CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

JULIA SALVADOR

PROTÓTIPO DE SISTEMA DE GESTÃO PARA ABRIGOS UTILIZADOS EM SITUAÇÕES DE CRISES CLIMÁTICAS NO MUNICÍPIO DE RIO DO SUL/SC

RIO DO SUL
2024

CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

JULIA SALVADOR

PROTÓTIPO DE SISTEMA DE GESTÃO PARA ABRIGOS UTILIZADOS EM SITUAÇÕES DE CRISES CLIMÁTICAS NO MUNICÍPIO DE RIO DO SUL/SC

Trabalho de Conclusão de Curso a ser apresentado ao curso de Sistemas da Informação, da Área das Ciências Naturais, da Computação e das Engenharias, do Centro Universitário para o Desenvolvimento do Alto Vale do Itajaí, como condição parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Orientador: Ademar Perfoll Junior

RIO DO SUL
2024

CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

JULIA SALVADOR

PROTÓTIPO DE SISTEMA DE GESTÃO PARA ABRIGOS UTILIZADOS EM SITUAÇÃO DE CRISES CLIMÁTICAS NO MUNICÍPIO DE RIO DO SUL/SC

Trabalho de Conclusão de Curso a ser apresentado ao curso de Sistemas da Informação, da Área das Ciências Naturais, da Computação e das Engenharias, do Centro Universitário para o Desenvolvimento do Alto Vale do Itajaí - UNIDAVI, a ser apreciado pela Banca Examinadora, formada por:

_____

Professor Orientador: Ademar Perfoll Junior

Banca Examinadora:

_____

Prof. Fernando Andrade Bastos

_____

Prof. Sandro Alencar Fernandes

Rio do Sul, 25 de novembro de 2024.

AGRADECIMENTOS

Agradeço imensamente ao meu orientador, Ademar Perfoll Junior, por todo o apoio e orientação durante o desenvolvimento deste sistema. Sua ajuda constante foi essencial para que este projeto se concretizasse.

Meus sinceros agradecimentos à minha família e amigos, que me apoiaram ao longo de todo o processo.

Agradeço especialmente aos meus pais Nelson Salvador, Neide Teresinha Rossetti Salvador e a minha namorada Ana Paula Schonarth, que estiveram ao meu lado, me incentivando a continuar e sempre acreditando no meu trabalho, sem o apoio de vocês nada disso seria possível.

Agradeço também a Larissa Ferreira, Guilherme Felix e a Mariana Dircksen apoio e motivação. Cada um de vocês foi fundamental para a conclusão deste projeto.

RESUMO

Entre 2019 e 2024, o município de Rio do Sul enfrentou crises climáticas recorrentes, com um total de 17 enchentes, de acordo com dados da Defesa Civil (2024). Esses eventos destacam a necessidade de um gerenciamento eficaz para enfrentar o aumento da frequência e intensidade das enchentes. Este trabalho tem como objetivo desenvolver um protótipo de sistema web para realizar a gestão de abrigos durante essas crises. A metodologia utilizada caracteriza-se como uma pesquisa aplicada e descritiva. Para atingir os objetivos, foi realizada uma revisão de literatura sobre as tecnologias e linguagens de programação aplicadas no desenvolvimento. Para compreender o processo atual de controle da Secretaria de Assistência Social, foi conduzido um levantamento de informações por meio de um questionário. O capítulo de análise apresenta uma visão geral do protótipo, incluindo o levantamento de requisitos, diagramas de caso de uso e histórias de usuários. Já o capítulo de implementação do projeto contém todos os detalhes necessários para o desenvolvimento do sistema, abordando desde a modelagem do banco de dados até a implementação do *backend* e *frontend*. Este protótipo possibilita o gerenciamento de abrigos em eventos de crise climática, permitindo o registro detalhado de enchentes, a disponibilização de abrigos com base em cada evento e o monitoramento da ocupação, incluindo a quantidade de famílias, integrantes e pets em cada abrigo. Além disso, o sistema oferece controle sobre os recursos disponíveis, com funcionalidades para registrar entradas, saídas e transferências de suprimentos entre abrigos, criando uma visão centralizada e integrada das operações. Dessa forma, o sistema facilita o acesso e a atualização das informações, tornando o processo de gestão mais eficiente e responsivo durante situações de crise climática. A validação do protótipo foi realizada junto à Secretaria de Assistência Social, onde o projeto recebeu uma boa avaliação. No entanto, foram sugeridas algumas melhorias para tornar o sistema ainda mais prático, aprimorando sua utilidade na gestão dos abrigos.

Palavras-Chave: Crises Climáticas, Gestão de Abrigos, Secretaria de Assistência Social, Sistemas de Informação.

ABSTRACT

Between 2019 and 2024, the municipality of Rio do Sul faced recurring climate crises, with a total of 17 floods, according to data from the Civil Defense (2024). These events highlight the need for effective management to face the increasing frequency and intensity of floods. This work aims to develop a prototype of a web system to manage shelters during these crises. The methodology used is characterized as applied and descriptive research. To achieve the objectives, a literature review was carried out on the technologies and programming languages applied in the development. To understand the current control process of the Social Assistance Department, an information survey was conducted through a questionnaire. The analysis chapter presents an overview of the prototype, including the requirements survey, use case diagrams, and user stories. The project implementation chapter contains all the details necessary for the development of the system, covering everything from database modeling to backend and frontend implementation. This prototype enables shelter management during climate crisis events, allowing detailed recording of floods, provision of shelters based on each event, and monitoring of occupancy, including the number of families, members, and pets in each shelter. In addition, the system provides control over available resources, with features for recording entries, exits, and transfers of supplies between shelters, creating a centralized and integrated view of operations. In this way, the system facilitates access and updating of information, making the management process more efficient and responsive during climate crisis situations. The prototype was validated with the Department of Social Welfare, where the project received a good evaluation. However, some improvements were suggested to make the system even more practical, enhancing its usefulness in shelter management.

Keywords: Climate Crises, Manage Shelters, Social Assistance Department, Information Systems.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

É evidente que as crises climáticas, como enchentes, têm se tornado uma preocupação crescente em várias partes do mundo, inclusive no município de Rio do Sul. Esse fenômeno está ocorrendo cada vez com mais frequência e intensidade, afetando diretamente a população da região, principalmente os moradores que residem em áreas de risco. Durante esses eventos climáticos, a gestão de risco é crucial para garantir a segurança e bem-estar das pessoas afetadas. Com base nos dados da Defesa Civil de Rio do Sul (2024), entre 2019 e 2024, o município enfrentou 17 enchentes. De acordo com Azevedo (2023), uma enchente, ou cheia, é a elevação do nível da água em um canal de drenagem até sua capacidade máxima. Esse fenômeno pode ter causas naturais ou humanas e impacta, de maneira particular, as populações mais vulneráveis.

Sem uma gestão adequada dos abrigos e das pessoas afetadas, surgem desafios significativos que podem agravar a situação durante crises climáticas. Embora o site da Defesa Civil atualmente ofereça o mapeamento dos abrigos disponíveis, ele não está integrado a um sistema de gestão, o que gera dificuldades, como a falta de atualização em tempo real sobre a capacidade dos abrigos, a ocupação por famílias e indivíduos e a disponibilidade de recursos. Essa desconexão dificulta o planejamento e a distribuição eficiente de suprimentos e torna mais complexa a coordenação das equipes de assistência, que precisam de informações precisas e atualizadas para oferecer suporte adequado às pessoas em risco.

Diante da necessidade de uma gestão mais eficiente dos abrigos e das pessoas afetadas durante crises climáticas, surge um problema: como um sistema de gestão de abrigos pode melhorar a segurança e a eficiência no atendimento à população durante crises climáticas em Rio do Sul. Existem diversas plataformas de monitoramento de emergências, como o *Global Disaster Alert and Coordination System (GDACS)*, que fornece alertas em tempo real e coordena respostas a desastres naturais em nível global e o Centro Nacional de Monitoramento e Alerta de Desastres Naturais (CEMADEN) no Brasil, que monitora e emite alertas sobre desastres como enchentes e deslizamentos, contribuindo para a tomada de decisões e resposta rápida em situações de crise, porém, para a gestão precisa dos abrigos e recursos durante eventos climáticos adversos, são necessárias soluções adaptadas e focadas nas necessidades específicas dessa situação. Essa solução não apenas melhoraria a resposta às crises climáticas, mas também contribuiria para a segurança e a resiliência da comunidade de Rio do Sul.

Com base na problemática apresentada, o sistema proposto visa preencher a lacuna existente no mercado, oferecendo uma solução personalizada e eficiente para atender às necessidades específicas da gestão de abrigos durante crises climáticas. Este estudo tem como objetivo explorar as funcionalidades essenciais e os potenciais benefícios que um sistema desse tipo pode proporcionar, contribuindo para a melhoria e aprofundamento da gestão de emergências climáticas na região de Rio do Sul.

1.1 PROBLEMA DE PESQUISA

Como um sistema de gestão pode auxiliar na organização, na utilização e na logística dos abrigos durante as crises climáticas?

1.2 OBJETIVOS

1.2.1 Geral

- Desenvolver um protótipo de aplicação web para auxiliar na organização, utilização e logística de abrigos durante crises climáticas.

1.2.2 Específicos

- Levantar informações sobre a gestão dos abrigos com a Secretaria de Assistência Social;
- Detalhar as tecnologias que serão utilizadas na implementação do protótipo por meio de uma revisão da literatura;
- Estabelecer o projeto do sistema com base na implementação dos requisitos;
- Construir o protótipo com as tecnologias escolhidas, atendendo aos requisitos funcionais;
- Validar o uso do protótipo com usuários da Secretaria de Assistência Social.

1.3 JUSTIFICATIVA

As crises climáticas, como as enchentes, tornaram-se cada vez mais frequentes e comuns em muitas partes do mundo, incluindo o município de Rio do Sul. É inevitável reconhecer que uma resposta rápida e eficiente é crucial para garantir a segurança e o bem-estar das pessoas afetadas. Nesse contexto, a gestão dos abrigos e das pessoas durante essas situações emergenciais se torna de suma importância.

Um sistema integrado de gestão de abrigos pode ser uma solução eficaz para os desafios enfrentados durante crises climáticas. Ao fornecer ferramentas para a gestão eficiente dos abrigos, incluindo o controle da quantidade de famílias, integrantes e pets abrigados, essa solução pode melhorar significativamente a resposta às emergências. Além disso, o sistema possibilita a gestão de recursos, garantindo que estoques sejam utilizados de forma otimizada e que as necessidades dos abrigos sejam atendidas prontamente. Esse sistema contribuirá para uma resposta mais coordenada e eficaz durante crises climáticas, facilitando o monitoramento da capacidade de abrigos e permitindo uma distribuição eficiente de recursos.

Atualmente, a Secretaria de Assistência Social e Defesa Civil não contam com uma ferramenta específica para a gestão de abrigos, recursos e necessidades das pessoas abrigadas. Conforme relatos, as informações começaram recentemente a ser organizadas em planilhas, visando melhorar a administração e o controle. Essas planilhas estarão disponíveis no Apêndice A, oferecendo uma visão detalhada da estrutura de dados atualmente utilizada e destacando a necessidade de uma solução integrada para otimizar o gerenciamento dos abrigos durante crises climáticas. Segundo a Secretaria, a implementação de um sistema específico para essa finalidade seria de grande importância.

Portanto, esse projeto não apenas aborda a necessidade de melhorar a resposta às crises climáticas, mas também visa realizar uma gestão eficiente de abrigos no município de Rio do Sul. A implementação desse sistema pode servir como um modelo para outras regiões que enfrentam desafios semelhantes, contribuindo para a construção de comunidades mais seguras e preparadas para enfrentar desastres naturais.

2. REFERENCIAL TEÓRICO

Neste capítulo, busca-se explorar os conceitos e fundamentos que sustentam o desenvolvimento do projeto, abordando aspectos essenciais como a engenharia de requisitos, suas subdivisões e as implicações práticas no processo de construção de sistemas. Esses fundamentos fornecem a base teórica necessária para a compreensão e a execução do trabalho.

2.1 ENGENHARIA DE REQUISITOS

A engenharia de requisitos é uma etapa fundamental no desenvolvimento de sistemas, pois visa garantir que as necessidades dos usuários sejam atendidas de maneira eficiente e alinhada às capacidades técnicas do sistema. Esse processo busca organizar as ideias e expectativas dos stakeholders, promovendo uma compreensão compartilhada entre todos os envolvidos no projeto.

Nesse contexto, os requisitos de um sistema representam descrições detalhadas das funcionalidades, serviços e limitações que o sistema deve possuir. Conforme destacado por Sommerville (2011), esses requisitos refletem as demandas dos clientes para um sistema projetado com um propósito específico, como controlar um dispositivo, realizar um pedido ou buscar informações. Identificar, analisar, documentar e verificar esses aspectos constitui o processo da engenharia de requisitos, essencial para o sucesso do desenvolvimento de qualquer sistema.

2.1.1 Requisitos Funcionais e Não Funcionais

Os requisitos de um sistema são classificados em funcionais e não funcionais, sendo ambos indispensáveis para garantir que o software atenda às expectativas dos usuários e às demandas organizacionais. Essa distinção permite uma análise mais detalhada das operações do sistema e de suas propriedades essenciais.

Segundo Sommerville (2011), os requisitos funcionais de um sistema descrevem suas operações essenciais, adaptando-se ao tipo de software, aos potenciais usuários e à abordagem organizacional. Quando expressos para os usuários, são abstratos, enquanto os requisitos específicos do sistema detalham suas funções, entradas e saídas.

Já os requisitos não funcionais, por outro lado, não estão diretamente ligados aos serviços do sistema, mas abrangem propriedades emergentes ou restrições de implementação.

São aspectos como confiabilidade, tempo de resposta e capacidades de dispositivos, que têm um impacto crítico no sistema como um todo e são mais importantes do que os requisitos funcionais individuais.

Os requisitos não funcionais vêm das necessidades dos usuários, limitações de orçamento, políticas da empresa, exigências de compatibilidade com outros *softwares* ou *hardwares*, e fatores externos, como regulamentos de segurança ou leis de privacidade. Eles podem ser baseados nas características necessárias para o software (requisitos do produto), na organização que o desenvolve (requisitos organizacionais) ou em fontes externas, conforme exemplifica o Quadro 1.

Quadro 1 - Tipos de requisitos não funcionais

Nome	Conceito
Requisitos de produto	Esses requisitos especificam ou restringem o comportamento do *software*. Exemplos incluem os requisitos de desempenho quanto à rapidez com que o sistema deve executar e quanta memória ele requer, os requisitos de confiabilidade que estabelecem a taxa aceitável de falhas, os requisitos de proteção e os requisitos de usabilidade
Requisitos organizacionais	Esses são os requisitos gerais de sistemas derivados das políticas e procedimentos da organização do cliente e do desenvolvedor. Exemplos incluem os requisitos do processo operacional, que definem como o sistema será usado, os requisitos do processo de desenvolvimento que especificam a linguagem de programação, o ambiente de desenvolvimento ou normas de processo a serem usadas, bem como os requisitos ambientais que especificam o ambiente operacional do sistema.
Requisitos externos	Esses são os requisitos gerais de sistemas derivados das políticas e procedimentos da organização do cliente e do desenvolvedor. Exemplos incluem os requisitos do processo operacional, que definem como o sistema será usado, os requisitos do processo de desenvolvimento que especificam a linguagem de programação, o ambiente de desenvolvimento ou normas de processo a serem usadas, bem como os requisitos ambientais que especificam o ambiente operacional do sistema.

Fonte: Elaborado a partir de Sommerville (2011).

2.1.2 Regra de Negócio

As regras de negócio são importantes no desenvolvimento de sistemas, pois transformam as necessidades da empresa em orientações claras. Elas ajudam a garantir que os objetivos e processos da empresa sejam entendidos e aplicados corretamente, conectando as áreas de negócios e tecnologia.

De acordo com Amoesei (2023), uma regra de negócio converte uma necessidade de negócio, como validações e restrições, em regras lógicas. Isso permite que as áreas de desenvolvimento, produto e negócios de uma empresa estejam alinhadas quanto a essas necessidades, utilizem-nas como guia e implementem as regras de forma clara e precisa. Tal abordagem garante que o desenvolvimento e a evolução do produto ocorram de maneira

otimizada. As regras de negócio ajudam as empresas a comunicar seus objetivos, determinações, limitações e procedimentos de seus produtos de forma transparente, tanto para os desenvolvedores quanto para os usuários.

2.2 ARQUITETURA DE SOFTWARE

A arquitetura de software é fundamental no desenvolvimento de sistemas, pois organiza os principais componentes de um projeto. Ela inclui decisões importantes que influenciam todas as etapas do desenvolvimento, ajudando a criar sistemas eficientes e bem estruturados.

Segundo Maxim e Pressman (2021), a arquitetura é composta por inúmeras decisões, sejam elas grandes ou pequenas. Algumas dessas decisões são tomadas no início da implementação do projeto, influenciando profundamente todas as etapas posteriores, enquanto outras são adiadas o máximo possível para evitar restrições que poderiam levar a uma implementação inadequada do estilo arquitetural.

"Acreditamos que um projeto de software pode ser considerado uma instância de uma arquitetura de software específica. Contudo, os elementos e estruturas definidos como parte de uma arquitetura são a raiz de todo projeto. Recomendamos que o projeto se inicie com uma consideração da arquitetura de software." (Maxim; Pressman, 2021, p.182).

2.2.1 Arquitetura Web

A arquitetura web é a base para o funcionamento de aplicações acessadas pela internet, organizando como os componentes internos e externos se conectam e interagem. Ela garante que páginas e serviços sejam entregues de forma eficiente aos usuários.

De acordo com o Viana (2023), a arquitetura de aplicações web define a estrutura interna e as interações entre seus componentes, bancos de dados e sistemas externos. Essa arquitetura é essencialmente distribuída e segue o modelo cliente-servidor. Os clientes são os dispositivos usados para acessar as páginas web por meio de navegadores, enquanto os servidores são os computadores responsáveis por hospedar e disponibilizar essas páginas para os clientes.

> Em um sistema cliente-servidor, o usuário interage com um programa em execução em seu computador local (por exemplo, um browser de Web ou uma aplicação baseados em telefone). Este interage com outro programa em execução em um computador remoto (por exemplo, um servidor Web). O computador remoto fornece

> serviços, como o acesso a páginas Web, que estão disponíveis para clientes externos. (Sommerville, 2011, p.339)

Além disso, Sommerville (2011) destaca que, em uma estrutura cliente-servidor, a aplicação é organizada como um grupo de serviços oferecidos por servidores. Os clientes podem utilizar esses serviços e exibir os resultados para os usuários. Embora os clientes precisem ter conhecimento sobre quais servidores estão acessíveis, eles não devem ter informações sobre outros clientes que também estão conectados.

2.2.2 MVC

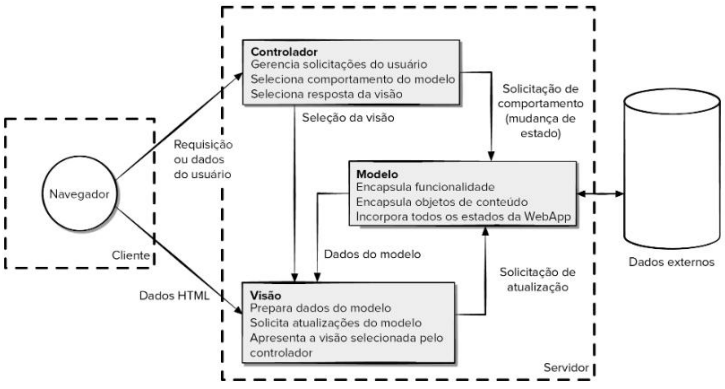
O padrão MVC (Model-View-Controller) organiza o desenvolvimento de sistemas ao separar suas responsabilidades em três partes distintas. O modelo gerencia os dados e a lógica, a visão apresenta as informações ao usuário, e o controlador atua como um elo, coordenando as ações entre eles. Essa abordagem traz clareza, facilita ajustes e promove melhorias no sistema.

Segundo Maxim e Pressman (2021), a arquitetura Modelo-Visão-Controlador (MVC, do inglês *Model-View-Controller*) é uma das diversas estruturas utilizadas no desenvolvimento de aplicações web.

> O modelo contém todo o conteúdo e a lógica de processamento específicos à aplicação. A visão contém todas as funções específicas à interface e possibilita a apresentação do conteúdo e lógica de processamento exigido pelo usuário. O controlador gerencia o acesso ao modelo e à visão e coordena o fluxo de dados entre eles. (Maxim; Pressman, 2021, p.189)

A Figura 1 ilustra essa estrutura de maneira esquemática.

Figura 1 - Arquitetura MVC



Fonte: Maxim; Pressman (2021, p.189).

Maxim e Pressman (2021) explicam que, o controlador escolhe o componente de visualização adequado com base no pedido do usuário. Quando o tipo de solicitação é identificado, o controlador envia uma instrução para o modelo, que então executa a funcionalidade necessária ou recupera o conteúdo requisitado. O modelo pode acessar dados de um banco de dados corporativo, de um repositório local ou de arquivos independentes. Os dados processados pelo modelo são então organizados e formatados pela visualização e enviados do servidor para o navegador do usuário, onde são exibidos.

Quando a etapa de engenharia de requisitos identifica as propriedades e limitações do sistema que será desenvolvido, torna-se possível selecionar o estilo arquitetônico ou a combinação de padrões que mais se adeque a essas especificidades. Frequentemente, há mais de uma opção viável, permitindo a criação e análise de alternativas arquiteturais para determinar a mais adequada.

2.3 PROTÓTIPOS

Os protótipos são ferramentas essenciais no desenvolvimento de sistemas, pois permitem visualizar e testar ideias antes da implementação final. Eles ajudam a explorar funcionalidades, identificar ajustes necessários e alinhar expectativas entre os envolvidos no projeto, sem a necessidade de investir em soluções definitivas desde o início.

De acordo com Paula Filho (2019), o protótipo de requisitos é um modelo visual de baixa fidelidade, cujo objetivo é explorar aspectos críticos dos requisitos de um produto, simulando rapidamente um pequeno subconjunto de suas funcionalidades. Esse subconjunto não visa representar com precisão as interfaces de usuário do produto final, podendo ter uma aparência bastante distinta, desde que seja funcionalmente equivalente. É possível haver variação no número e nos nomes dos elementos das interfaces, como telas, painéis e outros componentes, bem como nos campos e comandos. No protótipo de requisitos, são abordadas apenas questões de funcionalidade, sem tratar aspectos de usabilidade.

Alguns métodos para prototipagem de requisitos, ordenados da tecnologia mais baixa para a mais alta, incluem:

- desenhos à mão livre, em papel, possivelmente capturados *on-line*;
- leiautes alfanuméricos grosseiros, feitos com um editor de texto;
- leiautes feitos com planilhas e editores de hipertexto;

- desenhos feitos com uma ferramenta de desenho técnico;
- telas desenhadas em um ambiente de desenvolvimento rápido;
- telas desenhadas no ambiente definitivo de implementação.

Ainda segundo o autor, inicialmente, a criação do protótipo deve evitar o uso de ambientes de desenvolvimento ou ferramentas sofisticadas de design. O foco deve estar em uma construção ágil, descomplicada e desvinculada da tecnologia que será utilizada na solução final. Recursos como editores de sites, processadores de texto ou até mesmo planilhas eletrônicas são adequados para criar protótipos com excelente qualidade.

2.3.1 Pencil

O Pencil é uma ferramenta popular para quem precisa criar protótipos e maquetes de forma rápida e eficiente. Ele é amplamente utilizado por designers e desenvolvedores para planejar interfaces de usuário, facilitando a visualização e o ajuste de ideias antes da implementação.

Conforme destacado por Escolha Livre (2024), o Pencil é uma solução para criar protótipos de interfaces gráficas de usuário (GUI). Ele permite desenvolver diagramas, maquetes de interface e *wireframes* de maneira prática. Essa ferramenta foi criada com foco na simplicidade e leveza, sendo ideal para elaborar rapidamente conceitos de interfaces para programas, sites ou aplicativos móveis. O Pencil inclui uma vasta coleção de formas, modelos pré-configurados e elementos visuais que facilitam a criação dos protótipos. Além disso, ele é compatível com diversos sistemas operacionais, como Windows, MacOS e Linux.

2.3.2 Draw.io

O Draw.io é uma ferramenta fácil de usar para criar diagramas. Muito usada por estudantes e profissionais, ela ajuda a organizar ideias, planejar projetos e apresentar informações de forma visual.

De acordo com a documentação de Draw.io (2023), o editor draw.io é uma ferramenta de diagramação gratuita e de código aberto que pode ser utilizada tanto *on-line* quanto *off-line*. Ela oferece um conjunto de tecnologias para a criação de aplicativos de diagramação e é o *software* de diagramação baseado em navegador mais utilizado globalmente.

O Draw.io disponibiliza recursos para elaborar diversos tipos de diagramas, como fluxogramas, mapas mentais, organogramas, diagramas de Venn, infográficos, diagramas de rede e arquitetura, plantas baixas, diagramas elétricos e de rack, diagramas UML e muitos outros.

2.4 HTML

O HTML é uma das principais linguagens utilizadas na construção de páginas na web. Ela organiza e estrutura o conteúdo exibido nos sites, permitindo a criação de links, textos, imagens e outros elementos visuais.

De acordo com Silva (2015), HTML é a sigla para *HyperText Markup Language*, que, em português, significa linguagem de marcação de hipertexto. O termo hipertexto refere-se a qualquer texto em um documento web que permite a conexão com outros documentos na internet, utilizando links amplamente conhecidos e presentes nas páginas dos sites que costumamos acessar.

> Então, todo o conteúdo textual que você vê em uma página de um site é um hipertexto, assim como imagens, vídeos, gráficos, sons e conteúdos não textuais em geral são chamados de hipermídia. Quando a HTML foi inventada, os conteúdos eram essencialmente hipertextos, com a hipermídia surgindo posteriormente. Assim, hoje, a HTML é uma linguagem para marcação de conteúdos web em geral. (Silva, 2015, p.19)

Bertagnolli e Miletto (2014) apresentam as principais *tags* para estruturação de um documento básico em html, conforme apresentado no Quadro 2.

Quadro 2 - Tags HTML

Tags	**Conceito**
Html	Marca o início e o fim da página web, informando ao navegador que o texto contido no documento está escrito em HTML.
Head	Marca o início e fim do cabeçalho, a área onde serão descritos os cabeçalhos e o título da página. pode ser utilizado, ainda, para declarar *scripts* em Javascript ou definir formatações CSS.
Title	Marca o início e o fim do título da página, que sempre está posicionado na barra superior do *browser*.
Body	Marca o início e o fim do corpo da página, que contém imagens, textos, títulos, links e etc.

Fonte: Elaborado a partir de Bertagnolli e Miletto (2014, p.72).

2.5 CSS

O CSS é uma linguagem essencial no desenvolvimento web, responsável por definir o estilo e a aparência das páginas. Ele trabalha em conjunto com o HTML, garantindo que o conteúdo seja exibido de forma visualmente atraente e organizada.

De acordo com Silva (2012), CSS é a sigla para *Cascading Style Sheets*, que em português significa folhas de estilo em cascata. A função do CSS é devolver à marcação HTML/XML seu propósito original. A HTML foi desenvolvida como uma linguagem para marcar e estruturar conteúdos, e não para definir a aparência dos elementos. Segundo os criadores da HTML, a linguagem não deveria determinar aspectos visuais como cores das fontes, tamanhos dos textos, posicionamentos e demais características visuais de um documento. Essas funções são responsabilidade do CSS.

> Regra CSS é a unidade básica de uma folha de estilo. Nessa definição, o termo unidade básica significa a menor porção de código capaz de produzir efeito de estilização. Uma regra CSS é composta de duas partes: o seletor e a declaração. A declaração compreende uma propriedade e um valor. (Silva, 2012, p.26)

Ainda segundo o autor, a determinação dos elementos de uma regra é delineada conforme apresentado no Quadro 3.

Quadro 3 - Definição dos componentes de uma regra CSS

Nome	**Conceito**
Seletor	É o alvo da regra CSS.
Declaração	Determina os parâmetros de estilização. Compreende a propriedade e valor.
Propriedade	Define qual será a característica do seletor a ser estilizada.
Valor	É a quantificação ou qualificação da propriedade.

Fonte: Elaborado a partir de Silva (2012).

2.5.1 Tailwind CSS

O Tailwind CSS é uma ferramenta moderna e poderosa para estilização de interfaces web. Ele se destaca por sua abordagem prática, permitindo que desenvolvedores criem designs personalizados de forma rápida e eficiente, diretamente no HTML.

Segundo a Ariel (2024), Tailwind CSS é um *framework* de utilitários CSS, flexível e de código aberto, projetado para facilitar a criação de interfaces de usuário atraentes e adaptáveis. Diferente dos *frameworks* CSS tradicionais, ele não foca em componentes com estilos pré-definidos, mas oferece uma vasta coleção de classes utilitárias que podem ser

aplicadas diretamente ao HTML, permitindo a personalização de elementos de forma rápida e precisa.

> Em vez de escrever CSS personalizado para cada estilo, o Tailwind CSS oferece uma abordagem baseada em classes que representam estilos individuais. Cada classe utilitária é projetada para realizar uma tarefa específica, como definir cores, tamanhos, espaçamento, posicionamento e muito mais. Ao combinar essas classes, você pode criar estilos complexos e layouts responsivos sem escrever CSS customizado extensivo. (Ariel, 2024, n.p)

A autora também ressalta que, o Tailwind CSS tem como principal objetivo aumentar a produtividade dos desenvolvedores, possibilitando a criação de *designs* de maneira mais ágil e eficiente. Além disso, ele contribui para um fluxo de trabalho mais organizado, ao permitir que os estilos sejam definidos diretamente no HTML. Isso torna o código mais claro e intuitivo, facilitando a colaboração entre as equipes de design e desenvolvimento.

2.6 LINGUAGEM DE PROGRAMAÇÃO

As linguagens de programação são ferramentas essenciais para comunicar instruções aos computadores. Elas funcionam como um idioma, com regras e estrutura, permitindo criar soluções e desenvolver sistemas de forma eficiente.

De acordo com Santana (2023), a língua portuguesa falada no Brasil é semelhante às linguagens de programação, pois ambas utilizam um conjunto de símbolos, que no caso do português são as letras do alfabeto. A combinação dessas letras gera as palavras, e a maneira como estas são organizadas em uma frase corresponde à sintaxe. As palavras e as frases carregam um significado (ou semântica) e o contexto nos ajuda a compreender melhor esse significado.

“Sendo assim, podemos dizer que as linguagens de programação nada mais são do que um idioma com regras de escrita, para garantir a comunicação com o computador” (Santana, 2023, n.p).

O autor também aponta que, o computador não é capaz de interpretar informações da mesma forma que um ser humano, por isso são criadas várias linguagens de programação. Dessa maneira, podemos nos comunicar com a máquina de forma rápida e eficiente, conforme o tipo de programa que queremos desenvolver.

2.6.1 Lógica de Programação

A lógica de programação é a base para resolver problemas e criar soluções em tecnologia. Ela permite desenvolver algoritmos eficientes que podem ser transformados em código de programação.

Segundo Clemente (2023), a lógica de programação envolve um conjunto de regras e técnicas que programadores utilizam para criar e desenvolver programas de computador. É a capacidade de pensar de forma lógica e estruturada, quebrando problemas complexos em etapas simples. O objetivo é desenvolver algoritmos claros e eficientes que possam ser convertidos em código de programação.

Com a lógica de programação, os programadores podem resolver problemas ao dividi-los em partes menores e desenvolver algoritmos para cada parte. Além disso, algoritmos bem elaborados são mais eficientes, usam menos recursos computacionais, e são mais fáceis de manter e atualizar. Compreender a lógica de programação também facilita a aprendizagem de novas linguagens de programação, já que a lógica é universal.

A lógica de programação está intimamente ligada ao desenvolvimento de algoritmos. Um algoritmo é uma sequência finita de passos que, quando seguidos, resolvem um problema ou executam uma tarefa específica. A lógica é utilizada para projetar e melhorar algoritmos, garantindo que sejam eficazes e eficientes.

2.7 PHP

O PHP é uma linguagem de programação amplamente usada para criar páginas web dinâmicas. Ele se integra facilmente ao HTML, permitindo adicionar funcionalidades interativas aos sites.

De acordo com Basso e Miletto (2014), uma das vantagens do PHP é a sua integração com as marcações HTML, o que permite a incorporação de dinamismo nas páginas desenvolvidas com esta linguagem. Para que o servidor web identifique as partes que devem ser interpretadas como *scripts* em PHP, é necessário usar delimitadores (*tags*) de abertura (<?php, sendo o mais comum) e de fechamento (?>) nos trechos que contêm esse código, distinguindo-o, por exemplo, do HTML ou do JavaScript.

> O PHP é uma linguagem criada por Ramus Ledorf, em 1995, que explora a criação de scripts que são normalmente interpretados em um servidor Web no qual esses scripts estejam armazenados. O pré-requisito para que isso ocorra é que o servidor

> tenha o interpretador PHP devidamente configurado. No entanto, scripts PHP também podem ser executados localmente via linha de comando, mediante a presença de um interpretador. (Basso; Miletto, 2014, p.172)

"A interpretação dos scripts PHP possibilita a geração de códigos HTML, JavaScript, além de documentos PDF, XML, imagens ou textos, os quais podem ser enviados ao cliente ou simplesmente armazenados no servidor." (Basso; Miletto, 2014, p.172).

Ainda segundo os autores, uma característica única do PHP é que não é necessário declarar variáveis antes de utilizá-las, o que a distingue de muitas outras linguagens convencionais. Isso significa que uma variável será automaticamente criada quando receber seu primeiro valor. Embora não seja obrigatório inicializar uma variável no PHP, em algumas situações pode ser aconselhável fazê-lo, especialmente quando se deseja garantir que a variável não tenha sido utilizada anteriormente.

> O PHP se apresenta como uma linguagem dita fracamente tipada, sendo o tipo de dado da variável decidido em tempo de execução, dependendo do contexto em que ela está inserida. Em PHP não se tem, portanto, uma declaração formal do tipo de dado de uma determinada variável. São oito tipos primitivos de dados suportados pelo PHP: boolean, int, float, string, array, object, resource e null. Apresentaremos primeiramente as variáveis do tipo boolean (também chamado de tipo lógico), as quais podem assumir apenas dois valores: true ou false (ambas são case-insensitive). (Basso; Miletto, 2014, p.176)

"Além das variáveis definidas pelo programador, o PHP oferece variáveis nativas, que são chamadas de superglobais, pois estão disponíveis em qualquer local do código do programa, sem a necessidade de fazer sua inicialização ou definição." (Saraiva, 2018, p.67).

2.7.1 Orientação a Objetos

A orientação a objetos é um modelo de programação que organiza o código em torno de objetos, combinando dados e ações de forma lógica. Esse paradigma facilita a reutilização e a manutenção do código.

Conforme Soares (2013), uma das mudanças mais significativas no PHP foi a completa reestruturação da linguagem para se alinhar melhor ao paradigma de orientação a objetos (OO). Até a versão 4, o PHP tratava objetos como tipos primitivos, assim como inteiros ou strings. No entanto, com a introdução do PHP5, a manipulação de objetos foi redesenhada, resultando em melhor desempenho e uma ampla gama de novas funcionalidades. A principal diferença foi que, a partir dessa versão, os objetos passaram a ser tratados como entidades reais, com o uso de ponteiros para representá-los.

A orientação a objetos é um modelo de desenvolvimento que utiliza componentes reutilizáveis chamados objetos. Esses objetos possuem dados e executam ações que manipulam esses dados. Diferentemente da programação estruturada, que separa os dados das ações, a OO integra ambos de forma lógica e coesa, permitindo que os dados e seus manipuladores funcionem de maneira interdependente.

2.7.2 Laravel

O Laravel é um framework PHP amplamente utilizado, conhecido por sua eficiência e facilidade de uso. Ele oferece recursos avançados que agilizam o desenvolvimento de aplicações web.

Segundo Cardoso (2021), o Laravel surgiu em 2011 como uma resposta à necessidade de uma ferramenta mais abrangente para codificação em PHP. Este *framework* de aplicação web distingue-se por sua sintaxe expressiva e por fornecer uma estrutura robusta que serve como ponto de partida para o desenvolvimento de aplicativos. Estas características permitem aos desenvolvedores concentrarem-se na lógica de negócios, em vez de se preocuparem com detalhes minuciosos da codificação. O Laravel oferece uma gama de recursos avançados, tais como:

- testes de integração;
- injeção de dependências completa;
- agendamento de filas;
- testes de unidade;
- agendamento de tarefas;
- uma camada de abstração de banco de dados expressiva;
- entre outros.

Esses recursos tornam o Laravel uma ferramenta poderosa e eficaz para o desenvolvimento de aplicações web, facilitando a implementação de funcionalidades complexas e melhorando a produtividade do desenvolvedor.

> É recomendado o uso quando for utilizar aplicativos que precisam lidar com centenas de milhões de solicitações por mês, além da facilidade de escalonamento mais robustos, por meio de uso de plataformas como a denominada Laravel Vapor. Essa plataforma proporciona para o desenvolvedor executar o aplicativo em escala,

> quase ilimitada na mais recente tecnologia sem servidor da Amazon Web Services (AWS), plataforma de serviços de computação em nuvem. (Cardoso, 2021, p.39)

O autor também destaca que o Laravel pode servir como um *framework full stack*. Um *framework full stack* é aquele que pode ser utilizado tanto para o roteamento de requisições quanto para a aplicação em desenvolvimento. Além disso, é possível renderizar o *frontend* do projeto através de *templates* Blade ou utilizando uma tecnologia híbrida de aplicação de página única, como o Inertia.js. Esta é a forma mais comum de utilizar o *framework* Laravel.

2.7.2.1 Blade

O Blade é o sistema de templates nativo do Laravel, projetado para facilitar a criação de layouts dinâmicos. Ele combina simplicidade e eficiência, permitindo maior flexibilidade no desenvolvimento de views.

"O Laravel utiliza um sistema de *template* chamado Blade, que se diferencia de outras soluções PHP por não se restringir ao uso dessa linguagem em suas páginas. Além disso, no Blade, cada *view* é compilada e armazenada em cache até sofrer alguma alteração, deixando assim seus *templates* mais leves." (DevMedia, 2016, n.p)

> Blade é o mecanismo de template simples, mas poderoso, que está incluso no Laravel. Ao contrário de alguns mecanismos de template PHP, o Blade não o restringe de usar código PHP simples em seus templates. Na verdade, todos os *templates* Blade são compilados em código PHP simples e armazenados em cache até serem modificados, o que significa que o Blade adiciona essencialmente zero overhead ao seu aplicativo. (Laravel, 2024, n.p)

Ainda segundo a documentação, os *templates* Blade utilizam a extensão de arquivo .blade.php e geralmente são armazenados no diretório *resources/views*.

2.8 JAVASCRIPT

O JavaScript é uma linguagem fundamental para o desenvolvimento web, responsável por trazer interatividade e dinamismo às páginas. Ele complementa o HTML e o CSS, formando a base das tecnologias da web moderna.

Conforme Flanagan (2013) observa, JavaScript desempenha um papel central como a linguagem de programação dominante na web. É amplamente utilizado em praticamente todos os sites modernos, e todos os navegadores atuais, desde *desktops* até dispositivos móveis, estão equipados com interpretadores JavaScript. Essa ubiquidade faz do JavaScript a

linguagem de programação mais prevalente da história da web. Junto com HTML, que define o conteúdo das páginas da web, e CSS, que controla sua apresentação, JavaScript forma a tríade essencial de tecnologias que todo desenvolvedor web deve dominar para especificar o comportamento dinâmico das páginas.

> JavaScript nos permite fazer scripts do conteúdo HTML e da apresentação CSS de documentos em navegadores Web, mas também nos permite definir o comportamento desses documentos com rotinas de tratamento de evento. Uma rotina de tratamento de evento é uma função JavaScript que registramos no navegador e que este chama quando ocorre algum tipo de evento especificado. O evento de interesse pode ser um clique de mouse ou um pressionamento de tecla (ou, em um smartphone, pode ser um gesto de algum tipo, feito com dois dedos). Ou então, uma rotina de tratamento de evento pode ser ativada quando o navegador termina de carregar um documento, quando o usuário redimensiona a janela do navegador ou quando o usuário insere dados em um elemento de formulário HTML. (Flanagan, 2013, p.27).

Dando continuidade ao tema, Miletto e Silva (2014) complementam que dentro de uma página HTML, uma variedade de eventos pode ser desencadeada. Esses eventos podem incluir ações como clicar em um elemento com o mouse, inserir informações em um formulário usando o teclado ou até mesmo sair da página. Todos esses eventos são considerados interações e podem ocorrer em várias tags HTML. Os eventos são associados às tags HTML e são ativados conforme as ações do usuário, podendo ser acionados ou não, dependendo da situação.

Os autores também mencionam que o JavaScript permite a manipulação de eventos do mouse, como destacado no Quadro 4.

Quadro 4 - Eventos de mouse

Nome	**Conceito**
onClick	O usuário clica em um elemento.
onDblClick	O usuário clica duas vezes em um elemento.
onMouseDown	O usuário pressiona o botão do mouse sobre um elemento.
onMouseMove	O ponteiro do mouse está em movimento sobre o elemento.
onMouseOver	O ponteiro do mouse está sobre o elemento.
onMouseOut	O usuário move o ponteiro do mouse para fora do elemento.
onMouseUp	O usuário solta o botão do mouse sobre um elemento.

Fonte: Elaborado a partir de Miletto e Silva (2014, p.113).

> Uma característica importante da linguagem JavaScript é que ela não apresenta tipos de dados, isto é, qualquer variável definida é do tipo variante. Isso quer dizer que o tipo de dados é definido de acordo com a informação armazenada naquele momento. O tipo de dados de determinada variável também pode ser modificado ao longo da execução da aplicação, conforme seu conteúdo é alterado. (Oliveira; Zanetti, 2020, p.47)

Oliveira e Zanetti (2020) destacam que os navegadores oferecem ferramentas que permitem monitorar a execução das instruções JavaScript e emitir comandos diretamente para avaliar seu funcionamento.

2.9 BANCO DE DADOS

Os bancos de dados são essenciais para armazenar, organizar e acessar informações de forma eficiente. Eles permitem que grandes volumes de dados sejam gerenciados com segurança e facilidade.

“Banco de Dados (ou Base de Dados) é uma coleção de dados, organizada de modo a facilitar o armazenamento e acesso para programas que utilizem esses dados.” (Komatsu, 2011, p.3).

“Os bancos de dados surgiram principalmente em função da grande necessidade de armazenar dados de forma permanente e organizada. É importante salientar que o volume de dados que tem surgido é cada vez maior. ” (Vida, 2020, p.11).

Segundo Silberschatz (2020), os sistemas de banco de dados são criados para lidar com grandes quantidades de dados. Isso inclui definir como armazenar as informações e criar maneiras de acessá-las. Além disso, é importante que esses sistemas mantenham a segurança dos dados, mesmo se houver problemas no sistema ou se alguém tentar acessá-los sem permissão. Se várias pessoas estiverem compartilhando os mesmos dados, o sistema precisa garantir que não ocorram resultados estranhos ou inesperados.

Complementando essa visão, Vida (2020) explica que as consultas são realizadas por meio de um programa de aplicação, que é utilizado pelos usuários para enviar solicitações de consulta e visualizar os resultados. Esse programa de aplicação pode ser qualquer tipo de software que interaja com um banco de dados específico. Ao solicitar uma conexão, o programa envia informações de autenticação, como nome de usuário e senha, ao sistema de gerenciamento de banco de dados (SGBD). Após estabelecer a conexão, o programa pode solicitar consultas aos dados armazenados. Portanto, o SGBD é responsável por executar as consultas e fornecer os resultados ao programa de aplicação. Além disso, os SGBDs possuem mecanismos para minimizar a perda de dados em casos de falhas de *software* ou *hardware*, como medidas de segurança contra corrupção de arquivos.

Macêdo (2022) reforça que, para garantir que um SGBD esteja desempenhando adequadamente suas funções, é crucial que ele possua algumas características essenciais. Estas características fundamentais incluem:

- controle de redundâncias;
- compartilhamento de dados;
- controle de acesso;
- interfaceamento;
- esquematização;
- controle de integridade;
- *backups*.

2.9.1 Banco de Dados Relacional

Os bancos de dados relacionais são uma das formas mais comuns de organizar e armazenar informações. Eles estruturam os dados em tabelas, facilitando o acesso e a gestão de grandes volumes de informações.

Conforme Alves (2014) destaca, a maioria dos sistemas de gerenciamento de bancos de dados utilizados atualmente são do tipo relacional. Nesse tipo de banco de dados, os dados são organizados em tabelas, conhecidas como relações, compostas por linhas e colunas. Dessa forma, as tabelas se assemelham a conjuntos de elementos ou objetos, pois organizam informações relacionadas a um mesmo tema de maneira estruturada.

> A abordagem relacional representa uma forma de descrever o banco de dados por meio de conceitos matemáticos simples: a Teoria dos Conjuntos. Voltada principalmente a melhorar a visão dos dados pelos usuários, essa abordagem faz com que os usuários vejam o banco de dados como um conjunto de tabelas bidimensionais, originadas em linhas e colunas. (Machado, 2020, p.38)

"Um banco de dados relacional permite que se tenham informações divididas entre várias tabelas de dados. Porém, certas informações de uma tabela são obtidas a partir de outras." (Alves, 2014, p.22)

2.9.2 Linguagem SQL

A linguagem SQL é amplamente utilizada para gerenciar bancos de dados, possibilitando a criação, consulta e organização de dados de forma eficiente.

Conforme Cardoso (2013), a linguagem SQL é declarativa, focalizando o resultado desejado em vez dos detalhes do processamento. Baseada em álgebra e cálculo relacional,

apresenta uma sintaxe simples e amigável. Além de manipular dados, a SQL possibilita a definição da estrutura e a aplicação de regras e restrições de integridade. Para fornecer uma variedade de recursos, a SQL é subdividida em várias partes, cada uma com uma função específica, conforme Quadro 5.

Quadro 5 - Tipos de linguagem SQL

Nome	Conceito
DDL	Linguagem de definição de dados que permite determinar o esquema do banco de dados, bem como alterá-lo e excluí-lo, e trabalha com os metadados.
DML	Linguagem de manipulação de dados que permite a manipulação dos dados, ou seja, a inclusão, alteração e exclusão de dados.
DCL	Linguagem de controle dos dados que permite controlar a licença e a autorização de acesso dos usuários para com os dados.
DTL	Linguagem de transação de dados, que oferece comandos para trabalhar com as transações.
DQL	Linguagem de consulta de dados), que proporciona a consulta de dados, parte importantíssima dentro de um sistema de banco de dados.

Fonte: Elaborado a partir de Cardoso (2013).

"Uma das principais partes da linguagem SQL é a DML, que inclui os comandos que permitem a manipulação dos dados. Quando dizemos manipulação, estamos falando sobre inserção, atualização ou modificação de dados e exclusão de dados em uma tabela." (CARDOSO, 2013, p.20)

Segundo DevMedia (2016), os principais comandos básicos de SQL estão exemplificados no Quadro 6.

Quadro 6 - Comandos SQL

Nome	Conceito
Insert	Comando para inclusão de registros no banco, de dados
Update	Comando para atualizar registros do banco de dados
Delete	Comando para apagar registros do banco de dados.
Select	Comando para selecionar registros do banco de dados.

Fonte: Elaborado a partir de DevMedia (2016).

Ainda segundo DevMedia, Os comandos SQL oferecem uma linguagem clara e direta para a manipulação de dados em um Sistema de Gerenciamento de Banco de Dados (SGBD).

2.9.3 SQL Power Architect

O SQL Power Architect é uma ferramenta voltada para o design e análise de bancos de dados. Ele oferece recursos específicos para arquitetos de dados, otimizando processos e aumentando a eficiência no gerenciamento de informações.

Conforme o Best of BI (2024), a ferramenta SQL Power Architect foi criada especialmente para projetistas de repositórios de dados, proporcionando diversas funcionalidades exclusivas para arquitetos dessa área. Com essa ferramenta, os usuários conseguem fazer engenharia reversa de bancos de dados já existentes, realizar análise de dados nas fontes originais e gerar automaticamente metadados para processos de extração, transformação e carregamento de dados. Algumas de suas características incluem:

- acessar bancos de dados de origem via JDBC;
- conectar-se a vários bancos de dados de origem simultânea;
- comparar modelos de dados e estruturas de banco de dados e identificar discrepâncias;
- gerar relatórios de mapeamento visual de origem para destino;
- entre outros.

Ainda segundo Best of BI (2024), “o SQL Power Architect fornecerá uma visão completa de todas as estruturas de banco de dados necessárias e agilizará todos os aspectos do design do seu data warehouse.”

2.9.4 PostgreSQL

O PostgreSQL é um sistema de banco de dados de código aberto, conhecido por sua confiabilidade e conformidade com os padrões SQL. Ele fornece recursos modernos para atender a diferentes demandas de gerenciamento de dados.

De acordo com a documentação de software de PostgreSQL (2024), o PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional. Este sistema foi inovador em muitos conceitos que só foram adotados por sistemas de banco de dados comerciais posteriormente. O PostgreSQL é uma versão de código aberto do software original desenvolvido pela Universidade da Califórnia, em Berkeley. Ele adere amplamente ao padrão SQL e oferece uma ampla gama de recursos modernos, como:

- consultas complexas;
- chaves estrangeiras;
- gatilhos;

- visualizações atualizáveis;
- integridade transacional;
- controle de simultaneidade multiversão;
- entre outros.

Além disso, o PostgreSQL permite a extensão por parte do usuário, possibilitando a adição de novos tipos de dados, como:

- tipos de dados;
- funções;
- operadores;
- funções agregadas;
- métodos de índice;
- linguagens procedurais.

De acordo com Microsoft (2024), PostgreSQL é compatível com diversas linguagens de programação e protocolos populares, como C, C++, Go, Perl, Python, Java, .Net, Ruby, ODBC e Tcl. Isso permite que os usuários trabalhem na linguagem de sua preferência, evitando conflitos de sistema.

Abaixo estão alguns dos tipos de dados mais comuns do PostgreSQL, conforme mostrado no Quadro 7.

Quadro 7 - Tipos de dados PostgreSQL

Nome	Conceito
Booleano	O tipo de dados Booleano é projetado para expressar valores de dois estados, como valores verdadeiro/falso, ativado/desativado, sim/não e nulo. Normalmente, você usaria esse tipo de dados para avaliar instruções condicionais. O fluxo de controle pode depender do resultado verdadeiro ou falso, como ao usar a expressão CASE do PostgreSQL, com várias ações resultantes da avaliação.
Caractere	Esse tipo de dados consiste em sequências de caracteres, como letras ou números, e é usado para armazenar valores de texto. Tipos de dados de caractere e tipos de string podem ocorrer como um comprimento fixo, conhecido como char, ou comprimentos variáveis, conhecidos como varchar e long varchar. O comprimento selecionado pelo usuário afeta a validação da entrada.
Datas e horas	O tipo de dados de data e hora é usado para indicar datas, horas e intervalos temporais. O tipo de dados de carimbo de data/hora do PostgreSQL é preciso em microssegundos e oferece aos usuários a opção de armazenar dados de hora e data com ou sem informações de fuso horário anexadas.
Numérico	Os tipos de dados numéricos vêm em duas formas: exato e aproximado. Os tipos de dados numéricos exatos contêm tipos de dados inteiros e tipos de dados decimais. Os tipos de dados aproximados, por outro lado, contêm tipos de dados de ponto flutuante – por exemplo, números inteiros de 2, 4 e 8 bytes, números de ponto flutuante de 4 e 8 bytes e decimais de precisão selecionáveis.

Fonte: Elaborado a partir de Microsoft (2024).

Ainda segundo a Microsoft, cada tipo de dado possui um propósito específico, como habilitar a pesquisa de texto completo ou armazenar datas e horas. Ao criar uma tabela, os usuários escolhem um tipo de dado apropriado para cada coluna, determinando o tipo de informação que será armazenada em cada campo.

2.10 SYSTEM USABILITY SCALE

O System Usability Scale (SUS) é uma ferramenta prática e amplamente utilizada para medir a usabilidade de sistemas. Ele ajuda a compreender a experiência do usuário por meio de avaliações simples e objetivas.

De acordo com Soegaard (2023), o SUS (*System Usability Scale*) é uma ferramenta em formato de questionário criada para avaliar a usabilidade percebida de um sistema. Composto por dez afirmações sobre a experiência do usuário com um produto específico, o questionário utiliza uma escala Likert de cinco pontos. Os participantes indicam seu grau de concordância ou discordância com cada afirmação, permitindo a coleta de dados quantitativos que auxiliam na avaliação da facilidade de uso do sistema do ponto de vista do usuário. Alguns de seus benefícios são:

- melhorar a avaliação centrada do usuário;
- oferecer velocidade e simplicidade;
- facilitar o design iterativo;
- apoiar a tomada de decisões;
- melhorar a colaboração;
- promover a consistência;
- incentivar o feedback do usuário;
- entre outros.

Ainda segundo Soegaard (2023), “uma pontuação SUS pode variar de 0 a 100, com pontuações mais altas indicando melhor usabilidade. Normalmente, uma pontuação acima de 70 é considerada boa, enquanto uma pontuação acima de 85 é excelente.”

3. METODOLOGIA DE PESQUISA

O presente trabalho de conclusão de curso caracteriza-se como pesquisa aplicada e descritiva, onde o objetivo do projeto é desenvolver um protótipo de sistema de gestão para abrigos utilizados em situações de crises climáticas no município de Rio do Sul/SC.

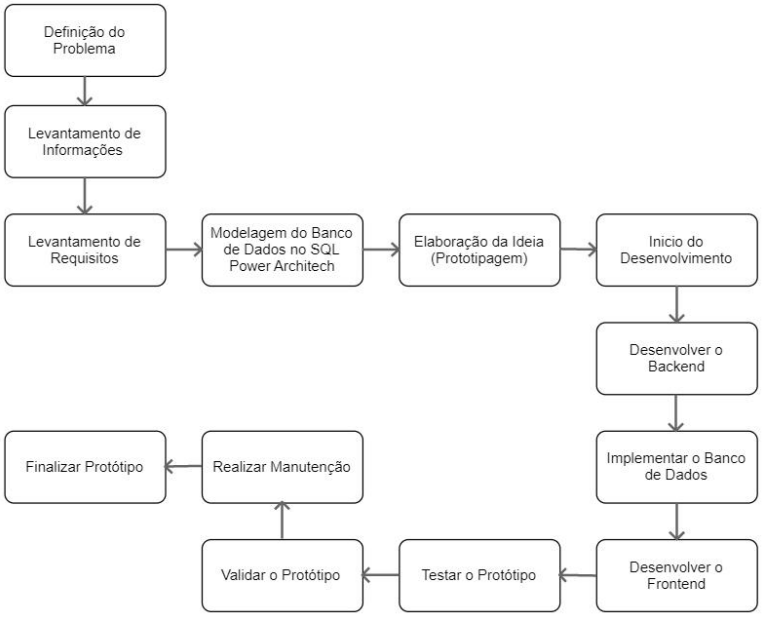
Foi realizada uma pesquisa de campo qualitativa com a Secretaria de Assistência Social do município, com o objetivo de responder algumas questões: É possível realizar a gestão dos abrigos disponíveis? É possível organizar a distribuição dos recursos durante as crises climáticas? É possível contabilizar os abrigados de forma mais eficiente?

Para o trabalho proposto, realizou-se um estudo teórico sobre as tecnologias e processos essenciais para o desenvolvimento do projeto. Este estudo inclui uma revisão da literatura, que aprofundou os conceitos e detalhou as tecnologias utilizadas.

Através dos resultados obtidos pela pesquisa com os usuários e *stakeholders*, decidiu-se que o protótipo de gestão de abrigos será desenvolvido em PHP na versão 8.2.4, utilizando o *framework* Laravel na versão 11.23. Esse *framework* oferece diversos recursos e funcionalidades avançadas que tornam a aplicação mais robusta e eficiente. Além disso, será utilizado JavaScript, com foco em dinamizar a página.

Com base nas informações, elaborou-se um guia para o desenvolvimento do protótipo, conforme fluxograma apresentado na Figura 2.

Figura 2 - Fluxograma do processo de desenvolvimento do protótipo



Fonte: Acervo do Autor (2024).

O desenvolvimento do sistema de gestão de abrigos seguiu uma série de etapas para garantir que todas as necessidades e desafios fossem abordados adequadamente. A seguir, cada etapa do processo é detalhada, conforme ilustrado na Figura 2.

Primeiramente, a definição do problema foi realizada para identificar as dificuldades enfrentadas pela Secretaria de Assistência Social e Defesa Civil na gestão de abrigos, famílias e recursos durante as crises climáticas. Observou-se a falta de um sistema integrado para organizar e centralizar informações, o que permitiria uma coordenação mais eficiente das ações e recursos em situações de emergência. Isso ressalta a importância de desenvolver uma solução específica para essas necessidades.

Em seguida, foi realizado o levantamento de informações, que consistiu na coleta de dados relevantes para o projeto, incluindo requisitos dos usuários, práticas atualmente adotadas e os principais desafios no gerenciamento de abrigos. Com essas informações, avançou-se para o levantamento de requisitos, no qual foram especificadas as funcionalidades essenciais para o sistema, como o cadastro de eventos, controle de ocupação de abrigos e gestão de recursos.

Com os requisitos definidos, iniciou-se a modelagem do banco de dados no SQL Power Architect na versão 1.0.9. Nessa fase, foi criado um modelo visual estruturado para o banco de dados, detalhando as tabelas e os relacionamentos necessários para armazenar informações sobre abrigos, eventos, famílias, pets e recursos. Esse modelo serviu de base para as próximas etapas do desenvolvimento.

A elaboração da ideia, no caso a prototipagem apresentada na história de usuários, foi a fase seguinte, onde um protótipo inicial foi desenvolvido para simular a interface e o fluxo do sistema. Esse protótipo permite visualizar as principais telas e funcionalidades antes da implementação completa, ajudando na verificação dos requisitos.

Com a prototipagem concluída, iniciou-se o desenvolvimento do sistema, que foi dividido em três partes principais: *backend*, banco de dados e *frontend*. Na fase de desenvolvimento do *backend*, foi implementada a lógica de negócios utilizando o Laravel, com funcionalidades como o gerenciamento de abrigos, controle de eventos e operações de recursos.

Em paralelo, ocorreu a implementação do banco de dados, onde o modelo previamente definido foi configurado e criado no ambiente real utilizando o PostgreSQL na versão 7.8, garantindo que ele suportasse todas as funcionalidades planejadas. A etapa de desenvolvimento do *frontend* completou essa fase, com a criação da interface do usuário utilizando Blade na versão 11.23, TailwindCSS na versão 3.4.11 e JavaScript para dinamismo.

Foram desenvolvidas telas para cadastro e visualização de informações de abrigos, famílias, pets e controle de recursos, proporcionando uma experiência de uso interativa e responsiva.

Após o desenvolvimento, entrou-se na fase de testes do protótipo, onde foram realizados testes iniciais para assegurar que cada funcionalidade atendesse aos requisitos especificados e que o sistema operasse corretamente. Em seguida, o protótipo foi apresentado para validação com os usuários na Secretaria de Assistência Social em outubro de 2024, permitindo que a equipe fornecesse *feedbacks* para possíveis ajustes e melhorias.

Com base nas sugestões e necessidades identificadas, foi realizada a manutenção do sistema, onde foram implementadas algumas melhorias. Esse processo garantiu ajustes importantes para melhor atender às necessidades dos usuários. No entanto, alguns pontos ainda serão desenvolvidos em trabalhos futuros, visando aprimorar ainda mais a funcionalidade e a eficiência do sistema.

Por fim, a primeira versão do protótipo foi concluída, com as validações previstas e ajustes iniciais. Embora ainda restem melhorias a serem implementadas, a versão já pode ser testada e usada como base para gerenciar abrigos em crises climáticas. Essa solução integrada permite uma gestão mais organizada dos abrigos, com controle detalhado de ocupação, famílias, recursos e atividades necessárias durante crises, promovendo uma resposta mais rápida e coordenada para as equipes de assistência.

4. PROTÓTIPO DE SISTEMA DE GESTÃO PARA ABRIGOS UTILIZADOS EM SITUAÇÕES DE CRISES CLIMÁTICAS NO MUNICÍPIO DE RIO DO SUL/SC

Este capítulo aborda os aspectos técnicos relacionados ao processo de análise e implementação do protótipo.

4.1 ANÁLISE

Este capítulo detalha os requisitos funcionais e não funcionais, além das regras de negócio que orientam o desenvolvimento do protótipo. Também oferece uma visão geral do sistema, destacando sua finalidade e benefícios para o usuário. As informações levantadas durante a fase de desenvolvimento serão expostas, incluindo os requisitos, as regras de negócio e as funcionalidades do sistema, que serão ilustrados com diagramas e protótipos.

4.1.1 Visão Geral do Protótipo

O protótipo proposto tem como objetivo principal auxiliar a Secretaria de Assistência Social e a Defesa Civil no gerenciamento dos abrigos de forma eficaz durante as crises climáticas que ocorrem no município de Rio do Sul/SC. Através desta aplicação, as secretarias poderão monitorar e registrar as ocorrências de eventos climáticos que demandem o uso de abrigos, mantendo um histórico detalhado das situações que envolvem o alojamento de pessoas e pets. Com base nesses registros, o sistema facilitará a organização e o direcionamento dos abrigos apropriados para cada evento

Além do gerenciamento de abrigos, a aplicação também proporcionará uma gestão completa das famílias acolhidas, permitindo que os responsáveis pela gestão dos abrigos registrem todos os membros do grupo familiar, especificando dados como idade, parentesco e se possuem pets, detalhando o tipo, porte e quantidade dos animais acolhidos. Esse controle preciso facilitará a organização interna dos abrigos e permitirá que a assistência oferecida seja personalizada de acordo com as necessidades específicas de cada grupo.

No que se refere à gestão de recursos, o sistema permitirá o controle e a distribuição detalhada dos materiais e insumos disponíveis, sejam eles provenientes de doações ou de outros recursos da secretaria. A aplicação possibilitará o registro de entrada e saída desses recursos, garantindo que a distribuição seja feita de forma justa e eficiente, de acordo com a demanda de cada evento e abrigo.

A principal vantagem da utilização desta aplicação é a capacidade de centralizar todas as informações em um único sistema, permitindo à Secretaria de Assistência Social e a Defesa Civil obter uma visão global e precisa das situações dos abrigos durante uma crise climática.

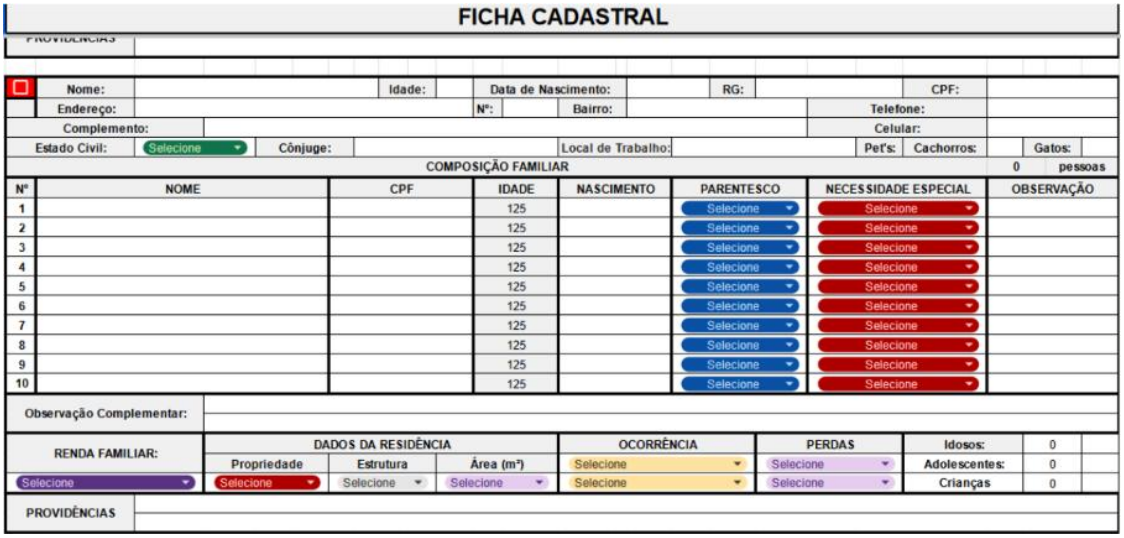
Além disso, a aplicação permitirá a coleta e armazenamento de dados reais e detalhados sobre o número de crianças, adultos e idosos afetados, quantas pessoas tiveram seus bens perdidos, quantos recursos foram utilizados, entre outros detalhes importantes. Essas informações poderão ser utilizadas tanto para prestação de contas quanto para a criação de relatórios que auxiliem na organização e planejamento para futuras crises, possibilitando a elaboração planos de ação e a criação de estratégias preventivas baseadas em dados reais coletados em eventos anteriores.

4.1.2 Levantamento de Informações

Para o desenvolvimento deste trabalho, foi elaborado e enviado um questionário à Secretaria de Assistência Social entre os meses de junho e julho de 2024, conforme apresentado no Apêndice A, com o intuito de compreender o processo de gestão dos abrigos. O questionário abordou temas como o controle das necessidades das pessoas abrigadas, a contabilização do número de indivíduos, famílias e animais de estimação alojados, além da gestão do estoque de recursos recebidos, entre outros aspectos essenciais para a administração eficiente dos abrigos. O questionário foi formalizado e enviado via e-mail. Adicionalmente, a Secretaria demonstrou o método atual de controle, que é feito por meio de planilhas eletrônicas, conforme ilustrado nas Figuras 3 e 4.

A Figura 3 apresenta a ficha cadastral da composição familiar, utilizada atualmente pela Secretaria de Assistência Social para controlar as famílias, integrantes e pets que se alojam nos abrigos. Essa ficha coleta informações do responsável pela família, dos membros e pets, além de dados sobre a residência da família e necessidades especiais dos integrantes. Sua função é centralizar todas as informações relevantes sobre as pessoas abrigadas, facilitando o controle de ocupação dos abrigos, o monitoramento de necessidades específicas e o registro de qualquer ocorrência significativa durante o período de crise. Trata-se de um recurso essencial para organizar e manter um histórico detalhado dos atendimentos em cada abrigo.

Figura 3 - Ficha cadastral de composição familiar



FICHA CADASTRAL

PROVIDÊNCIAS

Nome:		Idade:		Data de Nascimento:		RG:		CPF:	
Endereço:				Nº:		Bairro:		Telefone:	
Complemento:								Celular:	
Estado Civil:	Selecione	Cônjuge:				Local de Trabalho:		Pet's: Cachorros:	Gatos:

COMPOSIÇÃO FAMILIAR — 0 pessoas

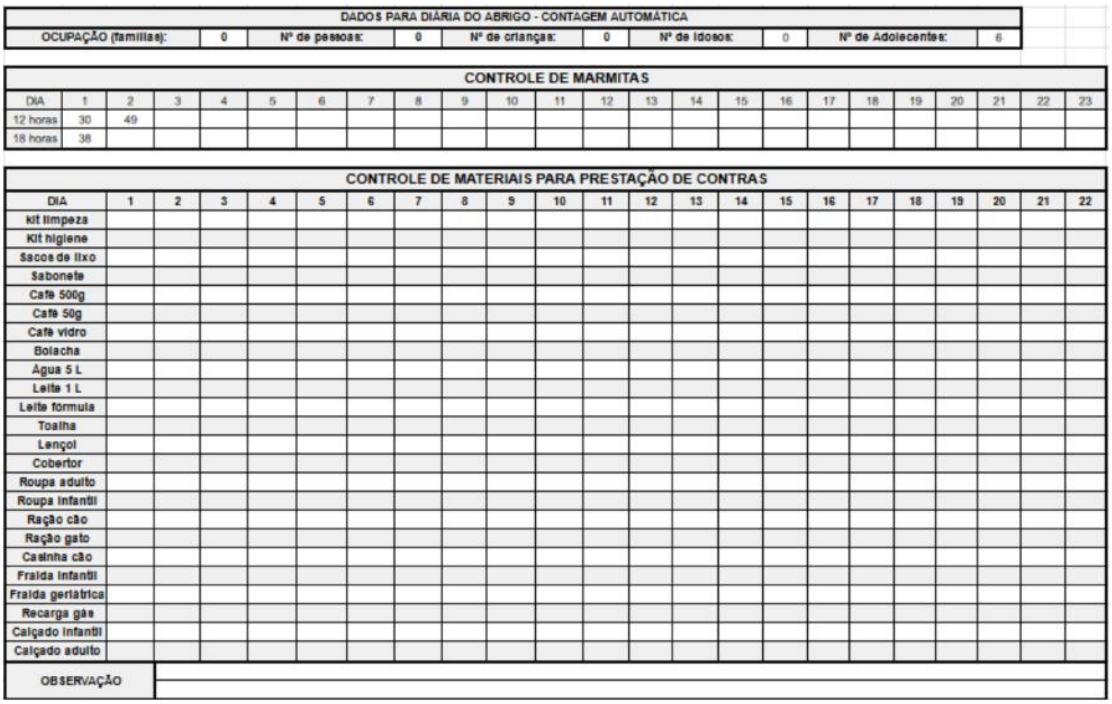
Nº	NOME	CPF	IDADE	NASCIMENTO	PARENTESCO	NECESSIDADE ESPECIAL	OBSERVAÇÃO
1			125		Selecione	Selecione	
2			125		Selecione	Selecione	
3			125		Selecione	Selecione	
4			125		Selecione	Selecione	
5			125		Selecione	Selecione	
6			125		Selecione	Selecione	
7			125		Selecione	Selecione	
8			125		Selecione	Selecione	
9			125		Selecione	Selecione	
10			125		Selecione	Selecione	

Observação Complementar:								
RENDA FAMILIAR:	DADOS DA RESIDÊNCIA			OCORRÊNCIA	PERDAS	Idosos:	0	
	Propriedade	Estrutura	Área (m²)	Selecione	Selecione	Adolescentes:	0	
Selecione	Selecione	Selecione	Selecione	Selecione	Selecione	Crianças	0	
PROVIDÊNCIAS								

Fonte: Secretaria de Assistência Social (2024).

A Figura 4 exibe uma planilha de controle diário utilizado nos abrigos após sua abertura, onde são monitorados diariamente a ocupação do abrigo e o consumo de recursos, proporcionando uma gestão detalhada e facilitando a prestação de contas.

Figura 4 - Controle de demandas do abrigo



DADOS PARA DIÁRIA DO ABRIGO - CONTAGEM AUTOMÁTICA

OCUPAÇÃO (famílias):	0	Nº de pessoas:	0	Nº de crianças:	0	Nº de idosos:	0	Nº de Adolecentes:	6

CONTROLE DE MARMITAS

DIA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
12 horas	30	49																					
18 horas	38																						

CONTROLE DE MATERIAIS PARA PRESTAÇÃO DE CONTRAS

DIA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
kit limpeza																						
Kit higiene																						
Sacos de lixo																						
Sabonete																						
Café 500g																						
Café 50g																						
Café vidro																						
Bolacha																						
Água 5 L																						
Leite 1 L																						
Leite fórmula																						
Toalha																						
Lençol																						
Cobertor																						
Roupa adulto																						
Roupa infantil																						
Ração cão																						
Ração gato																						
Caminha cão																						
Fralda infantil																						
Fralda geriátrica																						
Recarga gás																						
Calçado infantil																						
Calçado adulto																						

OBSERVAÇÃO	

Fonte: Secretaria de Assistência Social (2024).

4.1.3 Requisitos

Nesta seção, estão listados todos os requisitos funcionais, requisitos não funcionais e regras de negócio identificados para o desenvolvimento do protótipo do sistema de gestão de abrigos em crises climáticas. Esses elementos são fundamentais para assegurar que todas as operações sejam executadas corretamente, garantindo a conformidade com os objetivos e padrões estabelecidos, além de garantir que a aplicação funcione conforme as necessidades do negócio. Para isso, os requisitos devem ser desenvolvidos conforme planejados e listados.

No Quadro 8, são apresentados os requisitos funcionais para o desenvolvimento do protótipo e suas respectivas regras de negócio, que orientarão o comportamento do sistema desde o processo de autenticação de usuários até a gestão de recursos e doações nos abrigos.

Quadro 8 - Requisitos Funcionais

Número	Nome	Descrição	Regras de Negócio
RF01	Realizar *Login*	O sistema deverá conter uma tela de *login*.	Não se aplica.
RF02	Realizar *Logout*	O sistema deverá conter uma opção de *logout*, para o usuário encerrar sua sessão na plataforma.	Não se aplica.
RF03	Cadastrar Usuário	O sistema deverá conter uma tela de cadastro de usuários, onde será possível cadastrar o usuário que terá acesso ao sistema, informando o nome, e-mail e senha.	RN01
RF04	Cadastrar Pessoa	O sistema deverá conter uma tela de cadastro de pessoas, onde será informado o nome, CPF, data de nascimento, idade e telefone.	RN02
RF05	Cadastrar Produto	O sistema deverá conter uma tela de cadastro de produtos, onde será informado o nome, validade e categoria.	Não se aplica.
RF06	Cadastrar Família	O sistema deverá conter uma tela para cadastro de famílias, onde será informado o nome da família, responsável, contato de emergência, cep, uf, cidade, logradouro, bairro, número, complemento, renda familiar e necessidades. Além disso, deve ser possível vincular os integrantes do grupo familiar, especificando seu parentesco, e também vincular os pets pertencentes ao grupo familiar, informando o tipo, porte e quantidade de cada um.	RN03
RF07	Gerenciar Família	O sistema deverá conter uma tela de visualização da família, onde será possível vincular os integrantes e pets do grupo familiar.	RFN04
RF08	Cadastrar Evento	O sistema deverá conter uma tela de cadastro de eventos, onde será informada a data de início, data de fim e observações. Além disso, deve ser possível vincular os abrigos disponíveis no evento.	RN05, RN06
RF09	Gerenciar Evento	O sistema deverá conter uma tela de visualização do evento, onde será possível vincular os abrigos disponíveis e gerenciá-los.	RN06
RF10	Gerenciar Abrigos do Evento	O sistema deverá conter uma tela de visualização do abrigo do evento, onde será possível vincular as famílias abrigadas, informando a data de entrada e data de saída.	RN07, RN08

RF11	Cadastrar Abrigo	O sistema deverá conter uma tela de cadastro de abrigos, onde será informado o nome do abrigo, cep, uf, cidade, logradouro, bairro, número, complemento, capacidade máxima, se está ativo e se pode exceder a capacidade.	Não se aplica.
RF12	Cadastrar Depósito	O sistema deverá conter uma tela de cadastro de depósitos, onde será informado o nome do depósito, cep, uf, cidade, logradouro, bairro, número, complemento e abrigo relacionado.	RN09
RF13	Gerenciar Depósitos	O sistema deverá conter uma tela de visualização do depósito, onde será possível visualizar os recursos em estoque.	RN10
RF14	Cadastrar Lançamentos	O sistema deverá conter uma tela de cadastro de lançamentos, onde será possível informar a operação, depósito de origem, depósito destino, produtos, quantidades, se é doação e o doador.	RN11
RF15	Consultar Estoque	O sistema deverá conter uma tela de estoque, onde será possível visualizar os recursos disponíveis.	RN10
RF16	Consultar Doações	O sistema deverá conter uma tela de doações, onde será possível visualizar as doações recebidas.	RN11

Fonte: Acervo do Autor (2024).

O Quadro 9 apresenta os requisitos funcionais opcionais que poderão ser desenvolvidos e implementados em versões futuras do protótipo. Esses requisitos não integram a versão inicial da aplicação.

Quadro 9 - Requisitos Funcionais Opcionais

Número	**Nome**	**Descrição**
RF17	*Dashboard* Inicial	O sistema deverá exibir no menu inicial informações gerais dos abrigos, incluindo a quantidade de pessoas e pets abrigados e o tempo médio de permanência das famílias.
RF18	Relatórios	O sistema deverá gerar relatórios abrangendo informações sobre abrigos, famílias, integrantes, pets e recursos.
RF19	Registrar Auditorias	O sistema deverá registrar auditorias das alterações realizadas nos registros do sistema, promovendo segurança e transparência.
RF20	Portal da Transparência	O sistema deverá contar com um portal de transparência que apresente informações gerais sobre os abrigos, famílias, integrantes, pets e recursos.
RF21	Desativar Registros	O sistema deverá permitir a desativação dos registros em vez de sua exclusão permanente.
RF22	Administrar Permissões	O sistema deverá administrar as permissões de cada usuário com acesso à plataforma.
RF23	Revogação de Dados Pessoais	O sistema deverá oferecer a funcionalidade de revogação de dados pessoais.

Fonte: Acervo do Autor (2024).

O Quadro 10 apresenta os requisitos não funcionais, que abrangem pontos gerais da aplicação, envolvendo questões como desempenho, usabilidade e segurança.

Quadro 10 - Requisitos Não Funcionais

Número	Descrição
RNF01	O sistema será desenvolvido utilizando arquitetura web.
RNF02	O sistema será desenvolvido utilizando a linguagem de programação PHP.
RNF03	O sistema utilizará o *framework* Laravel para o desenvolvimento.
RNF04	Será utilizado o banco de dados PostgreSQL para o armazenamento das informações.
RNF05	A interface do usuário deve ser intuitiva e fácil de navegar, proporcionando uma experiência agradável aos usuários.
RNF06	As mensagens de erro devem ser claras e informativas, auxiliando o usuário em caso de problemas.
RNF07	O código deve ser organizado seguindo a arquitetura MVC (*Model-View-Controller*) do Laravel, permitindo uma estrutura lógica e modular.
RNF08	O sistema deverá garantir que apenas usuários autorizados tenham acesso às funcionalidades.
RNF09	O sistema deverá ser responsivo com dispositivos móveis
RNF10	O sistema deverá estar de acordo com a Lei Geral de Proteção de Dados (LGPD)

Fonte: Acervo do Autor (2024).

No Quadro 11, detalha-se as regras de negócio do protótipo, estabelecendo os parâmetros e comportamentos essenciais para o funcionamento adequado da aplicação.

Quadro 11 - Regras de Negócio

Número	Descrição
RN01	O e-mail informado no cadastro de usuários deve ser válido e único.
RN02	O CPF informado no cadastro de pessoas deve ser válido e único.
RN03	O responsável pela família deve ser uma pessoa maior de idade.
RN04	O integrante não pode ser adicionado mais de uma vez na mesma família.
RN05	A data de fim do evento não pode ser anterior a data de início do evento.
RN06	O abrigo não pode ser adicionado mais de uma vez no mesmo evento.
RN07	A família só pode ser associada em abrigos ativos e que não tiveram sua capacidade máxima excedida.
RN08	A data de saída da família não deve ser anterior à data de entrada no abrigo.
RN09	O depósito deve estar associado a um único abrigo.
RN10	O estoque deve ser atualizado em tempo real conforme as movimentações de produtos
RN11	Todas as movimentações de estoque, incluindo doações, devem ser registradas detalhadamente, informando a operação realizada, depósitos envolvidos, produtos, quantidades e dados do doador quando aplicável.
RN12	Não é permitido realizar movimentações de estoque que resultem em quantidades negativas de recursos.

Fonte: Acervo do Autor (2024).

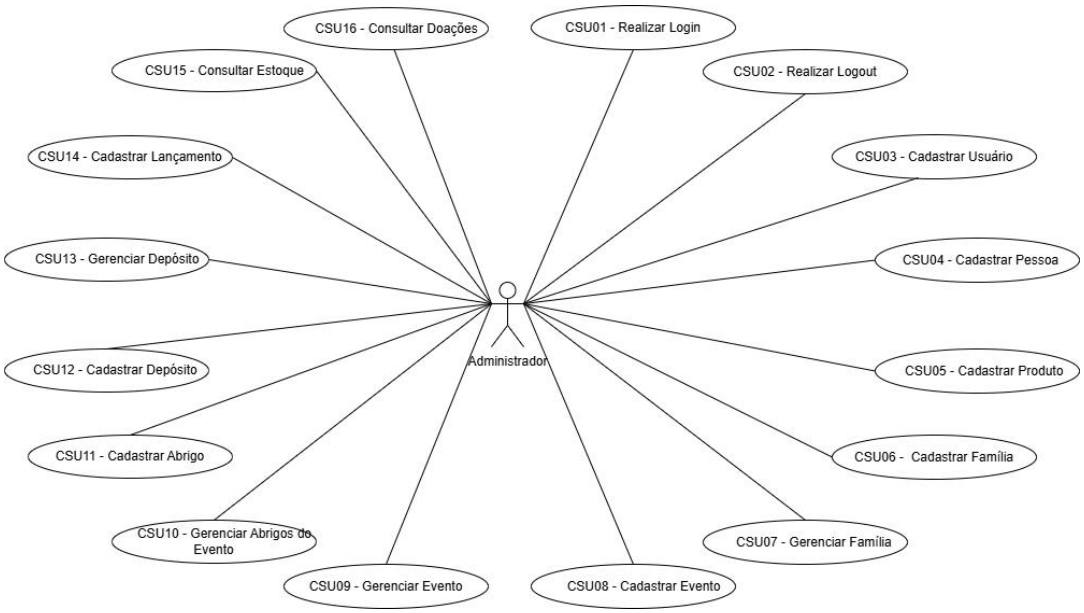
4.1.4 Diagramas

Para tornar a leitura dos requisitos mencionados no quadro 10 mais acessível e auxiliar no desenvolvimento da aplicação, foi elaborado um diagrama de casos de uso. Este diagrama ilustra os recursos disponíveis para os usuários, detalhando os casos de uso do administrador e do assistente. Ao representar visualmente as interações entre esses atores e as funcionalidades

do sistema, o diagrama fornece uma compreensão clara das necessidades e dos requisitos, facilitando tanto o planejamento quanto a implementação das funcionalidades previstas.

Conforme a Figura 5, para o usuário do tipo Administrador, estão disponíveis todas as funcionalidades do sistema. O Administrador tem acesso aos cadastros de usuários, pessoas, produtos, famílias, abrigos, eventos e depósitos. Além disso, pode gerenciar e visualizar lançamentos, estoque e doações. A opção "Meu Perfil" está disponível, possibilitando ao Administrador alterar suas próprias informações pessoais.

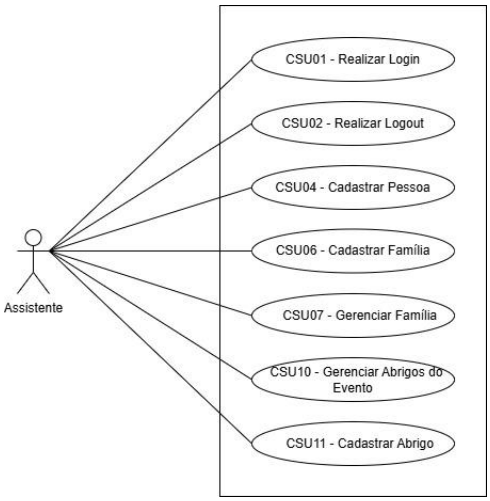
Figura 5 - Diagrama de caso de uso (administrador)



Fonte: Acervo do Autor (2024).

Conforme a Figura 6, o usuário do tipo Assistente tem acesso às funcionalidades relacionadas a pessoas, famílias e abrigos. O Assistente pode cadastrar novas pessoas e famílias, vinculando integrantes e pets aos grupos familiares. Também pode cadastrar abrigos e vincular famílias abrigadas, informando as datas de entrada e saída. Além disso, o Assistente tem acesso à opção "Meu Perfil", onde pode atualizar suas informações pessoais.

Figura 6 - Diagrama de caso de uso (assistente)



Fonte: Acervo do Autor (2024).

4.1.5 História de Usuários

Segundo Cohn (2019), histórias de usuários representam funcionalidades essenciais para quem utiliza ou adquire um sistema ou software. Elas são constituídas por três elementos principais: uma descrição textual, que serve como ferramenta de planejamento e registro; diálogos sobre a funcionalidade, que ajudam a esclarecer os detalhes específicos; e testes que documentam e validam os requisitos, sendo usados para verificar se a história foi completamente implementada. Neste trabalho, as histórias de usuário são organizadas com os seguintes itens:

a) US00 - Um identificador único para cada história de usuário, ajustado conforme novas histórias são adicionadas.

b) Critérios de aceite - Condições que a funcionalidade deve cumprir para ser aprovada pelo cliente ou equipe de desenvolvimento.

c) Requisitos - Especificações que a funcionalidade precisa atender para ser considerada completa e aceita pelo cliente e pela equipe.

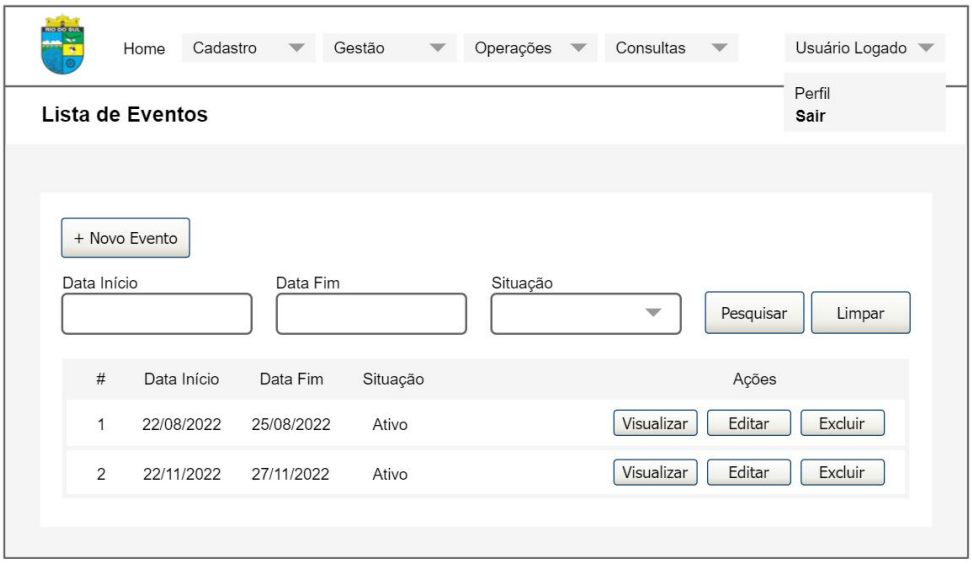
d) Protótipo de tela - Representação visual da interface do usuário que mostra a funcionalidade descrita na história de usuário.

Quadro 12 - História de Usuário 1

US01 - Como usuário, preciso realizar *login* para acessar o sistema
Critérios de Aceite: US01.01 - O sistema deve exibir uma tela de *login*, solicitando e-mail e senha para acesso. US01.02 - Se os dados inseridos forem inválidos, o acesso ao sistema deve ser bloqueado. US01.03 - Se os dados inseridos forem válidos, o usuário deve ser redirecionado para a tela inicial.
Requisitos: RF01

Protótipos:

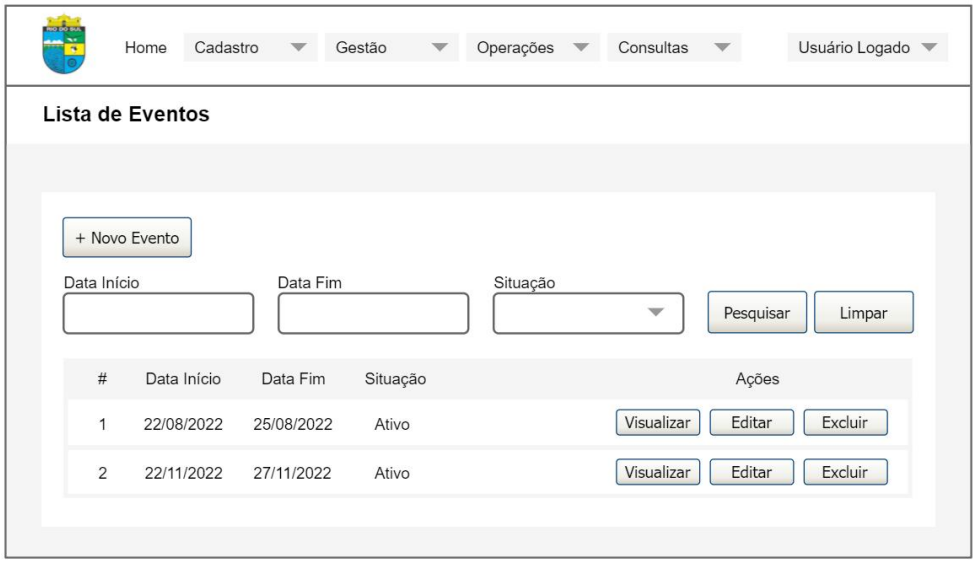

Quadro 13 - História de Usuário 2

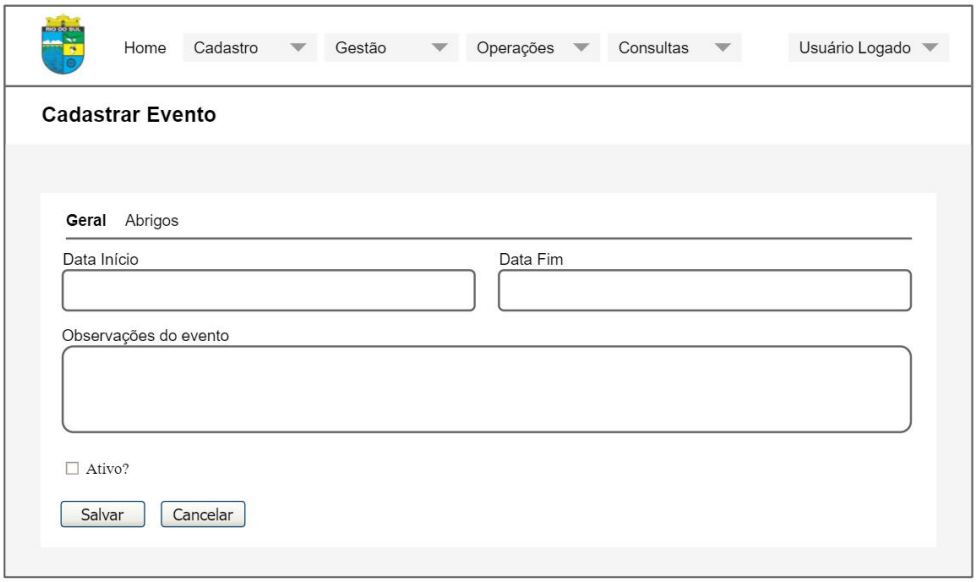
US02 - Como usuário, preciso conseguir realizar *logout* do sistema
Critérios de Aceite: US02.01 - Após o *login*, o sistema deve exibir um botão de *logout* em local de fácil visualização. US02.02 - Ao clicar em "Sair", o usuário deve ser imediatamente desconectado do sistema. US02.03 - Após o *logout*, o usuário deve ser redirecionado para a tela de login.
Requisitos: RF02
Protótipos: 

Quadro 14 - História de Usuário 3

US03 - Como usuário, preciso conseguir realizar o cadastro do evento
Critérios de Aceite: US03.01 - A listagem de eventos deve ter um botão para incluir um novo evento. US03.02 - O sistema deve validar o preenchimento de todos os campos obrigatórios. US03.03 - No cadastro de um evento, deve ser possível selecionar os abrigos que estarão disponíveis. US03.04 - O registro do evento deve ser salvo e integrado ao banco de dados.
Requisitos: RF08

Protótipos:





As histórias de usuário restantes serão apresentadas no Apêndice B.

4.2 TECNOLOGIAS E ARQUITETURA

No processo de análise mencionado no capítulo 4.1, utilizou-se a ferramenta draw.io para desenvolver os diagramas de caso de uso. Para realizar a modelagem física do banco de dados foi utilizado o SQL Power Architect, essa ferramenta auxiliou na definição das tabelas e dos relacionamentos, proporcionando uma visualização clara da estrutura do sistema e facilitando o entendimento das interações entre os componentes do banco de dados. Para criar os protótipos da história de usuários, foi utilizada a ferramenta Pencil, permitindo o

desenvolvimento de interfaces interativas e colaborativas que ajudaram a definir a experiência visual e funcional do sistema.

Para o desenvolvimento do sistema, foram utilizadas as ferramentas HTML, Tailwind CSS, JavaScript, PHP e Laravel.

O HTML, uma linguagem de marcação essencial na construção de páginas web, foi utilizado na versão HTML5, que oferece compatibilidade com os navegadores mais populares e permite a criação de uma estrutura responsiva. As *tags* descritivas do HTML5 também auxiliam na otimização para mecanismos de busca, como o Google, melhorando a indexação e a acessibilidade do conteúdo.

O Tailwind CSS é um *framework* que facilita a estilização de páginas web ao permitir que estilos sejam aplicados diretamente nas tags HTML por meio de classes utilitárias. Essa abordagem elimina a necessidade de um arquivo CSS global, pois cada classe do Tailwind representa uma propriedade de estilo específica, permitindo maior agilidade e controle no design dos elementos. Dessa forma, o desenvolvimento torna-se mais eficiente, com estilos centralizados e ajustáveis no próprio código, mantendo a consistência visual do projeto sem exigir a criação de estilos externos. Já o Blade, o sistema de *templates* do Laravel, é uma poderosa ferramenta para construir *views* dinâmicas e organizadas. Ele oferece uma sintaxe simples para manipulação de templates e permite a inclusão de lógica leve dentro das *views*, como loops e condicionais. Blade também facilita o uso de componentes e layouts reutilizáveis, promovendo uma estrutura modular que melhora a manutenibilidade do código. A integração do Blade com Tailwind CSS torna o desenvolvimento ainda mais eficiente, pois o Blade permite aplicar classes utilitárias diretamente nos componentes e seções de layout, mantendo o estilo e a lógica organizados em um único arquivo.

O JavaScript foi utilizado para definir o comportamento dinâmico das interfaces do sistema, incluindo a criação de modais para vinculação e pesquisa de registros. Além disso, foram implementados avisos e validações exibidos ao usuário com base no evento *DOMContentLoaded*, garantindo uma experiência de uso interativa e responsiva desde o carregamento inicial da página. O evento *onclick* também foi utilizado para acionar ações específicas ao clicar em elementos interativos, como botões e links, permitindo que o usuário abra modais e realize pesquisas de maneira rápida e intuitiva.

Para o desenvolvimento do sistema, foi utilizado PHP com o *framework* Laravel, adotando sua arquitetura MVC (*Model-View-Controller*) para estruturar o sistema. Com essa abordagem, o c*ontroller* gerencia a lógica do sistema, solicitando os dados ao *model* que lida com as interações com a base de dados e em seguida, passando as informações para a v*iew*,

que as renderiza o conteúdo para o usuário. O padrão MVC é amplamente utilizado para facilitar a separação de responsabilidades, tornando o código mais organizado. Além disso, a implementação das tabelas do banco de dados foi conduzida por meio das *migrations* do Laravel, o que permitiu definir e gerenciar a estrutura do banco de dados de forma versionada e integrada ao desenvolvimento do sistema.

Para o armazenamento das informações, foi utilizado o banco de dados PostgreSQL, reconhecido por sua robustez, segurança e capacidade de expansão. Ele oferece suporte a diversos tipos de dados e funcionalidades, além de possibilitar consultas avançadas, atendendo bem às necessidades do sistema.

4.3 IMPLEMENTAÇÃO DO PROJETO

A implementação do projeto envolve três etapas: modelagem do banco de dados, desenvolvimento *backend* e desenvolvimento *frontend*. A modelagem estrutura e organiza os dados, o *backend* gerencia a lógica de negócio e a comunicação com o banco de dados, e o *frontend* oferece uma interface interativa para o usuário, permitindo visualização e manipulação das informações. Essas etapas trabalham juntas para formar um sistema completo e eficiente.

4.3.1 Modelagem de Banco de Dados

Para a modelagem do banco de dados, iniciou-se com a criação de um diagrama de entidades e relacionamentos na ferramenta SQL Power Architect. Após a modelagem, a estrutura foi implementada no banco de dados PostgreSQL com o uso de *migrations* fornecidas pelo framework Laravel.

Para o desenvolvimento do sistema, foram mapeadas as seguintes tabelas:

a) eventos: representa os eventos (crises climáticas) do sistema, principais colunas incluem id, data_inicio, data_fim, observacao_evento e situacao.

b) abrigos: representa os abrigos do sistema, principais colunas incluem id, nome, dados de localização, capacidade_maxima, ativo e permite_exceder_capacidade.

c) familias: registra as famílias que se alojam nos abrigos em tempos de crise, principais colunas incluem id, nome, usuario_id, responsavel_id, contato_emergencia, dados de localização, renda_familiar e observacao_necessidades.

d) pessoas: detalha os integrantes de cada família alojada, principais colunas incluem id, nome, cpf, data_nascimento, idade e telefone.

e) familia_pets: armazena informações sobre os pets das famílias abrigadas, principais colunas incluem id, familia_id, tipo, porte e quantidade.

f) usuarios: registra os usuários do sistema, principais colunas incluem id, name, email, email_verified_at, situação e password.

g) produtos: representa os recursos disponíveis no sistema, principais colunas incluem id, nome, validade e categoria.

h) depositos: define os depósitos onde os recursos são armazenados, principais colunas incluem id, nome, dados de localização e abrigo_id.

i) evento_abrigos: estabelece a ligação entre os eventos e os abrigos associados, principais colunas incluem id, evento_id, abrigo_id e situacao.

j) evento_abrigo_familias: vincula famílias aos abrigos para cada evento, principais colunas incluem id, evento_abrigo_id, familia_id, data_entrada e data_saida.

k) familia_pessoas: faz a ligação entre os integrantes de cada família, principais colunas incluem id, familia_id, pessoa_id e parentesco.

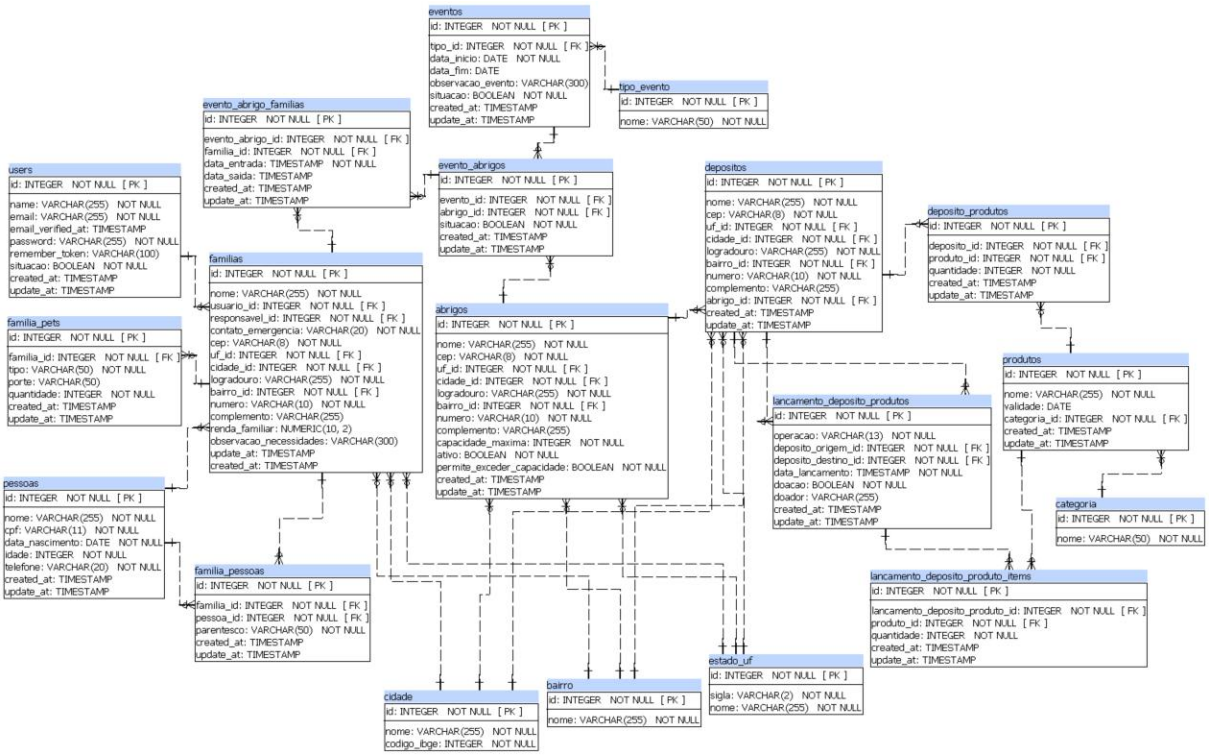
l) deposito_produtos: define a relação entre produtos e depósitos, principais colunas incluem id, deposito_id, produto_id e quantidade.

m) lancamento_deposito_produtos: representa os lançamentos de recursos nos depósitos, principais colunas incluem id, operacao, deposito_origem_id, deposito_destino_id, data_lancamento, doacao e doador.

n) lancamento_deposito_produto_items: detalha os itens específicos de cada lançamento, principais colunas incluem id, lancamento_deposito_produto_id, produto_id e quantidade.

Essas tabelas estruturam o sistema para o gerenciamento eficaz de eventos e recursos, conforme apresentado no diagrama de entidades e relacionamento da Figura 7.

Figura 7 - Diagrama de entidades e relacionamento



Fonte: Acervo do Autor (2024).

Para uma análise mais detalhada do diagrama de entidades e relacionamentos, recomenda-se consultar o Apêndice C, onde o diagrama é apresentado em formato ampliado, permitindo melhor visualização das entidades, seus atributos e os relacionamentos entre elas.

Nas Figuras 8 e 9, podem-se observar dois exemplos de *scripts* para criação de tabelas, enquanto os demais *scripts* estão disponíveis no Apêndice D.

Figura 8 - Criar Tabela de Eventos

```
(
    id bigint NOT NULL DEFAULT nextval('eventos_id_seq'::regclass),
    tipo_id bigint NOT NULL,
    data_inicio date NOT NULL,
    data_fim date,
    observacao_evento character varying(300) COLLATE pg_catalog."default",
    situacao boolean NOT NULL DEFAULT true,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT eventos_pkey PRIMARY KEY (id)
    CONSTRAINT eventos_tipo_id_foreign FOREIGN KEY (tipo_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 8, o *script* cria a tabela eventos no banco de dados, com uma estrutura que inclui um identificador único, datas de início e fim, observações e um indicador de situação do evento.

Figura 9 - Criar Tabela de Abrigos

```
CREATE TABLE IF NOT EXISTS public.abrigos
(
    id bigint NOT NULL DEFAULT nextval('abrigos_id_seq'::regclass),
    nome character varying(255) COLLATE pg_catalog."default" NOT NULL,
    cep character varying(8) COLLATE pg_catalog."default" NOT NULL,
    uf character varying(2) COLLATE pg_catalog."default" NOT NULL,
    cidade character varying(255) COLLATE pg_catalog."default" NOT NULL,
    logradouro character varying(255) COLLATE pg_catalog."default" NOT NULL,
    bairro character varying(255) COLLATE pg_catalog."default" NOT NULL,
    numero character varying(10) COLLATE pg_catalog."default" NOT NULL,
    complemento character varying(255) COLLATE pg_catalog."default",
    capacidade_maxima integer NOT NULL,
    ativo boolean NOT NULL DEFAULT true,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    permite_exceder_capacidade boolean NOT NULL DEFAULT false,
    CONSTRAINT abrigos_pkey PRIMARY KEY (id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 9, o *script* cria a tabela abrigos no banco de dados. Essa tabela inclui colunas para armazenar dados de identificação, localização e capacidade de cada abrigo, além de campos booleanos para indicar se o abrigo está ativo e se permite exceder a capacidade.

4.3.2 Camada de Aplicação (Backend)

Para a implementação da camada de aplicação (*backend*), optou-se por utilizar PHP, devido à familiaridade do autor com a linguagem, juntamente com o *framework* Laravel, que adota a arquitetura MVC, conforme abordado no capítulo 4.3. A escolha do Laravel permitiu uma organização clara do código, facilitando o desenvolvimento e a manutenção do sistema.

4.3.2.1 Roteamento

O Laravel oferece um sistema de roteamento padrão que define como as URLs do sistema direcionam para os métodos específicos dos controladores. As rotas são configuradas no arquivo web.php, onde cada rota pode ser associada a um método HTTP específico, facilitando o direcionamento das requisições para as ações corretas no *backend*.

Os métodos HTTP utilizados são apresentados no Quadro 15.

Quadro 15 - Métodos HTTP

Nome	Descrição
GET	Utilizado para exibir informações.
POST	Utilizado para criar novos registros.
PUT	Utilizado para atualizar registros existentes.
DELETE	Utilizado para excluir registros.

Fonte: Acervo do Autor (2024).

Nas rotas do protótipo, foi implementado um *middleware* que atua como uma camada intermediária entre a requisição HTTP e a aplicação, permitindo inspecionar, modificar ou restringir o acesso a certas rotas antes de serem executadas. Em outras palavras, ele age como um filtro para as requisições, decidindo se podem ou não acessar uma rota específica.

O recurso de *resource routing* do Laravel foi utilizado para gerar automaticamente rotas padrão para operações CRUD (*Create, Read, Update, Delete*) em cada um dos *controllers* especificados. Com o uso do *Route::resource*, o Laravel cria automaticamente as rotas para listar, criar, editar e excluir registros, sem necessidade de definições manuais para cada uma. Esse recurso simplifica a implementação de funcionalidades no sistema, tornando o código mais organizado e fácil de manter.

Na Figura 10, observa-se a implementação de um *middleware* de autenticação, que restringe o acesso aos recursos do sistema apenas a usuários autenticados, protegendo funcionalidades e dados sensíveis. A figura também apresenta o roteamento das principais telas do sistema, incluindo rotas para o gerenciamento de abrigos, pessoas, depósitos, produtos, famílias, lançamentos, estoques, doações e eventos.

Figura 10 - Exemplo de roteamento

```php
Route::get('/', function () {
    return view('auth.login');
});

Route::middleware('auth')->group(function () {
    Route::resource('/abrigos', AbrigoController::class);
    Route::resource('/pessoas', PessoaController::class);
    Route::resource('/depositos', DepositoController::class);
    Route::resource('/produtos', ProdutoController::class);
    Route::resource('/familias', FamiliaController::class);
    Route::resource('/lancamentos', LancamentoDepositoProdutoController::class);
    Route::resource('/estoques', EstoqueController::class);
    Route::resource('/doacoes', DoacaoController::class);
    Route::resource('/eventos', EventoController::class);
```

Fonte: Acervo do Autor (2024).

Conforme descrito por Laravel (2024), o *middleware* de autenticação verifica se o usuário está autenticado antes de permitir o acesso às rotas protegidas, caso contrário, ele redireciona o usuário para a página de *login*.

4.3.2.2 Models

O *Model* é o núcleo do padrão MVC, representando a lógica, as regras e o comportamento do sistema de forma independente da interface do usuário. Ele gerencia e armazena os dados que são recuperados pelo controlador e em seguida, exibidos pela camada de visualização. Na Figura 11, é possível observar o Modelo de Abrigos.

Figura 11 - Exemplo de model

```php
class Abrigo extends Model
{
    use HasFactory;

    protected $fillable = [
        'nome',
        'cep',
        'uf',
        'cidade',
        'logradouro',
        'bairro',
        'numero',
        'complemento',
        'capacidade_maxima',
        'ativo',
        'permite_exceder_capacidade',
    ];

    protected $casts = [
        'capacidade_maxima' => 'integer',
        'ativo' => 'boolean',
        'permite_exceder_capacidade' => 'boolean',
    ];

    public function familias()
    {
        return $this->belongsToMany(Familia::class, 'abrigo_familias', 'abrigo_id', 'familia_id')
                    ->withPivot('id', 'data_entrada', 'data_saida')
                    ->withTimestamps();
    }

    public function deposito()
    {
        return $this->hasOne(Deposito::class);
    }

    public function eventos()
    {
        return $this->belongsToMany(Evento::class, 'evento_abrigos')
                    ->withTimestamps();
    }
}
```

Fonte: Acervo do Autor (2024).

Com base no modelo acima, a propriedade *$fillable* define os campos que podem ser preenchidos em massa ao criar ou atualizar registros, protegendo o modelo e permitindo que apenas esses campos sejam atribuídos diretamente. Isso inclui campos como nome, cep, uf,

cidade, capacidade_maxima, ativo e demais campos, que representam as características principais de um abrigo e são essenciais para o gerenciamento dos dados no sistema.

A propriedade *$casts*, por sua vez, garante que os tipos de dados sejam interpretados corretamente ao serem manipulados no sistema, convertendo automaticamente capacidade_maxima para um tipo *integer* e os campos ativo e permite_exceder_capacidade para boolean, assegurando que essas propriedades se comportem de forma adequada em operações lógicas e matemáticas.

Além disso, o modelo define relacionamentos importantes para o sistema de gestão de abrigos. A função familias() estabelece uma relação de muitos para muitos entre abrigos e famílias, permitindo que um abrigo hospede várias famílias e que cada família possa estar em múltiplos abrigos ao longo do tempo. A tabela intermediária abrigo_familias permite armazenar informações adicionais sobre essa relação, como data_entrada e data_saida de cada família no abrigo, o que é essencial para o controle de ocupação.

A função deposito() define uma relação de um para um entre o abrigo e um depósito, permitindo que cada abrigo tenha um depósito específico associado para armazenar os recursos disponíveis. Esse vínculo ajuda a manter o controle dos suprimentos em cada abrigo.

A função eventos() cria uma relação de muitos para muitos entre abrigos e eventos, possibilitando associar um abrigo a diferentes eventos climáticos ou emergências. Essa estrutura é registrada na tabela intermediária evento_abrigos, que permite associar múltiplos abrigos a um evento e vice-versa, armazenando também as informações de criação e atualização dos registros com o uso de *withTimestamps()*.

Essas propriedades e métodos juntos fazem do modelo abrigo uma representação completa da entidade abrigo no sistema, facilitando o gerenciamento de dados e interações com outras entidades de maneira organizada e segura.

4.3.2.3 Controllers

Os controladores são responsáveis por gerenciar as solicitações HTTP, processar a lógica de negócio e fornecer as respostas adequadas ao cliente. Atuam como intermediários entre o Modelo e a Visão, organizando o fluxo de informações e garantindo o tratamento correto para cada solicitação, seja para exibir dados, criar, atualizar ou excluir registros. Com o uso de controladores, a estrutura do código é simplificada e a lógica de negócio do sistema fica centralizada em um único ponto, facilitando a manutenção e o desenvolvimento.

Na Figura 12 e 13 , é possível observar o *Controller* de Produtos.

Figura 12 - Exemplo de controller (index, create e store)

```php
class ProdutoController extends Controller
{
    public function index(Request $request)
    {
        $produtos = Produto::query();

        if($request->nome){
            $produtos->where('nome', 'like', "%{$request->nome}%");
        }

        if ($request->validade) {
            $produtos->whereDate('validade', $request->validade);
        }

        if($request->categoria){
            $produtos->where('categoria', 'like', "%{$request->categoria}%");
        }

        $produtos->orderBy('nome');
        $produtos = $produtos->get();
        return view('produtos.index', compact('produtos'));
    }


    public function create()
    {
        return view('produtos.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'nome' => 'required|string|min:4|max:255',
            'validade' => 'nullable|date',
            'categoria' => 'required|string|max:50',
        ]);

        try {
            Produto::create($request->all());
        } catch (\Exception $e) {
            dd($e->getMessage());
        }

        return redirect()->route('produtos.index');
    }
```

Fonte: Acervo do Autor (2024).

A classe *ProdutoController* é responsável por gerenciar as operações relacionadas aos produtos no sistema, permitindo exibição, criação, edição, atualização e exclusão de registros. Ela atua como um intermediário entre as requisições do usuário e o modelo de dados, controlando as operações de CRUD (*Create, Read, Update, Delete*) para a entidade de produtos.

Na Figura 12, são apresentados os métodos principais de criação e visualização de produtos. O método *index(Request $request)* permite exibir a lista de produtos, com a possibilidade de o usuário aplicar filtros por nome, data de validade e categoria. Após a aplicação dos filtros, os produtos são ordenados pelo nome e exibidos na *view produtos.index*,

proporcionando uma visualização personalizada. O método *create()* exibe a *view produtos.create*, onde o usuário pode inserir informações de um novo produto, preparando a página para o cadastro, sem realizar operações no banco de dados. O método *store(Request $request)* é responsável por validar e armazenar um novo produto no banco de dados. Ele assegura que o nome e a categoria sejam obrigatórios e que a validade seja opcional, caso a validação seja bem-sucedida, o produto é salvo, caso contrário, os erros são exibidos ao usuário.

Figura 13 - Exemplo de controller (show, edit, update e destroy)

```php
public function show(string $id)
    {
        $produto = Produto::findOrFail($id);
        return view('produtos.show', compact('produto'));
    }

    public function edit(string $id)
    {
        $produto = Produto::findOrFail($id);
        return view('produtos.edit', compact('produto'));
    }

    public function update(Request $request, string $id)
    {
        $request->validate([
            'nome' => 'required|string|min:4|max:255',
            'validade' => 'nullable|date',
            'categoria' => 'required|string|max:50',
        ]);

        $produto = Produto::findOrFail($id);
        $produto->update($request->all());

        return redirect()->route('produtos.index');
    }

    public function destroy(string $id)
    {
        $produto = Produto::findOrFail($id);

        if ($produto->depositos()->exists()) {
            return redirect()->back()->with('error', 'Não é possível excluir este produto porque ele está vinculado a um depósito.');
        }

        $produto->delete();
        return redirect()->route('produtos.index')->with('status', 'Produto excluído com sucesso!');
    }
}
```

Fonte: Acervo do Autor (2024).

Na Figura 13, são apresentados os métodos para exibir, editar, atualizar e excluir produtos específicos. O método *show($id)* exibe os detalhes de um produto, identificando-o pelo id e retornando-o para a *view produtos.show*. O método *edit($id)* exibe a *view produtos.edit*, permitindo que o usuário modifique as informações do produto. O método *update*(*Request $request, $id*) atualiza os dados de um produto no banco de dados, validando as entradas e aplicando as alterações caso estejam corretas. Já o método *destroy($id)* permite a exclusão de um produto, mas verifica se ele está vinculado a algum depósito antes de realizar a exclusão. Se houver vínculo, a exclusão é impedida e uma mensagem de erro é exibida, se não houver, o produto é removido, e o usuário é redirecionado.

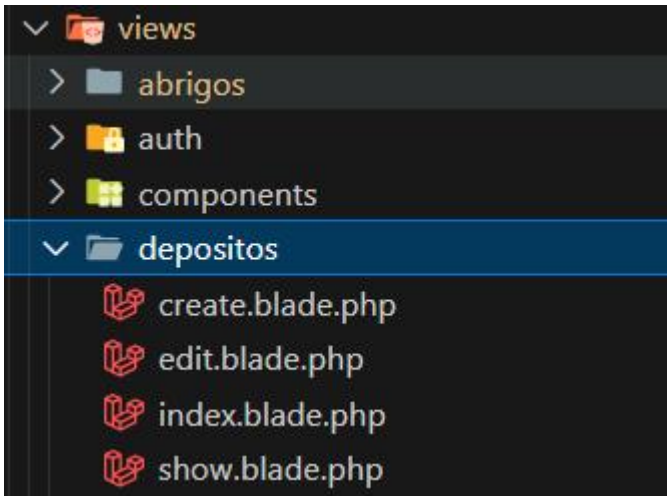
Em resumo, a classe *ProdutoController* organiza e centraliza as operações de CRUD para produtos, facilitando a interação do usuário com funcionalidades de listagem, criação, edição e exclusão de produtos no sistema. Ela garante a integridade dos dados e oferece uma interface eficiente para o gerenciamento de produtos no sistema de gestão de abrigos.

4.3.2.4 Views

A visão do projeto representa a estrutura de interface que é exibida para o usuário com base em cada método do *Controller*. Cada operação, como listar (*index*), criar (*create*), editar (*edit*) e detalhar (*show*) registros, possui sua própria estrutura de exibição. Por exemplo, quando o método *index*() do *controller* é executado, ele retorna a *view* correspondente, *index*, que apresenta a listagem de registros. Dessa forma, cada método do *Controller* está associado a uma *view* específica, fornecendo uma interface adequada para cada ação no sistema.

A Figura 14 mostra a organização dos arquivos da *view* de depósitos.

Figura 14 - Organização dos arquivos da view



Fonte: Acervo do Autor (2024).

A organização apresentada na Figura 14 segue uma estrutura de pastas para facilitar o gerenciamento e a manutenção das *views* no Laravel. Nesse caso, as *views* estão organizadas em subpastas dentro da pasta principal *views*, com cada subpasta representando um módulo ou recurso específico do sistema, como abrigos e depósitos.

Essa organização é feita para manter o código mais organizado, especialmente em projetos com múltiplos componentes e funcionalidades. Ao agrupar as *views* relacionadas a cada recurso em uma pasta separada, torna-se mais fácil localizar, editar e manter as *views*

correspondentes. Por exemplo, a pasta depositos contém arquivos Blade (*create.blade.php, edit.blade.php, index.blade.php* e *show.blade.php*) que representam as *views* de criação, edição, listagem e visualização dos depósitos, respectivamente.

Essa estrutura é recomendada pelo Laravel para manter o código organizado e facilitar a navegação, permitindo que desenvolvedores identifiquem rapidamente quais *views* estão associadas a cada parte do sistema e realizem modificações de maneira eficiente.

4.3.3 Camada de Apresentação (Frontend)

Para implementar a camada de apresentação (*frontend*), foram utilizadas tecnologias como HTML, Tailwind CSS, Blade e JavaScript.

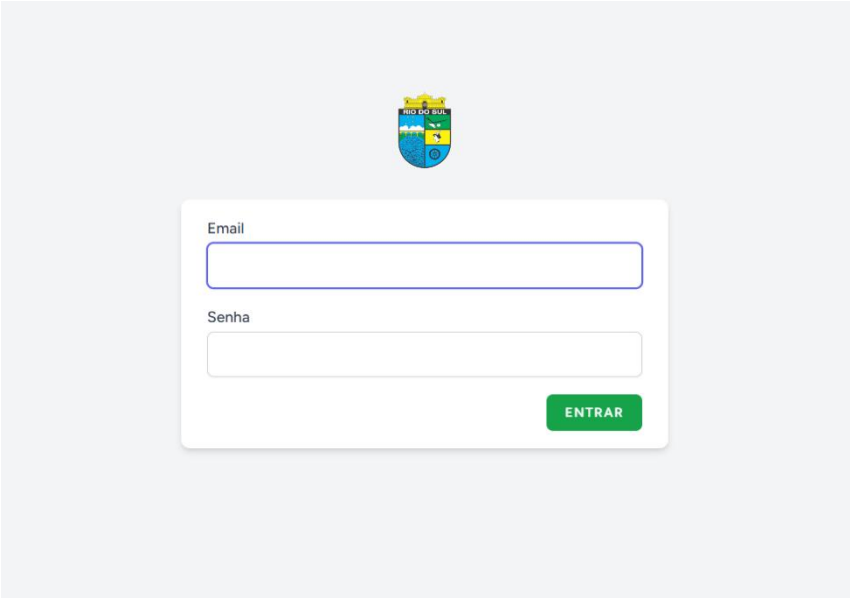
O HTML define a estrutura semântica da página web, enquanto o Tailwind CSS, um *framework* utilitário, permite estilizar diretamente os elementos HTML com classes específicas. O Blade, mecanismo de templates do Laravel, facilita a criação de interfaces dinâmicas, integrando lógica diretamente nas *views* de maneira simples e eficiente. Já o JavaScript adiciona interatividade à interface, incluindo a exibição de modais, alertas, validações e outros comportamentos dinâmicos, aprimorando a experiência do usuário.

4.3.3.1 Apresentação das Interfaces

Conforme mencionado anteriormente, o protótipo foi desenvolvido com o objetivo de facilitar o gerenciamento dos abrigos. Neste capítulo, são detalhadas as principais rotinas de uso do sistema pelos usuários, abordando como é realizado o acesso e quais funcionalidades estão disponíveis no protótipo.

Para acessar as funcionalidades do protótipo, é necessário que o usuário realize o *login*, fornecendo seu e-mail e senha. A tela de *login* (Figura 15) é a porta de entrada para o sistema, garantindo que apenas usuários autorizados tenham acesso às funcionalidades.

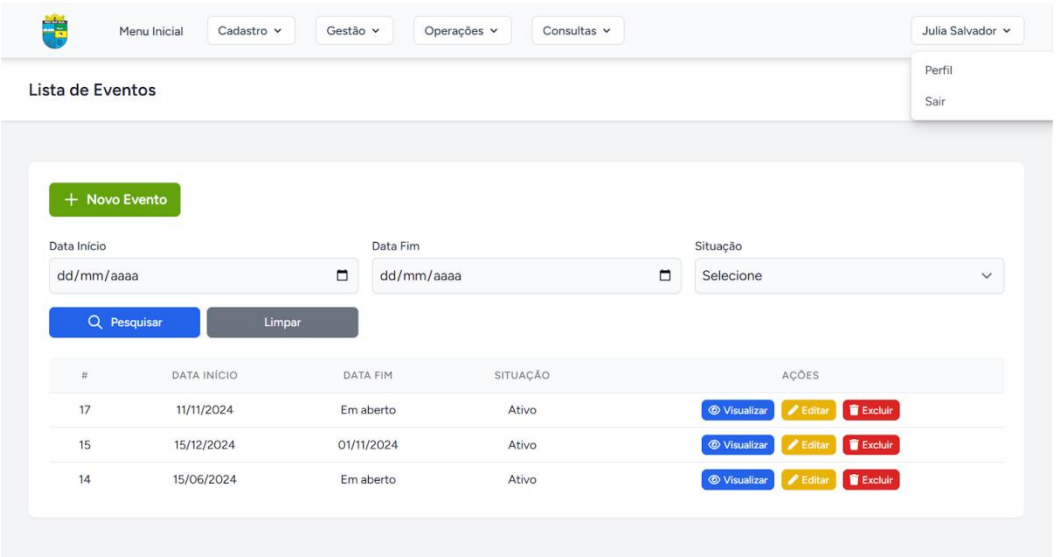
Figura 15 - Realizar Login (RF01)



Fonte: Acervo do Autor (2024).

O protótipo disponibiliza uma opção de *logout*, que permite ao usuário encerrar sua sessão de forma segura (Figura 16). Esse recurso é essencial para proteger os dados do usuário e impedir o acesso não autorizado às informações do sistema. Com o *logout*, o usuário pode ter a certeza de que suas informações estarão protegidas ao encerrar sua utilização do sistema.

Figura 16 - Realizar Logout (RF02)

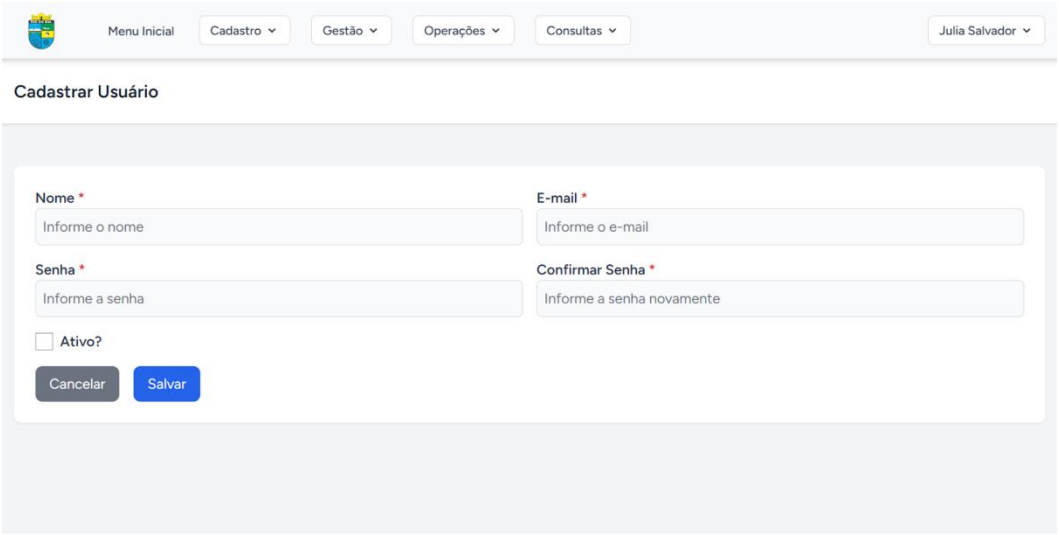


Fonte: Acervo do Autor (2024).

Para facilitar a gestão de acesso, o protótipo possui uma tela de cadastro de usuários (Figura 17). Nessa tela, é possível registrar novos usuários informando nome, e-mail e senha.

Esse cadastro é essencial para definir quem terá permissão de acesso ao sistema, e a funcionalidade permite que o administrador do sistema mantenha um controle rigoroso sobre os usuários que podem utilizar o protótipo.

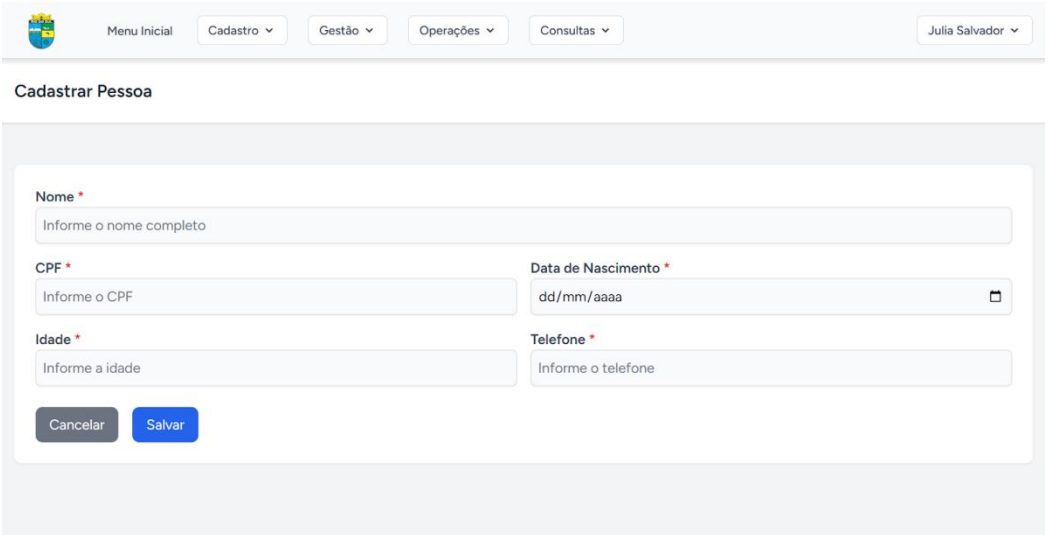
Figura 17 - Cadastrar Usuário (RF03)



Fonte: Acervo do Autor (2024).

A tela de cadastro de pessoas (Figura 18) permite que os operadores do sistema registrem as pessoas que serão monitoradas e assistidas. Os dados cadastrados incluem nome, CPF, data de nascimento, idade e telefone. Essas informações são fundamentais para identificar e documentar as pessoas abrigadas, possibilitando um acompanhamento detalhado de cada indivíduo.

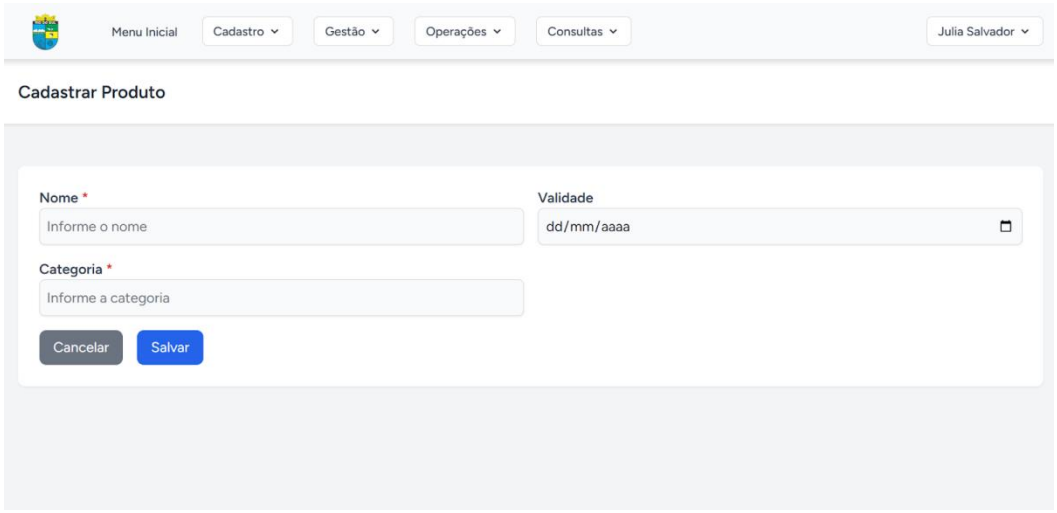
Figura 18 - Cadastrar Pessoa (RF04)



Fonte: Acervo do Autor (2024).

O protótipo inclui uma tela de cadastro de produtos (Figura 19), onde o usuário pode registrar informações básicas dos produtos, como nome, validade e categoria. Esse cadastro inicial serve para identificar os itens no sistema, facilitando seu gerenciamento em operações futuras. Após o registro, os produtos podem ser movimentados e acompanhados por meio da funcionalidade de lançamento, permitindo um controle mais detalhado de estoque e distribuição conforme necessário.

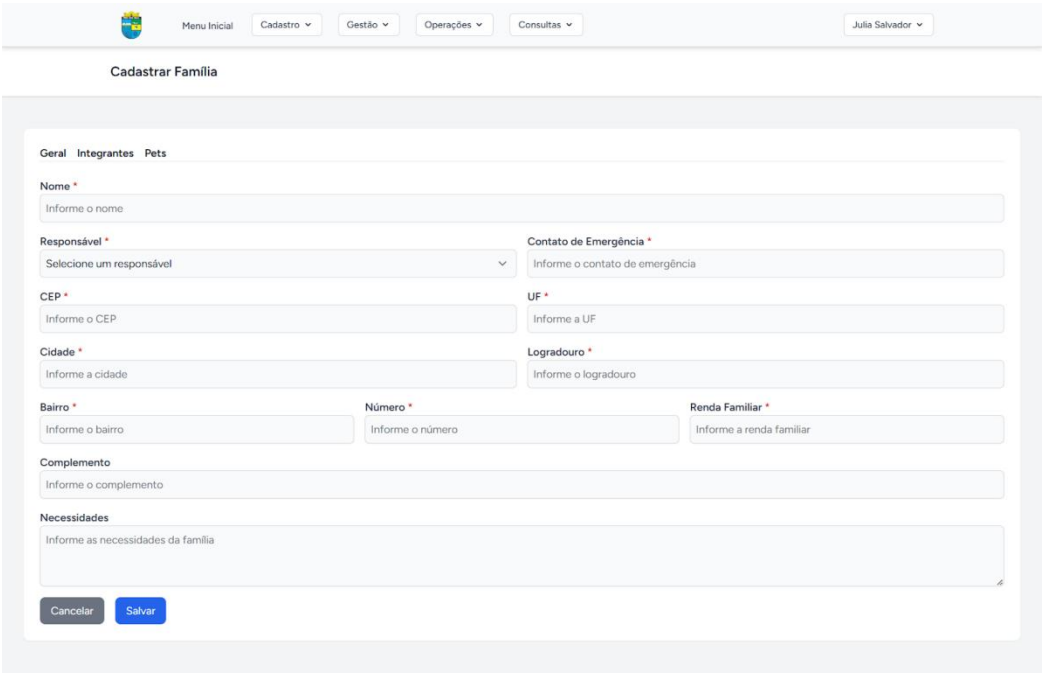
Figura 19 - Cadastrar Produto (RF05)



Fonte: Acervo do Autor (2024).

A tela de cadastro de famílias (Figura 20) permite registrar dados sobre as famílias acolhidas nos abrigos. Nessa tela, o usuário informa dados como nome da família, responsável, contato de emergência, e informações detalhadas do endereço (CEP, UF, cidade, logradouro, bairro, número, complemento). Além disso, é possível vincular os integrantes da família, especificando o parentesco com o responsável, e registrar os pets pertencentes ao grupo familiar, informando o tipo, porte e quantidade de cada um.

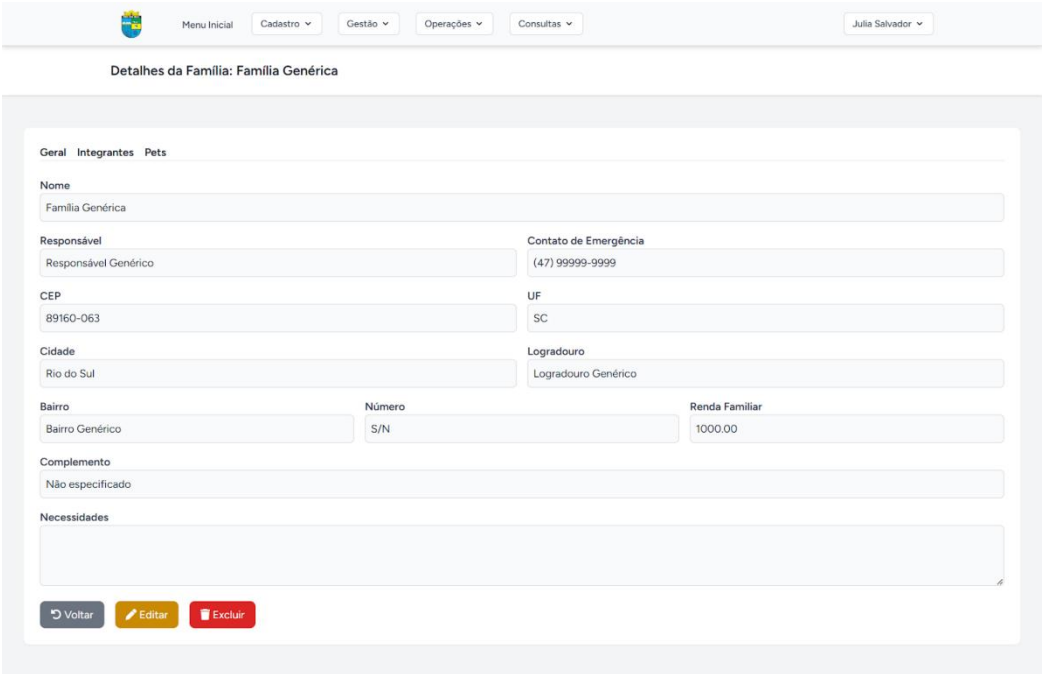
Figura 20 - Cadastrar Família (RF06)



Fonte: Acervo do Autor (2024).

A visualização das famílias cadastradas é realizada em uma tela específica (Figura 21), onde o usuário pode gerenciar e consultar os dados da família. Essa tela permite que as informações gerais sobre as famílias sejam acessadas e atualizadas conforme necessário.

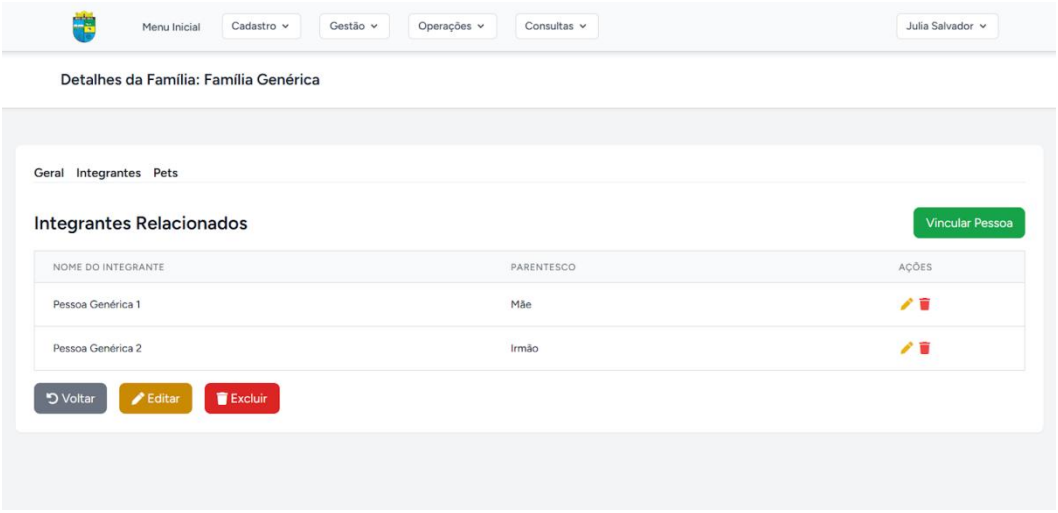
Figura 21 - Gerenciar Família - Geral (RF07)



Fonte: Acervo do Autor (2024).

Além disso, há uma tela dedicada aos integrantes da família (Figura 22), que detalha cada pessoa associada ao grupo familiar, facilitando a organização das informações dos membros da família.

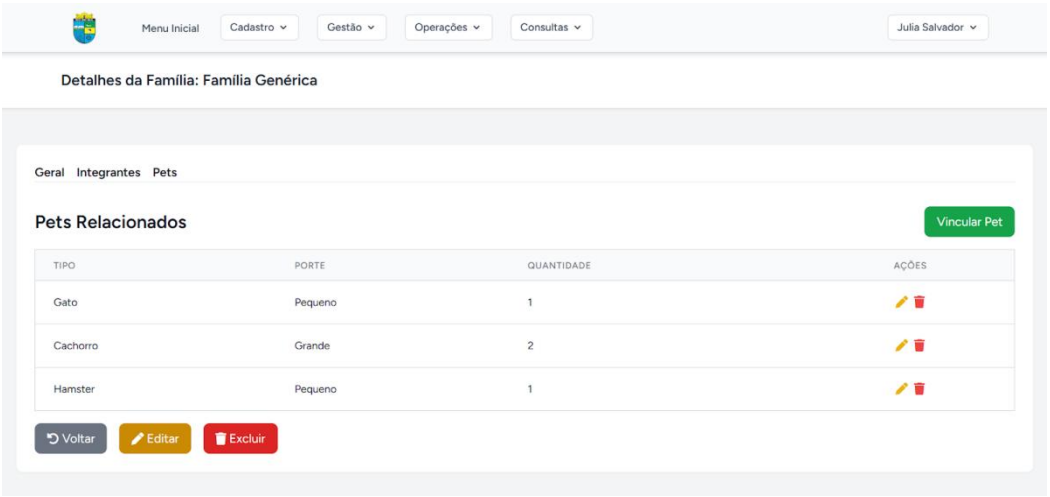
Figura 22 - Gerenciar Família - Integrantes (RF07)



Fonte: Acervo do Autor (2024).

Por fim, a visualização dos pets vinculados à família também é realizada em uma tela específica (Figura 23), permitindo gerenciar os animais sob cuidado das famílias acolhidas, garantindo um controle eficiente.

Figura 23 - Gerenciar Família - Pets (RF07)

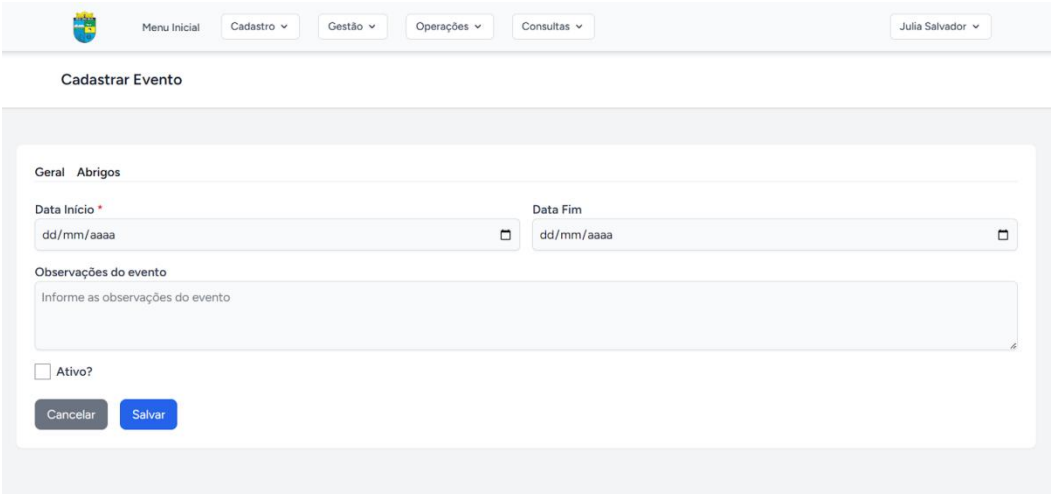


Fonte: Acervo do Autor (2024).

Para gerenciar crises ou emergências, o protótipo conta com uma tela de cadastro de eventos (Figura 24). Nessa tela, o usuário pode registrar informações como a data de início e fim do evento, além de observações relevantes. A funcionalidade permite também associar

abrigos ao evento, registrando os abrigos que foram disponibilizados especificamente em resposta à ocorrência do evento.

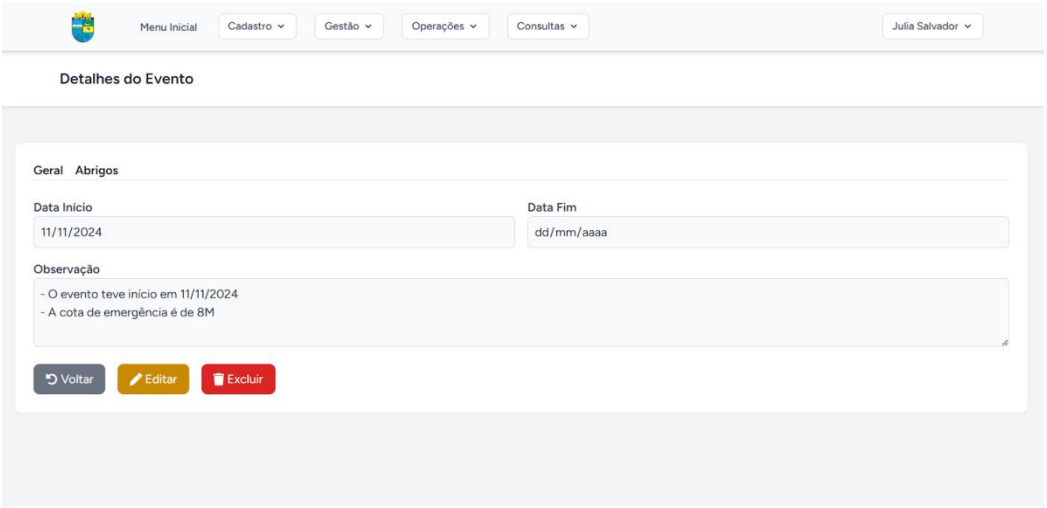
Figura 24 - Cadastrar Evento (RF08)



Fonte: Acervo do Autor (2024).

A visualização dos eventos cadastrados é disponibilizada em uma tela específica (Figura 25), onde o usuário pode consultar e gerenciar os detalhes de cada evento. Nessa tela, é possível visualizar as informações relevantes de cada evento, garantindo que as ações necessárias possam ser tomadas de maneira adequada.

Figura 25 - Gerenciar Evento - Geral (RF09)

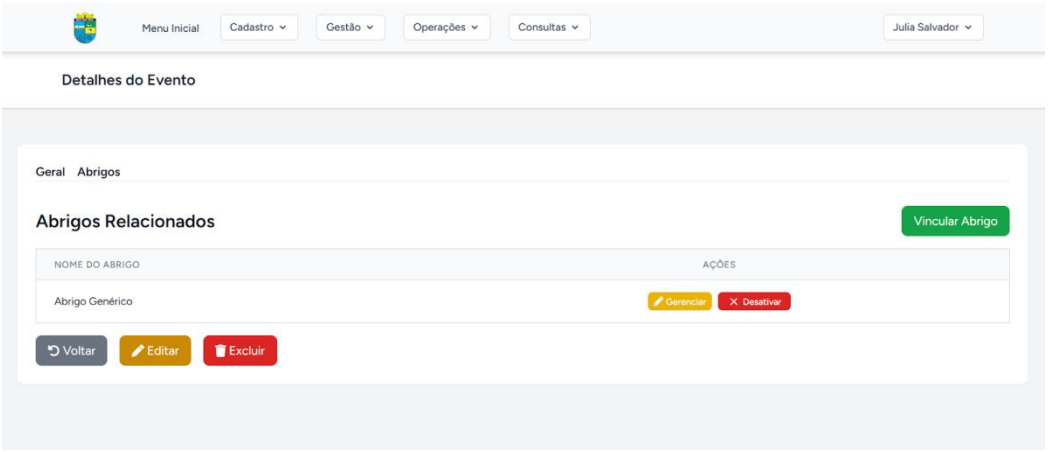


Fonte: Acervo do Autor (2024).

Além disso, a tela permite vincular novos abrigos conforme a necessidade do evento e gerenciá-los (Figura 26), possibilitando o controle das famílias acolhidas em cada abrigo disponibilizado. Essa funcionalidade assegura uma administração eficiente do acolhimento

durante o evento, permitindo que as necessidades de cada abrigo sejam atendidas de maneira organizada.

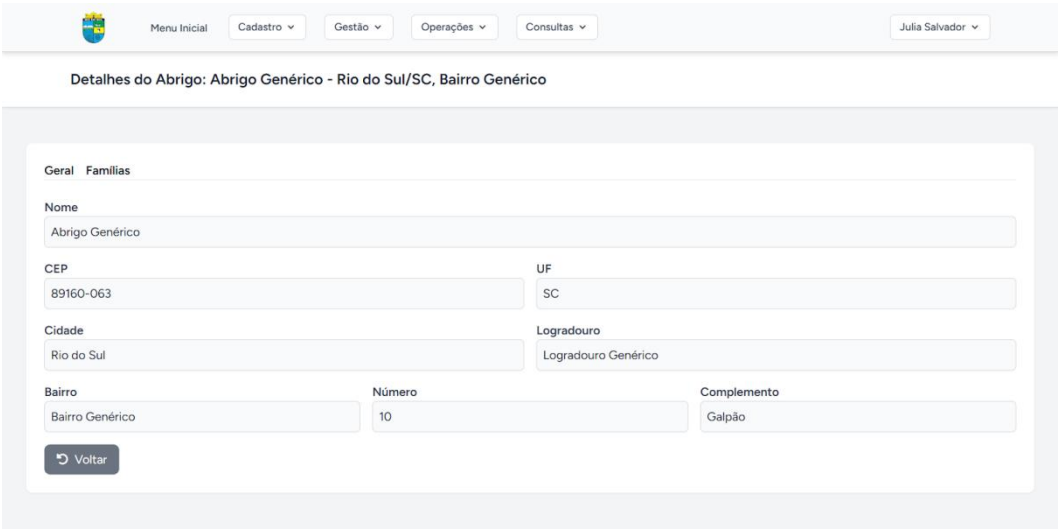
Figura 26 - Gerenciar Evento - Abrigos (RF09)



Fonte: Acervo do Autor (2024).

Na tela de visualização dos abrigos de um evento específico, o usuário pode acessar os detalhes do abrigo (Figura 27), permitindo consultar todas as informações importantes relacionadas ao abrigo e acompanhar sua situação atual.

Figura 27 - Gerenciar Abrigos do Evento - Geral (RF10)

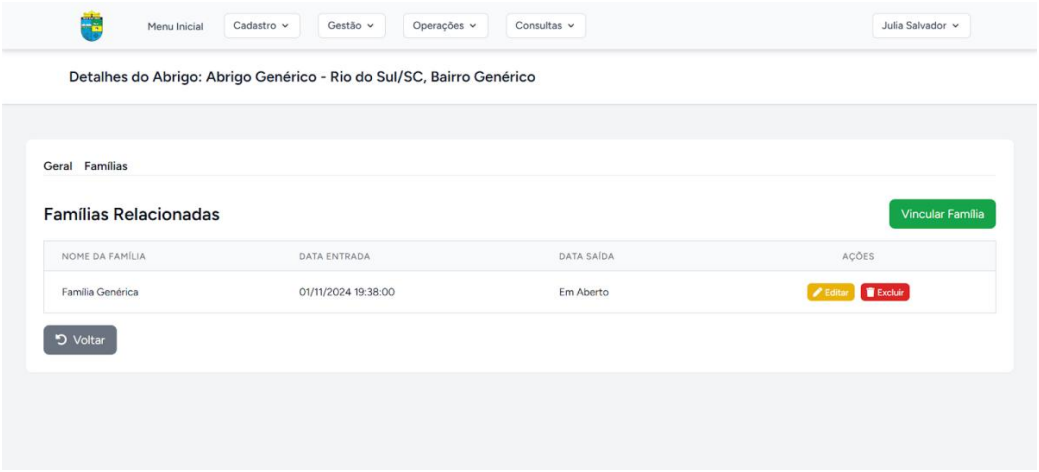


Fonte: Acervo do Autor (2024).

Além disso, é possível gerenciar as famílias acolhidas (Figura 28), incluindo o registro da data de entrada e, posteriormente, da data de saída de cada família, garantindo um fluxo organizado para o controle de ocupação dos abrigos.

Essa funcionalidade oferece um acompanhamento eficiente da lotação dos abrigos, facilitando o gerenciamento do acolhimento durante crises e garantindo que a organização seja mantida ao longo do processo.

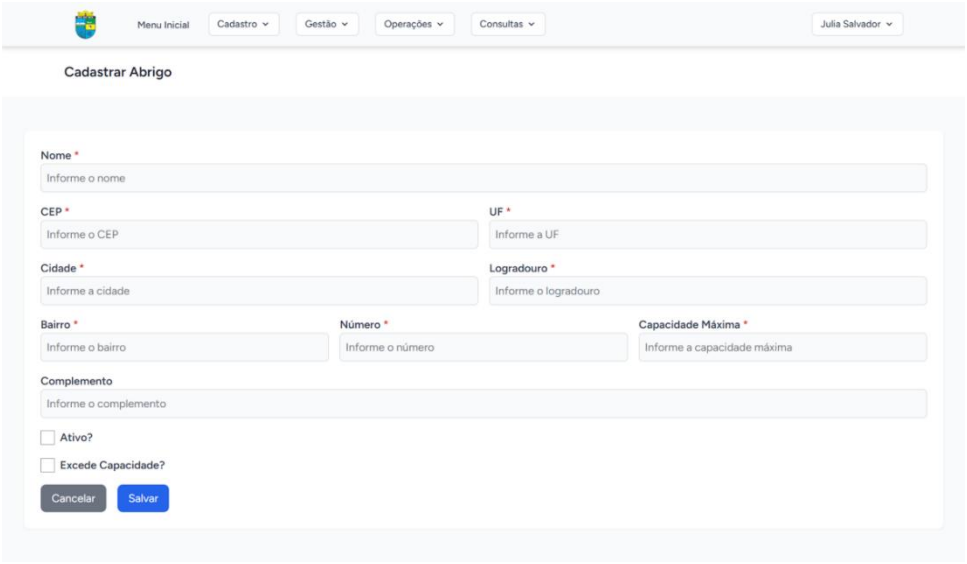
Figura 28 - Gerenciar Abrigos do Evento - Famílias (RF10)



Fonte: Acervo do Autor (2024).

Para manter um controle eficiente dos locais de acolhimento, o protótipo oferece uma tela de cadastro de abrigos (Figura 29). Nessa tela, são registradas informações detalhadas como o nome do abrigo, endereço completo (CEP, UF, cidade, logradouro, bairro, número, complemento), capacidade máxima de pessoas, se o abrigo está ativo e se permite exceder a capacidade. Esse cadastro é essencial para garantir que cada abrigo seja gerenciado de acordo com suas características e limitações.

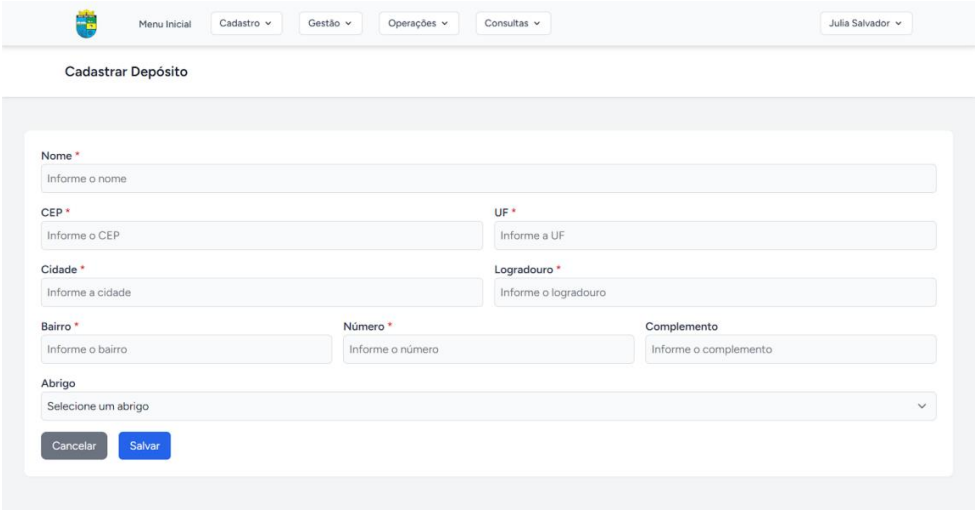
Figura 29 - Cadastrar Abrigo (RF11)



Fonte: Acervo do Autor (2024).

A tela de cadastro de depósitos (Figura 30) permite registrar os depósitos onde serão armazenados os recursos e suprimentos necessários. O usuário informa o nome do depósito, o abrigo relacionado, e o endereço completo (CEP, UF, cidade, logradouro, bairro, número, complemento). Esse recurso facilita a gestão dos suprimentos e assegura que cada depósito seja registrado de maneira organizada e precisa.

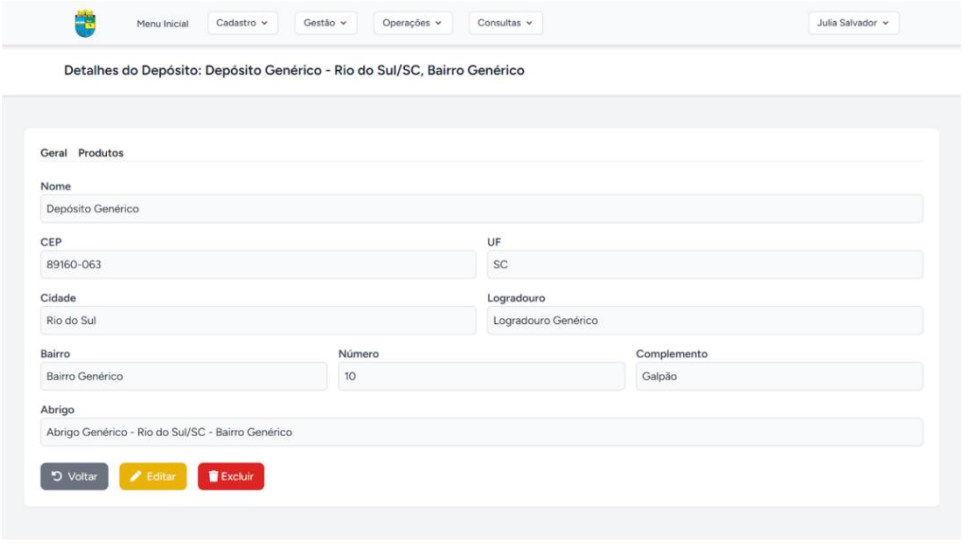
Figura 30 - Cadastrar Depósito (RF12)



Fonte: Acervo do Autor (2024).

Na tela de visualização dos depósitos (Figura 31), o usuário pode acessar informações detalhadas de cada depósito, como localização e capacidade, garantindo um acompanhamento preciso das características e da situação de cada depósito.

Figura 31 - Gerenciar Depósitos - Geral (RF13)

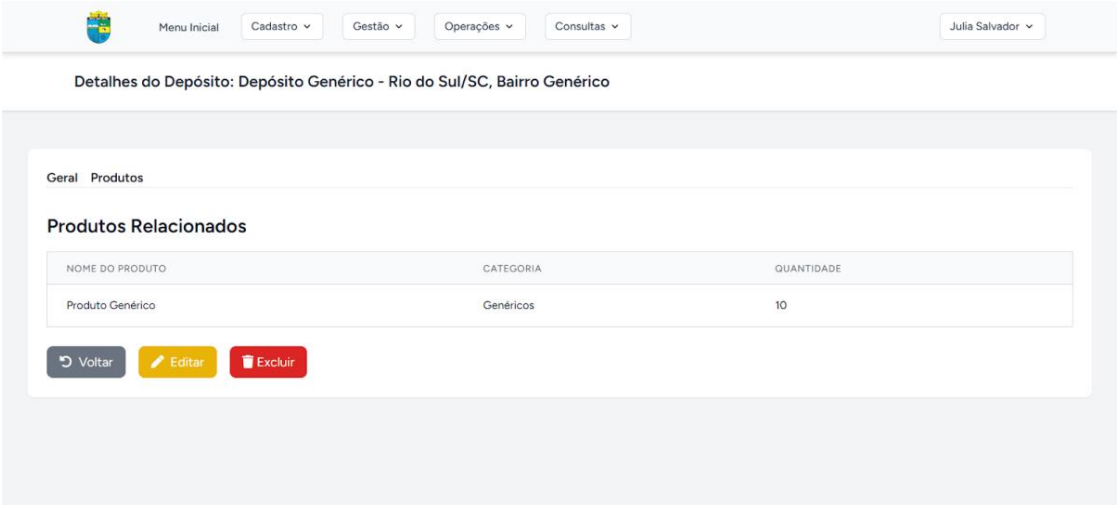


Fonte: Acervo do Autor (2024).

Além disso, é possível monitorar os recursos disponíveis em estoque (Figura 32), consultando os suprimentos armazenados em cada depósito e facilitando o gerenciamento de entradas e saídas de recursos.

Essa funcionalidade promove um controle mais eficaz do estoque e facilita a distribuição eficiente dos itens em situações de necessidade, garantindo que os recursos estejam disponíveis de forma organizada e acessível.

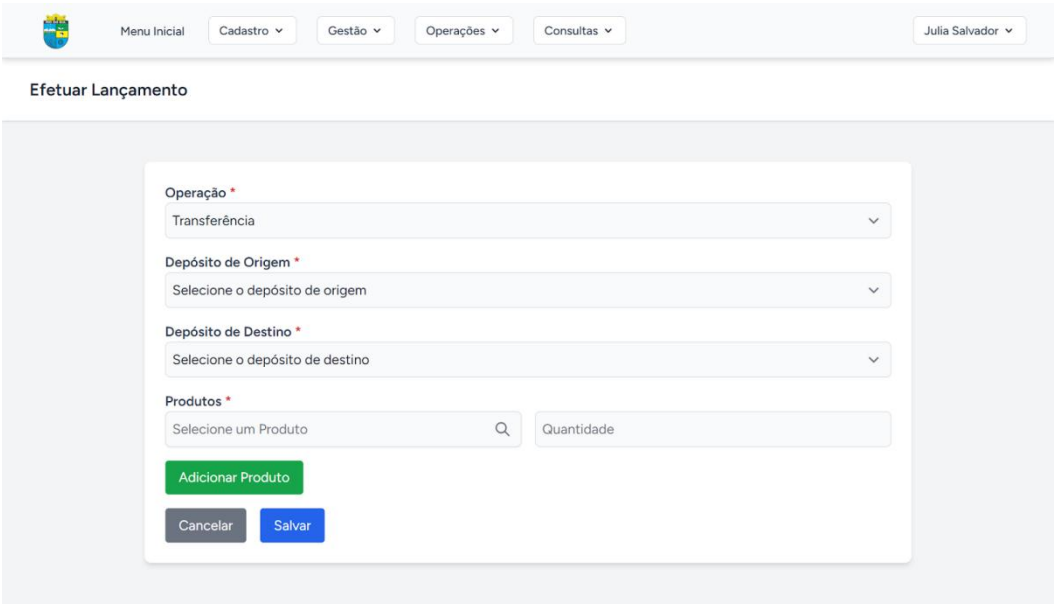
Figura 32 - Gerenciar Depósitos - Produtos (RF13)



Fonte: Acervo do Autor (2024).

A tela de cadastro de lançamentos (Figura 33) possibilita o registro detalhado da movimentação de recursos entre depósitos, incluindo transferências, entradas e saídas de produtos. Nessa tela, o usuário pode especificar o tipo de operação (entrada, saída ou transferência), selecionar o depósito de origem e o de destino, além de informar os produtos envolvidos, a quantidade e, em caso de doação, o nome do doador. Essa funcionalidade proporciona um controle rigoroso do fluxo de recursos, assegurando que todas as movimentações sejam devidamente registradas e acompanhadas.

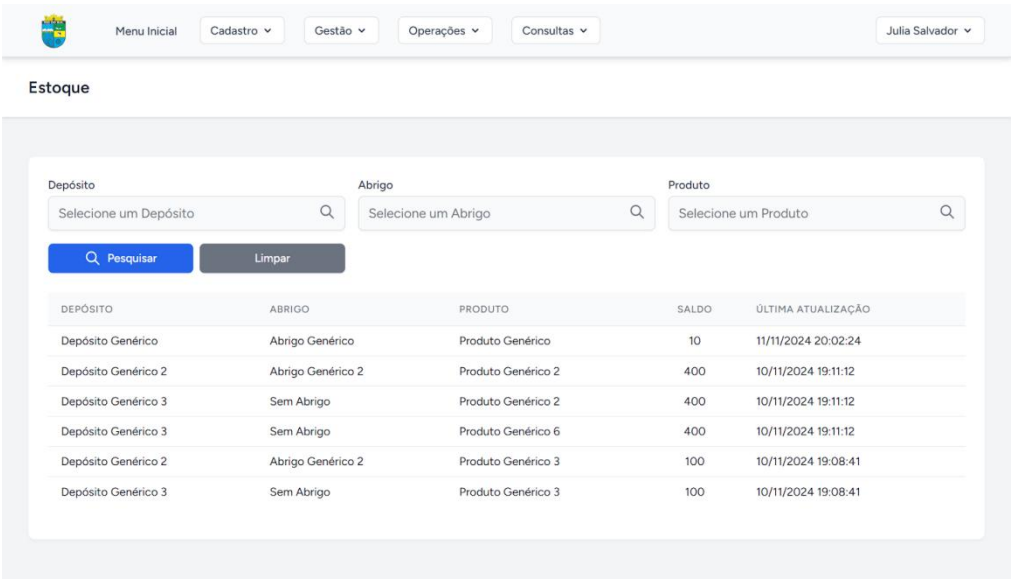
Figura 33 - Cadastrar Lançamentos (RF14)



Fonte: Acervo do Autor (2024).

A tela de estoque (Figura 34) fornece uma visão consolidada dos recursos disponíveis em cada depósito, permitindo que o usuário visualize a quantidade de suprimentos em estoque. Esse recurso é essencial para o planejamento e distribuição dos itens, assegurando que os recursos sejam alocados conforme a necessidade.

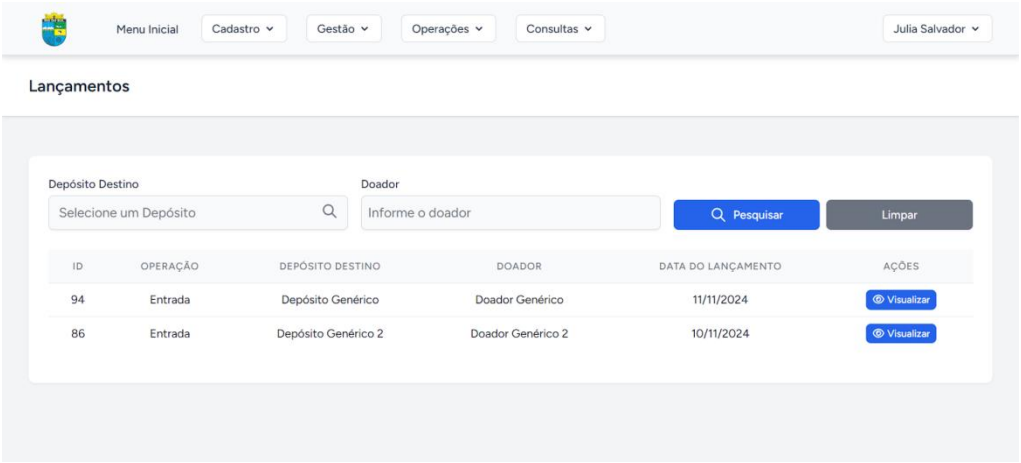
Figura 34 - Consultar Estoque (RF15)



Fonte: Acervo do Autor (2024).

Para promover a transparência e o controle das contribuições, o protótipo conta com uma tela de doações (Figura 35), onde o usuário pode visualizar todas as doações recebidas de forma organizada.

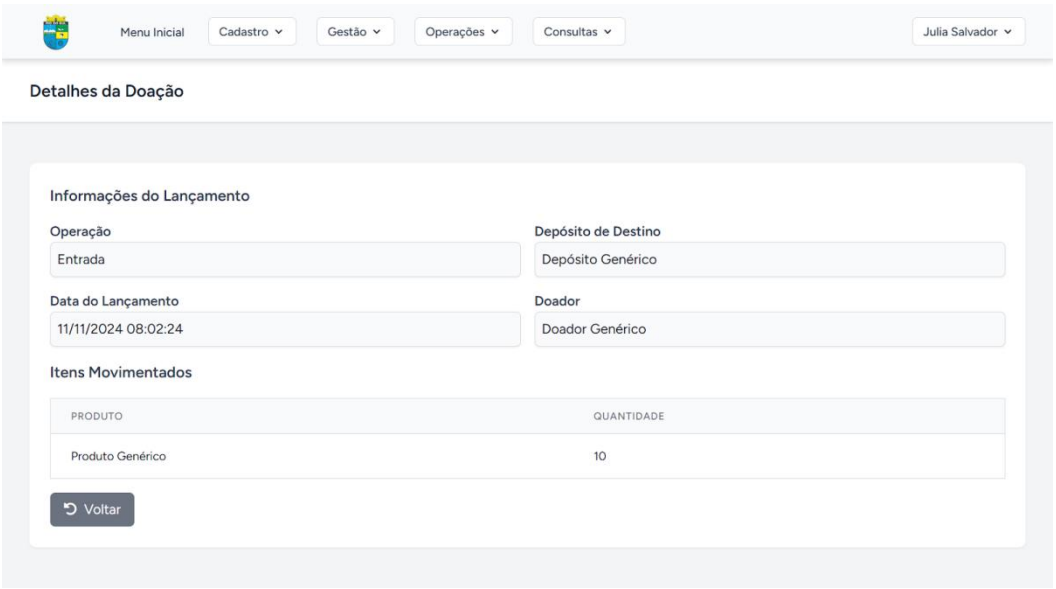
Figura 35 - Consultar Doações (RF16)



Fonte: Acervo do Autor (2024).

Nessa tela, também é possível acessar os detalhes sobre os itens doados e os doadores (Figura 36), permitindo um acompanhamento detalhado das contribuições e promovendo uma visão clara e completa das doações realizadas.

Figura 36 - Consultar Doações Detalhadas (RF16)



Fonte: Acervo do Autor (2024).

Esse recurso facilita o acompanhamento das doações e garante um controle mais transparente e eficiente das contribuições, permitindo que o sistema tenha uma visão detalhada das doações realizadas.

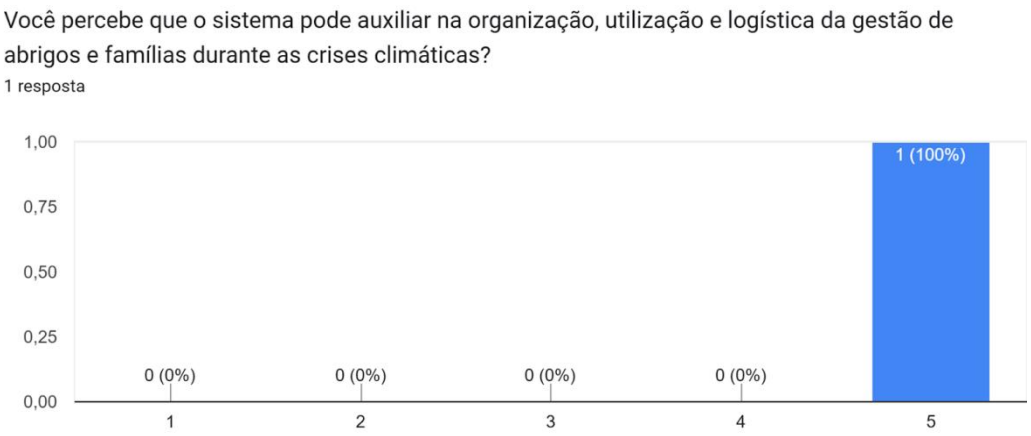
4.4 TESTES E VALIDAÇÕES

Foi realizada uma validação com os usuários através de uma visita à Secretaria de Assistência Social em outubro de 2024, onde o sistema em desenvolvimento foi apresentado para avaliação e discussão de possíveis melhorias. Dois usuários participaram do teste, destacando que ele seria um apoio significativo para a organização durante crises climáticas, considerando que, atualmente, toda a gestão é realizada por meio de planilhas Excel.

O sistema inclui uma funcionalidade de gestão de recursos, algo que a secretaria não possui atualmente, pois o controle é feito por estimativas e, quando disponível, com o auxílio de notas fiscais. Foram sugeridas algumas melhorias, principalmente na área de emissão de relatórios, uma funcionalidade que será implementada futuramente.

Ao final da reunião de validação, foi disponibilizado um formulário para que os usuários envolvidos da Secretaria de Assistência Social respondessem com base no sistema apresentado. O questionário utilizou a metodologia SUS (*System Usability Scale*) para avaliar a usabilidade do sistema. Vale ressaltar que o formulário teve apenas uma resposta, baseada na avaliação conjunta dos usuários participantes. O *score* utilizado no questionário é: 1 - discordo completamente, 2 - discordo, 3 - neutro, 4 - concordo e 5 - concordo completamente. Abaixo seguem as respostas obtidas.

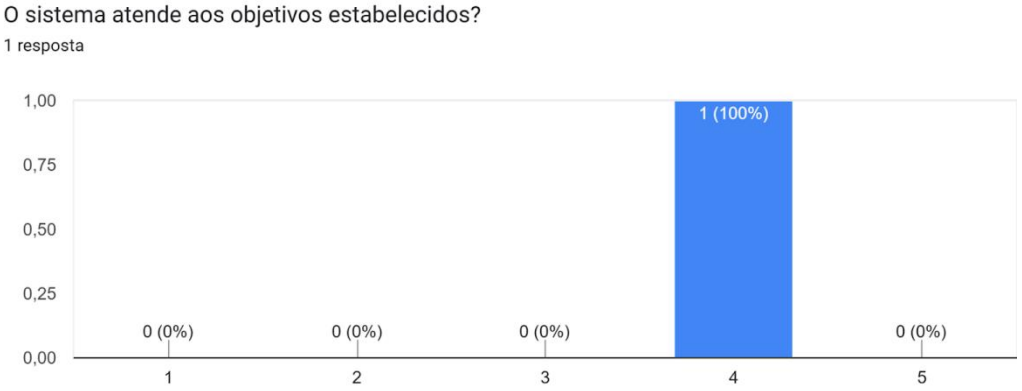
Figura 37 - Pergunta 01 do Questionário



Fonte: Acervo do Autor (2024).

Com base na metodologia SUS (*System Usability Scale*), que mede a usabilidade de um sistema por meio de uma escala de pontuação, a avaliação apresentada na Figura 37 indica que o sistema foi avaliado positivamente para o apoio à gestão de abrigos e famílias durante crises climáticas. Na pergunta sobre a capacidade do sistema de auxiliar na organização, utilização e logística dos abrigos, a resposta obteve nota máxima (5). Isso sugere que os usuários percebem o sistema como uma ferramenta potencialmente útil para organizar e facilitar a logística dos abrigos em situações de crise climática.

Figura 38 - Pergunta 02 do Questionário



Fonte: Acervo do Autor (2024).

Na avaliação da Figura 38, a pergunta sobre se o sistema atende aos objetivos estabelecidos obteve uma pontuação de 4 na única resposta coletada. Essa nota sugere que o usuário considera o sistema bastante eficaz para cumprir seus propósitos, embora ainda haja algum espaço para melhorias. A pontuação próxima do máximo indica uma percepção positiva em relação ao atendimento dos objetivos, reforçando a utilidade do sistema para o que foi planejado.

Os demais resultados recebidos serão apresentados e detalhados no Apêndice E, proporcionando uma visão mais completa da avaliação realizada.

5. CONSIDERAÇÕES FINAIS

O protótipo desenvolvido teve como objetivo principal apoiar a Secretaria de Assistência Social na gestão dos abrigos disponibilizados em situações de crises climáticas, proporcionando um gerenciamento simplificado e intuitivo para os envolvidos.

Para o desenvolvimento, as tecnologias escolhidas foram suficientes e adequadas para essa primeira versão do protótipo. Para o desenvolvimento do *backend*, utilizou-se o *framework* Laravel, que fornece uma variedade de métodos e uma estrutura avançada para PHP, facilitando diversos aspectos do desenvolvimento. Com a adoção da arquitetura MVC (*Model-View-Controller*), o Laravel organiza o código de maneira mais limpa e estruturada e oferece funcionalidades como roteamento flexível, sistema de *templates* com Blade e ferramentas de manipulação de banco de dados, tornando-se uma opção robusta para o desenvolvimento de aplicações web de forma eficiente e escalável. No *frontend*, foi utilizado o Tailwind CSS para estilizar as páginas, melhorando a usabilidade das interfaces HTML. O comportamento e dinamismo das telas foi implementado com JavaScript, enquanto o armazenamento de dados foi realizado pelo banco de dados PostgreSQL.

Entre os objetivos definidos para este trabalho, o primeiro visava obter informações sobre a gestão dos abrigos junto à Secretaria de Assistência Social, o que foi detalhado no capítulo de levantamento de informações. O segundo objetivo era descrever as tecnologias utilizadas no desenvolvimento do protótipo, conforme exposto no capítulo de referencial teórico. O terceiro objetivo foi cumprido com base no levantamento dos requisitos funcionais, não funcionais e das regras de negócio para a implementação das funcionalidades do protótipo. O quarto objetivo consistia na construção do protótipo com as tecnologias escolhidas, e está descrito no capítulo de implementação do projeto. Finalmente, o quinto objetivo foi validar a usabilidade do protótipo junto à Secretaria de Assistência Social, o que foi realizado com êxito. No entanto, futuras revisões serão necessárias conforme novas versões do protótipo forem desenvolvidas. Além disso, é importante ampliar a base de usuários para validação, envolvendo mais profissionais e equipes que possam fornecer uma variedade de *feedbacks* e enriquecer o aprimoramento contínuo do sistema.

O protótipo busca possibilitar e simplificar o gerenciamento de abrigos, eventos climáticos, famílias, integrantes, pets, depósitos e recursos durante as crises climáticas no município, promovendo um controle mais preciso e uma resposta ágil nessas situações.

Seu principal benefício é auxiliar na organização, utilização e logística dos abrigos durante crises climáticas. Com sua utilização, será possível mapear todos os eventos

climáticos que ocorrem no município, identificar os abrigos disponíveis para cada evento, gerenciar os recursos disponíveis e distribuí-los conforme necessário. O sistema também facilita a avaliação das necessidades das famílias alojadas e fornece um controle mais preciso das informações, substituindo a abordagem atual baseada em planilhas e estimativas.

Por fim, é possível observar que todos os objetivos do trabalho foram atendidos, incluindo o objetivo geral. No entanto, durante o desenvolvimento do protótipo e nas validações realizadas com a Secretaria, surgiram novas ideias para aprimorar a aplicação e torná-la ainda mais eficiente. Devido ao tempo limitado, essas melhorias não foram implementadas e estão registradas no capítulo de recomendações de trabalhos futuros.

5.1 RECOMENDAÇÃO DE TRABALHOS FUTUROS

Para aprimorar a usabilidade e a funcionalidade do protótipo, foram levantados alguns requisitos opcionais descritos no Quadro 11.

O primeiro requisito opcional (RF17) sugere a implementação de um *dashboard* inicial no menu principal do protótipo, exibindo informações gerais dos abrigos, como a quantidade de pessoas e pets abrigados e o tempo médio de permanência das famílias.

O segundo requisito opcional (RF18) propõe a geração de Relatórios abrangendo dados detalhados sobre abrigos, famílias, integrantes, pets e recursos, facilitando a análise e gestão das informações no sistema.

O terceiro requisito opcional (RF19) recomenda a inclusão de um registro de auditorias, que acompanhará todas as alterações realizadas nos registros do sistema, promovendo maior segurança e transparência nas operações.

O quarto requisito opcional (RF20) visa a criação de um portal de transparência, permitindo que informações gerais sobre abrigos, famílias, integrantes, pets e recursos sejam disponibilizadas ao público, para que possam acompanhar a situação dos abrigos durante crises climáticas e outras emergências.

O quinto requisito opcional (RF21) é a implementação de uma funcionalidade para desativar registros no sistema, permitindo que registros possam ser desativados em vez de excluídos permanentemente, de modo a manter um histórico completo das informações.

Por fim, o sexto requisito opcional (RF22) trata da administração de permissões, possibilitando que o protótipo gerencie as permissões de acesso para cada usuário do sistema, garantindo controle sobre quem pode visualizar e editar diferentes dados.

Além disso, o requisito não funcional (RNF09) sugere tornar o protótipo responsivo para dispositivos móveis, garantindo que as funcionalidades possam ser acessadas e utilizadas facilmente em diferentes tipos de dispositivos, como smartphones e tablets, facilitando o acesso ao sistema por equipes em campo e outras partes interessadas.

Já o requisito não funcional (RNF10) aborda a implementação da conformidade com a Lei Geral de Proteção de Dados (LGPD). Este requisito assegura que os dados pessoais das famílias e pessoas cadastradas sejam tratados em conformidade com os regulamentos de privacidade e proteção de dados, garantindo o respeito aos direitos dos usuários. Ele também inclui a possibilidade de revogação dos dados pessoais (RF23), conforme previsto pela LGPD, minimizando os riscos de vazamento ou uso inadequado das informações e assegurando que os titulares mantenham controle sobre seus dados.

Esses requisitos, opcionais e não funcionais visam fortalecer a usabilidade, a segurança e a transparência do sistema, proporcionando uma experiência mais robusta e eficiente aos usuários e facilitando o gerenciamento das informações pela Secretaria de Assistência Social.

Finalmente, sugere-se uma nova rodada de validação com a Secretaria de Assistência Social para revisar as regras de negócio e a usabilidade do sistema, assegurando que ele atenda plenamente às necessidades identificadas. Além disso, recomenda-se uma avaliação do sistema após sua utilização em um evento de crise climática real, já que, durante a fase de construção deste protótipo, não foi possível testá-lo em um cenário real de crise.

REFERÊNCIAS

ALVES, William Pereira. **Banco de dados.** São Paulo: Érica, 2014. *E-book.*

AMOESEI, Juliana. **O que são regras de negócio?**. 2023. Disponível em: <https://www.alura.com.br/artigos/o-que-sao-regras-de-negocio>. Acesso em: 03 jun. 2024.

ARIEL, Úrsula. **Guia Completo do Tailwind CSS:** Desenvolvimento Rápido e Estilizado - Parte I. 2024. Disponível em: <https://community.revelo.com.br/guia-completo-do-tailwind-css-desenvolvimento-rapido-e-estilizado-parte-i/>. Acesso em: 03 nov. 2024.

AZEVEDO, Julia. **Enchente:** o que é e principais causas. 2023. Disponível em: <https://www.ecycle.com.br/enchente/>. Acesso em: 01 nov. 2024.

BASSO, Lourenço de Oliveira; MILETTO, Evandro Manara. Linguagem PHP. In: MILETTO, Evandro Manara; BERTAGNOLLI, Silvia de Castro (Org.). **Desenvolvimento de software II:** introdução ao desenvolvimento web com HTML, CSS, javascript e PHP. Porto Alegre: Bookman, 2014. *E-book.*

BERTAGNOLLI, Silvia de Castro; MILETTO, Evandro Manara. Criação e formatação de páginas web com HTML/CSS. In: MILETTO, Evandro Manara; BERTAGNOLLI, Silvia de Castro (Org.). **Desenvolvimento de software II:** introdução ao desenvolvimento web com HTML, CSS, javascript e PHP. Porto Alegre: Bookman, 2014. *E-book.*

BEST OF BI. **SQL Power Architect:** Data Modeling & Profiling Tool. 2024. Disponível em: <https://bestofbi.com/products/sql-power-architect-data-modeling/>. Acesso em: 03 jun. 2024.

BRASIL. **Lei nº 13.709, de 14 de agosto de 2018.** Dispõe sobre a proteção de dados pessoais e altera a Lei nº 12.965, de 23 de abril de 2014 (Marco Civil da Internet). Brasília, DF: Diário Oficial da União, 2018. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm>. Acesso em: 03 nov. 2024.

CARDOSO, Leandro da Conceição. **Frameworks back end.** São Paulo: Platos Soluções Educacionais S.A, 2021. *E-book.*

CARDOSO, Vírginia M. **Linguagem SQL:** fundamentos e práticas. São Paulo: Saraiva, 2013. *E-book.*

CLEMENTE, Paulo. **Lógica de programação para iniciantes em programação.** 2023. Disponível em: <https://blog.rocketseat.com.br/logica-de-programacao-para-iniciantes-em-programacao/>. Acesso em: 03 jun. 2024.

COHN, Mike. **User Stories Applied for Agile Software Development.** Addison-Wesley, 2009. *E-book.*

DEFESA CIVIL. **Enchentes na história de Rio do Sul.** 2024. Disponível em: <https://defesacivil.riodosul.sc.gov.br/index.php?r=externo%2Fplanilha>. Acesso em: 03 jun. 2024.

DEVMEDIA. **Blade Engine:** Utilizando templates no Laravel. 2016. Disponível em: <https://www.devmedia.com.br/blade-engine-utilizando-templates-no-laravel/36749>. Acesso em: 03 nov. 2024.

DEVMEDIA. **Comandos básicos em SQL - insert, update, delete e select.** 2016. Disponível em: <https://www.devmedia.com.br/comandos-basicos-em-sql-insert-update-delete-e-select/37170>. Acesso em: 03 jun. 2024.

DRAW.IO. **Introduction to diagrams.** 2023. Disponível em: <https://www.drawio.com/doc/>. Acesso em: 03 jun. 2024.

ESCOLHA LIVRE. **Pencil Project.** 2024. Disponível em: <https://escolhalivre.org.br/pencil-project/#>. Acesso em: 03 jun. 2024.

FLANAGAN, David. **JavaScript:** o guia definitivo. 6. ed. Porto Alegre: Bookman, 2013. *E-book.*

GDACS. **O que é GDACS?**. s.d. Disponível em: <https://gdacs.org/About/overview.aspx>. Acesso em: 03 jun. 2024.

GOV.BR. **Centro Nacional de Monitoramento e Alertas de Desastres Naturais - Cemaden/MCTI.** s.d. Disponível em: <https://www.gov.br/cemaden/pt-br>. Acesso em: 03 jun. 2024.

KOMATSU, André Suzuki Veiga. **Interpretador de linguagens de consulta relacionais.** São Paulo, 2011, 44 f. Monografia Universidade de São Paulo.

LARAVEL. **Blade Templates.** 2024. Disponível em: <https://laravel.com/docs/11.x/blade>. Acesso em: 03 nov. 2024.

MACÊDO, Diego. **SGBD - sistema de gerenciamento de banco de dados.** 2022. Disponível em: <https://www.diegomacedo.com.br/sgbd-sistema-de-gerenciamento-de-banco-de--dados/>. Acesso em: 03 jun. 2024.

MACHADO, Felipe Nery Rodrigues. **Banco de dados:** projeto e implementação. 4. ed. São Paulo: Érica, 2020. *E-book.*

MAXIM, Bruce R.; PRESSMAN, Roger S. **Engenharia de Software:** uma abordagem profissional. 9. ed. Porto Alegre: AMGH Editora Ltda, 2021. *E-book.*

MICROSOFT. **O que é PostgreSQL?**. 2024. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-postgresql>. Acesso em: 03 jun. 2024.

OLIVEIRA, Cláudio Luís Vieira; ZANETTI, Humberto Augusto Piovesana. **JavaScript descomplicado:** programação para a Web, IoT e dispositivos móveis. São Paulo: Érica, 2020. *E-book.*

PAULA FILHO, Wilson de Pádua. **Engenharia de Software - Produtos.** 4. ed. Rio de Janeiro: Grupo GEN, 2019. *E-book.*

POSTGRESQL. **The PostgreSQL Global Development Group.** 2024. Disponível em: <https://www.postgresql.org/docs/>. Acesso em: 03 jun. 2024.

SANTANA, André. **Linguagens de programação:** uma breve introdução contextualizada. 2023. Disponível em: <https://www.alura.com.br/artigos/linguagem-programacao>. Acesso em: 03 jun. 2024.

SARAIVA, Maurício de Oliveira. Linguagem PHP - Introdução. In: SARAIVA, Maurício de Oliveira; BARRETO, Jeanine dos Santos (Org.). **Desenvolvimento de sistemas com PHP.** Porto Alegre: SAGAH, 2018. *E-book.*

SILBERSCHATZ, Abraham. **Sistema de banco de dados.** 7. ed. Rio de Janeiro: GEN LTC, 2020. *E-book.*

SILVA, Júlia Marques Carvalho; MILETTO, Evandro Manara. Comportamento com JavaScript. In: MILETTO, Evandro Manara; BERTAGNOLLI, Silvia de Castro (Org.). **Desenvolvimento de software II:** introdução ao desenvolvimento web com HTML, CSS, javascript e PHP. Porto Alegre: Bookman, 2014. *E-book.*

SILVA, Maurício Samy. **CSS3:** Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3. São Paulo: Novatec, 2012.

SILVA, Maurício Samy. **Fundamentos de HTML5 e CSS3.** São Paulo: Novatec, 2015.

SOARES, Walace. **PHP 5:** conceitos, programação e integração com banco de dados. 7. ed. São Paulo: Érica, 2013. *E-book*.

SOEGAARD, Mads. **System Usability Scale for Data-Driven UX.** 2023. Disponível em: <https://www.interaction-design.org/literature/article/system-usability-scale?>. Acesso em: 03 jun. 2024.

SOMMERVILLE, Ian. **Engenharia de software.** 9. ed. São Paulo: Pearson, 2011.

VIANA, Jesiel. **Arquitetura de aplicações web.** 2023. Disponível em: <https://jesielviana.gitbook.io/guiaweb/apendices/i.-arquitetura-de-aplicacoes-web>. Acesso em: 03 jun. 2024.

VIDA, Edinilson da Silva. Arquitetura de banco de dados. In: PICHETTI, Roni Francisco; VIDA, Edinilson da Silva; CORTES, Vanessa Stangherlin Machado Paixão (Org.). **Banco de Dados.** Porto Alegre: SAGAH, 2020. *E-book.*

APÊNDICE A - QUESTIONÁRIO

1. Vocês possuem algum sistema de gestão de abrigos em casos de crises climáticas (como enchentes)? Se sim, poderiam descrever brevemente como ele funciona?

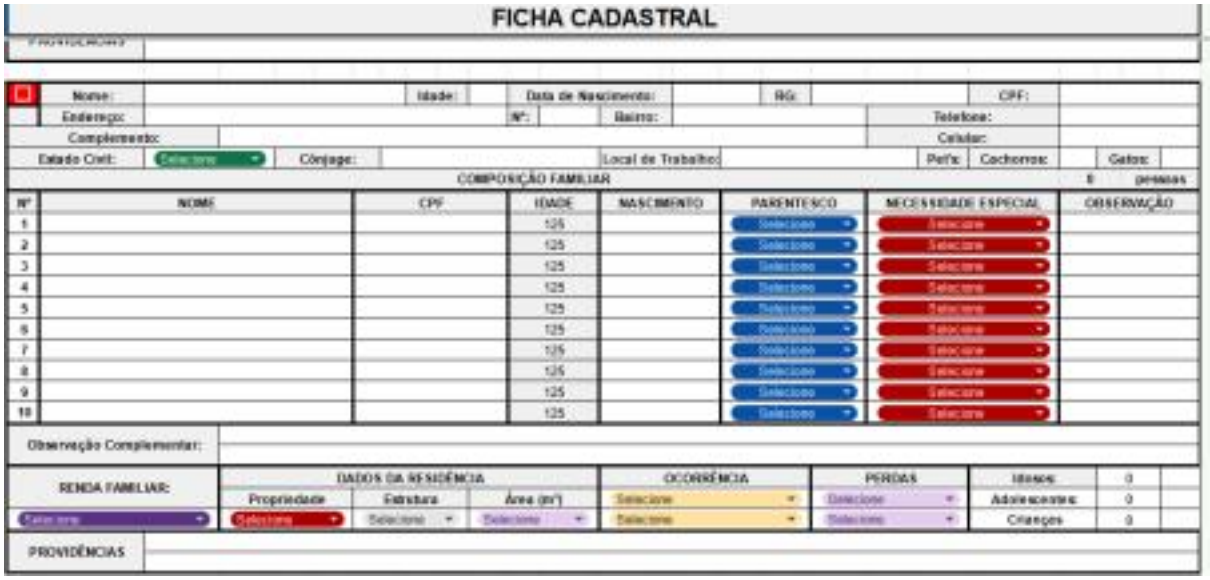
Resposta: A gestão dos abrigos acontece pelos seguintes passos:

a) A abertura de abrigos é feita pela Defesa Civil;

b) Após a abertura é repassado para a Secretaria de Assistência Social, a qual aciona os servidores que se dirigem para a coordenação dos abrigos;

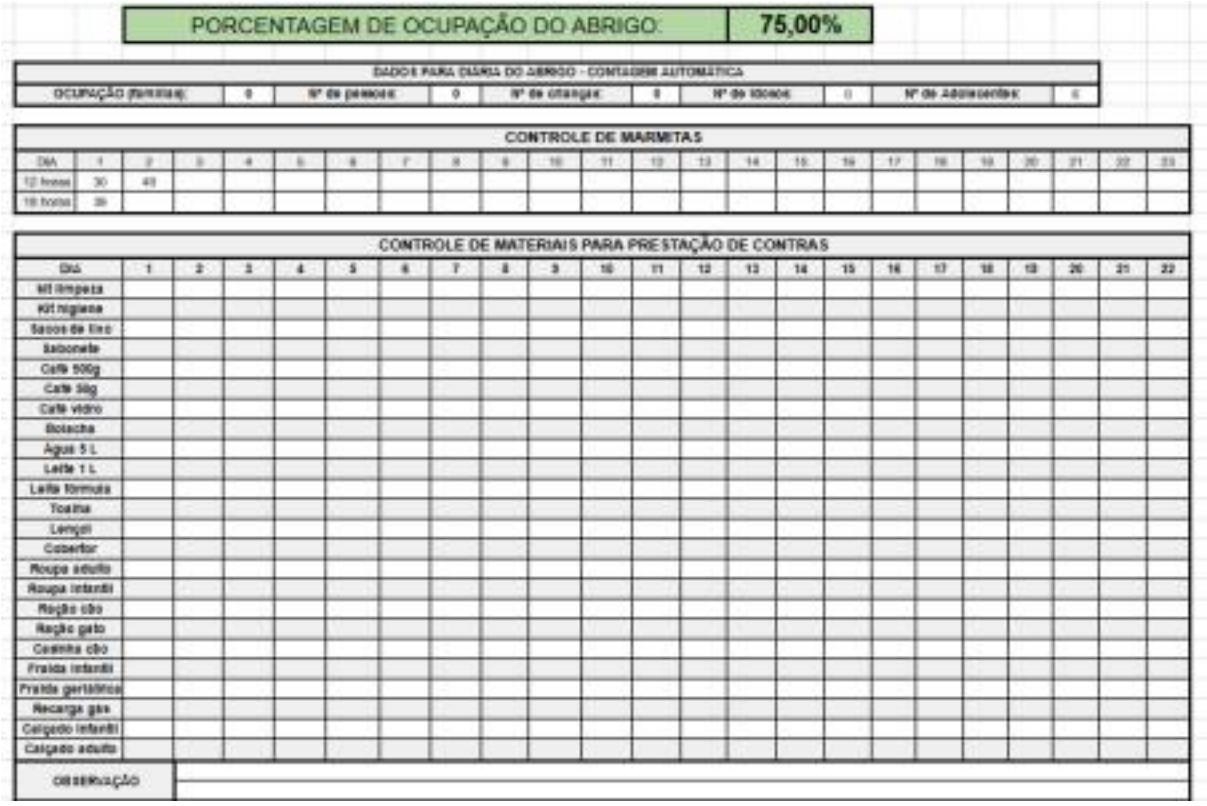
c) Os coordenadores recebem fichas de cadastro para preenchimento de dados como: nome, endereço, idade, CPF, telefone, estado civil, quantidade de membros familiares,

d) A partir disso, é feito o cadastro em planilhas Excel para controle de cada abrigo, como exposto na imagem abaixo:



e) O cadastro nos permite ter controle do número de pessoas, famílias, crianças, adolescentes, idosos e pets.

f) Além disso, quando um abrigo é aberto, fazemos o controle das demandas, tais como:



PORCENTAGEM DE OCUPAÇÃO DO ABRIGO:	75,00%

DADOS PARA DIÁRIA DO ABRIGO - CONTAGEM AUTOMÁTICA

OCUPAÇÃO (famílias):	0	Nº de pessoas:	0	Nº de crianças:	0	Nº de idosos:	0	Nº de Adolescentes:	0

CONTROLE DE MARMITAS

DIA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
12 horas	30	48																					
18 horas	39																						

CONTROLE DE MATERIAIS PARA PRESTAÇÃO DE CONTAS

Dia	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
kit limpeza																						
Kit higiene																						
Sacos de lixo																						
Sabonete																						
Café 500g																						
Café 50g																						
Café vidro																						
Bolacha																						
Água 5 L																						
Leite 1 L																						
Leite fórmula																						
Toalha																						
Lençol																						
Cobertor																						
Roupa adulto																						
Roupa infantil																						
Ração cão																						
Ração gato																						
Caminha cão																						
Fralda infantil																						
Fralda geriátrica																						
Recarga gás																						
Calçado infantil																						
Calçado adulto																						
classificação																						

2. Quais dificuldades vocês enfrentam no gerenciamento dos abrigos durante situações de problemas climáticos em Rio do Sul?

Resposta: A organização e contagem de doações; o controle da entrega dos benefícios advindos do estado. O repasse das informações dos coordenadores dos abrigos (número de pessoas, número de marmitas e demais demandas);

3. Como é controlada a quantidade de pessoas nos abrigos? Vocês fazem uma média ou contabilizam uma a uma?

Resposta: É feito uma contagem diária da quantidade de pessoas, a quantidade de famílias e a quantidade de crianças. Tanto para ter uma média da quantidade total de desabrigados, como para ter um controle de quantas famílias e pessoas chegam ou saem por dia. Essa contagem é feita duas vezes por dia (09h e 16h) para serem repassadas à Defesa Civil. A quantidade de pessoas que cada abrigo comporta está descrita no PLANCOM - Plano Municipal de Contingência de Rio do Sul da Defesa Civil (mas há abrigos que não têm esse registro completo). Em geral, na medida que o coordenador do abrigo observa que o local está lotado, é avisado a Defesa Civil, a qual direciona para os demais abrigos já abertos. Se o cenário da enchente se agrava, a Defesa Civil abre novos espaços e avisa à população para se direcionarem a eles.

4. O controle de pets das famílias abrigadas é realizado também? Se sim, como é feito esse controle?
Resposta: Atualmente fazemos a contagem de pets por famílias e registramos em planilha. Estamos construindo a possibilidade do Bem-estar animal vinculado a Secretaria de Saúde do município fazer um controle específico dos animais.

5. Vocês têm uma lista dos abrigos disponíveis? Caso sim, poderiam me fornecer essa lista?
Resposta: No site da Defesa Civil a o PLANCOM - Plano Municipal de Contingência de Rio do Sul. A partir da página 33 tem a listagem - https://drive.google.com/file/d/1SPtysJqCX1YJeUHFVV0Hja1WSec1sA-q/view

3. O controle dos abrigos é realizado também por voluntários? Se sim, qual é o papel deles nesse processo?
Resposta: Não, o gerenciamento dos abrigos é feito pela Secretaria de Assistência e Desenvolvimento Social do município.

7. Vocês têm uma contagem ou média de quantas pessoas utilizaram abrigos nos últimos 5 anos?
Resposta: Dos registros de 2019 até então, segue na contagem abaixo:

	Famílias	**Pessoas**
2019	48	158
2020	21	95
2022	162	566
2023	1.161	3.325
2024 (até agora)	207	738
Total:	1.599	4.880

8. Quanto de alimento, água, colchões e outros recursos são utilizados em um abrigo em média?
Resposta: As demandas de cada abrigo são avaliadas de acordo com as necessidades, com o período de duração de uma enchente e da proporção da enchente. Sempre que inicia a abertura de abrigos, é enviado kit de higiene e limpeza para o abrigo e kit de café e água para as

famílias. Na medida que a quantidade de famílias aumenta, estabelecemos entregas de marmitas no almoço e jantar. Itens relacionados a colchão, kit acomodação e outros, são solicitados para a Defesa Civil do Estado e entregues de acordo com a demanda. A média desses itens, portanto, varia de acordo com a quantidade de abrigos abertos e o tempo em que as pessoas ficam abrigadas.

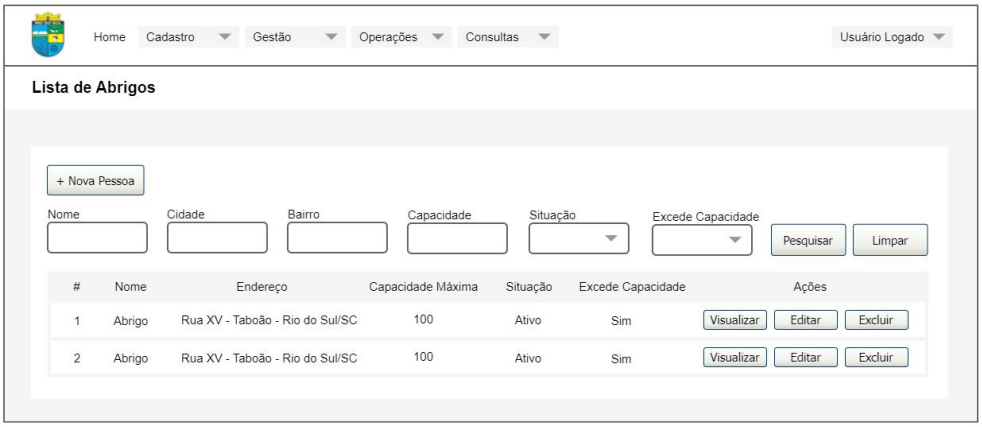
9. O que vocês acham importante ter em um sistema de gestão de abrigos para melhorar o gerenciamento durante crises climáticas?
Resposta: Cadastro de famílias, membros familiares. Levantamento de números e de demandas de cada abrigo; registros de necessidade e entrega de benefícios. Data de entrada e saída das famílias.

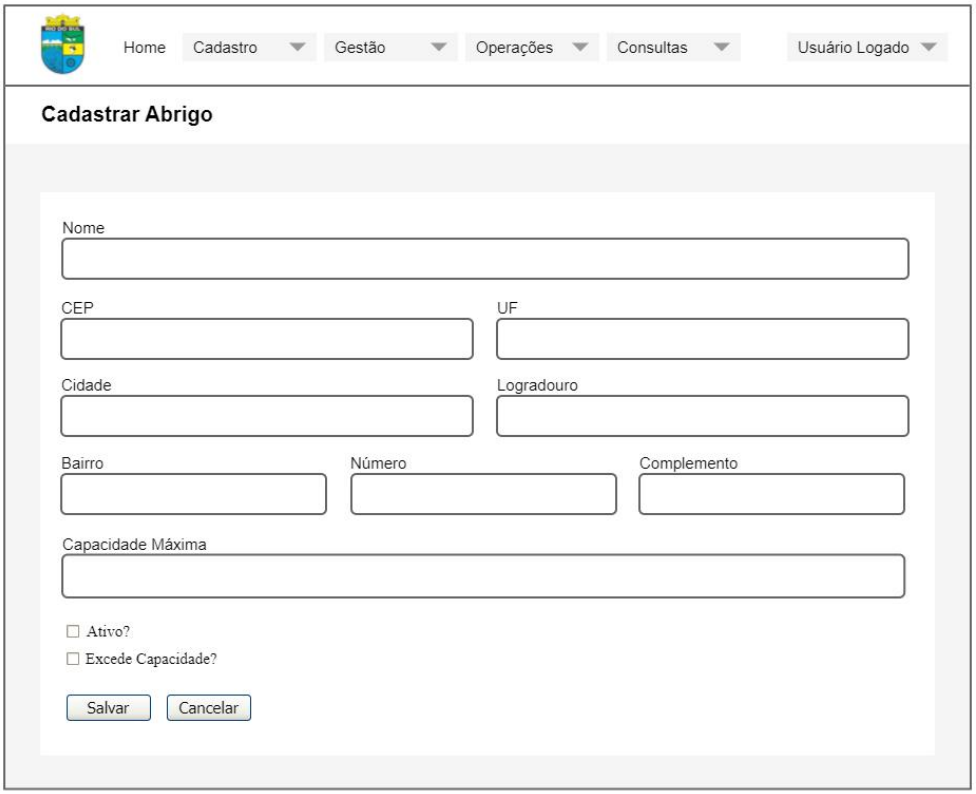
10. Há alguma informação adicional que não questionei e que vocês acham importante para o desenvolvimento do meu projeto?
Resposta: A Secretaria de Assistência Social gerencia tanto itens doados pela população e demais instituições, como os itens específicos da Defesa Civil do Estado que chegam para a população afetada. Quem faz o cadastro das famílias e entrega a doação concedida pelo Estado é a Secretaria de Assistência Social

APÊNDICE B - HISTÓRIAS DE USUÁRIO

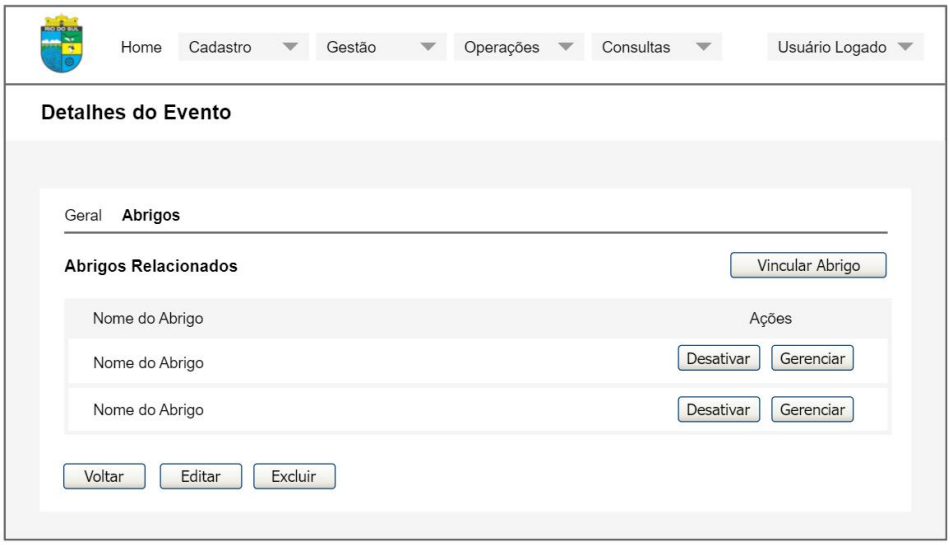
Quadro 16 - História de Usuário 4

US04 - Como usuário, preciso conseguir realizar o cadastro dos abrigos
Critérios de Aceite: US04.01 - A listagem de abrigos deve ter um botão para incluir um novo abrigo. US04.02 - O sistema deve validar o preenchimento de todos os campos obrigatórios. US04.03 - O registro do abrigo deve ser salvo e integrado ao banco de dados.
Requisitos: RF11
Protótipos:



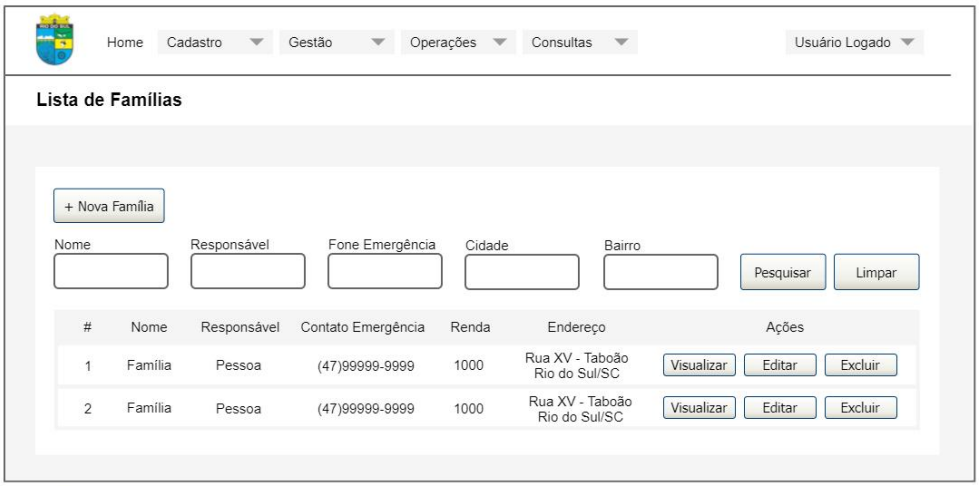


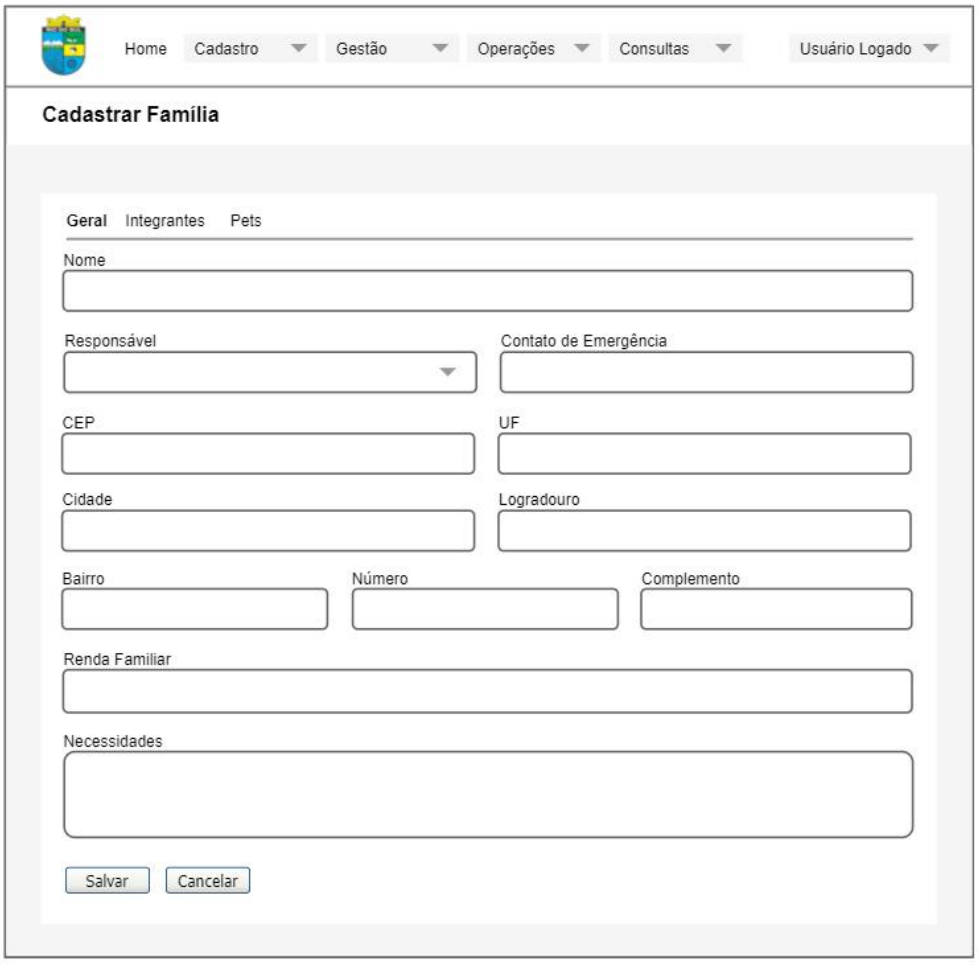
Quadro 17 - História de Usuário 5

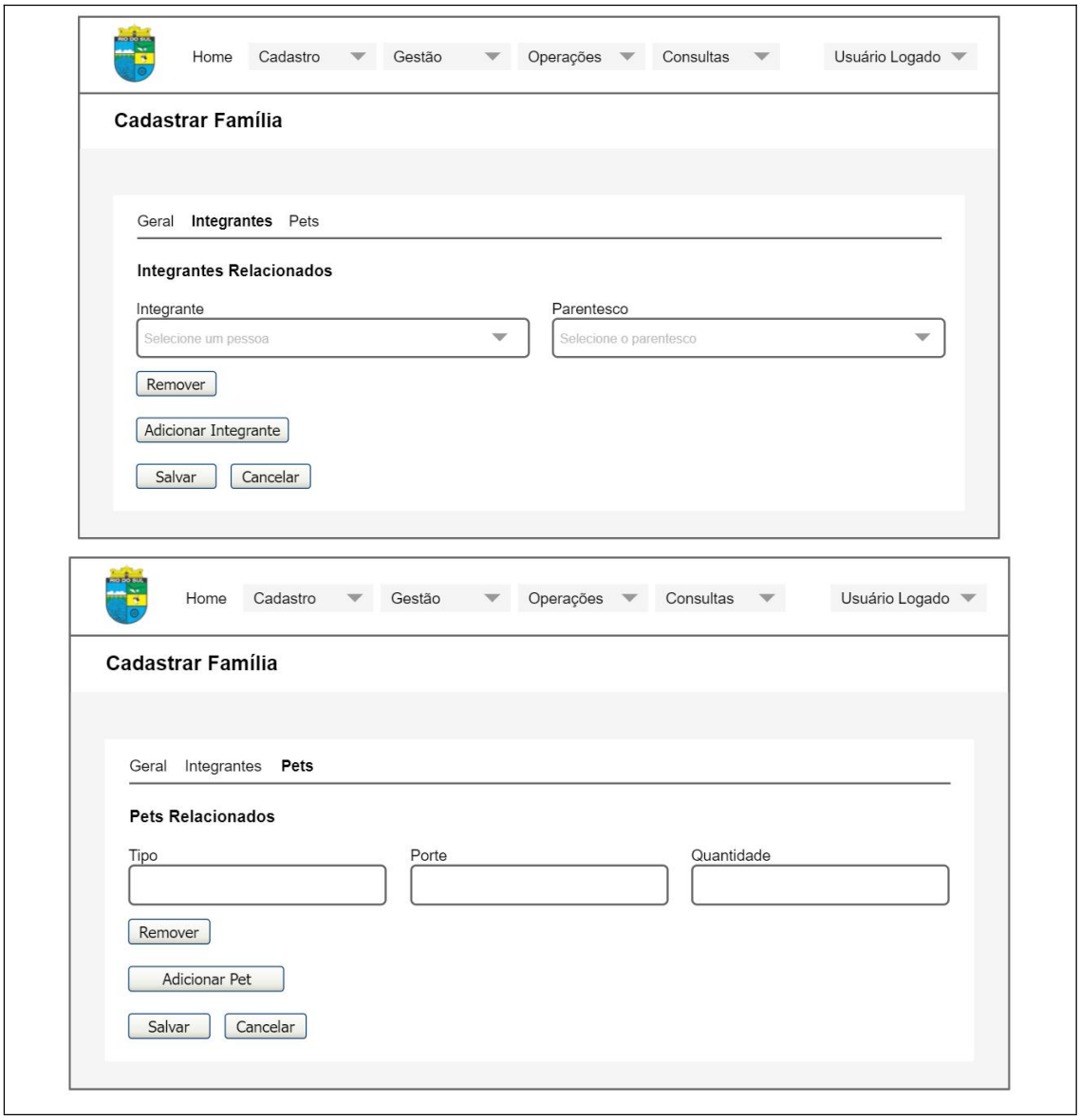
US05 - Como usuário, preciso conseguir gerenciar os abrigos do evento
Critérios de Aceite: US05.01 - Na visualização do evento, deve existir uma página específica para abrigos. US05.02 - Na página de abrigos dentro dos detalhes do evento, deve haver um botão para gerenciar cada abrigo. US05.03 - Ao clicar em "Gerenciar", o usuário deve ser direcionado para a página de detalhes do abrigo. US05.04 - Nos detalhes do abrigo, deve ser possível vincular as famílias relacionadas. US05.05 - O vínculo das famílias deve ser salvo e integrado ao banco de dados.
Requisitos: RF09, RF10
Protótipos:    

Quadro 18 - História de Usuário 6

US06 - Como usuário, preciso conseguir realizar o cadastro de famílias
Critérios de Aceite: US06.01 - A listagem de famílias deve ter um botão para incluir uma nova família. US06.02 - O sistema deve validar o preenchimento de todos os campos obrigatórios no cadastro. US06.03 - O responsável pela família deve ser maior de idade (18 anos ou mais). US06.04 - No cadastro da família, deve ser possível selecionar os integrantes. US06.05 - No cadastro da família, deve ser possível informar os pets. US06.06 - O registro da família deve ser salvo e integrado ao banco de dados.
Requisitos: RF06, RF07
Protótipos:



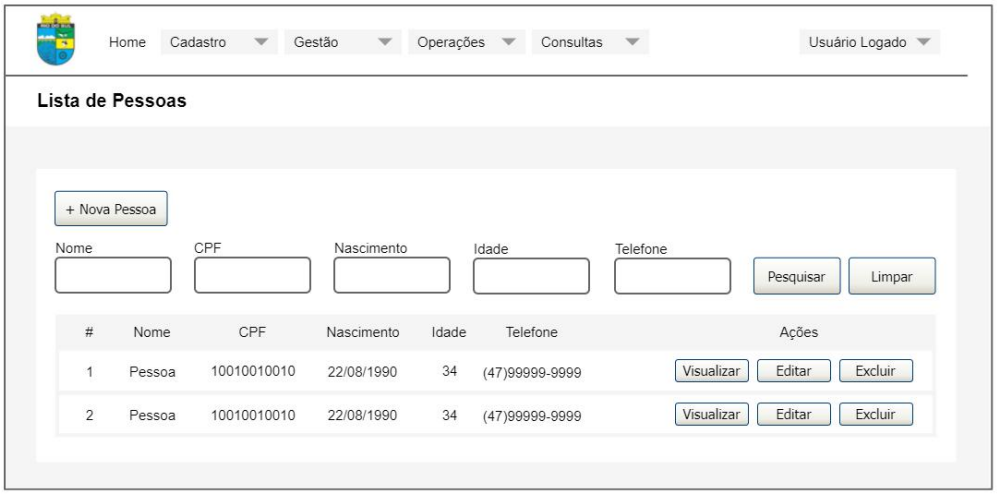


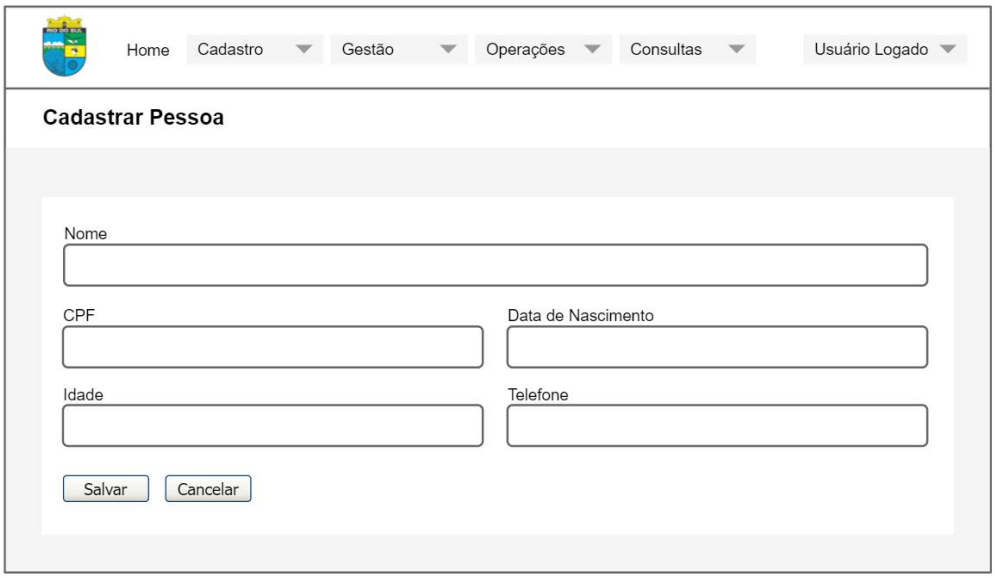


Quadro 19 - História de Usuário 7

US07 - Como usuário, preciso conseguir realizar o cadastro de pessoas
Critérios de Aceite: US07.01 - A listagem de pessoas deve ter um botão para incluir uma nova pessoa. US07.02 - O sistema deve validar o preenchimento de todos os campos obrigatórios durante o cadastro. US07.03 - O sistema não deve permitir o cadastro de duas pessoas com o mesmo CPF. US07.04 - O registro da pessoa deve ser salvo e integrado ao banco de dados.
Requisitos: RF04

Protótipos:

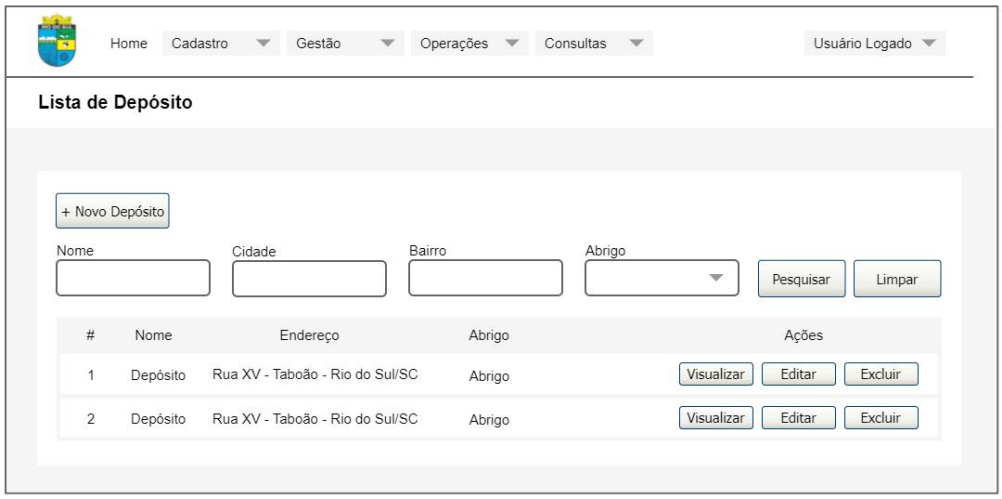


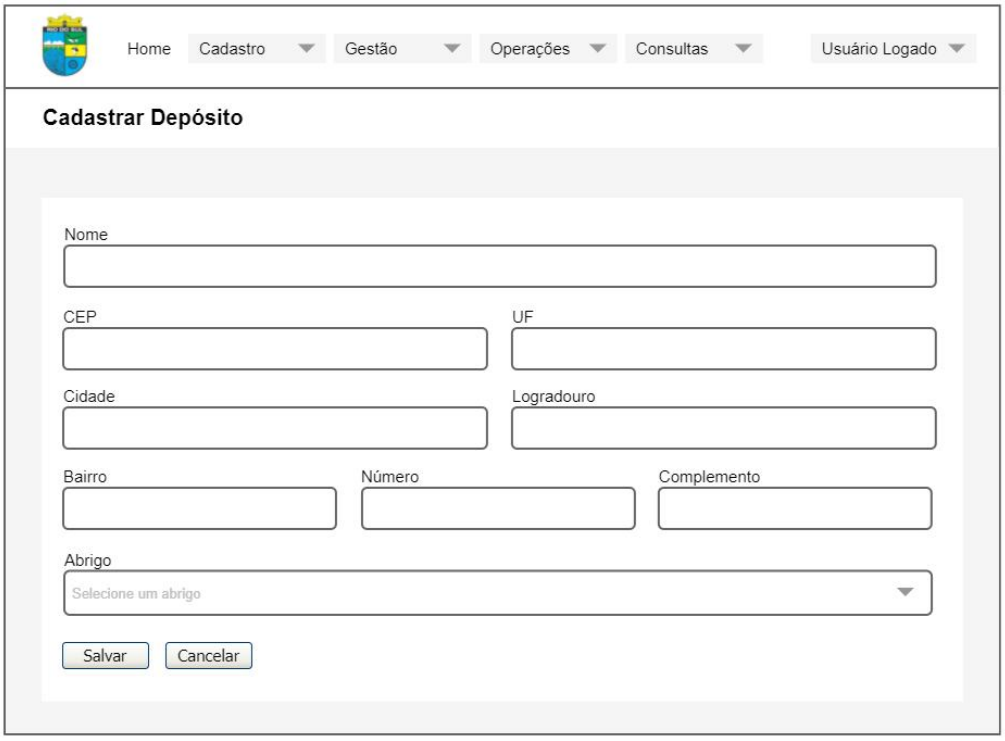


Quadro 20 - História de Usuário 8

US08 - Como usuário, preciso conseguir realizar o cadastro de depósitos
Critérios de Aceite: US08.01 - A listagem de depósitos deve ter um botão para incluir um novo depósito. US08.02 - O sistema deve validar o preenchimento de todos os campos obrigatórios durante o cadastro. US08.03 - O registro de depósito deve ser salvo e integrado ao banco de dados.
Requisitos: RF12

Protótipos:

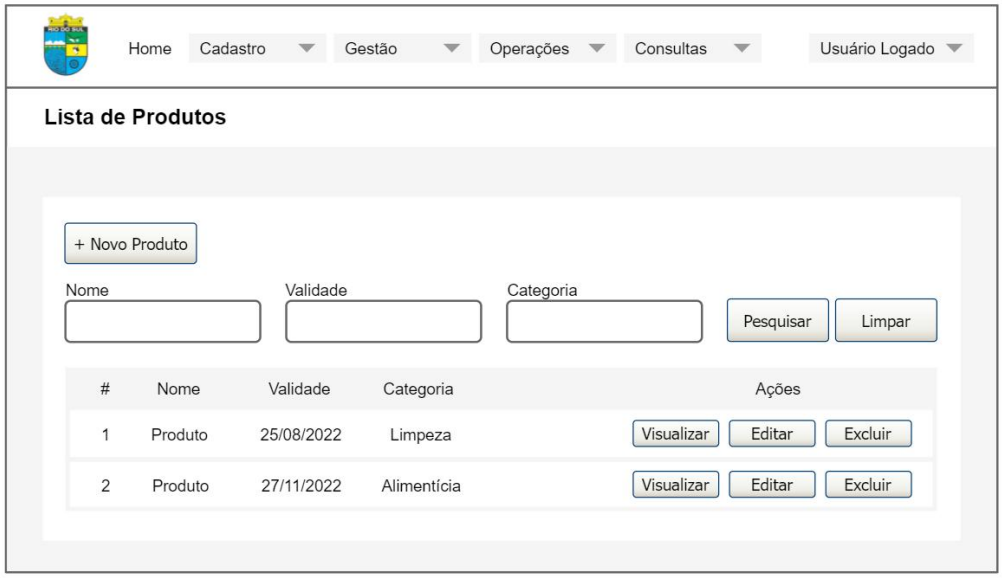


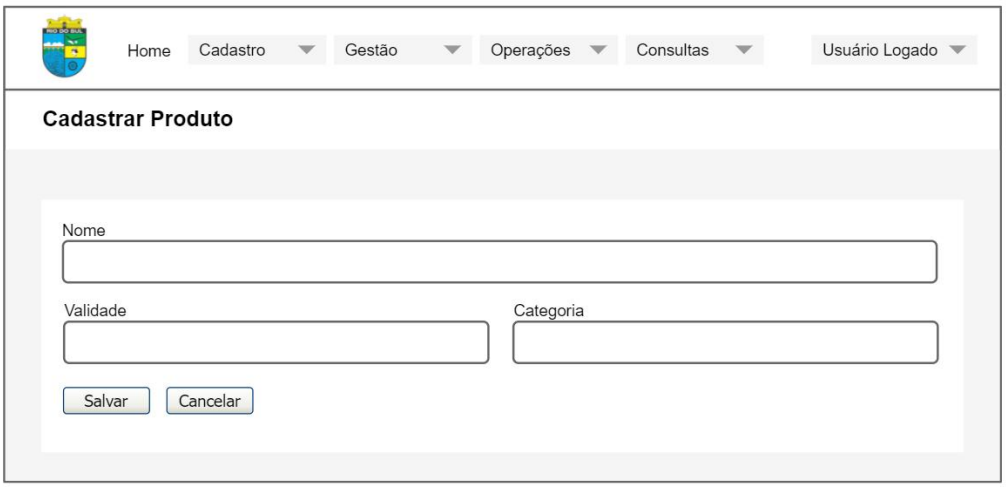


Quadro 21 - História de Usuário 9

US09 - Como usuário, preciso conseguir realizar o cadastro de produtos.
Critérios de Aceite: US09.01 - A listagem de produtos deve ter um botão para incluir um novo produto. US09.02 - O sistema deve validar o preenchimento de todos os campos obrigatórios durante o cadastro. US09.03 - O registro de produto deve ser salvo e integrado ao banco de dados.
Requisitos: RF05

Protótipos:

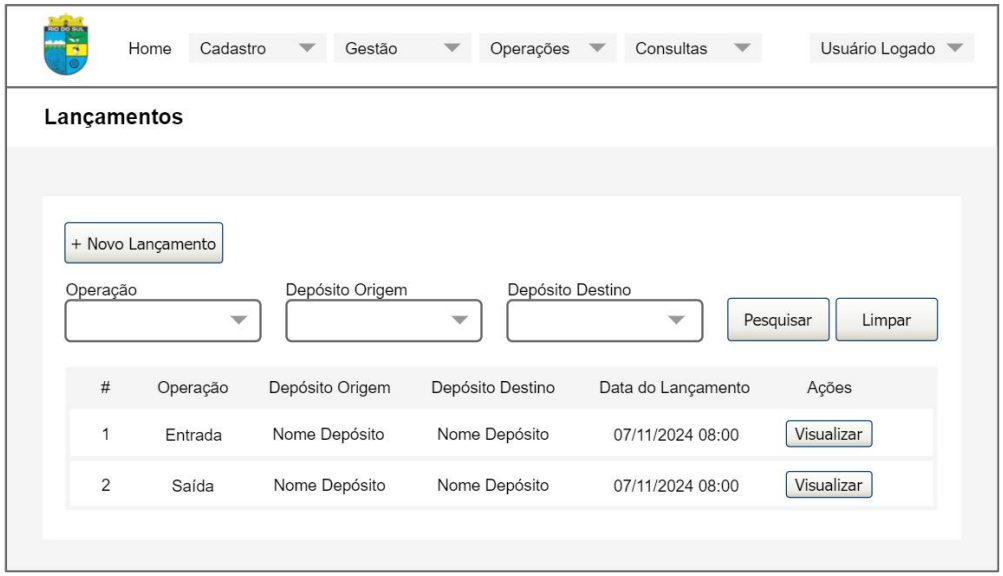


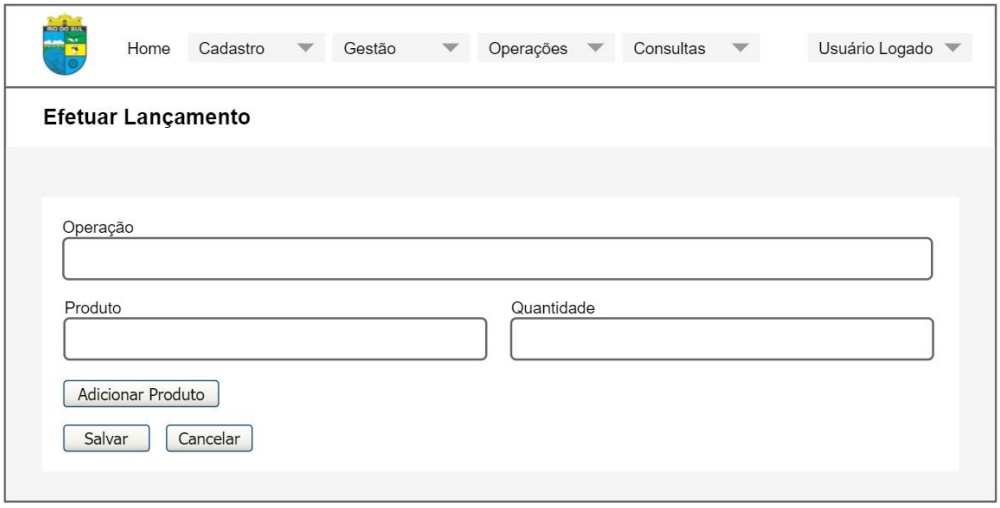


Quadro 22 - História de Usuário 10

US10 - Como usuário, preciso conseguir realizar o lançamento de produtos.
Critérios de Aceite: US10.01 - A listagem de lançamentos deve ter um botão para incluir um novo lançamento. US10.02 - O sistema deve validar o preenchimento de todos os campos obrigatórios no cadastro. US10.03 - Se a operação do lançamento for "entrada", o campo de depósito de destino deve ser habilitado, permitindo também marcar se o lançamento é uma doação. US10.04 - Se a operação do lançamento for "saída", o campo de depósito de origem deve ser habilitado. US10.05 - Se a operação do lançamento for "transferência", ambos os campos de depósito de origem e destino devem ser habilitados. US10.06 - O registro de lançamentos deve ser salvo e integrado ao banco de dados.
Requisitos: RF14

Protótipos:

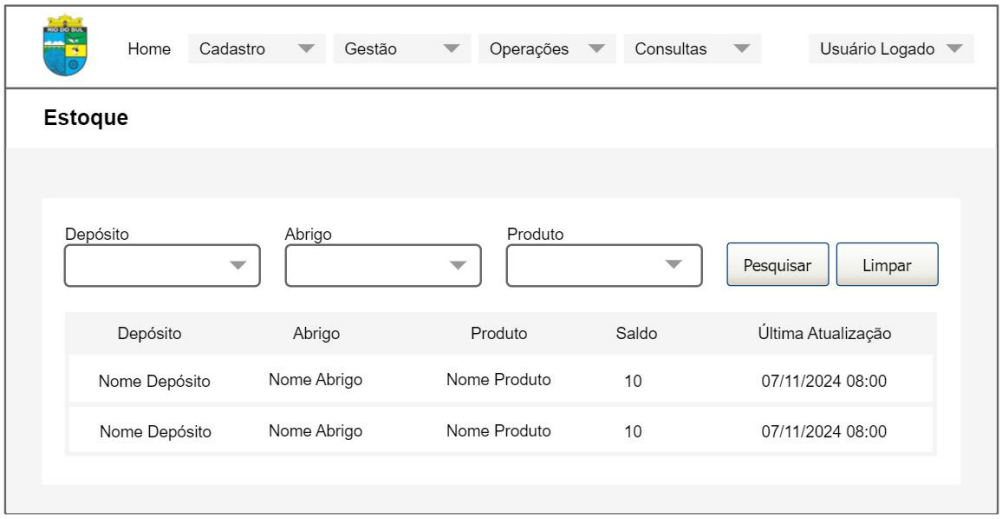




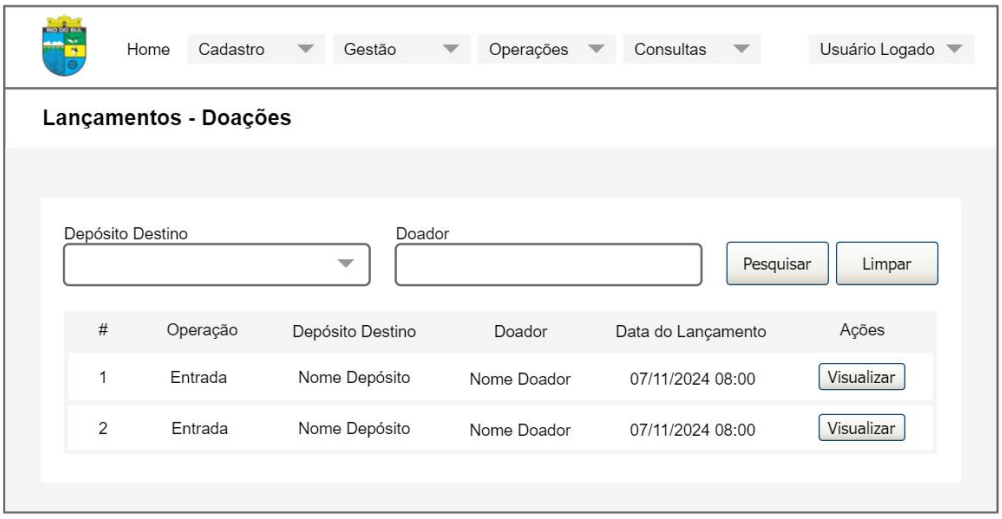
Quadro 23 - História de Usuário 11

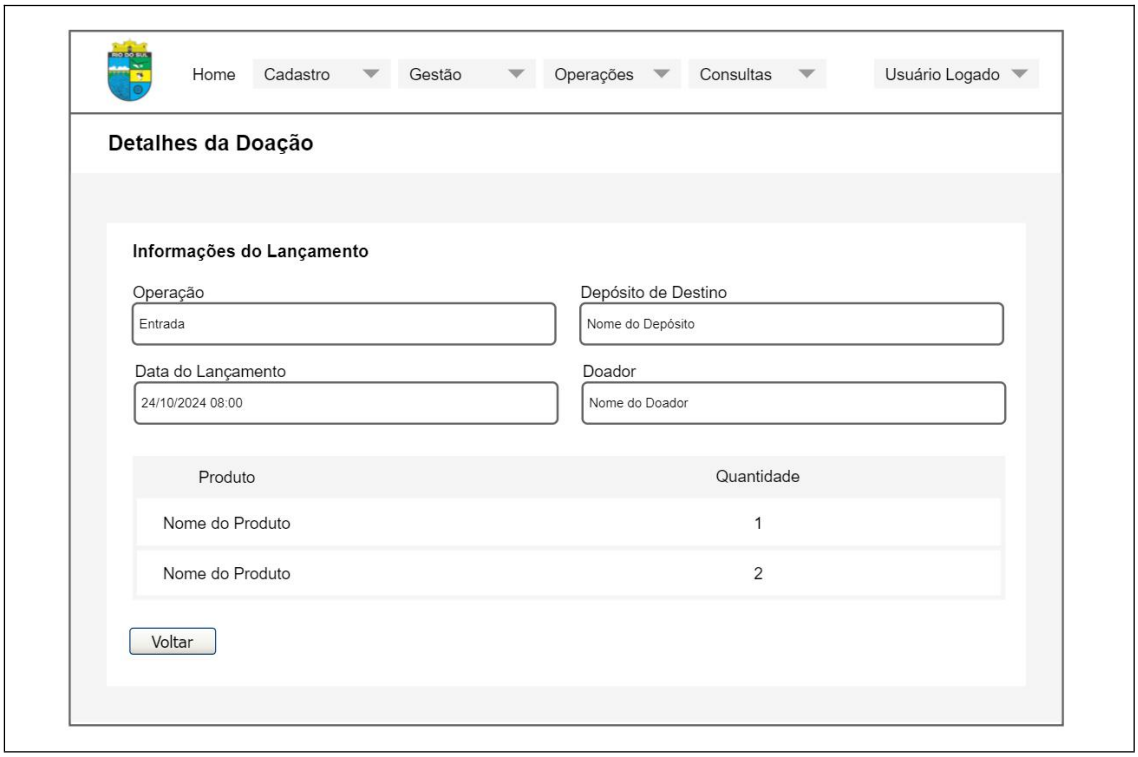
US11 - Como usuário, preciso conseguir consultar o estoque de produtos.
Critérios de Aceite: US11.01 - A consulta de estoque deve dispor de filtros para depósito, abrigo e produto, permitindo refinar a busca. US11.02 - O sistema deve exibir a quantidade disponível de cada produto conforme os filtros aplicados. US11.03 - A consulta de estoque deve ser integrada ao banco de dados, garantindo que as informações estejam sempre atualizadas.
Requisitos: RF15

Protótipos:

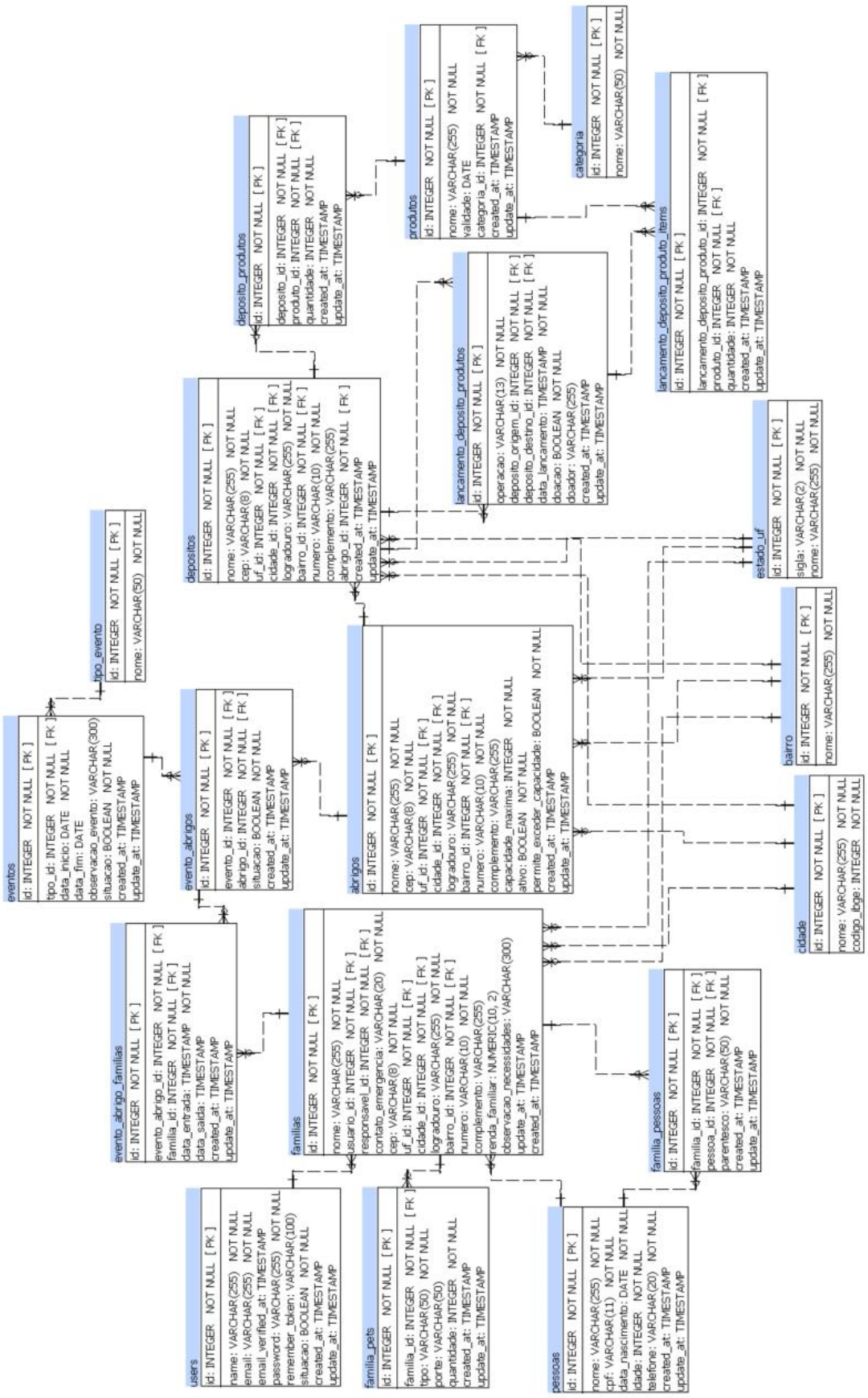


Quadro 24 - História de Usuário 12

US12 - Como usuário, preciso conseguir consultar as doações.
Critérios de Aceite: US12.01 - A consulta de doações deve incluir filtros para depósito de destino e doador, permitindo refinar a busca. US12.02 - Na visualização da doação, o sistema deve exibir todas as informações da doação, incluindo os itens recebidos. US12.03 - A consulta de doações deve ser integrada ao banco de dados, garantindo que as informações estejam sempre atualizadas.
Requisitos: RF16
Protótipos: 


Home
Cadastro
Gestão
Operações
Consultas
Usuário Logado
Detalhes da Doação
Informações do Lançamento
Operação
Entrada
Depósito de Destino
Nome do Depósito
Data do Lançamento
24/10/2024 08:00
Doador
Nome do Doador
Produto
Quantidade
Nome do Produto
1
Nome do Produto
2
Voltar

APÊNDICE C - DIAGRAMA DE ENTIDADES E RELACIONAMENTO



APÊNDICE D - SCRIPTS PARA CRIAÇÃO DE TABELAS

Figura 39 - Criar Tabela de Famílias

```
CREATE TABLE IF NOT EXISTS public.familias
(
    id bigint NOT NULL DEFAULT nextval('familias_id_seq'::regclass),
    nome character varying(255) COLLATE pg_catalog."default" NOT NULL,
    usuario_id bigint NOT NULL,
    responsavel_id bigint NOT NULL,
    contato_emergencia character varying(20) COLLATE pg_catalog."default" NOT NULL,
    cep character varying(8) COLLATE pg_catalog."default" NOT NULL,
    uf character varying(2) COLLATE pg_catalog."default" NOT NULL,
    cidade character varying(255) COLLATE pg_catalog."default" NOT NULL,
    logradouro character varying(255) COLLATE pg_catalog."default" NOT NULL,
    bairro character varying(255) COLLATE pg_catalog."default" NOT NULL,
    numero character varying(10) COLLATE pg_catalog."default" NOT NULL,
    complemento character varying(255) COLLATE pg_catalog."default",
    renda_familiar numeric(10,2),
    observacao_necessidades character varying(300) COLLATE pg_catalog."default",
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT familias_pkey PRIMARY KEY (id),
    CONSTRAINT familias_responsavel_id_foreign FOREIGN KEY (responsavel_id)
    CONSTRAINT familias_usuario_id_foreign FOREIGN KEY (usuario_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 39, o *script* cria a tabela familias no banco de dados. Essa tabela armazena informações sobre as famílias, incluindo nome, dados de contato, endereço, renda familiar e observações de necessidades. Além disso, ela possui chaves estrangeiras para os campos responsavel_id e usuario_id, que vinculam a família a um responsável e a um usuário específico no sistema.

Figura 40 - Criar Tabela de Pessoas

```
CREATE TABLE IF NOT EXISTS public.pessoas
(
    id bigint NOT NULL DEFAULT nextval('pessoas_id_seq'::regclass),
    nome character varying(255) COLLATE pg_catalog."default" NOT NULL,
    cpf character varying(11) COLLATE pg_catalog."default" NOT NULL,
    data_nascimento date NOT NULL,
    idade integer NOT NULL,
    telefone character varying(20) COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT pessoas_pkey PRIMARY KEY (id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 40, o *script* cria a tabela pessoas no banco de dados. Essa tabela armazena informações básicas sobre as pessoas, incluindo nome, CPF, data de nascimento, idade e telefone.

Figura 41 - Criar Tabela de Pets da Família

```
CREATE TABLE IF NOT EXISTS public.familia_pets
(
    id bigint NOT NULL DEFAULT nextval('familia_pets_id_seq'::regclass),
    familia_id bigint NOT NULL,
    tipo character varying(50) COLLATE pg_catalog."default" NOT NULL,
    porte character varying(50) COLLATE pg_catalog."default",
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    quantidade integer NOT NULL,
    CONSTRAINT familia_pets_pkey PRIMARY KEY (id),
    CONSTRAINT familia_pets_familia_id_foreign FOREIGN KEY (familia_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 41, o *script* cria a tabela familia_pets no banco de dados. Essa tabela armazena informações sobre os pets de cada família, incluindo o tipo, porte e quantidade de pets. Ela também possui uma chave estrangeira (familia_id) que vincula o pet à família correspondente no sistema.

Figura 42 - Criar Tabela de Usuários

```
CREATE TABLE IF NOT EXISTS public.users
(
    id bigint NOT NULL DEFAULT nextval('users_id_seq'::regclass),
    name character varying(255) COLLATE pg_catalog."default" NOT NULL,
    email character varying(255) COLLATE pg_catalog."default" NOT NULL,
    email_verified_at timestamp(0) without time zone,
    password character varying(255) COLLATE pg_catalog."default" NOT NULL,
    remember_token character varying(100) COLLATE pg_catalog."default",
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    situacao boolean NOT NULL DEFAULT false,
    CONSTRAINT users_pkey PRIMARY KEY (id),
    CONSTRAINT users_email_unique UNIQUE (email)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 42, o *script* cria a tabela users no banco de dados. Essa tabela armazena informações dos usuários, incluindo nome, email, senha, situação e tokens de verificação e autenticação. A tabela possui uma restrição de unicidade no campo email, garantindo que cada endereço de email seja único.

Figura 43 - Criar Tabela de Produtos

```
CREATE TABLE IF NOT EXISTS public.produtos
(
    id bigint NOT NULL DEFAULT nextval('produtos_id_seq'::regclass),
    nome character varying(255) COLLATE pg_catalog."default" NOT NULL,
    validade date,
    categoria character varying(50) COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT produtos_pkey PRIMARY KEY (id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 43, o *script* cria a tabela produtos no banco de dados. Essa tabela armazena informações sobre produtos, incluindo nome, data de validade e categoria.

Figura 44 - Criar Tabela de Depósitos

```
CREATE TABLE IF NOT EXISTS public.depositos
(
    id bigint NOT NULL DEFAULT nextval('depositos_id_seq'::regclass),
    nome character varying(255) COLLATE pg_catalog."default" NOT NULL,
    cep character varying(8) COLLATE pg_catalog."default" NOT NULL,
    uf character varying(2) COLLATE pg_catalog."default" NOT NULL,
    cidade character varying(255) COLLATE pg_catalog."default" NOT NULL,
    logradouro character varying(255) COLLATE pg_catalog."default" NOT NULL,
    bairro character varying(255) COLLATE pg_catalog."default" NOT NULL,
    numero character varying(10) COLLATE pg_catalog."default" NOT NULL,
    complemento character varying(255) COLLATE pg_catalog."default",
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    abrigo_id bigint,
    CONSTRAINT depositos_pkey PRIMARY KEY (id),
    CONSTRAINT depositos_abrigo_id_foreign FOREIGN KEY (abrigo_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 44, o *script* cria a tabela depositos no banco de dados. Essa tabela armazena informações de localização dos depósitos, incluindo endereço completo, e possui uma chave estrangeira (abrigo_id) que vincula o depósito ao abrigo correspondente.

Figura 45 - Criar Tabela de Abrigos do Evento

```
CREATE TABLE IF NOT EXISTS public.evento_abrigos
(
    id bigint NOT NULL DEFAULT nextval('evento_abrigos_id_seq'::regclass),
    evento_id bigint NOT NULL,
    abrigo_id bigint NOT NULL,
    situacao boolean NOT NULL DEFAULT true,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT evento_abrigos_pkey PRIMARY KEY (id),
    CONSTRAINT evento_abrigos_abrigo_id_foreign FOREIGN KEY (abrigo_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 45, o *script* cria a tabela evento_abrigos no banco de dados. Essa tabela armazena informações de associação entre abrigos e eventos, incluindo os identificadores do evento e do abrigo, além de uma coluna booleana (situacao) que indica a situação da associação, com valor padrão true. A tabela também possui chaves estrangeiras que vinculam o registro aos respectivos evento_id e abrigo_id.

Figura 46 - Criar Tabela de Famílias dos Abrigos do Evento

```
CREATE TABLE IF NOT EXISTS public.evento_abrigo_familias
(
    id bigint NOT NULL DEFAULT nextval('evento_abrigo_familias_id_seq'::regclass),
    evento_abrigo_id bigint NOT NULL,
    familia_id bigint NOT NULL,
    data_entrada timestamp(0) without time zone NOT NULL,
    data_saida timestamp(0) without time zone,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT evento_abrigo_familias_pkey PRIMARY KEY (id),
    CONSTRAINT evento_abrigo_familias_evento_abrigo_id_foreign FOREIGN KEY (evento_abrigo_id)
    CONSTRAINT evento_abrigo_familias_familia_id_foreign FOREIGN KEY (familia_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 46, o *script* cria a tabela evento_abrigo_familias no banco de dados. Essa tabela armazena informações sobre as famílias abrigadas durante eventos, incluindo os identificadores do evento e do abrigo, bem como o identificador da família. Ela também registra as datas de entrada e saída da família no abrigo, além de conter chaves estrangeiras para vincular o registro ao evento, ao abrigo e à família correspondentes.

Figura 47 - Criar Tabela de Pessoas da Família

```
CREATE TABLE IF NOT EXISTS public.familia_pessoas
(
    id bigint NOT NULL DEFAULT nextval('familia_pessoas_id_seq'::regclass),
    familia_id bigint NOT NULL,
    pessoa_id bigint NOT NULL,
    parentesco character varying(50) COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT familia_pessoas_pkey PRIMARY KEY (id),
    CONSTRAINT familia_pessoas_familia_id_pessoa_id_unique UNIQUE (familia_id, pessoa_id),
    CONSTRAINT familia_pessoas_familia_id_foreign FOREIGN KEY (familia_id)
    CONSTRAINT familia_pessoas_pessoa_id_foreign FOREIGN KEY (pessoa_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 47, o *script* cria a tabela familia_pessoas no banco de dados. Essa tabela armazena informações sobre os membros de cada família, incluindo o identificador da família e o da pessoa, além de um campo para indicar o parentesco. A tabela possui chaves estrangeiras para vincular cada membro à família correspondente e uma restrição de unicidade para impedir a duplicação de registros para a mesma combinação de família e pessoa.

Figura 48 - Criar Tabela de Produtos do Depósito

```
CREATE TABLE IF NOT EXISTS public.deposito_produtos
(
    id bigint NOT NULL DEFAULT nextval('deposito_produtos_id_seq'::regclass),
    deposito_id bigint NOT NULL,
    produto_id bigint NOT NULL,
    quantidade integer NOT NULL,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT deposito_produtos_pkey PRIMARY KEY (id),
    CONSTRAINT deposito_produtos_deposito_id_produto_id_unique UNIQUE (deposito_id, produto_id),
    CONSTRAINT deposito_produtos_deposito_id_foreign FOREIGN KEY (deposito_id)
    CONSTRAINT deposito_produtos_produto_id_foreign FOREIGN KEY (produto_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 48, o *script* cria a tabela deposito_produtos no banco de dados. Essa tabela armazena informações sobre os produtos presentes em cada depósito, incluindo o identificador do depósito, o identificador do produto e a quantidade armazenada. Ela possui chaves estrangeiras para vincular cada registro ao depósito e ao produto correspondentes, além de uma restrição de unicidade que evita duplicação de registros para a mesma combinação de depósito e produto.

Figura 49 - Criar Tabela de Lançamentos

```
CREATE TABLE IF NOT EXISTS public.lancamento_deposito_produtos
(
    id bigint NOT NULL DEFAULT nextval('lancamento_deposito_produtos_id_seq'::regclass),
    operacao character varying(13) COLLATE pg_catalog."default" NOT NULL,
    deposito_origem_id bigint,
    deposito_destino_id bigint,
    data_lancamento timestamp(0) without time zone NOT NULL,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    doacao boolean NOT NULL DEFAULT false,
    doador character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT lancamento_deposito_produtos_pkey PRIMARY KEY (id),
    CONSTRAINT lancamento_deposito_produtos_deposito_destino_id_foreign FOREIGN KEY (deposito_destino_id)
    CONSTRAINT lancamento_deposito_produtos_deposito_origem_id_foreign FOREIGN KEY (deposito_origem_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 49, o *script* cria a tabela lancamento_deposito_produtos no banco de dados. Essa tabela armazena informações sobre os lançamentos de produtos entre depósitos, incluindo o tipo de operação, depósito de origem, depósito de destino, data do lançamento e, se aplicável, o nome do doador. Ela também possui uma coluna booleana para indicar se o lançamento é uma doação, além de chaves estrangeiras para vincular cada lançamento aos depósitos de origem e destino.

Figura 50 - Criar Tabela de Lançamento de Itens

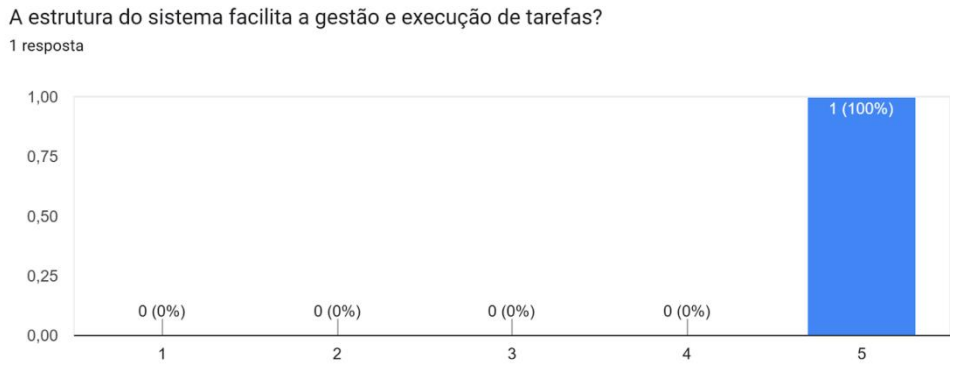
```
CREATE TABLE IF NOT EXISTS public.lancamento_deposito_produto_items
(
    id bigint NOT NULL DEFAULT nextval('lancamento_deposito_produto_items_id_seq'::regclass),
    lancamento_deposito_produto_id bigint NOT NULL,
    produto_id bigint NOT NULL,
    quantidade integer NOT NULL,
    created_at timestamp(0) without time zone,
    updated_at timestamp(0) without time zone,
    CONSTRAINT lancamento_deposito_produto_items_pkey PRIMARY KEY (id),
    CONSTRAINT lancamento_deposito_produto_items_produto_id_foreign FOREIGN KEY (produto_id)
)
```

Fonte: Acervo do Autor (2024).

Conforme mostrado na Figura 50, o *script* cria a tabela lancamento_deposito_produto_items no banco de dados. Essa tabela registra os itens específicos em cada lançamento de depósito, incluindo o identificador do lançamento, o identificador do produto e a quantidade movimentada. Ela possui chaves estrangeiras para vincular o registro ao lançamento e ao produto correspondente.

APÊNDICE E - VALIDAÇÃO E RESULTADOS

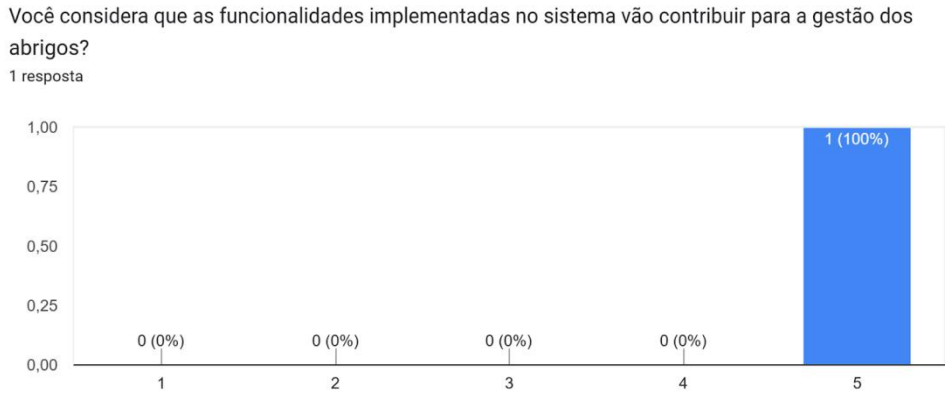
Figura 51 - Pergunta 03 do Questionário



Fonte: Acervo do Autor (2024).

Na avaliação da Figura 51, referente à estrutura do sistema e sua capacidade de facilitar a gestão e execução de tarefas, a resposta obteve nota máxima (5). Isso indica que os usuários consideram a estrutura do sistema muito eficaz para apoiar na organização e realização das tarefas necessárias, reforçando sua utilidade como uma ferramenta de suporte na gestão de abrigos e famílias durante crises climáticas.

Figura 52 - Pergunta 04 do Questionário

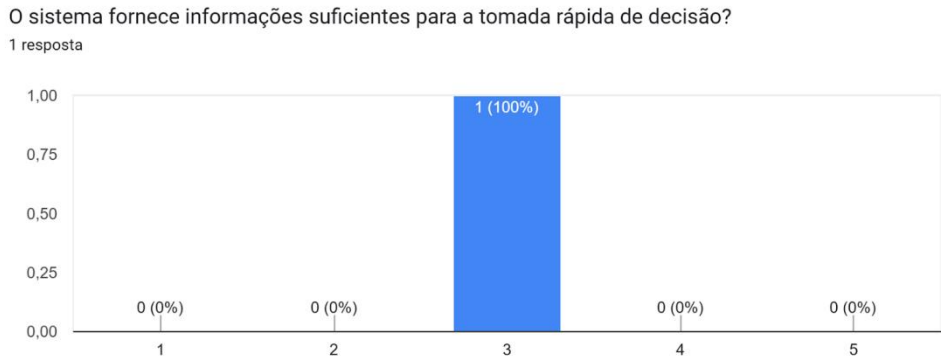


Fonte: Acervo do Autor (2024).

Na avaliação da Figura 52, sobre a contribuição das funcionalidades implementadas no sistema para a gestão dos abrigos, a resposta obteve nota máxima (5). Esse resultado sugere que os usuários acreditam fortemente que as funcionalidades desenvolvidas irão colaborar de

maneira significativa para a administração dos abrigos, tornando o sistema um recurso valioso para apoiar as atividades de gestão em momentos de crise climática.

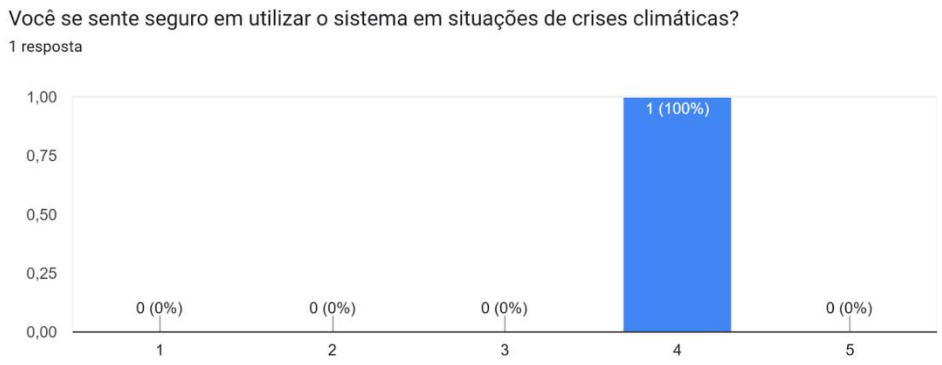
Figura 53 - Pergunta 05 do Questionário



Fonte: Acervo do Autor (2024).

Na avaliação da Figura 53, sobre a capacidade do sistema de fornecer informações suficientes para a tomada rápida de decisão, a resposta obteve uma nota 3. Esse resultado indica uma percepção neutra dos usuários quanto à eficácia do sistema para apoiar decisões rápidas, sugerindo que ele considera o sistema útil, mas com espaço para aprimoramentos na disponibilização de informações para tomada de decisão em tempo hábil.

Figura 54 - Pergunta 06 do Questionário

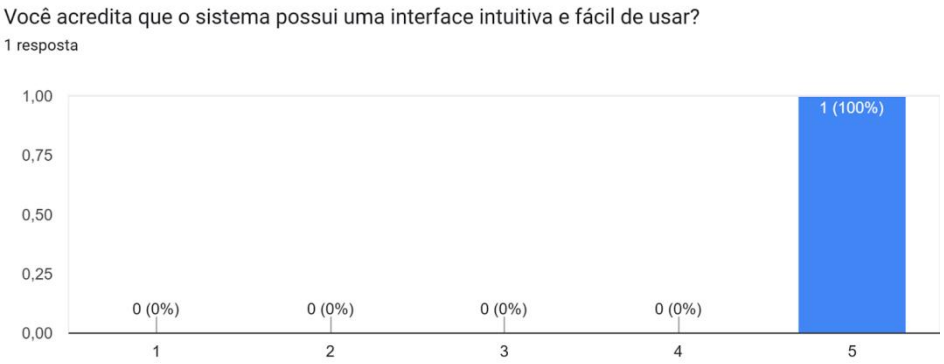


Fonte: Acervo do Autor (2024).

Na avaliação da Figura 54, sobre a segurança em utilizar o sistema em situações de crises climáticas, a resposta obteve nota 4. Esse resultado indica que os usuários se sentem confiantes ao usar o sistema nessas circunstâncias, embora identifique que ainda há espaço

para melhorias para alcançar um nível máximo de segurança. Essa percepção positiva sugere que o sistema já oferece uma boa base de suporte para uso em cenários críticos.

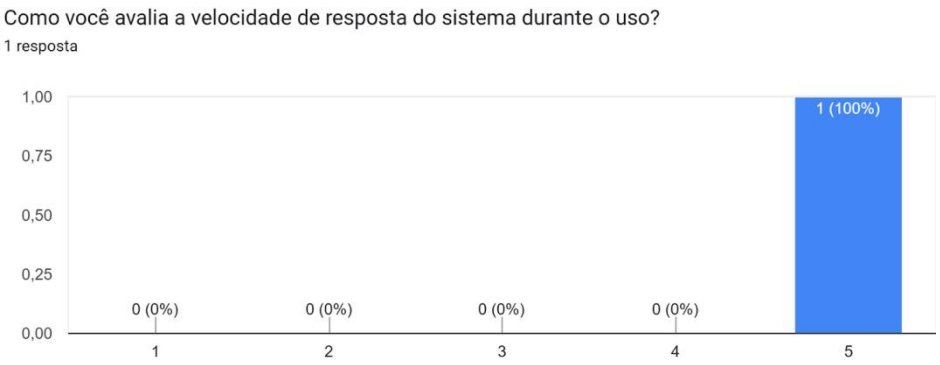
Figura 55 - Pergunta 07 do Questionário



Fonte: Acervo do Autor (2024).

Na avaliação da Figura 55, sobre a interface do sistema, a pergunta abordou se o usuário considera o sistema intuitivo e fácil de usar, resultando em nota máxima (5). Esse feedback indica que os usuários percebem a interface como muito acessível e amigável, o que contribui positivamente para a experiência de uso e a eficácia do sistema no suporte às atividades de gestão de abrigos em situações de crise climática.

Figura 56 - Pergunta 08 do Questionário

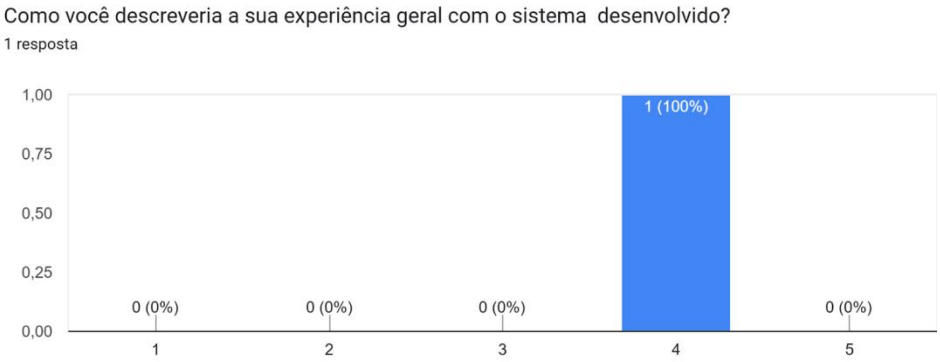


Fonte: Acervo do Autor (2024).

Na avaliação da Figura 56, sobre a velocidade de resposta do sistema durante o uso, a pergunta recebeu nota máxima (5). Esse resultado indica que os usuários consideram o

sistema rápido e responsivo, atendendo bem às expectativas de desempenho. Essa característica é essencial para garantir uma experiência de uso fluida e eficiente, especialmente em situações de crise, onde a agilidade é fundamental.

Figura 57 - Pergunta 09 do Questionário



Fonte: Acervo do Autor (2024).

Na avaliação da Figura 57, sobre a experiência geral com o sistema desenvolvido, a resposta recebeu nota 4. Esse resultado demonstra que os usuários tiveram uma experiência positiva com o sistema, embora haja uma margem para aprimoramentos que poderiam elevar ainda mais a satisfação. A pontuação indica que o sistema atende bem às necessidades principais, proporcionando uma experiência satisfatória para o usuário.