RAFAELA BÁRBARA CUSTÓDIO

DESENVOLVIMENTO DE UMA PLATAFORMA DE COMUNICAÇÃO INTERNA

LAVRAS – MG

2022

RAFAELA BÁRBARA CUSTÓDIO

DESENVOLVIMENTO DE UMA PLATAFORMA DE COMUNICAÇÃO INTERNA

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Profa. Dra. Renata Teles Moreira
Orientadora

LAVRAS – MG

2022

RAFAELA BÁRBARA CUSTÓDIO

DESENVOLVIMENTO DE UMA PLATAFORMA DE COMUNICAÇÃO INTERNA

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 28 de Abril de 2022.

Profa. Dra. Renata Teles Moreira	DCC/ICET
Prof. Dr. Maurício Ronny de Almeida Souza	DCC/ICET
Prof. Dr. Neumar Costa Malheiros	DCC/ICET

Profa. Dra. Renata Teles Moreira
Orientadora

LAVRAS – MG
2022

Esta monografia é dedicada a minha família e meus eternos amigos.

AGRADECIMENTOS

Aos professores da minha graduação em especial a professora e orientadora deste trabalho Renata Teles Moreira e o professor orientador do estágio Neumar Costa Malheiros agradeço pela paciência e por me instruírem dentro do possível.

Agradeço à minha família e amigos, especialmente aos meus eternos amigos Natanaele, Kaio, Brian e a minha companheira Jaqueline que me apoiou durante todo período da graduação. Por fim, agradeço também a minha psicanalista Rafaella Alves Pacheco que me permitiu enxergar o que sou, meu potencial e o que posso me tornar.

O que sabemos é uma gota; o que ignoramos é um oceano.
(Isaac Newton)

;

RESUMO

O presente documento descreve as atividades realizadas durante o estágio supervisionado na FUNDECC (Fundação de Desenvolvimento Ciêntifico Cultural), onde a estagiária desenvolveu uma plataforma de comunicação interna. Dentre as tecnologias utilizadas para o desenvolvimento da plataforma vale citar, principalmente, Vue.js para desenvolvimento *front-end*, Node.js para desenvolvimento *back-end* e a utilização do Docker. No decorrer deste documento são apresentadas metodologias, funcionalidades do sistema e tecnologias utilizadas durante o estágio e a contribuição do curso de Bacharelado em Ciência da Computação para as atividades da estudante.

Palavras-chave: Desenvolvimento Web. *Front-end*. *Back-end*. Node.js. Vue.js. JavaScript.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O objetivo deste capítulo é apresentar de forma breve o assunto tratado neste trabalho, abordando a introdução sobre a empresa, a necessidade e objetivo do projeto, a atuação da estagiária no desenvolvimento da plataforma de comunicação e uma visão geral sobre os capítulos seguintes.

1.1 Contextualização

A Fundação de Desenvolvimento Cientifico Cultural sofria da falta de centralização de informações e notícias institucionais e um canal de comunicação oficial da instituição e por algumas questões burocráticas estava até aquele momento sem a possibilidade de adquirir alguma plataforma já existente no mercado. Surge então a necessidade de desenvolver um sistema próprio da instituição.

A solução então foi criar uma plataforma de comunicação interna, onde os colaboradores pudessem ter acesso para visualizar notícias, avisos, comunicados e poder se comunicar com os demais colaboradores.

1.2 Objetivos

O objetivo do estágio foi aplicar os conhecimentos aprendidos durante a graduação no desenvolvimento da plataforma de comunicação interna, além de preparar a estagiária para ingressar no mercado de trabalho de desenvolvimento de software. Como objetivos específicos do estágio pode-se citar: (i) Coletar e definir requisitos da plataforma de comunicação interna; (ii) Contribuir com a arquitetura do software; (iii) Realizar modelagem de banco de dados; (iv) Participar do desenvolvimento da plataforma de comunicação interna na criação do *back-end* e *front-end*.

Este documento tem como finalidade apresentar as atividades desenvolvidas pela estagiária durante o estágio supervisionado realizado na Fundação de Desenvolvimento Científico Cultural (FUNDECC) no intervalo de agosto de 2019 à março de 2020. Neste período, a estagiária atuou como desenvolvedora de uma plataforma de comunicação interna para a Fundação de desenvolvimento Cientifico Cultural.

A equipe do projeto foi formada por dois estagiários atuando com desenvolvimento *full stack*, sob supervisão do orientador do estágio.

1.3 Sobre a FUNDECC

A Fundação de Desenvolvimento Científico Cultural (FUNDECC) é uma pessoa jurídica de direito privado e sem fins lucrativos situada na cidade de Lavras, Minas Gerais. A fundação tem como objetivo apoiar o desenvolvimento de atividades de ensino, pesquisa e extensão, assim como, os desenvolvimentos institucionais, científicos e tecnológicos da Universidade Federal de Lavras (UFLA) (FUNDECC, 2022).

A FUNDECC é reconhecida como entidade cuja atuação serve de base para que as ideias desenvolvidas na UFLA possam se transformar em projetos com resultados imediatos, produtivos, levando a universidade além de sua função primordial, a produção de conhecimento e inteligência.

1.4 Estrutura do trabalho

Este trabalho está estruturado da seguinte forma: no Capítulo 2 é apresentado o referencial teórico, com objetivo de explanar as tecnologias utilizadas e aprendidas; o Capítulo 3 descreve as funcionalidades do sistema e atividades desenvolvidas pela estagiária. Por fim, as considerações finais se encontram no Capítulo 4.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentadas as principais tecnologias e bibliotecas utilizadas no desenvolvimento da plataforma de comunicação interna ocorrido durante o estágio.

2.1 Arquitetura MVC

A Arquitetura que utiliza padrão MVC (*model, view, controller*) separa a apresentação e a interação dos dados do sistema.

O sistema é estruturado em três componentes lógicos que interagem entre si. O componente *model* é responsável por gerenciar o sistema de dados e operações associadas aos dados; o componente *view* gerencia como os dados são apresentados ao usuário e por fim o componente *controller* gerência a interação do usuário passa para a *view* e para o *model*. A vantagem de utilizar o MVC é que ele permite alteração de dados de forma independente de sua apresentação e vice-versa.(GALLOTI, 2017).

2.2 API REST

API significa *Application Programming Interface* e representa um conjunto de rotinas e padrões estabelecidos e documentados para que uma determinada aplicação de software tenha autorização para utilizar as funcionalidades oferecidas por essa aplicação, sem precisar conhecer as particularidades dessa implementação (NOLETO, 2022). A API utiliza requisições HTTP responsáveis pelas operações básicas necessárias para a manipulação dos dados.

O HTTP(*HyperText Transfer Protocol*) é o principal protocolo de comunicação para sistemas Web, conforme (MOZILLA, 2021b) o protocolo HTTP define um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada para um dado recurso. Esses métodos também são comumente referenciados como *HTTP Verbs*. Cada um deles implementa uma semântica diferente e os principais utilizados nesse trabalho foram:

- GET: Requisições utilizando o GET devem retornar apenas dados.
- POST: É utilizado para submeter uma entidade a um recurso específico, frequentemente causando mudança no estado do recurso.
- PUT: Método PUT substitui todas as atuais representações do recurso de destino pela carga de dados da requisição, frequentemente utilizado para atualizar dados.

- DELETE: Remove um recurso específico.
- PATCH: Aplica modificações parciais em um recurso.

O conjunto de restrições utilizadas para que as requisições HTTP atendam as diretrizes definidas na arquitetura é o REST (*Representational State Transfer*). O conjunto de boas práticas utilizadas nas requisições HTTP realizadas por uma API em uma aplicação web é conhecido com API REST.

2.3 HTML

A maior parte das páginas de internet e aplicativos web são escritas em HTML (Linguagem de Marcação de Hipertexto). O HTML é o bloco de construção mais básico da web que permite a criação de documentos que podem ser lidos e interpretados em um dispositivo e transmitidos pela internet (MOZILLA, 2021a).

Segundo (LONGEN, 2022) um hipertexto é um texto usado para fazer referência a outros textos, enquanto uma linguagem de marcação é composta por uma série de marcações que dizem para os servidores da web qual é o estilo e a estrutura de um documento. Como exemplo de marcações temos as utilizadas para estruturar seções, parágrafos, cabeçalhos, imagens e *hyperlinks*. Para delimitar esses conteúdos utiliza-se *tags* (blocos de construção de páginas), definidos por palavras reservadas e entre os símbolos: '<' e '>'.

Um exemplo de código com a estrutura básica do html é mostrado na Figura 2.1. A utilização das *tags* <*h1*></*h1*>, <*p*></*p*>, <*h2*></*h2*>, <*img*>, <*a*></*a*> e <*button*> representam Título 1, parágrafo, Título 2, imagem, link e um botão, respectivamente.

Na Figura 2.2 é ilustrada a página Web correspondente ao arquivo HMTL mostrado na Figura 2.1.

2.4 CSS

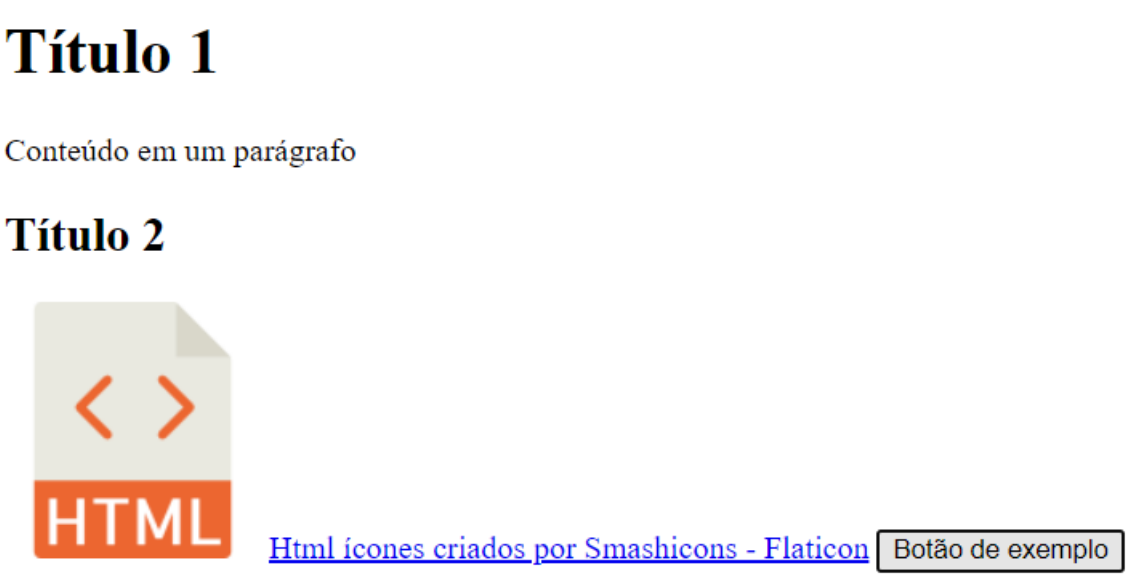
O *Cascading Style Sheet*, mais conhecido como CSS, são folhas de estilo em cascata. O CSS é uma linguagem que complementa e formata o HTML, organizando melhor as linhas e adicionando novas possibilidades ao código (VIEIRA, 2010). Com o CSS é possível modificar praticamente tudo dentro do layout (como cores, background, fontes, margens e outros).

Figura 2.1 – Exemplo código HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Página HTML - Exemplo</title>
</head>
<body>
    <h1>Título 1</h1>
    <p>Conteúdo em um parágrafo</p>
    <h2>Título 2</h2>
    <img src="img.png" alt="Está imagem é um exemplo de imagem">
    <a
        href="https://www.flaticon.com/br/icones-gratis/html"
        title="html ícones">Html ícones criados por Smashicons - Flaticon
    </a>
    <button type="submit">Botão de exemplo</button>
</body>
</html>
```

Fonte: Autora

Figura 2.2 – Exemplo de página web utilizando HTML



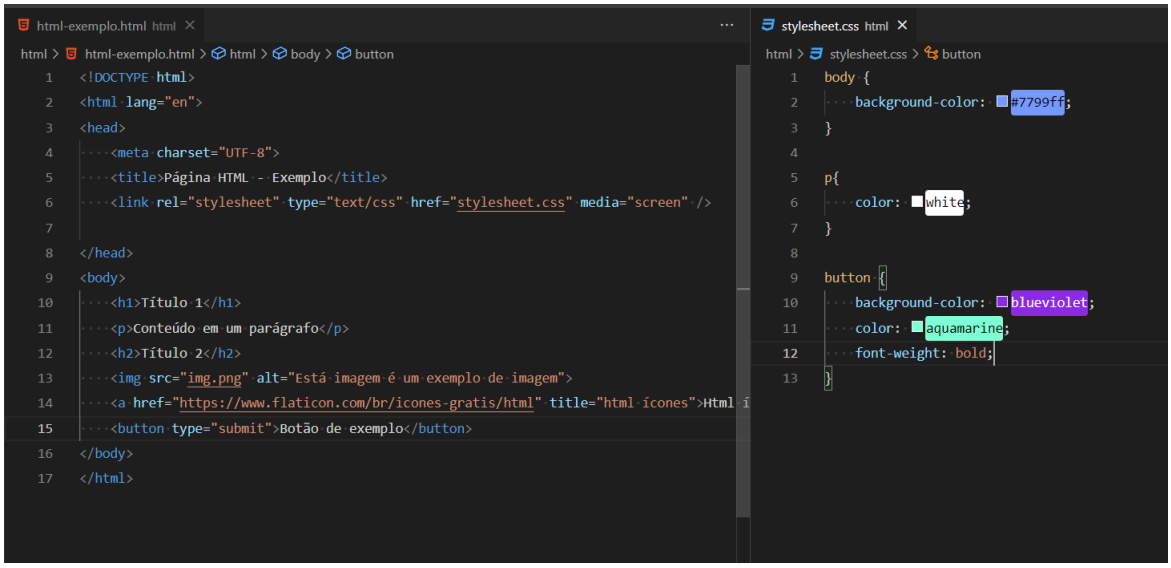
Fonte: Autora

A Figura 2.4 mostra o código CSS que configura cores de fundo e estilo de fontes, também mostra o código HTML que está sendo utilizado para estilização. O resultado da página utilizando CSS pode ser visto na Figura 2.3.

Figura 2.3 – Exemplo de página HTML estilizada

Fonte: Autora

Figura 2.4 – Exemplo de código CSS utilizado em HTML



Fonte: Autora

2.5 JavaScript

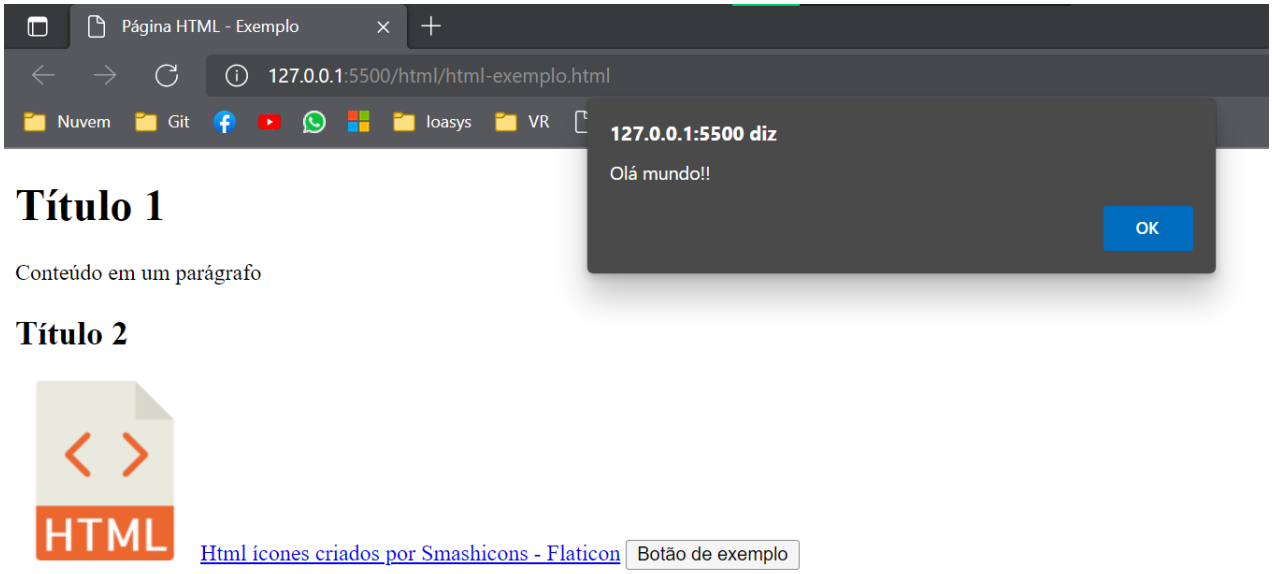
O JavaScript é uma linguagem de script orientada a objetos, multiplataforma, pequena e leve (MOZILLA, 2020). Utilizando como host um navegador web por exemplo, o JavaScript pode ser ligado aos objetos e permite o controle programático sobre eles.

Criada em 1995 a pedido da empresa Netscape, o JavaScript é uma linguagem de programação interpretada e não tipada que permite a implementação de itens complexos em páginas

web e com a crescente popularidade do JavaScript trouxe mudanças para o desenvolvimento web.

Na Figura 2.6 é mostrado o código da exibição de um alerta criado no trecho entre as linhas 7 a 11. A função *exibirAlerta* é acionada quando o usuário clica no botão "Botão de exemplo"que está declarado na linha 22. O resultado do código pode ser visto na Figura 2.5.

Figura 2.5 – Exemplo de página HTML utilizando Javascript



Fonte: Autora

Figura 2.6 – Exemplo de código Javascript utilizado em HTML

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Página HTML - Exemplo</title>
    <script>
        function exibirAlerta() {
            alert("Olá mundo!!");
        }
    </script>
</head>

<body>
    <h1>Título 1</h1>
    <p>Conteúdo em um parágrafo</p>
    <h2>Título 2</h2>
    <img src="img.png" alt="Está imagem é um exemplo de imagem">
    <a href="https://www.flaticon.com/br/icones-gratis/html"
        title="html ícones">Html ícones criados por Smashicons -
        Flaticon</a>
    <button onclick="exibirAlerta()" type="button">Botão de exemplo</button>
</body>

</html>
```

Fonte: Autora

2.6 Node.js

O Node.js é baseado no interpretador V8 do Google e permite executar código JavaScript fora do navegador. Com essa tecnologia é possível construir APIs e microsserviços (DIAS, 2021).

Para desenvolvimento do projeto foi utilizado o *Express*[1] que é o *framework* Node mais popular(MOZILLA, 2021c). A escolha desse *framework* foi pelo fato de ser bastante minimalista e que permite bastante liberdade para o desenvolvimento.

[1] https://expressjs.com/pt-br/

2.7 Vue.js

Segundo (PICOLLO, 2020) o Vue.js é um *framework* Javacript *open source* muito utilizado para criar aplicações *single page* e também para desenvolver vários tipos de interfaces que tem a necessidade de maior interação e experiencia de mais valor para o usuário.

O Vue.js permite reutilizar componentes facilmente em toda a aplicação. Não é necessário seguir nenhuma sintaxe especial, seus modelos são objetos Javascript simples, que podem ser vinculados a tudo o que você quer em suas *views*.

2.8 Git

O Git[2] é o sistema de controle de versão mais utilizado atualmente (VALENTE, 2022). É um projeto de código aberto e foi desenvolvido por Linus Torvalds (também criador do kernel do sistema operacional Linux). Segundo (ATLASSIAN, 2022) hoje o Git é a melhor escolha para a maioria das equipes de desenvolvimento de software.

Dentre os motivos que tornam o Git a melhor escolha pode-se citar: (i) Funcionalidades, desempenho e segurança que a maioria dos desenvolvedores precisam; (ii) É a ferramenta mais adotada da categoria se tornando de fato um padrão; (iii) É um projeto de código aberto com bastante suporte. O desenvolvimento da plataforma de comunicação interna utilizou como plataforma de hospedagem de código-fonte o GitLab[3].

2.9 Vuex

O Vuex[4] é um padrão de gerenciamento de estado e biblioteca para aplicações Vue.js. Sua função é servir como um *store* centralizado para todos os componentes de uma aplicação. (VUEX, 2022).

2.10 Socket.IO

Socket.IO é uma biblioteca que permite a comunicação bidirecional e baseada a eventos entre o browser (navegador web) e o servidor web.

[2] https://git-scm.com/
[3] https://about.gitlab.com/
[4] https://vuex.vuejs.org/ptbr/

Segundo (GABRIEL, 2020) o Socket.io é um framework que facilita a implementação de Socket e está disponível para diversas plataformas e linguagens de programação, tanto *front-end* quanto *back-end*. O Socket.IO deve ser utilizado quando é necessário manter uma conexão de sockets aberta entre o navegador e o servidor.

2.11 Docker

O Docker[5] é uma plataforma de software que facilita o processo de construção, execução, gerenciamento e distribuição de aplicativos. Ele faz isso virtualizando o sistema operacional do computador no qual está instalado e sendo executado; isso permite que sejam criados ambientes independentes e isolados para iniciar e implantar seus aplicativos. Esses ambientes são chamados de contêineres (MOLL, 2022).

O Docker compose é o orquestrador de contêineres da Docker. Com ele é possível configurar todos os parâmetros necessários para executar cada contêiner a partir de um arquivo de definição. Neste projeto o Docker compose foi utilizado para montar o ambiente *back-end* criando um container para a API e um para o banco de dados. A utilização do Docker compose é interessante principalmente com o aumento de número de contêineres em execução e torna-se necessário um melhor gerenciamento entre as comunicações.

Na Figura 2.7 retirada de (MOLL, 2022) é ilustrado um exemplo de arquivo Dockerfile que contém todas as dependências que o aplicativo Python precisa, incluindo o próprio Python. O arquivo Dockerfile em questão diz ao Docker para: (i) Criar uma imagem começando com a imagem do Python 3.7; (ii) Definir o diretório de trabalho para /code; (iii) Definir as variáveis de ambiente usadas pelo comando flask; (iv) Instalar o gcc e outras dependências; (v) Copiar o requirements.txt e instalar as dependências do Python; (vi) Adicionar metadados à imagem para descrever que o contêiner está escutando na porta 5000; (vii) Copiar o diretório atual no projeto para o workdir na imagem; (viii) Definir o comando padrão para o contêiner para flask run; (MOLL, 2022).

Na Figura 2.8 é mostrado um um exemplo de arquivo *docker-compose.yaml*. Esse arquivo *Compose* encontrado como exemplo em (MOLL, 2022) define dois serviços web e redis

.

[5] https://www.docker.com/

Figura 2.7 – Exemplo de arquivo Dockerfile

```
# syntax=docker/dockerfile:1
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Fonte: (MOLL, 2022)

Figura 2.8 – Exemplo de arquivo docker-compose

```
version: "3.9"
services:
web:
build: .
ports:
- "5000:5000"
redis:
image: "redis:alpine"
```

Fonte: (MOLL, 2022)

2.12 MongoDB

O MongoDB é categorizado como um banco de dados NoSQL (*not only SQL*), pois o armazenamento e a recuperação de dados no MongoDB não são feitas no formato de tabelas. As informações são armazenadas, mas o processo é mais fluido, independente, com os elementos tendo identificações únicas(KOVACS, 2021).

O MongoDB é um banco de dados de documentos com escalabilidade e flexibilidade, junto com a consulta e indexação que um desenvolvedor precisa. É possível utilizá-lo localmente ou em nuvem e foi projetado para armazenar uma grande escala de dados (MONGODB, 2022).

3 ATIVIDADES DESENVOLVIDAS

Neste capítulo são descritas as atividades realizadas durante o estágio desde o levantamento de requisitos até o desenvolvimento das funcionalidades da plataforma de comunicação interna. As atividades estão divididas em *back-end*, *front-end* e entre os módulos do sistema.

3.1 Estudo das tecnologias

Ao iniciar o estágio foi necessário um tempo para estudo das tecnologias definidas para utilização no projeto, sendo elas, principalmente, API Rest, Docker, Node.js e Vue.js e outras citadas no Capítulo 2. A prática e estudos das tecnologia ocorreu em dupla com outro estagiário e individualmente guiadas pelo orientador. O período de estudo das tecnologias foi durante o primeiro mês do estágio.

3.2 Levantamento de Requisitos

O principal problema da FUNDECC era a falta de centralização de avisos, noticias e também de comunicação entre funcionários e setores da empresa. A partir disso foram coletadas informações e realizada a análise de requisitos juntamente com o orientador do estágio. Os requisitos mapeados para a plataforma de comunicação interna estão listados a seguir.

1. O sistema deve suportar 2 tipos de comunicação: a comunicação direta entre dois usuários e a comunicação em grupo.

2. A comunicação pode envolver mensagens de texto e também o envio de mídias, como fotos ou documentos.

3. O sistema deve permitir a criação de um mural de avisos. O mural é caracterizado por um tema/assunto ou setor e somente o coordenador do mural pode postar avisos.

4. O sistema tem que incluir um calendário para eventos e reservas de espaços físicos (como auditório, sala de reunião, etc.). O sistema deve notificar os usuários sobre os eventos.

5. Os colaboradores e administradores devem fazer login para ter acesso a sua página. Nesta página conterá lembretes e avisos, além do menu para poder cadastrar seus próprios avisos, marcar reuniões, e iniciar uma conversa com outra pessoa.

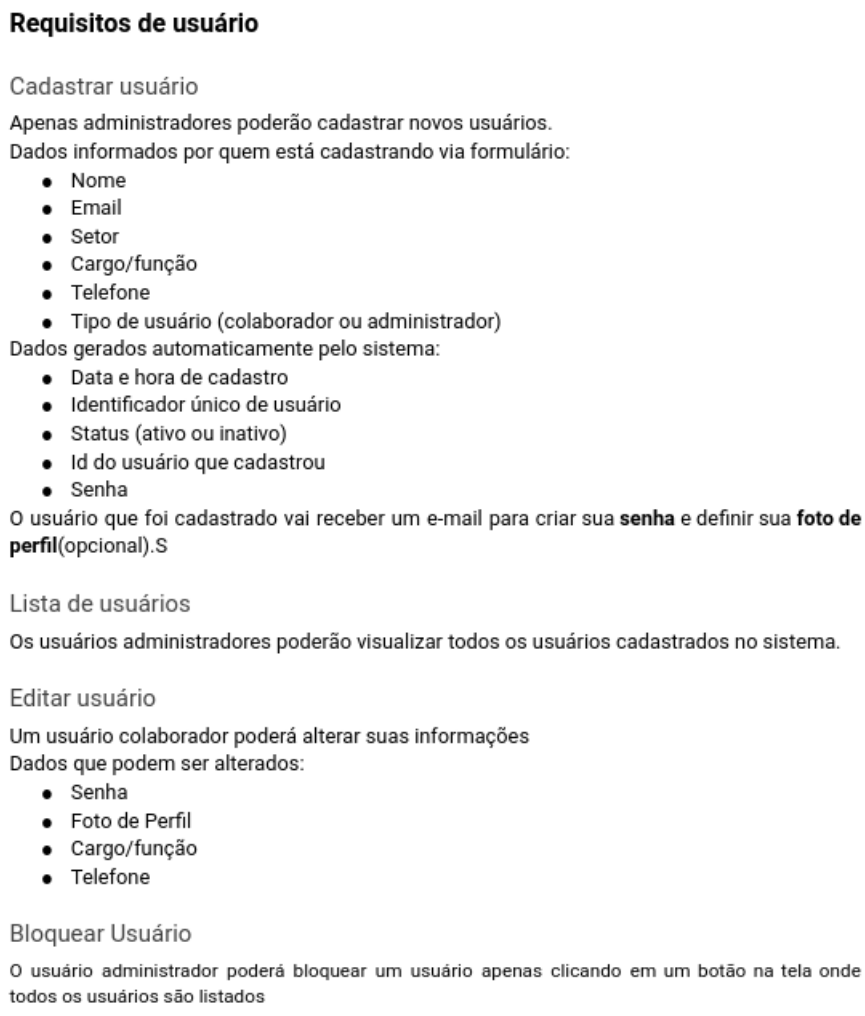
6. O administrador pode cadastrar novos usuários e novos setores.

7. O sistema deve salvar todas as conversas e os horários que os usuários conectam e desconectam do sistema (log).

8. Os administradores terão uma lista com todos os usuários contendo nome, setor, cargo/-função, telefone e e-mail.

A partir dos requisitos levantados, foi possível especificar cada um deles. Na Figura 3.1 é apresentado um exemplo de requisito de usuário detalhado.

Figura 3.1 – Exemplo de especificação de requisitos de usuário



Requisitos de usuário

Cadastrar usuário

Apenas administradores poderão cadastrar novos usuários.
Dados informados por quem está cadastrando via formulário:

- Nome
- Email
- Setor
- Cargo/função
- Telefone
- Tipo de usuário (colaborador ou administrador)

Dados gerados automaticamente pelo sistema:

- Data e hora de cadastro
- Identificador único de usuário
- Status (ativo ou inativo)
- Id do usuário que cadastrou
- Senha

O usuário que foi cadastrado vai receber um e-mail para criar sua **senha** e definir sua **foto de perfil**(opcional).S

Lista de usuários

Os usuários administradores poderão visualizar todos os usuários cadastrados no sistema.

Editar usuário

Um usuário colaborador poderá alterar suas informações
Dados que podem ser alterados:

- Senha
- Foto de Perfil
- Cargo/função
- Telefone

Bloquear Usuário

O usuário administrador poderá bloquear um usuário apenas clicando em um botão na tela onde todos os usuários são listados

Fonte: Autora

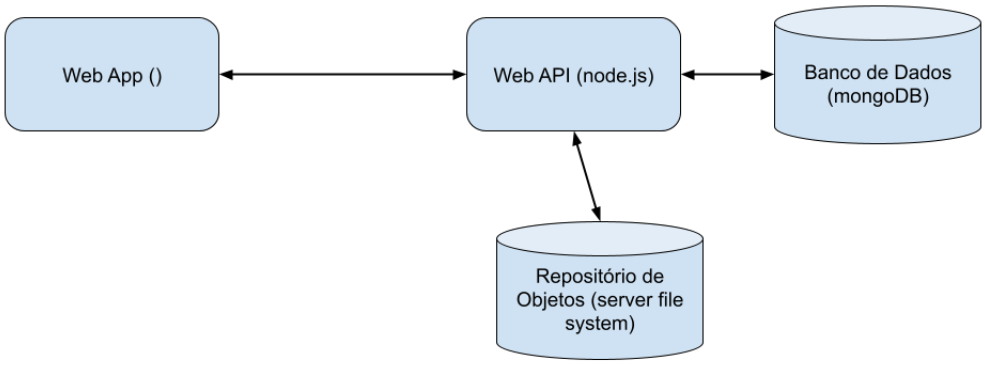
3.3 Arquitetura do Sistema

A solução de arquitetura adotada para o desenvolvimento da plataforma de comunicação interna foi a de Cliente/Servidor. Como ilustrado na Figura 3.2, o sistema foi pensado para possuir *back-end*, *front-end*, banco de dados e um *server file system*.

Um *server file system* é um servidor de arquivos e pode ser qualquer servidor de uma rede de computadores que tenha como função fornecer um sistema de arquivos. A principal função dos servidores de arquivos é oferecer aos usuários um local de armazenamento centralizado para que pastas, arquivos e objetos estejam disponíveis a todos os clientes autorizados. A utilização do *server file system* não foi implementada no contexto do estágio e, portanto, não será abordado neste documento.

O *back-end* desenvolvido foi construído em Node.js[1]; o *front-end* foi desenvolvido utilizando Vue.js[2] e o banco de dados utilizado foi o MongoDB[3].

Figura 3.2 – Modelo de Arquitetura



Fonte: Autora

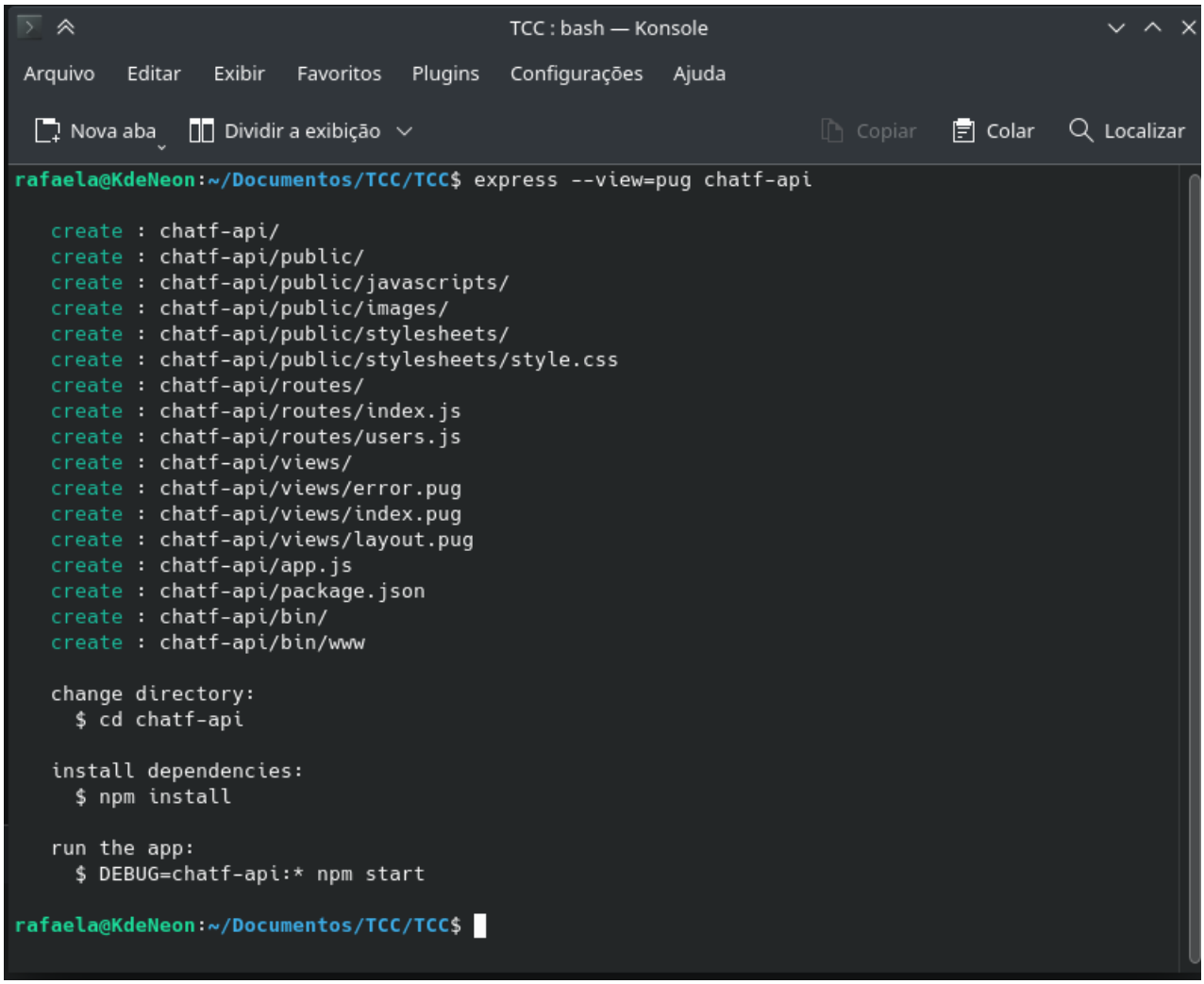
[1] https://nodejs.org/en/
[2] https://vuejs.org/
[3] https://www.mongodb.com/pt-br

3.4 Criação do *back-end* e sua estrutura inicial

O desenvolvimento da plataforma teve inicio no *back-end*. Como era um novo sistema, uma das atividades foi as configurações iniciais do projeto.

Para o desenvolvimento do *back-end* foi utilizado o *framework Express*. Para dar *start* no projeto *Express*, foi utilizado o ***express-generator*** executando o comando *express –view=pug chatf-api*; o *output* deste comando e após sua execução pode ser visto na Figura 3.3.

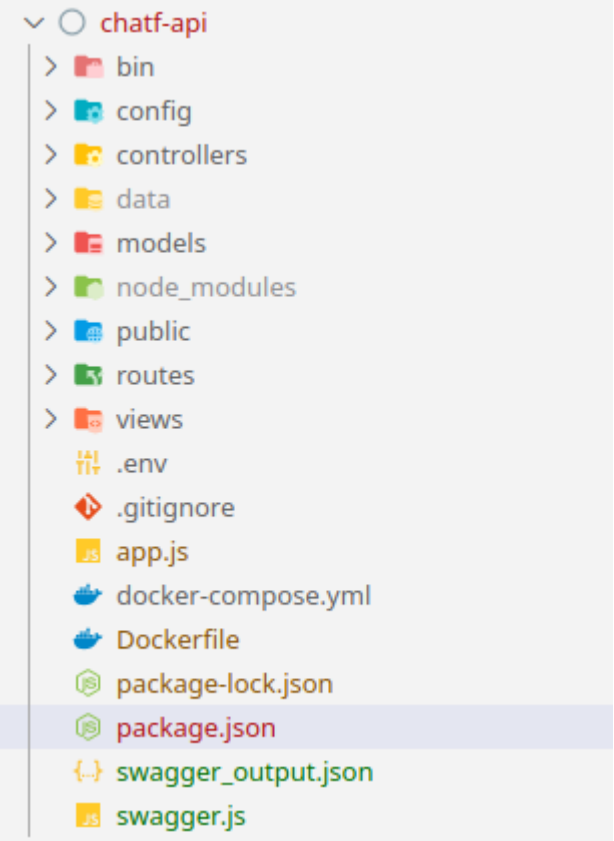
Figura 3.3 – Comando *express-generator*



Fonte: Autora

Com a estrutura do projeto criada (Figura 3.4), foi adicionado algumas bibliotecas nas dependências do projeto e suas configurações. Essas bibliotecas tem por finalidade facilitar o desenvolvimento de algumas funcionalidades como encriptar senhas, acesso a banco de dados, comunicação do chat e documentação.

Figura 3.4 – Estrutura de diretórios e arquivos do projeto



Fonte: Autora

Na Figura 3.4 é ilustrado como os diretórios *controllers*, *config* e *models* e os arquivos *swagger.js*, *.env*, *Dockerfile*, *docker-compose.yml* foram criados na estrutura inicial do *back-end*. O diretório *data* e o arquivo *swagger-output.json* foram gerados ao executar o *back-end*.

Os arquivos contidos no pacote *controllers* têm a função de receber e atender requisições feitas a partir do *client*; os arquivos desse diretório também contém as regras de negócio e consulta ao banco de dados. Os arquivos que possuem a função de modelar os objetos do sistema e fazer o mapeamento objeto relacional com a ajuda do *mongoose* estão no diretório *models*. No diretório *config* estão os arquivos de configuração de conexão ao banco de dados, conexão com *Socket.IO*, criação de *tokens* e algumas constantes.

O arquivo *swagger.js* possui configurações para geração da documentação com *Swagger*; no arquivo *.env* contém as variáveis de ambiente e os arquivos *Dockerfile* e *docker-compose.yml* possuem as configurações necessárias para rodar a aplicação como um container *docker* e executá-lo juntamente com o container do banco de dados *MongoDB*. Por fim, o diretório *data* e seus arquivos são gerados pelo *mongoose* ao executar o projeto e o arquivo *swagger-output.json* é a documentação do *Swagger* que também é gerada ao rodar o *back-end*.

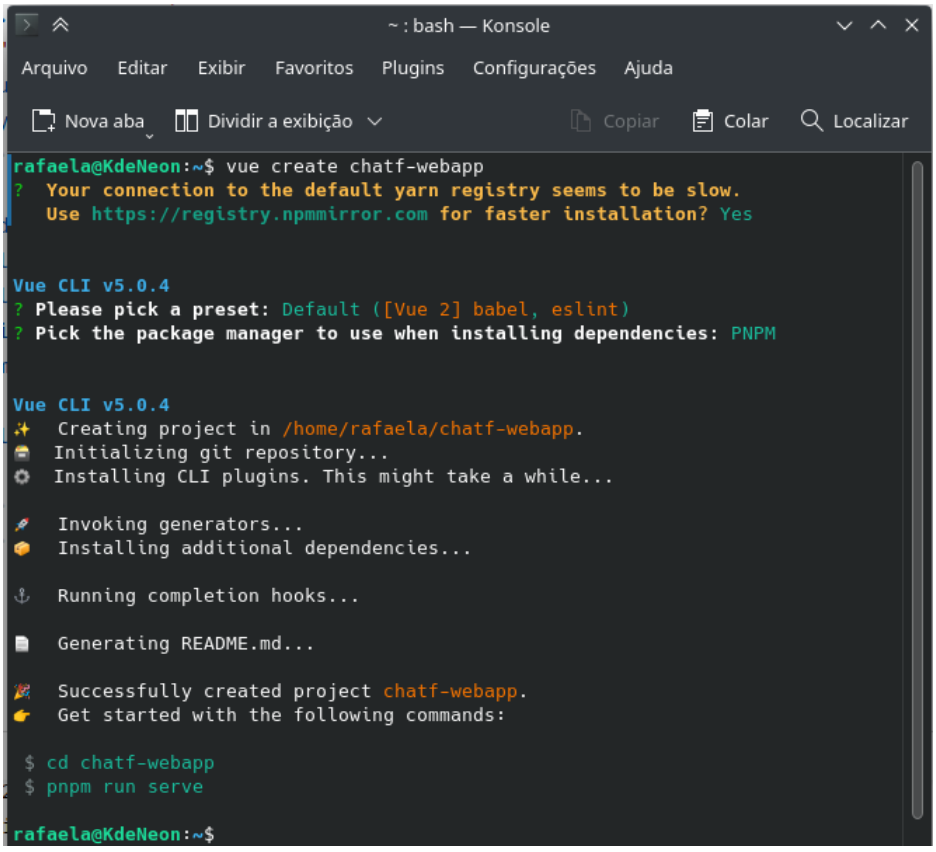
Sobre os arquivos e diretórios gerados pela ferramenta *express-generator*, temos o diretório *routes* que é onde estão as rotas mapeadas de acordo com seus devidos métodos e *controllers*. Para as configurações gerais do projeto e do *framework* (definição de portas, utilização de rotas, importação e configurações de bibliotecas e outros) tem-se o diretório *bin* e arquivo *app.js*.

3.5 Criação do *front-end* e sua estrutura inicial

O *front-end* do projeto foi desenvolvido em Vue.js. Para dar inicio no projeto e gerar os arquivos iniciais, foi utilizado o ***Vue CLI***[4] executando o comando *vue create chatf-webapp*. Este comando e seu *output* após a execução pode ser visto na Figura 3.5.

Assim como no projeto *back-end*, após gerar os arquivos e diretórios da estrutura do projeto, também foi adicionado as bibliotecas utilizadas nas dependências do projeto e suas configurações (ver Figura 3.6).

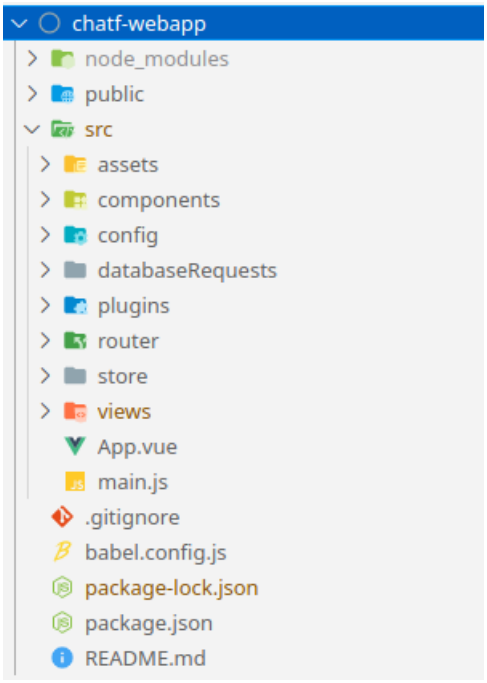
Figura 3.5 – Vue CLI



Fonte: Autora

[4] https://cli.vuejs.org/

Figura 3.6 – Estrutura do projeto front-end



Fonte: Autora

Como mostrado na Figura 3.6, no arquivo main.js é onde estão as configurações e importações das bibliotecas utilizadas. O diretório *views* contém todos os arquivos que constroem uma página, como por exemplo a página de Login e o diretório *assets* possui imagens utilizadas no sistema.

No diretório *config* está o arquivo onde são criadas as principais constantes utilizada em todo o projeto. O arquivo contido em *router* é responsável pela criação das rotas e suas respectivas *views*. Em *store* é onde o **Vuex** está sendo utilizado e no diretório *plugins* encontra-se a declaração do **Vuetify** que é a *UI Library* que foi utilizada no desenvolvimento.

O diretório *components* possui os componentes que são utilizados em *views*, como por exemplo modais, *cards* e o *footer*. Por fim, para fazer as requisições no *back-end* foi criado um arquivo para cada módulo do sistema. Esses arquivos se encontram na pasta *databaseRequests*.

3.6 Implementação do sistema

Esta seção contém as funcionalidades implementadas na plataforma de comunicação interna. O desenvolvimento dos componentes e *views* citados a seguir foram desenvolvidos utilizando o projeto em Vue.js, mostrado na Seção 3.5 e consomem a API, mostrada na Seção 3.4.

3.6.1 Cadastro de usuários e Login

Para utilizar a plataforma de comunicação é necessário que o usuário se cadastre no sistema utilizando um formulário que pode ser visto na Figura 3.7. Neste formulário o usuário preenche os campos solicitados, sendo um *combo-box* os campos **Cargo** e **Setor** que são populados com dados vindos do banco de dados.

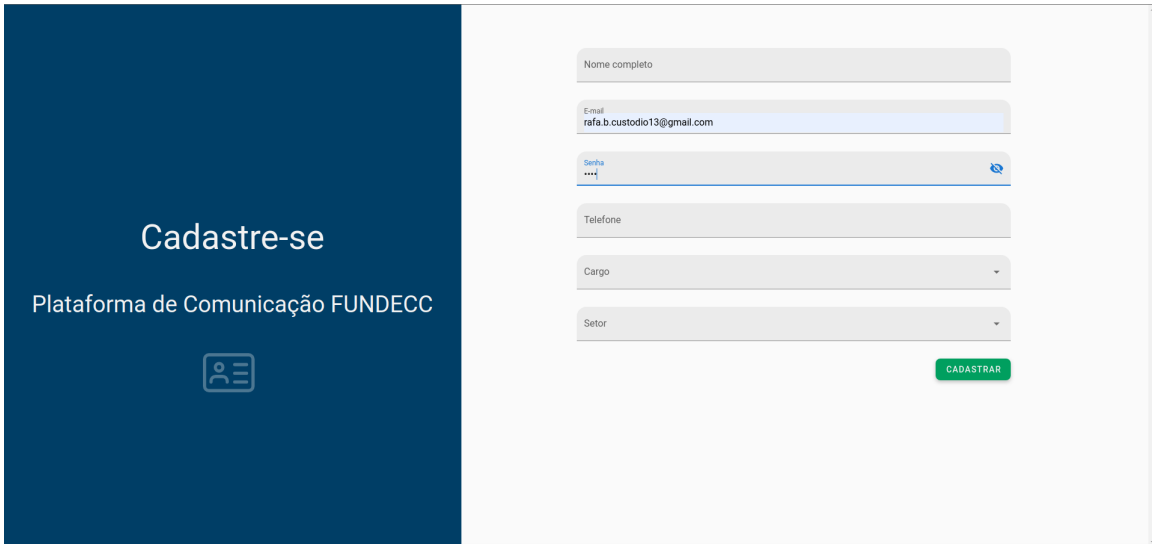
Caso o usuário preencha o formulário de forma correta, seu cadastro será realizado, porém seu *login* ainda estará inativo até que um administrador ative seu usuário. Esse gerenciamento de usuário será visto na seção 3.6.5.1.

Para efetuar *Login*, é necessário que o usuário preencha o formulário (Figura 3.8) com seus dados de **E-mail** e **Senha**, e clique em **Entrar**. Se os dados estiverem corretos, o usuário entrará no sistema.

Os dados que preenchem os campos **Cargo** e **Setor** na tela de cadastro são buscados por meio de uma requisição HTTP do tipo GET. O retorno dessa requisição preenche um *array* da linguagem JavaScript e os dados desse *array* são impressos no *combo-box*. O cadastro de usuário e o *login* são efetuados também por meio de uma requisição HTTP, mas nesse caso a requisição do cadastro é do tipo POST e a requisição do *login* é do tipo PATCH. Em ambos formulários também são feitas validações nos campos obrigatórios e também possuem mensagens de *feedback*.

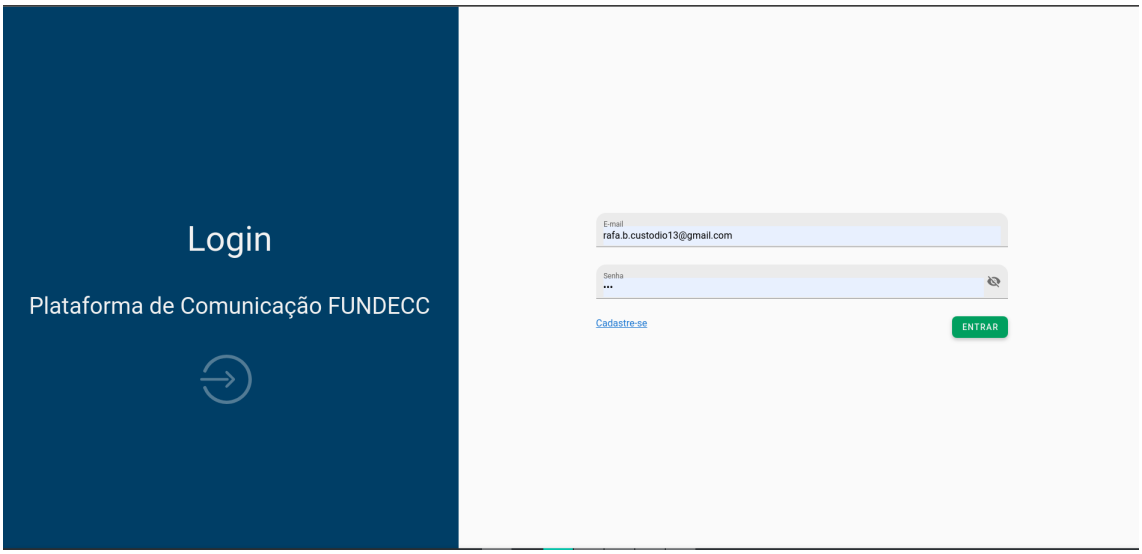
Após ter um login e senha validados pelo servidor, será criado um token que o usuário receberá em resposta e que permitirá o acesso aos recursos do sistema. O padrão adotado é o formato JWT (*JSON Web Token*) e ele fará com que o token seja assinado da forma correta para haver a autenticação da requisição a um recurso no *back-end*.

Figura 3.7 – Tela de cadastro de usuários



Fonte: Autora

Figura 3.8 – Tela de *login*



Fonte: Autora

3.6.2 *Chats* e Grupos

No menu chat, o usuário logado pode iniciar uma conversa com outros usuários do sistema (Figura 3.9). Para enviar mensagem é necessário digitá-la no campo de texto localizado na parte inferior da tela. As conversas e mensagens ficam salvas no banco de dados e são listadas no menu lateral (ver Figura 3.10).

A página de Chat possui um botão para criar um novo grupo. Este botão abre uma modal onde os dados são preenchidos e os usuários que irão fazer parte daquele grupo são escolhidos. Existem dois *checkboxs* que questionam o usuário se aquele grupo será também um mural e/ou

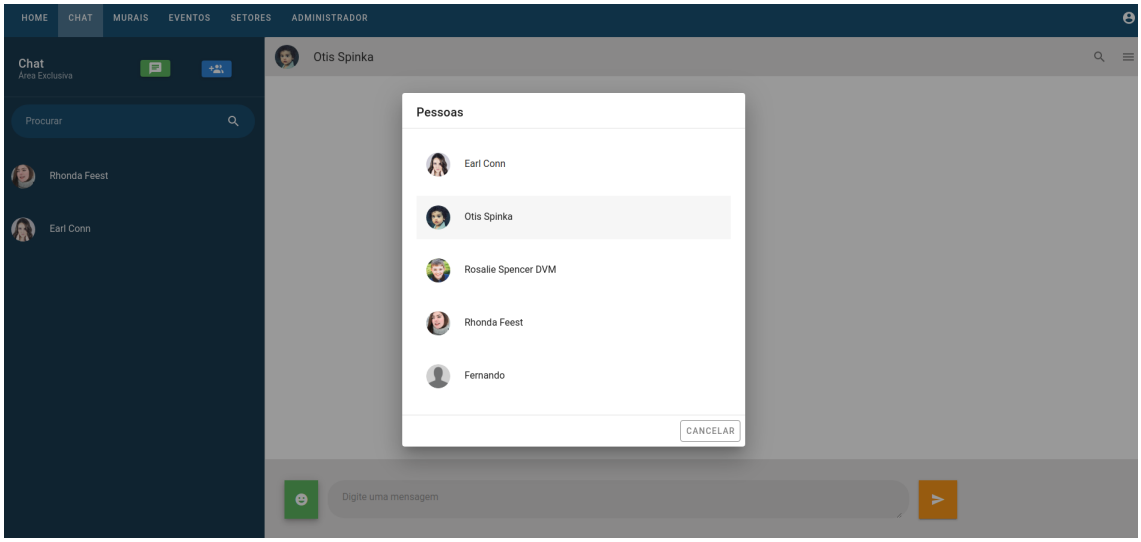
um setor (será detalhado nas sub sessões 3.6.3 e 3.6.4 respectivamente), caso ambos não sejam selecionados, o grupo será apenas um grupo de conversas. A modal de adicionar grupos pode ser vista na Figura 3.11).

Os dados que preenchem a lista de conversas, listas de usuários e mensagens em cada conversa são buscados por meio de uma requisição HTTP do tipo GET. O envio de mensagens e criação de um novo grupo utiliza o método HTTP do tipo POST.

A *modal* utilizada para criar um novo grupo possui componentes de *upload* de arquivos (imagens) e a seleção de usuários utiliza um componente que permite seleção múltipla para que seja possível adicionar vários usuários ao mesmo tempo.

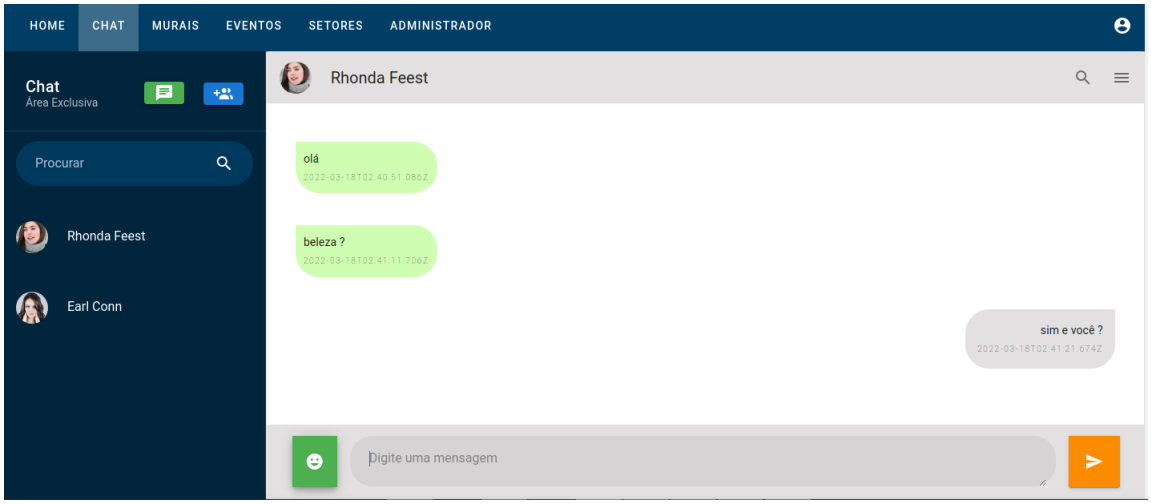
Para que a conversa entre dois usuários seja dinâmica, o Socket.io é utilizado para envio e recebimento de mensagens instantâneas tanto do lado do *back-end* quanto do lado do *front-end*.

Figura 3.9 – Modal iniciar nova conversa



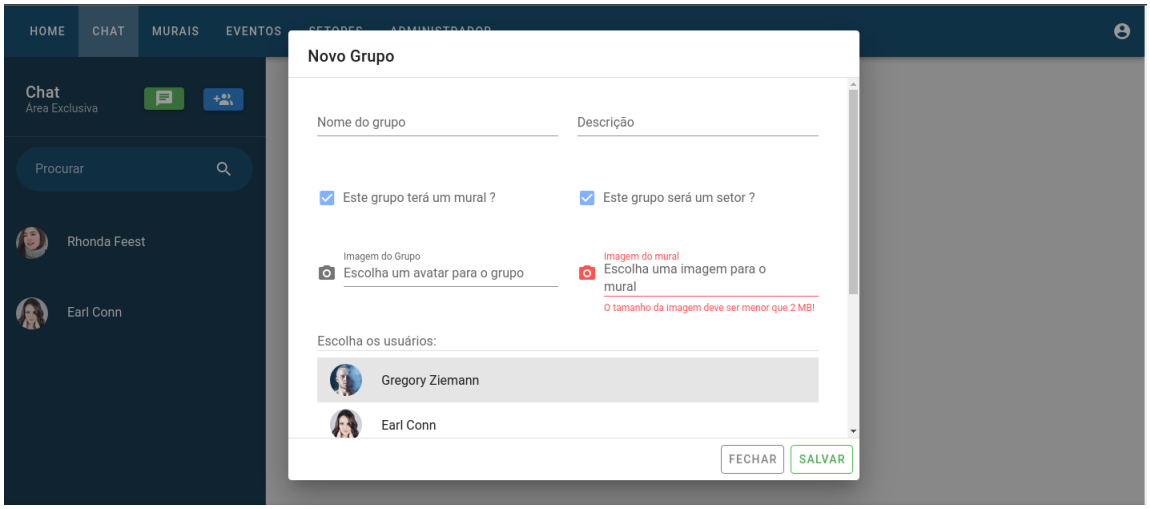
Fonte: Autora

Figura 3.10 – Tela de *chat*



Fonte: Autora

Figura 3.11 – Modal criar novo grupo



Fonte: Autora

3.6.3 Murais e Avisos

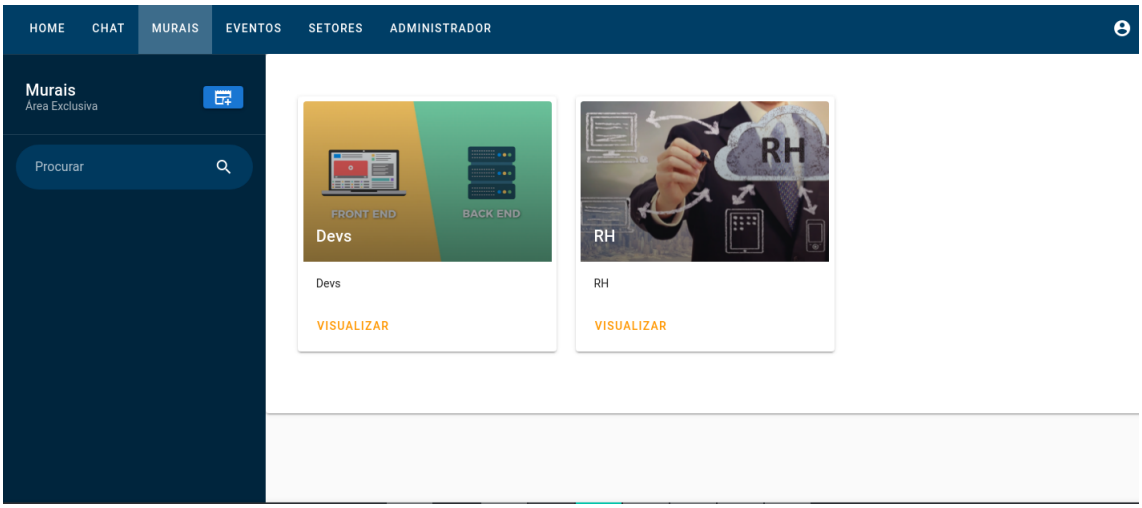
Nos murais é onde se encontram os avisos que podem ser enviados para determinado grupo de pessoas. Por esse motivo, murais são grupos que foram sinalizados em sua criação que atuariam também como murais. Eles são listados em *cards*, como pode ser visto na Figura 3.12.

Para adicionar um aviso é necessário clicar no botão que se encontra no menu lateral e uma modal (Figura 3.13) será aberta para que os campos sejam preenchidos e que seja informado em quais murais aquele aviso estará presente. Como os avisos estão associados a um mural ao clicar em visualizar um mural específico, será listado os avisos do mural em questão (ver Figura 3.14).

Os dados que preenchem a lista de *cards* de murais e também a listas de avisos são buscados por meio de uma requisição HTTP do tipo GET. A criação de um novo aviso utiliza o método HTTP do tipo POST.

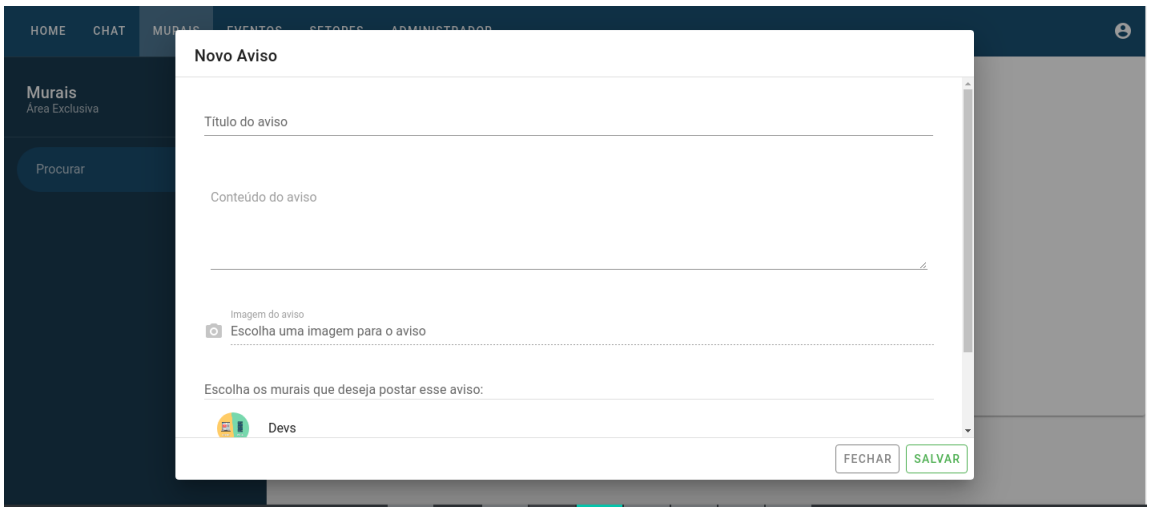
A *modal* utilizada para criar um novo aviso possui componentes de *upload* de arquivos (imagens) e a seleção de murais utiliza um componente que permite seleção múltipla para que seja possível adicionar avisos em vários murais ao mesmo tempo.

Figura 3.12 – Lista de murais



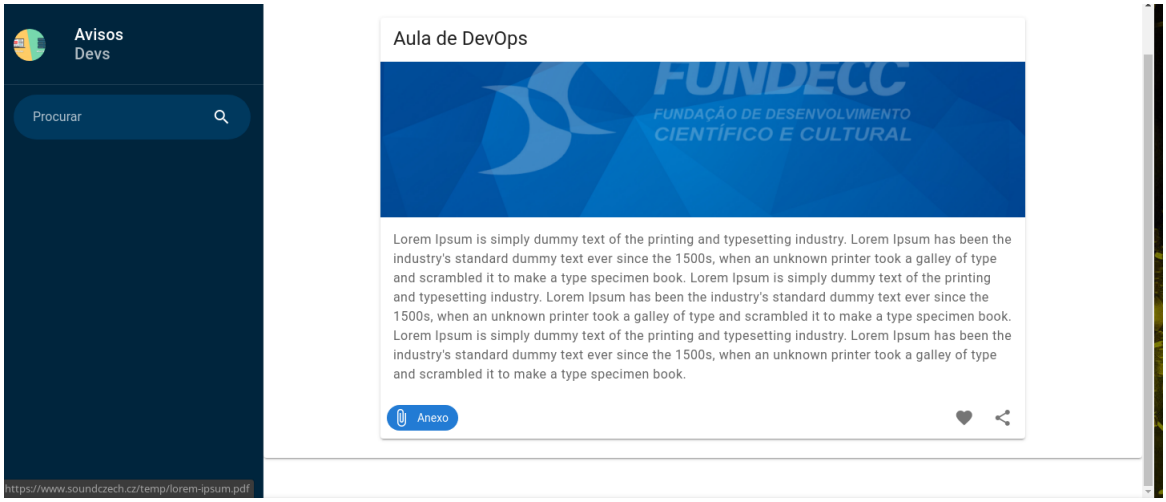
Fonte: Autora

Figura 3.13 – Adicionar aviso



Fonte: Autora

Figura 3.14 – Visualizar avisos



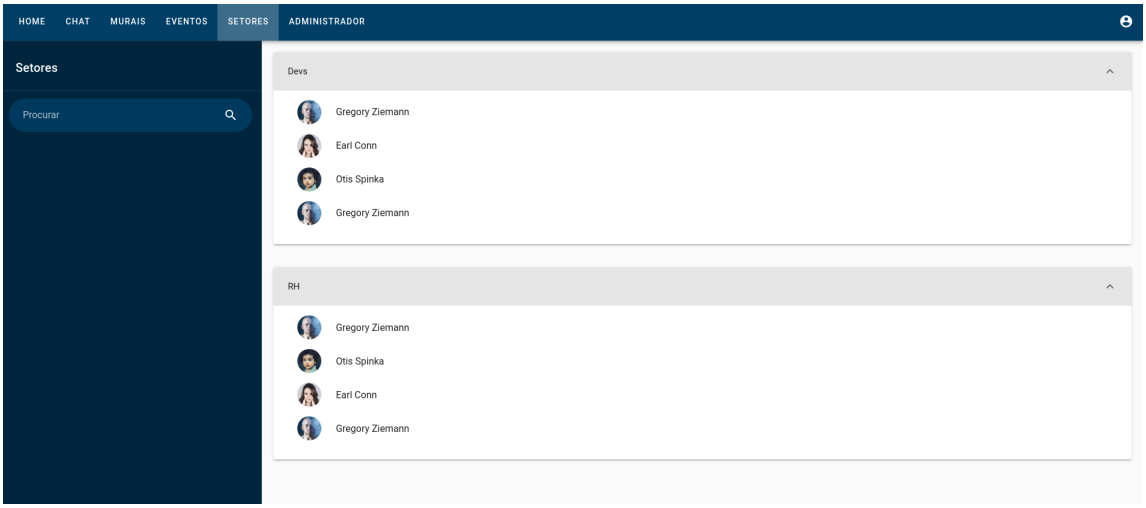
Fonte: Autora

3.6.4 Setores

Os setores existem para que usuários da plataforma consigam saber de qual setor é cada colaborador da FUNDECC e, assim como os murais, os setores são grupos que em sua criação foram sinalizados que atuariam também como um setor. Na Figura 3.15 é possível ver a tela de listagem de setores e os integrantes de cada setor.

Os dados que preenchem a lista de setores e seus integrantes são buscados por meio de uma requisição HTTP do tipo GET. Os setores são exibidos em uma estrutura chamada *colapse* que exibe e oculta os integrantes do setor de acordo com a vontade do usuário.

Figura 3.15 – Lista de setores



Fonte: Autora

3.6.5 Área do Administrador

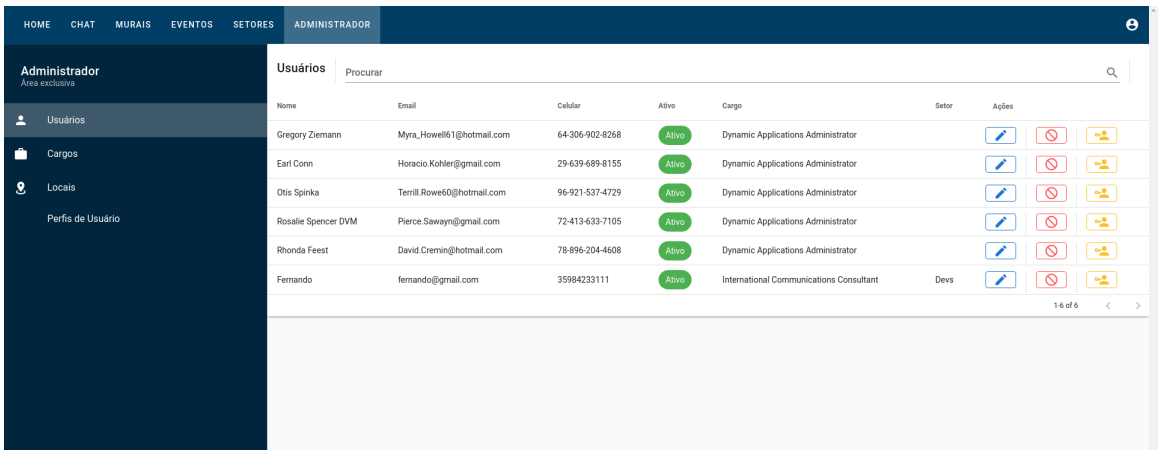
A área do administrador tem o objetivo de permitir que usuários do tipo administrador gerenciar usuários, podendo adicionar, editar e desabilitar cargos e perfis. Além disso, o administrador pode adicionar locais da FUNDECC como salas de reuniões ou de eventos, que seriam utilizados como informação na criação de um Evento. A funcionalidade e tela de Evento não foi abordada nesse trabalho.

3.6.5.1 Gerenciamento de usuários

Como mostra a Figura 3.16, o usuário administrador tem acesso a listagem de todos os usuários do sistema, onde ele pode editar, bloquear e/ou alterar o perfil atribuído ao usuário (ver Figura 3.17).

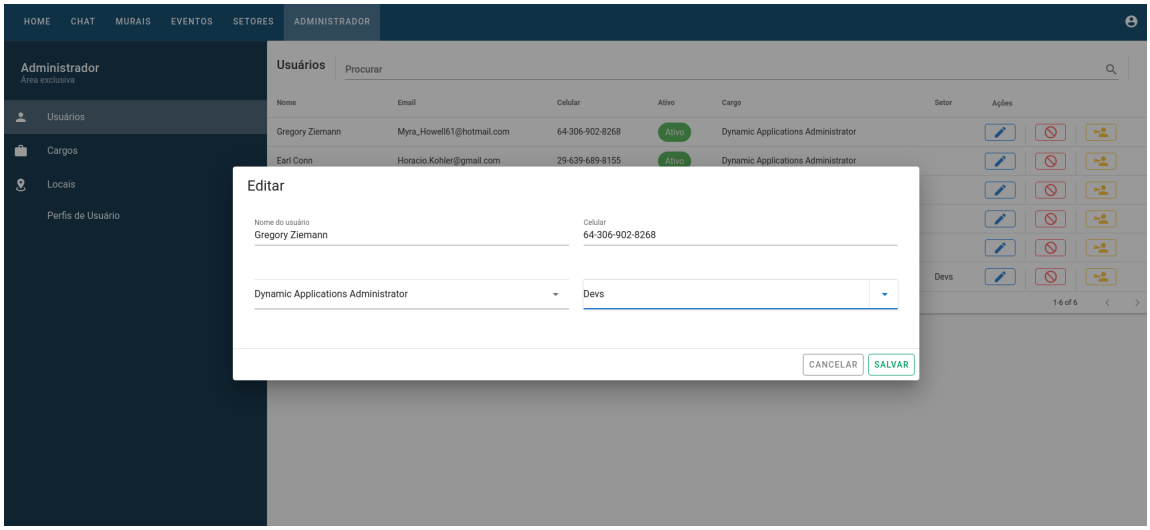
Os usuários listados na tabela e os setores listados na *combo-box* da *modal* de edição são buscados por meio de uma requisição HTTP do tipo GET. A edição do usuário é efetuada utilizando um requisição HTTP do tipo PUT, e a funcionalidade que permite desativar ou ativar um usuário utiliza o PATCH.

Figura 3.16 – Tabela de listagem de usuários do sistema



Fonte: Autora

Figura 3.17 – Modal de edição de usuários



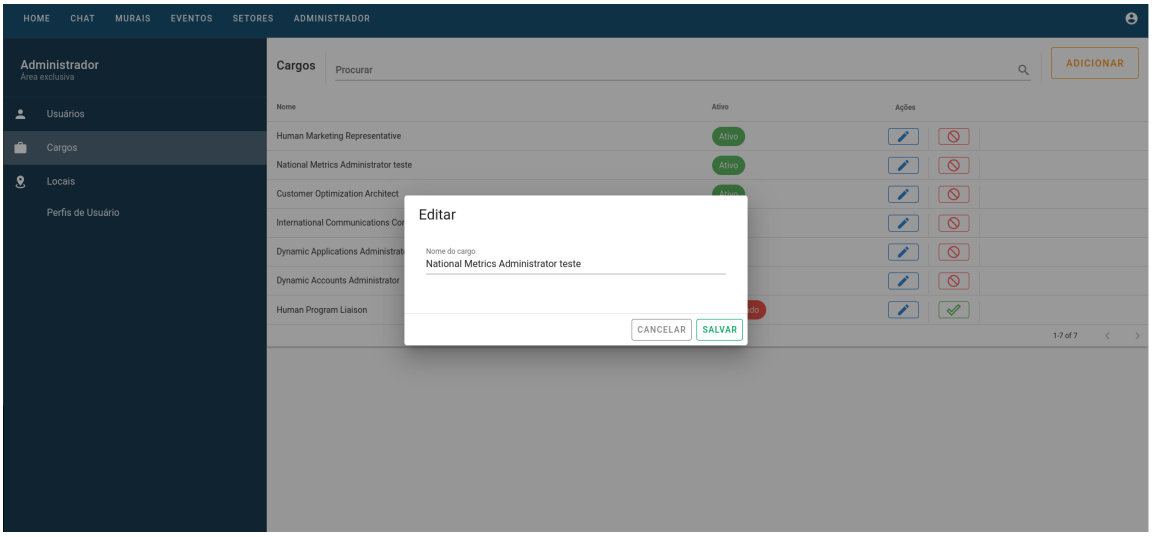
Fonte: Autora

3.6.5.2 Gerenciamento de Cargos

Assim como na listagem de usuário, a tabela trás uma listagem de todos os cargos cadastrados no sistema que podem estar ativos ou não (ver Figura 3.19). O usuário administrador consegue adicionar um novo cargo ou editar o nome de um cargo existente (Figura 3.18). Caso um cargo não exista mais, o administrador pode desativá-lo.

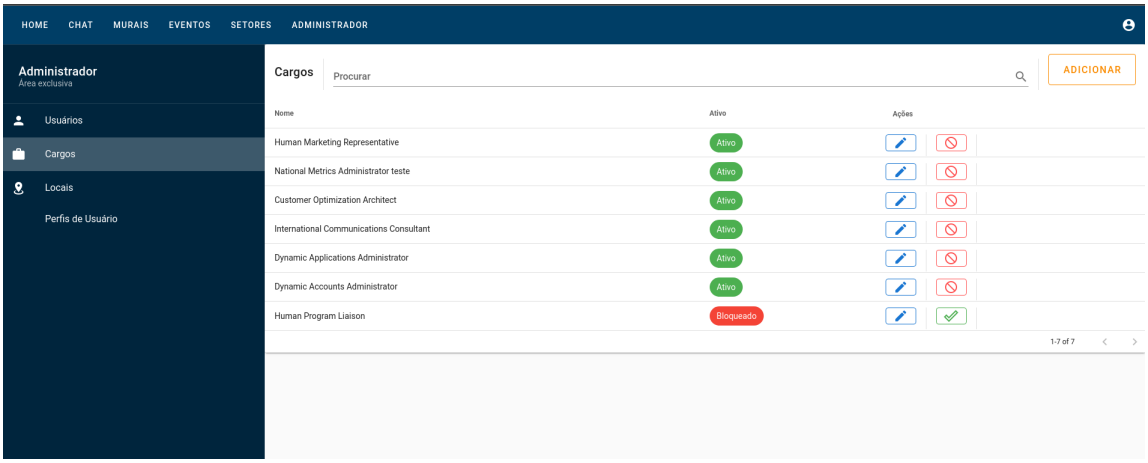
A atualização do status do cargo utiliza o método HTTP do tipo PATCH, para adicionar um novo cargo utiliza-se o método HTTP do tipo POST, e para editar utiliza-se o método HTTP do tipo PUT.

Figura 3.18 – Modal de edição de cargo



Fonte: Autora

Figura 3.19 – Tabela de listagem de cargos



Fonte: Autora

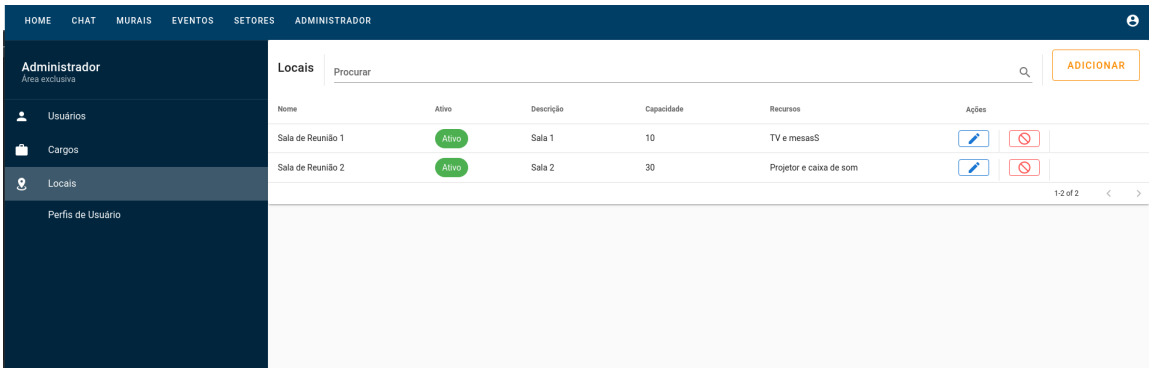
3.6.5.3 Gerenciamento de Locais

Os locais foram idealizados para serem locais físicos da FUNDECC e serem usados como informações sobre uma sala onde ocorre um evento. Porém, como a funcionalidade de eventos não foi concluída, ela não será abordada neste trabalho.

A área de gerenciamento de locais, assim como todas as subseções descritas anteriormente na Seção 3.6.5, possui uma tabela que lista todos os locais já cadastrados, que são buscados utilizando o método HTTP do tipo GET (Figura 3.20).

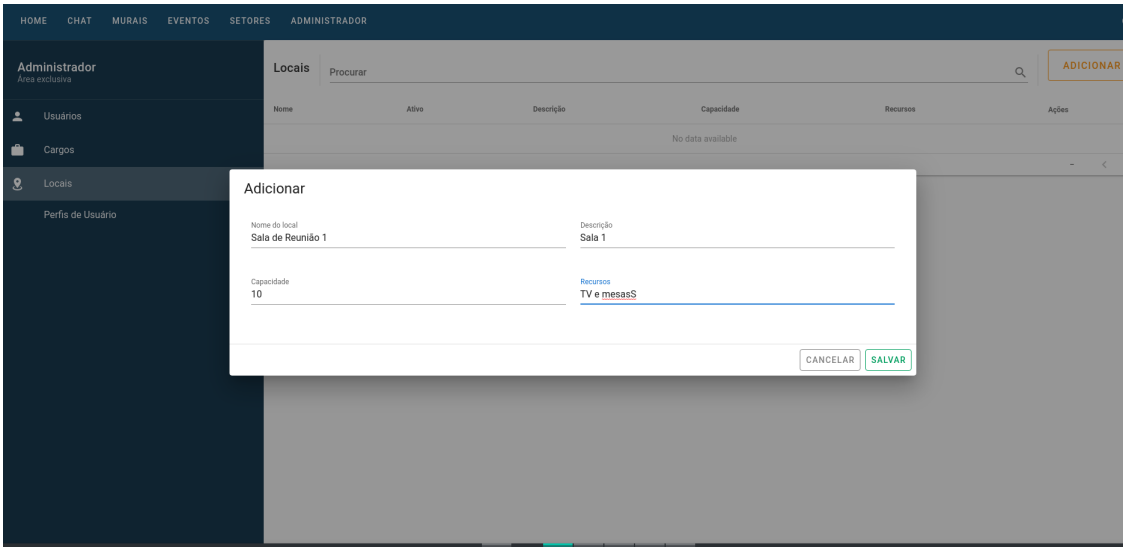
Os locais também possuem as funcionalidades de bloqueio e desbloqueio (método HTTP tipo PATCH), criar e editar um novo local informando um **nome**, **descrição**, **capacidade de pessoas** e **recursos** que pode ser encontrados no local. Criar e editar um novo local utilizam métodos HTTP tipo POST e PUT, respectivamente (Figura 3.21).

Figura 3.20 – Tabela de listagem de locais



Fonte: Autora

Figura 3.21 – Modal de adicionar local



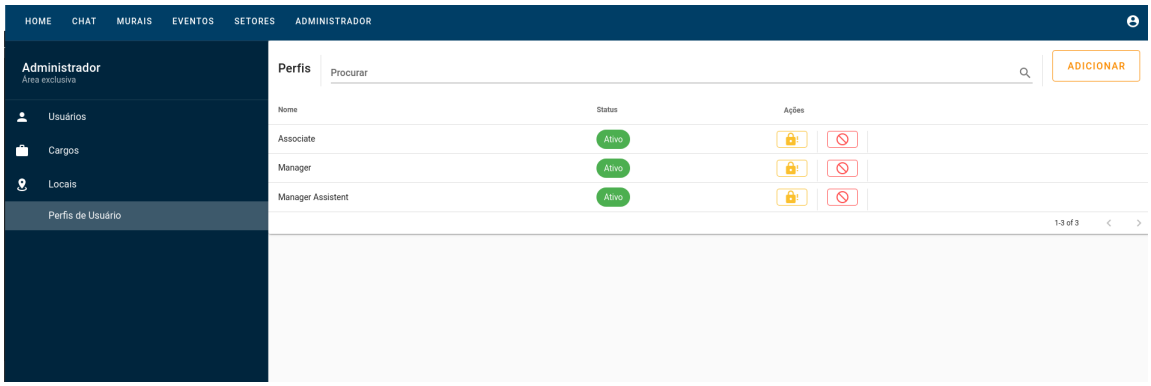
Fonte: Autora

3.6.5.4 Gerenciamento de Perfis

Os perfis de usuários são o último item da área do administrador. A listagem de perfis em tabela (ver Figura 3.22) permite que o administrador desative ou ative um perfil e possa alterar as permissões daquele perfil.

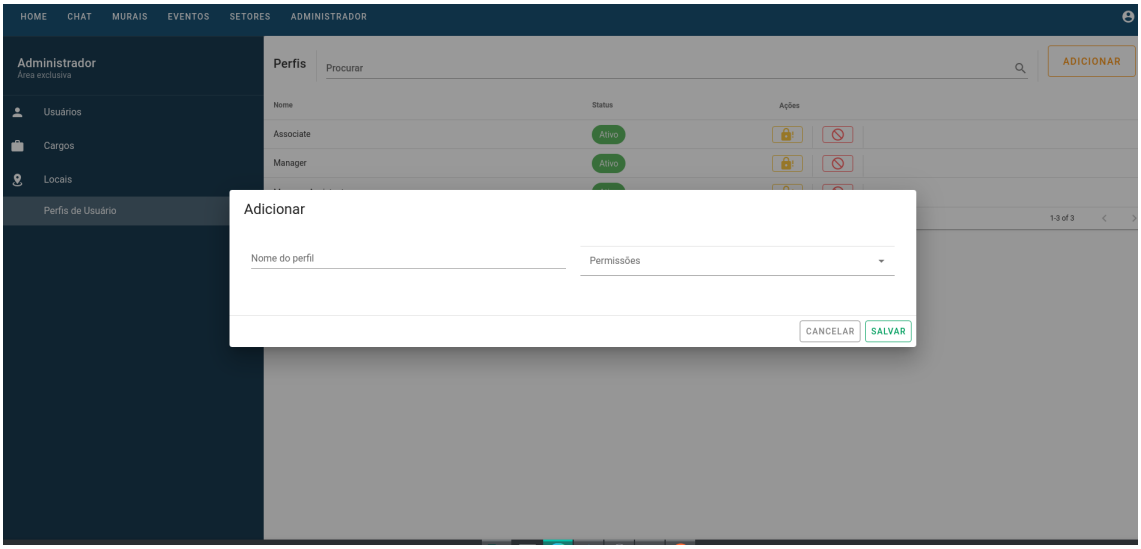
O administrador também pode criar um novo perfil informando seu **nome** e suas **permissões** que estarão previamente adicionadas no banco de dados (ver Figura 3.23).

Figura 3.22 – Tabela de listagem de perfis



Fonte: Autora

Figura 3.23 – Modal de adicionar perfis



Fonte: Autora

4 CONSIDERAÇÕES FINAIS

A experiência prática adquirida durante o período de estágio foi essencial para o desenvolvimento profissional da estagiária. O contato com as tecnologias mais utilizadas no mercado, além de consolidar conteúdos aprendidos durante o curso, também estimulou a escolha da área de desenvolvimento como profissão e a entrada no mercado de trabalho. Também proporcionou a vontade de aprender e se aprofundar nos conteúdos e ferramentas utilizadas, aplicando a base que foi aprendida também em outras linguagens e tecnologias.

Os desafios durante o desenvolvimento foram o combustível para a aprendizagem. A maior curva de aprendizado foi para utilização do banco de dados MongoDB devido ao seu conceito NoSql e também a utilização do Docker devido ao conceito de *container* ser bastante abstrato ainda para a aluna.

As disciplinas cursadas durante o curso de Ciência da Computação foram excelentes bases para cada etapa do desenvolvimento. As disciplinas Introdução aos Algoritmos, Estrutura de Dados, Práticas de Programação Orientada a Objetos e Paradigmas de Linguagem de Programação foram bases essenciais para solidificar a lógica de programação utilizada na resolução de problemas.

Operações realizadas no banco de dados como a persistência de dados, consulta, edição e remoção de informações foram atividades com pequena curva de aprendizado devido às disciplinas de Introdução a Sistemas de Banco de Dados e Sistemas Gerenciadores de Banco de Dados. A disciplinas de Engenharia de Software contribuiu para as primeiras fases do projeto desenvolvido como levantamento e especificação de requisitos. Para construção do *front-end* a disciplina Interação Humano-Computador foi muito importante para o pensamsento em melhores interações com o usuário do sistema.

Após a finalização do estágio não houve notícias se o desenvolvimento da plataforma de comunicação interna foi continuada pela FUNDECC e se existiu implantação para uso em ambiente de produção.

Por fim, o período de estágio e o desenvolvimento da plataforma de comunicação interna tratada nesse trabalho, foi de muita importância e inestimável valor. Trouxe novas experiências à aluna, além do desenvolvimento profissional e técnico. Houve também o crescimento pessoal e de *soft skills* como trabalho em equipe, comunicação e proatividade.

REFERÊNCIAS

ATLASSIAN. **O que é Git**. 2022. Acessado em abril de 2022. Disponível em: <https://www.atlassian.com/br/git/tutorials/what-is-git>.

DIAS, L. G. **Por que usar o Node.js?** 2021. Acessado em março de 2022. Disponível em: <https://blog.geekhunter.com.br/por-que-usar-node-js-uma-justificativa-passo-a-passo/>.

FUNDECC. **Sobre a FUNDECC**. 2022. Acessado em março de 2022. Disponível em: <http://www.fundecc.org.br/sobre-a-fundecc>.

GABRIEL, J. **Comunicação cliente-servidor em tempo real com Socket.IO**. 2020. Acessado em maio de 2022. Disponível em: <https://medium.com/digitalproductsdev/comunica%C3%A7%C3%A3o-cliente-servidor-em-tempo-real-com-socket-io-9d3930484b80>.

GALLOTI, G. M. **Arquitetura de Software**. [S.l.]: Editora Pearson, 2017. v. 1.

KOVACS, L. **O que é e para que serve o MongoDB?** 2021. Acessado em abril de 2022. Disponível em: <https://tecnoblog.net/responde/o-que-e-e-para-que-serve-o-mongodb>.

LONGEN, A. **O Que é HTML? Guia Básico Para Iniciantes**. 2022. Acessado em março de 2022. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-html-conceitos-basicos>.

MOLL, V. **Como construir uma aplicação com Docker?** 2022. Acessado em abril de 2022. Disponível em: <https://blog.geekhunter.com.br/docker-na-pratica-como-construir-uma-aplicacao/>.

MONGODB. **O que é o MongoDB?** 2022. Acessado em abril de 2022. Disponível em: <https://www.mongodb.com/pt-br/what-is-mongodb>.

MOZILLA. **Introdução JavaScript**. 2020. Acessado em março de 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Introduction>.

MOZILLA. **HTML: Linguagem de Marcação de Hipertexto**. 2021. Acessado em maio de 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>.

MOZILLA. **Métodos de requisição HTTP**. 2021. Acessado em março de 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>.

MOZILLA. **Métodos de requisição HTTP**. 2021. Acessado em maio de 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction>.

NOLETO, C. **O que é API REST**. 2022. Acessado em março de 2022. Disponível em: <https://blog.betrybe.com/desenvolvimento-web/api-rest-tudo-sobre/#1>.

PICOLLO, L. **Vue JS: o que é, como funciona e vantagens**. 2020. Acessado em abril de 2022. Disponível em: <https://blog.geekhunter.com.br/vue-js-so-vejo-vantagens-e-voce/>.

VALENTE, M. T. **Engenharia de Software Moderna**. 2022. Disponível em: <https://engsoftmoderna.info/>.

VIEIRA, K. **O que é CSS e como ele funciona**. 2010. Acessado em abril de 2022. Disponível em: <https://www.hostgator.com.br/blog/o-que-e-css/?gclid=CjwKCAjw3cSSBhBGEiwAVII0Z3ZUsSBPBQ3n6w62oeawkGoPlS66q1pe3Ou8HXZSay3Sb6B_ZKxxIxoCvjIQAvD_BwE>.

VUEX. **O que é Vuex?** 2022. Acessado em abril de 2022. Disponível em: <https://vuex.vuejs.org/ptbr/>.