INSTITUTO FEDERAL GOIANO
CAMPUS URUTAÍ
NÚCLEO DE INFORMÁTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

GABRIEL DE OLIVEIRA DIAS, JOÃO VICTOR QUEIROZ SILVA

ONVET - APLICAÇÃO PARA GESTÃO DE BOVINOS

Urutaí, 22 de agosto de 2023

INSTITUTO FEDERAL GOIANO
NÚCLEO DE INFORMÁTICA
SISTEMAS DE INFORMAÇÃO

GABRIEL DE OLIVEIRA DIAS, JOÃO VICTOR QUEIROZ SILVA

ONVET - APLICAÇÃO PARA GESTÃO DE BOVINOS

Monografia apresentada ao Núcleo de Informática, curso de Sistemas de Informação, do Instituto Federal Goiano, como parte das exigências para obtenção do título de Bacharel em Sistemas de Informação.

Orientador(a):
Júnio César Lima

Urutaí, 22 de agosto de 2023

Sistema desenvolvido pelo ICMC/USP
Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas - Instituto Federal Goiano

D541o

Dias, Gabriel de Oliveira
ONVET - APLICAÇÃO PARA GESTÃO DE BOVINOS / Gabriel de Oliveira Dias; orientador Júnio César Lima. -- Urutaí, 2023.
98 p.

TCC (Graduação em Bacharel em Sistemas de Informação) -- Instituto Federal Goiano, Campus Urutaí, 2023.

1. Pecuária. 2. Desenvolvimento. 3. Laravel. 4. Modelagem. I. César Lima, Júnio , orient. II. Título.

Responsável: Johnathan Pereira Alves Diniz - Bibliotecário-Documentalista CRB-1 n°2376

Sistema desenvolvido pelo ICMC/USP
Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas - Instituto Federal Goiano

S586o

Silva, João Victor Queiroz
ONVET - APLICAÇÃO PARA GESTÃO DE BOVINOS / João Victor Queiroz Silva; orientador Júnio César Lima. -- Urutaí, 2023.
98 p.

TCC (Graduação em Bacharel em Sistemas de Informação) -- Instituto Federal Goiano, Campus Urutaí, 2023.

1. Pecuária. 2. Desenvolvimento. 3. Laravel. 4. Modelagem. I. César Lima, Júnio , orient. II. Título.

Responsável: Johnathan Pereira Alves Diniz - Bibliotecário-Documentalista CRB-1 n°2376

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO

PARA DISPONIBILIZAR PRODUÇÕES TÉCNICO-CIENTÍFICAS NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Com base no disposto na Lei Federal nº 9.610, de 19 de fevereiro de 1998, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano a disponibilizar gratuitamente o documento em formato digital no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

IDENTIFICAÇÃO DA PRODUÇÃO TÉCNICO-CIENTÍFICA

- ☐ Tese (doutorado)
- ☐ Dissertação (mestrado)
- ☐ Monografia (especialização)
- ☑ TCC (graduação)
- ☐ Artigo científico
- ☐ Capítulo de livro
- ☐ Livro
- ☐ Trabalho apresentado em evento
- ☐ Produto técnico e educacional - Tipo:

Nome completo do autor:
Gabriel de Oliveira Dias, João Victor Queiroz Silva

Matrícula:
2017101202010077, 201611202010

Título do trabalho:
ONVET - APLICAÇÃO PARA GESTÃO DE BOVINOS

RESTRIÇÕES DE ACESSO AO DOCUMENTO

Documento confidencial: ☑ Não ☐ Sim, justifique:

Informe a data que poderá ser disponibilizado no RIIF Goiano: 18 /08 /2023

O documento está sujeito a registro de patente? ☐ Sim ☑ Não

O documento pode vir a ser publicado como livro? ☑ Sim ☐ Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O(a) referido(a) autor(a) declara:

• Que o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;

• Que obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autoria, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;

• Que cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Urutaí 11 /08 /2023
Data

gov.br Documento assinado digitalmente
GABRIEL DE OLIVEIRA DIAS
Data: 14/08/2023 11:49:25-0300
Verifique em https://validar.iti.gov.br

gov.br Documento assinado digitalmente
JOAO VICTOR QUEIROZ SILVA
Data: 19/08/2023 11:29:09-0300
Verifique em https://validar.iti.gov.br

Assinatura do autor e/ou detentor dos direitos autorais

Ciente e de acordo:

gov.br Documento assinado digitalmente
JUNIO CESAR DE LIMA
Data: 21/08/2023 11:04:07-0300
Verifique em https://validar.iti.gov.br

Assinatura do(a) orientador(a)

GABRIEL DE OLIVEIRA DIAS
JOÃO VICTOR QUEIROZ SILVA

ONVET - APLICAÇÃO PARA GESTÃO DE BOVINOS

Monografia, defendida por Gabriel de Oliveira Dias e João Victor Queiroz Silva, apresentado ao Instituto Federal de Educação, Ciência e Tecnologia Goiano, como parte das exigências para a obtenção do título de Bacharel em Sistemas de Informação, aprovados pela banca examinadora.

COMISSÃO EXAMINADORA

Documento assinado digitalmente
gov.br JUNIO CESAR DE LIMA
Data: 26/06/2023 11:28:18-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Júnior César de Lima
Orientador

Documento assinado digitalmente
gov.br AMAURY WALBERT DE CARVALHO
Data: 26/06/2023 12:50:52-0300
Verifique em https://validar.iti.gov.br

Prof. Me. Amaury Walbert de Carvalho
Avaliador

Documento assinado digitalmente
gov.br CRISTIANE DE FATIMA DOS SANTOS CARD
Data: 27/06/2023 20:37:56-0300
Verifique em https://validar.iti.gov.br

Profa. Dra. Cristiane de Fátima dos Santos Cardoso
Avaliadora

Urutaí (GO), 26 de junho de 2023.

Dedicamos este trabalho aos nossos queridos familiares e amigos, que sempre nos apoiaram e encorajaram em nossa jornada acadêmica. Sem o amor, o carinho e o suporte que sentimos de vocês, certamente não teríamos chegado até aqui. Este trabalho é uma pequena demonstração de nossa gratidão e reconhecimento por tudo o que vocês representam em nossas vidas.

AGRADECIMENTOS

Agradecemos a todos que contribuíram para a realização deste trabalho.

RESUMO

A monografia apresenta o desenvolvimento de um software para gestão de gado leiteiro utilizando o *Framework* Laravel. O objetivo do trabalho é criar uma ferramenta que permite aos produtores de leite gerenciar de forma eficiente as informações do rebanho, visando melhorar a produtividade e rentabilidade do negócio. São considerados conceitos relacionados à pecuária leiteira, bem como o desenvolvimento WEB, padrões arquiteturais e linguagem de programação PHP. O *software* desenvolvido possui diversas funcionalidades, incluindo o cadastro de animais e funcionários, controle de produção de leite, gerenciamento de lotes de animais e áreas de pastagem, controle de fornecedores, entre outras. O código fonte está disponível em um repositório público no GitHub. O resultado final do trabalho é um sistema funcional e de fácil utilização, capaz de auxiliar os produtores de leite no gerenciamento de suas atividades.

Palavras-chave:

Pecuária. Desenvolvimento. Laravel. Modelagem. *Framework*.

ABSTRACT

The monograph presents the development of a software for dairy cattle management using the Laravel Framework. The objective of the work is to create a tool that allows dairy producers to efficiently manage herd information, aiming to improve productivity and business profitability. Concepts related to dairy farming, as well as web development, architectural patterns, and PHP programming language, were considered. The developed software features various functionalities, including animal and employee registration, milk production control, management of animal batches and pasture areas, supplier management, among others. The source code is available in a public repository on GitHub. The final result of the work is a functional and user-friendly system capable of assisting dairy producers in managing their activities.

Key words:

Livestock. Development. Laravel. Modeling. Framework.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Programming Interface*. Pag.24

CSS *Cascading Style Sheets*. Pag.16

DER Diagrama de Entidade e Relacionamento. Pag.38

EMBRAPA Empresa Brasileira de Pesquisa Agropecuária. Pag.13

FAO Organização das Nações Unidas para Agricultura e Alimentos. Pag.13

HTML *Hyper Text Markup Language*. Pag.16

HTTP *Hypertext Transfer Protocol*. Pag.15

HTTPS *Hypertext Transfer Protocol Secure*. Pag.15

JSON *JavaScript Object Notation*. Pag.17

MVC *Model, View, Controller*. Pag.19

PDF *Portable Document Format*. Pag.23

PHP *Personal Home Page*. Pag.16

REST *Representational State Transfer*. Pag.24

SQL *Structured Query Language*. Pag.42

TI Tecnologia da Informação. Pag.26

UML *Unified Modeling Language*. Pag.32

URL *Uniform Resource Locator*. Pag.15

VBP Valor Bruto da Produção. Pag.14

XLS *Excel Spreadsheet*. Pag.23

XML *Extended Markup Language*. Pag.17

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

A pecuária leiteira é uma importante atividade econômica no Brasil, representando uma parcela significativa do agronegócio nacional. Com uma demanda crescente por alimentos de qualidade e segurança, é fundamental que os produtores adotem medidas que garantam a eficiência da gestão do rebanho e, consequentemente, a qualidade do leite produzido. Nesse contexto, a tecnologia tem se mostrado uma importante aliada na busca por soluções mais eficientes e rentáveis para a atividade leiteira. Dentre as tecnologias disponíveis, destaca-se o desenvolvimento de *softwares* específicos para a gestão do gado leiteiro.

A Organização das Nações Unidas para Agricultura e Alimentos (FAO) afirma que a produção global de leite foi estimada em 929,9 milhões de toneladas em 2022 (FAO. . . , 2022). Segundo o site Agrofy *News* (AGROFY, 2022), a Índia é o maior produtor de leite do mundo, com cerca de 221 milhões de toneladas em 2022, seguida pela União Europeia e pelos Estados Unidos. O Brasil, por sua vez, ocupa atualmente a 6ª colocação, produzindo aproximadamente 34,8 milhões de toneladas. Isso evidencia a importância da produção de leite no país e a necessidade de pequenos e grandes produtores em gerir dados para aumentar a produção e medir sua qualidade.

Segundo dados da EMBRAPA (Empresa Brasileira de Pesquisa Agropecuária) (AGROPECUARIA, 2021), o leite está entre os seis produtos mais importantes do ramo agropecuário, sendo que quase 47% do volume total produzido no país vem de pequenas fazendas. Além disso, a produção leiteira não tem apenas um papel econômico, mas, também social e nutricional. O setor gera empregos no país e contribui para o crescimento e a manutenção de uma vida saudável (COOPERATIVA, 2018). Nos últimos anos, a indústria do leite avançou muito no

Brasil, passando a representar cerca de 24 por cento do VBP (Valor Bruto da Produção) gerado pela pecuária (COOPERATIVA, 2018).

De acordo com (DIARURAL, 2021), os avanços tecnológicos têm promovido melhorias significativas na eficiência do setor, refletindo em aumentos na produção de leite por hectare e na média diária de produção. Além disso, a incorporação de novas tecnologias tem contribuído para a redução de custos de produção, aumento do controle de qualidade e segurança do leite, bem como para a melhoria da produtividade e rentabilidade da atividade a médio e longo prazos. *Softwares* de gestão têm sido utilizados para gerenciar e comparar índices produtivos do rebanho, auxiliando em uma tomada de decisão mais assertiva e resultando em maior lucratividade para o produtor.

Pensando nesta evolução do setor, este trabalho apresenta o desenvolvimento de um *software* para gestão de gado leiteiro, utilizando o *framework* Laravel. O objetivo do trabalho foi criar uma ferramenta que permitisse aos produtores de pequeno ou grande porte gerenciar de forma eficiente as informações do rebanho, visando melhorar a produtividade e rentabilidade do negócio. O *software* desenvolvido possui diversas funcionalidades, incluindo o cadastro de animais e funcionários, controle de produção de leite, controle de lotes e pastagens entre outras.

O código-fonte desenvolvido está disponível em um repositório público no GitHub, permitindo que outros desenvolvedores possam contribuir para a evolução do projeto. Dessa forma, espera-se que este trabalho possa contribuir para o avanço da pecuária leiteira, oferecendo uma ferramenta eficiente e de fácil utilização para os produtores de leite.

Este trabalho está organizado em cinco capítulos, sendo o primeiro a presente introdução. O Capítulo 2 apresenta trechos relevantes referentes às leituras realizadas no embasamento teórico, para que o leitor conheça sobre a pecuária leiteira e a tecnologia utilizada no desenvolvimento do *software*. O Capítulo 3 revela ao leitor qual metodologia de desenvolvimento foi utilizada e como foram feitos os levantamentos de requisitos. O Capítulo 4 apresenta a modelagem do sistema, trazendo ao leitor uma definição e explicação sobre o Diagrama de Entidade, Relacionamento e Diagrama de Pacotes. Já o Capítulo 5 apresenta como foi feita toda a implementação do sistema em si, revelando os principais trechos de código e as principais funcionalidades do sistema. Por fim, é apresentada a conclusão e trabalhos futuros.

CAPÍTULO 2

FUNDAMENTOS TEÓRICOS

Neste capítulo, são apresentados trechos relevantes referentes às leituras realizadas no embasamento teórico. Serão abordados os principais assuntos relacionados à tecnologia utilizada no desenvolvimento da aplicação e também ao problema abordado. Serão discutidos conceitos relacionados ao desenvolvimento WEB, serviços WEB, aos tipos de padrões arquiteturais existentes na construção de um *software*, à linguagem de programação utilizada e ao *Framework* utilizado.

2.1 Desenvolvimento Web

No desenvolvimento para WEB, o foco está em como desenvolver uma aplicação correta e completa, de acordo com os requisitos do usuário. O diferencial está no fato de que esta deve ser desenvolvida no contexto de um projeto que deve considerar a infraestrutura WEB para sua execução e disponibilização (CONTE et al., 2005).

Inicialmente, para o desenvolvimento de uma aplicação WEB, é necessário entender o que são os protocolos HTTP (*Hypertext Transfer Protocol*) e HTTPS (*Hypertext Transfer Protocol Secure*) em um sistema. Uma aplicação WEB tem como característica a utilização de protocolos, na maioria das vezes o HTTP (LARISSA GASPAR, 2022) ou HTTPS. O protocolo HTTP possibilita que o cliente insira a URL (*Uniform Resource Locator*) que deseja acessar e garante que os conteúdos presentes sejam apresentados. Quando o cliente acessa uma determinada URL, o navegador cria solicitações HTTP e as envia para o destino que está presente na URL, e então o servidor que representa aquela URL responde à requisição enviando os dados necessários como resposta. O protocolo HTTPS é semelhante ao HTTP, porém o seu diferencial é a comunicação

criptografada, que aumenta a segurança durante a troca de dados.

Para que uma aplicação esteja acessível através de um navegador, é necessário a sua execução em um servidor, que será responsável pelo processamento das informações recebidas através do protocolo HTTP/HTTPS. No servidor, o *hardware*, que é a parte física de um computador, e o *software*, que é a parte lógica do computador, trabalham juntos para que seja possível o armazenamento de dados, a execução de tarefas e a interação com o cliente. É possível citar, por exemplo, o servidor Apache (**https://www.apache.org/**).

Para definir para quais páginas ou caminhos as URLs irão levar o usuário que realiza uma requisição ao servidor, são utilizadas rotas. A *Framework* Laravel, possui o arquivo **web.php**, que faz o tratamento de rotas, este é encontrado no diretório **routes/web.php**. Na Figura 2.1 é mostrada a definição da rota principal do projeto, em que a URL principal traz o conteúdo do arquivo **frontend.index**.

```
Route::get('/', function () {
    return view('frontend.index');
});
```

Figura 2.1: Rota principal contida no arquivo **web.php**.

Geralmente, o desenvolvimento WEB é dividido em *back-end* e *front-end*. O *back-end*, parte responsável por carregar a lógica do sistema e processar todas as requisições recebidas, foi desenvolvido em PHP (*Personal Home Page*) , linguagem de programação mundialmente conhecida, utilizada para a construção de sites dinâmicos e ágeis que funcionam em grande parte dos sistemas operacionais. Esta linguagem traz como um de seus pontos positivos o fácil manuseio, alta utilização no mercado de trabalho e documentação abrangente, o que faz com que seja escolhida pelos desenvolvedores, visando rapidez e praticidade na hora de desenvolver seus *softwares* (CARLOS ESTRELLA, 2022). O PHP é uma linguagem de programação de código aberto, amplamente utilizada para o desenvolvimento de aplicações WEB do lado do servidor. De acordo com (W3TECHS, 2023), esta linguagem é utilizada por cerca de 79,1% dos websites que utilizam alguma linguagem de programação no servidor, tornando-o uma das linguagens mais populares para o desenvolvimento WEB.

O *front-end*, parte responsável por apresentar ao cliente as páginas WEB que contém recursos gráficos para que seja possível a sua interação com o sistema, foi desenvolvido utilizando HTML (*Hyper Text Markup Language*) , CSS (*Cascading Style Sheets*) e JavaScript. O

HTML é a linguagem padrão utilizada para exibir páginas WEB e tem suas linhas de código interpretadas diretamente pelo navegador, mostrando o resultado final sem a necessidade de compilação (COSTA, 2007). O CSS é responsável por acrescentar um estilo ao HTML, através de elementos de *design* para formatação de seções e mudança de cores, garantindo assim uma melhor apresentação ao usuário. O JavaScript é uma linguagem de *script* que permite a criação de páginas interativas, possibilitando o enriquecimento de uma página WEB executando tarefas importantes como exibir mensagens ou diferentes efeitos dinâmicos, como, por exemplo, uma apresentação de slides.

2.2 *Web Services*

Ao falar sobre o desenvolvimento de uma aplicação WEB um dos principais recursos utilizados são os *Web Services*, um serviço capaz de interagir com diferentes sistemas utilizando padrões e protocolos de comunicação de aplicação para aplicação. Para que qualquer sistema possa utilizar um determinado *Web Service*, o mesmo precisa oferecer uma estrutura baseada em padrões (CURBERA et al., 2001). Aplicações podem utilizar serviços WEB de diferentes formas, seja enviando dados e aguardando o processamento e, respectivamente, a resposta relacionada a esses dados ou executando-os na própria nuvem (TAURION, 2009).

Web services utilizam XML (*Extended Markup Language*) e JSON (*JavaScript Object Notation*) como formatos de dados para permitir a comunicação entre diferentes sistemas de *software* de forma eficiente e interoperável. Ambos os formatos tem como objetivo principal representar e estruturar dados de forma legível para máquinas, facilitando a troca de informações entre aplicações em diferentes plataformas e linguagens de programação.

O XML é um formato de arquivo que possibilita que ambas as partes de diferentes sistemas utilizem um esquema pré-definido para se comunicarem. Este arquivo XML pode conter várias informações diferentes, como dados e transmissão de sua estrutura (ALMEIDA, 2002).

O JSON é um arquivo que contém apenas texto e é semelhante ao XML, porém, mais simples e mais leve, utilizando uma sintaxe composta de chaves e valores, como é possível observar na Figura 2.2, que apresenta um exemplo de arquivo JSON consumido pelo sistema. Para a construção deste projeto, foi escolhido o JSON, pois encaixa-se melhor nos padrões e tecnologias utilizadas.

Figura 2.2: Arquivo JSON que contém informações através do CEP inserido pelo usuário.

2.3 Padrões Arquiteturais

Para que seja possível o desenvolvimento de um sistema protegido e bem estruturado, é necessário utilizar algum padrão arquitetural. Os padrões arquitetônicos trazem diferentes formas de organização, definindo uma estrutura fundamental do sistema. Um sistema pode ter mais de um padrão arquitetônico, dependendo do seu nível de complexidade. A escolha deste padrão deve ser feita levando em consideração o tipo do sistema, seus requisitos funcionais e não funcionais (FIGUEIREDO, 2019). Dentre os padrões arquitetônicos, é possível citar alguns dos principais, tais como:

***Layers* (Camadas)** - Aplicações com esta estrutura apresentam módulos hierárquicos, onde cada camada oferece seus serviços à camada acima dela, servindo também como cliente para a camada inferior.(GUILHERME MANZANO, 2020).

Cliente-Servidor - Neste padrão, ocorre a divisão do processamento da informação, sendo o servidor responsável por executar serviços e retornar os resultados, enquanto o cliente é responsável por enviar as requisições ao servidor e aguardar sua resposta.

***Microservices* (Microsserviços)** - Os microsserviços facilitam a escalabilidade e agilidade do desenvolvimento de aplicações, permitindo a criação de uma aplicação com componentes independentes que executam cada processo como um serviço. Isso permite que cada serviço possa ser atualizado, escalado ou aprimorado de diferentes formas (AMAZON, 2022).

MVC (***Model, View, Controller***) - Neste padrão, a aplicação é dividida em três camadas, a fim de permitir a reutilização de código, visando baixo acoplamento e alta coesão.

Destacando o padrão arquitetural MVC, ao utilizar esta arquitetura são apresentadas ao usuário diferentes interpretações de um mesmo modelo, facilitando a construção da interface do sistema. Neste padrão, a interface, os controladores e as entradas do usuário são tratados por três tipos de objetos chamados de *Model*, *View* e *Controller*, onde cada um realiza suas tarefas específicas (LUCIANO; ALVES, 2017).

No **Modelo** (*Model*), ocorre contato direto com o banco de dados utilizando classes de entidade. Ele não é uma classe que somente armazena ou consulta dados, pois na sua composição podem haver regras de negócio. O Modelo carrega a lógica da aplicação e a persistência no banco de dados, recebendo requisições dos controladores e gerando respostas a partir destas requisições (MARCIO, 2011). Esta camada também é chamada de camada de negócios.

Na **Visão** (*View*), contém a forma exata de como o modelo deve ser apresentado, ou seja, esta é a interface do usuário. Ela é dinâmica, podendo alterar conforme o usuário interage com o sistema, e é por ela que o mesmo irá inserir todos os dados necessários e também receber as respostas de suas requisições. Não contém lógica de negócios, diferente da camada de modelo, sendo também chamada de camada de apresentação.

O **Controlador** (*Controller*) é o intermediário entre a visão e o modelo. Ele permite a comunicação entre todos os componentes da arquitetura, traduzindo as interações do usuário com a visão para que seja possível mapear quais tarefas o modelo irá realizar. Com a utilização do controlador, é possível reutilizar o mesmo modelo para diferentes visualizações, eliminando dependências entre o modelo e a visão.

A Figura 2.3 exemplifica o ciclo da arquitetura MVC, no qual a interface aciona os métodos dos controladores, que enviam as requisições para o modelo. Após alterar ou consultar o banco de dados, o modelo envia sua resposta para o controlador, que por sua vez atualiza a visão e apresenta os dados ao usuário. O *Framework* escolhido para construção do projeto utiliza este padrão arquitetural.

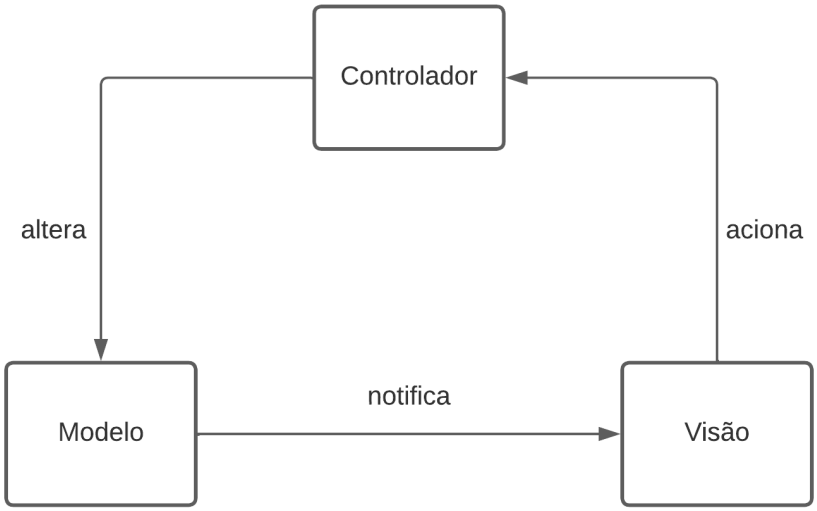

Figura 2.3: Ciclo da arquitetura MVC, *Model*, *View* e *Controller*.
Adaptada de: (BUCANEK, 2009)

2.4 *Framework* Laravel

Frameworks são um agrupamento de códigos genéricos que possuem a finalidade de facilitar o desenvolvimento de sistemas e aplicações. Assim como um modelo, um *Framework* busca oferecer diversas ferramentas e artifícios básicos para que um desenvolvedor possa criar um *software* (BRUNA B. BARRO, 2022).

Laravel é um *Framework* que utiliza como base a linguagem PHP, utilizado para desenvolvimento WEB, e para isso, utiliza a arquitetura MVC e o padrão PSR-2 como guia para estilizar o código (DHYOGO ALMEIDA, 2022; DEVMEDIA, 2022). O objetivo principal da arquitetura é fornecer ao desenvolvedor ferramentas de fácil utilização, enquanto que a modelagem cuida de itens como a injeção de dependências (LARAVEL, 2022).

Para utilizar o Laravel, é necessário o uso de componentes, tais como *Composer*, *Artisan*, *Blade* e RESTful. Esses componentes serão explicados nas próximas seções. A instalação do Laravel é realizada via comando, como mostrado na Figura 2.4 e deve ser feita após a instalação do *Composer*.

```
composer global require "laravel/installer=~1.1"
```

Figura 2.4: Comando para instalação do Laravel via *Composer*.
Fonte:(LARAVEL, 2022)

2.4.1 *Composer*

Para gerenciar as dependências que surgiram com a evolução da linguagem PHP, foi criada uma ferramenta chamada *Composer*. Essa ferramenta permite ao desenvolvedor declarar as bibliotecas das quais o seu projeto depende e é responsável pela gestão dessas bibliotecas (SHAHZED AHMED, 2021).

Existem comandos que são frequentemente usados por profissionais que optam por usar o *Framework* Laravel. Grande parte desses comandos depende da instalação prévia do *Composer* na máquina. A Figura 2.5 apresenta o comando necessário para instalar o *Composer*.

```
PS C:\wamp64\www\Onvet-4-master> composer install
Installing dependencies from lock file (including require-dev)
Verifying lock file contents can be installed on current platform.
Package operations: 142 installs, 0 updates, 0 removals
  - Installing symfony/polyfill-php80 (v1.26.0): Extracting archive
  - Installing symfony/deprecation-contracts (v3.1.1): Extracting archive
  - Installing symfony/finder (v5.4.11): Extracting archive
  - Installing symfony/polyfill-mbstring (v1.26.0): Extracting archive
```

Figura 2.5: Comando `composer install` para instalação do composer.

Um dos principais comandos do *Composer* é o de criação de projetos. Com este comando, o *Composer* cria para o desenvolvedor um projeto Laravel, utilizando todos os padrões, bibliotecas e extensões necessários para o desenvolvimento. A Figura 2.6 mostra o comando utilizado para a criação de projetos. Nele, especificamos o diretório onde o projeto será salvo e a versão do Laravel escolhida.

```
composer create-project laravel/laravel {directory} 4.2 --prefer-dist
```

Figura 2.6: Comando utilizado para criar um novo projeto Laravel com o *Composer*.
Fonte:(LARAVEL, 2022)

Em uma arquitetura Laravel, existe sempre um arquivo chamado **composer.json**. Neste arquivo, são listadas todas as dependências e informações sobre o projeto, além de conter propriedades comuns e metadados. Por esse motivo, este arquivo é incluso por padrão nos modelos de arquitetura Laravel (PEACHPIE, 2022). A Figura 2.7 exemplifica um arquivo **composer.json**, contendo comandos que são utilizados para evitar o processo de carregamento automático durante a compilação.

```
{
    "autoload": {
        "psr-4": {
            "Vendor\\Namespace\\": "src/",
            "": "src/",
        },
        "psr-0": {
            "Vendor\\Namespace\\": "src/",
            "Vendor_Namespace_": "src/"
        },
        "classmap": ["src/", "lib/", "Something.php"],
        "exclude-from-classmap": ["/src/Tests/", "/lib/tests/"],
        "files": [],
    }
}
```

Figura 2.7: Comandos contidos em um arquivo JSON.
Fonte: (PEACHPIE, 2022)

2.4.2 *Artisan*

Artisan é o nome dado à interface de linha de comando que está incluída no Laravel. Conduzido pelo componente *Symfony Console*, essa interface permite realizar várias ações importantes, tais como: verificar rotas existentes e criar vários arquivos dentro do projeto, como *migrations*, *controllers* e *models*.

Dentre as várias possibilidades que o *artisan* proporciona ao desenvolvedor, uma das mais utilizadas é a facilidade de interação com o projeto, para isso, o mesmo permite criar um servidor de desenvolvimento, isto é, ele cria um ambiente que facilita o desenvolvimento de testes e sites (DEVMEDIA, 2022). A Figura 2.8 mostra o comando utilizado para iniciar o servidor, após iniciado, basta copiar a URL indicada no terminal em um navegador para ter acesso ao site.

```
PS C:\wamp64\www\OnVetPlus> php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Sun Oct 30 21:58:44 2022] PHP 8.1.6 Development Server (http://127.0.0.1:8000) started
```

Figura 2.8: Comando *Artisan* responsável por iniciar o servidor local.

Para o acesso à lista total de comandos do *Artisan*, o mesmo disponibiliza um comando que facilita a busca por uma funcionalidade específica. A Figura 2.9 traz a utilização do comando `list`, que tem o objetivo de mostrar ao desenvolvedor quais são e qual a funcionalidade de cada comando existente no *Artisan*.

```
PS C:\wamp64\www\OnVetPlus> php artisan list
Laravel Framework 8.83.23

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display help for the given command. When no command is given display help for the list command
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi|--no-ansi  Force (or disable --no-ansi) ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --env[=ENV]       The environment the command should run under
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

Figura 2.9: Comando `list`, utilizado para listar os comandos do artisan.

Buscando facilitar a interação com o banco de dados, o *Artisan* traz um comando que realiza a migração entre as classes *migrations* do Laravel e o servidor de banco de dados. As *migrations*, por sua vez, são classes que tratam da manipulação de dados, criando, alterando e removendo tabelas. A Figura 2.10 exemplifica a utilização do comando *migrate*. Como não houve nenhuma alteração nas classes *migrations*, nada foi inserido, editado ou removido.

```
PS C:\wamp64\www\OnVetPlus> php artisan migrate
Nothing to migrate.
PS C:\wamp64\www\OnVetPlus>
```

Figura 2.10: Comando `artisan migrate` utilizado para realizar as migrações.

2.4.3 *Blade*

O Laravel possui seu próprio mecanismo de templates chamado *blade*. Um template tem o objetivo de otimizar a performance de um site, por meio de algumas funcionalidades, como por exemplo, a compilação de arquivos para evitar a interpretação de códigos PHP a cada requisição e também melhorar sua aparência, ou seja, trazer aspectos visuais mais bonitos.

O *blade* não se restringe apenas ao uso da linguagem PHP, mas também a CSS, JavaScript e HTML, ampliando as funcionalidades e facilitando o desenvolvimento de telas mais bonitas e eficientes. Os arquivos *blade* são compilados e armazenados em cache, tornando o carregamento mais leve e evitando sobrecarga na aplicação. Arquivos desse tipo são declarados com a extensão **.blade.php** e, seguindo as boas práticas, geralmente são armazenados no diretório **resources/views**.

A Figura 2.11 apresenta um exemplo de arquivo *blade* utilizado na aplicação para listar os dados da tabela áreas e realizar algumas operações, como exclusão de dados e geração de arquivos nos formatos PDF (*Portable Document Format*) e XLS (*Excel Spreadsheet*) para o

utilização do usuário. O arquivo está localizado no diretório **resources/views** e utiliza a extensão **.blade.php.** As funcionalidades são implementadas utilizando a sintaxe do *blade* combinada com a linguagem PHP.

```
@extends('layouts/contentLayoutMaster')

@section('title', 'Áreas')

@section('content')
    <div class="row" id="table-hover-row">
        <div class="col-12">
            <div class="card">
                <div class="card-header">
                    <h4 class="card-title">Listagem de Áreas</h4>
                    <div style="float: right;">
                        <a href="{{ url('cadastros/areas/create/0') }}" class="btn btn-outline-primary waves-effect">
                            <i data-feather="plus-circle" class="mr-50"></i>
                            <span>Novo</span>        You, há 6 dias • atualizacao_crud_gabi
                        </a>
                    </div>
                </div>
                <div class="card-body">
                    @include('shared.alerts')
                    <form class="form-horizontal form-label-left" id="formSearch" method="post">
                        {!! csrf_field() !!}
                        <input type="hidden" name="page" id="page" value="{{ $request->page or ' ' }}" />
                        <div class="row d-flex">
                            <div class="col-md-4 col-12">
                                <div class="form-group">
                                    <input type="text" class="form-control" placeholder="Busca Livre..." id="search"
```

Figura 2.11: Exemplo de arquivo *blade* utilizado para listar dados e realizar operações na tabela áreas.

2.4.4 API

Uma API (*Application Programming Interface*) é uma interface de programação de aplicação, onde os programadores podem definir regras para que seus sistemas possam se comunicar com outros. Este conjunto de regras possibilita a comunicação entre plataformas, tendo como principal objetivo simplificar o trabalho de desenvolvedores, permitindo que os mesmos possam desenvolver *softwares* que se comuniquem com outros de maneira mais rápida (CLARA FABRO, 2020).

Em vários sistemas robustos encontrados hoje no mercado, pode-se notar a necessidade de comunicação com sistemas internos ou de terceiros, em um *software* de negócios, por exemplo, existe a necessidade de comunicação com um *software* para aprovação de notas fiscais, e para isso, foram criados os padrões arquiteturais REST (*Representational State Transfer*) e RESTFul, que impõe condições sobre como uma API deve funcionar (AMAZON AWS, 2022).

REST é um estilo arquitetural usado para projetar serviços WEB escaláveis, interoperáveis e flexíveis. Baseado em princípios fundamentais, como arquitetura cliente-servidor, ausência de estado, operações bem definidas, recursos facilmente identificáveis e uso de representações padronizadas, o REST possibilita uma comunicação efetiva entre sistemas distribuídos. Os benefícios incluem escalabilidade, simplicidade e independência de linguagem, permitindo a

evolução autônoma dos componentes do sistema.

RESTful refere-se aos princípios REST implementados por meio de uma API. Seguindo as diretrizes e restrições do REST, uma API RESTful permite que os clientes interajam com os recursos por meio de solicitações HTTP, utilizando métodos como *GET, POST, PUT* e *DELETE*. Essa abordagem facilita a criação de serviços WEB padronizados, compatíveis e de fácil integração.

Entre os benefícios de utilizar o padrão RESTful, existem três que destacam-se: escalabilidade, flexibilidade e independência. O RESTful otimiza as interações entre cliente e servidor, permitindo que sistemas possam ser escalados com eficiência. Além disso, simplifica vários componentes do servidor, permitindo que as partes evoluam independentemente, tornando o sistema mais flexível. Por fim, permite ao desenvolvedor utilizar diferentes linguagens de programação no desenvolvimento de aplicações cliente-servidor, sem afetar a API.

2.5 Pecuária Leiteira

Na pecuária leiteira, a qualidade do resultado obtido pelos produtores está diretamente relacionada ao percentual de vacas que tiveram sucesso no processo de lactação. Para aumentar as chances de sucesso, é recomendado realizar a composição adequada do rebanho, o que auxiliará o zootécnico/veterinário no gerenciamento deste processo e, consequentemente, terá um impacto positivo na economia da atividade leiteira. Além disso, a mortalidade dos animais pode aumentar devido ao manejo incorreto. Sistemas voltados para a gestão desses animais auxiliam no controle e sucesso de sua produção (CAMPOS et al., 2001).

Em um sistema de gerenciamento de gado leiteiro, é possível a divisão dos principais segmentos que compõem a estrutura do negócio, sendo uma parte o planejamento das atividades que irão ser realizadas na produção, e a outra parte a gerência destas atividades e os seus meios (TUNG, 1990). Este sistema demanda uma série de atividades, como o controle de sêmens, da produção, armazenamento, fornecedores, reprodução e exibição clara dos dados.

A sustentabilidade também é um ponto importante relacionado à pecuária que demanda uma maior preocupação a longo prazo, pois trata-se de um conceito amplo que envolve questões ambientais, econômicas e sociais. Cuidados com o desmatamento, emissão de carbono, controle exato de áreas e pastagens, direitos dos trabalhadores rurais e trabalho infantil são algumas das principais preocupações relacionadas à sustentabilidade (CAIO, GABRIEL, 2021). É esperado que um sistema voltado à pecuária seja capaz de apresentar possíveis soluções para um ou mais

dos problemas apresentados.

O intuito da implementação de um *software* relacionado à pecuária leiteira é auxiliar no equilíbrio de algumas desvantagens econômicas e, por esse motivo, a tecnologia da informação tem aumentado seu valor entre os produtores. Isso ocorre porque a tecnologia facilita mudanças no campo administrativo, fornecendo controle de custos, padronização de processos e estabelecendo fluxos de produção para garantir a melhora do produto final (FIGUEIRA et al., 2004).

Existem alguns obstáculos relacionados à adoção da TI (Tecnologia da Informação) na pecuária que fazem com que a implantação da mesma seja feita com maior atenção. A implantação, o uso e a manutenção da tecnologia utilizada pode gerar diferentes reações nos indivíduos diante da TI, como fascínio, perplexidade, deslumbramento ou descrença (MACHADO; NANTES, 2011), fazendo com que alguns aceitem facilmente a tecnologia, enquanto outros apresentam maior resistência ou não a utilizam. Uma possível ideia para solucionar este problema é apresentar uma base teórica bem estruturada que relata as melhorias obtidas ao utilizar um sistema para auxiliar a produção.

Ao analisar algumas soluções existentes nessa área, foram identificadas algumas deficiências em relação às funcionalidades oferecidas. Durante a utilização do sistema SUPER-PEC **(https://site.superpec.com.br/)**, uma aplicação complexa para gerenciamento de bovinos, dividida em dois módulos (gado leiteiro e gado de corte), constatou-se que, apesar de possuir diversas funcionalidades para atender aos produtores, a interface utilizada no projeto OnVet apresenta uma experiência mais agradável.

Além disso, o sistema SUPER-PEC não permite a inserção de imagens dos animais ou do usuário em questão. Durante a implementação dos protocolos de reprodução de animais, foram realizadas pesquisas e consultas a veterinários e produtores, a fim de analisar a possibilidade de informações adicionais. Esses são os pontos em que o sistema OnVet busca melhorar, visando garantir uma melhor usabilidade para o usuário.

CAPÍTULO 3

METODOLOGIA DE DESENVOLVIMENTO

Para a construção do sistema, adotou-se a metodologia de desenvolvimento em cascata, que se baseia na divisão das tarefas em diferentes etapas predefinidas que são executadas sequencialmente. Nessa metodologia, é necessário concluir a etapa anterior para avançar à próxima. As etapas definidas foram: levantamento de requisitos, projeto do sistema, implementação, realização de testes, implantação e manutenção do sistema. As subseções a seguir apresentam a análise e levantamento de requisitos e o diagrama de Casos de Uso.

3.1 Análise de Requisitos

Inicialmente, foram realizadas reuniões com veterinários que trouxeram diferentes problemas em sua área que poderiam ser resolvidos por meio de sistemas. Nestas entrevistas, o objetivo foi reunir informações e materiais para que fosse possível contextualizar o problema específico que o sistema abordaria e também citar alguns sistemas semelhantes já construídos. Para registrar os pontos importantes dessas reuniões, foi utilizado a aplicação Notion (**https://www.notion.so/**), que fornece um conjunto de ferramentas para auxiliar nas anotações. Ao estudar esse material, foi possível observar algumas falhas no controle do rebanho leiteiro, como o controle de tanques de leite, pastagens e áreas, armazenamento de informações específicas sobre cada animal e o processo de inseminação dos mesmos.

Após a contextualização do problema relacionado à dificuldade vivenciada no controle do rebanho leiteiro, o próximo passo realizado foi o levantamento dos requisitos não funcionais, juntamente com a elaboração do diagrama de casos de uso e também o modelo relacional para auxiliar no início da construção do sistema e sua base de dados.

Durante a especificação e análise de requisitos, é possível dividi-los em requisitos funcionais e não funcionais. Os requisitos materializam a necessidade que o *software* deve atender, ou seja, define quais serão as funcionalidades integradas e suas prioridades, de forma clara e objetiva. Já os requisitos não funcionais definem de que forma o sistema será executado, relatando todas as necessidades que não podem ser atendidas através das funcionalidades integradas, como a arquitetura de *software* utilizada, plataforma de *hardware*, segurança do sistema e manutenibilidade. As próximas seções apresentarão detalhadamente os requisitos funcionais e não funcionais na construção deste sistema.

3.2 Requisitos Funcionais

Os requisitos representam as condições necessárias para o desenvolvimento de um sistema capaz de solucionar problemas reais, estabelecendo uma concordância entre os clientes e os desenvolvedores. Eles são a base para estimativas, modelagens, projetos, execução e testes (CANGUÇU, RAPHAEL, 2021), sendo cruciais para garantir o ciclo de vida de um *software*.

A tabela 3.1 apresenta os requisitos funcionais, detalhando suas funções, categorias, prioridades e o caso de uso elaborado a partir deste requisito. A prioridade está dividida em Alta, Média ou Baixa, sendo esses valores definidos a partir do grau de importância da funcionalidade relacionada ao requisito em questão, definida pela equipe durante o levantamento de requisitos.

Tabela 3.1: Requisitos Funcionais

Ref.	Função	Prioridade	Casos de Uso
RFUN1	Definir nível de acesso: O sistema deve permitir a definição de diferentes níveis de acesso para os usuários, garantindo que cada usuário tenha permissões adequadas de acordo com sua função e responsabilidades.	Alta	UC01
RFUN2	Criar usuário: O sistema deve permitir a criação de novos usuários, com informações como nome, e-mail e senha, para que possam acessar e utilizar o sistema.	Alta	UC02
RFUN3	Registrar funcionários: O sistema deve permitir o registro de informações dos funcionários, como nome, cargo, contato e outras informações relevantes para o gerenciamento da equipe de trabalho.	Média	UC03
RFUN4	Realizar login: O sistema deve fornecer uma funcionalidade de login, onde os usuários possam inserir suas credenciais (como e-mail e senha) para acessar o sistema e suas respectivas funcionalidades.	Alta	UC04

RFUN5	Consultar dados: O sistema deve permitir a consulta de informações relacionadas à produção de leite, animais, funcionários, culturas, pastagens, tanques de armazenamento, protocolos de reprodução e outros dados relevantes para a pecuária leiteira.	Alta	UC05
RFUN6	Definir áreas: O sistema deve oferecer a possibilidade de definir e gerenciar diferentes áreas da propriedade, como pastos, galpões, salas de ordenha e outras áreas utilizadas na produção de leite.	Médio	UC06
RFUN7	Definir culturas: O sistema deve permitir a definição e gerenciamento das culturas utilizadas na alimentação dos animais, como capim, milho, soja, entre outros.	Média	UC07
RFUN8	Definir pastagens: O sistema deve permitir a definição e gerenciamento das pastagens utilizadas para o pastoreio dos animais, como área total, tipo de capim, manejo e outras informações relevantes.	Média	UC08
RFUN9	Adicionar tanques: O sistema deve possibilitar o registro e gerenciamento dos tanques de armazenamento de leite, incluindo informações como capacidade, localização e outras características relevantes.	Média	UC09
RFUN10	Registrar animais: O sistema deve permitir o registro e gerenciamento dos animais presentes na propriedade, incluindo informações como identificação, raça, idade, histórico de saúde, produção de leite e outras características.	Média	UC10
RFUN11	Definir protocolos: O sistema deve permitir a definição e gerenciamento de protocolos de reprodução, alimentação, vacinação e outros procedimentos relacionados aos cuidados e manejo dos animais.	Média	UC11
RFUN12	Gerenciar embriões: O sistema deve oferecer funcionalidades para o gerenciamento de embriões utilizados em programas de melhoramento genético, incluindo o registro, rastreamento e transferência de embriões entre animais.	Média	UC12
RFUN13	Controle de sêmen: O sistema deve permitir o controle e registro de informações relacionadas ao sêmen utilizado na inseminação artificial, incluindo informações sobre os touros, amostras de sêmen, data de coleta e outras informações relevantes.	Média	UC13
RFUN14	Realizar logout: O sistema deve fornecer uma funcionalidade de logout, permitindo que os usuários encerrem suas sessões e saiam do sistema de forma segura.	Alta	UC14

RFUN15	Adicionar lote de animais: O sistema deve permitir o registro e gerenciamento de lotes de animais, agrupando-os de acordo com critérios específicos, como idade, raça, produção de leite, entre outros. Isso facilitará o monitoramento e a gestão dos animais de forma mais eficiente.	Média	UC15

3.3 Requisitos Não Funcionais

Nesta seção serão apresentadas as tabelas que relatam os requisitos não funcionais do sistema, divididas em subseções.

3.3.1 Requisitos de Arquitetura de *Software*

A tabela 3.2 apresenta os Requisitos de Arquitetura de *Software*, explicados na seção 2.3.

Tabela 3.2: Requisitos de Arquitetura de *Software*

Ref.	Descrição	Casos de Uso
RARQ1	Será utilizada a arquitetura de *software* MVC, visando baixo acoplamento e alta coesão.	Nenhum

3.3.2 Requisitos de Plataforma de *Hardware*

A tabela 3.3 apresenta os requisitos não funcionais de *hardware* mínimos estipulados para que o servidor seja capaz de hospedar a aplicação.

Tabela 3.3: Requisitos de Plataforma de *Hardware*

Ref.	Descrição	Casos de Uso
RPWH1	O *software* deverá ser capaz de ser hospedado em um servidor com processadores mais recentes.	Todos

3.3.3 Requisitos de Desempenho

A tabela 3.4 apresenta os requisitos não-funcionais relacionados ao desempenho esperado do sistema.

Tabela 3.4: Requisitos de Desempenho

Ref.	Descrição	Casos de Uso

RDES1	O ambiente onde o *software* será mantido deve permitir usuários acessando o banco de dados sem queda de velocidade.	Todos
RDES2	O tempo de resposta máximo permitido responder uma requisição é de 10 segundos, evitando longas esperas no carregamento do sistema.	Todos

3.3.4 Requisitos de Disponibilidade

A tabela 3.5 apresenta os requisitos não-funcionais relacionados à disponibilidade do sistema para todos os usuários.

Tabela 3.5: Requisitos de Disponibilidade

Ref.	**Descrição**	**Casos de Uso**
RDIS1	O *software* deverá estar disponível 24 horas por dia.	Todos
RDIS2	O tempo para alteração de senha quando solicitada se limita a 60 minutos por *token* de recuperação. Este prazo é definido pelos desenvolvedores durante a codificação do sistema.	UC03

3.3.5 Requisitos de Segurança

A tabela 3.6 apresenta os requisitos não funcionais relacionados à segurança do sistema. Todos são indispensáveis para que seja construído um ambiente seguro.

Tabela 3.6: Requisitos de Segurança

Ref.	**Descrição**	**Casos de Uso**
RSEG1	O *software* deverá solicitar senha de no mínimo 8 caracteres	UC01
RSEG2	Todas as senhas cadastradas no banco de dados deverão ser criptografadas.	UC01
RSEG3	Todo usuário do *software* deverá ser associado a um perfil que define as funcionalidades que poderão ser utilizadas por ele.	Todos
RSEG4	O sistema deverá se comunicar com o banco de dados rapidamente.	Todos
RSEG5	O sistema necessitará de conexão com a internet.	Todos
RSEG4	O sistema deverá verificar as credenciais para login.	Todos
RSEG4	O sistema deverá garantir a confidencialidade dos dados dos usuários.	Todos
RSEG4	O sistema deverá garantir ao usuário que a recuperação de senha seja feita de forma segura.	Todos

3.3.6 Requisitos de Manutenibilidade

A tabela 3.7 apresenta os requisitos não funcionais responsáveis por garantir a manutenibilidade do sistema.

Tabela 3.7: Requisitos de Manutenibilidade

Ref.	Descrição	Casos de Uso
RMAN1	Todo sistema deve estar documentado de acordo com as orientações contidas na Norma de Documentação do Instituto Federal Goiano.	Todos

3.4 Casos de Uso

A utilização da UML (*Unified Modeling Language*) é importante para organização e documentação do projeto desenvolvido. Ela fornece padrões para a preparação do sistema, esquemas de bancos de dados e componentes de *software* reutilizáveis (BOOCH, 2006). Nesta seção devem ser identificados os casos de uso do *software* que será desenvolvido para que seja resumido alguns detalhes dos usuários do sistema e suas interações com o mesmo, os atores envolvidos e os requisitos funcionais e não funcionais tratados.

3.4.1 Atores

Para a elaboração do diagrama de casos de uso apresentado na Figura 3.1, foi imaginado uma possibilidade de uso do sistema, envolvendo três atores diferentes que serão apresentados à seguir. Estes atores são utilizados para representar os tipos de usuários que utilizarão o sistema e suas responsabilidades.

1. **Administrador**: Responsável por definir e gerenciar o nível de acesso de todos os outros usuários que forem criados ele.

2. **Financeiro**: Responsável por inserir e consultar as informações relacionadas à parte financeira da fazenda.

3. **Zootécnico**: Responsável por inserir e consultar as informações relacionadas aos animais.

3.5 Diagrama de Caso de Uso

A Figura 3.1 apresenta o Diagrama de Casos de Uso elaborado durante a fase inicial da construção do sistema.

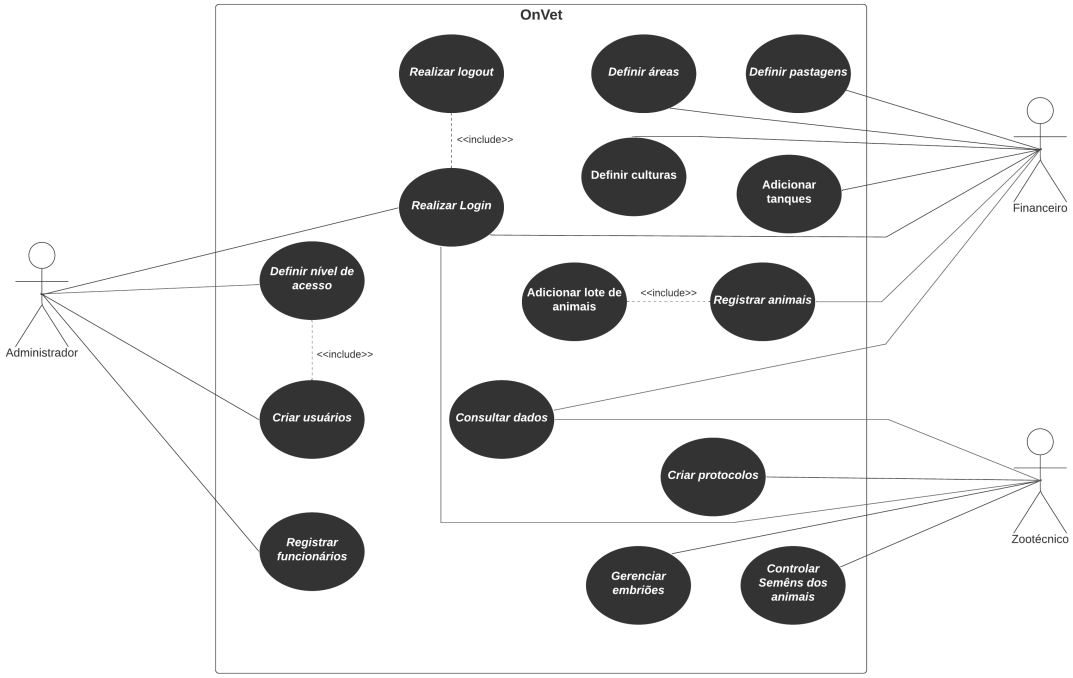

Figura 3.1: Casos de Uso do sistema OnVet.

3.5.1 Descrição de Casos de Uso

A tabela 3.8 apresenta uma descrição detalhada do Casos de Uso apresentado na figura 3.1. O item **Ref.** apresenta o código utilizado como referência ao caso de uso, o item **Nome** apresenta o nome atribuído e o item **Descrição** explica do que se trata este caso de uso em específico.

Tabela 3.8: Casos de Uso

Ref.	Nome	Descrição
UC01	Definir nível de acesso	O administrador irá definir os níveis de acesso dos usuários, ou seja, quais telas do sistema cada tipo de usuário pode acessar de acordo com suas funções na fazenda.
UC02	Criar usuário	O administrador irá criar novos usuários e então definir à qual nível de acesso este usuário pertence.
UC03	Registrar funcionários	O administrador irá definir quais os são os funcionários que compõem toda a equipe que trabalha na fazenda em questão.

UC04	Realizar login	O usuário irá logar em sua conta e o sistema irá manter a conexão/autenticação e controlar a sessão.
UC05	Consultar dados	O usuário irá consultar os dados fornecidos pelo sistema de acordo com seu nível de acesso, e o sistema fornecerá estes dados através de tabelas, gráficos ou arquivos PDF e XLS.
UC06	Definir áreas	O usuário financeiro irá definir quais são áreas da fazenda, inserindo no sistema seu tipo, data de início, fim, área em hectares e vida útil.
UC07	Definir culturas	O usuário financeiro irá definir as áreas de culturas, inserindo no sistema seu tipo, início, fim, área em hectares e custo de formação por hectar.
UC08	Definir pastagens	O usuário financeiro irá definir as áreas de pastagens, inserindo no sistema seu tipo, início, fim, área em hectares.
UC09	Adicionar tanques	O usuário financeiro irá adicionar os tanques, inserindo no sistema a sua identificação e capacidade em litros.
UC10	Registrar animais	O usuário financeiro irá registrar os animais, inserindo no sistema a foto do animal, seu nome, brinco, sexo, grau de sangue, raça, lote que está incluso, origem no rebanho, nascimento, peso de nascimento, nome de registro, número de registro, pelagem. Se o animal for comprado, deve inserir qual foi o fornecedor, a data de entrada, o peso de entrada e também sua categoria e valor pago.
UC11	Definir protocolos	O usuário zootécnico irá definir os protocolos de inseminação, inserindo no sistema o seu tipo, nome, descrição e mãe/pai utilizados no processo.
UC12	Gerenciar embriões	O usuário zootécnico irá adicionar e gerenciar os embriões gerados nos animais, inserindo no sistema o nome, raça, mãe/pai e o tipo de embrião.
UC13	Controlar sêmens dos animais	O usuário zootécnico irá adicionar e controlar os sêmens recolhidos dos animais, inserindo no sistema o nome, raça, mãe/pai e o tipo de sêmen.
UC14	Realizar logout	O usuário poderá realiar logout de sua conta, e o sistema deve relatar o registro de logs.
UC15	Adicionar lote de animais	O usuário financeiro irá definir quais são os lotes de animais, inserindo no sistema o nome do lote, sua descrição, abreviação, sexo e fase atual.

3.6 Ferramentas e Tecnologias Utilizadas

Nesta seção, serão apresentadas as principais tecnologias e ferramentas utilizadas no desenvolvimento da aplicação em questão. Essas ferramentas desempenharam um papel fundamental em diferentes aspectos do projeto, desde o planejamento e versionamento até a implementação do front-end, back-end, banco de dados e hospedagem.

3.6.1 Planejamento e Versionamento

No processo de planejamento e versionamento do projeto, foram utilizadas as seguintes tecnologias e ferramentas:

- **Notion**: Uma plataforma colaborativa para gerenciamento de projetos, notas e documentação. O Notion foi utilizado para organizar tarefas, definir cronogramas e documentar o progresso do projeto.
- **LucidChart**: Uma ferramenta de criação de diagramas que auxiliou na modelagem e visualização da arquitetura do sistema. Foram criados diagramas de fluxo, diagramas de entidade-relacionamento e outros tipos de diagramas relevantes para o projeto.
- **brModelo**: Uma ferramenta para modelagem de bancos de dados que possibilitou a criação e visualização dos modelos conceituais, lógicos e físicos do banco de dados do sistema.
- **Figma**: Uma plataforma de design de interface que permitiu a criação de protótipos interativos e a definição da aparência visual da aplicação. O Figma facilitou a comunicação entre designers e desenvolvedores durante o processo de criação da interface do usuário.
- **Git e GitHub**: O Git foi utilizado como sistema de controle de versão para registrar todas as alterações feitas no código-fonte da aplicação. O GitHub, por sua vez, foi utilizado como repositório remoto para hospedar o projeto e permitir a colaboração entre os membros da equipe.

3.6.2 Ferramentas e Tecnologias: Editor de código-fonte e testes

No desenvolvimento da aplicação, foram utilizadas as seguintes ferramentas e tecnologias relacionadas ao editor de código-fonte e realização de testes:

- **Visual Studio Code**: Um editor de código-fonte leve e poderoso, que proporcionou uma experiência de desenvolvimento eficiente e produtiva. O Visual Studio Code oferece recursos avançados, como realce de sintaxe, depuração integrada e extensibilidade.
- **Insomnia**: Uma plataforma para testes de APIs que facilitou a realização de requisições HTTP e a validação das respostas. O Insomnia permitiu testar e depurar as funcionalidades da aplicação de forma simplificada e eficaz.

3.6.3 Ferramentas e Tecnologias: Front-end e Back-end

No desenvolvimento do front-end e back-end da aplicação, foram utilizadas as seguintes tecnologias e ferramentas:

Front-end:

- **HTML (Hyper Text Markup Language)**: A linguagem de marcação padrão para a criação de páginas web. O HTML foi utilizado para estruturar e organizar o conteúdo da aplicação.
- **CSS (Cascading Style Sheets)**: A linguagem de estilização que define a aparência visual dos elementos HTML. O CSS foi utilizado para aplicar estilos e personalizar o layout da aplicação.
- **JavaScript**: A linguagem de programação utilizada para adicionar interatividade e dinamismo às páginas web. O JavaScript foi utilizado para implementar funcionalidades interativas e realizar requisições assíncronas ao servidor.
- **Vuexy**: Um framework de front-end baseado em Vue.js que oferece um conjunto de componentes e recursos pré-construídos para agilizar o desenvolvimento. O Vuexy foi utilizado para criar a interface de usuário responsiva e moderna.

Back-end:

- **PHP**: Uma linguagem de programação amplamente utilizada para o desenvolvimento de aplicações web. O PHP foi escolhido como a linguagem principal para implementação da lógica de negócio e integração com o banco de dados.

3.6.4 Ferramentas e Tecnologias: Banco de Dados

No que se refere ao banco de dados da aplicação, foram utilizadas as seguintes tecnologias e ferramentas:

- **MySQL**: Um sistema de gerenciamento de banco de dados relacional amplamente utilizado. O MySQL foi escolhido como o banco de dados principal para armazenar e gerenciar os dados da aplicação.
- **WAMP Server**: Um ambiente de desenvolvimento que combina o servidor web Apache, o banco de dados MySQL e a linguagem de programação PHP. O WAMP Server proporcionou um ambiente local para o desenvolvimento e teste da aplicação.

- **phpMyAdmin**: Uma ferramenta web para administração do banco de dados MySQL. O phpMyAdmin permitiu a criação e manipulação das tabelas e registros do banco de dados de forma visual e intuitiva.

3.6.5 Ferramentas e Tecnologias: Frameworks

No desenvolvimento da aplicação, foram utilizados os seguintes frameworks:

- **Laravel**: Um framework de desenvolvimento web em PHP que oferece uma estrutura robusta e elegante para a construção de aplicações web. O Laravel foi escolhido como o framework principal para agilizar o desenvolvimento, fornecendo recursos como roteamento, autenticação, manipulação de banco de dados e muito mais.

- **Bootstrap**: Um framework de front-end que oferece componentes e estilos pré-construídos para agilizar o desenvolvimento da interface do usuário. O Bootstrap foi utilizado para criar uma interface responsiva e esteticamente agradável.

3.6.6 Ferramentas e Tecnologias: Hospedagem

Para hospedar a aplicação, foi utilizada a plataforma de hospedagem:

- **Hostinger**: Uma plataforma de hospedagem que oferece recursos para publicação e disponibilização da aplicação na internet. A Hostinger foi escolhido como provedor de hospedagem para disponibilizar a aplicação de forma acessível aos usuários.

Essas são as principais tecnologias e ferramentas utilizadas no desenvolvimento da aplicação, cada uma desempenhando um papel específico para garantir a eficiência, qualidade e funcionalidade do sistema como um todo.

CAPÍTULO 4

MODELAGEM

Neste capítulo, será apresentada a fase de modelagem do sistema **OnVet**, dividida em três seções, que apresentam, respectivamente, o Diagrama de Entidade e Relacionamento, o Modelo Relacional do projeto e o Diagrama de Pacotes.

4.1 Diagrama de Entidade e Relacionamento

Com o advento das tecnologias da informação e a necessidade cada vez maior de se armazenar e gerenciar grandes quantidades de dados, tornou-se essencial o uso de ferramentas que auxiliem na modelagem e organização desses dados. Nesse contexto, o DER (Diagrama de Entidade e Relacionamento) surge como uma ferramenta importante para representação e análise dos dados de um sistema.

4.1.1 Definição

Segundo (SILBERSCHATZ et al., 2011), um Diagrama de Entidade e Relacionamento é uma técnica de modelagem de dados que tem como objetivo representar as entidades relevantes para um determinado domínio de negócio, bem como os relacionamentos entre elas. Ele é composto por entidades, que representam os objetos relevantes do domínio, e pelos relacionamentos, que representam as associações entre essas entidades.

As entidades representam objetos do mundo real, como pessoas, lugares, coisas ou conceitos, que possuem características que precisam ser armazenadas e gerenciadas pelo sistema. Já os relacionamentos representam as conexões entre esses objetos, indicando como as entidades se relacionam entre si.

4.1.2 Objetivo

O DER é uma ferramenta muito útil na fase de análise de sistemas, pois permite ao analista entender melhor o domínio do problema a ser resolvido, modelando e organizando as informações em entidades e relacionamentos. Além disso, ele permite identificar de forma clara as regras de negócio e as restrições que devem ser aplicadas aos dados.

Segundo (HEUSER, 2009), o DER tem como principal objetivo permitir a visualização clara e objetiva dos dados de um sistema, facilitando a comunicação entre os envolvidos no processo de desenvolvimento do sistema. Dessa forma, ele é capaz de ajudar na identificação de problemas e erros no modelo de dados, antes mesmo da implementação do sistema.

4.1.3 DER e sua descrição

A figura 4.1 apresenta o Diagrama de Entidade e Relacionamento do sistema **OnVet**.

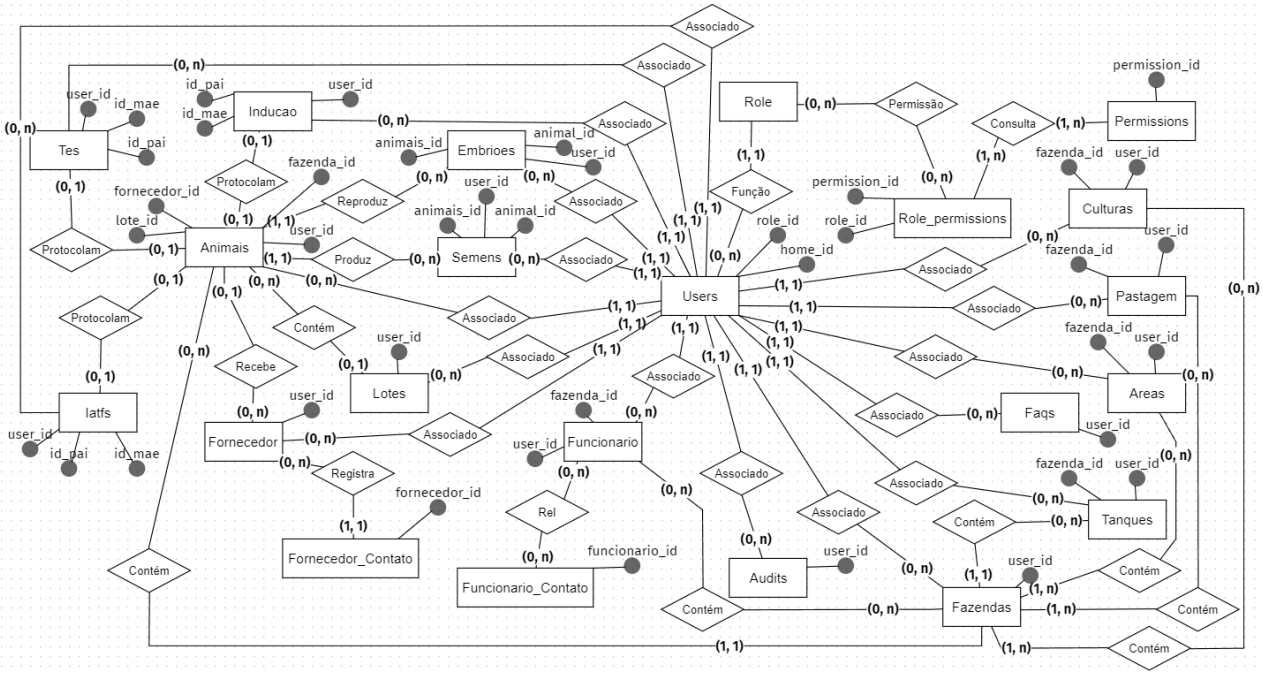

Figura 4.1: Diagrama de Entidade e Relacionamento.

Devido à complexidade do banco de dados, foram incluídos no diagrama apenas os atributos que representam as chaves estrangeiras, as tabelas, seus relacionamentos e suas cardinalidades. Pode-se observar que a chave estrangeira **user_id** está presente na maioria das tabelas, pois é por meio desse atributo que o sistema realiza o gerenciamento do nível de acesso, ou seja, com base no identificador do usuário, o sistema define quais telas ele pode acessar. A seguir, estão listados todos os atributos de cada tabela:

1. **Animais:** id, video, imagem, nome, sexo, sangue, raca, brinco, origem, dt_nasc, peso, nome_reg, num_reg, raca_2, pelagem, dt_entrada, cat_macho, cat_femea, valor, desmame, lote_id, peso_entrada, fornecedor_id, user_id.
2. **Fornecedor:** id, nome, cpf, cnpj, razao, tipo, email, telefone, cep, endereco, numero, complemento, bairro, cidade, uf, ativo, user_id.
3. **Fornecedor_Contato:** id, fornecedor_id, nome, telefone, email.
4. **Funcionario:** id, nome, cpf, dat_nasc, sexo, funcao, telefone, cep, endereco, numero, complemento, bairro, cidade, uf, ativo, user_id, fazenda_id.
5. **Funcionario_Contato:** id, funcionario_id, nome, telefone, email.
6. **Embrioes:** id, nome, tipo, animal_id, animais_id, observacao, congelamento, grau, user_id.
7. **Semens:** id, registro, nome, raca, central, tipos, animal_id, animais_id, sangue, raca_2, observacao, tec, pertida, user_id.
8. **Tes:** id, nome, desc, animal_id, animais_id, user_id.
9. **Iatfs:** id, nome, desc, animal_id, animais_id, user_id.
10. **Inducao:** id, nome, desc, dias_lactacao, animal_id, user_id, dt_prt.
11. **Users:** id, role_id, home_id, name, email, email_verified_at, password, remember_token, jobtitle, phone, cellphone, active, redefinir_senha.
12. **Fazendas:** id, nome, cep, endereco, uf, ativo, user_id.
13. **Faqs:** id, pergunta, resposta, ativo, user_id.
14. **Role:** id, name.
15. **Role_permissions:** role_id, permission_id.
16. **Permissions:** id, permission_id, menu, position, name, url, icon.
17. **Audits:** id, user_type, user_id, event, auditable_type, auditable_id, old_values, new_values, url, ip_adress, user_agent, tags.

18. **Pastagem:** id, nome, dt_ini, dt_fim, area, tipo, custo, total, ativo, observacao, user_id, fazenda_id.

19. **Culturas:** id, nome, dt_ini, dt_fim, tipo, ha, custo, total, ativo, observacao, user_id, fazenda_id.

20. **Areas:** id, nome, dt_ini, dt_fim, tipo, ha, util, ativo, observacao, user_id, fazenda_id.

21. **Tanques:** id, nome, litros, observacao, user_id, fazenda_id.

22. **Lotes:** id, nome, desc, abv, sexo, fase, ativo, observacao, user_id.

4.2 Modelo Relacional

O modelo relacional é um modelo de dados que representa os dados em forma de tabelas, também chamadas de relações, e as relações entre elas. É um modelo mais simples e intuitivo, onde cada tabela representa uma entidade e as relações entre elas são definidas através de chaves estrangeiras.

4.2.1 Definição

Segundo (SILBERSCHATZ et al., 2011), o modelo relacional é baseado em três conceitos fundamentais: atributos, tuplas e relacionamentos. Os atributos são as características ou propriedades que descrevem as entidades que estão sendo modeladas, como nome, idade, endereço, entre outros. As tuplas são as linhas ou registros em uma tabela que representam uma instância de uma entidade. Já os relacionamentos são as conexões entre as entidades, representados através de chaves estrangeiras em uma tabela que se relaciona com outra.

Uma das principais vantagens do modelo relacional é a sua simplicidade e facilidade de uso, já que é baseado em um conceito comum e intuitivo: a tabela. Além disso, ele permite a manipulação de dados de maneira estruturada e organizada, tornando mais fácil a implementação de regras de negócio e a geração de relatórios. Outra vantagem é a capacidade de garantir a integridade dos dados através de restrições de chave primária e estrangeira, evitando a duplicação ou perda de dados.

4.2.2 Objetivo

O modelo relacional é amplamente utilizado em sistemas de gerenciamento de banco de dados relacionais, como MySQL, PostgreSQL, Oracle, SQL Server, entre outros. Ele oferece uma grande variedade de operações que podem ser realizadas com os dados, como inserção, atualização, exclusão, seleção, ordenação, agrupamento, entre outras. Além disso, ele permite a criação de visões, que são representações lógicas dos dados em forma de consultas SQL (*Structured Query Language*) afim de facilitar a análise e a apresentação dos dados para os usuários.

A Figura 4.2 apresenta o Modelo Relacional, desenvolvido com base no DER, com o propósito de auxiliar na documentação do projeto e na criação do banco de dados. As principais entidades neste banco de dados são as tabelas: Usuários, Fazendas, Lotes, Fornecedores e Animais. Essas tabelas são consideradas principais, pois o fluxo de utilização do sistema, conforme descrito na subseção 5.1.2, requer o preenchimento dessas tabelas na ordem mencionada anteriormente. Como explicado na subseção 4.1.3, a tabela **user** se relaciona com a maioria das outras tabelas, inclusive com a tabela **role**, que será responsável por definir o que este usuário poderá acessar no sistema.

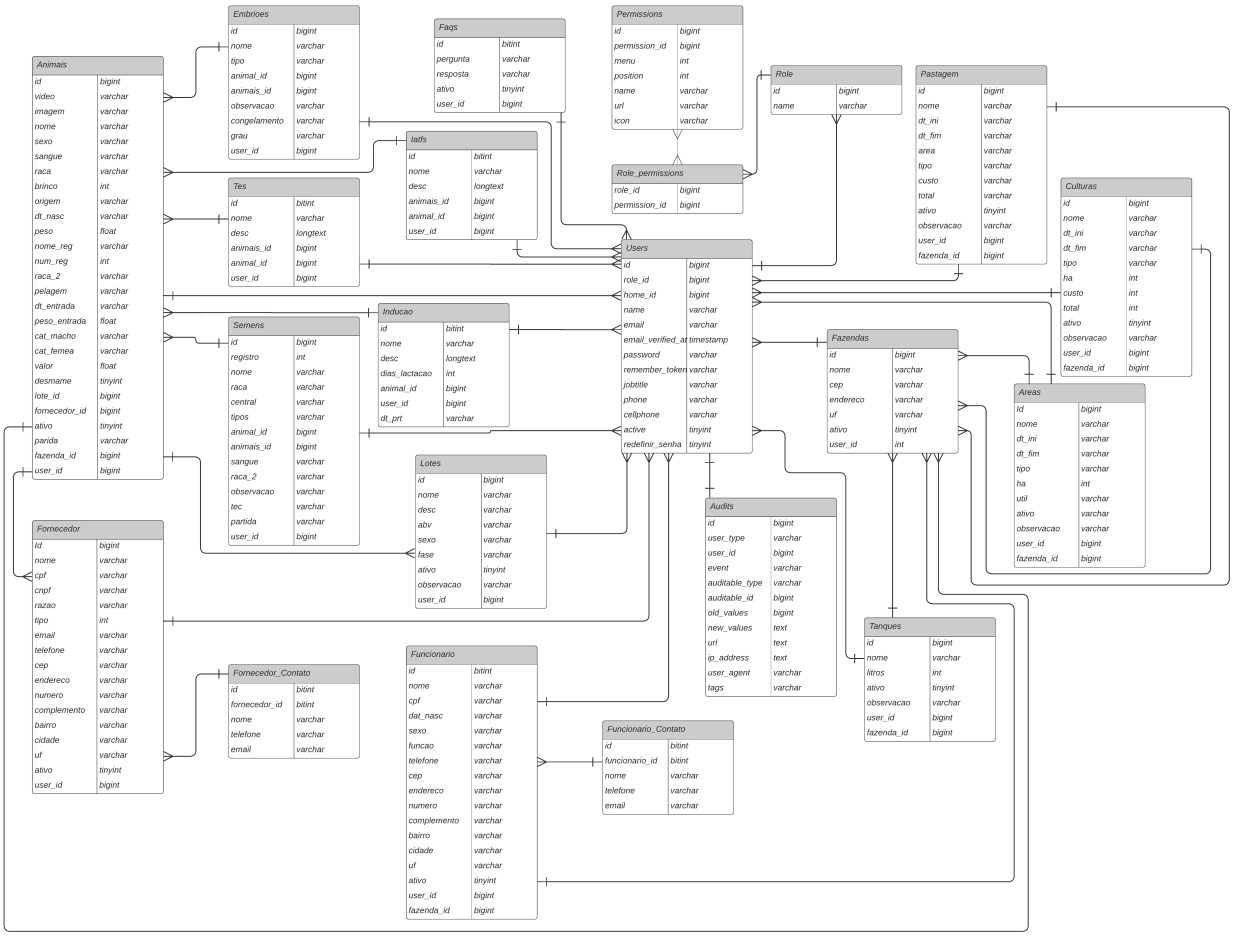

Figura 4.2: Modelo Relacional do sistema **OnVet**.

4.3 Diagrama de Pacotes

Esta seção apresentará o Diagrama de Pacotes elaborado para auxiliar no entendimento da organização dos pacotes do projeto, bem como sua definição e objetivo.

4.3.1 Definição

Este diagrama é uma ferramenta utilizada para representar a estrutura organizacional de uma aplicação, ou seja, como as diferentes partes do mesmo estão organizadas em módulos ou pacotes. É utilizado na Engenharia de *Software* para representar a estrutura organizacional de um sistema. Cada pacote é representado por um retângulo que contém o nome do pacote e seus atributos ou outros pacotes que fazem parte de sua estrutura (GOMES; GARCIA, 2007).

4.3.2 Objetivo

Este diagrama de pacotes, apresentado na Figura 4.3, é uma ferramenta muito útil na fase de modelagem pois ele permite que os desenvolvedores visualizem a estrutura organizacional de forma clara e organizada. Além disso, ele pode ser utilizado para identificar e organizar as diferentes partes do sistema em módulos ou pacotes, facilitando a manutenção e evolução do mesmo. Ele também facilita a comunicação entre os membros da equipe de desenvolvimento, permitindo que todos tenham uma visão geral da estrutura. O Diagrama de Pacotes pode ainda auxiliar na definição de interfaces entre os diferentes módulos do sistema e na documentação (SOMMERVILLE, 2003; BOOCH et al., 2000).

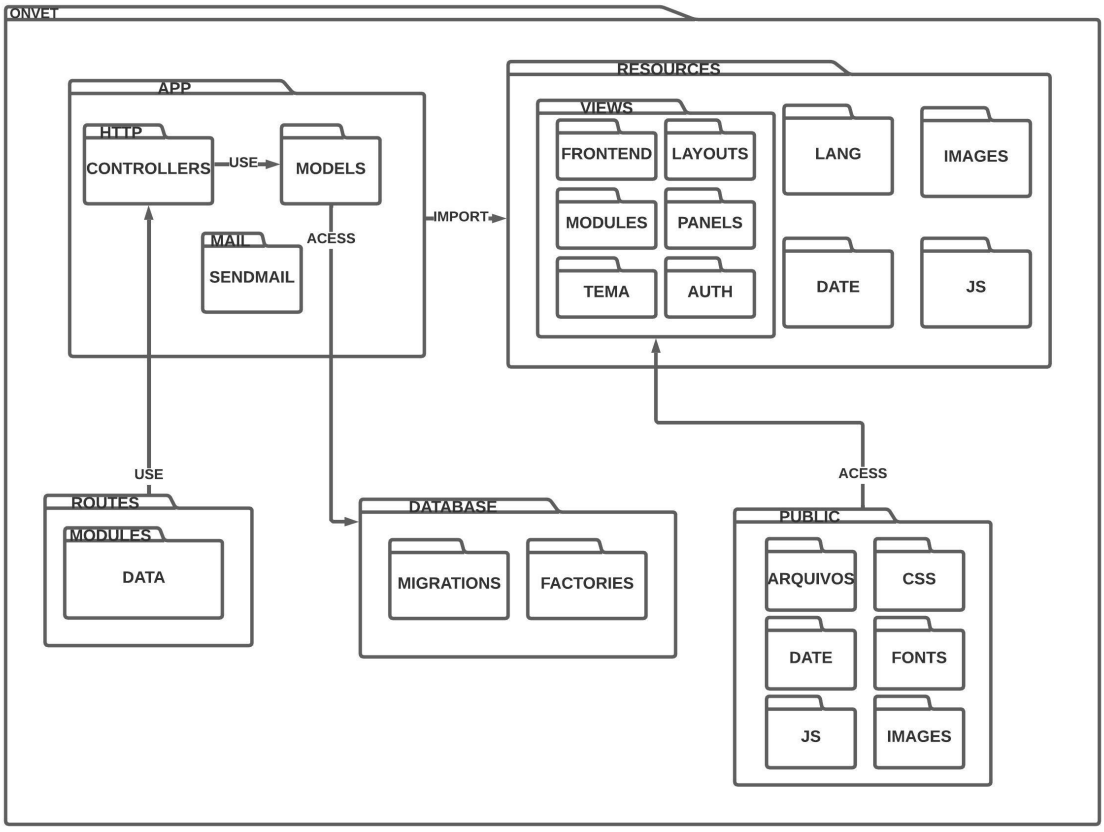

Figura 4.3: Diagrama de Pacotes do sistema **OnVet**.

O diagrama apresentado na Figura 4.3 está organizado da seguinte forma: O pacote **ONVET** contém todos os outros pacotes. O pacote **SRC** contém todo o código fonte construído da aplicação, e importa os componentes contídos no pacote **NODE_MODULES**. O pacote **APP** acessa os pacotes ***ROUTES***, **API** e ***DATABASE*** para execução das classes modelo, controladoras e utilização de API's, para então o pacote ***RESOURCES*** se comunicar com o pacote **APP**, importando suas informações e também acessando o pacote ***PUBLIC***, que contém arquivos

como códigos JavaScript e imagens que são utilizadas no sistema, e então apresenta todas as informações ao usuário.

CAPÍTULO 5

IMPLEMENTAÇÃO

Este capítulo apresenta os principais aspectos relacionados à implementação do sistema, incluindo suas funcionalidades e a forma como foi realizada. Este é um capítulo técnico, repleto de comandos e trechos de código, divididos em doze subseções. A primeira subseção apresenta uma introdução ao OnVet para que o leitor conheça melhor o sistema e entenda com mais facilidade as outras subseções. Cada subseção posterior abordará uma determinada função do sistema, também explicando os motivos de sua implementação.

Na primeira subseção, 5.1, é apresentada a introdução ao sistema em si, juntamente com o fluxo de utilização do sistema. Na subseção 5.2 é abordada a funcionalidade de envio de e-mail, enquanto na 5.3 é descrito o processo de recuperação de senha. A subseção 5.4 trata da criptografia de senha usada para proteger os dados dos usuários. Na subseção 5.5, são explicados os contadores utilizados no sistema para exibir informações sobre as tabelas, e na 5.6 é descrita a API *Google Charts* utilizada para gerar gráficos a partir dos dados cadastrados. A funcionalidade de exportação de arquivos é apresentada na 5.7, e na 5.8 é explicada a integração com a API CEP para obter dados de endereço. A subseção 5.9 descreve a funcionalidade de contato com funcionários e fornecedores. A inserção de imagens é abordada na 5.10 e as funções JavaScript utilizadas para ocultar ou exibir informações na *View* é abordada na subseção 5.11. Por fim, a funcionalidade de filtragem de filiação é descrita na subseção 5.12.

Desta forma, este capítulo fornece uma visão geral das principais funcionalidades implementadas no sistema, com informações detalhadas sobre cada uma das subseções apresentadas anteriormente.

5.1 Introdução ao Sistema OnVet

O OnVet é projetado para auxiliar produtores de leite a monitorar e gerir suas atividades relacionadas ao rebanho, proporcionando um ambiente organizado e eficiente para a gestão de informações e processos. Ele apresenta uma interface intuitiva e amigável, com menus bem definidos que facilitam a navegação e acesso às diferentes funcionalidades. As próximas subseções apresentam os menus disponíveis e o fluxo de utilização da aplicação.

5.1.1 Menus do Sistema

As telas são divididas em menus, para que seja possível melhor organização e utilização do sistema. Os menus disponíveis são:

- **Informações Gerais:** Este menu apresenta a tela de *dashboard*, apresentado na Figura 5.1, onde são exibidos dados relevantes ao usuário, como a quantidade total de dados cadastrados no banco de dados, gráficos com quantidade de animais classificados por raça no rebanho, entre outras informações essenciais para o acompanhamento do desempenho do rebanho leiteiro.

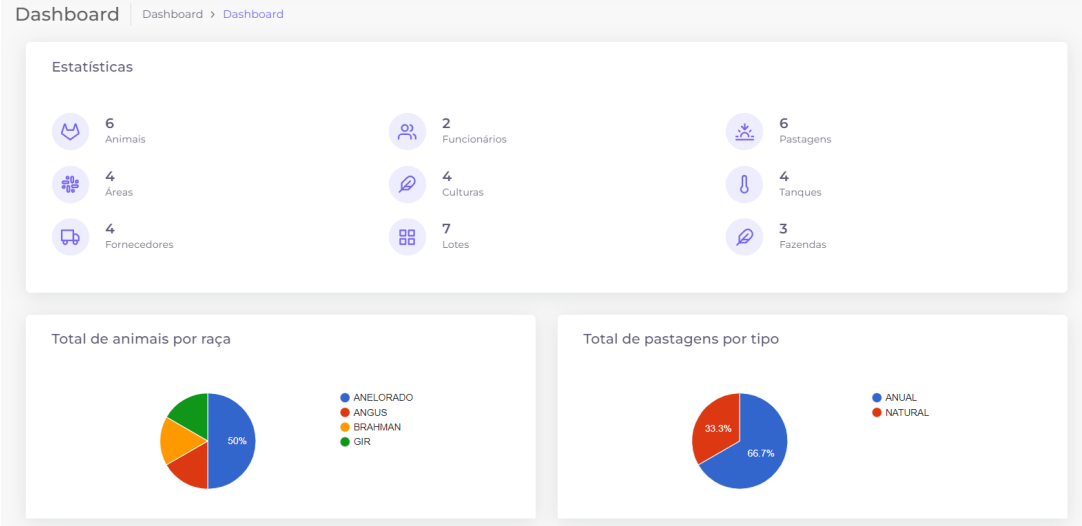

Figura 5.1: *Dashboard.*

- **Cadastros:** Neste menu, é possível realizar o cadastro de diversos itens que compõem o sistema. Entre os itens que podem ser cadastrados estão os usuários, o nível de acesso de usuários, funcionários envolvidos no manejo do rebanho, tanques de leite, áreas de pastagem, culturas agrícolas utilizadas na alimentação dos animais e fornecedores de insumos ou animais. Estes são os itens que aparecem no menu:

– Nível de acesso, Figura 5.2.

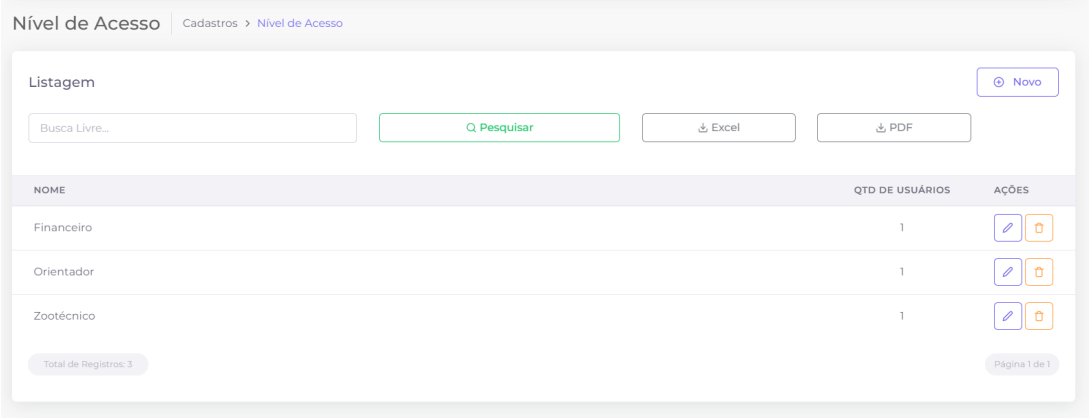

Figura 5.2: Cadastro de níveis de acesso.

– Usuários, Figura 5.3.

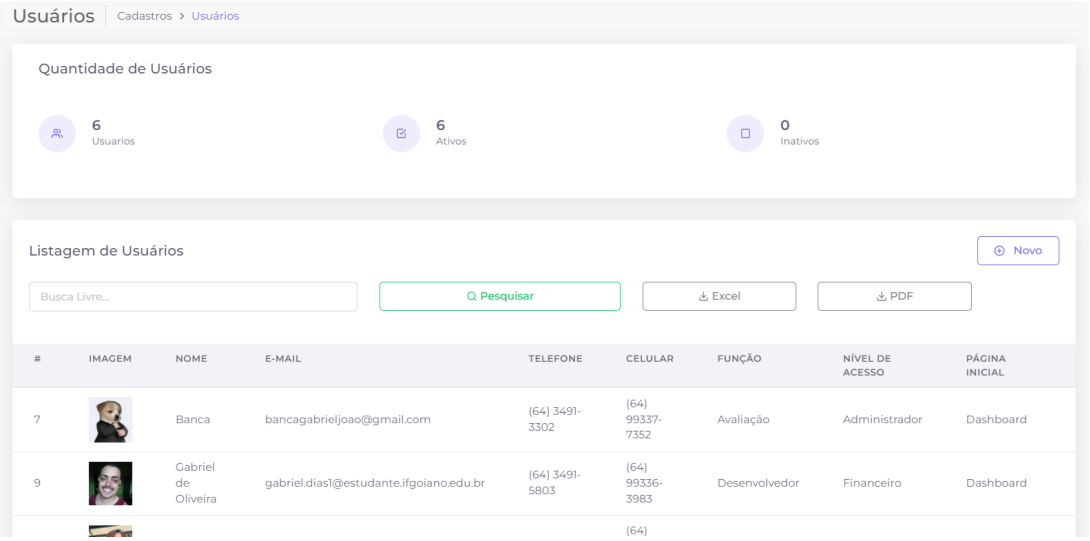

Figura 5.3: Cadastro de usuários.

– Funcionários, Figura 5.4.

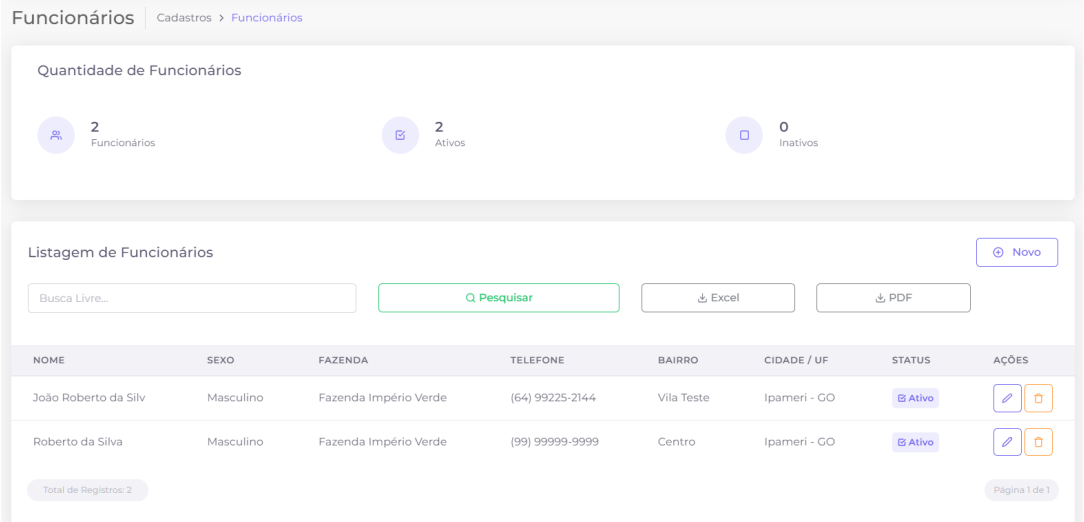

Figura 5.4: Cadastro de funcionários.

– Tanques, Figura 5.5.

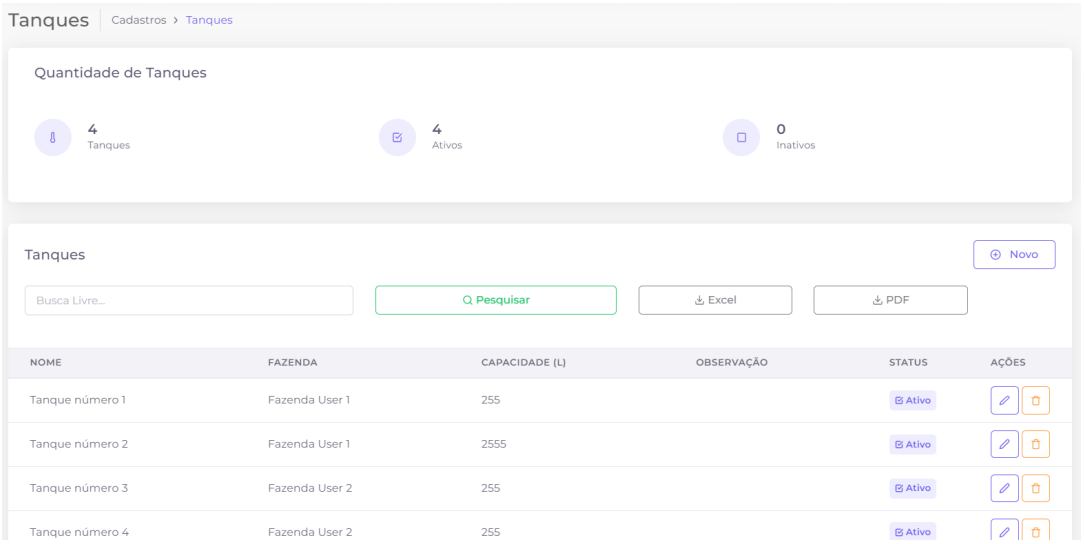

Figura 5.5: Cadastro de tanques.

– Áreas, Figura 5.6.

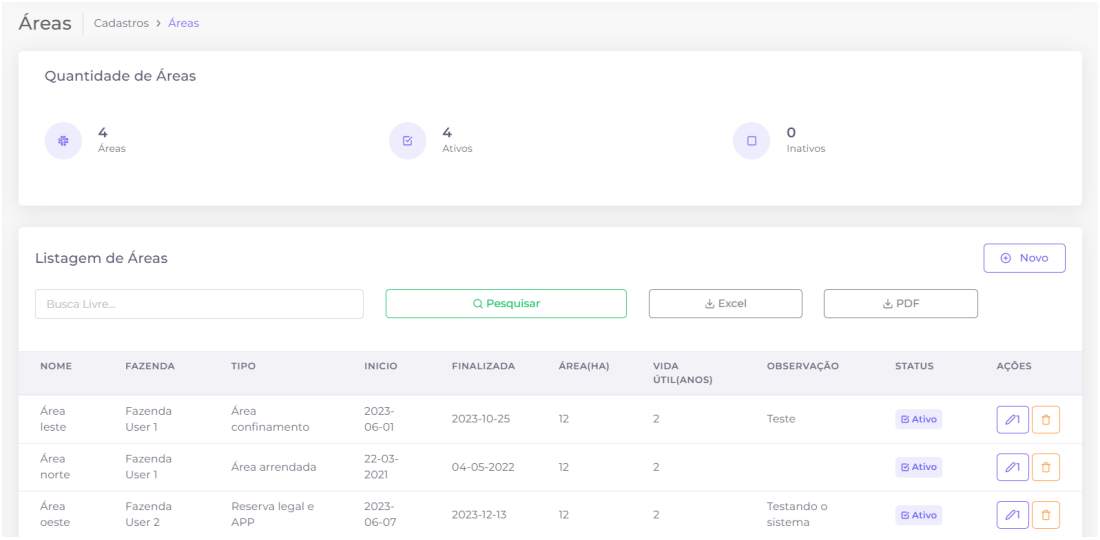

Figura 5.6: Cadastro de áreas.

– Culturas, Figura 5.7.

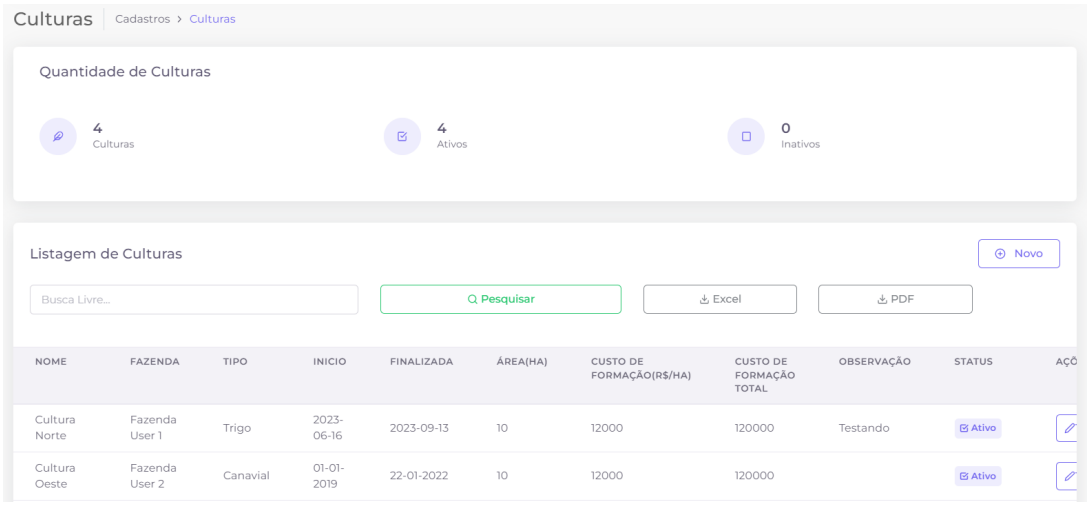

Figura 5.7: Cadastro de culturas.

– Fornecedores, Figura 5.8.

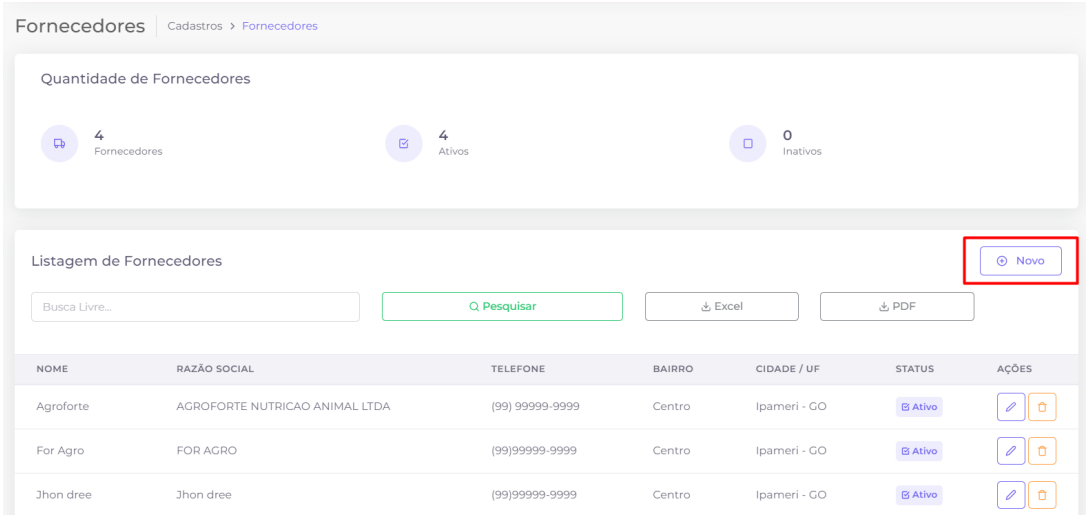

Figura 5.8: Cadastro de fornecedores.

– Pastagens, Figura 5.9.

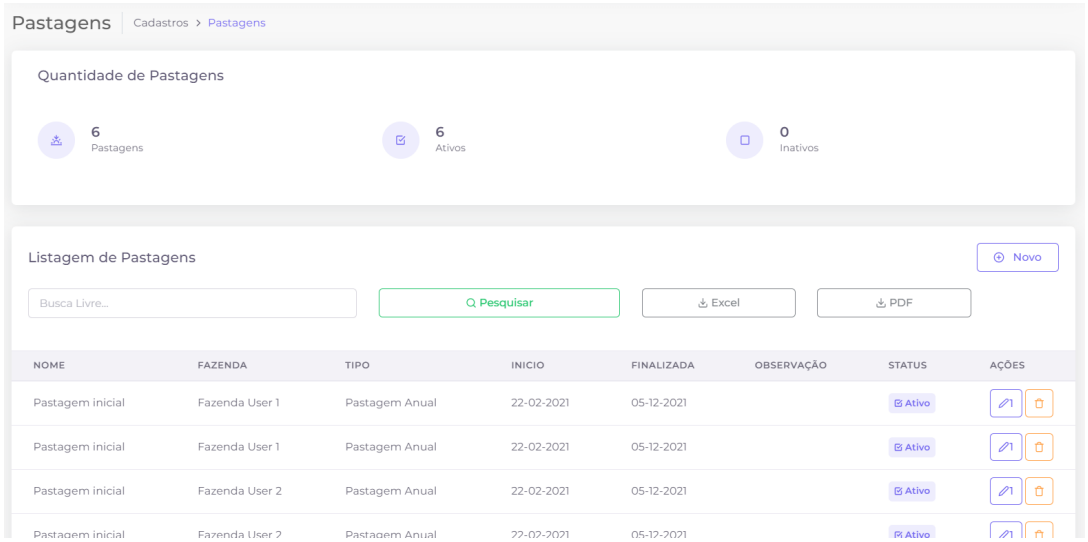

Figura 5.9: Cadastro de pastagens.

– Fazendas, Figura 5.10.

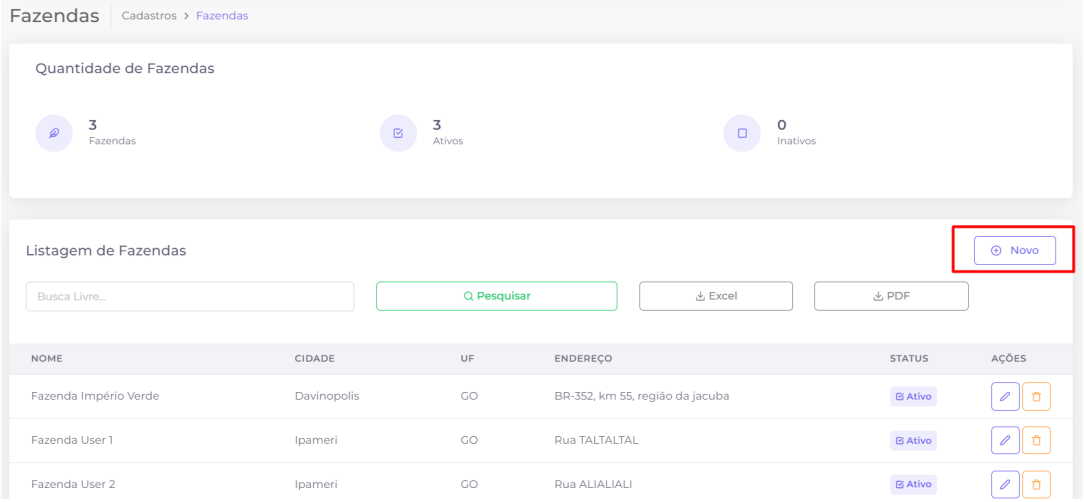

Figura 5.10: Cadastro de fazendas.

- **Protocolos:** Este menu abrange os diferentes protocolos utilizados no manejo do gado leiteiro. Inclui o protocolo de transferência de embrião, que permite a reprodução seletiva do rebanho por meio da transferência de embriões de animais de alto valor genético para fêmeas receptoras. Além disso, estão disponíveis o protocolo de indução à lactação e o protocolo de inseminação artificial em tempo fixo. Estes são os itens que aparecem no menu:

  – Protocolo TE, Figura 5.11.

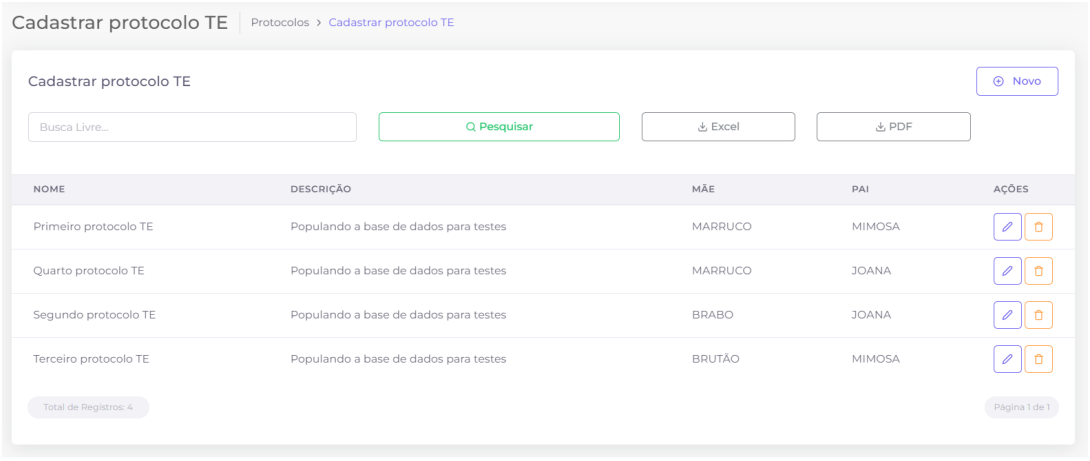

Figura 5.11: Protocolo TE.

  – Protocolo Indução, Figura 5.12.

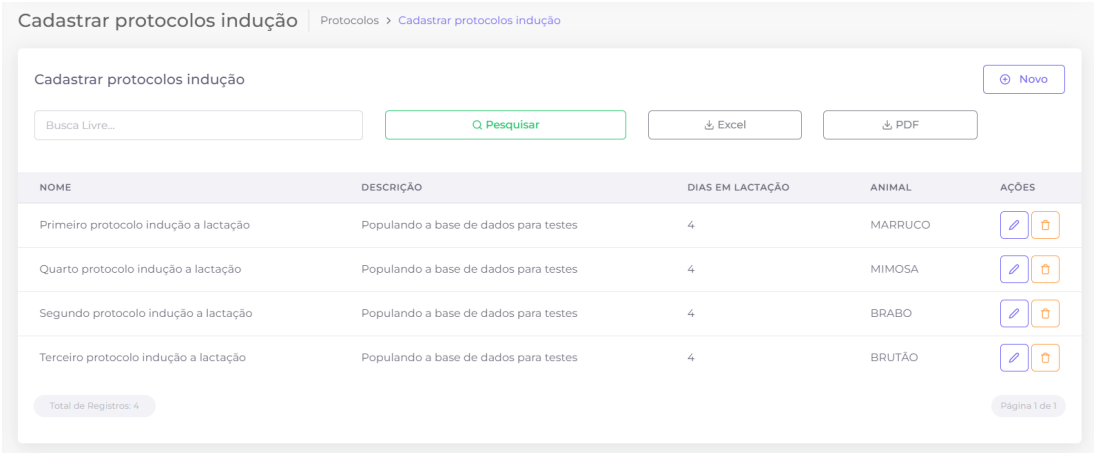

Figura 5.12: Protocolo indução.

– Protocolo IATF, Figura 5.13.

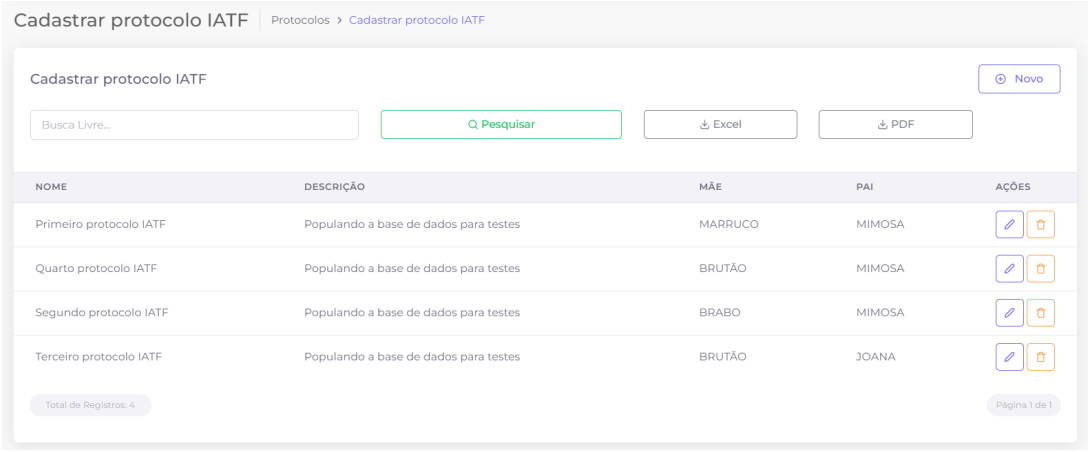

Figura 5.13: Protocolo IATF.

- **Dúvidas Gerais:** Neste menu, encontra-se a tela de perguntas frequentes (FAQs), apresentada na Figura 5.14, que fornece respostas para questões comuns e orientações sobre o uso do sistema OnVet. Essa seção é especialmente útil para sanar dúvidas rápidas e fornecer suporte aos usuários.

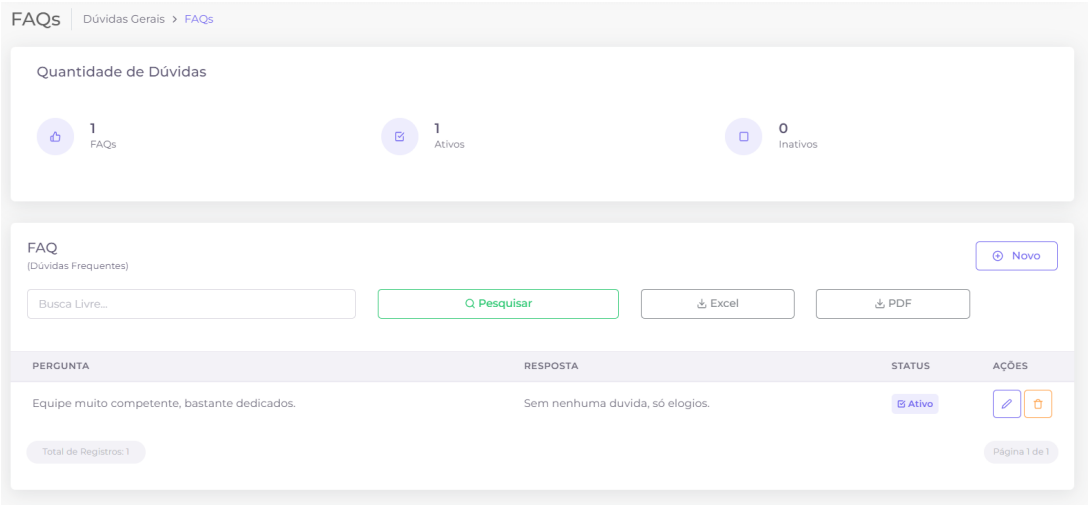

Figura 5.14: Cadastro de Faqs.

- **Rebanho:** O menu Rebanho engloba as funcionalidades relacionadas ao controle dos animais. Inclui telas de cadastro de animais, lotes de animais, informações sobre o sêmen utilizado no rebanho e detalhes dos embriões existentes. Estes são os itens que aparecem no menu:

  – Animais, Figura 5.15.

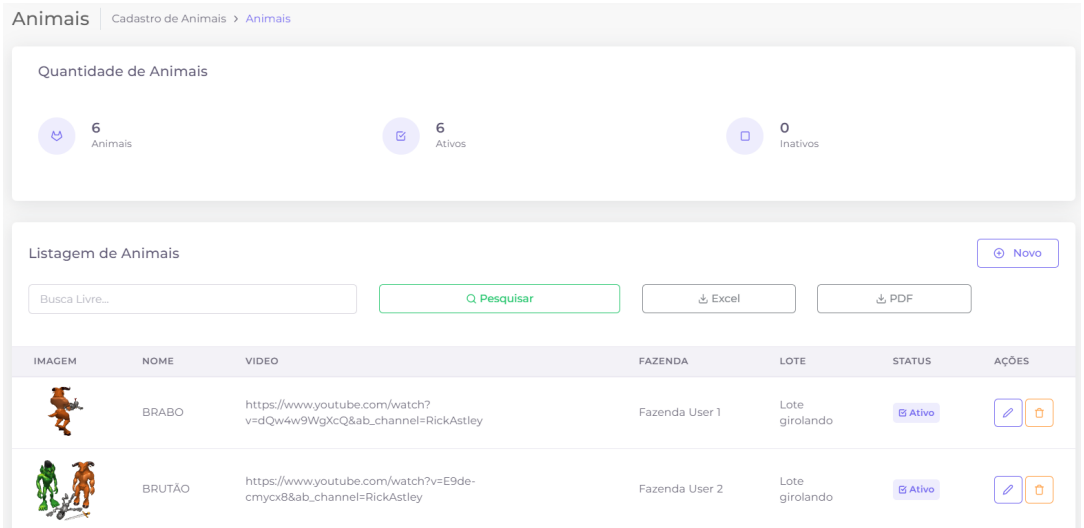

Figura 5.15: Cadastro de animais.

  – Lotes, Figura 5.16.

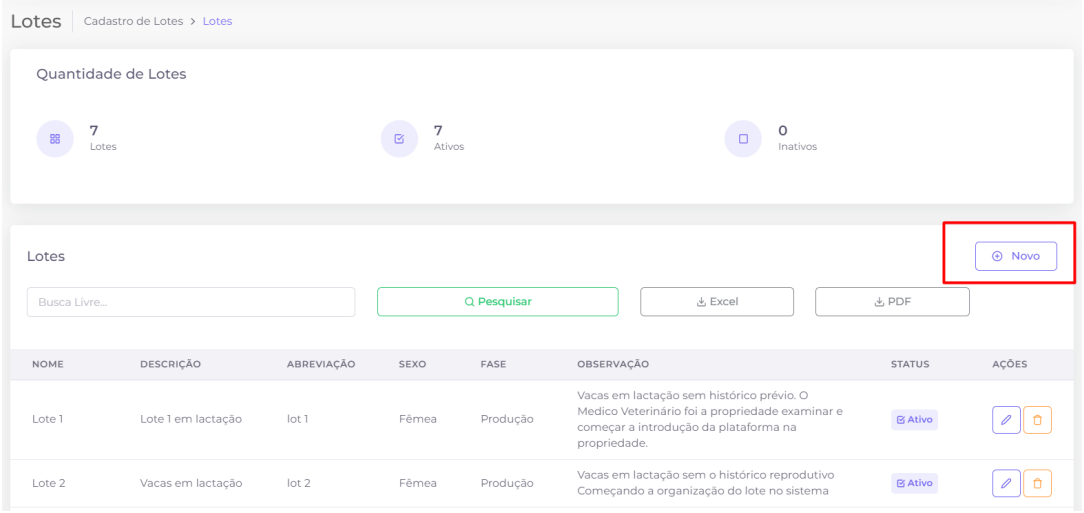

Figura 5.16: Cadastro de lotes.

– Sêmens, Figura 5.17.

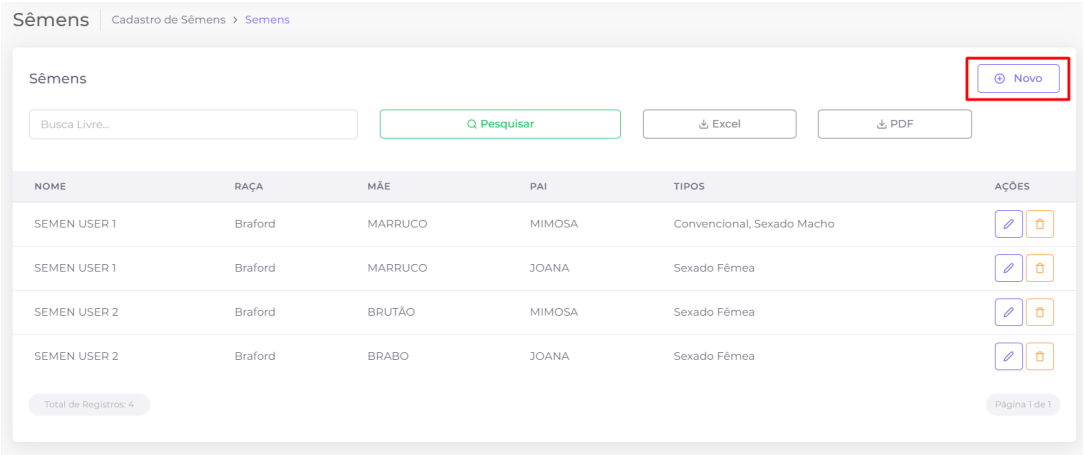

Figura 5.17: Cadastro de sêmens.

– Embriões, Figura 5.18.

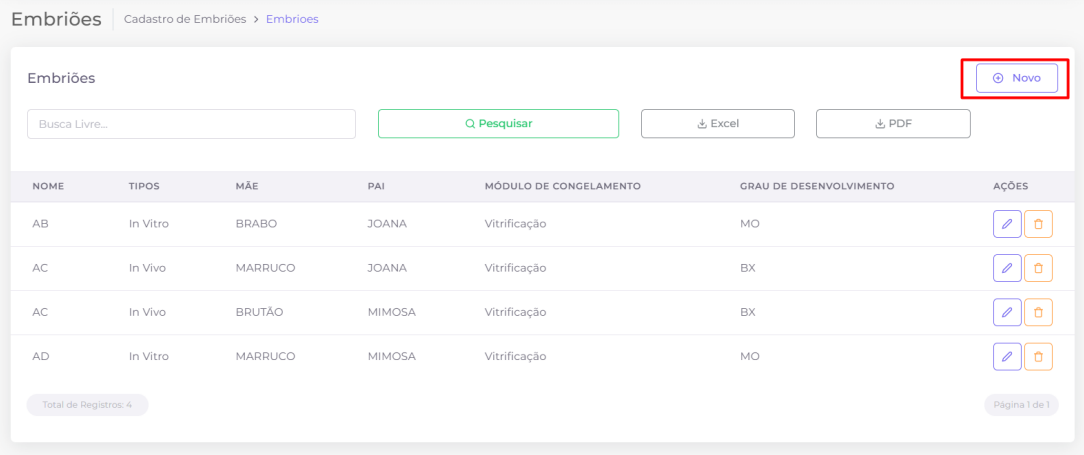

Figura 5.18: Cadastro de embriões.

5.1.2 Fluxo do Sistema

Definir o fluxo do sistema é uma etapa essencial no processo de desenvolvimento de *software*. Isso envolve mapear e descrever as etapas, interações e decisões que normalmente ocorrem durante a execução do sistema, desde o início até a conclusão de um determinado processo ou funcionalidade. A definição do fluxo do sistema permite uma compreensão clara de como os usuários interagem com o sistema, quais são as etapas necessárias para realizar determinadas tarefas e como as informações são processadas e fluem entre os diferentes componentes do sistema.

Para que usuário consiga inserir animais na base de dados, é necessário que siga o sguinte fluxo: inserir os dados sobre sua(s) fazenda(s), tela esta retrada na Figura 5.10. Seguido dos lotes, Figura 5.16, que esses animais ocuparão e seus fornecedores (no caso de compra), Figura 5.8.

Após cadastrar os animais no sistema, o usuário poderá cadastrar os protocolos, sêmens e embriões, Figuras 5.17 e 5.18. Este sistema busca proporcionar aos proprietários de gado leiteiro ou veterinários uma ferramenta completa e eficiente para o controle e gestão de rebanhos, permitindo maior organização, tomada de decisões embasadas em dados precisos e otimização dos processos relacionados à produção de leite.

5.2 Envio de E-mail

A funcionalidade de envio de e-mail foi implementada neste *software* com o objetivo de permitir que os usuários entrem em contato com a equipe responsável. Tal funcionalidade possibilita que os usuários possam enviar solicitações de acesso ao sistema, elogios e sugestões de melhorias, proporcionando uma forma eficiente e ágil de comunicação entre os usuários e a equipe responsável pelo sistema.

A tela inicial do sistema apresenta um formulário de contato, ilustrada na Figura 5.19, onde o usuário pode preencher um formulário com seus dados pessoais, como nome, e-mail e telefone, a fim de enviar uma mensagem à equipe responsável.

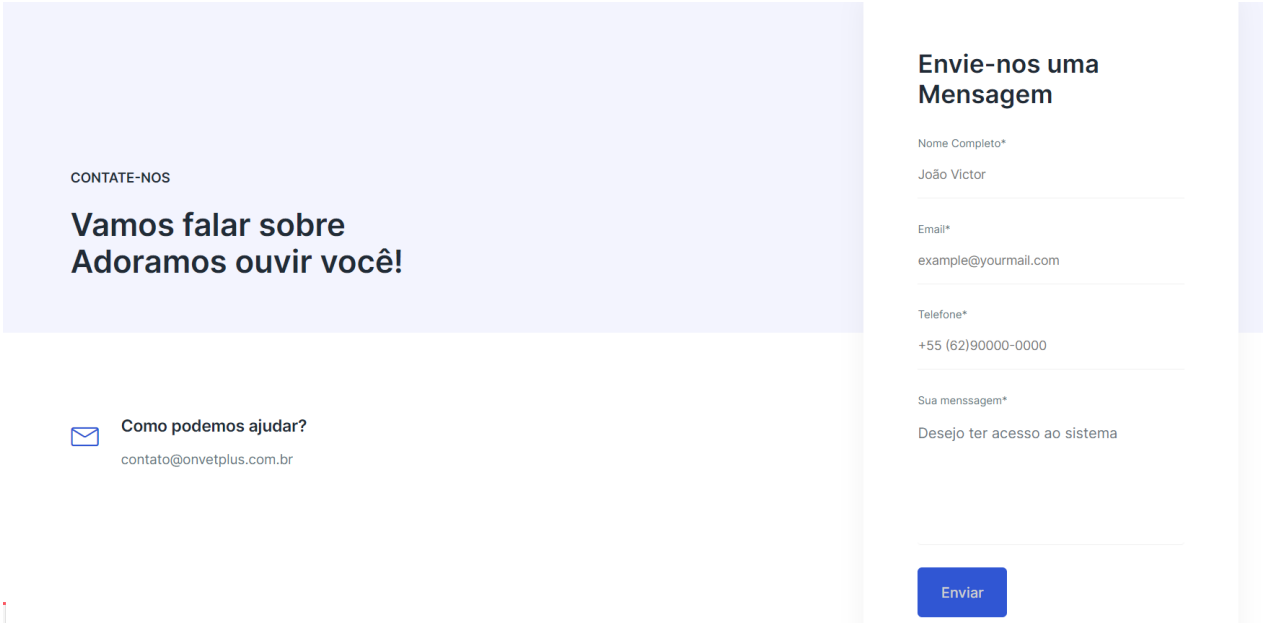

Figura 5.19: Formulário de envio de email apresentado ao usuário.

Para que os e-mails dos usuários sejam encaminhados para a caixa de e-mail específica (**contato@onvetplus.com.br**), é necessário configurar o arquivo `.env` do projeto com as informações de autenticação do servidor de e-mail utilizado.

Em seguida, na *controller*, parte da estrutura apresentada na Figura 4.3, responsável por receber os dados do formulário de contato e enviar o e-mail, é utilizado o método `send` da classe **Mail** do Laravel. Esse método recebe como parâmetro a classe **View** do e-mail a ser enviado, os dados do formulário e uma função de `callback` que configura o e-mail a ser enviado, incluindo o assunto, o destinatário e o remetente.

Assim que o usuário envia o formulário de contato, a *controller* recebe os dados preenchidos e configura o e-mail com o assunto, destinatário e remetente por meio de uma função `callback`. Em seguida, é utilizado o método `send` da classe **Mail** do Laravel para enviar o e-mail para a caixa de correio.

Na Figura 5.20, é apresentada a configuração do arquivo `.env`, que realiza a conexão entre o projeto Laravel e o servidor de e-mail da Hostinger, utilizando o host e a porta disponibilizados.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.hostinger.com.br
MAIL_PORT=587
MAIL_USERNAME=contato@onvetplus.com.br
MAIL_PASSWORD=Trabalho_curso2022
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=contato@onvetplus.com.br
MAIL_FROM_NAME="Contato Onvet"
```

Figura 5.20: Configuração de email presente no arquivo `.env` do projeto.

Após a configuração do servidor de e-mail, as rotas **web** são definidas no arquivo `web.php` como apresentado na Figura 5.21. A rota `/contact/submit` é configurada para ser acionada pelo *front-end* e gerenciada pela classe **ContactController.php**.

```
Route::get('/loginOnvet', function () { return redirect('dashboard');})->name('home');
Route::get('login', [LoginController::class, 'showLoginForm'])->name('login');
Route::post('login', [LoginController::class, 'authenticate']);
Route::get('logout', [LoginController::class, 'logout'])->name('logout');
Route::any('primeiro-acesso/{id}', [LoginController::class, 'primeiroAcesso'])->name('primeiroAcesso');
Route::post('redefinir-senha', [LoginController::class, 'redefinirSenha'])->name('redefinirSenha');
Route::post('/contact/submit', [\App\Http\Controllers\ContactController::class, 'submit']);
```

Figura 5.21: Rotas relacionadas a configuração de email no arquivo `web.php`.

Na classe *controller* **ContactController.php**, a função `submit`, apresentada na Figura 5.22, recebe um objeto `request` como parâmetro, o qual é também passado como parâmetro para o objeto `$contato`, que é uma instância da classe **ContactForm.php**. Essa classe é responsável por encapsular o envio de e-mail e o método `sendMail` é chamado pelo objeto `$contato` para enviar um e-mail. Se a tentativa de envio for bem-sucedida, a função retorna uma mensagem de sucesso, porém, caso ocorra um erro, uma mensagem de erro será exibida.

```
public function submit(Request $request)
{
    $request->validate([
        'fullName' => 'required',
        'email' => 'required|email',
        'phone' => 'required',
        'message' => 'required'
       ]);

//AQUI É A DIFERENÇA DO HACKER
$contato = new ContactForm($request);
try{
    $contato->sendMail();
      return back()
     ->with('sucess', 'Obrigado por nos contactar');
} catch(\Exception $error){
   return back()->with("error", "Ocorreu um erro inesperado: {$error->getMessage()}");
}
}
```

Figura 5.22: Classe **ContactController.php**, que contém a função `submit`.

No *front-end*, implementado no arquivo `index.blade.php`, é utilizado um formulário que utiliza a rota `/contact/submit` para enviar os dados do e-mail. Em seguida, um trecho de código verifica se existem erros de validação no formulário de contato e, caso o resultado seja positivo, exibe uma mensagem de alerta. Esse trecho de código utiliza a função `count`, que recebe o objeto `$errors` como parâmetro, para verificar se há erros de validação na requisição. Se houverem, uma mensagem de alerta será exibida contendo os erros encontrados. O objeto

`$error` é utilizado dentro do loop `foreach` para exibir cada erro encontrado na validação. Caso não haja erros, o trecho de código apresentado na Figura 5.23 não será executado.

```
<div class="col-xl-4 col-lg-5">
    <div class="ud-contact-form-wrapper wow fadeInUp" data-wow-delay=".2s">
        <h3 class="ud-contact-form-title">Envie-nos uma Mensagem</h3>
        <form action={{ url('/contact/submit') }} method="POST">      You, há 6 meses • send_ema
            {{ csrf_field() }}

            @if (count($errors) > 0)
                <div class="alert alert-warning alert-dismissible fade show" role="alert">
                    <strong>Preencha os dados corretamente!</strong>
                    <ul>
                        @foreach ($errors->all() as $error)
                            <li>{{ $error }}</li>
                        @endforeach
                    </ul>
                    <button type="button" class="btn-close" data-bs-dismiss="alert"
                        aria-label="Close"></button>
                </div>
            @endif
```

Figura 5.23: Trecho de código pertencente ao arquivo `index.blade.php`.

As mensagens de sucesso ou erro são armazenadas na sessão utilizando a classe **Session** do Laravel, conforme apresentado na Figura 5.24.

```
@if ($message = Session::get('sucess'))
    <div class="alert alert-sucess alert-dismissible fade show" role="alert">
        <strong>Obrigado!</strong> {{ $message }}
        <button type="button" class="btn-close" data-bs-dismiss="alert"
            aria-label="Close"></button>
    </div>
@endif      You, há 6 meses • send_email

@if ($message = Session::get('error'))
    <div class="alert alert-danger alert-dismissible fade show" role="alert">
        <strong>OOOPPPSSS!</strong> {{ $message }}
        <button type="button" class="btn-close" data-bs-dismiss="alert"
            aria-label="Close"></button>
    </div>
@endif
```

Figura 5.24: Classe responsável por armazenar as mensagens de sucesso ou erro.

Caso o usuário consiga enviar o email com sucesso, um alerta é exibido contendo a mensagem de sucesso, apresentado na figura 5.25, enquanto que, caso haja uma mensagem de erro, um alerta vermelho é exibido contendo a mensagem, como mostrado na figura 5.26.

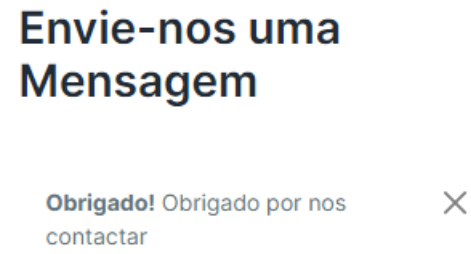

Figura 5.25: Mensagem de sucesso.

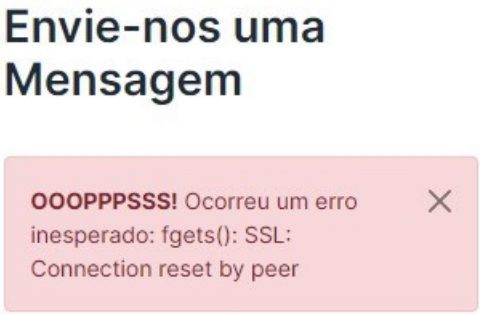

Figura 5.26: Exemplo de mensagem de erro.

Ao final do processo, os dados são enviados através do botão `Enviar`, como mostrado no trecho de código apresentado na Figura 5.27.

```
<div class="ud-form-group mb-0">
    <button type="submit" class="ud-main-btn">
        Enviar
    </button>
```

Figura 5.27: Botão `Enviar` utilizado no envio de e-mail.

5.3 Recuperação de Senha

Para permitir que o usuário redefina sua senha, foi inicialmente configurada uma nova rota no arquivo `web.php`, conforme mostrado na Figura 5.28. O processo é gerenciado pela classe **LoginController.php**, a função específica chamada `redefinirSenha` é responsável por tratar todas as ações relacionadas à redefinição da senha.

```
Route::get('/loginOnvet', function () { return redirect('dashboard');})->name('home');
Route::get('login', [LoginController::class, 'showLoginForm'])->name('login');
Route::post('login', [LoginController::class, 'authenticate']);
Route::get('logout', [LoginController::class, 'logout'])->name('logout');
Route::any('primeiro-acesso/{id}', [LoginController::class, 'primeiroAcesso'])->name('primeiroAcesso');
Route::post('redefinir-senha', [LoginController::class, 'redefinirSenha'])->name('redefinirSenha');
Route::post('/contact/submit', [\App\Http\Controllers\ContactController::class, 'submit']);
```

Figura 5.28: Rotas de recuperação de senha no arquivo **web.php**.

Esta seção está organizada em três subseções, a fim de apresentar detalhadamente a implementação da funcionalidade de recuperação de senha. Cada subseção aborda um aspecto específico do processo, proporcionando uma compreensão clara e coerente do código apresentado. Na subseção 5.3.1 são explicadas as rotas **web** utilizadas, na subseção 5.3.2 são explicadas as classes responsáveis por gerênciar as funcionalidades, e por fim, na subseção 5.3.3 é explicado como o usuário irá interagir com esta funcionalidade no sistema.

Esta organização em subseções é adotada para uma melhor estruturação e explicação do código envolvido na implementação da funcionalidade de recuperação de senha. Cada parte desempenha um papel específico e contribui para o funcionamento global dessa funcionalidade no sistema. Através dessa divisão o leitor entenderá com mais facilidade como o processo de recuperação de senha foi desenvolvido e integrado ao sistema.

5.3.1 Rotas de Redefinição de Senha

O código apresentado na Figura 5.29 mostra um trecho de rotas definidas no arquivo `web.php`, onde são definidas algumas rotas relacionadas à funcionalidade de recuperação de senha.

```
Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])
->name('password.request');

Route::post('forgot-password', [PasswordResetLinkController::class, 'store'])
->name('password.email');

Route::get('reset-password/{token}', [NewPasswordController::class, 'create'])
->name('password.reset');

Route::post('reset-password', [NewPasswordController::class, 'store'])
->name('password.update');
```

Figura 5.29: Rotas do arquivo `web.php` relacionadas a recuperação de senha.

Quatro rotas são mostradas no trecho de código da Figura 5.29, cada uma delas possui uma função diferente, que são:

1. A primeira rota, definida pelo método `GET`, é responsável por exibir um formulário para que

o usuário possa inserir o e-mail associado à sua conta para solicitar um *link* de redefinição de senha. Essa rota é nomeada como `password.request` e está presente no arquivo `web.php`, conforme mostrado na Figura 5.29.

2. A segunda rota é responsável por receber as informações do formulário da rota anterior, validar e, se tudo estiver correto, enviar um e-mail com um *link* de redefinição de senha para o endereço de e-mail informado. Essa rota tem o nome de `password.email`. Para tal funcionalidade é enviado um *token* com duração de 60 minutos ao e-mail do usuário.

3. No arquivo `web.php` é definida uma terceira rota para a funcionalidade de recuperação de senha, que permite ao usuário inserir uma nova senha após acessar o *link* de redefinição enviado para o e-mail. Essa rota recebe como parâmetro um *token* que identifica a solicitação de redefinição de senha e tem o nome de `password.reset`. Ela é definida pelo método `GET` e é responsável por exibir o formulário para a criação da nova senha.

4. A quarta rota, definida pelo método `post`, é responsável por receber as informações do formulário da rota anterior, validar e, se tudo estiver correto, atualizar a senha do usuário no banco de dados. Ela é acessada por meio do nome de rota `password.update`.

5.3.2 *Controllers* de Redefinição de Senha

As rotas implementadas possuem funções presentes em classes *controllers* específicas. Nesta subseção serão detalhadas as classes e as respectivas funções responsáveis por garantir o funcionamento adequado dessas rotas.

Na *controller* **LoginController.php**, a função `redefinirSenha` foi criada para recuperar o usuário a partir do ID especificado pelo valor do campo `id` no objeto `Request` utilizando o método `findOrFail()` da classe `User`. Em seguida, a função utiliza a função `bcrypt` para criptografar a senha do usuário e definir como valor do campo `resetPasswordNew` no objeto `Request`.

O valor da varável `redefinir_senha` do usuário é definido como `false` e as alterações no registro do usuário são salvas utilizando o método `save` da classe **User**. Por fim, o usuário é redirecionado para a página inicial, que pode ser definida por ele (`$user->home->url`) ou para a raiz do site (`'/'`). A Figura 5.30 apresenta a classe **LoginController.php**.

```
1 reference | 0 overrides
public function redefinirSenha(Request $request)
{
    $user = User::findOrFail($request->id);
    $user->password = bcrypt($request->resetPasswordNew);
    $user->redefinir_senha = false;
    $user->save();
    $redirect = $user->home->url ?? '/';
    return redirect($redirect);
}
```

Figura 5.30: Função `redefinirSenha` na classe **LoginController.php**.

A função **create** mostrada na Figura 5.31, está presente na classe **PasswordResetLinkController.php** e tem a finalidade de retornar ao usuário a *view* cujo caminho é **auth.forgot-password**, a qual contém o formulário para que o usuário possa solicitar a redefinição de sua senha através do preenchimento do e-mail associado à sua conta.

```
4 references | 0 overrides
public function create()
{
    return view('auth.forgot-password');
}       You, há 7 meses • commit_mail_init …
```

Figura 5.31: Função `create` presente na classe **PasswordResetLinkController.php**.

Na Figura 5.32 é apresentada a implementação da função `store` presente na classe **PasswordResetLinkController.php**. Essa função é responsável por enviar um *link* de redefinição de senha para o e-mail fornecido pelo usuário, validar o campo de e-mail recebido através de uma solicitação *POST* e chamar a função `Password::sendResetLink` para enviar o *link* de redefinição de senha para o e-mail fornecido.

Se o envio do *link* de redefinição de senha for bem sucedido, o usuário é redirecionado para a página anterior com uma mensagem de sucesso. Caso contrário, o usuário é redirecionado para a página anterior com uma mensagem de erro.

```php
public function store(Request $request)
{
    $request->validate([
        'email' => ['required', 'email'],
    ]);

    // We will send the password reset link to this user. Once w
    // to send the link, we will examine the response then see t
    // need to show to the user. Finally, we'll send out a prope
    $status = Password::sendResetLink(
        $request->only('email')
    );

    return $status == Password::RESET_LINK_SENT
                ? back()->with('status', __($status))
                : back()->withInput($request->only('email'))
                        ->withErrors(['email' => __($status)]);
}
```

Figura 5.32: Função `store` responsável por enviar o *link* de redefinição de senha.

A função apresentada na Figura 5.33 pertence à classe **NewPasswordController.php** e tem como objetivo retornar ao usuário a *view* **auth.reset-password** e passar para a mesma um parâmetro `request`, que é um *array* contendo os dados da requisição HTTP recebidos. Essa função é acionada quando o usuário acessa a página de redefinição de senha e precisa ser apresentado um formulário para que ele possa inserir a sua nova senha.

```php
public function create(Request $request)
{
    return view('auth.reset-password', ['request' => $request]);
}
```

Figura 5.33: Função `create` pertencente a classe **NewPasswordController.php**.

O código apresentado na Figura 5.34 contém a função `store`, que faz o tratamento da requisição de redefinição de senha do usuário na classe **NewPasswordController.php**. Nessa função, é realizada a validação dos campos enviados pelo usuário, como o *token*, e-mail e senha, além de tentar redefinir a senha do usuário e retornar uma resposta apropriada. Essa função é acionada quando o usuário submete o formulário de redefinição de senha presente na *view* **auth.reset-password**.

```php
public function store(Request $request)
{
    $request->validate([
        'token' => ['required'],
        'email' => ['required', 'email'],
        'password' => ['required', 'confirmed', Rules\Password::defaults()],
    ]);

    // Here we will attempt to reset the user's password. If it is successful we
    // will update the password on an actual user model and persist it to the
    // database. Otherwise we will parse the error and return the response.
    $status = Password::reset(
        $request->only('email', 'password', 'password_confirmation', 'token'),
        function ($user) use ($request) {
            $user->forceFill([
                'password' => Hash::make($request->password),
                'remember_token' => Str::random(60),
            ])->save();

            event(new PasswordReset($user));
        }
    );
```

Figura 5.34: Função `store` da classe **NewPasswordController.php**.

A validação é feita utilizando o método `validate()` do objeto `$request`, que define as regras de validação para cada campo, e em seguida, a lógica para redefinir a senha é executada utilizando o método `Password::reset`, que recebe como parâmetro um *array* com as informações necessárias para redefinir a senha (e-mail, *token* e nova senha) e uma função de retorno que é chamada se a redefinição for bem-sucedida.

Na função de retorno, a nova senha é criptografada utilizando o método `Hash::make`, o *token* de lembrança de senha é gerado utilizando o método `Str::random` e o modelo do usuário é atualizado com essas informações e salvo no banco de dados. Por fim, um evento `PasswordReset` é disparado para notificar outras partes do sistema sobre a redefinição da senha.

5.3.3 *Views* de Redefinição de Senha

Para permitir a interação do usuário com o sistema, são utilizadas diversas *views* que são chamadas pelas funções das classes controladoras. Essas *views* apresentam ao usuário formulários, botões e outras funcionalidades que permitem a realização das ações desejadas. A seguir serão explicadas as *views* utilizada nestas funcionalidades.

A Figura 5.35 apresenta a *view* para recuperar senha, chamada de `forgot-password.blade.php`. Nela, é exibido um formulário onde o usuário pode inserir seu endereço de e-mail para receber

um link de redefinição de senha. O link enviado terá validade de 60 minutos. Essa *view* é chamada pelo método `create` da classe `PasswordResetLinkController`.

Figura 5.35: *View* apresentada ao usuário para que insira seu e-mail a ser recuperado.

Clicando no *link*, como mostrado na Figura 5.36, o usuário é redirecionado para uma página na aplicação. Essa página é responsável por verificar se a senha fornecida está correta e salvá-la novamente, concedendo assim acesso ao usuário. Esse *link* também é enviado pelo método `store` pertencente a classe *controller* chamada `PasswordResetLinkController`.

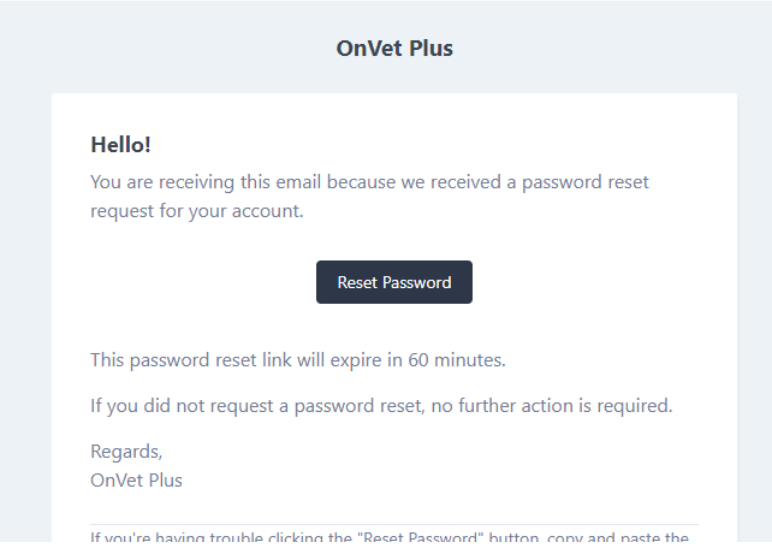

Figura 5.36: Mensagem que o usuário recebe em seu email após solicitar a recuperação de senha.

5.4 Criptografia de Senha

Na seção 5.3, foi mencionado um dos dois momentos em que existe a necessidade de criptografar uma senha, sendo ele o caso em que o usuário precisa redefinir sua senha. Nesta subseção, é detalhada a criptografia de senha que ocorre durante o registro de um novo usuário.

A função `save` é implementada neste *software* para registrar um novo usuário. Ele está presente na classe **UserController.php**, conforme mostrado na Figura 5.37. O objeto `validate` é uma instância da classe **Validator**, e é usado para construir um arquivo JSON que será verificado posteriormente. Se tudo estiver correto na inserção dos dados, o código segue seu fluxo normal.

```
public function save(Request $request)
{
    $validate = Validator::make($request->all(), [
        'name' => 'required|max:255',
        'role_id' => 'required',
        'home_id' => 'required',
        'email' => 'required|max:255',

    ]);
```

Figura 5.37: Função ***save***, utilizada no cadastro de usuários.

Na função `save` do código em questão, existe uma condicional, mostrada na Figura 5.38 que verifica se o campo `password` do objeto `request` não é vazio. Caso não seja vazio, o campo `password` do objeto `user` é criptografado usando o método `bcrypt`. O método `bcrypt` é uma função de `hash` de senhas disponibilizada pela biblioteca `Facades` do Laravel (LARAVEL, 2023). Ele usa o algoritmo de `hash` **bcrypt**, que é um algoritmo de criptografia de senha forte e seguro. O `bcrypt` gera um `hash` de senha seguro, que é armazenado no banco de dados no lugar da senha em texto puro. Isso garante a segurança dos dados, pois mesmo que o banco de dados esteja comprometido, a senha não pode ser tratada facilmente.

```
if($request->password != "") {
    $user->password = bcrypt($request->password);
}
```

Figura 5.38: Condição para que aconteça a criptografia.

5.5 Contadores

Os contadores são elementos apresentados nas *views* que são resultados da soma dos itens armazenados no banco de dados que correspondem ao usuário em questão. São úteis em sistemas

para fins de monitoramento, controle de fluxo e rápida tomada de decisões. Para explicar essa funcionalidade, foi escolhido o cadastro de Áreas. O processo inicia-se sempre na classe modelo, neste caso a classe em questão é a **Area.php**, na qual é desenvolvido a função `scopeFiltros`. A Figura 5.39 mostra a função que recebe como parâmetros os objetos `query` e `request`.

```
0 references | 0 overrides
public function scopeFiltros($query, $request)
{
    if (isset($request->search) && $request->search != "") {
        $query->where(function ($q) use ($request) {
            $q->where('nome', 'like', "%{$request->search}%");
        });
    }

    if (isset($request->ativo) && $request->ativo != "") {
        $query->where(function ($q) use ($request) {
            $q->where('ativo', $request->ativo);
        });
    }

    return $query;
}
```

Figura 5.39: Função `scopeFiltros` da classe **Area.php**.

Nessa função, realiza-se o filtro para uma consulta no banco de dados com base nas informações recebidas na rota, ou seja, via HTTP. O objeto `query` que é uma instância da classe **QueryBuilder**, representa a consulta que será executada na base de dados. Na primeira condição, caso o parâmetro `request` contenha um campo `search`, ou seja, não esteja vazio, a função adiciona uma cláusula `where` para procurar registros que atendam à palavra-chave digitada no campo de busca. Essa busca é feita utilizando o operador `like`.

Para contar quantos registros existem, utilizam-se os campos ativo e inativo. Na segunda condição, caso o parâmetro `request` receba algum campo ativo e não vazio, a função adiciona uma cláusula `where` para filtrar registros com base no valor do campo ativo. Ao final da função, retorna-se a `query`, ou seja, as consultas na base de dados atualizadas com as novas cláusulas `where`.

Utilizando o padrão de projeto MVC, a função filtros será chamada e as consultas feitas previamente na *model* serão utilizadas na classe *controller* **AreaController.php**. Neste trecho de código, apresentado na Figura 5.40, são filtrados os resultados da classe modelo com base nas informações recebidas pelo objeto `request`, utilizando o método filtros. Em seguida, é selecionada a contagem de áreas ativas e inativas utilizando a função `SUM` e a condição `IF` do `MySql`, respectivamente, passadas no método `select` do modelo.

```php
if (auth()->user()->role_id == 1) {
    $resume = $this->model::filtros($request)
        ->select(
            DB::raw('SUM(IF(ativo = 1, 1 ,0)) as ativos'),
            DB::raw('SUM(IF(ativo = 0, 1 ,0)) as inativos')
        )
        ->where('id', '>', 0)
        ->first();
} else {
    $resume = $this->model::filtros($request)
        ->select(
            DB::raw('SUM(IF(ativo = 1, 1 ,0)) as ativos'),
            DB::raw('SUM(IF(ativo = 0, 1 ,0)) as inativos')
        )
        ->where('id', '>', 0)
        ->where('user_id', $user_id) // Filtar fazendas pelo I
        ->first();

}
```

Figura 5.40: Condições para que o usuário tenha acesso aos dados filtrados.

Verifica-se se o usuário logado é um administrador, se este for o caso, todas as áreas com id maiores que 0 serão exibidas. Caso o usuário não seja um administrador, apenas as áreas do usuário autenticado serão selecionadas, utilizando uma cláusula `where` que filtra pela chave estrangeira `user_id`. Por fim, o resultado é armazenado no objeto `resume`, que é retornado pela função e será utilizado na *view*.

Para exibir esses dados na tela, é necessário acessar o objeto `resume` vindo da classe *controller* para obter a quantidade total de itens cadastrados, é realizada a soma dos resultados de itens ativos e inativos. A Figura 5.41 mostra como é realizado este processo na *view*.

```
<div class="card-body statistics-body">
    <div class="row">
        <div class="col-md-4 col-sm-6 col-12 mb-2 mb-md-0" style="paddin
            <div class="media">
                <div class="avatar bg-light-primary mr-2">
                    <div class="avatar-content">
                        <i data-feather='slack'></i>
                    </div>
                </div>
                <div class="media-body my-auto">
                    <h4 class="font-weight-bolder mb-0">        You, há 2
                        {{ $resume->ativos + $resume->inativos }}
                    </h4>
                    <p class="card-text font-small-3 mb-0">Áreas</p>
                </div>
            </div>
        </div>
```

Figura 5.41: Código HTML e PHP responsável pela apresentação dos contadores.

Com a implementação do código apresentado e com a inserção de novas áreas pelo usuário, o resultado é como o mostrado na Figura 5.42, facilitando a visualização pelo usuário da quantidade de áreas cadastradas e quais dessas áreas estão ativas ou inativas. Pode-se observar na imagem que existem quatro áreas cadastradas no sistema, e todas estão sendo utilizadas (ativos). Se o usuário não mais utilizá-las, ele poderá alterar o cadastro para inativo a qualquer momento.

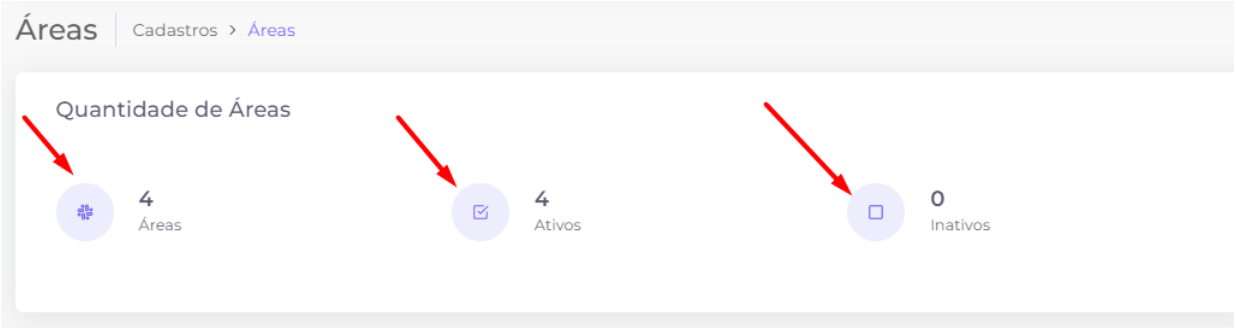

Figura 5.42: Painel informativo sobre a quantidade de áreas cadastradas.

5.6 API Google Charts

Google Charts é uma API de visualização de dados que permite aos desenvolvedores criar gráficos interativos para seus aplicativos da WEB. É uma biblioteca de gráficos JavaScript gratuita que oferece uma variedade de opções de gráficos, incluindo gráficos de barras, gráficos de pizza, gráficos de linhas, gráficos de dispersão, gráficos de área, entre outros (GOOGLE, 2022). É de fácil uso, pois requer apenas conhecimentos básicos de HTML e JavaScript para

incorporar gráficos em uma página da WEB.

Além disso, os gráficos gerados pela API do *Google Charts* são compatíveis com a maioria dos navegadores modernos e responsivos, adaptando-se automaticamente a diferentes tamanhos de tela. Embora ofereça muitas opções de personalização, a API do Google Charts pode ter limitações em termos de customização avançada, dependendo das necessidades específicas do desenvolvedor.

Para a construção da *view* Animais, onde o usuário registra todos os seus animais no sistema, foi necessário elaborar uma tabela específica para estes animais, onde serão armazenados os dados dos mesmos, como peso, sexo, tipo sanguíneo, raça, brinco, sua origem, data de nascimento, entre outros. A partir desta tabela do banco de dados, foi construída a classe modelo chamada **Animal.php**, responsável por representar uma entidade 'animal' no sistema. Esta modelo fornece um conjunto de métodos e funcionalidades que possibilita a interação com os dados armazenados no banco de dados, assim permitindo que a **DashboardController.php** e a **AnimalController.php** acessem e manipulem estes dados da forma necessária.

A Figura 5.43 apresenta um trecho de código presente na **DashboardController.php**, classe controladora responsável por gerar os *arrays* que alimentam todos os gráficos presentes no *Dashboard*. A diferença nessa controladora é que a classe modelo utilizada para importação de dados é a **Animal.php**, mesma classe modelo utilizada pela **AnimalController.php**, responsável pela *view* Animais. A reutilização de uma mesma *model* em diferentes *controllers* é possível após seguir os padrões da arquitetura MVC.

A função apresentada na Figura a seguir começa definindo a variável `$breadcrumbs` e obtendo o ID do usuário autenticado através do método `Auth::id()`. Em seguida, verifica se o usuário autenticado tem o papel de administrador (`role_id = 1`). Se sim, ele realiza uma consulta ao banco de dados para contar todos os animais que estão ativos e inativos. Caso seja um usuário comum, ele conta somente os animais que estão relacionados ao seu nível de acesso e fazendas. O resultado final é armazenado em uma matriz chamada `$array`, que é usada para gerar o gráfico na *view* correspondente.

```php
public function index(Request $request)
{
  $breadcrumbs = $this->breadcrumbs;
  $user_id = Auth::id(); // Obter o ID do usuário autenticado

  if (auth()->user()->role_id == 1) {
    $resume_animal = Animal::filtros($request)
      ->select(
        DB::raw('SUM(IF(ativo = 1, 1 ,0)) as ativos'),
        DB::raw('SUM(IF(ativo = 0, 1 ,0)) as inativos')
      )
      ->where('id', '>', 0)
      ->first();

      //  função para o gráfico de animais
    $data = DB::table('animais')
      ->select(
      DB::raw('raca as raca'),
      DB::raw('count(*) as number')
      )
      ->groupBy('raca')
      ->get();
    $array[] = ['Raca', 'Number'];

  foreach ($data as $key => $value) {

    $array[++$key] = [$value->raca, $value->number];
  }
```

Figura 5.43: Trecho de código responsável por gerar o *array* que alimenta o gráfico.

A 5.44 apresenta o trecho de código presente na *view* que é responsável por exibir os gráficos aos usuários. Este trecho de código responsável pela chamada da API através da tag `<script>`, onde é passado o *array* `$raca` que contém as informações necessárias para alimentar o gráfico de pizza que exibe a quantidade de animais por raça. Todos os *arrays* são importados da classe **DashboardController.php**. Este código pode ser replicado, alterando o *array* passado como parâmetro na variável `analytics` para que diferentes gráficos de pizza sejam gerados.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>

{{-- Grafico de animais por raça --}}

<script type="text/javascript">
    var analytics = <?php echo $raca; ?>

    google.charts.load('current', {
        'packages': ['corechart']
    });

    google.charts.setOnLoadCallback(drawChart);

    function drawChart() {
        var data = google.visualization.arrayToDataTable(analytics);
        var chart = new google.visualization.PieChart(document.getElementById('pie_chart'));
        chart.draw(data);
    }
</script>
```

Figura 5.44: Código JavaScript responsável por importar e gerar o gráfico na *view*.

Por fim, a Figura 5.45 apresenta um dos gráficos de pizza gerados pela API Google *Charts*, apresentado no *Dashboard*. Este gráfico é gerado a partir dos dados inseridos pelo usuário na tabela **Animais**, utilizando as raças dos animais como filtro e apresentando o percentual de cada raça em questão.

Figura 5.45: Gráfico de pizza apresentado ao usuário no *Dashboard*.

5.7 Exportação de Arquivos

Como explicado na seção 3.2, existem necessidades que os usuários têm em relação ao *software*, uma delas é a consulta de dados, mas de modo que o usuário consiga realizar a exportação de arquivos do sistema. Trata-se da possibilidade de gerar um arquivo, contendo nele os dados cadastrados pelo usuário no sistema, tais como dados de funcionários e animais, permitindo ao mesmo visualizar e estudar seus dados para uma análise mais profunda de sua

fazenda. Neste *software*, essa necessidade é atendida por meio do desenvolvimento de uma funcionalidade que permite ao usuário gerar arquivos nos formatos PDF e XLS.

Para a explicação da funcionalidade utilizada em todos os cadastros do projeto, foi escolhido o método utilizado no cadastro de fazendas. Na classe **FazendaController.php**, foi desenvolvido o método `indexPdf`, que recebe como parâmetro o objeto `fazendas`, oriundo da *model* **Fazenda.php**. A Figura 5.46 mostra o trecho de código que implementa a função `indexPdf` que retorna o *download* do arquivo no formato PDF.

```
private function indexPdf($fazendas)
{
    $fazendas = $fazendas->get();
    return \PDF::loadView('modules/cadastro/fazenda/indexPdf', compact('fazendas'))
        ->setPaper('a4')
        ->download('Fazendas.pdf')
    ;
}
```

Figura 5.46: Método `indexPdf` responsável por retornar o arquivo em PDF.

Após executar essas operações, a rota responsável pela tela de exportação do arquivo, assim como o nome e tipo do arquivo, são atribuídas ao objeto `view`. Além disso, os dados da tabela são obtidos por meio do objeto `dados`. Esses objetos são enviados para a *view* através do comando *return*, que utiliza a classe Excel e a função `download` como parâmetros.

A função inicialmente executa o método ***GET*** no objeto `fazendas`, carregando todos os dados das fazendas no banco de dados. Em seguida, a função utiliza a biblioteca PDF do *Framework* Laravel para carregar uma `view` que será utilizada para gerar o arquivo PDF. Finalmente, a função retorna para a `view` o objeto `fazendas` através do método `compact`, e então chama o `download`, que retornará o arquivo `Fazendas.pdf`.

Na função `indexExcel` da classe **FazendaController.php**, o objeto `fazendas` é recebido como parâmetro e é aplicado o método ***GET*** para buscar todos os dados da tabela `fazendas`. Em seguida, é definida a rota da tela onde será exportado o arquivo e passados o nome e tipo do arquivo para o objeto, juntamente com os dados vindos da tabela por meio do objeto `dados`. A Figura 5.47 apresenta a função `indexExcel`, que retorna o *download* do arquivo no formato XLS.

```
1 reference
private function indexExcel($fazendas)
{
    $fazendas = $fazendas->get();
    $view = 'modules/cadastro/fazenda/indexExcel';
    $arquivo = 'Fazendas.xlsx';       Joao Queiroz, há 3 semanas • fa
    $dados = ['fazendas' => $fazendas];

    return Excel::download(new ExcelExport($view, $dados), $arquivo);
}
```

Figura 5.47: A classe *ExcelExport* é responsável por gerar o arquivo XLS.

Por fim, os objetos mencionados são enviados para a `view` por meio do comando *return*, utilizando a classe Excel e a função `download` como parâmetros. Dessa forma, a função `indexExcel` irá gerar e retornar o arquivo no formato XLS com os dados da tabela `fazendas`.

Ao clicar em um dos botões para gerar o arquivo PDF ou XLS na tela principal do cadastro de fazendas, é feita uma requisição ao servidor, passando como parâmetro o tipo de arquivo desejado (PDF ou XLS). Em seguida, a função correspondente no controlador é acionada, recebendo como parâmetro o objeto contendo os dados da tabela de fazendas. A partir desse objeto, é utilizado o método apropriado para buscar os dados da tabela no banco de dados e, em seguida, é gerada a visualização correspondente, utilizando a biblioteca PDF ou XLS do *Framework* Laravel. Por fim, o arquivo é retornado como um *download* para o usuário.

No trecho de código apresentado na Figura 5.48, é gerado um arquivo PDF que contém uma tabela com informações de fazendas. O tamanho da página do PDF é definido como `A4` e o arquivo é baixado com o nome `Fazendas.pdf`.

```
<div class="col-md-2 col-12 mb-50">
    <div class="form-group">
        <button class="btn btn-outline-secondary btn-block text-nowrap px-1 waves-effect"
            type="submit" name="export" value="XLS">
            <i data-feather='download'></i>
            <span>Excel</span>
        </button>
    </div>
</div>
<div class="col-md-2 col-12 mb-50">
    <div class="form-group">
        <button class="btn btn-outline-secondary btn-block text-nowrap px-1 waves-effect"
            type="submit" name="export" value="PDF">
            <i data-feather='download'></i>
            <span>PDF</span>
        </button>
    </div>
</div>
</div>
```

Figura 5.48: Código HTML responsável por gerar os botões para *download* dos arquivos PDF e XLS.

No cadastro de fazendas, são disponibilizados dois botões para geração de arquivos,

um em formato PDF e outro em formato XLS. Esses botões são exibidos na tela principal do cadastro, como mostrado na Figura 5.50. Ao clicar em um dos botões, é feita uma requisição ao servidor, passando como parâmetro o tipo de arquivo desejado (PDF ou XLS). O arquivo PDF é gerado a partir da *view* `'modules/cadastro/fazenda/indexPdf'`, apresentada na Figura 5.49, que contém uma tabela com informações das fazendas, exibindo os campos nome, endereço, cidade/uf e status. Para cada objeto da classe **Fazenda.php**, as informações são percorridas com um laço de repetição `foreach` e exibidas em uma linha da tabela. A função `Helper::GETAtivoInativo` é usada para retornar se a fazenda está ativa ou inativa.

```
@extends('layouts.templatePDF', ['header' => 'Fazendas', 'title' => ''])
@section('content')
    <table class="table-linhas">
        <thead>
            <tr>
                <th>Nome</th>
                <th>Endereço</th>
                <th>Cidade / UF</th>
                <th>Status</th>
            </tr>
        </thead>
        <tbody>
            @foreach ($fazendas as $fazenda)
                <tr>
                    <td>
                        {{ $fazenda->nome }}
                    </td>
                    <td>
                        {{ $fazenda->endereco }}
                    </td>P
                    <td>
                        {{ $fazenda->cidade }} - {{ $fazenda->uf }}
                    </td>
                    <td>
                        {!! Helper::getAtivoInativo($fazenda->ativo, true) !!}
                    </td>
                </tr>
            @endforeach
        </tbody>
    </table>
@endsection
```

Figura 5.49: *Front-end* do arquivo PDF de fazendas.

Já o arquivo XLS é gerado a partir da função `indexExcel` da classe **FazendaController.php**, que recebe como parâmetro o objeto `fazendas` e aplica o método ***GET*** para buscar todos os dados da tabela `fazendas`. O arquivo é então definido com um nome e tipo, e os dados são passados por meio do objeto `dados` para serem exportados no formato XLS.

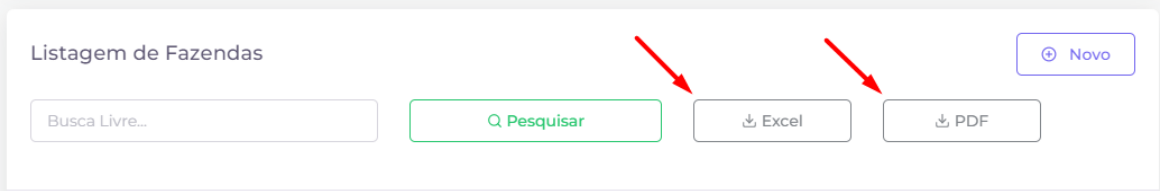

Figura 5.50: Apresentação dos botões de arquivos ao usuário.

A Figura 5.51 trás o arquivo gerado pelo botão e que está salvo na máquina do usuário, este arquivo contém todas as informações das fazendas cadastradas que estão relacionadas ao usuário logado em questão. A estrutura do arquivo segue de acordo com as tabelas do banco de dados que são exibidas ao usuário, que no caso das fazendas são: Nome, Endereço, Cidade/Uf e Status.


onvet

Fazendas

Nome	Endereço	Cidade / UF	Status
Fazenda User 1	Rua Claudina Gomes da Silva	Ipameri - GO	Ativo
Fazenda User 2	Rua VS7	Ipameri - GO	Ativo

Figura 5.51: Arquivo PDF exportado pelo sistema contendo informações sobre fazendas de um determinado usuário.

A Figura 5.52 apresenta o arquivo XLS gerado pelo botão de exportação de dados das fazendas. Este arquivo contém todas as informações cadastradas das fazendas e pode ser salvo na máquina do usuário. A estrutura do arquivo é composto de linhas e colunas, sendo as colunas dividas em Nome, Endereço, Cidade/Uf e Status, e as linhas variando de acordo com o número de registros do usuário.

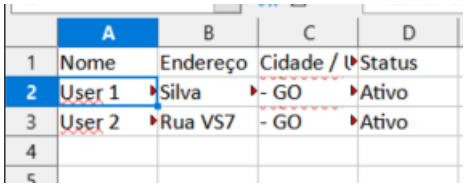

Figura 5.52: Arquivo XLS exportado pelo sistema contendo informações sobre fazendas de um determinado usuário.

5.8 API CEP

Ao adicionar novas fazendas, funcionários ou fornecedores o usuário deve inserir o endereço em questão. Ao inserir no campo `CEP` o código postal correspondente ao endereço, a API Postmon realiza a consulta utilizando este valor para retornar um arquivo JSON contendo informações como bairro, cidade, logradouro, estado, e então este arquivo é consumido pelo sistema, assim preenchendo automaticamente os campos `endereço, bairro, cidade e UF`,

como é apresentado na Figura 5.53.

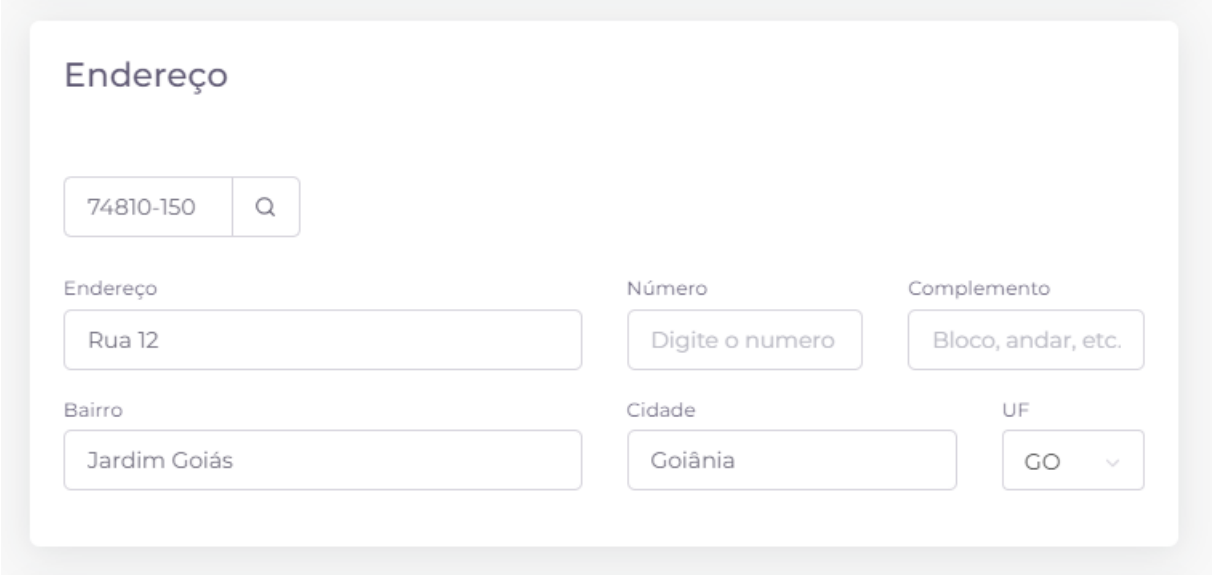

Figura 5.53: Formulário de inserção de dados relacionados ao endereço.

A Figura 5.54 apresenta a função que utiliza o *Framework* jQuery para fazer uma requisição **AJAX** para a API de consulta de CEP do site Postmon (**https://postmon.com.br/**). Ela recebe como parâmetro o número do CEP informado pelo usuário no campo `CEP` e utiliza a função `replace` para remover o traço que pode estar presente no valor.

Em seguida, a função realiza a requisição ***GET*** para a API Postmon, utilizando o valor do CEP para buscar as informações de endereço correspondentes. Se a requisição for bem sucedida, as informações de endereço são preenchidas automaticamente nos campos `endereco, bairro, cidade e uf`, utilizando a resposta da API.

```
function searchPostalCode() {
    $.getJSON('https://api.postmon.com.br/v1/cep/' + $('#cep').val().replace('-', '')).done(function(data2) {
        if (!data2) {
            notify('error', 'CEP não encontrado!');
        } else {
            $('#endereco').val(data2.logradouro);
            $('#bairro').val(data2.bairro);
            $('#cidade').val(data2.cidade);
            $('#uf').val(data2.estado);       João, 7 months ago • correcoes_campos_inteiros
            $("#uf").select2();
        }
    });
}
```

Figura 5.54: Função responsável pela utilização da API `Postmon`.

5.9 Contato de Funcionários e Fornecedores

Esta funcionalidade, presente no cadastro de Funcionários e Fornecedores, consiste na possibilidade de vincular um item cadastrado específico à quantidade de contatos necessários. Por exemplo, para um fornecedor que tenha vários contatos, o usuário pode salvá-los e excluí-los separadamente. Para isso, este cadastro necessitou de duas tabelas no banco de dados: a tabela **fornecedor** e a tabela **fornecedor_contato**. Esta segunda tabela recebe uma chave estrangeira chamada `id_fornecedor`, que referencia a tabela **fornecedor** pelo seu campo `id`. Além disso, esta tabela também recebe as colunas `telefone`, `email` e `timestamps`, e status de cada registro.

A Figura 5.55 mostra como foi desenvolvida a classe *migration* **CreateCadastrosFornecedorContato**, para a criação da tabela `fornecedor_contato`. Nesta classe, foram definidas todas as colunas, chave primária e chave estrangeira citadas no parágrafo acima. As classes *migrations* são explicadas na subseção 2.4.2. Para que este tipo de classe seja interpretada e suas alterações sejam aplicadas no banco de dados, é necessário o uso da biblioteca *artisan*, como mostrado na Figura 2.10.

```php
class CreateCadastrosFornecedorContato extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    0 references | 0 overrides
    public function up()
    {
        Schema::create('fornecedor_contato', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('fornecedor_id');
            $table->foreign('fornecedor_id')
                ->references('id')
                ->on('fornecedor')      You, há 9 meses • commit_migration
            ;
            $table->string('nome', 50);
            $table->string('telefone', 20);
            $table->string('email');
            $table->timestamps();
        });
    }
```

Figura 5.55: Classe *migration* relacionada ao contato de fornecedores.

Houve também a necessidade de implementar outras duas *migrations* para que fosse possível realizar o tratamento da tabela `fornecedor_contato`. A primeira *migration*, **AlterCadastrosFornecedorContato.php**, é apresentada na Figura 5.56. Nessa *migration*, apenas o tipo

de dados da coluna `telefone` é alterado para `string` com tamanho máximo de 20 caracteres.

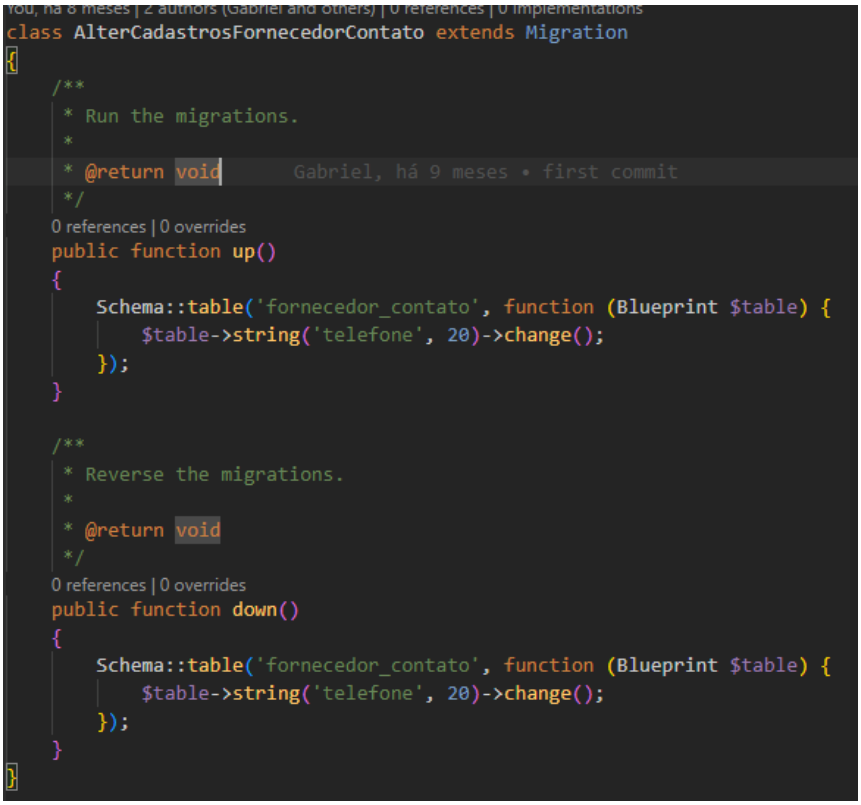

Figura 5.56: Classe **AlterCadastrosFornecedorContato.php**, primeira classe responsável pelo cadastro de contatos.

Já a segunda *migration*, **AlterCadastrosFornecedorContato2.php**, é apresentada na Figura 5.57. Essa migração realiza várias alterações na tabela, tornando as colunas `telefone`, `nome` e `email` opcionais (anuláveis), ou seja, permitindo que seus valores sejam nulos, enquanto anteriormente eram obrigatórios (`nullable(false)`). Isso é possível adicionando o método `nullable` antes do método `change`. Além disso, essa *migration* também altera o tipo de dados da coluna `email` para `string` sem tamanho máximo definido.

```php
class AlterCadastrosFornecedorContato2 extends Migration
{
    /**
     * Run the migrations.
     *       You, há 9 meses • commit_migrations_check
     * @return void
     */
    0 references | 0 overrides
    public function up()
    {
        Schema::table('fornecedor_contato', function (Blueprint $table) {
            $table->string('telefone', 20)->nullable()->change();
            $table->string('nome', 50)->nullable()->change();
            $table->string('email')->nullable()->change();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    0 references | 0 overrides
    public function down()
    {
        Schema::table('fornecedor_contato', function (Blueprint $table) {
            $table->string('telefone', 20)->nullable(false)->change();
            $table->string('nome', 50)->nullable(false)->change();
            $table->string('email')->nullable(false)->change(); ;
        });
    }
}
```

Figura 5.57: Classe **AlterCadastrosFornecedorContato2.php**, segunda classe responsável pelo cadastro de contatos.

É necessário dividir estas alterações em duas *migrations* distintas, pois cada uma deve ser responsável por apenas uma lógica específica na estrutura da tabela, para garantir que as operações de *rollbacks* e atualizações sejam consistentes e previsíveis. Além disso, dividir as mudanças em várias *migrations* torna mais fácil rastrear e corrigir possíveis erros que possam surgir.

Seguindo o padrão MVC, foram desenvolvidas duas classes modelos para este cadastro: **Fornecedor.php** e **FornecedorContato.php**. O trecho de código apresentado na Figura 5.58 define um relacionamento entre as classes usando o método `hasMany`, indicando que um fornecedor pode ter vários contatos. A função `contatos` retorna uma coleção de objetos da classe **FornecedorContato.php** que estão relacionados a um objeto da classe **Fornecedor.php**.

No método `hasMany`, o primeiro parâmetro é o nome do modelo relacionado e o segundo parâmetro é a chave estrangeira que será usada para realizar a ligação entre as duas tabelas. Neste caso, a chave estrangeira é `fornecedor_id`, presente na tabela `fornecedor_contato`.

```php
0 references | 0 overrides
public function contatos()
{
    return $this->hasMany(FornecedorContato::class, 'fornecedor_id');
}
```

Figura 5.58: Função `contatos` retornando o objeto da classe **FornecedorContato.php**.

Na *controller* **FornecedorController.php**, foi implementado um trecho de código relacionado aos contatos, mostrado na Figura 5.59. Utiliza-se o método `withCount()` para contar os contatos de fornecedores que possuem o mesmo nome e armazenar o resultado na coluna `contato`.

```php
public function index(Request $request)
{
    $breadcrumbs = $this->breadcrumbs;
    $user_id = Auth::id(); // Obter o ID do usuário autenticado
    $fornecedores = Fornecedor::filtros($request)
        ->withCount([
            'contatos as contato' => function ($query) {
                $query->where('nome', "Ale");
            }
        ])      You, há 2 meses • :constructions:feat filtros e chaves estrangeiras…
        ->where('user_id', $user_id) // Filtar fazendas pelo ID do usuário autenticado
        ->orderBy('nome', 'ASC');
```

Figura 5.59: Função `index` presente na classe `FornecedorController.php`.

Para possibilitar a inserção e remoção de vários contatos para cada fornecedor, foi desenvolvido uma interface onde o usuário tem a opção de adicionar os valores de `nome, telefone e e-mail` do contato. A Figura 5.60 mostra como o usuário visualizará esta funcionalidade.

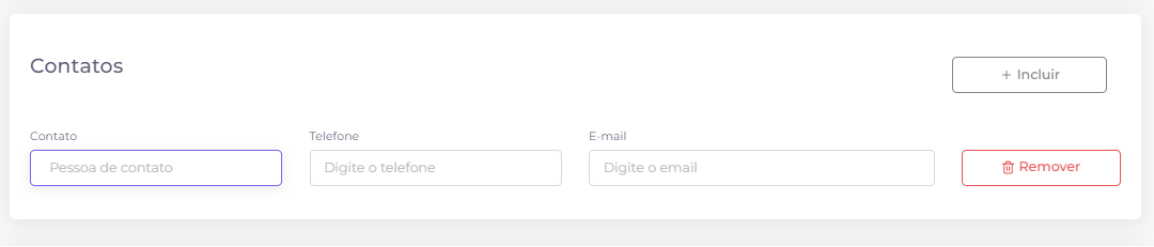

Figura 5.60: Formulário de inserção de contatos.

Para possibilitar a exibição e ocultação dinâmica das linhas de dados, foi necessário utilizar uma função em JavaScript, conforme mostrado na Figura 5.61. Esse trecho de código configura um `repetidor` que permite a criação dinâmica de elementos em uma página da WEB. A variável `qtd` é inicializada como 1 e representa a quantidade de elementos repetidos que já foram adicionados na página.

```
var qtd = 1;
var $repeater = $('.repeater').repeater({
    show: function() {
        $(this).slideDown();
        // Feather Icons
        if (feather) {
            feather.replace({
                width: 14,
                height: 14
            });
        }
        $(this).show(function() {
            qtd++;
            $(this).find('.telefone').addClass('telefone_' + qtd);
            new Cleave('.telefone_' + qtd, {
                numericOnly: true,
                blocks: [0, 2, 5, 4],
                delimiters: ["(", ") ", "-"]
            });
        });
    },
    hide: function(deleteElement) {
        if (confirm('Tem certeza que deseja excluir esse Contato?')) {
            $(this).slideUp(deleteElement);
        }
    }
});      You, há 9 meses • commit_cruds
```

Figura 5.61: Função JavaScript responsável por adicionar e remover elementos dinamicamente.

A função `show` é acionada quando um novo elemento é adicionado. Ela arrasta o elemento para baixo, adiciona os ícones relacionados e, em seguida, exibe o novo elemento. O contador `qtd` é incrementado e adicionado à classe CSS `telefone` com um sufixo `_qtd` para que cada elemento tenha uma classe única. Um novo objeto `Cleave` é criado para formatar o campo de telefone no formato `(99)9999-9999`. A função `hide` é acionada quando um elemento é removido e exibe uma mensagem de confirmação antes de deslizar o elemento para cima e excluí-lo.

Em seguida, no trecho de código mostrado na Figura 5.62, um objeto `repeater` é criado usando o seletor `.repeater`. Então, o método `setList` é chamado nesse objeto para definir a lista inicial de contatos do fornecedor. Essa lista é criada usando um laço de repetição `@foreach` em PHP, assim gerando uma coleção de contatos do fornecedor ou funcionário. Dentro do *loop* `@foreach`, os valores de cada campo do contato são definidos no objeto. Resumidamente, este trecho de código está sendo usado para criar um formulário de repetição para adicionar contatos de fornecedores e também preencher uma lista de contatos iniciais do fornecedor.

```
$repeater.setList([
    @foreach ($fornecedor->contatos as $contato)
        {
            'contato_nome': '{{ $contato->nome }}',
            'contato_telefone': '{{ $contato->telefone }}',
            'contato_email': '{{ $contato->email }}',
        },
    @endforeach
]);
```

Figura 5.62: Criação do objeto `repeater` e definição da lista inicial de contatos.

5.10 Inserção de Imagem

Para armazenar as imagens dos animais, foi criada uma coluna na tabela `animais` chamada `imagem_id`, conforme ilustrado na Figura 5.63. Essa coluna é do tipo inteiro e será utilizada para armazenar o ID da imagem associada a cada animal.

Por padrão, o valor 0 é inserido na coluna `imagem_id` para indicar que nenhuma imagem foi adicionada para o animal. Quando um registro é salvo, todas as consultas e referências à imagem específica são feitas através da coluna `imagem_id`, facilitando sua exibição ou inclusão em documentos quando necessário.

```
Joao Queiroz, mês passado | 2 authors (You and others) | 0 references | 0 implementations
class CreateAnimais extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    0 references | 0 overrides
    public function up()
    {
        Schema::create('animais', function (Blueprint $table) {
            $table->id();
            $table->integer('imagem_id')->default(0);
            $table->string('video')->nullable();
            $table->string('nome', 50);
            $table->string('sexo', 20);
```

Figura 5.63: Criação da tabela `animais`, feita na classe **CreateAnimais.php**.

Dentro do método `save` da classe **AnimalController.php**, foi implementado o código mostrado na Figura 5.64 para permitir que a imagem inserida pelo usuário seja salva. Nesse trecho de código, é recebido o arquivo de imagem via protocolo HTTP. Se a imagem foi de fato inserida pelo usuário, um nome de arquivo é gerado usando o ID do animal, os dados atuais e a extensão original do arquivo, e em seguida o arquivo é salvo no diretório `/arquivos/rebanhos/` usando

o método `move` da classe **UploadedFile** do Laravel. Por fim, a referência da imagem (caminho do arquivo) é armazenada no banco de dados, no campo `imagem` do registro do animal.

Dessa forma, quando um usuário envia uma imagem ao inserir um novo animal, a imagem é salva em um diretório e o caminho para a imagem é armazenado no banco de dados junto com os demais dados do registro do animal. Com isso, a imagem pode ser exibida na tela principal ou em qualquer outra tela por meio de seu caminho de acesso.

```php
if (isset($request->imagem)) {
    $data = Carbon::now();
    $path = '/arquivos/rebanhos/';
    $arquivo = $animal->id
        . $data->format("_Y-m-d-H-i-s")      You, há 7 meses
        . '.'
        . $request->imagem->getClientOriginalExtension()
    ;
    $request->imagem->move(public_path() . $path, $arquivo);
    $animal->imagem = "{$path}/{$arquivo}";
    $animal->save();
}
```

Figura 5.64: Condição para verificar a inserção da imagem no método `save` da classe **AnimalController.php**.

Para que o usuário possa inserir a imagem no banco de dados, o sistema tem no cadastro de animais um botão para que ele selecione a imagem salva em seu dispositivo. A Figura 5.65 mostra como é codificado o *front-end* para que este botão abra o explorador de arquivo da máquina do usuário e que ao selecionar uma imagem, ela aparece logo à cima do botão.

O bloco `<img>` exibe uma imagem selecionada, caso exista e o botão **Escolher Foto** abre o explorador de arquivos para que o usuário possa selecionar a imagem. Quando uma imagem é selecionada, a função `loadFile` é executada, assim carregando a imagem selecionada no campo `<img>` e então exibindo-a na página.

```
<div class="row">
    <div class="col-2">
        <div class="form-group">
            <label class="form-label" for="btnDestaque">
                Imagem de Destaque
                <img src="{{ isset($animal->imagem) ? asset($animal->imagem) : '' }}"
                    class="img-fluid" id="imagePreview" />
            </label>
            <button type="button" id="btnDestaque" class="btn btn-outline-secondary btn-block"
                onclick="$('#imagem').click();">
                <i data-feather='upload'></i>
                Escolher Foto      You, há 8 meses • animais_img
            </button>
            <input type="file" id="imagem" name="imagem" style="display: none;"
                accept="image/*" onchange="loadFile(event)" />
        </div>
    </div>
</div>
```

Figura 5.65: Implementação do botão de seleção de imagem no *front-end*.

A Figura 5.66 apresenta o *layout* visualizado pelo usuário na tela de cadastro de animais. Ao clicar no botão o sistema abre o explorador de arquivos da sua máquina e então permite que o mesmo escolha a imagem que deseja enviar.

Figura 5.66: *Layout* do botão **Escolher Foto** na tela de cadastro de animais.

Para exibir esta imagem ao usuário após ser salva, ela é inserida em uma tabela na tela principal do cadastro de animais, como mostrado na Figura 5.67. Esta imagem apresenta o resultado exibido usuário após registrar um novo animal no sistema, inserindo sua imagem, nome, *link* de vídeo, nome da fazenda, nome do lote do animal e seu status.

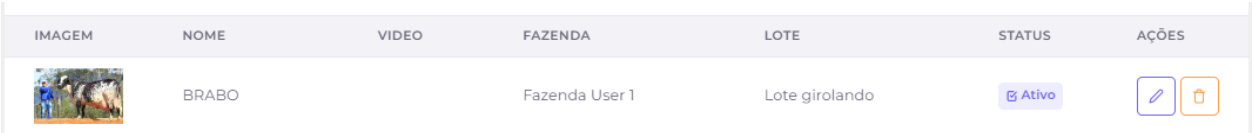

Figura 5.67: Tabela que contém os dados dos animais inseridos.

5.11 Exibir ou Ocultar Campos

No cadastro de animais, existem alguns campos que devem ser preenchidos, apenas quando em outros campos são inseridas determinadas respostas, sendo assim, necessário que estes campos sejam ocultos ao usuário, caso a resposta seja diferente da necessária. A Figura 5.68 apresenta estes campos.

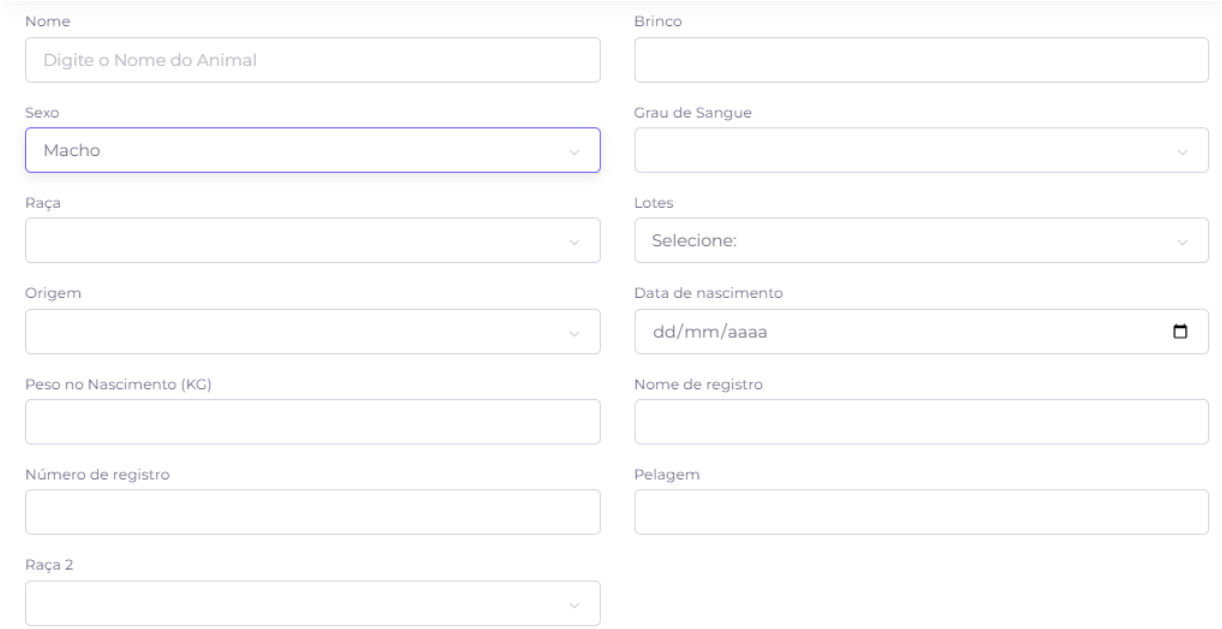

Figura 5.68: Formulário de cadastro de animais.

Observando o campo `sexo` no cadastro de animais, ao escolher a opção `macho`, nenhum novo campo será exibido ao usuário. No entanto, se a opção escolhida for `fêmea`, os novos campos **Número de Crias** e **"Parida?"** serão exibidos abaixo, como apresentado na Figura 5.69.

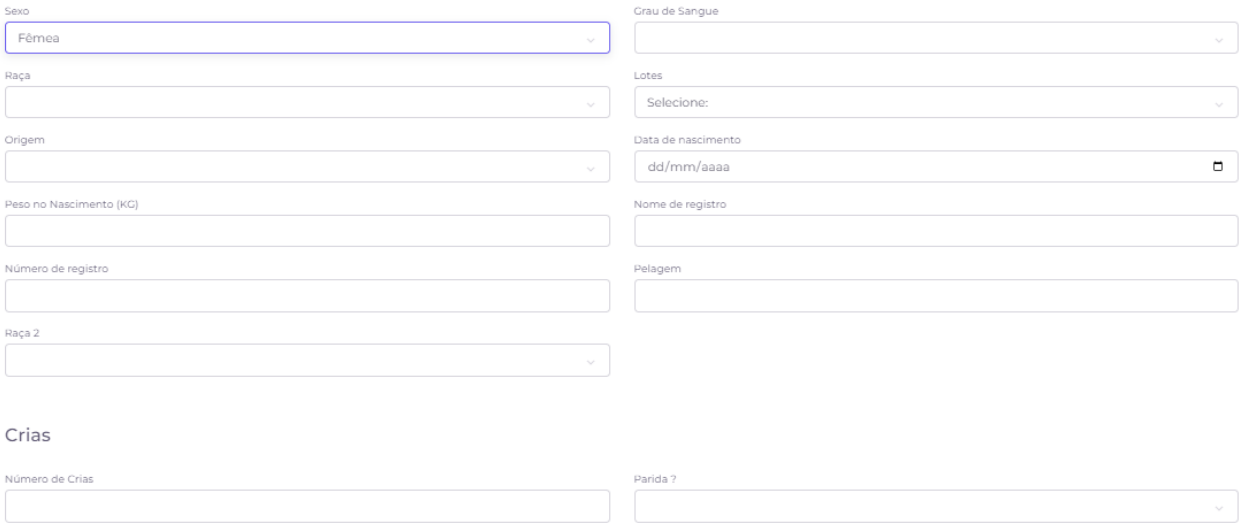

Figura 5.69: Novas opções de **Crias** após selecionar **Fêmea** no campo **Sexo**.

Para que seja possível a exibição dinâmica de campos na tela, é necessário definir regras e respostas para campos específicos, como `sexo` e `origem`, por exemplo. Essa funcionalidade é implementada utilizando a linguagem JavaScript, que manipula o surgimento de novos campos de acordo com as respostas fornecidas pelo usuário.

No trecho de código apresentado na Figura 5.70, é definida uma função em JavaScript que recebe o valor selecionado no campo `sexo` e realiza verificações condicionais para exibir ou ocultar a `div` correspondente. Caso o valor selecionado seja **FEMEA**, a `div` com o identificador `cria` é exibida na tela. Se o valor selecionado for **MACHO**, a `div` com o identificador `cria` fica oculta, ainda, caso não tenha sido selecionado nenhum valor, a `div` permanece oculta.

```
$("#sexo").change(function() {
    if ($("#sexo").val() === "FEMEA") {
        $('#cria').show();
    } else if ($("#sexo").val() === "MACHO") {
        $('#cria').hide();
    } else {
        $('#cria').hide();
    }
});
```

Figura 5.70: Função JavaScript responsável pelo revelar ou ocultar os campos.

5.12 Filtragem de Filiação

No processo de transferência embrionária, após a coleta e fecundação dos gametas (óvulos e espermatozóides) em laboratório, os embriões gerados são analisados e os de melhor

qualidade são transferidos para o útero da futura gestante (PROCRIAR, 2023). Para garantir o controle da origem dos embriões e da gestante, é necessário realizar a filtragem dos animais cadastrados por sexo, permitindo ao usuário selecionar o pai e a mãe do embrião em questão.

No método `index`, presente na classe **TeController.php**, carregam-se algumas relações da classe *model* **Tes.php**, usando o método `with`, que carrega inicialmente as relações, evitando que o Laravel faça uma consulta ao banco de dados para cada registro relacionado a ser carregado.

Mais especificamente, são carregadas as relações `user` (com os atributos `id` e `name`), `animal` (com os atributos `id` e `nome`) e `animais` (com os atributos `id` e `nome`) da classe model **Tes.php**. Além disso, paginam-se os resultados com a quantidade de registros por página definida na configuração da aplicação. Feito isso, os objetos são enviados para a *view*, por meio do método `compact`, e então é retornado por meio de rotas. O método `index`, que possui um trecho mostrado na Figura5.71 é utilizado na primeira rota do cadastro de protocolo de transferência embrionária, desta forma, essa função retorna os filtros a serem exibidos na tela inicial.

```
$tes = $tes
->with('user:id,name')
->with('animal:id,nome', 'animais:id,nome')
->paginate(config('app.paginate'));        You, há 2 semanas •

//caminho para salvar o objeto no banco
//errar aqui, gera um 404 !
$dataView = compact('breadcrumbs', 'request', 'tes');
return view('modules/protocolo/te/index', $dataView);
```

Figura 5.71: Consulta realizada no método `index` para que seja possível a utilização dos filtros.

Para filtrar o campo `sexo` pelos valores **MACHO** ou **FÊMEA** e apresentar essa opção ao usuário durante o cadastro, foi necessário realizar duas consultas na função `create`, presente na classe **TeController.php**, e então cada consulta foi inserida em um objeto. A Figura 5.72 mostra como foi realizada esta consulta.

```php
public function create($id)
{
    $breadcrumbs = $this->breadcrumbs;
    $te = $this->model::findOrNew($id);
    $users = User::select('id', 'name')->orderBy('name')->get();
    if (Auth::user()->role_id == 1) {
        $animais = Animal::select('id', 'nome')->where('sexo', '=', 'MACHO')->orderBy('nome')->get();
        $animais2 = Animal::select('id', 'nome')->where('sexo', '=', 'FEMEA')->orderBy('nome')->get();
    } else {
        $user_id = Auth::id();
        $animais = Animal::select('id', 'nome')
            ->where(function ($query) use ($user_id) {
                $query->where('user_id', $user_id)
                    ->orWhereNull('user_id');
            })
            ->where('sexo', '=', 'MACHO')
            ->orderBy('nome')
            ->get();

        $animais2 = Animal::select('id', 'nome')
            ->where(function ($query) use ($user_id) {
                $query->where('user_id', $user_id)
                    ->orWhereNull('user_id');
            })
            ->where('sexo', '=', 'FEMEA')
            ->orderBy('nome')
            ->get();
    }
    $dataView = compact('breadcrumbs', 'te', 'animais', 'animais2', 'users');
    return view('modules/protocolo/te/create', $dataView);
```

Figura 5.72: Função `create` existente na classe **TeController.php**.

Dependendo do papel do usuário que está acessando a visualização, a lista de animais será filtrada para exibir apenas os animais que pertencem a ele ou exibir todos os animais, se o usuário for um administrador. Por fim, é criado um *array* chamado de `$dataView` que contém todas as variáveis definidas anteriormente e é retornado a *view* `modules/protocolo/te/create` com este *array* de dados.

Os dados são exibidos para o usuário, permitindo que ele visualize quem são a mãe e o pai do embrião. Para isso, a classe recebe o objeto `TE` oriundo da classe **TeController.php**, percorre um *array* para garantir que todos os registros da tabela pertencentes ao usuário sejam exibidos. Para os campos `Mãe` e `Pai` da Figura 5.73, as consultas realizadas dentro dos objetos `animal` e `animais` foram recebidas pela *view* a partir da *Controller*. Cada um desses dados são exibidos por nome e separados por sexo, classificando-os assim em `Mãe` e `Pai`.

```
<div class="table-responsive">
    <table class="table table-hover">
        <thead>
            <tr>
                <th>Nome</th>
                <th>Descrição</th>
                <th>Mãe</th>
                <th>Pai</th>
                <th style="width: 5%;">Ações</th>
            </tr>
        </thead>
        <tbody>
            @foreach ($tes as $te)
                <tr>
                    <td>
                        {{ $te->nome }}
                    </td>
                    <td>
                        {{ $te->desc }}
                    </td>
                    <td>
                        {{ $te->animal->nome ?? '' }}
                    </td>
                    <td>
                        {{ $te->animais->nome ?? '' }}
                    </td>
```

Figura 5.73: Arquivo *Blade* responsável por apresentar os campos filtrados na *view*.

Para filtrar e exibir os dados em dois *selects* distintos no cadastro, é necessário realizar o filtro novamente nesta lista e salvar os dados nas colunas `animal_id` e `animais_id`. É necessário novamente percorrer a tabela, pelos objetos com as consultas vindos da *controller*. Na Figura 5.74 é mostrado como é feito o filtro dos dados, onde os *selects* são utilizados para que o usuário selecione quem é a mãe e o pai naquele protocolo de transferência embrionária.

```
<div class="col-md-6 col-12">
    <div class="form-group">
        <label class="form-label" for="animais_id">Pai</label>
        <select name="animais_id" id="animais_id" class="form-control" required>
            <option value="">Selecione:</option>
            @foreach ($animais as $animal)
                <option value="{{ $animal->id }}"
                    {{ $animal->id == $te->animais_id ? 'selected="selected"' : '' }}>
                    {{ $animal->nome }} </option>
            @endforeach
        </select>
    </div>
</div>
<div class="col-md-6 col-12">
    <div class="form-group">
        <label class="form-label" for="animal_id">Mãe</label>
        <select name="animal_id" id="animal_id" class="form-control" required>
            <option value="">Selecione:</option>
            @foreach ($animais2 as $animal)
                <option value="{{ $animal->id }}"
                    {{ $animal->id == $te->animal_id ? 'selected="selected"' : '' }}>
                    {{ $animal->nome }} </option>
            @endforeach
        </select>
    </div>
</div>
</div>
```

Figura 5.74: Arquivo *Blade* responsável por filtrar os campos `Pai` e `Mãe` e exibir os *selects*.

Na tela principal do cadastro de protocolo de transferência de embrião, os dados sobre o sexo são apresentados filtrados por macho ou fêmea e classificados como `Mãe` e `Pai`, respectivamente. Essa classificação é feita com base nas consultas realizadas aos objetos `animal` e `animais` na *Controller*. Cada animal é exibido pelo seu nome e separado pelo seu sexo, permitindo ao usuário visualizar facilmente quem são os pais do embrião.

A Figura 5.75 apresenta a *view* **Protocolo TE**, que consiste em um protocolo de transferência embrionária desenvolvida com o objetivo de aumentar as chances de sucesso na implantação e gestação do embrião, proporcionando a melhor condição gestacional possível. A transferência embrionária é o procedimento pelo qual o embrião é transferido para o útero que irá apoiar o seu desenvolvimento. A estrutura da visualização inclui o nome do protocolo, a descrição do protocolo, a mãe do embrião e o pai do embrião. (PROCRIAR, 2023).

Figura 5.75: *View* apresentada ao usuário, contendo os dados sobre Protocolos de Transferência Embrionária.

Na tela de cadastro, os dados são filtrados em *selects* e apresentados como na Figura 5.76. Esta Figura apresenta a tela de cadastro de um novo protocolo de transferência embrionária, onde o usuário irá inserir as informações sobre o embrião e depois o sistema irá armazenar os dados e então exibi-los.

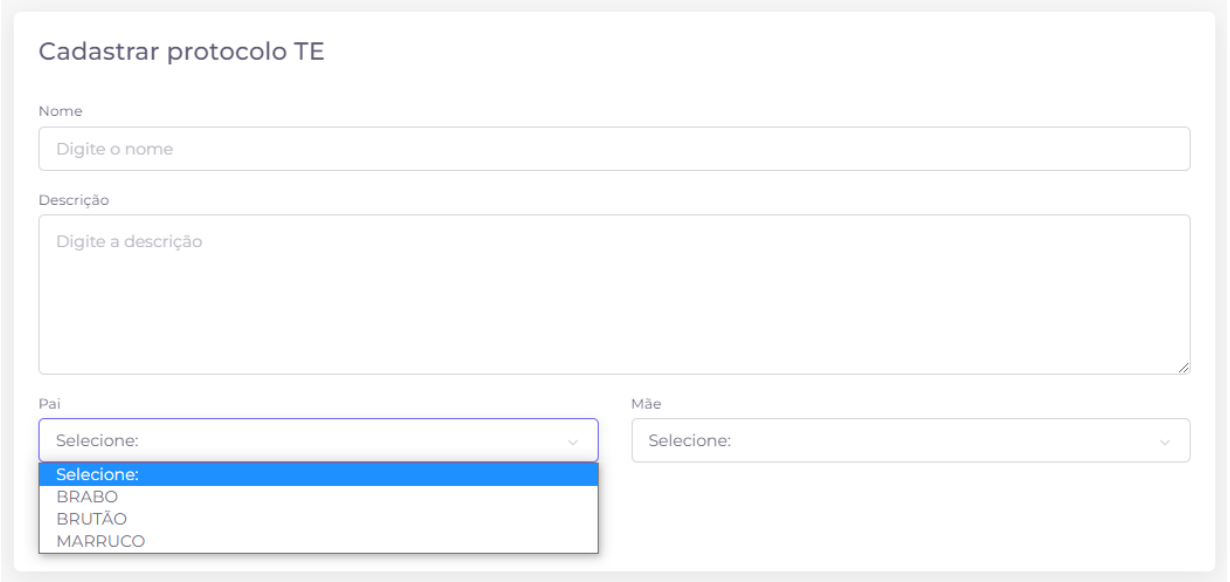

Figura 5.76: Filtro `Pai` sendo mostrado na *view* de cadastro de protocolo de transferência embrionária.

CONCLUSÃO E TRABALHOS FUTUROS

Para a escrita deste trabalho, foi levantado um estudo acerca de algumas das principais necessidades que produtores de leite enfrentam diariamente e quais as funcionalidades que poderiam ser implementadas para tratar estas mesmas necessidades. Pode-se observar neste estudo que para a construção de um sistema bem estruturado, com múltiplas funções e otimizado é necessário o uso de diferentes tecnologias e metodologias, assim garantindo alta escalabilidade e fácil manutenção.

Utilizando *Web Services* para automatizar funções básicas do sistema, tornou-se mais simples o desenvolvimento da aplicação. Para este projeto, o padrão arquitetural MVC mostrou-se o mais adequado, possibilitando a reutilização de uma única *Model* em múltiplas *Controllers* e *Views*, o que proporciona maior agilidade na criação das funcionalidades propostas. O *Framework* Laravel traz uma nova abordagem à linguagem PHP, utilizando vários *containers*, como o *Composer*, para melhor gerenciamento de dependências, o *Artisan* para auxiliar em toda a interface de linha de comando do Laravel em si e o *Blade* para criação de *Views* com seu mecanismo próprio, ampliando todas as funcionalidades do sistema de maneira eficiente e elegante.

A análise de requisitos foi essencial para o sucesso da aplicação, pois com ela foi possível organizar todas as informações coletadas, identificar requisitos funcionais e não funcionais e elaborar diagramas que auxiliam na criação do banco de dados. Os resultados alcançados comprovam a viabilidade e efetividade da aplicação, e demonstram que essa análise foi fundamental para garantir a implementação dos requisitos definidos. Ao concluir este projeto, foi evidente o impacto do Laravel no mercado de trabalho do PHP. A experiência de trabalhar com um software baseado em um *Framework* bem estruturado revelou-se de grande importância para a equipe.

Algumas funcionalidades foram identificadas como passíveis de melhorias em trabalhos futuros, visando melhores resultados e maior aceitação pelos usuários. Entre elas, destaca-se a disponibilidade de um aplicativo *mobile* para ser utilizado com maior facilidade por agrônomos durante o período de trabalho, buscando oferecer maior comodidade e velocidade na geração de dados. Por fim, sugere-se a inclusão de novas funcionalidades na aplicação atual, como a opção de criação de gado de corte e suas respectivas funcionalidades.

REFERÊNCIAS BIBLIOGRÁFICAS

AGROFY. *Quem são os maiores produtores de leite do mundo*. 2022. Disponível em: <https://news.agrofy.com.br/noticia/201002/quem-sao-os-maiores-produtores-leite-do-mundo>. Acesso em: 15 de novembro de 2022.

AGROPECUARIA, E. B. de P. Produção de leite no brasil: números que impressionam. *Embrapa*, 2021. Disponível em: <https://www.embrapa.br/busca-de-noticias/-/noticia/63618684/producao-de-leite-no-brasil-numeros-que-impressionam>.

ALMEIDA, M. B. Uma introdução ao xml, sua utilização na internet e alguns conceitos complementares. *Ciência da informação*, SciELO Brasil, v. 31, p. 5–13, 2002.

AMAZON. *Microsserviços*. 2022. Disponível em: <https://aws.amazon.com/pt/?nc2=h_lg>. Acesso em: 15 de novembro 2022.

AMAZON AWS. *O que é API RESTful?* 2022. Disponível em: <https://aws.amazon.com/pt/what-is/restful-api>. Acesso em: 31 de outubro 2022.

BOOCH, G. *UML: guia do usuário*. [S.l.]: Elsevier Brasil, 2006.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The Unified Modeling Language User Guide*. [S.l.]: Addison-Wesley, 2000.

BRUNA B. BARRO. *O Que São Frameworks e Quais os Mais Utilizados*. 2022. Disponível em: <https://www.hostinger.com.br/tutoriais/frameworks>. Acesso em: 26 de outubro 2022.

BUCANEK, J. Model-view-controller pattern. *Learn Objective-C for Java Developers*, Springer, p. 353–402, 2009.

CAIO, GABRIEL. *Pecuária Leiteira: perspectivas e desafios*. 2021. Disponível em: <https://www.milkpoint.com.br/artigos/producao-de-leite/pecuaria-leiteira-perspectivas-e-desafios-225972/>. Acesso em: 16 de novembro 2022.

CAMPOS, A. T. D.; FERREIRA, A. D. M.; LEITE, E. G. D. *Composição do rebanho e sua importância no manejo*. [S.l.]: Embrapa Gado de Leite, 2001.

CANGUÇU, RAPHAEL. *O que são Requisitos Funcionais e Requisitos Não Funcionais?* 2021. Disponível em: <https://codificar.com.br/requisitos-funcionais-nao-funcionais/>. Acesso em: 22 de novembro 2022.

CARLOS ESTRELLA. *Composer.json*. 2022. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-php-guia-basico>. Acesso em: 14 de Novembro 2022.

CLARA FABRO. *O que é API e para que serve?* 2020. Disponível em: <https://www.techtudo.com.br/listas/2020/06/o-que-e-api-e-para-que-serve-cinco-perguntas-e-respostas.ghtml>. Acesso em: 16 de Novembro 2022.

CONTE, T.; MENDES, E.; TRAVASSOS, G. H. Processos de desenvolvimento para aplicações web: Uma revisão sistemática. In: *Proceedings of the 11th Brazilian Symposium on Multimedia and Web (WebMedia 2005)*. [S.l.: s.n.], 2005. v. 1, p. 107–116.

COOPERATIVA. *A importância econômica da produção leiteira para o Brasil*. 2018. Disponível em: <https://cooperativa.coop.br/a-importancia-economica-da-producao-leiteira-para-o-brasil/#:~:text=A%20produ%C3%A7%C3%A3o%20leiteira%20desempenha%20um,manuten%C3%A7%C3%A3o%20de%20uma%20vida%20saud%C3%A1vel.>

COSTA, C. J. *Desenvolvimento para web*. [S.l.]: ITML press/Lusocredito, 2007.

CURBERA, F.; NAGY, W.; WEERAWARANA, S. Web services: Why and how. In: SN. *Workshop on Object-Oriented Web Services-OOPSLA*. [S.l.], 2001. v. 2001.

DEVMEDIA. *Laravel Tutorial*. 2022. Disponível em: <https://www.devmedia.com.br/laravel-tutorial/33173>. Acesso em: 26 de outubro 2022.

DHYOGO ALMEIDA. *PSR — Padrões necessários para ter um código de qualidade*. 2022. Disponível em: <https://medium.com/@dhyogoalmeida/psr-padr%C3%B5es-necess%C3%A1rios-para-ter-um-c%C3%B3digo-de-qualidade-92b5d81f70fe>. Acesso em: 26 de outubro 2022.

DIARURAL. Saiba qual é a importância da tecnologia para a pecuária leiteira. *Diarural*, 2021. Disponível em: <https://diarural.com.br/saiba-qual-e-a-importancia-da-tecnologia-para-a-pecuaria-leiteira/>.

FAO (Organização das Nações Unidas para Agricultura e Alimentos). 2022. Disponível em: <http://www.fao.org/faostat/pt/#data/QL>. Acesso em: 15 de novembro de 2022.

FIGUEIRA, A.; SOUKI, G.; ZAMBALDE, A.; ANTONIALLI, L. Impactos da tecnologia da informação na dimensão competitiva de agentes da cadeia produtiva do leite. In: *42º Congresso da Sociedade Brasileira de Economia, Administração e Sociologia Rural*. [S.l.: s.n.], 2004. v. 42.

FIGUEIREDO, E. *Padrões Arquiteturais*. [S.l.]: Retrieved, 2019.

GOMES, A. M.; GARCIA, A. C. B. *Engenharia de Software: Uma Abordagem Prática e Didática*. [S.l.]: Elsevier, 2007.

GOOGLE. *Como usar os gráficos do Google.* 2022. Disponível em: <https://developers.google.com/chart/interactive/docs?hl=pt-br>. Acesso em: 27 de abril 2023.

GUILHERME MANZANO. *Padrões e estilos de arquitetura de software*. 2020. Disponível em: <https://dev.to/guilhermemanzano/padroes-e-estilos-de-arquitetura-de-software-1498>. Acesso em: 15 de novembro 2022.

HEUSER, C. A. *Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS*. [S.l.]: Bookman Editora, 2009.

LARAVEL. *MeetLaravel*. 2022. Disponível em: <https://laravel.com/docs/9.x>. Acesso em: 26 de outubro 2022.

LARAVEL. *Laravel - Hashing*. 2023. Disponível em: <https://laravel.com/docs/5.0/hashing>. Acesso em: 16 de maio de 2023.

LARISSA GASPAR. *Protocolo HTTP: entenda o que é e para que serve!* 2022. Disponível em: <https://www.hostgator.com.br/blog/o-que-e-protocolo-http/>. Acesso em: 09 de Novembro 2022.

LUCIANO, J.; ALVES, W. J. B. Padrão de arquitetura mvc: Model-view-controller. *EPeQ Fafibe*, v. 1, n. 3a, p. 102–107, 2017.

MACHADO, J. G. d. C. F.; NANTES, J. F. D. Adoção da tecnologia da informação em organizações rurais: o caso da pecuária de corte. *Gestão & Produção*, SciELO Brasil, v. 18, p. 555–570, 2011.

MARCIO. *Padrão MVC - Java Magazine*. 2011. Disponível em: <https://www.devmedia.com.br/padrao-mvc-java-magazine/21995#4>. Acesso em: 27 de Outubro 2022.

PEACHPIE. *Composer.json*. 2022. Disponível em: <https://docs.peachpie.io/php/composer-json/>. Acesso em: 30 de Outubro 2022.

PROCRIAR. 7 coisas que você precisa saber sobre transferência embrionária. *Procriar Blog*, 2023. Disponível em: <https://www.procriar.com.br/blogprocriar/7-coisas-que-voce-precisa-saber-sobre-transferencia-embrionaria/>.

SHAHZED AHMED. *Como usar o Composer no Laravel 8*. 2021. Disponível em: <https://www.cloudways.com/blog/composer-with-laravel/>. Acesso em: 29 de Outubro 2022.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Database system concepts. McGraw-Hill Education, 2011.

SOMMERVILLE, I. *Engenharia de Software*. [S.l.]: Addison Wesley, 2003.

TAURION, C. *Cloud computing-computação em nuvem*. [S.l.]: Brasport, 2009.

TUNG, N. H. *Planejamento e controle financeiro das empresas agropecuárias*. [S.l.]: Ed. Universidade-Empresa, 1990.

W3TECHS. *Usage statistics of server-side programming languages for websites*. 2023. Disponível em: <https://w3techs.com/technologies/overview/programming_language/all>. Acesso em: 14 de maio de 2023.