UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUCAS DANIEL BATISTA

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB NO CONTEXTO DE GERENCIAMENTO DE DOAÇÕES

TOLEDO

2022

LUCAS DANIEL BATISTA

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB NO CONTEXTO DE GERENCIAMENTO DE DOAÇÕES

Development of a WEB application in the context of donations management

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Alexandre Huff

TOLEDO

2022

LUCAS DANIEL BATISTA

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB NO CONTEXTO DE GERENCIAMENTO DE DOAÇÕES

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 29/11/2022

Alexandre Huff
Dr.
Universidade Tecnológica Federal do Paraná

Rosane Fátima Passarini
Dra.
Universidade Tecnológica Federal do Paraná

Marcelo Alexandre da Cruz Ismael
MSc.
Universidade Tecnológica Federal do Paraná

TOLEDO

2022

RESUMO

Com o intuito de aplicar os conteúdos e conhecimentos adquiridos em sala de aula durante o curso, este trabalho descreve o desenvolvimento de um sistema Web que utiliza *Web Services* com estilo arquitetural REST (*REpresentational State Transfer*) para disponibilizar *endpoints* para manipulação dos recursos que serão exibidos em uma interface Web. A aplicação desenvolvida tem o objetivo de controlar o recebimento e distribuição de doações de alimentos, roupas e outros itens pela prefeitura de Ouro Verde do Oeste/PR, possibilitando que os próprios doadores possam informar à prefeitura que desejam doar e também tornando transparente todas as doações, uma vez que a população poderá conferir o que tem sido distribuído. Atualmente a prefeitura controla as doações para famílias necessitadas de maneira manual, com documentos e formulários impressos que ficam arquivados, dificultando a visibilidade e controle do processo, realização de auditorias, além de necessitar mão de obra considerável por parte dos funcionários.

Palavras-chave: rest; serviço web; api; doação.

ABSTRACT

In order to apply the knowledge acquired in the classroom throughout my undergraduate, a Web system was developed employing a Web Service using the *REpresentational State Transfer* (REST) architectural style. This Web Service provides endpoints that allow the manipulation of resources that are shown in a Web interface. The goal of the developed application mainly includes controlling the reception and distribution of donations of food, clothes, and other items by the City Hall of Ouro Verde do Oeste/PR. The developed system allows the donors themselves to inform the City Hall that they want to donate items and allows the citizens to check what is being distributed and donated to the population under vulnerability. Currently, the donation control is done manually by the City Hall, which still uses documents and printed forms that need to be filled out by hand, resulting in significant efforts to provide visibility and control of the process. It also results in hurdles to performing auditing while requiring a large workforce of the City Hall employees.

Keywords: rest; web service; api; donation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

Siglas

API	*Application Programming Interface*
CSS	*Cascading Style Sheet*
HTML	*Hyper Text Markup Language*
HTTP	*Hypertext Transfer Protocol*
JPQL	*Java Persistence Query Language*
OMG	*Object Management Group*
PDF	*Portable Document Format*
SGBD	Sistema de Gerenciamento de Banco de Dados
SPA	*Single Page Application*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*

Acrônimos

CEP	Código de Endereçamento Postal
CRUD	*Create, Read, Update e Delete*
JSON	*JavaScript Object Notation*
REST	*REpresentational State Transfer*

SUMÁRIO

1 INTRODUÇÃO

A comunicação entre dois ou mais sistemas pode ser feita de maneira segura e padronizada utilizando *Web Services* REST (*REpresentational State Transfer*), quando tanto cliente quanto servidor implementam os conceitos REST. Em geral, *Web Services* são aplicações que permitem a comunicação entre dois sistemas na Web (PAPAZOGLOU, 2008), enquanto o REST reúne um conjunto de padrões arquiteturais para qualidade e expansividade de um *Web Service* (FIELDING, 2000).

Este trabalho aborda o desenvolvimento de uma aplicação utilizando *Web Services* REST que disponibiliza *endpoints* para manipulação dos recursos da aplicação por meio de uma *Application Programming Interface* (API). Para se comunicar com a API, a interface do usuário foi desenvolvida como uma aplicação à parte usando um *framework Web* que será descrito posteriormente e como base para o desenvolvimento, foram necessárias atividades de análise de software, como levantamento de requisitos e construção de diagramas. A aplicação desenvolvida é responsável por gerenciar as coletas e distribuições de doações de itens essenciais à famílias necessitadas, que são realizadas por uma prefeitura.

1.1 Objetivo Geral

Desenvolver uma aplicação que utilize uma API REST para comunicação entre o lado cliente e o lado servidor. A aplicação servirá para controle e gerenciamento de doações recebidas e distribuídas por prefeituras.

1.2 Justificativa

Na cidade de Ouro Verde do Oeste, existem famílias em situação financeira delicada e que dependem de doações de roupas, calçados, alimentos e outros itens essenciais, que atualmente já são distribuídos pela prefeitura. Dessa forma, em conversa com o prefeito de Ouro Verde do Oeste/PR, juntamente com a secretaria do trabalho (atualmente responsável por controlar as doações para famílias necessitadas) foi encontrada a oportunidade de automatizar o processo através de um sistema. Atualmente o controle é realizado via formulários físicos que ficam armazenados em arquivos, sendo necessária consulta manual toda vez que uma família necessita de alguma doação.

A aplicação a ser desenvolvida, servirá como facilitador e tornará o controle de coletas e distribuição dessas doações mais fácil e ágil pelos funcionários da prefeitura. Também possibilitará que as pessoas interessadas em realizar doações possam o fazê-las por conta própria, acessando a aplicação Web e registrando sua intenção de doação. Somado à isso, relatórios

com todas as doações efetuadas pela prefeitura podem ser gerados tanto para auditoria quanto para prestação de contas com a população.

1.3 Estrutura do trabalho

A estrutura deste trabalho está organizada da seguinte maneira: O Capítulo 2 trata da apresentação dos conceitos e conteúdos que serão utilizados no desenvolvimento do projeto. Já o Capítulo 3 descreve os diagramas, tecnologias, arquiteturas, *frameworks* e as etapas do projeto seguidas para alcançar o resultado final do trabalho. O Capítulo 4 busca apresentar todos os artefatos obtidos durante as especificação, análise e implementação do sistema. Por fim, o Capítulo 5 discute as considerações finais do trabalho.

2 REFERENCIAL TEÓRICO

Uma série de boas práticas, orientações e conceitos de programação, desenvolvimento e análise de software foram seguidas para o desenvolvimento deste trabalho de conclusão de curso. Este capítulo descreve o referencial teórico utilizado no processo de análise e desenvolvimento da aplicação.

2.1 UML

A *Unified Modeling Language* (UML) é um conjunto de representações gráficas, criadas para a documentação e representação visual de softwares, em especial os orientados a objetos. A UML possui diversos padrões e estilos de representações com objetivos distintos mas que compartilham uma ideia em comum: A comunicação e o entendimento, transmitir a ideia, as funcionalidades e todos os detalhes de um projeto de maneira unificada e simples (FOWLER, 2005).

A UML é um padrão moderadamente aberto, controlado pelo consórcio aberto de empresas *Object Management Group* (OMG) que sugere padrões para o desenvolvimento de sistemas orientados a objetos. A UML surgiu em 1997 como unificação de muitas linguagens gráficas para modelagem orientada a objetos. De acordo com FOWLER (2005), existem 13 tipos de diagramas oficiais que são descritos pela UML em sua versão mais recente, dos quais 4 foram escolhidos para este trabalho, sendo eles: Casos de uso, Classes, Sequência e de Comunicação.

2.2 Princípios arquiteturais REST

A integração entre sistemas é uma forma rápida e prática de unir funcionalidades de duas aplicações diferentes. No contexto de sistemas Web, APIs são ótimas escolhas para essa comunicação. É importante adotar algum estilo arquitetural para definir a estrutura do *Web Service*, bem como estabelecer um acordo na comunicação entre cliente e servidor, para que a integração ocorra com sucesso. Fielding (2000) propôs o REST (REpresentational State Transfer) e descreveu uma série de princípios e restrições arquitetônicas para o desenvolvimento de aplicações distribuídas. Basicamente, o REST aborda os princípios que são descritos nas subseções seguintes.

2.2.1 Cliente-servidor

No paradigma cliente-servidor, a separação desses dois contextos resulta em uma série de vantagens, tais como, ganhos em testes, escalabilidade e integrações entre sistemas.

O conceito deste princípio está na separação de responsabilidades, também conhecido como *Separation of concerns*. Separando as finalidades de apresentação e manipulação dos dados, há um ganho na portabilidade, pois permite que a interface de usuário esteja disponível em diversas plataformas (FIELDING, 2000).

2.2.2 Comportamento *Stateless*

Implica que o Servidor não pode armazenar dados da sessão do usuário. O armazenamento dos dados de usuário é de responsabilidade de uma entidade externa, que pode ser no próprio cliente ou em uma base de dados externa acessada pelo servidor. O servidor trata apenas de disponibilizar e alterar o estado desses dados. Outra característica importante, é que as requisições devem conter todas as informações necessárias para sua execução, além do uso correto do protocolo escolhido (FIELDING, 2000).

2.2.3 Interface Uniforme

A interface uniforme é a principal característica que identifica um Web Service REST. Primeiramente, qualquer informação que o Web Service disponibiliza para clientes remotos, é chamada de recurso. Recursos podem ser representados em diversos formatos, como JSON (*JavaScript Object Notation*), *Portable Document Format* (PDF) ou *eXtensible Markup Language* (XML) e são endereçados através de um identificador único, o *Uniform Resource Identifier* (URI). A representação de um recurso nada mais é do que a exibição de suas propriedades em um determinado momento.

A interface uniforme é uma forma de proporcionar desacoplamento entre as aplicações cliente e servidor, respeitando a semântica do protocolo utilizado. Comumente é escolhido o protocolo *Hypertext Transfer Protocol* (HTTP) e cabe ao servidor o bom uso dos verbos HTTP e seus códigos de estado (FIELDING, 2000).

2.2.4 Controles Hipermídia

Com foco na representação de recursos, os controles hipermídia são facilitadores para manipulação dos recursos. Em vez de se utilizar um JSON puro, pode-se utilizar diferentes representações para os recursos, possibilitando diferenciar texto puro de um *link*, por exemplo. Um recurso disponibilizado pelo Web Service que contém links para outros recursos e que não foi representado com controles hipermídia, fará a aplicação cliente ter dificuldades na navegação. Desta forma, será necessário um trabalho manual do desenvolvedor da aplicação cliente para identificar links de navegação. Ainda assim, caso a estrutura dos recursos seja modificada no servidor, o cliente poderá não funcionar corretamente (FIELDING, 2000).

2.3 Arquitetura de software

Durante o desenvolvimento de um software, existem algumas práticas que devem ser evitadas ao máximo para que o projeto seja duradouro, passível de alterações e que sua manutenção não seja extremamente complicada, ou mesmo impossível de ser realizada no futuro. O uso de práticas incorretas pode acarretar em problemas que podem ser reunidos em:

- Ao alterar um trecho de código, outro trecho totalmente distinto acaba quebrando.
- Não é possível fazer alterações no sistema, pois gera um número de dependências muito grande.
- Existem funcionalidades duplicadas, gerando pontos de manutenção distintos com o mesmo objetivo.

Na maioria dos casos, esses problemas têm a mesma origem, a maneira como se está definida a interdependência entre os componentes do software. Quando isso acontece, o sistema possui um alto acoplamento, ou seja, suas classes dependem muito umas das outras. Uma das formas de evitar o acoplamento com a dependência entre as classes é a utilização de interfaces. As interfaces contém somente a assinatura dos métodos, enquanto as classes de implementação que de fato contém a lógica dos métodos, implementam as interfaces. Dessa forma, as interfaces servem como um contrato, que define quais métodos devem ser implementados nas classes. Caso seja necessário alterar a implementação, o contrato definido pela interface não é modificado (WEISSMANN, 2014).

O conceito de “Inversão de controle” surge para auxiliar na remoção dos problemas de acoplamento. De maneira geral, a inversão de controle consiste em mover a responsabilidade de verificar a ocorrência de eventos para um nível superior, alguém que conhece o sistema como um todo, deixando o programador livre para focar na lógica de negócio. Como uma especialização da inversão de controle, existe a “Injeção de dependências”. Na injeção de dependências, não é a classe cliente, ou seja, a classe que está utilizando alguma outra classe externa, a responsável por instanciar suas dependências, isso deve ser feito por algum outro componente. (WEISSMANN, 2014).

3 MATERIAIS E MÉTODOS

Este capítulo descreve as técnicas, diagramas, tecnologias, linguagens de programação, arquiteturas, *frameworks*, bibliotecas e escolhas feitas para que fosse possível alcançar o resultado final do trabalho.

3.1 Materiais

A escolha das tecnologias utilizadas no desenvolvimento da aplicação foi baseada nos conceitos de arquitetura de software e princípios REST. Na aplicação servidora foi utilizada a linguagem Java com o *framework* Spring Boot, para construção do Web Service. O Spring Boot aplica os conceitos de inversão de controle e injeção de dependência mencionados anteriormente. Além da base para o projeto, o Spring também fornece uma série de bibliotecas auxiliares, como o *Lombok*, que automatiza a criação de *getter* e *setters* nas entidades, Spring *Validation*, responsável por facilitar validações dos atributos de uma entidade, como atributos obrigatórios e padrão esperado para cada tipo de informação (SPRING, 2022a).

Ainda dentro do Spring, foi utilizada a biblioteca Spring *Data* JPA (*Java Persistence API*), responsável pela camada de persistência da aplicação comunicando-se diretamente com o banco de dados e oferecendo uma série de interfaces para facilitar o desenvolvimento na camada de persistência do software (SPRING, 2022b).

Para persistência dos dados, foi escolhido o Sistema de Gerenciamento de Banco de Dados (SGBD) relacional PostgreSQL por atender as necessidades da aplicação servidora, além de ser *open source* e de fácil utilização/manutenção (POSTGRESQL, 2022).

No *front end*, foi utilizado o *framework Typescript Angular*, para desenvolvimento de *Single Page Application* (SPA)s, tendo controle para as rotas da aplicação, organização em módulos e componentes, suporte a injeção de dependências, entre outros (LLC, 2022).

O processo de análise e documentação do software, por sua vez, precisou da definição de duas ferramentas para montagem dos diagramas da UML que serão apresentados no Capítulo 4. As plataformas escolhidas foram o Visual Paradigm Online e o Astah UML.

3.2 Métodos

3.2.1 Levantamento requisitos

Consistiu em duas reuniões juntamente à prefeitura de Ouro Verde do Oeste, a fim de entender como é realizado o controle das doações, mostrar as vantagens de ter uma aplicação que automatiza o processo e sugerir melhorias na rotina de coletas e entregas, que poderiam ser proporcionadas com a aplicação a ser desenvolvida. Durante as reuniões, foram coletadas

as informações necessárias para especificar os requisitos necessários para o desenvolvimento do software, os quais serão descritos no Capítulo 4.

3.2.2 Especificação dos casos de uso

Após levantar os requisitos, foi necessário definir as funcionalidades e fluxos do sistema. As funcionalidades foram documentadas e especificadas em um diagrama, com as relações entre elas, chamado Diagrama de Casos de Uso e um documento com a descrição individual de cada um dos casos de uso, chamado de Especificação de Casos de Uso. O objetivo dos casos de uso é obter uma visão macro do sistema e também um detalhamento de cada funcionalidade que deve ser executada.

3.2.3 Construção do diagrama de classes

A camada de modelo é a base para o desenvolvimento do sistema, pois serão os recursos manipulados pela aplicação. O diagrama de classes consiste em mapear cada classe do sistema, especificando quais são seus atributos e métodos, além das relações entre cada uma das classes. Esse diagrama foi construído após o diagrama de casos de uso, pois somente nesse momento tem-se conhecimento das necessidades técnicas da aplicação e de como as informações manipuladas pelos casos de uso devem ser encapsuladas e registradas pelo *software*.

3.2.4 Construção do diagrama de sequência

Consiste no mapeamento das interações entre as partes do *software*. As partes podem ser tanto pessoas, como classes, o usuário, um servidor, o próprio sistema, dentre outros. As interações são comumente troca de mensagens e/ou dados. O diagrama também especifica quais são os eventos que disparam cada uma das ações durante o ciclo de vida das partes que interagem para executar um determinado caso de uso do sistema. No contexto da aplicação a ser desenvolvida, o diagrama de sequência é importante para definir, por exemplo, qual a resposta do sistema quando um doador realiza um agendamento de doação ou quando a própria instituição faz esse mesmo processo.

3.2.5 Desenvolvimento da aplicação

Com todos os documentos mencionados anteriormente construídos, o desenvolvimento foi iniciado. O processo consistiu em desenvolver primeiramente o *Web Service* REST com um projeto Java Maven, utilizando o *framework* Spring Boot, que tem o objetivo disponibilizar *end-*

points necessários para todas as funcionalidades previstas. Com as *Uniform Resource Locator* (URL)s prontas e funcionando, foi iniciado o desenvolvimento da aplicação *front end* utilizando o *framework Angular* que faz o consumo das URLs disponibilizadas pelo *back end*.

3.2.6 Testes da aplicação

Consistiu em baterias de testes realizadas tanto na interface do usuário, simulando o uso final, como também de testes das interfaces REST utilizando a ferramenta de requisições Web Postman. O objetivo foi garantir que a aplicação como um todo esteja funcionando corretamente e executando exatamente o que foi especificado na etapa de análise, mantendo a integridade das regras de negócio e sem apresentar erros.

4 RESULTADOS

Como resultado das atividades e métodos descritos anteriormente, foi gerada uma série de artefatos, tais como, levantamento de requisitos, diagramas, documentação em forma textual e executáveis do software. Estes resultados serão descritos neste capítulo.

4.1 Escopo do sistema

O sistema gerenciador de doações deverá controlar o processo desde a intenção de doação de um doador até a entrega dos itens aos beneficiários. O sistema será disponibilizado em um site Web e poderá ser acessado em qualquer navegador atual tanto pelos funcionários da prefeitura quanto por doadores que podem se registrar e ter acesso restrito às funcionalidades de doação no sistema.

O sistema possibilitará ao doador se cadastrar e agendar uma doação. As doações podem ser coletadas pela prefeitura na residência do doador, ou este pode se comprometer em entregá-las na prefeitura. Isso permite à prefeitura cadastrar lembretes de doação, avisos de falta de itens específicos e principalmente, é um facilitador para prestação de contas à comunidade tendo em vista que qualquer doador poderá consultar as doações que estão sendo realizadas pela prefeitura.

Por fim, quando um beneficiário necessitar de alguma doação, ele poderá se dirigir à prefeitura onde será cadastrado pelo funcionário responsável. Assim, o beneficiário poderá receber a doação desde que esteja nos critérios configurados a ele ou às restrições da prefeitura.

Não caberá ao sistema permitir acesso aos beneficiários ou rastreamento de famílias necessitadas. Este processo já existente na prefeitura. O sistema contemplará o gerenciamento de doações, o qual consiste no beneficiário se dirigir à prefeitura para receber alguma doação quando é de sua necessidade.

4.2 Modelagem do sistema

4.2.1 Levantamento de requisitos

4.2.1.1 Requisitos funcionais - Acesso

- RF01 - O sistema deverá permitir que os usuários se cadastrem e loguem via usuário e senha para ter acesso às funcionalidades do sistema.
- RF02 - O sistema deverá permitir o registro de doadores ou instituição distribuidora ao acessar o sistema, vinculando o login a esse registro.

- RF03 - O sistema deverá possibilitar o cadastro inicial dos dados da instituição no primeiro acesso da aplicação.

4.2.1.2 Requisitos funcionais - Operacional

- RF04 - O sistema deverá permitir que doadores registrem suas doações;
- RF05 - O sistema deverá permitir que a prefeitura também registre doações caso o doador se dirija à ela;
- RF06 - O sistema deverá ter uma opção que indique se a doação foi concluída. Somente assim a doação será contabilizada;
- RF07 - O sistema deverá permitir indicar o tipo de item que está sendo doado, seja ele alimento, roupa, dentre outros;
- RF08 - O sistema deverá permitir o cadastro de lembretes de doações que ao ser agendado irá disparar um e-mail ao doador;
- RF09 - O sistema deverá permitir que a instituição informe no cadastro dos itens a sua quantidade mínima. Quando essa quantidade mínima for atingida os usuários são alertados, tanto via e-mail quanto via aplicação, no momento da doação;
- RF10 - O sistema deverá possuir uma interface onde tanto as doações concluídas quanto as não concluídas sejam exibidas, contendo uma opção para concluí-las.
- RF11 - O sistema deverá possuir uma interface onde a prefeitura possa consultar o estoque das doações.
- RF12 - O sistema deverá contar com uma função para registrar as entregas/distribuições para as famílias/pessoas recebedoras.
- RF13 - O sistema deverá contar com um cadastro para as famílias/pessoas recebedoras.
- RF14 - O sistema deverá permitir que a instituição distribuidora defina um limite de tempo entre o recebimento de duas doações por pessoa/família.

4.2.1.3 Requisitos funcionais - Estratégico

- RF15 – O sistema deverá conter um relatório com as doações recebidas por período (de acesso somente da prefeitura)

- RF16 - O sistema deverá conter um relatório com as distribuições das doações. Este relatório poderá ser gerado de maneira detalhada pela prefeitura e de maneira geral pelos doadores.

4.2.1.4 Requisitos não funcionais - Usabilidade

- RNF01 - O sistema deverá ser totalmente Web, acessado via navegador de internet;

4.2.1.5 Requisitos não funcionais - Segurança

- RNF02 - O sistema não deverá permitir o acesso sem autenticação do usuário;
- RNF03 - O sistema deverá utilizar criptografia no tráfego dos dados tanto na rede de computadores quanto na internet.

4.2.1.6 Requisitos não funcionais - Confiabilidade

- RNF04 - O sistema não deve permitir o agendamento de doações sem itens;
- RNF05 - O sistema somente deve permitir a confirmação de doações pela prefeitura;
- RNF06 - O sistema não pode permitir que doações não confirmadas sejam contabilizadas como recebidas;
- RNF07 - O sistema não deve permitir que doadores tenham acesso aos dados de doações feitas por outras pessoas, nem aos detalhes das entregas das doações realizadas pela prefeitura. Estas informações somente serão disponibilizadas de maneira geral via relatório.

4.2.1.7 Requisitos não funcionais - Portabilidade

- RNF08 - O sistema deve funcionar nos principais navegadores, tais como Chrome, Opera, Edge, Firefox, dentre outros;
- RNF09 - O sistema deverá ser responsivo, tendo sua interface adaptada aos vários tamanhos de dispositivos;

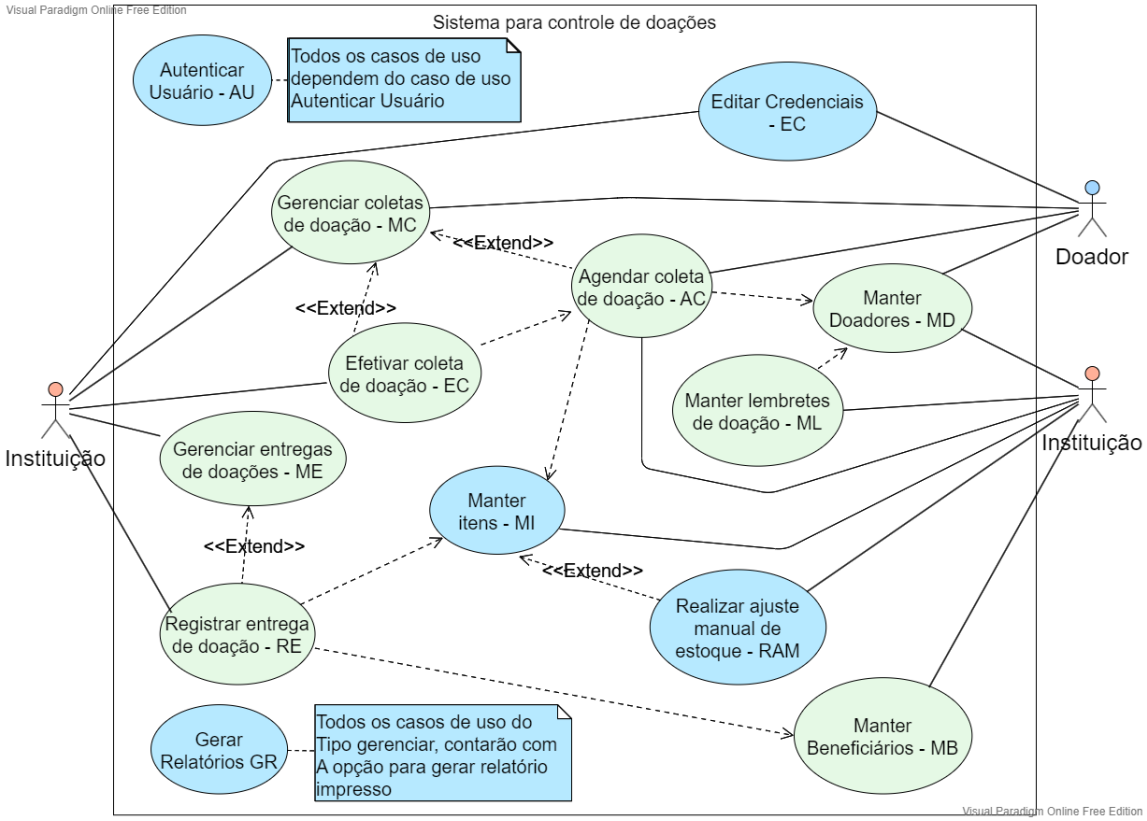
4.2.1.8 Requisitos não funcionais - Portabilidade

- RNF10 - O sistema deve responder às requisições dos usuários em no máximo 3 segundos em condições normais de operação.

4.2.2 Diagrama de casos de uso

A Figura 1 exibe o diagrama casos de uso do sistema. A cor azul representa os itens relacionados ao módulo base, já a cor verde representa os itens relacionados ao módulo de doações.

Figura 1 – Diagrama de Casos de Uso.



Fonte: Autoria própria (2022).

4.2.3 Especificação dos casos de uso

A seguir será apresentado o detalhamento individual dos casos de uso ilustrados na Figura 1, apresentando a os eventos e ações do ator e do sistema.

Caso de Uso	Autenticar Usuário - AU
Atores	Doador (Inicializador), Instituição (Inicializador).
Finalidade	Garantir confidencialidade dos dados de cada usuário, bem como apenas garantir acesso ao sistema para usuários que tenham seu cadastro efetuado.
Pré-Condições	N/A Sem dependências.
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - O ator acessa a URL do sistema.	2 - O sistema apresenta uma interface, que permite ao ator fazer login informando seu usuário e senha.
3 - O Ator informa seu login e senha e confirma a ação.	4 - O sistema valida e salva as informações, logando o usuário no sistema.
Sequência alternativa de eventos	
Ação do Ator	**Resposta do sistema**
1 - O ator acessa a URL do sistema.	2 - O sistema apresenta uma interface, que permite ao ator se cadastrar para se tornar um usuário.
3 - O Ator informa seus dados pessoais e credenciais de acesso.	4 - O sistema valida e salva as informações, redirecionando o usuário para tela de login.
Exceções	
Passo 3 - (Típico): O Ator informou usuário e senha que não coincidem entre si ou não tem registro com o usuário selecionado; **Passo 3 - (Alternativo):** Ao se cadastrar, o Ator informou senhas que não coincidem entre si; **Passo 3 - (Alternativo):** Ao se cadastrar, o Ator informou um e-mail e/ou usuário que já possuem contas registradas; **Passo 3:** Algum dos campos não foi informado no cadastro.	
Pós-Condições (Típicas)	O Ator terá acesso à interface principal do sistema.
Pós-Condições (Alternativas)	O Ator será registrado no sistema.

Tabela 1 – Caso de uso Autenticar Usuário.

Caso de Uso	Agendar coleta de doação - AC
Atores	Doador (Inicializador), Instituição (Inicializador).
Finalidade	Agendar uma coleta de doação, quando se tem a intenção de doar algo para a instituição.
Pré-Condições	Ter o item previamente cadastrado. Dependências: **Manter itens - MI**.
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - O doador seleciona a função para agendar a doação.	2 - O sistema apresenta um formulário contendo campos de observação para doação, item(s) e quantidade(s) a ser(em) doada(s).
3 - O doador informa os dados e salva o agendamento da doação.	4 - O sistema valida as informações e apresenta uma mensagem de confirmação
Sequência alternativa de eventos	
Ação do Ator	**Resposta do sistema**
1 - A instituição seleciona a função para agendar a doação.	2 - O sistema apresenta um formulário contendo campos de observação para doação, item(s) e quantidade(s) a ser(em) doada(s), além do campo para informação do doador que fez a doação.
3 - A instituição informa os dados e salva o agendamento	4 - O sistema valida e salva as informações, apresentando uma mensagem que o registro foi gravado com sucesso.
Exceções	
Passo 3: O ator informou uma quantidade negativa do(s) item(s). **Passo 3:** Algum dos campos (com exceção da observação) não foi informado no cadastro.	
Pós-Condições (Típicas)	Uma coleta de doação será lançada, ficando com status pendente até ser efetivada pela instituição.
Pós-Condições (Alternativas)	Uma coleta de doação será lançada, e será efetivada automaticamente.

Tabela 2 – Caso de uso Agendar Coleta de doação.

Caso de Uso	Efetivar coleta de doação - EC
Atores	Instituição (Inicializador).
Finalidade	Quando algum doador lança uma coleta de doação, esta precisa ser efetivada pela prefeitura para poder ser contabilizada de fato.
Pré-Condições	Ter uma coleta de doação previamente agendada por algum doador. Dependência: **Agendar coleta de doação - AC**
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - A instituição seleciona a função para efetivar a coleta de doação na listagem de doações.	2 - Será apresentada uma interface para o ator confirmar a ação.
3 - A instituição confirma a efetivação da coleta.	4 - O sistema valida e salva as informações, apresentando uma mensagem de que o registro foi salvo.
Exceções	
Passo 3: O ator cancela a ação.	
Pós-Condições	A coleta de doação será efetivada, e contabilizará no estoque de itens da prefeitura.

Tabela 3 – Caso de uso Efetivar coleta de doação.

Caso de Uso	Manter lembretes de doação - ML
Atores	Instituição (Inicializador).
Finalidade	Função na qual a instituição poderá manter os lembretes para que os doadores recebam um e-mail lembrando-os de fazer suas doações.
Pré-Condições	N/A Sem dependências.
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - A instituição seleciona a função para cadastrar um novo lembrete de doação.	2 - O sistema apresenta uma interface com as possíveis operações, **inclusão** de um lembrete, a **exclusão** de algum lembrete e uma listagem para **consulta** dos lembretes cadastrados.
3 - A instituição seleciona a operação desejada: Inclusão, Cancelamento ou Consulta.	
4 - Em caso de Inclusão, cancelamento, a instituição pode confirmar ou voltar ao passo 2.	5 - O sistema valida e salva as informações e apresenta uma mensagem confirmando a ação.
Exceções	
Passo 3 Inclusão ou edição: O lembrete ou a data do lembrete estão vazios;	
Pós-Condições	Um lembrete foi inserido, removido, alterado ou consultado pela instituição.

Tabela 4 – Caso de uso Manter lembretes de doação.

Caso de Uso	Realizar ajuste manual de estoque - RAM
Atores	Instituição (Inicializador).
Finalidade	Corrigir estoque dos itens.
Pré-Condições	Ter o item previamente cadastrado. Dependência: **Manter item - MI.**
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - A instituição seleciona a função para ajustar o estoque, dentro da listagem de itens.	2 - Será apresentada uma interface para informar a quantidade a ser retirada ou acrescentada, bem como uma observação para o ajuste.
3 - A instituição confirma o ajuste no estoque.	4 - O sistema valida e salva as informações e apresenta uma mensagem confirmando que o registro foi salvo.
Exceções	
Passo 3: Não foi informada mensagem ou quantidade a ser movimentada do item. **Passo 3:** Se for uma saída do estoque e o item não tem a quantidade necessária.	
Pós-Condições	O estoque do item em questão será ajustado, as movimentações envolvendo o item poderão ser visualizadas dentro do próprio cadastro do item.

Tabela 5 – Caso de uso Ajuste Manual de estoque.

Caso de Uso	Registrar entrega de doação - RE
Atores	Instituição (Inicializador).
Finalidade	Registrar as distribuições das doações às pessoas necessitadas, identificados por beneficiários dentro do sistema.
Pré-Condições	Ter o(s) item(s) previamente cadastrado(s); Ter o beneficiário previamente cadastrado; Dependências: **Manter item - MI**, **Manter beneficiário - MB.**
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - A instituição seleciona a função para registrar nova entrega de doação.	2 - Será apresentada uma interface com campos para seleção do(s) item(s) e quantidades, seleção do Beneficiário, informação do nome da pessoa responsável pela retirada, além de uma observação.
3 - A instituição confirma a entrega da doação ao beneficiário.	4 - O sistema valida e salva as informações, apresentando uma mensagem que o registro foi gravado com sucesso.
Exceções	
Passo 3: A quantidade informada de algum item é negativa. **Passo 3:** Com exceção da observação, algum campo não foi informado. **Passo 3:** Algum dos itens informados não tem o estoque suficiente.	
Pós-Condições	A entrega da doação será gravada e contabilizada junto às demais, o estoque do(s) item(s) selecionado(s) será movimentado.

Tabela 6 – Caso de uso Registrar entrega de doação.

Caso de Uso	Gerenciar entregas de doação - GE
Atores	Instituição (Inicializador).
Finalidade	Possibilitará à instituição alterar, cancelar e consultar as entregas de doação. Também servirá de ponto de partida alternativo para disparar o caso de uso **Registrar entrega de doação - RE.**
Pré-Condições	N/A Sem dependências
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - A instituição seleciona a função para gerenciar as entregas de doação.	2 - Será apresentada uma interface com a listagem das entregas de doação já realizadas e as possíveis ações que podem ser tomadas para elas.
3 - A instituição escolhe a ação alterar ou cancelar.	4 - Em caso de **alteração**, é exibida uma nova interface, com os dados da entrega e a instituição poderá alterá-los e salvar; Em caso de **cancelamento**, será exibido um modal para confirmação.
5 - A instituição confirma a ação escolhida.	6 - O sistema valida e salva as informações e apresenta a mensagem correspondente.
Exceções	
Passo 5 - Alteração: O estoque do(s) item(s) informado(s) não é suficiente para a nova quantidade. **Passo 5 - Alteração:** A quantidade informada para o(s) item(s) é negativa. **Passo 5 - Alteração:** Com exceção da observação, algum campo não foi informado.	
Pós-Condições	Em caso de **alteração**, a entrega terá seus dados alterados e o estoque do item será ajustado; Em caso de **cancelamento**, a entrega terá seu status alterado e o estoque do item será ajustado.

Tabela 7 – Caso de uso Gerenciar entregas de doação.

Caso de Uso	Gerenciar coletas de doação - GC
Atores	Instituição (Inicializador).
Finalidade	Possibilitará à instituição alterar, cancelar e consultar as coletas de doação geradas no caso de uso **Agendar coleta de doação - AC.** Também servirá de ponto de partida alternativo para disparar o caso de uso mencionado acima.
Pré-Condições	N/A Sem dependências
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - A instituição seleciona a função para gerenciar coletas de doação.	2 - Será apresentada uma interface com a listagem das coletas de doação já realizadas e as possíveis ações que podem ser tomadas para elas.
3 - A instituição escolhe a ação **alterar** ou **cancelar**.	4 - Em caso de **alteração**, é exibida uma nova interface, com os dados da coleta e a instituição poderá alterá-los e salvar; Em caso de **cancelamento**, será exibida uma mensagem de confirmação.
5 - A instituição confirma a ação escolhida.	
Exceções	
Passo 5 - Alteração: O estoque do(s) item(s) informado(s) não é suficiente para a nova quantidade. **Passo 5 - Alteração:** A quantidade informada para o(s) item(s) é negativa. **Passo 5 - Alteração:** Com exceção da observação, algum campo não foi informado.	
Pós-Condições	Em caso de **alteração**, a coleta terá seus dados alterados e o estoque do item será ajustado; Em caso de **cancelamento**, a coleta terá seu status alterado e o estoque do item será ajustado.

Tabela 8 – Caso de uso Gerenciar coletas de doação.

Caso de Uso	Editar Credenciais - EC
Atores	Instituição (Inicializador). Doador (inicializador).
Finalidade	Possibilitará ao ator alterar seus dados de login e seus dados pessoais, definidos na primeira autenticação.
Pré-Condições	N/A Sem dependências
Sequência típica de eventos	
Ação do Ator	**Resposta do sistema**
1 - O ator seleciona a função para alterar seus dados	2 - Será apresentado um formulário com seus dados, onde o doador pode alterá-los. Além de um botão para alterar usuário ou senha.
3 - O ator altera seus dados e pressiona o botão “salvar”.	4 - Os dados são alterados.
5 - O ator pressiona o botão para alterar a senha.	6 - É apresentada uma tela com os campos "Novo usuário", “Nova senha”, “Repetir nova senha”.
7 - O ator confirma a alteração dos dados.	8 - Os dados são validados, se estiver tudo certo, o sistema alerta que as credenciais foram alteradas com sucesso.
Exceções	
Passo 3: Algum dos campos do formulário dos dados cadastrais não foi preenchido. **Passo 8:** As senhas não coincidem.	
Pós-Condições	Os novos dados informados para o cadastro serão mantidos.

Tabela 9 – Caso de uso Gerenciar coletas de doação.

4.2.4 Demais casos de uso

Os demais casos de uso destinados às operações de CRUD (*Create, Read, Update e Delete*) possuem funcionamento semelhante ao caso de uso **Manter Beneficiários - MB**, porém para entidades diferentes. Estes casos de uso são listados a seguir.

Caso de Uso	Manter itens - MI
Finalidade	Possibilitará à instituição cadastrar, alterar, cancelar e consultar os itens cadastrados. Estes, que serão usados posteriormente nas doações e aquisição de recursos. Também servirá de ponte de partida para o caso de uso **Realizar ajuste manual de estoque - RAM**

Tabela 10 – Caso de uso Manter Itens.

Caso de Uso	Manter Beneficiários - MB
Finalidade	Possibilitará à instituição cadastrar, alterar, cancelar e consultar os beneficiários cadastrados. Estes, que serão usados posteriormente nas entregas de doações, sendo a pessoa que receberá a doação.

Tabela 11 – Caso de uso Manter lembretes de doação.

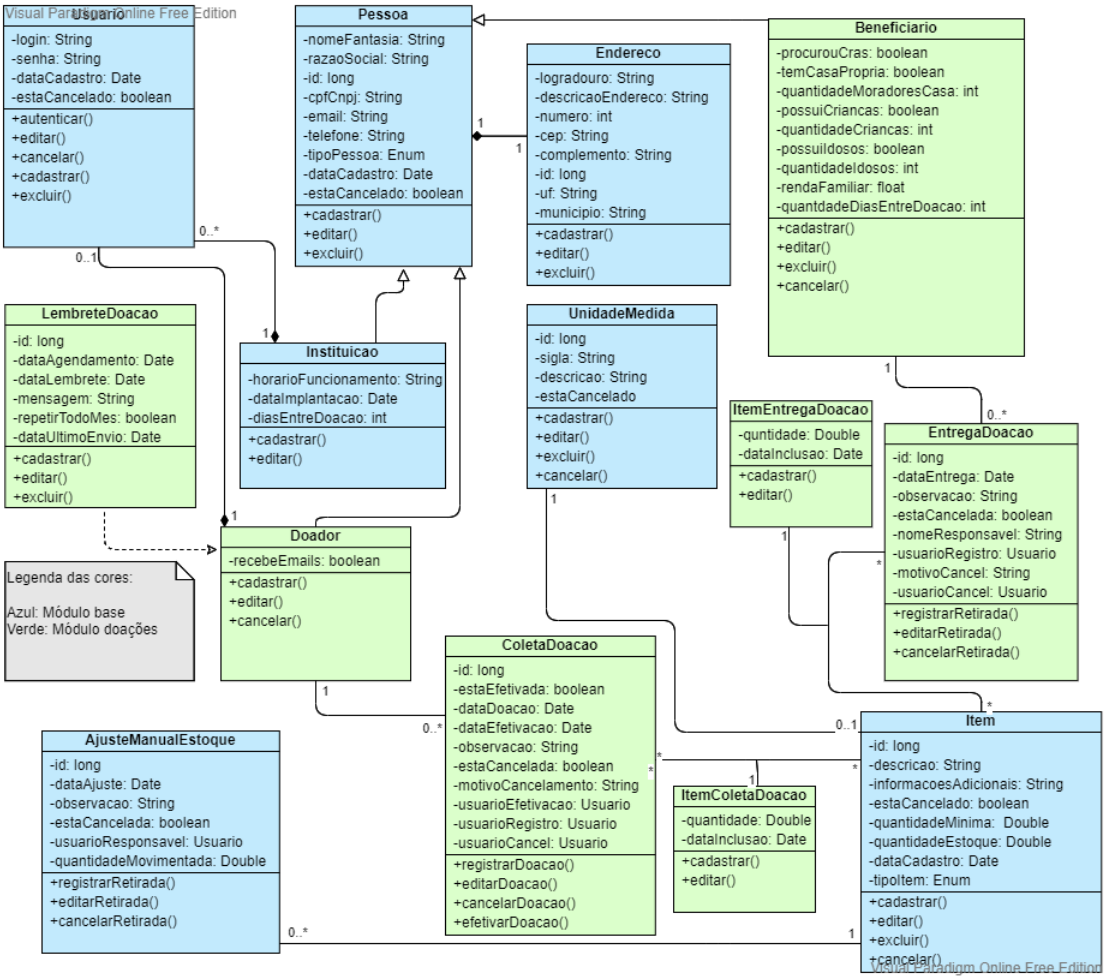
Caso de Uso	Gerar Relatórios - GR
Finalidade	Permitirá que a Instituição ou Doador, faça a impressão dos dados presentes nas telas de gerenciamento, gerando assim um documento PDF. O ator também terá a possibilidade de gerar um relatório geral de coletas e entregas de doação

Tabela 12 – Caso de uso Manter lembretes de doação.

4.2.5 Diagrama de classes

Uma representação visual da camada de modelo da aplicação contém as classes que serão utilizadas no sistema, representando seus atributos, principais métodos e relacionamentos entre si. Diante disso, a Figura 2 exibe o diagrama de classes da aplicação.

Figura 2 – Diagrama de Classes.



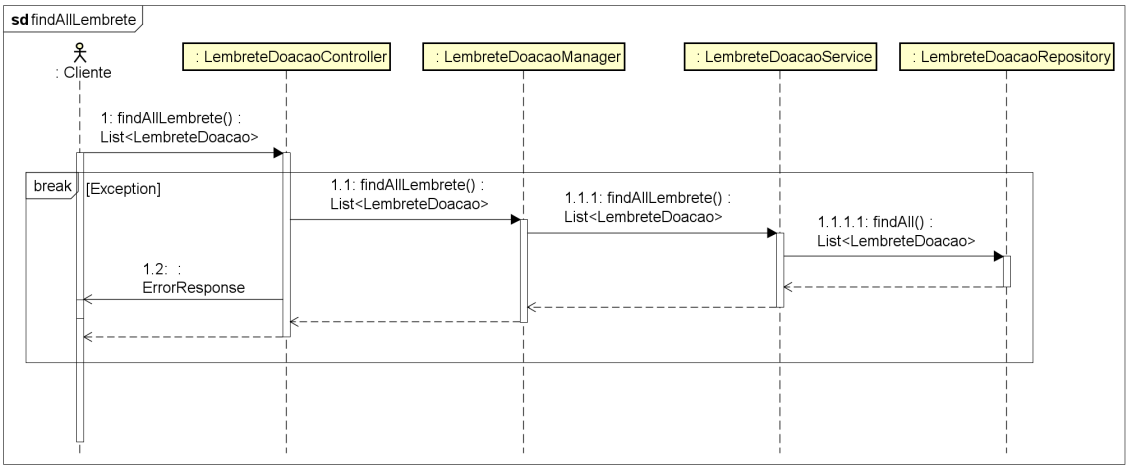
Fonte: Autoria própria (2022).

4.2.6 Diagramas de sequência

Para estruturar como será o fluxo de execução da aplicação e as mensagens trocadas entre os componentes da aplicação, foram criados diagramas de sequência de alguns fluxos de execução que são comuns entre os variados recursos da aplicação.

A Figura 3 exibe o diagrama de sequência para busca de lembretes de doação.

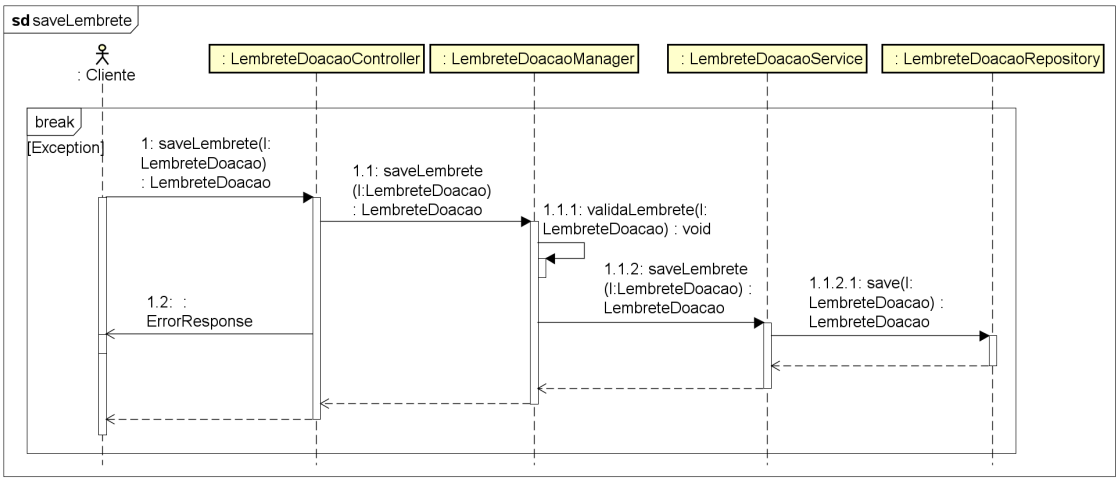
Figura 3 – Diagrama de Sequência Buscar Lembretes de doação.



Fonte: Autoria própria (2022).

A Figura 4 exibe o diagrama de sequência para criação de um novo lembrete de doação.

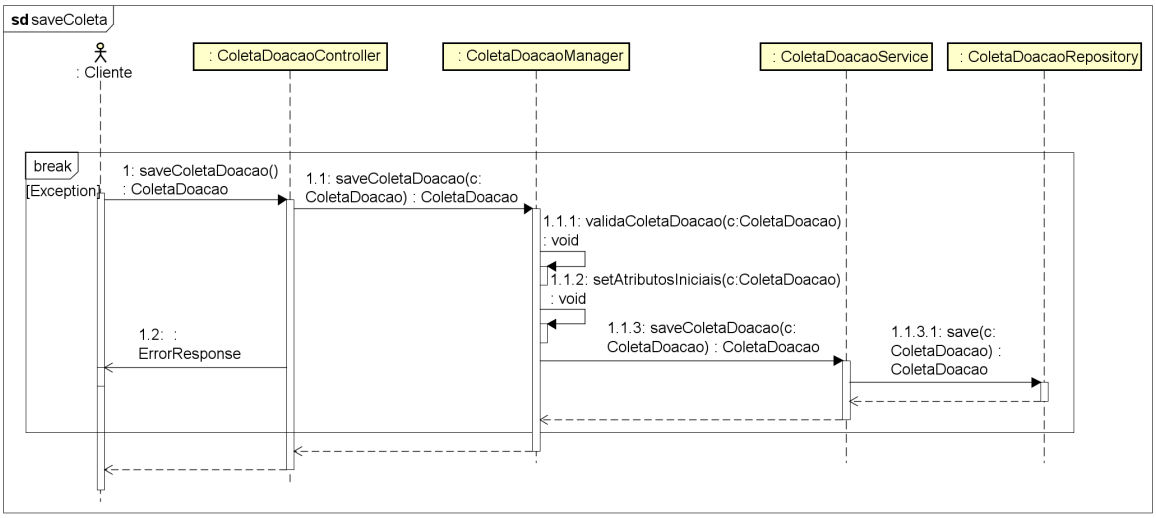
Figura 4 – Diagrama de Sequência Registrar novo Lembrete de doação.



Fonte: Autoria própria (2022).

A Figura 5 exibe o diagrama de sequência para criação de uma nova coleta de doação.

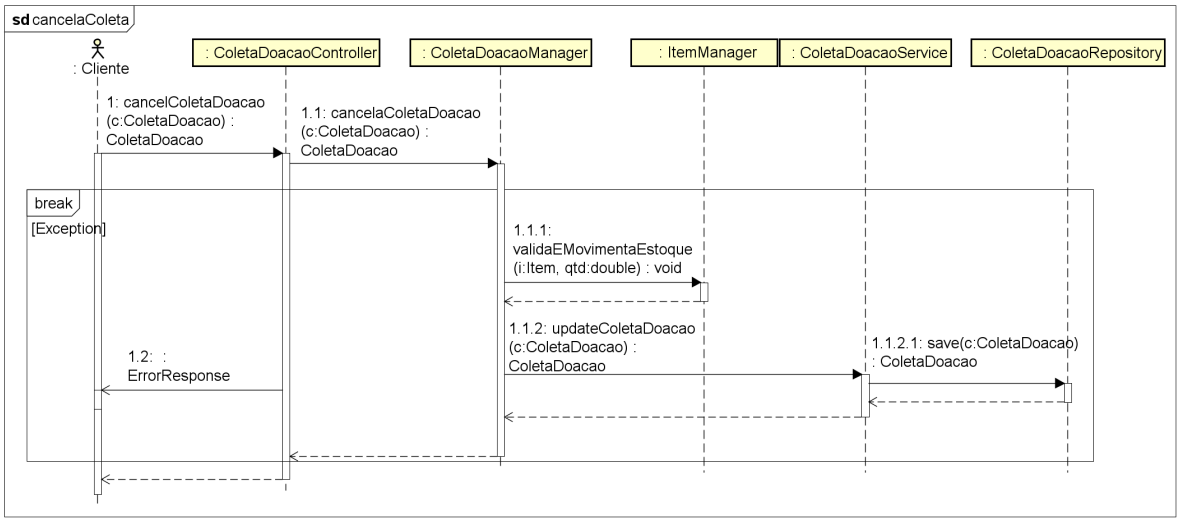
Figura 5 – Diagrama de Sequência Registrar nova Coleta de Doação.



Fonte: Autoria própria (2022).

A Figura 6 exibe o diagrama de sequência para cancelamento de uma coleta de doação.

Figura 6 – Diagrama de Sequência Cancelar Coleta de Doação.



Fonte: Autoria própria (2022).

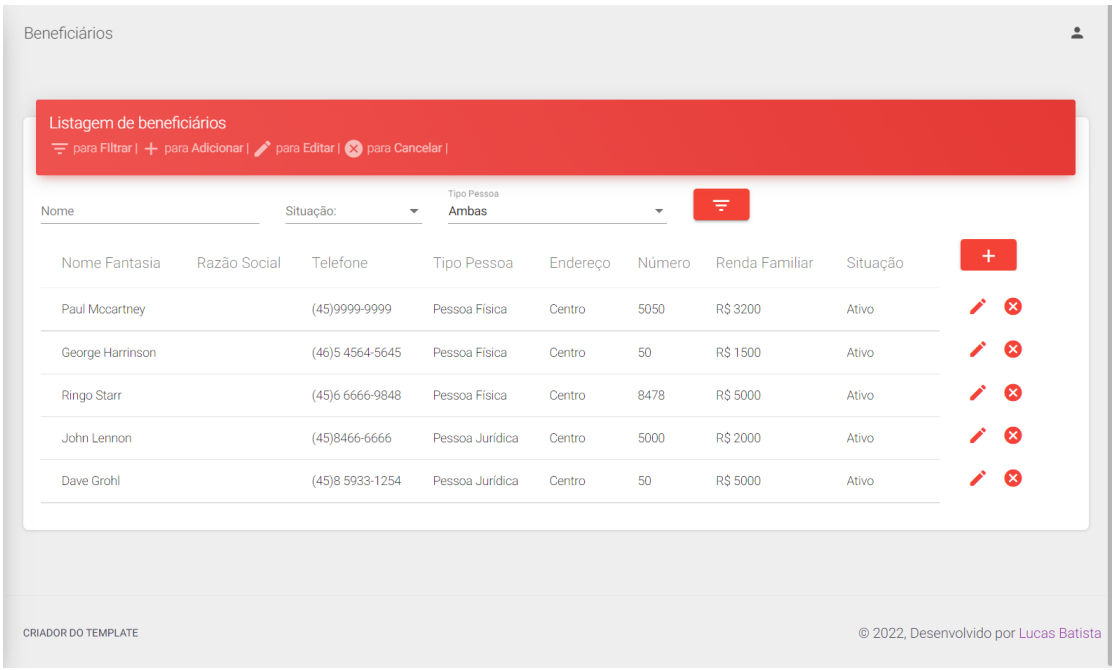
4.3 Apresentação do sistema

4.3.1 Telas de cadastro

O sistema possui uma série de telas de cadastro que servem como base para as funcionalidades que controlam as doações. De maneira geral, essas telas contam com as operações CRUD onde cada cadastro conta com duas interfaces. A primeira para listagem, onde é exibida

uma lista contendo todos os registros e filtros para consulta. Nessa mesma interface, existem botões para cadastro de novos registros, edição e cancelamento (ou exclusão) dos itens listados. Alguns cadastros em particular podem contar com funções específicas além das operações de CRUD, como por exemplo a tela de cadastro de itens que conta com as opções de ajuste de estoque e listagem das movimentações de estoque. A Figura 7 exibe um exemplo de interface de listagem, nesse caso, dos beneficiários cadastrados.

Figura 7 – Interface de listagem de beneficiários.



Fonte: Autoria própria (2022).

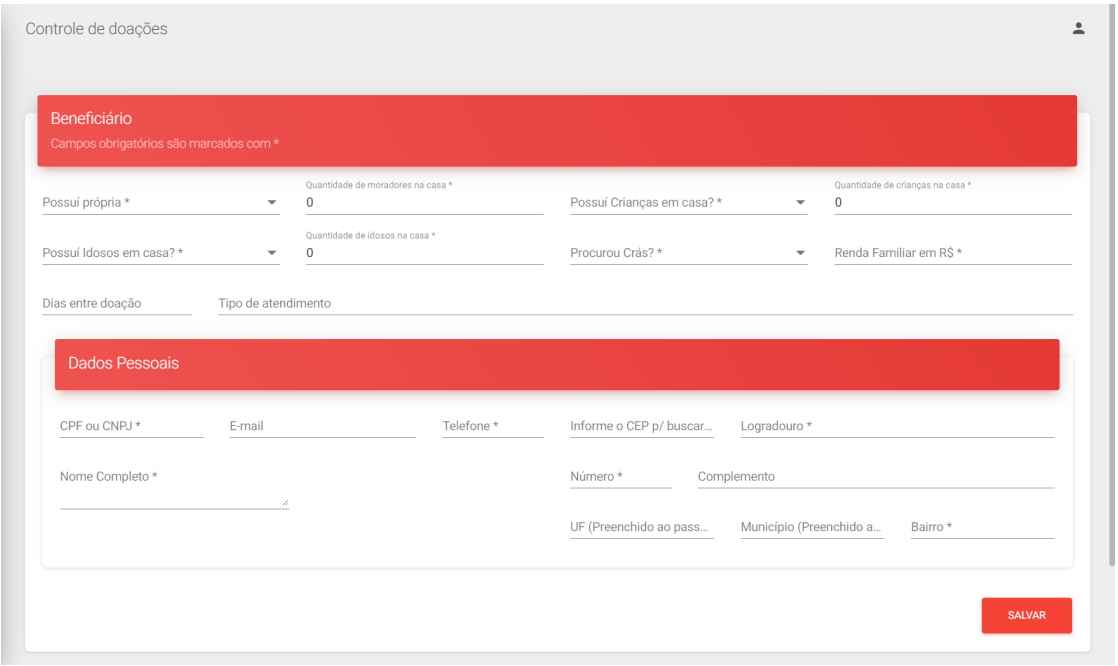
A segunda interface relativa aos cadastros, é o formulário para criação de novos registros e contém uma série de campos que compõem o objeto a ser cadastrado e que são preenchidos pelo usuário. Alguns destes campos contém máscaras de formato para facilitar a inserção e visualização dos dados, outros campos possuem integração com serviços externos para carregamento de informações, como a consulta do CEP (Código de Endereçamento Postal) para retornar os dados do endereço. Além disso, os fomulários apresentam validações sobre campos obrigatórios ou inválidos para facilitar a experiência do usuário. A Figura 8 exibe um exemplo da interface de formulário para cadastro de novos beneficiários.

4.3.2 Telas de Gerenciamento e lançamento

Na sessão anterior, foram mencionadas as telas de cadastro, que são a base para execução das atividades que serão controladas pelo sistema, essas telas também são divididas em duas interfaces.

A primeira delas é utilizada para listagem e gerenciamento, onde há as opções de edição, conferência, cancelamento e no caso das coletas de doação, efetivação. Essas opções

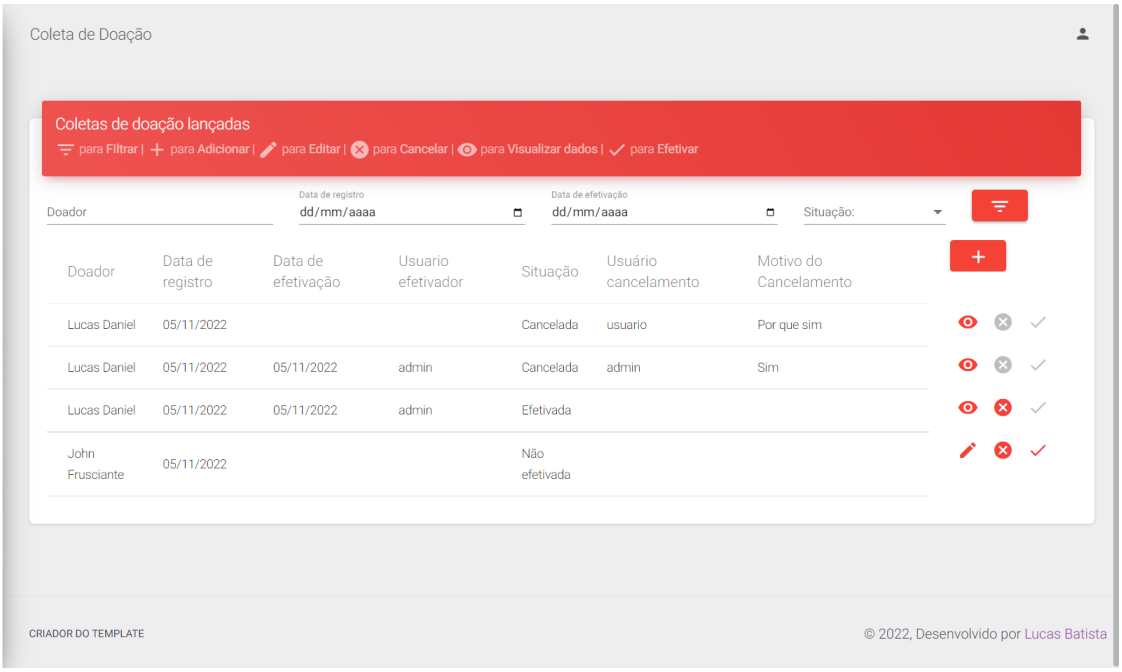
Figura 8 – Interface com formulário para novo beneficiário



Fonte: Autoria própria (2022).

podem estar, ou não, disponíveis dependendo do usuário logado. Por exemplo, um usuário que é doador não pode efetivar coletas de doação, pois é uma atividade exclusiva de usuários relacionados à instituição. A Figura 9 exibe um exemplo da interface de gerenciamento de coletas de doação.

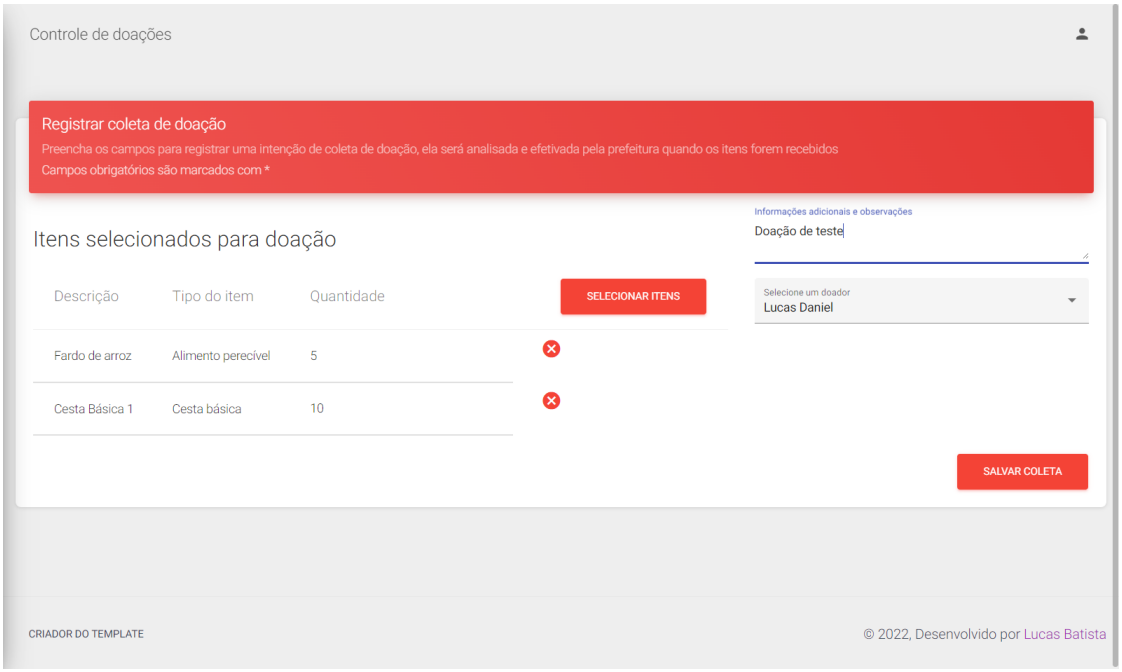
Figura 9 – Interface de gerenciamento.



Fonte: Autoria própria (2022).

A segunda interface é a de lançamento de novos registros, de maneira similar aos formulários, ela também contém uma série campos a serem preenchidos. Porém, ela conta com validações como estoque de itens, permissões de doadores e beneficiários, dentre outras, variando conforme o lançamento a ser efetuado. Além disso, há regras para inserção e atualização dos itens que compõem o lançamento. A Figura 10 exibe um exemplo da interface para lançamento de uma coleta de doação.

Figura 10 – Interface de lançamento.



Fonte: Autoria própria (2022).

4.3.3 Implementação do Sistema

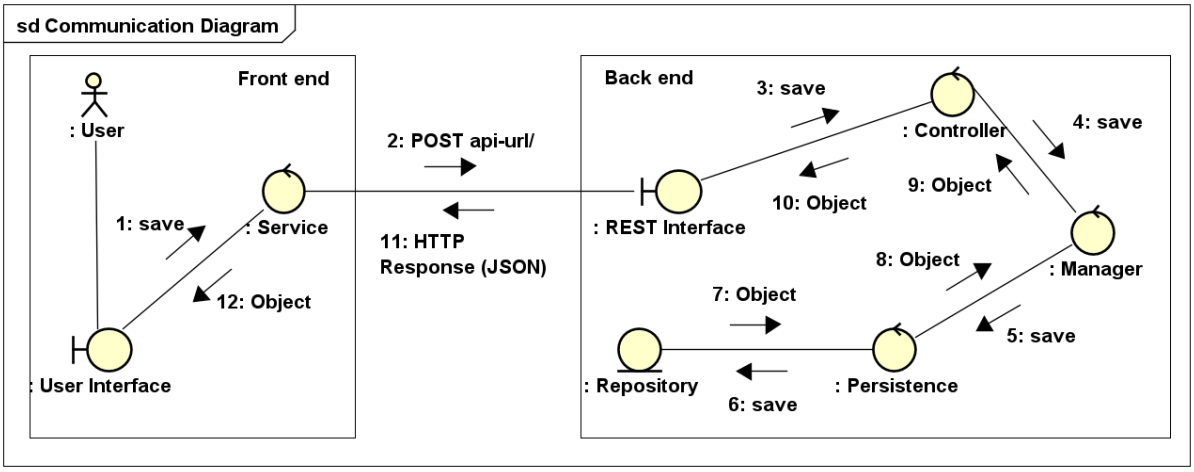
A implementação foi dividida em dois softwares individuais, uma API REST atuando como *back end* e outra aplicação Web SPA atuando como *front end*, que será acessada através de navegadores de internet. Nessa arquitetura, a responsabilidade por disponibilizar, manipular e persistir os recursos é total do *back end*, enquanto o *front end* é responsável por consumir os recursos entregues pela API e apresentá-los em uma interface amigável para o usuário final.

A Figura 11 exibe o diagrama de comunicação entre as aplicações, trazendo uma visão macro do fluxo de execução das funcionalidades do sistema por meio de troca de mensagens.

4.3.3.1 API

A aplicação *back end* tem uma arquitetura dividida em 4 níveis (*Controller, Manager, Persistence e Repository*), onde cada um é responsável por suas funções e a comunicação entre esses níveis é realizada através de contratos, que são as interfaces. Além disso, para instanciar

Figura 11 – Diagrama de Comunicação.



Fonte: Autoria própria (2022).

objetos é utilizada a injeção de dependências do Spring boot, ambas as práticas, contribuem para redução do acoplamento interno da aplicação, conforme mencionado na Sessão 2.3.

Na camada de *Controller* são mapeadas as URIs que servirão como *endpoints* para a API. Um exemplo desta camada é apresentado na Listagem 1. A anotação *@RestController* do Spring é quem permite à alguma classe conter esse mapeamento, além disso, a anotação *@AutoWired* é quem faz a injeção de dependência de determinada classe, instanciando um objeto dela, já a anotação *@GetMapping*, informa ao Spring qual verbo HTTP aquele método irá responder e pode ou não receber o parâmetro contendo uma URL específica e como no exemplo a seguir, um parâmetro passado via URL. Também é possível ver a anotação *@PathVariable* nos parâmetros do método *findItemById()* que é responsável por traduzir o valor passado via parâmetro na URI para algum valor de entrada no método. Por fim, é possível notar que a classe de *Controller* estende a classe *EntityValidateExceptionHandler*, que nesse contexto é responsável por validar os dados da entidade recebida em métodos POST ou PUT, conforme as anotações presentes na classe de entidade a ser exibida na sequência. O trecho de código na Listagem 1 exemplifica como é construída uma classe *Controller* no *back end*.

A camada de *Manager* contém além das classes que representam os recursos da aplicação e a integração com o banco de dados para criação das tabelas, uma interface que disponibiliza os métodos para manipulação dos recursos pela camada de *Controller* e uma classe que implementa essa interface, sendo essa responsável pelas regras de negócio que cabem aquele recurso. Um exemplo desta camada é apresentado na Listagem 2. A anotação *@Component* indica ao Spring que essa é uma classe que precisa ser gerenciada por ele e permite que seja feita a injeção de dependências. Em conjunto, a anotação *@Trasactional* que pertence ao Spring *Data* JPA indica que todas as operações realizadas em qualquer método serão executadas em uma única transação do banco de dados, ou seja, se algo der errado durante a execução, nenhuma alteração é salva.

Listagem 1 – Classe *Controller* de Itens

```
1  @RestController
2  @RequestMapping("/item")
3  public class ItemController extends EntityValidateExceptionHandler {
4      @Autowired
5      ItemManager managerItem;
6
7      @GetMapping("/{id}")
8      public ResponseEntity<Item> findItemById(@PathVariable("id") Long id){
9          Optional<Item> item = managerItem.findById(id);
10
11         return new ResponseEntity<Item>(item.get(), HttpStatus.OK);
12     }
13 }
```

Fonte: Autoria própria (2022).

O método *findById()* da interface *ItemManager* é sobrescrito e faz a chamada do método correspondente na camada inferior, a qual é instanciada via injeção de dependência. O trecho de código da Listagem 2 exemplifica como é construída uma classe *Manager* no *back end*.

Listagem 2 – Classe *Manager* de itens

```
1  @Component
2  @Transactional
3  public class ItemManagerImp implements ItemManager {
4
5      @Autowired
6      ItemService itemService;
7
8      @Override
9      public Optional<Item> findById(Long id) {
10         Optional<Item> item = itemService.findById(id);
11         if(item.isEmpty()){
12             throw new ResourceNotFoundException("Item não encontrado");
13         }else{
14             return item;
15         }
16     }
17 }
```

Fonte: Autoria própria (2022).

A próxima camada é a de *Persistence* apresentada na Listagem 3. é responsável por validações de integridade do banco de dados e se comunicar com o nível mais baixo da aplicação. O que há de novo nesse nível é a anotação *@Service* do Spring, que também serve para dizer ao Spring que essa é uma classe que precisa ser gerenciada por ele, permite que seja feita a injeção de dependência e que tal classe contém as validações de integridade. Em alguns casos, as próprias validações de negócio podem ficar nas classes to tipo *Service* eliminando a necessidade do *Manager*, o que não foi feito nesse projeto para separar as responsabilida-

des de cada camada. Por fim, é injetada uma instancia da classe *ItemRepository*, para que seja possível utilizar seus métodos para concluir a execução. O trecho de código na Listagem 3 exemplifica como é construída uma classe *Service* no *back end*.

Listagem 3 – Classe *Service* de itens

```
1  @Service
2  public class ItemImplService implements ItemService {
3
4      @Autowired
5      ItemRepository itemRepository;
6
7      @Override
8      public Optional<Item> findById(Long id) {
9          return itemRepository.findById(id);
10     }
11 }
```

Fonte: Autoria própria (2022).

Por fim, temos a camada *Repository*, que basicamente é o nível mais próximo do banco de dados. Esta camada faz a comunicação direta com o banco de dados e persiste os recursos da aplicação. A anotação *@Repository* apresentada na Listagem 4 indica ao Spring que essa classe é a responsável pela comunicação com o banco de dados. Ao estender a classe *JpaRepository* e passar a entidade a ser persistida junto com o tipo de dados do identificador único (id), já é disponibilizada uma série de métodos padrão, como por exemplo o método *findById(Long id)* visto no trecho de código da Listagem 3. Nessa classe, também é possível criar consultas personalizadas, seja através do nome do método ou de maneira manual, com a linguagem *Java Persistence Query Language* (JPQL), que não foi utilizado neste projeto. O Trecho de código 4 exemplifica como é construída uma classe *Repository* no *back end*.

Listagem 4 – Classe *Repository* de itens

```
1  @Repository
2  public interface ItemRepository extends JpaRepository<Item, Long> {
3
4      List<Item> findByEstaCanceladoFalse();
5
6  }
```

Fonte: Autoria própria (2022).

Além do fluxo de execução base de todas as funcionalidades da API, também existe uma classe responsável pela captura de exceções que são disparadas na aplicação. A Listagem 5 apresenta um trecho de código desta classe. A anotação *@ControllerAdvice* Permite com que a classe fique responsável por capturar qualquer exceção disparadas na execução dos *Controllers*, e com a anotação *@ExceptionHandler* é possível criar métodos que capturam e tratam exceções específicas, ou alguma exceção geral. Tendo o mapeamento das exceções,

é possível retornar o HTTP *Status Code* adequado. O trecho de código da Listagem 5 mostra onde exceções de recursos não encontrados são tratadas para retornar Status HTTP 404.

Listagem 5 – Classe contendo *Handler* de exceções

```
1  @ControllerAdvice
2  public class ApplicationExceptionHandler extends
3          ResponseEntityExceptionHandler {
4
5      @ExceptionHandler(ResourceNotFoundException.class)
6      public ResponseEntity<ErrorResponse>
7      handleResourceNotFoundException(ResourceNotFoundException e){
8
9          e.printStackTrace();
10         List<String> mensagem = new ArrayList<>();
11         mensagem.add(e.getMessage());
12         return new ResponseEntity<>(new ErrorResponse(mensagem),
13                 HttpStatus.NOT_FOUND);
14
15     }
16 }
```

Fonte: Autoria própria (2022).

4.3.3.2 *Front end*

A aplicação *front end* é uma SPA, desenvolvida com o *Framework Angular* na linguagem *Typescript*, conforme mencionado no Capítulo 3. A arquitetura do *front end*, foi separada em *Services*, Componentes e Módulos, que basicamente são conjuntos de componentes. Os componentes do *Angular* são uma estrutura que contém uma classe *Typescript*, um arquivo *Hyper Text Markup Language* (HTML) denomidado *Template* e um arquivo de *Cascading Style Sheet* (CSS) exclusivo para o HTML de tal componente. Com isso, os componentes podem ser reutilizados dentro e outros componentes e também podem ser organizados da maneira que melhor fizer sentido na aplicação a ser desenvolvida.

Como boa prática recomendada pelo *Angular*, a comunicação com a APIs deve ser feita por *Services*, que tem a responsabilidade por alguma regra de negócio que seja necessária nos dados e também por essa comunicação com aplicações externas. A anotação *@Injectable* do *Angular*, como pode ser vista na Listagem 6, permite que o Serviço seja injetado via construtor nas classes que farão uso dos seus métodos. É possível notar também que há uma constante chamada *API* que recebe o valor da variável de ambiente *APIPATH*, a qual contém a URL base do *back end*. No construtor da classe *ItemService*, também existe a injeção da classe *HttpClient* do *Angular* que é responsável por realizar a comunicação via HTTP. O trecho de código na Listagem 6 mostra um exemplo de um *Service* no *front end*.

Acima dos *Services*, temos os componentes que fazem uso dos métodos disponibilizados por eles, conforme dito no início da sessão, os componentes tem uma classe *Typescript*,

Listagem 6 – Classe Service de item no *front end*

```
1  @Injectable({
2    providedIn: "root",
3  })
4  export class ItemService {
5    private readonly API = `${APIPATH}item`;
6
7    constructor(private httpClient: HttpClient) {}
8
9    getById(id: string) {
10     return this.httpClient.get<Item>(this.API + `/${id}`).pipe(first());
11   }
12 }
```

Fonte: Autoria própria (2022).

que é responsável por manipular os dados enviados ou recebidos dos serviços, bem como conter alguma lógica referente à interface visual, presente no arquivo HTML do componente. Na Listagem 7 é possível verificar que a anotação *@Component* do *Angular* recebe alguns parâmetros, sendo eles *selector*, que é a *tag* HTML que representará esse componente para se usado dentro de outros componentes. Já os parâmetros *templateUrl* e *styleUrls* recebem a localização dos arquivos HTML e CSS do componente. No construtor da classe, é feita a injeção de dependências que serão utilizadas, nesse caso o *ItemService* mostrado no trecho de código da Listagem 6 e outro *GeneralApiRequestService* que será utilizado posteriormente para tratar os erros vindos da API. Por fim, o método *carregaItens()* faz uso do método *getAll()* do *Service* para receber a lista de itens, e caso haja algum erro, chama o *Service* de validação e retorna uma lista vazia. O trecho de código da Listagem 7 mostra um exemplo de um *Component* no *front end*.

Outros exemplos de componente são aqueles que contém algum formulário ou de alguma forma precisam enviar informações ao serviço. Como a estrutura de uma classe componente já foi exemplificada na Listagem 7, a seguir somente serão mostrados os métodos que são responsáveis por enviar os dados do formulário ao serviço ou validar o formulário para exibição de mensagens através do componente de notificações *NotificationsComponent*. Este componente é injetado via construtor e atribuído ao objeto *notification*. Além disso, na chamada do *Service*, temos a função *subscribe* que aguarda o retorno da API. Caso haja algum erro, chama o serviço de validação de erros *generalApi*. Caso tudo der certo, o fomulário é limpo e é exibida uma mensagem de sucesso. A Listagem 8 mostra um exemplo de um *Component* de formulário no *front end*.

Listagem 7 – Classe *Component* de listagem de itens no *Front end*

```
@Component({
  selector: "app-item",
  templateUrl: "./item.component.html",
  styleUrls: ["./item.component.scss"],
})
export class ItemComponent implements OnInit {
  itemListObservable: Observable<Item[]>;

  constructor(
    private itemService: ItemService,
    private generalApi: GeneralApiRequestService,
  ) {
    this.carregaItens();
  }

  carregaItens(){
    this.itemListObservable = this.itemService.getAll().pipe(
      catchError((error) => {
        this.generalApi.validateErrors(error);
        return of([]);
      })
    );
  }
}
```

Fonte: Autoria própria (2022).

4.3.4 Implantação do sistema

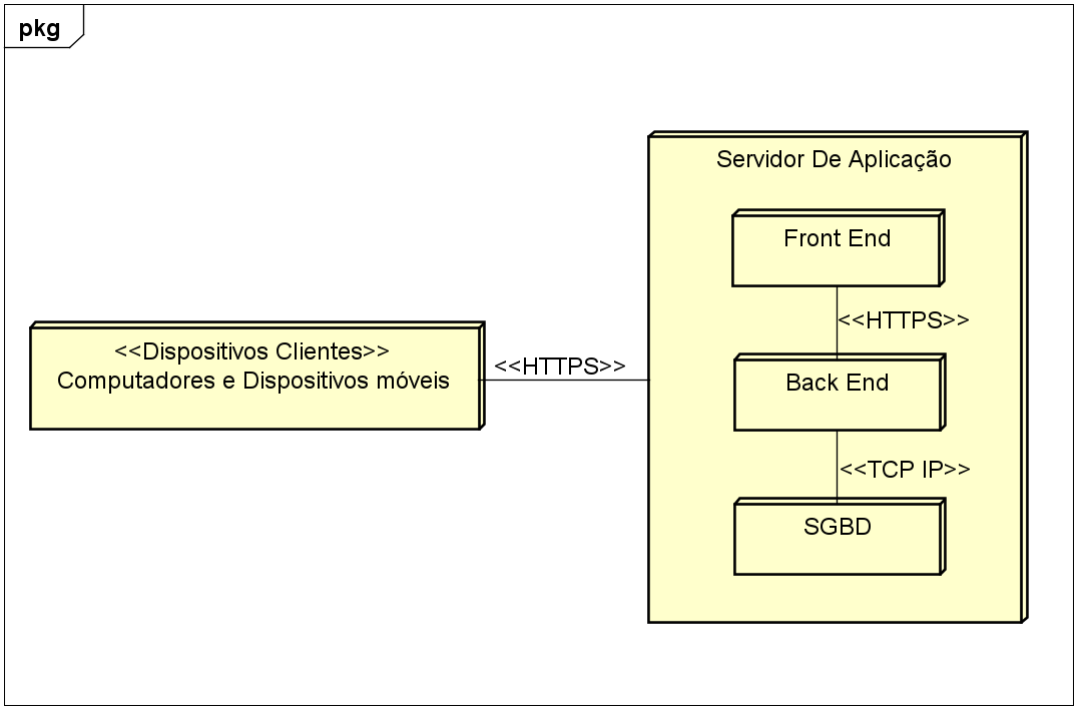
Para ilustrar como o sistema será implementado em produção, foi construído um diagrama de implantação da UML. Este diagrama, tem como objetivo mostrar o *layout* físico do sistema, exibindo as partes de software e de hardware e seus relacionamentos. O diagrama de implantação é dividido em nós, que podem conter outros nós dentro de si, conforme no ilustra a Figura 12. Esta figura mostra um nó chamada Servidor De Aplicação, representando uma máquina física, que contém as aplicações de *Back End* e *Front End*, além do SGBD. O Servidor De Aplicação então, permite a conexão com os Dispositivos Clientes utilizados pelos usuários finais para acessar o sistema implantado.

Listagem 8 – Classe *Component* de item no *front end*

```
submit(){

    if(this.itemForm.invalid){
      this.notification.showWarningNotification('Há campos
        obrigatórios não preenchidos ou inválidos!');
      return;
    }

    const itemEnvio: Item = this.itemForm.value;
    itemEnvio.unidadeMedida = { id: this.unidadeMedida.value };

    this.itemService.save(this.itemForm.value)
    .subscribe({
      error: (e) => this.generalApi.validateErrors(e),
      complete: () => this.onItemSuccess()
    });

}

private onItemSuccess(){
    this.notification.showSuccessNotification('Item salvo com sucesso');
    this.itemForm.reset();
}
```

Fonte: Autoria própria (2022).

Figura 12 – Diagrama de Implantação



Fonte: Autoria própria (2022).

5 CONCLUSÃO

Este trabalho de conclusão de curso apresentou o processo de modelagem e desenvolvimento de duas aplicações, que se comunicam através de uma interface REST. Essas aplicações serão úteis para controlar as doações recebidas e distribuídas por uma prefeitura.

Ficou claro que o desenvolvimento de uma aplicação de qualidade, que deverá ser duradoura e ter fácil manutenção e expansão no futuro inclui uma série de etapas, desde o levantamento de requisitos até a escrita do código de fato. Na análise, o objetivo é que não haja ruídos de comunicação, falhas de entendimento e/ou desvios da regra de negócio original. Já no desenvolvimento, a aplicação deve fazer uso de boas práticas de modelagem, padronização e codificação, além de aderir aos padrões arquiteturais sofisticados para melhorar a qualidade e manutenibilidade da aplicação.

A utilização do *Framework* Spring no *back end* permite aplicar uma série de boas práticas, além de gerar como resultado final uma aplicação expansível e segura, que poderá ser consumida por diversas outras aplicações. A arquitetura usada também permite ao programador ter controle sobre a execução do software. Em conjunto com o Spring Data JPA, a integração com o banco de dados também é feita de maneira simples e descomplicada.

Por fim, o *front end* provou ser um desafio, pois o *Angular* tem um poder enorme sendo projetado para grandes aplicações. Muitos de seus recursos não se fizeram necessários no contexto desse trabalho. Pode-se concluir que o resultado obtido foi muito satisfatório, uma vez que a aplicação pode ser considerada apresentável e de fácil uso, além de ser apresentar estabilidade durante a sua execução.

REFERÊNCIAS

FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures. **Dissertação**, 2000. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.html.

FOWLER, M. **UML essencial**: Um breve guia para linguagem padrão. Porto Alegre, RS: bookman, 2005.

LLC, G. **Angular**. 2022. Disponível em: https://angular.io/docs.

PAPAZOGLOU, M. P. **Web Services**: Principles and technology. New Jersey, USA: Pearson Prentice Hall, 2008. ISBN 9780321155559.

POSTGRESQL, G. D. G. **PostgreSQL Documentation**. 2022. Disponível em: https://www.postgresql.org/docs/current/.

SPRING. **Spring Boot**. 2022. Disponível em: https://spring.io/projects/spring-boot.

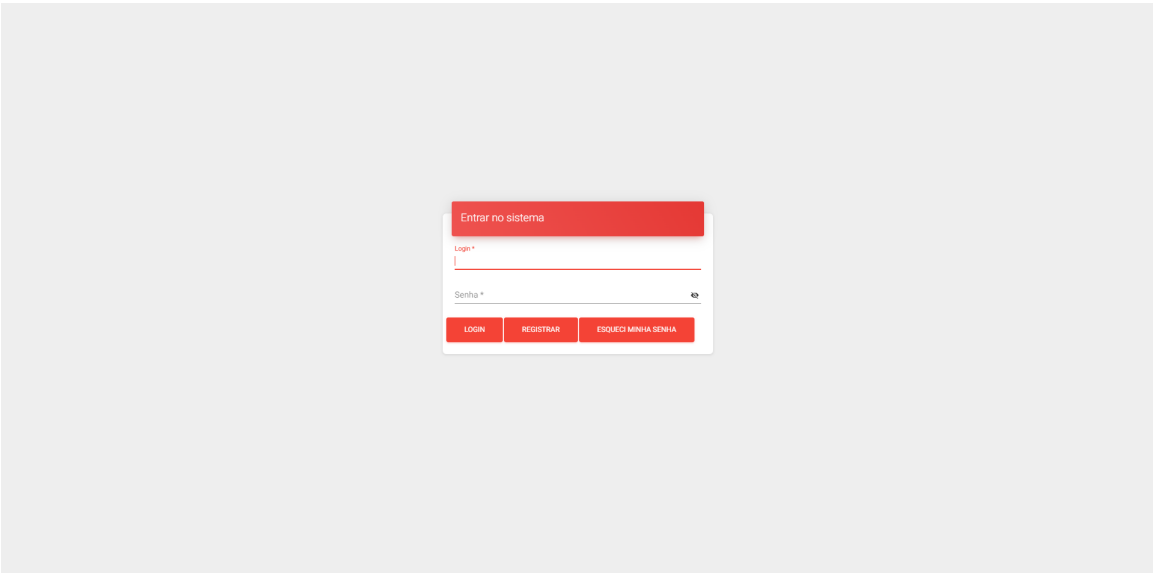
SPRING. **Spring Data JPA - Reference Documentation**. 2022. Disponível em: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.

WEISSMANN, H. L. **Vire o jogo com Spring Framework**. São Paulo, SP: Casa do Código, 2014.

APÊNDICES

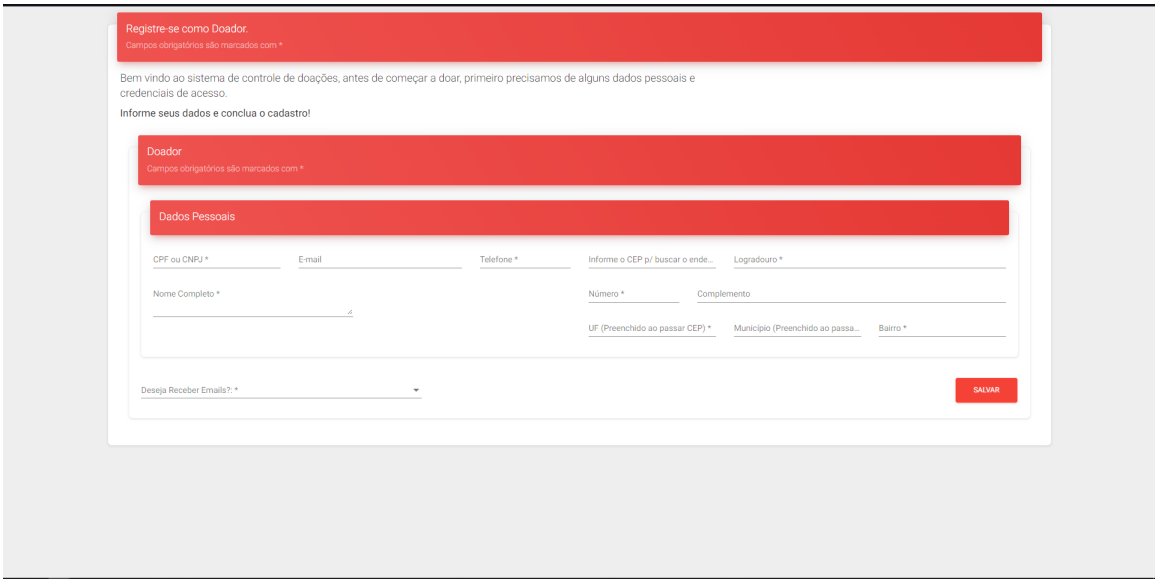
APÊNDICE A – Listagem de todas as telas do Sistema

Figura 13 – Tela de login



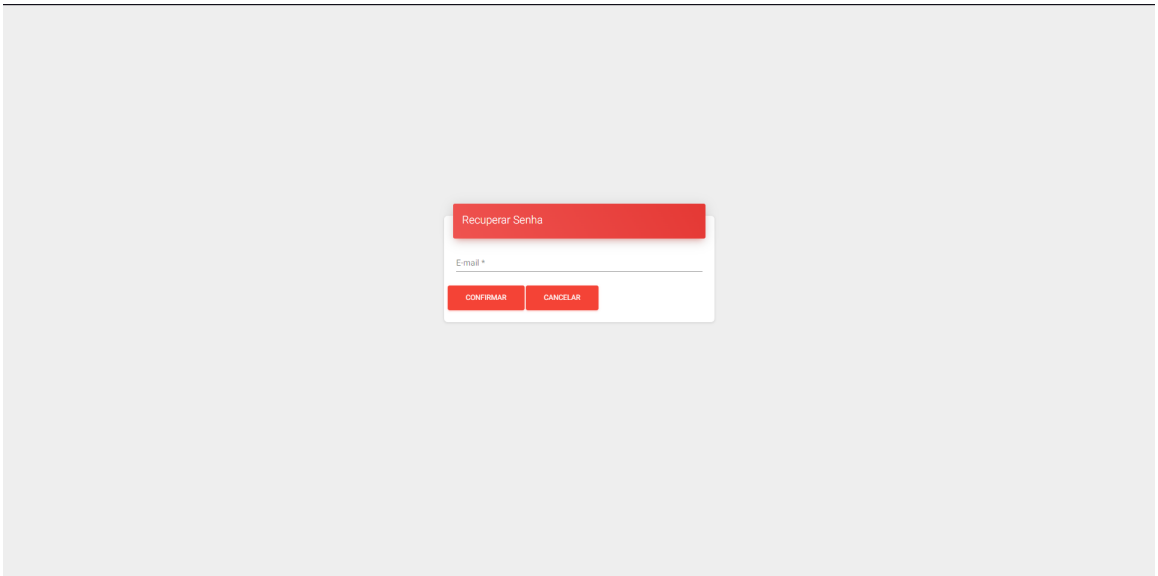
Fonte: Autoria própria (2022).

Figura 14 – Tela para registro de novo usuário



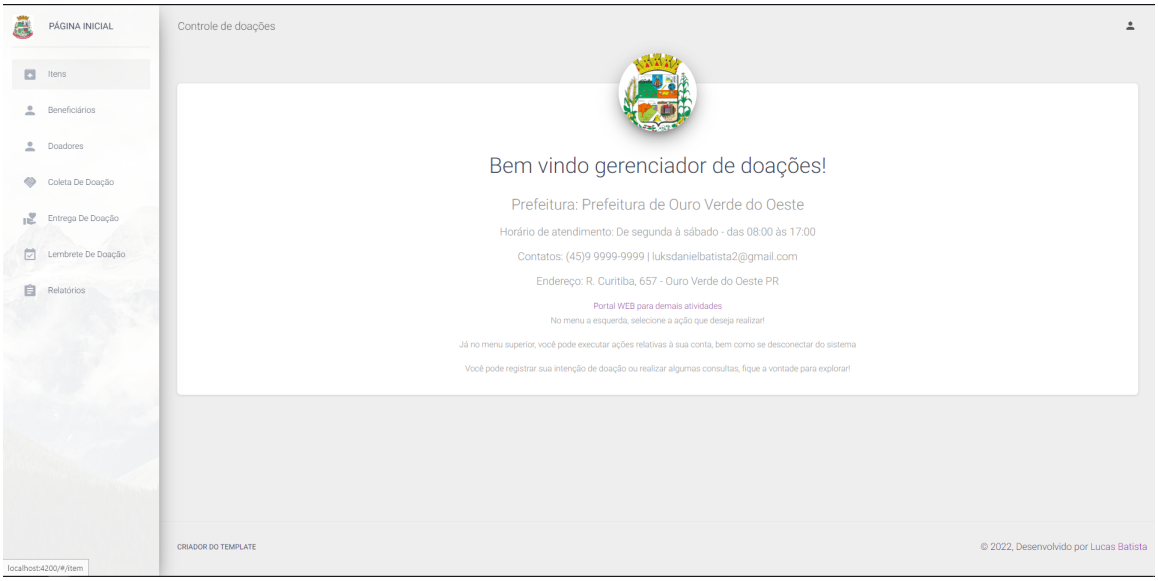
Fonte: Autoria própria (2022).

Figura 15 – Tela para recuperação de senha



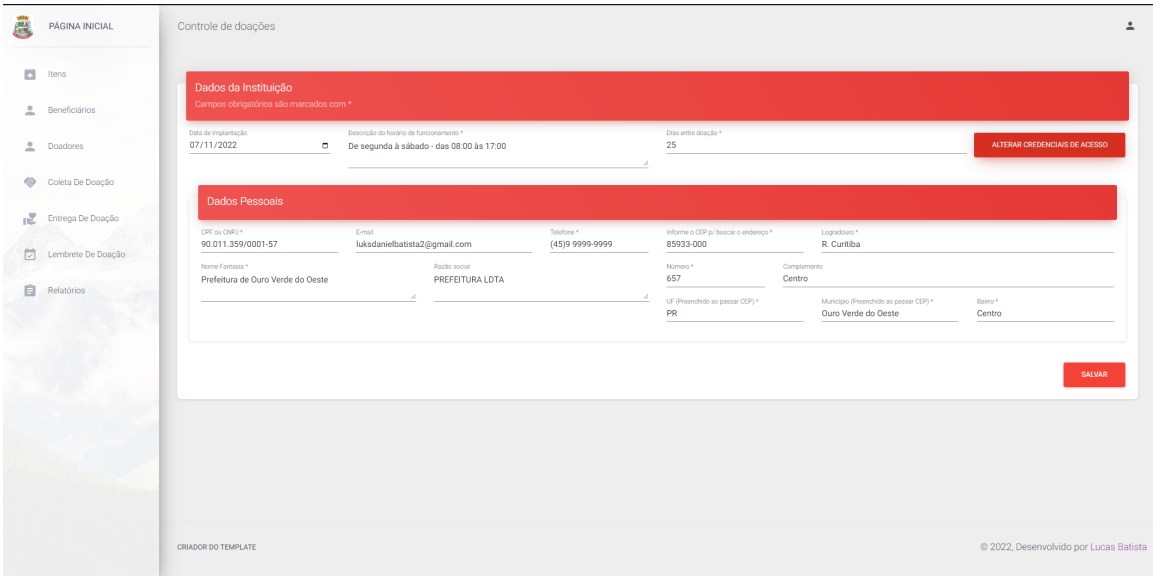
Fonte: Autoria própria (2022).

Figura 16 – Tela inicial do sistema



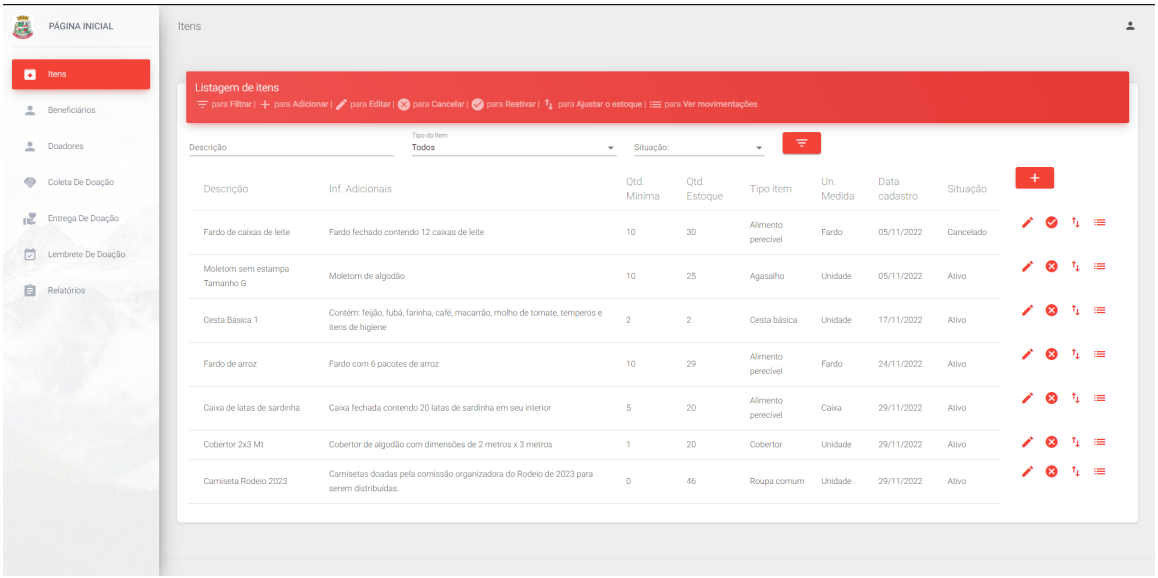
Fonte: Autoria própria (2022).

Figura 17 – Tela para alterar credenciais



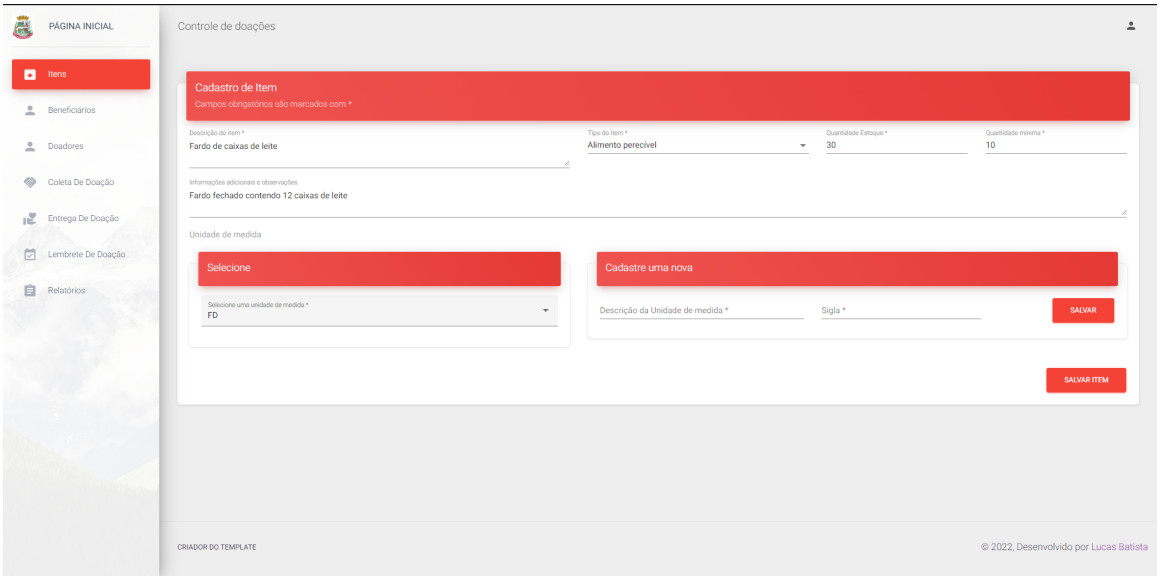
Fonte: Autoria própria (2022).

Figura 18 – Listagem de Itens



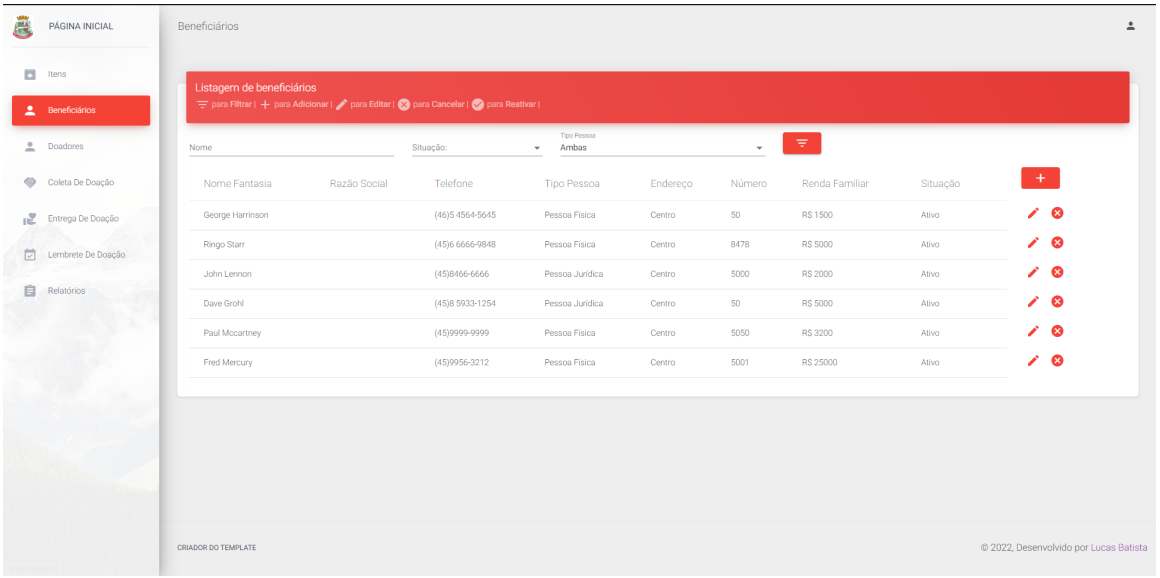
Fonte: Autoria própria (2022).

Figura 19 – Formulário de Item



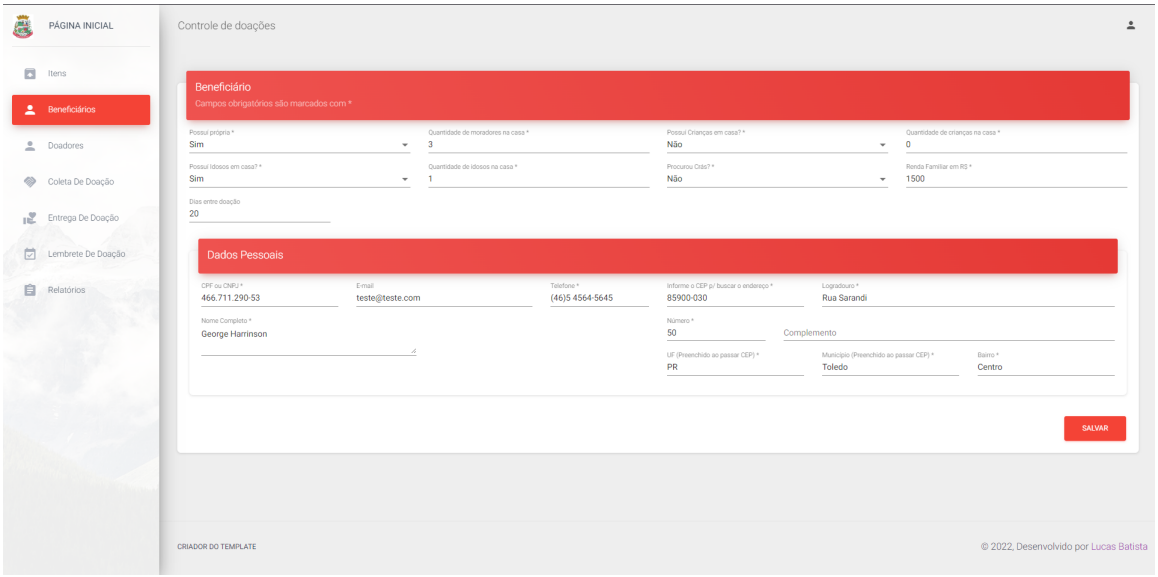
Fonte: Autoria própria (2022).

Figura 20 – Listagem de Beneficiários



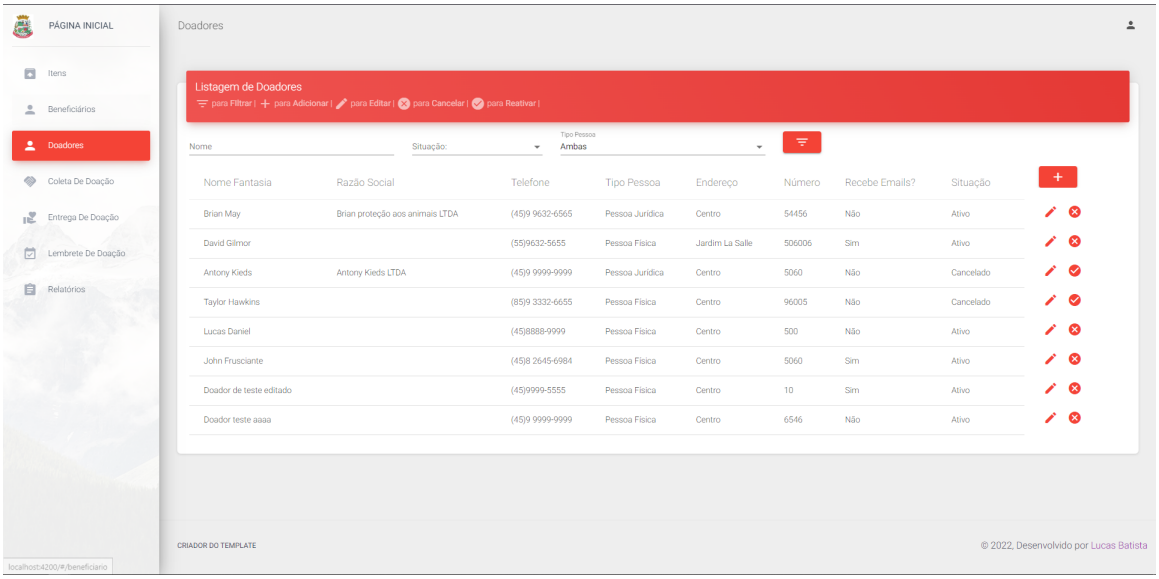
Fonte: Autoria própria (2022).

Figura 21 – Formulário de Beneficiário



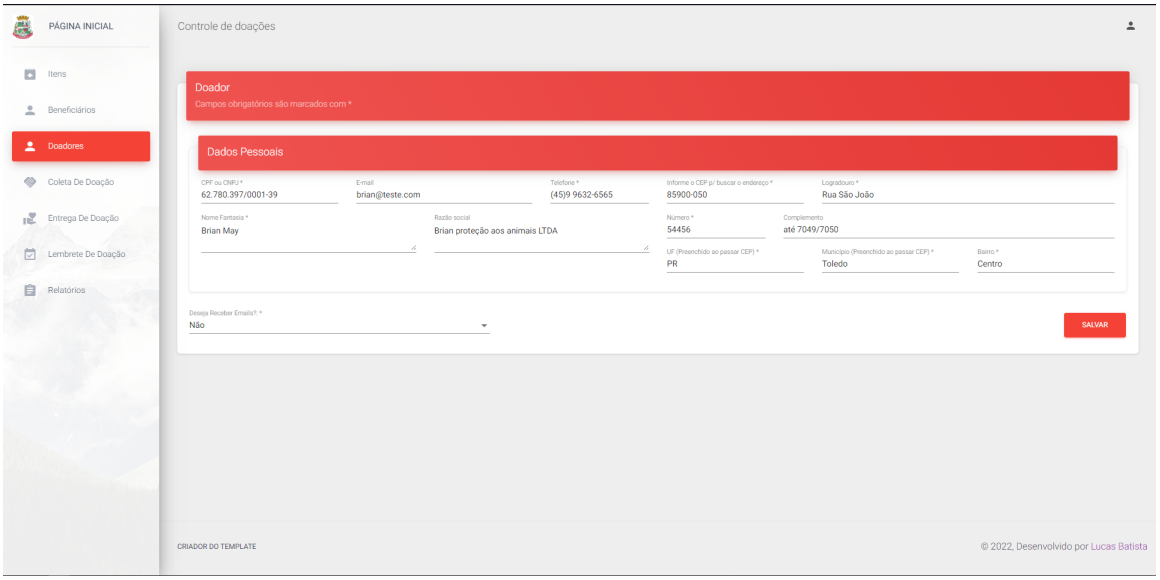
Fonte: Autoria própria (2022).

Figura 22 – Listagem de Doadores



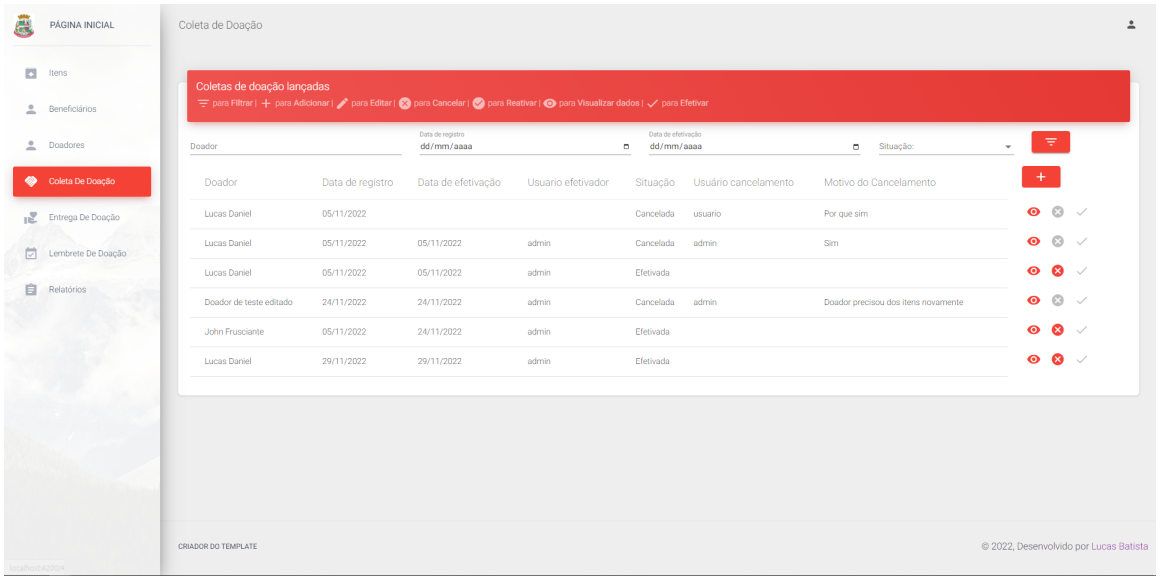
Fonte: Autoria própria (2022).

Figura 23 – Formulário de Doador



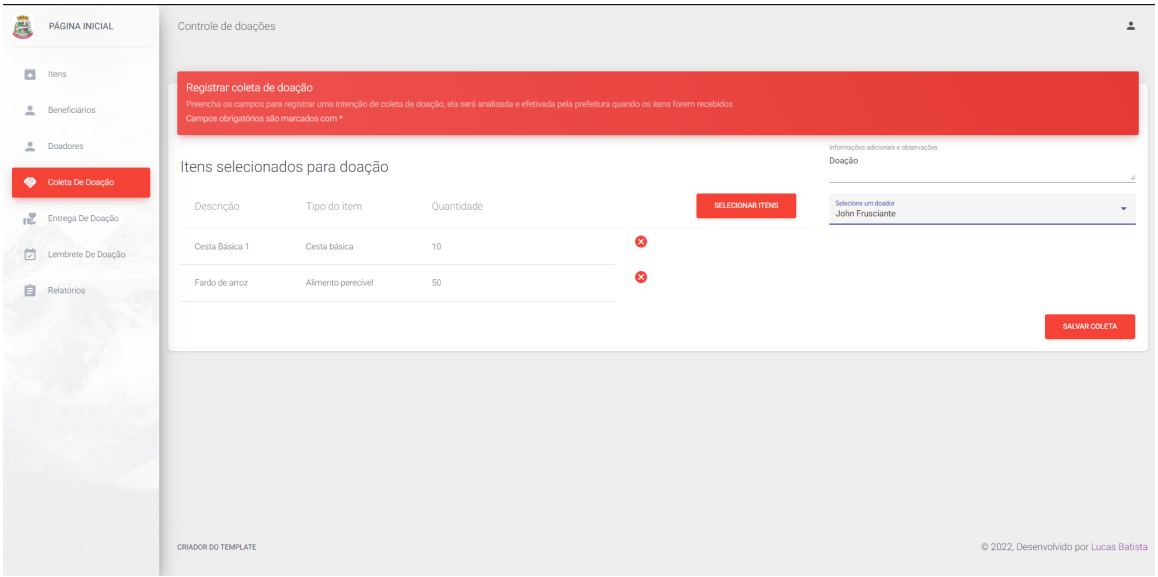
Fonte: Autoria própria (2022).

Figura 24 – Listagem de Coletas de Doação



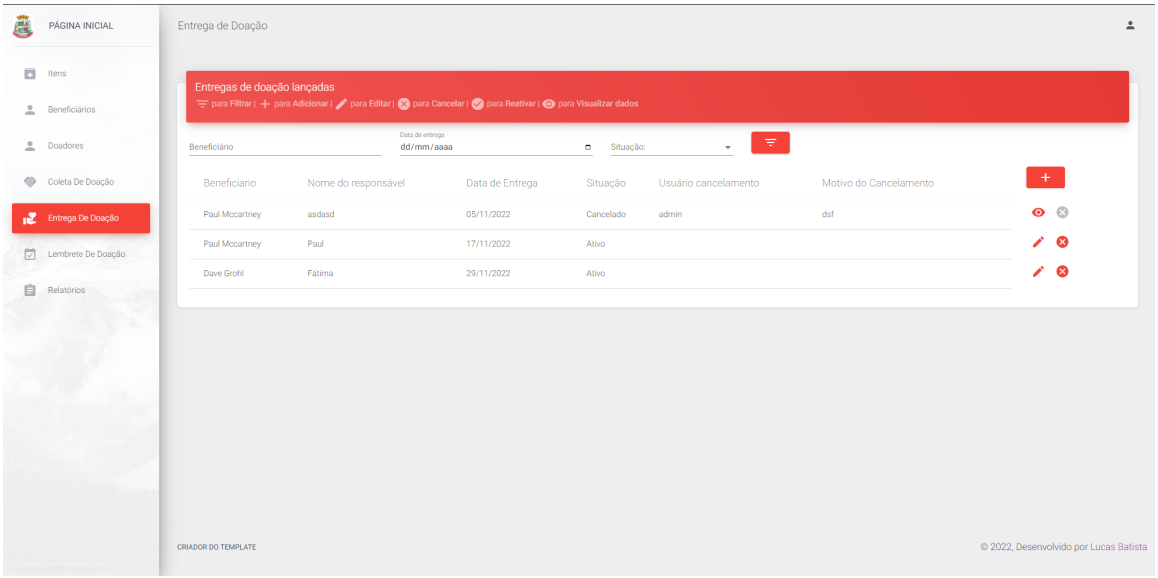
Fonte: Autoria própria (2022).

Figura 25 – Formulário de Coleta de Doação



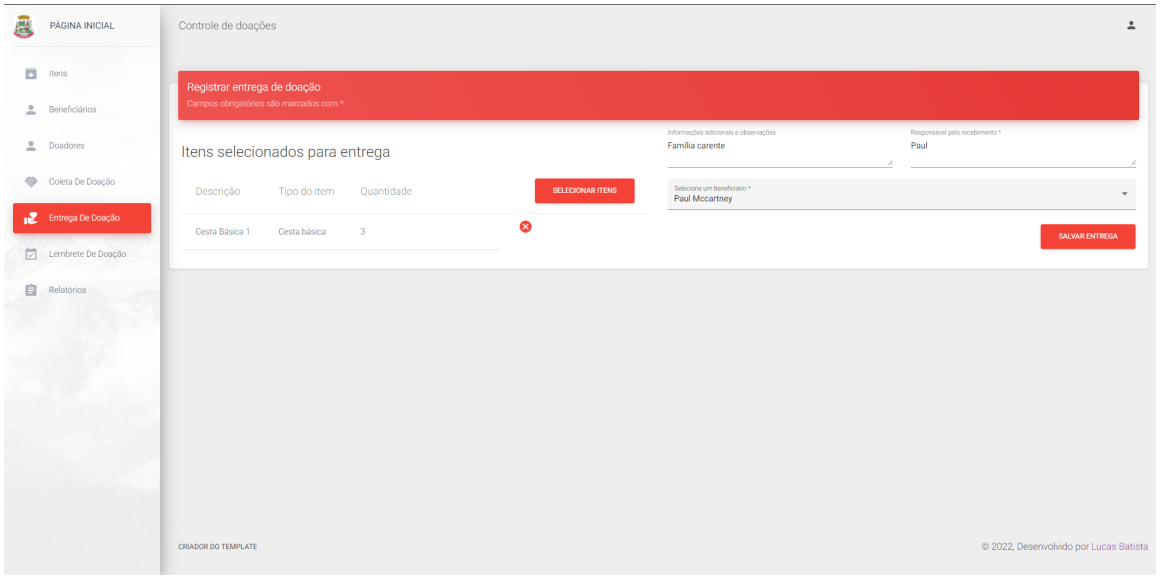
Fonte: Autoria própria (2022).

Figura 26 – Listagem de Entregas de Doação



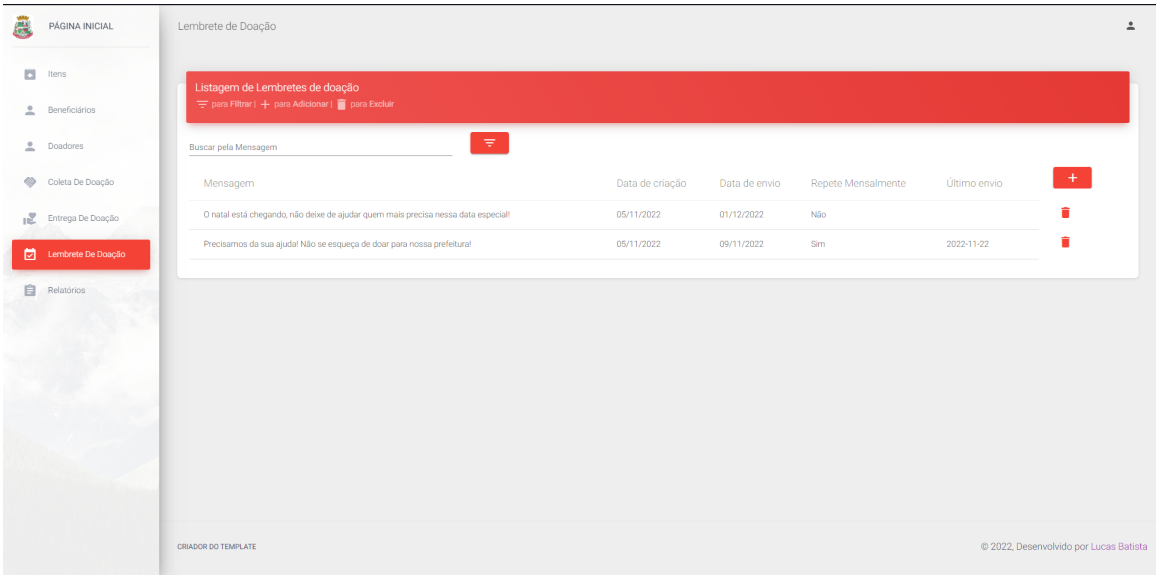
Fonte: Autoria própria (2022).

Figura 27 – Formulário de Entrega de Doação



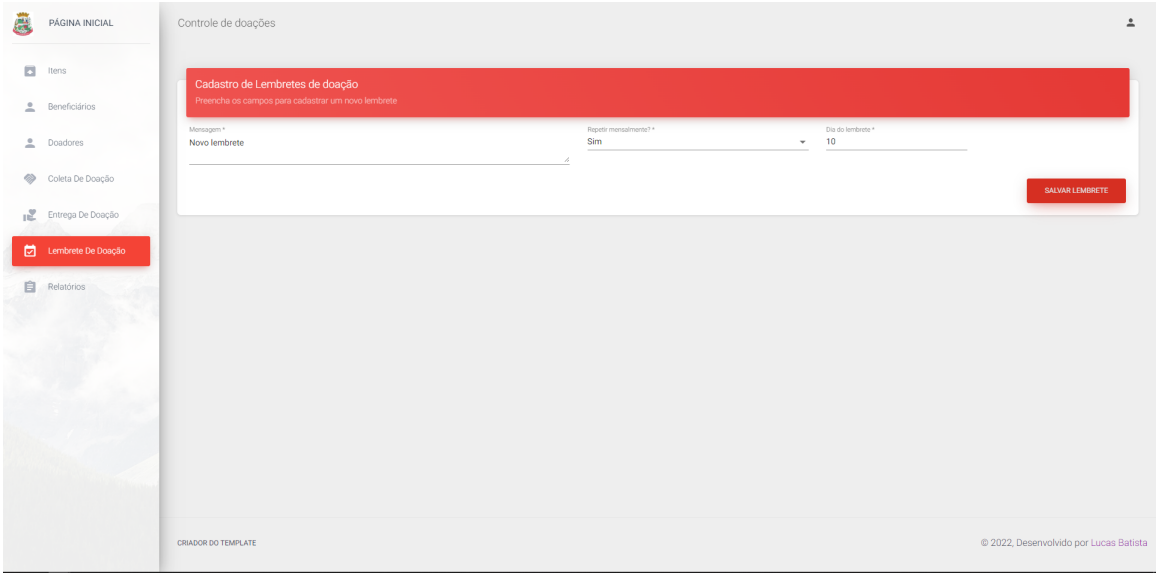
Fonte: Autoria própria (2022).

Figura 28 – Listagem de Lembretes de Doação



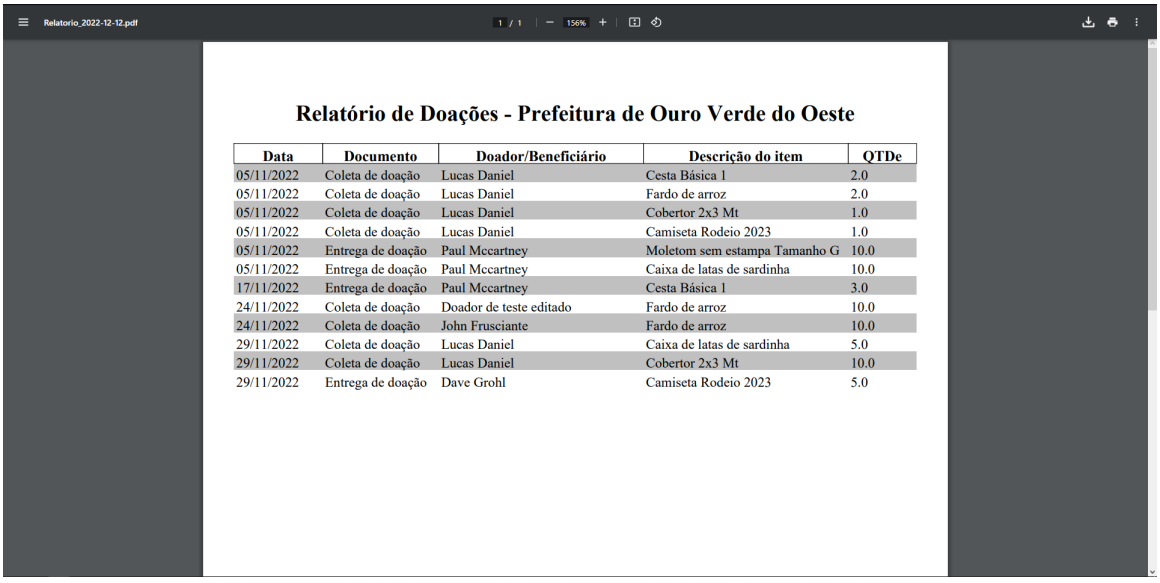
Fonte: Autoria própria (2022).

Figura 29 – Formulário de Lembrete de Doação



Fonte: Autoria própria (2022).

Figura 30 – Relatório de Coletas e Entregas



Fonte: Autoria própria (2022).