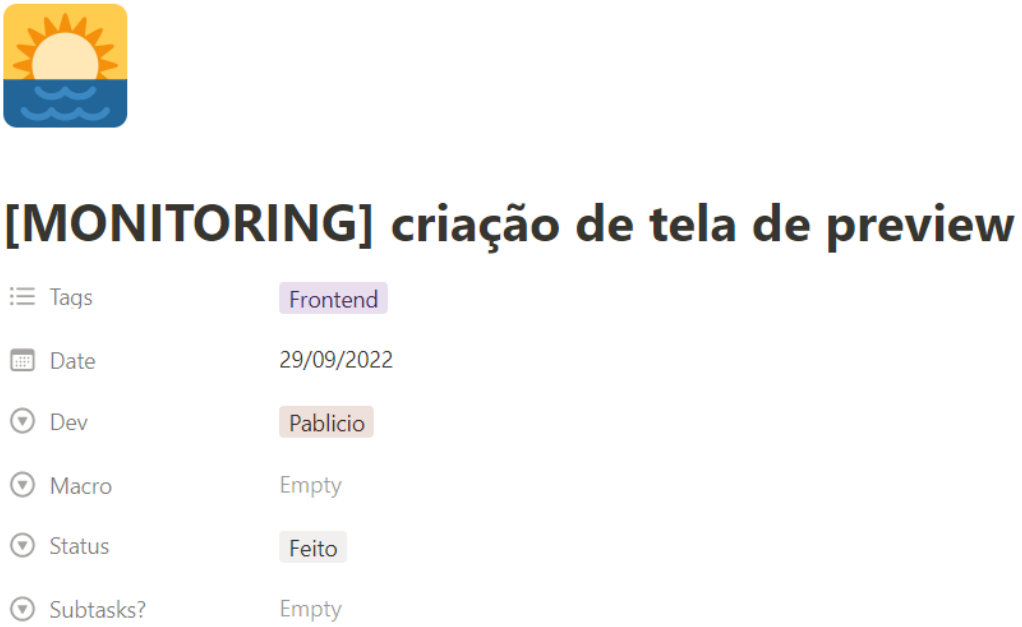
Utilizamos o Notion como ferramenta para registrar as tarefas, descrição das funcionalidades e demais anotações, além de fazer um controle do tempo de execução de cada tarefa e poder marcar os status delas como a fazer, feito, cancelado, priorizado, etc. No notion tem vários modos de organizar as tarefas, como em listas simples, tabelas, quadros Kanban[28], textos, páginas, entre outros, escolhemos organizar páginas onde fizemos um detalhamento da funcionalidade e onde conseguimos anexar protótipos, links úteis, quais seriam as etapas de desenvolvimento, anotações, e exemplos de objetos JSON[29] que fariam parte do desenvolvimento. Cada pedaço (incremento) é desenvolvido de forma linear, como no Modelo em Cascata, e em seguida exposto aos comentários dos clientes (SOMMERVILLE, 2011), dessa forma, tanto os desenvolvedores quanto os clientes conseguiram acompanhar todas as etapas do desenvolvimento, no Modelo Incremental as atividades de Especificação, Projeto, Implementação e Validação são intercaladas, acontecendo em cada nova versão, com rápido feedback entre todas as atividades (SOMMERVILLE, 2011), além de que todos esses itens criados servem de documentação para servir de consulta futuramente.

Abaixo mostramos como exemplo a organização do caso de uso 01, onde foi criada uma tarefa para a funcionalidade de pré-visualização de conteúdo sendo executado em um determinado terminal. Além de um protótipo de como ficaria a pré-visualização, ainda anexamos o método do backend que geraria os dados, um exemplo de payload para a montagem da tela, e como gostaríamos de ter as descrições e o conteúdo dispostos na tela. As Figuras 15, 16, 17 e 18 mostram a tela do Notion onde detalhamos a tarefa.

[28] https://kanbanize.com/

[29] JSON = JavaScript Object Notation, do inglês.

Figura 15 - Cabeçalho de uma página do Notion com a descrição de uma tarefa



Fonte: Imagem do Autor

Figura 16 - Descrição e rascunho da interface da tarefa Preview

Fonte: Imagem do Autor

Figura 17 - Método e seu respectivo retorno do backend para guiar a tarefa Preview

Fonte: Imagem do Autor

Figura 18 - Exemplo de código no frontend e lista de itens faltantes para o Preview

Fonte: Imagem do Autor

4 RESULTADOS

Nesta seção descreveremos como foi construída uma das principais funcionalidades do sistema, o agendamento de tarefas a serem executadas, desde o levantamento das entidades até a implementação. Iremos mostrar tanto a implementação do backend usando o Laravel, quanto o front end utilizando o VueJS.

Aqui também mostraremos como levantamos os casos de teste que foram feitos de forma manual para garantir a validação da funcionalidade.

4.1 AGENDAMENTO DE TAREFAS

Uma das funcionalidades que precisamos implementar foi a possibilidade de agendar tarefas que seriam executadas nos terminais, poderíamos criar, editar e excluir um agendamento. Para criar um agendamento precisamos passar a data de início, a data de termino, qual template seria usado, qual seria a zona de conteúdo em que esse agendamento seria criado e se ele seria recorrente. Para ser recorrente, dariam as horas de início e fim como também em quais meses e dias da semana esse agendamento sempre iria ser tocado.

Uma das restrições impostas a essa funcionalidade era que caso já houvessem agendamentos com algum conflito antes, durante ou depois da faixa de horário escolhido, uma mensagem seria apresentada ao usuário informando do conflito e perguntando se ele desejava alterar os registros conflitantes, ou simplesmente tornados inativos para que pudesse prosseguir com o novo agendamento. A regra dos conflitos valia inclusive se existisse algum agendamento em dia diferente do atual mas que fosse um agendamento recorrente.

Abaixo apresentamos um gráfico na Figura 19 contendo uma explicação sobre quais os casos possíveis de conflitos, na seção de testes deste capítulo entraremos em maiores detalhes sobre este fluxo, mas para adiantar aqui podemos ver que podem haver conflitos em quatro situações.

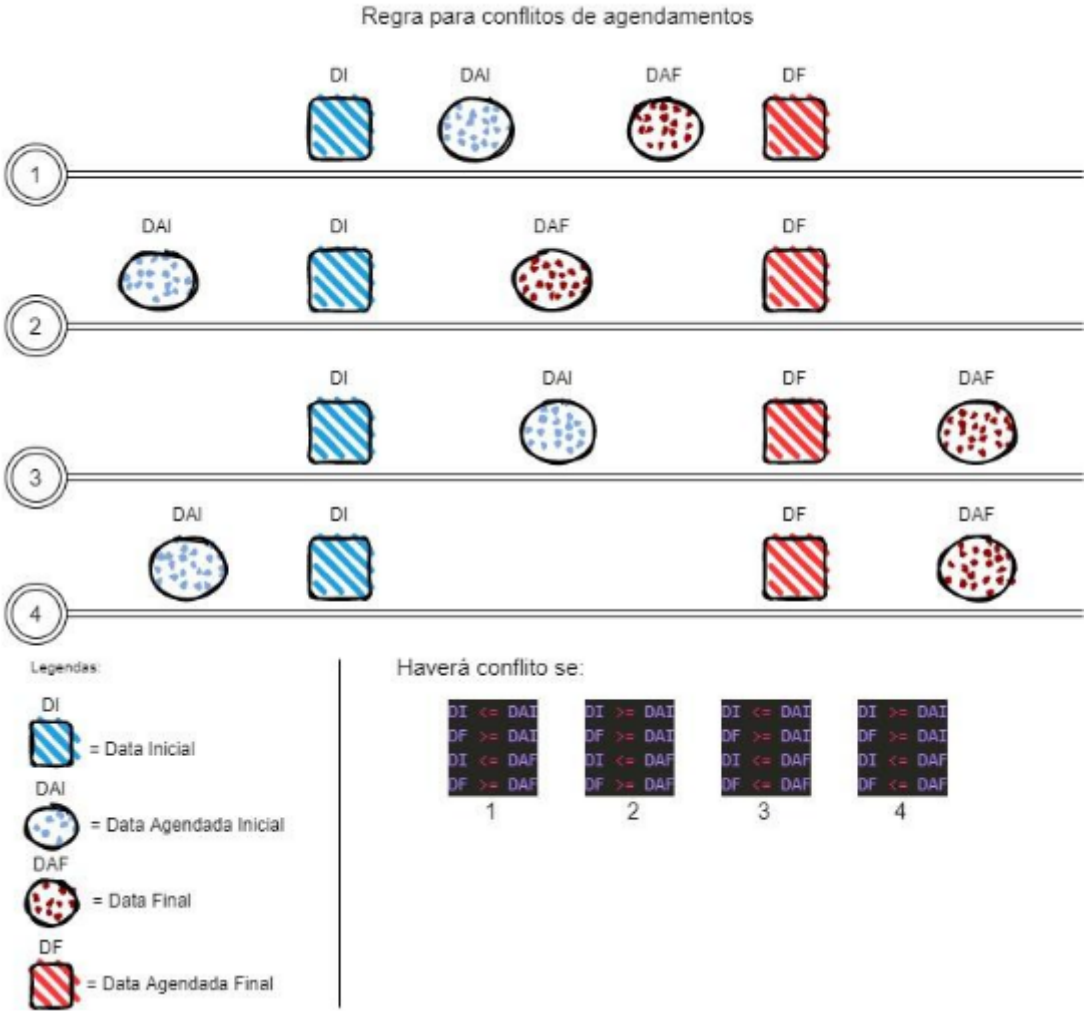
- Na linha 1 há um choque pois a data inicial e final estão sobre datas iniciais e finais agendadas.

- Na linha 2 a data inicial está depois de uma data inicial agendada e antes de uma data final agendada.

● Na linha 3 a data inicial está antes de uma data inicial agendada e a data final depois de uma data inicial agendada.

● Na linha 4 as datas iniciais e finais estão entre datas já agendadas.

Figura 19 - Gráfico para o entendimento da regra de conflitos de agendamentos



Fonte: Imagem do Autor

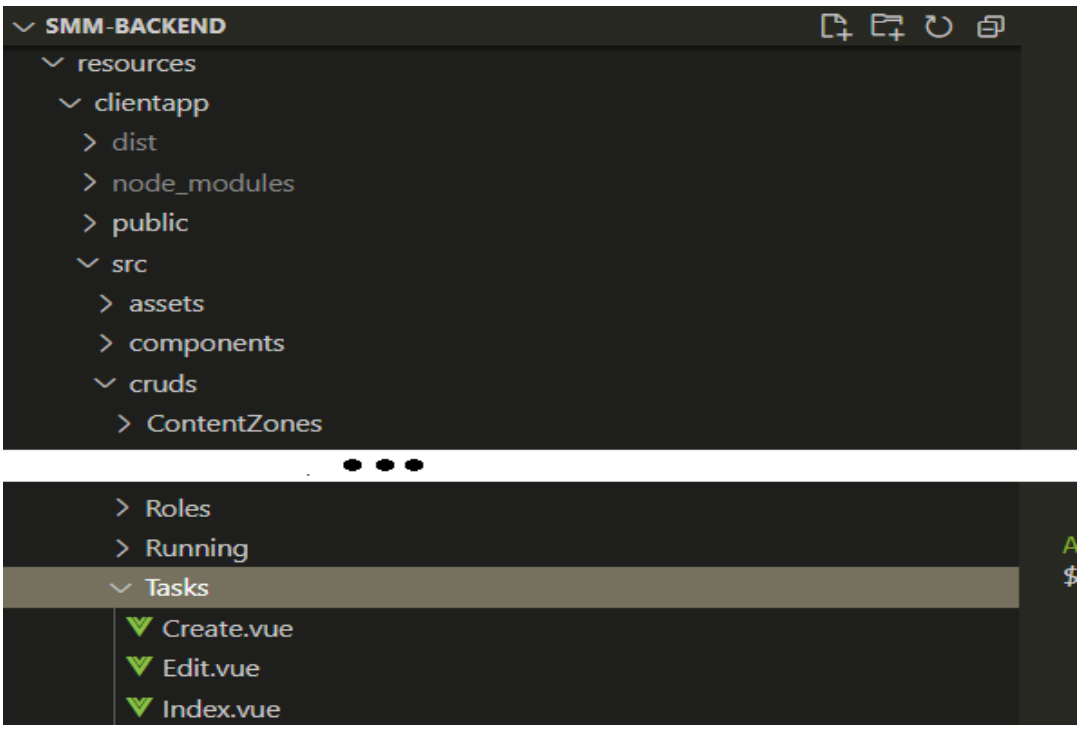
4.2 IMPLEMENTAÇÃO

Detalharemos a implementação do backend e frontend do agendamento, mostrando o diagrama de entidade e relacionamento desse módulo e a organização dos arquivos que foram criados.

4.2.1 Estrutura de arquivos no frontend

Como já mencionado usamos o VueJS para criar as páginas do SMM, para o agendamento organizamos as páginas em Index.vue que seria a tela de visualização de todos os agendamentos criados, Create.vue que seria a tela de criação de novos agendamentos e Edite.vue que seria a de edição. Segundo (Mozilla, c2022), CRUD (Create, Read, Update, Delete) é um acrônimo para as maneiras de se operar em informação armazenada. Para melhor organização criamos uma pasta chamada cruds, onde estará a nossa camada de visualização do frontend como veremos a seguir na Figura 20.

Figura 20 - Estrutura de arquivos das páginas do SMM



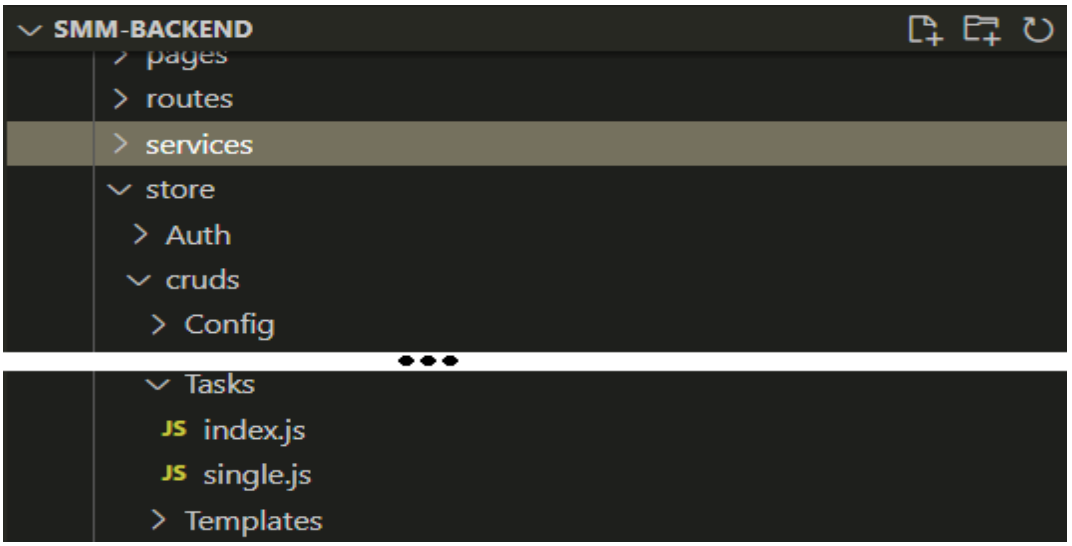
Fonte: Imagem do Autor

Para organizar a parte do Vuex, que é a biblioteca de controle de estados da aplicação, criamos uma pasta store, dentro dela criamos um mapeamento um para um com a parte da

camada de visualização citada anteriormente, a Figura 21 mostra a organização desses arquivos.

Por padrão criamos um arquivo chamado index.js para as requisições do tipo métodos GET que estão sendo chamadas neste arquivo e single.js para os métodos do tipo, POST, PUT e DELETE que são chamados a partir deste arquivo. Posteriormente iremos detalhar alguns desses métodos.

Figura 21 - Estrutura de arquivos do vuex



Fonte: Imagem do Autor

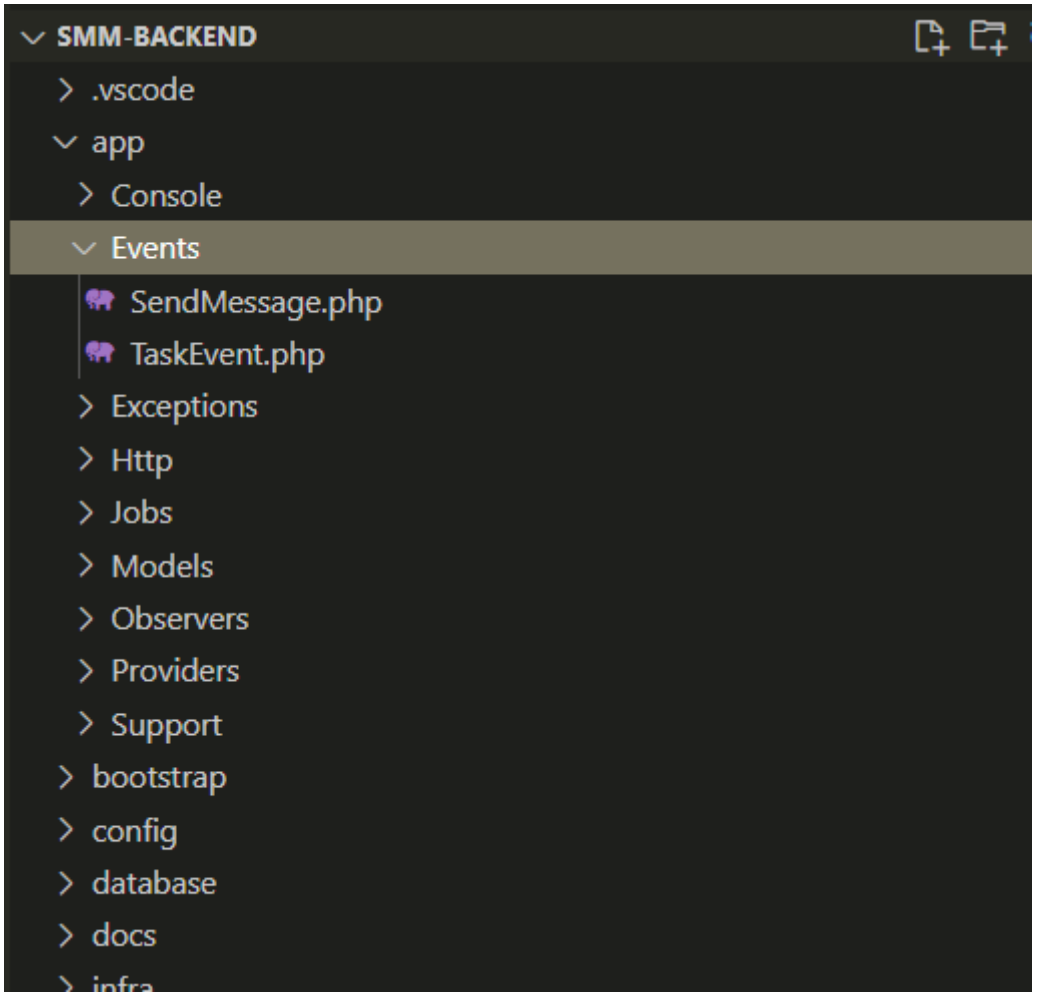
4.2.2 Estrutura de arquivos no backend

Assim como no VueJS, também podemos organizar os arquivos no Laravel da forma que desejarmos dependendo das necessidades. No nosso caso organizamos nossa funcionalidade em algumas camadas, Começando pela pasta Models, onde criamos uma pasta chamada SMM e um arquivo chamado Task.php para ser o nosso modelo.

Em programação uma classe é uma forma de definir um tipo de dado em uma linguagem orientada a objeto. Ela é formada por dados e comportamentos (devmedia, c2022). Para a camada dentro da camada HTTP criamos diversas classes. Nosso controller, TaskApiController.php, dois arquivos para nossas regras de validação de request chamados TaskUpdateRequest.php e TaskStoreRequest.php, criamos na camada de Obsevers um TaskObserver.php que realiza realiza algumas ações quando criamos, editamos ou excluímos algum agendamento, na camada de Events e criamos um TaskEvent.php que envia dados a alguns canais quando esse arquivo é chamado por alguma outra classe.

Na imagem abaixo mostramos como exemplo, o arquivo TaskEvent.php, os demais arquivos listados acima estão organizados de maneira parecida, mais adiante mostraremos internamente a codificação de alguns desses arquivos como podemos observar na figura 22.

Figura 22 - Estrutura de arquivos do backend



Fonte: Imagem do Autor

4.2.3 Desenvolvimento

Nesta seção explicaremos a codificação de um dos fluxos da funcionalidade de agendamento de ponta a ponta, desde a requisição no backend até a apresentação no frontend.

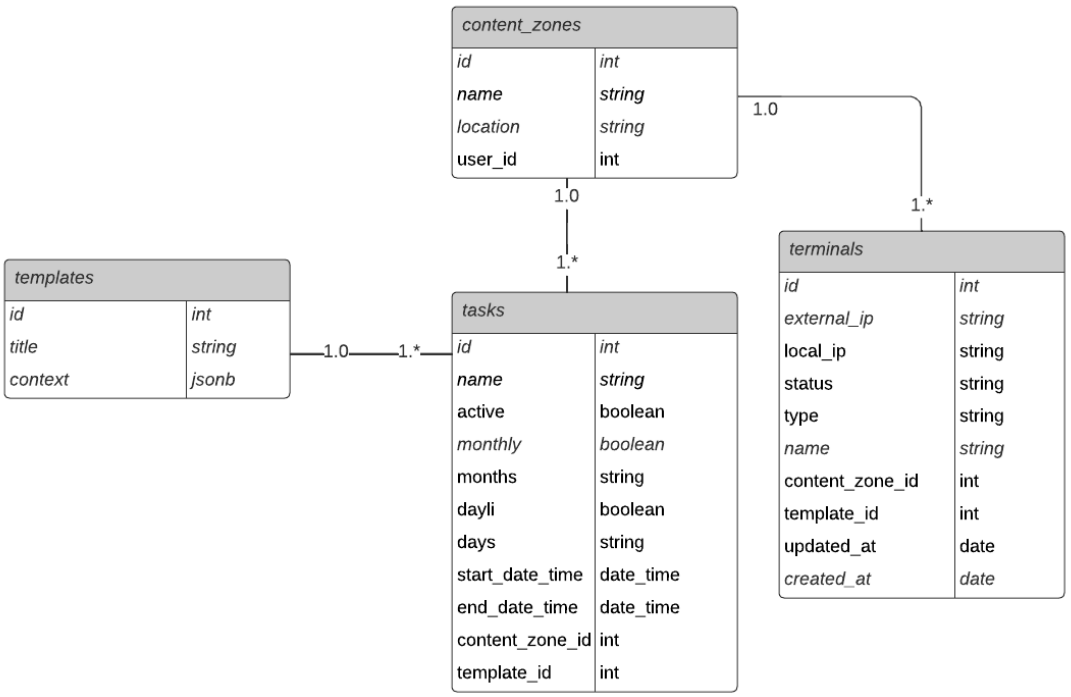
Escolhemos a funcionalidade criar um agendamento que consiste em preencher um formulário com os dados já descritos e em caso de sucesso salvar, ou caso haja conflitos apresentar os fluxos alternativos.

Um Diagrama de Entidade e Relacionamento (DER) é uma representação gráfica do Modelo de Entidade e Relacionamento. Enfim, isso evita excesso de abstração, trazendo para

a realidade informações pertinentes de uma forma mais visual e, consequentemente, mais intuitiva. (Codesh, c2022).

Apresentaremos o diagrama de DER (Figura 23) desse módulo e mostraremos os detalhes (Tabela 4) para facilitar o entendimento. Primeiramente, a nível de codificação o nome da entidade que representa o agendamento é uma task, ou uma tarefa que o sistema irá executar mediante um agendamento, essa task se relaciona diretamente a um template pois toda task possui um template que contém todo o conteúdo a ser exibido. Esse agendamento está atribuído a uma ou mais zonas de conteúdo que por sua vez, possuem um ou vários terminais.

Figura 23 - Diagrama de entidade e relacionamento do módulo de agendamentos



Fonte: Imagem do Autor

Tabela 4 - Modelo de agendamentos de tarefas

Nome	**Tipo**	**Descrição**
id	integer	Identificador único da tabela tasks, serve como chave primária.
name	string	Nomeia o agendamento para que seja melhor

		identificado nas buscas.
active	boolean	Se for verdadeiro, o agendamento estará ativado, sendo possível visualizar seu conteúdo nos terminais, caso esteja falso, não será possível.
monthly	boolean	Indica se há recorrência mensal do agendamento.
months	string	Guarda os meses em que o agendamento está configurado, ex.: jan, fev, abril…
daily	boolean	Indica se há recorrência diária do agendamento.
days	string	Guarda os dias em que o agendamento está configurado, ex.: seg, ter, sex, sab…
start_date_time	date time	Guarda a o horário de início do agendamento.
end_date_time	date time	Guarda o horário de fim do agendamento.
content_zone_id	integer	Chave estrangeira para a entidade Zona de Conteúdo.
template_id	integer	Chave estrangeira para a entidade Template.

Fonte: Próprio Autor (2022)

O modelo que mapeia essa entidade é a classe Task.php, como estamos usando o Laravel, atribuímos a tabela ao modelo através da variável $table figura abaixo e os atributos através da variável $fillable como vemos na Figura 24.

Figura 24 - Classe para o modelo de agendamentos

Task.php M

app > Models > Smm > Task.php > ...

```php
<?php

namespace App\Models\Smm;

use App\Models\BaseEntity;
use App\Support\TaskSupport;
use Illuminate\Support\Facades\DB;

class Task extends BaseEntity
{
    use TaskSupport;

    const LIMIT_VALUE_MONTHS_DAYS = 1;

    public $table = 'tasks';

    protected $fillable = [
        'id',
        'name',
        'active',
        'monthly',
        'months',
        'dayli',
        'days',
        'start_date_time',
        'end_date_time',
        'content_zone_id',
        'user_id',
        'template_id'
    ];
```

Fonte: Imagem do Autor

No fluxo de criação de um novo agendamento há uma requisição do tipo POST ao backend através da roda /api/v1/tasks, o mapeamento dessa rota foi feito no arquivo routes/api.php mostrado na Figura 25 a seguir:

Figura 25 - Rota para criação de agendamentos

```php
// Tasks
Route::post('/tasks', 'TasksApiController@store');
```

Fonte: Imagem do Autor

Essa rota cai no método store do TaskApiController.php e na linha 28 Figura 26 é checado se existe conflito de agendamentos (Explicaremos com mais detalhes essa regra no capítulo de testes). Se houver e for não for solicitado que essa task sobrescreva as conflitantes e ela estiver com o campo active true, é mostrada uma mensagem de erro, informando que há conflitos é dada a opção de sobrescrevê-los. Se o usuário optar por sobrescrever, cai no else if da linha 36 e os demais conflitos são inativados para que o atual agendamento seja criado.

Figura 26 - Método de criação do agendamento no backend

```
25
26    public function store(StoreTaskRequest $request)
27    {
28      $has_conflict = Task::scheduleConflictCreate($request);
29
30      if ($has_conflict->count() && !$request->override && $request->active) {
31        throw ValidationException::withMessages([
32          "errors" => "Foram encontrados conflítos nas datas selecionadas com os agendamentos abaixo!",
33          "tasks_conflicts" => $has_conflict,
34        ]);
35      } elseif ($has_conflict->count() && $request->override) {
36        Task::inactiveConflicts($has_conflict->pluck('id'));
37      }
38
39      $task = Task::create($request->validated());
40
41      return (new TaskResource($task))
42        ->response()
43        ->setStatusCode(Response::HTTP_CREATED);
44    }
45
```

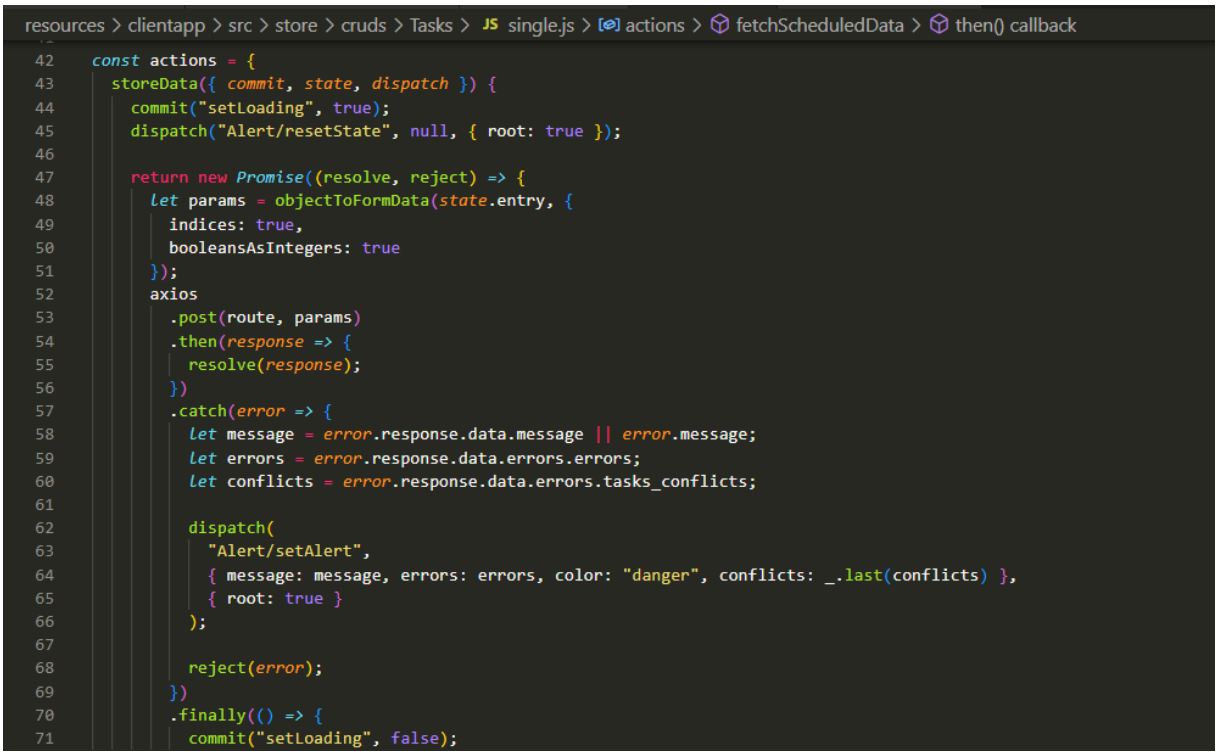
Fonte: Imagem do Autor

Em caso de sucesso, o registro é retornado ao front end com status 201 (created).

No front a iteração começa com o carregamento da página de agendamentos, onde são trazidas as listas de templates e zonas de conteúdo previamente cadastradas.

Devido a grande extensão do arquivo Create.vue, aqui mostrarei apenas o trecho em que os dados da tela são carregados pelo método fetchCreateData() da linha 407 da Figura 27 e dos métodos que setam os campos compreendidos entre as linhas 383 a 405, onde a cada mudança é setado um novo valor, em todos esses casos, são chamados métodos que estão implementados na camada do Vuex que já apresentamos antes. Após o preenchimento do formulário, caso esteja correto, é chamado o método storeData() que faz o post para o backend completando o fluxo.

Figura 27 - Método para a criação de agendamentos no frontend

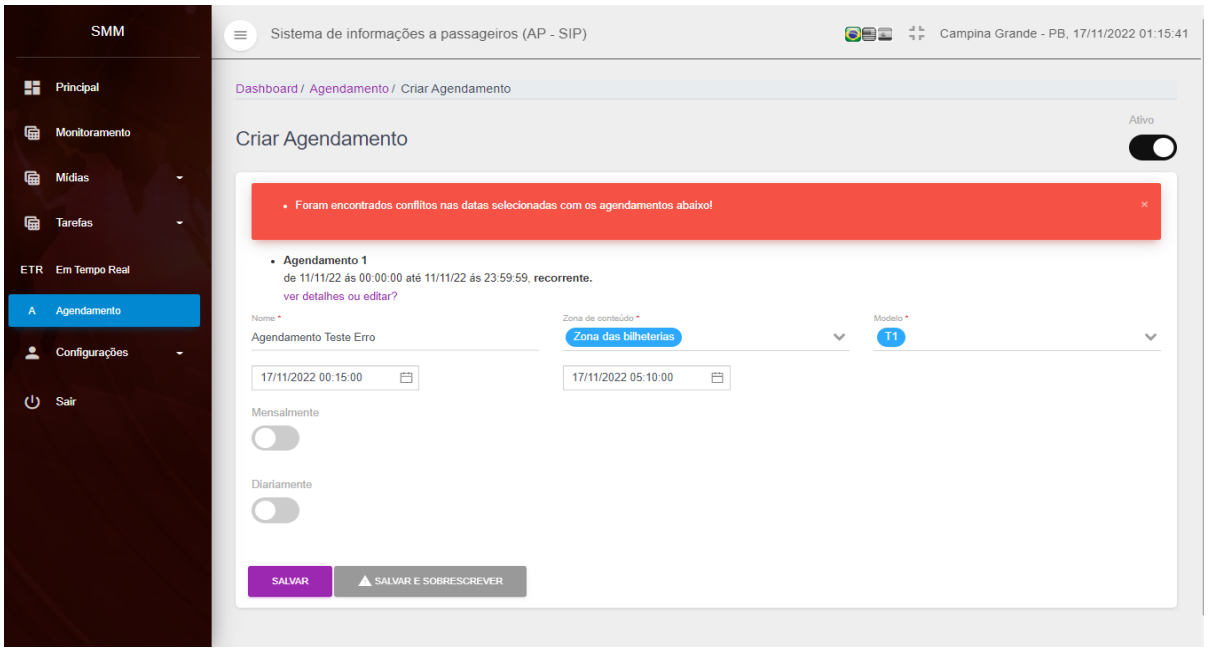


Fonte: Imagem do Autor

Caso não haja conflitos oriundos do backend, o fluxo termina e é apresentada uma mensagem de sucesso, caso haja conflitos é apresentada uma mensagem de erro e um botão para sobrescrever também é mostrado, um detalhe a ser levado em consideração é que só há conflitos para agendamentos pertencentes à mesma zona de conteúdo.

A Figura 28 mostra a tela em caso de erro na zona das bilheterias onde o Agendamento Teste Erro conflita com o Agendamento 1 que é recorrente todas as quintas de novembro, portanto não seria possível fazer o Agendamento Teste Erro por ser justamente uma quinta de novembro a data escolhida.

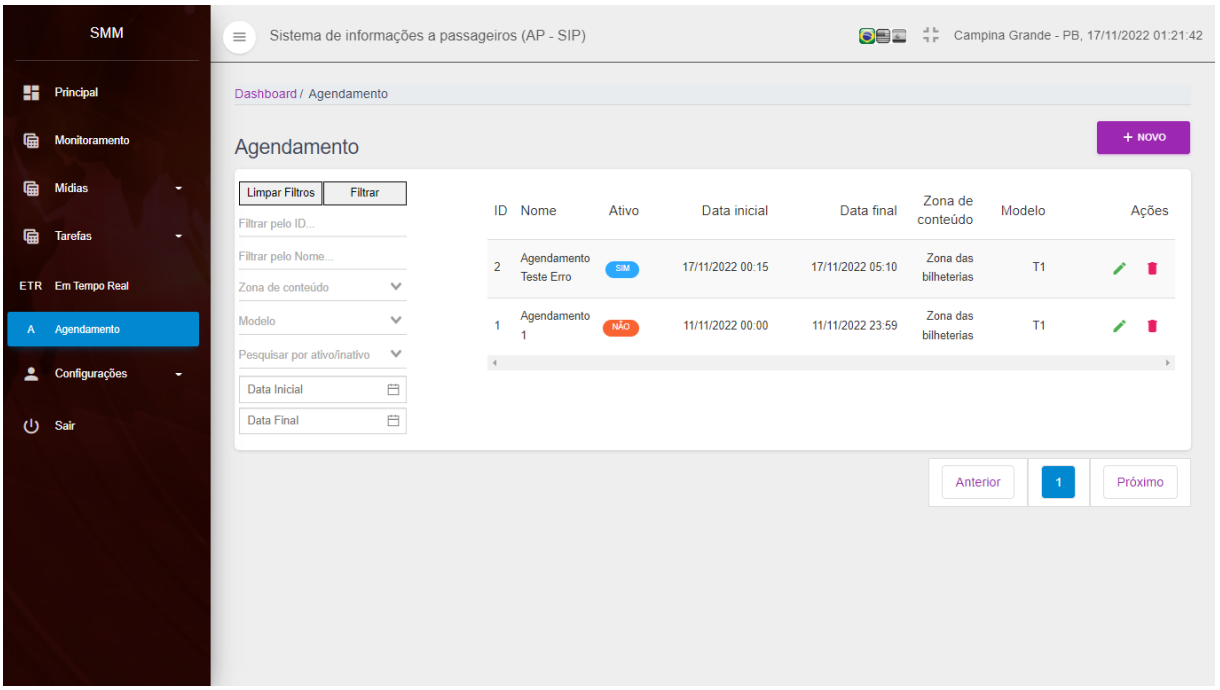
Figura 28 - Protótipo da tela de agendamentos quando ocorre um conflito



Fonte: Imagem do Autor

Como houve conflito um fluxo possível é clicar em Salvar e Sobrescrever, neste caso os agendamentos conflitantes seriam desativados para que o nosso pudesse prosseguir, na Figura 29 está o resultado após a escolha dessa opção:

Figura 29 - Tela de listagem de agendamentos



Fonte: Imagem do Autor

4.3 TESTES

De acordo com (Cron App, c2022) o teste de software é um processo de checagem aplicado a programas de computador em fase de desenvolvimento. É essencial para que o produto final seja entregue ao cliente funcionando dentro das expectativas.

Os testes do SMM foram realizados de forma manual, sempre após a finalização de cada funcionalidade. A cada incremento foi gerada uma versão para que o cliente pudesse realizar suas validações e quando algum ponto não atendia eles retornavam com os erros para que fossem corrigidos. Não tivemos a implementação de testes unitários pois o tempo de desenvolvimento foi limitado, porém é sabido da importância desse tipo de testes por trazer a segurança de que os requisitos foram atendidos e além disso, a adição de novas funcionalidades se torna menos suscetível à inserção de erros onde antes estava funcionando corretamente.

A seguir vamos mostrar os testes feitos para os agendamentos, descrevendo os cenários, as possibilidades de agendamentos e o resultado. Para facilitar a leitura dos testes convencionamos que “di” seria a data inicial do agendamento e “df” a data final.

4.3.1 Casos de teste para os agendamentos de tarefas

Em cada cenário, criamos um agendamento válido, e posteriormente tentamos criar agendamentos em horários que podiam ou não gerar conflitos, usamos o “não” para informar que a entrada não seria aceita por gerar conflitos, e sim para dizer que a entrada seria aceita, por não gerar conflitos. Quando falamos de recorrência, estamos dizendo que já há um agendamento em outro dia para aquele horário, que deve ser repetido, então não podemos ocupar aquele horário, até que o agendamento recorrente seja desativado.

Cenário 1: ENTRE AGENDAMENTOS SEM RECORRÊNCIA.

1.1 Agendamentos sem recorrência di 09/06/2022 01:00 e df 09/06/2022 04:00 ativo para o mesmo dia.

Tabela 5 - Agendamento para o mesmo dia sem recorrência.

Entrada	Permitir agendamento?	Resultado
09/06/2022 02:00 e df 09/06/2022 03:00	não	OK
09/06/2022 00:30 e df 09/06/2022 02:00	não	OK
09/06/2022 02:00 e df 09/06/2022 05:00	não	OK
09/06/2022 00:30 e df 09/06/2022 05:00	não	OK

Fonte: Próprio autor

1.2 Agendamentos sem recorrência di 09/06/2022 01:00 e df 09/06/2022 04:00 ativo para dias anteriores.

Tabela 6 - Agendamentos para dias anteriores sem recorrência.

Entrada	Permitir agendamento?	Resultado
08/06/2022 02:00 e df 08/06/2022 03:00	sim	OK
08/06/2022 00:30 e df 08/06/2022 02:00	sim	OK
08/06/2022 02:00 e df 08/06/2022 05:00	sim	OK
08/06/2022 00:30 e df 08/06/2022 05:00	sim	OK

Fonte: Próprio autor

1.3 Agendamentos sem recorrência di 09/06/2022 01:00 e df 09/06/2022 04:00 ativo para dias posteriores.

Tabela 7 - Agendamentos para dias posteriores sem recorrência.

Entrada	Permitir agendamento?	Resultado
10/06/2022 02:00 e df 10/06/2022 03:00	sim	OK
10/06/2022 00:30 e df 10/06/2022 02:00	sim	OK
10/06/2022 02:00 e df 10/06/2022 05:00	sim	OK
10/06/2022 00:30 e df 10/06/2022 05:00	sim	OK

Fonte: Próprio autor

Cenário 2: ONDE O AGENDAMENTO ALVO É RECORRENTE E O AGENDAMENTO ASPIRANTE NÃO É RECORRENTE.

Agendamento recorrente di 09/06/2022 01:00 e df 09/06/2022 04:00 para qualquer agendamento que haja colisão de dia, mês e horário com o alvo.

Tabela 8 - Agendamento recorrente com conflitos.

Entrada	Permitir agendamento?	Resultado
XX/06/2022 02:00 e df XX/06/2022 03:00	não	OK
XX/06/2022 00:30 e df XX/06/2022 02:00	não	OK
XX/06/2022 02:00 e df XX/06/2022 05:00	não	OK
XX/06/2022 00:30 e df XX/06/2022 05:00	não	OK

Fonte: Próprio autor

Cenário 3: ONDE O AGENDAMENTO ALVO NÃO É RECORRENTE E O AGENDAMENTO ASPIRANTE É RECORRENTE

Agendamento não recorrente di 11/06/2022 06:00 e df 11/06/2022 09:00 em que o agendamento aspirante tenha colisão de dia, mês e horário com o alvo.

Tabela 9 - Agendamento não recorrente com conflitos.

Entrada	Permitir agendamento?	Resultado
XX/06/2022 07:00 e df XX/06/2022 08:00	não	OK
XX/06/2022 05:30 e df XX/06/2022 07:00	não	OK
XX/06/2022 07:00 e df XX/06/2022 10:00	não	OK
XX/06/2022 05:30 e df XX/06/2022 10:00	não	OK

Fonte: Próprio autor

Cenário 4: ONDE O AGENDAMENTO ALVO E ASPIRANTE SÃO RECORRENTES

Agendamento recorrente di 09/06/2022 01:00 e df 09/06/2022 04:00 em que o agendamento aspirante tenha colisão de dia, mês e horário com o alvo.

Tabela 10 - Agendamento recorrente, onde o agendamento aspirante também é recorrente.

Entrada	Permitir agendamento?	Resultado
XX/06/2022 02:00 e df XX/06/2022 03:00	não	OK
XX/06/2022 00:30 e df XX/06/2022 02:00	não	OK
XX/06/2022 02:00 e df XX/06/2022 05:00	não	OK
XX/06/2022 00:30 e df XX/06/2022 05:00	não	OK

Fonte: Próprio autor

5 CONCLUSÃO

Com o grande número de pessoas que precisam se deslocar diariamente e com a tendência de crescimento desse volume, ficou claro a importância de se construir um sistema multimídia que conseguisse informar a todos de forma rápida, precisa e sem a necessidade de pessoas para trocar as mídias presencialmente em cada terminal. A implementação do SMM tornou possível criar conteúdos, agendamentos e enviar mensagens em tempo real de maneira remota, diminuindo o tempo de troca das informações e gerando benefícios aos passageiros, pois não são apenas mensagens informativas e de entretenimento, como também mensagens de emergência e mensagens inclusivas para pessoas com necessidades especiais que são possíveis graças a esse tipo de sistema.

Com a automação desse processo, foi possível liberar o tempo dos funcionários que precisam configurar presencialmente cada terminal sempre que a programação mudasse, gerando economia de tempo e dinheiro para as empresas que tiverem esse sistema.

O SMM foi implementado de maneira incremental separado por módulos, tanto no front end quanto no backend, seguindo boas práticas de arquitetura e desenvolvimento, facilitando futuramente a inclusão de outras funcionalidades.

Em todas as etapas do desenvolvimento procuramos levantar de forma clara os requisitos, além de criar documentações que serviram de guia para evitar erros ao máximo no desenvolvimento. Este sistema foi desenvolvido de forma incremental, portanto, devem surgir mais iterações buscando criar novas funcionalidades para a melhoria da eficiência e usabilidade.

Para trabalhos futuros buscaremos fazer um estudo de como a utilização de sistemas multimídia impactam a vida dos usuários utilizando, para isso uma comparação entre lugares que possuam ou não esses sistemas.

REFERÊNCIAS

AMAZON. **O que é o armazenamento em cache?**. 2022? Disponível em: <https://aws.amazon.com/pt/caching/#:~:text=Na%20%C3%A1rea%20de%20computa%C3%A7%C3%A3o%2C%20um,acessar%20o%20local%20de%20armazenamento>. Acesso em: 14 nov. 2022.

ASHWIN P. HTTP And Web Servers. **Medium**. 24 jun. 2019. Disponível em: <https://medium.com/@ashwin7411/http-and-web-servers-a8b71bd58763>. Acesso em: 20 nov. 2022.

ATLASSIAN. **Git Branch**. 2022? Disponível em: <https://www.atlassian.com/git/tutorials/using-branches>. Acesso em: 18 nov. 2022.

ATLASSIAN. **O que é Git.** 2022? Disponível em: <https://www.atlassian.com/br/git/tutorials/what-is-git>. Acesso em: 18 nov. 2022.

CADU. **ORM : Object Relational Mapper.** 2011. Disponível em: <https://www.devmedia.com.br/orm-object-relational-mapper/19056>. Acesso em: 15 nov. 2022.

CLOUDFARE. **O que é um proxy reverso? | Servidores proxy explicados.** 2022? Disponível em: <https://www.cloudflare.com/pt-br/learning/cdn/glossary/reverse-proxy/>. Acesso em: 15 nov. 2022.

CLOUDFLARE. **O que é uma LAN (rede local)?.** 2022? Disponível em: <https://www.cloudflare.com/pt-br/learning/network-layer/what-is-a-lan/>. Acesso em: 15 nov. 2022.

COODESH. **Entenda o que é Diagrama de Entidade e Relacionamento (DER).** 2022? Disponível em: <https://coodesh.com/blog/candidates/entenda-o-que-e-diagrama-de-entidade-e-relacionamento-der/>. Acesso em: 16 nov. 2022.

COODESH. **O que é arquitetura MVC?.** 2022? Disponível em: <https://coodesh.com/blog/dicionario/o-que-e-arquitetura-mvc/>. Acesso em: 22 nov. 2022.

DEVMEDIA. **Principais conceitos da Programação Orientada a Objetos.** 2022? Disponível em: <https://www.devmedia.com.br/principais-conceitos-da-programacao-orientada-a-objetos/32285>. Acesso em: 8 nov. 2022.

DIAS, R.D. O Modelo Incremental. **Medium.** 22 ago. 2019. Disponível em: <https://medium.com/contexto-delimitado/o-modelo-incremental-b41fc06cac04>. Acesso em: 17 nov. 2022.

LANZONI, C.O.L.; SCARIOT;, C.A.S.; SPINILLO, C.G.S. Sistema de informação de transporte público coletivo no Brasil: algumas considerações sobre demanda de informação dos usuários em pontos de parada de ônibus. **Infodesign**, São Paulo, v. 8, ano 2, n. 1808-5377, p. 57, maio 2011. Disponível em: <https://www.infodesign.org.br/infodesign/article/viewFile/114/109>. Acesso em: 10 nov. 2022.

LENON. Node.js – O que é, como funciona e quais as vantagens. **Opus-software**. 5 set. 2018. Disponível em: <https://www.opus-software.com.br/node-js/#>. Acesso em: 10 nov. 2022.

LONGEN, A.L. O Que é MySQL – Guia para Iniciantes. **Weblink**. 25 set. 2019. Disponível em: <https://www.weblink.com.br/blog/o-que-e-mysql/>. Acesso em: 11 nov. 2022.

MANSUR BERNARDO *et al*, C.T. **SISTEMAS DE INFORMAÇÃO AO USUÁRIO DO TRANSPORTE COLETIVO:** Análise de Projeto Tecnológico para a Empresa Unida de Transporte Coletivo Rodoviário. 2019. TCC (Especialização em Gestão de Negócios.) - FUNDAÇÃO DOM CABRAL, Belo Horizonte.

MELO, D.M. O que é Laravel? [Guia para iniciantes]. **Terra**. 9 fev. 2021. Disponível em: <https://www.terra.com.br/byte/o-que-e-laravel-guia-para-iniciantes,484f79c68a65056f70849370f2726a8537c5xnwj.html>. Acesso em: 9 nov. 2022.

MOZILLA. **HTML básico.** 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Getting_started_with_the_web/HTML_basics>. Acesso em: 12 nov. 2022.

MOZILLA. **O que é CSS?.** 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/CSS/First_steps/What_is_CSS>. Acesso em: 12 nov. 2022.

MOZILLA. **O que é JavaScript?.** 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript>. Acesso em: 13 nov. 2022.
MOZILLA. **Uma visão geral do HTTP.** 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>. Acesso em: 16 nov. 2022.

PHP. **O que é o PHP?.** 2022? Disponível em: <https://www.php.net/manual/pt_BR/intro-whatis.php>. Acesso em: 13 nov. 2022.

RAD HAT. Rad Hat. **O que é uma API?**. 2 jun. 2022. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Acesso em: 14 nov. 2022.

RICARDO DIAS, R.S. O Modelo Incremental. **Medium**. 22 ago. 2019. Disponível em: <https://medium.com/contexto-delimitado/o-modelo-incremental-b41fc06cac04>. Acesso em: 22 nov. 2022.

ROVEDA, U.R. O QUE É BACK-END, PARA QUE SERVE E COMO APRENDER EM 2021**. Kenzie**. 15 jan. 2021. Disponível em: <https://kenzie.com.br/blog/back-end/>. Acesso em: 16 nov. 2022.

ROVEDA, U.R. O QUE É FRONT-END, PARA QUE SERVE E COMO APRENDER FRONT-END**. Kenzie**. 18 dez. 2020. Disponível em: <https://kenzie.com.br/blog/front-end/>. Acesso em: 16 nov. 2022.

ROVEDA, U.R. O QUE É UM PROTÓTIPO, QUAIS OS TIPOS, POR QUE USAR E COMO FAZER?. **Kenzie**. 16 ago. 2021. Disponível em: <https://kenzie.com.br/blog/prototipo/>. Acesso em: 13 nov. 2022.

SCHEIN, A.L.S. **SISTEMA DE INFORMAÇÃO AO USUÁRIO COMO ESTRATÉGIA DE FIDELIZAÇÃO E ATRAÇÃO.** 2003. Dissertação (MESTRE EM ENGENHARIA DE PRODUÇÃO) - UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, Porto Alegre.

SCHMITZ, M.S. Monitoramento de aplicativos do WebSocket. **Dotcom-monitor.** 13 mai. 2021. Disponível em: <https://www.dotcom-monitor.com/blog/pt-br/2021/05/13/monitoramento-de-aplicativos-do-websocket/>. Acesso em: 21 nov. 2022.

SOMMERVILLE, I.S. **Engenharia de Software.** Tradução de Ivan Bosnic e Kalinga G. de O. Gonçalves. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

USES OF NODE.JS. Uses Of Node.js. 2022? Disponível em: <https://www.educba.com/uses-of-node-dot-js/>. Acesso em: 10 nov. 2022.

STAUFFER, M. The auth scaffold in Laravel 5.2. **Matt Stauffer**. 8 jan. 2016. Disponível em: <https://mattstauffer.com/blog/the-auth-scaffold-in-laravel-5-2/>. Acesso em: 10 nov. 2022.

VALENTE, M.T.V. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade. 9. ed. Belo Horizonte: Independente, 2022.

VASCONCELOS, L.A.V. O que é um Servidor Web e como funciona?. **Hostgator**. 18 mai. 2022. Disponível em: <https://www.hostgator.com.br/blog/o-que-e-um-servidor-web-e-como-funciona/>. Acesso em: 8 nov. 2022.

VISUAL STUDIO CODE. **Getting Started.** 2022? Disponível em: <https://code.visualstudio.com/docs>. Acesso em: 8 nov. 2022.

VUEJS. O que é Vue.js?. 2022? Disponível em: <https://br.vuejs.org/v2/guide/index.html>. Acesso em: 18 nov. 2022.