RuralEasy: Uma solução para gestão de agronegócio baseada em PWA

Gabriel Henrique Almeida Ludwig[1], **Matheus de Andrade**[1], **Wilson Castello Branco Neto**[1]

[1]Instituto Federal Santa Catarina - Câmpus Lages (IFSC)
Rua Heitor Vilas Lobos, 225 - 88.506-400 - Lages - SC - Brasil

gabriel.h1997@aluno.ifsc.edu.br, matheus.a3@aluno.ifsc.edu.br

wilson.castello@ifsc.edu.br

Abstract. *Agribusiness is an increasingly important activity for the Brazilian economy, generating employment and Gross Domestic Product (GDP) growth. The adoption of technologies becomes a necessity for good agribusiness management in the current scenario, especially for small producers. This article addresses the development of the RuralEasy application, a PWA system for managing beef cattle properties aimed at small producers. Its implementation considered the needs of producers in the region and recommendations from SENAR. At the end of development, verification and validation tests were carried out, where it was found that the application met all the requirements defined for a management system.*

Resumo. *O agronegócio é uma atividade cada vez mais importante para a economia brasileira, gerando emprego e crescimento do PIB. A adoção de tecnologias torna-se uma necessidade para uma boa gestão do agronegócio no cenário atual, principalmente para os pequenos produtores. Este artigo aborda o desenvolvimento da aplicação RuralEasy, um sistema PWA para gestão de propriedades de gado de corte voltado ao pequeno produtor. Sua implementação considerou as necessidades dos produtores da região e recomendações do SENAR. No final do desenvolvimento foram realizados testes de verificação e validação, onde constatou-se que a aplicação satisfazia todos os requisitos definidos para um sistema de gestão.*

1. Introdução

A agroindústria é o conjunto de atividades referentes à transformação de insumos utilizados pela produção e transformação de matérias-primas provenientes da agropecuária em produtos manufaturados (Fundação Getúlio Vargas, 2019). Dentro deste cenário, o agronegócio engloba todas as atividades do setor agroindustrial. Esse conjunto de atividades econômicas envolve desde os processos necessários para a produção (sementes, defensivos, máquinas e implementos), as funções realizadas durante a produção (agropecuária básica ou primária), e após a produção (indústria e serviços), envolvendo ainda o processamento, a distribuição e o consumo (Sociedade Nacional da Agricultura, 2020).

O agronegócio é considerado um ator importante no crescimento da economia brasileira. Em 2020, os bens e serviços do agronegócio somaram R$ 1,98 trilhão ou 27%

do PIB (Produto Interno Bruto) brasileiro. O ramo agrícola corresponde à 70% desse valor (R$ 1,38 trilhão) e o segmento da pecuária representa 30% (R$ 602,3 bilhões). De acordo com dados de 2015 coletados pela Pesquisa Nacional por Amostra de Domicílios (PNAD), 32,3% (30,5 milhões) do total de 94,4 milhões de trabalhadores brasileiros estavam vinculados ao agronegócio (Confederação da Agricultura e Pecuária, 2021).

No segmento da pecuária, o setor de carnes é uma das principais vertentes do agronegócio brasileiro, representando 31% do PIB em 2017 e gerando ganhos de R$ 433 bilhões. A cadeia produtiva é formada por três principais seções: carne bovina, suína e de frango. Mesmo com diferenças, todas essas cadeias possuem uma expressiva representatividade no mercado interno e contribuem para o comércio internacional. O setor de carnes é fundamental para a economia nacional, gerando emprego, renda e contribuindo para o superavit da balança comercial e formação de reservas internacionais (Fundação Getúlio Vargas, 2018).

Segundo dados do Instituto Brasileiro de Geografia e Estatística (IBGE), o ano de 2021 foi bastante expressivo para o setor pecuário, mostrando que essa atividade possui tamanha magnitude no Brasil. Dentre os diversos tipos de criações, o Brasil registrou um total de 224.602.112 cabeças em rebanhos bovinos (IBGE, 2021). Por números expressivos como estes, é notável que o mercado está cada vez mais competitivo. Seja por cooperativas ou associações de produtores, é natural que o mercado exija do produtor uma maior qualidade geral em sua produção. Tal qualidade pode envolver aumento na produção, redução dos custos e produção de qualidade, agregando valor aos produtos (Embrapa, 2022).

Em relação à boa gestão e redução de custos, se faz necessária a utilização de tecnologias que venham a ajudar o pecuarista, pois a administração da maioria dos criadores de gado ainda se restringe a registros manuais em papéis ou nenhuma forma de monitoramento. Com o incremento de gastos e a exigência por mercadorias de alto nível, o pecuarista necessita de gerenciamento mais aprimorado do que possui atualmente (Centro de Inteligência da Carne Bovina, 2020).

Costa et al. (2014) notaram em sua pesquisa com empresas da cadeia da pecuária de corte no estado de Goiás que a utilização de Tecnologias da Informação Móveis e Sem Fio (TIMS), como a identificação por rádio frequência (*Radio Frequency Identification* - RFID), trouxeram ganhos significativos do ponto de vista gerencial, como o controle de estoque e da produção, o controle de despesas individuais, o controle de desempenho dos funcionários e a gestão dos rendimentos e perdas por animal.

Demonstrada a importância do Agronegócio para o Brasil, é possível reafirmar a sua relevância para a economia nacional. Graças a trajetória acadêmica e profissional dos autores do trabalho, foi possível perceber oportunidades de melhoria dentro do próprio Agronegócio, confirmadas pelo contato direto com pecuaristas da região com interesse em soluções que tragam modernização para a gestão pecuária, principalmente voltada aos pequenos produtores. A demanda por uma solução capaz de agilizar a rotina do pecuarista foi o que conduziu a escolha deste tema para o Trabalho de Conclusão de Curso.

Dentro deste contexto, do uso de tecnologias para facilitar a gestão de atividades pecuárias, este trabalho tem como objetivo desenvolver um sistema *web* para gestão pecuária gratuito e que atenda às necessidades dos pequenos produtores, visando melhorar

tal processo de gerência. Para que o objetivo do trabalho seja alcançado, foram definidos os seguintes objetivos específicos:

- Estudar as tecnologias *Progressive Web Application* (PWA), *Vue, Python* e *Django* para realização do projeto;
- Modelar e implementar uma PWA para o front-end;
- Modelar e implementar uma API REST para o back-end da aplicação;
- Realizar o *Deploy* e os testes da aplicação desenvolvida.

A metodologia utilizada neste trabalho é a de pesquisa aplicada, porque a intenção é gerar conhecimentos para aplicações práticas e uma solução para um problema específico. Será realizada uma abordagem de forma qualitativa, já que não será necessário a utilização de análise estatística. Relacionado aos objetivos, a pesquisa será exploratória, visando ampliar o conhecimento de algo específico (Silva e Menezes, 2005). Do ponto de vista dos procedimentos técnicos, será efetivada a pesquisa bibliográfica, embasando a pesquisa em artigos com ideias similares às do trabalho.

O trabalho foi dividido em quatro etapas. A primeira etapa consiste em um estudo acerca da utilização dos *frameworks* e linguagens *Vue, Python* e *Django*, que permitirão o desenvolvimento de uma aplicação *PWA*, que são aplicações construídas utilizando tecnologias *web* e podem ser instaladas e utilizadas em dispositivos móveis, utilizando o mesmo código base (Microsoft, 2023). A segunda etapa será a construção da *PWA* do *front-end* utilizando o *framework Vue*. A terceira etapa consiste na modelagem e implementação de uma API REST para o *back-end* da aplicação, utilizando a linguagem *Python* e o *framework Django*. A quarta etapa consiste na realização do *deploy* da aplicação. Além disso, serão realizados os testes e a avaliação da aplicação, na qual serão verificados se todos os objetivos propostos foram atingidos.

Além da seção introdutória, este trabalho possui mais quatro seções. A seção 2 trata do referencial teórico, mostrando e descrevendo os assuntos essenciais ao entendimento deste artigo como um todo, assim como alguns trabalhos semelhantes. A seção 3 apresenta o desenvolvimento realizado, expondo as atividades desenvolvidas e os sistemas resultantes. A seção 4 apresenta as considerações finais.

2. Referencial Teórico

Esta seção aborda os trabalhos similares, assim como as ferramentas utilizadas para a construção do sistema proposto. A seção 2.1 expõe os trabalhos relacionados pesquisados. A seção 2.2 apresenta as informações sobre as tecnologias escolhidas para a formação da aplicação.

2.1. Trabalhos Similares

Esta seção apresenta seis sistemas com objetivos análogos ao apresentado neste artigo, com o intuito de verificar funcionalidades relevantes para a utilização neste trabalho. Para a realização das buscas foram utilizadas plataformas como o *Google* e *Google Scholar*, utilizando o seguinte conjunto de palavras “Sistema de gestão pecuária”. Durante a busca, foram encontrados 12 trabalhos e selecionados dez trabalhos dos quais, após leitura e análise, seis foram escolhidos para um estudo mais completo. Dentre os seis selecionados apenas dois são aplicativos pagos, o Jetbov e o iRancho.

Os critérios utilizados para a seleção dos trabalhos foram a presença das funções de cadastro de animais, histórico de pesagens, controle de vacinas e vermifugações, rastreio de produtos utilizados, módulo de gestão financeira, plataforma de compatibilidade e principalmente a função de leitura por RFID, função de vital importância para a fácil identificação de um animal que geralmente não fica parado durante o manejo. Com o uso de tecnologias RFID para a leitura do número de identificação é necessário apenas passar o leitor perto da orelha do animal. Além disso, também foram utilizadas como métricas de avaliação a confiabilidade do fabricante do *software* e dos autores dos trabalhos, a base bibliográfica dos artigos e também a relevância dos títulos e resumos.

2.1.1. iRancho

O aplicativo iRancho tem como principal objetivo simplificar o registro de todas as atividades do pecuarista, tornando os lançamentos de informações mais práticos e fáceis de serem realizados através de um sistema *mobile* e *web*. Como principais funcionalidades destacam-se o cadastro e controle de animais. Várias informações importantes sobre o animal como peso, idade, raça, sexo, vacinações, entre outros dados relevantes podem ser registrados no sistema. O *app mobile* funciona como um complemento ao sistema web. O sistema *web* possui diversas outras funcionalidades, tais como gestão financeira, controle do manejo, histórico de exames, controle de insumos e nutrição, controle da reprodução, relatórios e diversas outras funcionalidades (iRancho, 2023).

Dentre as funcionalidades já mencionadas, a parte de gestão financeira destaca-se por possuir indicativos de receitas e despesas apresentados de forma gráfica, auxiliando o usuário na compreensão dos dados. Além disso, o sistema *web* ainda conta com relatórios estratégicos que contribuem nas tomadas de decisões dos produtores. A interface de ambos os sistemas é completa e fácil de ser entendida, possuindo um design amigável ao usuário.

A empresa responsável oferece uma versão de teste limitada para a apresentação do sistema, mas o iRancho não possui uma versão gratuita e o seu valor é vendido mediante orçamento combinado com o cliente. Na análise deste trabalho, foi levado em consideração o sistema vendido no mercado em sua versão completa conforme as funcionalidades ofertadas no portal da aplicação.

Em geral, o iRancho oferece várias vantagens para o pecuarista. Coletar e agrupar dados é algo importante para um sistema de gestão, mas interpretar tais dados e transformá-los em informações é um diferencial importante para o mercado, e esta é uma característica significativa presente no sistema.

2.1.2. JetBov

O JetBov é uma aplicação *mobile* para *Android* e *iOS* destinada à gestão de gado de corte que foca nas etapas de cria (ciclo reprodutivo, seleção para descarte e índices zootécnicos), recria (acelerar o desempenho, coleta de informações para tomada de decisão) e engorda (controle de custos na nutrição, no pasto e confinamento) dos bovinos. O JetBov permite a coleta de dados para o manejo de campo e também para o manejo

do curral, disponibilizando funções para a separação de animais para venda, descarte e outros procedimentos. Na área financeira, o aplicativo também fornece opções para entrada de notas fiscais, supervisão de atividades e ainda disponibiliza os dados da fazenda de forma visual para tomadas de decisões. Além disso, o usuário pode cadastrar novos animais e registrar informações como a pesagem, o apontamento nutricional, vacinações, vermifugações e inseminações (Jetbov, 2023).

Assim como o iRancho, o Jetbov não disponibiliza uma versão gratuita mas permite uma demonstração do aplicativo através de agendamento prévio com a empresa proprietária. Os planos e preços do sistema variam conforme a extensão das atividades realizadas pelo produtor, permitindo também um orçamento customizado para grandes e médios produtores. Para este trabalho, o Jetbov foi analisado em sua versão vendida para o mercado.

Os animais cadastrados possuem uma ficha detalhada para visualização de informações, como o cálculo GMD (ganho médio diário de peso) e os históricos sanitários e reprodutivos de forma gráfica. O software também permite a edição dos dados dos bovinos e a adição de comentários individuais. O grande diferencial do Jetbov é a possibilidade de integração com dispositivos *Bluetooth*, como equipamentos de pesagem e sensores de leitura RFID, previamente homologados.

2.1.3. DSGB

O trabalho desenvolvido por Rauta (2016), consiste em um sistema para gerenciamento bovino que tem como principais funcionalidades o cadastro de animais, registro de visitas veterinárias e também o registro de sanidade do animal. Tais funcionalidades atuam sobre um sistema *Android* (4.4) e possuem uma integração a um sistema web, através de sincronização *webservice*, permitindo a visualização dos dados e geração de relatórios.

Um destaque para o projeto é a possibilidade da utilização offline, em contrapartida seu primeiro login deve ser realizado obrigatoriamente em rede. Além disso, o sistema ainda utiliza uma versão antiga do *Android*, o que pode ser um problema tratando-se de questões de vulnerabilidades de segurança, por exemplo.

2.1.4. RebanhoApp

Medeiros (2022), assim como o já citado Jetbov, realizou o desenvolvimento de um sistema voltado a criação de gado de corte, o RebanhoApp. Entre as funcionalidades mais comuns como permitir o cadastro de animais, gerenciamento de vendas, gerenciamento de despesas e o cadastro de vacinas, o software oferece um diferencial com a sua função de cálculo do preço dos animais conforme a cotação do Arroba, a unidade de medida para definição do preço do bovino baseada em seu peso.

O software foi desenvolvido para plataforma *Android*, utilizando Java. Suas interfaces são objetivas e claras ao usuário. A função de cálculo do preço conforme cotação do Arroba foi implementada utilizando um *webservice* próprio em *PHP*, porém a função requer que o usuário atualize manualmente o valor. Por ter este trabalho manual, a função torna-se contraintuitiva ao usuário por conta da necessidade diária de atualização.

2.1.5. DeskPec

No trabalho realizado por Rodrigues et al. (2021), foi desenvolvida uma aplicação *desktop* em *Java* com foco no cadastro de animais e no controle de vacinas. O software oferece a possibilidade de registro para bovinos e também para suínos, possuindo uma interface em modo resumo que apresenta a quantidade atual de animais na fazenda.

Não há funcionalidades extras em comparação aos outros aplicativos mencionados, o software desenvolvido é objetivo e simples, com uma interface fácil de usar e funções essenciais ao ambiente do pecuarista. Além disso, o sistema é *offline*, podendo ser um diferencial para pequenas fazendas que não possuem acesso a internet.

2.1.6. Sistema de gestão Pecuária Offline (SGPO)

O software desenvolvido por Carvalho (2019), teve como principal problemática a ser resolvida a questão sobre o acesso a internet em meios rurais, o qual é escasso. Além de ser um sistema *offline*, o mesmo possui como principais funcionalidades o cadastro de animais, registro do fluxo de caixa, registro de funcionários e parceiros, manejos, registro de compra e venda de rebanho, pesagem, manejos sanitários e a possibilidade de registro do ciclo reprodutivo.

O aplicativo foi desenvolvido em *Java* baseado em plataformas *web*. O que destaca esse software é a utilização do controle de fluxo de caixa, atualizado diariamente por meio de entradas e saídas. Com esta funcionalidade, o usuário pode consultar as receitas e despesas obtidas de forma diária ou mensal, validando se houve lucro ou prejuízo, auxiliando nas estratégias que serão tomadas pelo produtor.

2.1.7. Comparação entre os sistemas

Além dos critérios de seleção, todos os softwares selecionados possuem as funções de cadastros de usuários, registros de pesos, apontamentos nutricionais e controle da entrada e saída de animais. No Quadro 1 é possível visualizar as principais funcionalidades analisadas de cada aplicação e também o comparativo com o trabalho proposto.

Dentre os sistemas apresentados, o SGPO possui mais semelhanças ao que foi idealizado para este trabalho. Os demais sistemas buscam melhorar o processo de gestão assim como aumentar a eficiência dos pecuaristas quanto ao registro de informações da fazenda e da produção.

Quadro 1: Comparativo entre sistemas

Nome	**Leitura RFID**	**Histórico pesagens**	**Vermífugos e vacinas**	**Rastreio produtos**	**Gestão financeira**	**Plataforma**	**Licença**
iRancho	Sim	Sim	Sim	Não	Sim	*Android* e *web*	Paga
JetBov	Sim	Sim	Sim	Não	Sim	*Android* e iOS	Paga

DSGB	Não	Não	Não	Não	Não	*Android* e *web*	Gratuita
Rebanho App	Não	Não	Vacinas	Não	Sim	*Android*	Gratuita
DeskPec	Não	Não	Vacinas	Não	Não	*Desktop*	Gratuita
SGPO	Não	Sim	Sim	Não	Sim	*web*	Gratuita
TCC	Sim	Sim	Sim	Sim	Sim	*Android*, iOS e *web*	Gratuita

2.2. Tecnologias

As seções a seguir trazem informações descritivas sobre as tecnologias empregadas para o desenvolvimento do sistema.

2.2.1. PWA

Segundo Tandel e Jamadar (2018), uma PWA (*Progressive Web Application*) é um *website* que age como um aplicativo sem a necessidade de uma instalação nativa no dispositivo. Desta forma, esses aplicativos permitem a experiência de um *software* nativo com acesso rápido e funcionamento em modo *offline*. Por conta de sua natureza, as aplicações PWA dispensam custos de desenvolvimento para portabilidade entre sistemas operacionais e são compatíveis com diversos dispositivos, como *smartphones*, *tablets* e *desktop*.

De acordo com Mozilla (2023a), uma aplicação PWA pode ser definida através dos seguintes princípios-chave:

- **Descobrível**: permite a identificação como um aplicativo por mecanismos de pesquisa, utilizando os manifestos W3C (*World Wide Web Consortium*);
- **Instalável**: permite que os usuários salvem o aplicativo na tela inicial do dispositivo sem a necessidade de uma loja de aplicativos;
- **Linkável**: permite o compartilhamento via URL (*Uniform Resource Locator*), sem a requisição de uma instalação complexa;
- **Independência de conectividade**: permite a utilização do aplicativo mesmo *offline*;
- **Progressivo**: através do aprimoramento progressivo, permite o funcionamento do aplicativo independente do navegador utilizado;
- **Reengajamento**: permite manter o aplicativo atualizado através de notificações *push*;
- **Responsivo**: permite a adaptação da interface ao dispositivo;
- **Seguro**: permite a segurança do conteúdo através do protocolo HTTPS (*Hyper Text Transfer Protocol Secure*).

A abordagem PWA é utilizada no desenvolvimento desta aplicação por conta de suas vantagens supracitadas, oferecendo um sistema de fácil acesso ao usuário, sem a exigência de conexão à internet ou lojas de aplicativos.

2.2.2. Vue

O Vue é um *framework* progressivo para construção de interfaces para usuário, baseado em HTML, CSS e Javascript. O Vue fornece modelos de programação declarativos e focados na utilização de componentes, centralizando o núcleo da tecnologia em duas ideias principais: renderização declarativa e reatividade.

A renderização declarativa permite a geração de dados diretamente no DOM (*Document Object Model*), interligando o documento web com as informações da aplicação. Por conta dessa conexão, a estrutura do Vue possibilita a reatividade dos elementos, mantendo cada elemento atualizado conforme as alterações realizadas via código (Vue, 2023).

No contexto do trabalho, o Vue é usado para a construção do *front-end* da aplicação, desenvolvendo as interfaces para o acesso e visualização dos usuários e permitindo a conexão com o *back-end* implementado em Python, através do *framework* Django. As ações realizadas no *front-end* são transmitidas para o *back-end* e armazenadas no banco de dados da aplicação.

2.2.3. Django

O Django é um *framework* de código-aberto escrito em Python para desenvolvimento de aplicações *web* de forma rápida e segura. O Django utiliza a arquitetura *Model View Template* (MVT) para agrupar os códigos referentes à manipulação de solicitações HTTP (*Hyper Text Transfer Protocol*), organizando-os em quatro arquivos separados: URLs, *Views*, *Models* e *Templates* (Mozilla, 2023b).

Nas URLs, um mapeador é utilizado para redirecionar as solicitações HTTP de uma *View*, baseando-se na URL que consta na solicitação. A *View* é responsável pela manipulação das solicitações, acessando os dados necessários para contemplar as solicitações por meio dos *Models*, e traduzindo a resposta formada para os *Templates*. Os *Models* são representações da estrutura de dados do aplicativo, fornecendo o gerenciamento e consulta dos registros no banco de dados. Por fim, os *Templates* são arquivos de texto que definem o layout de arquivos como as páginas HTML (Mozilla, 2023b).

Dentro deste contexto, o Django é o responsável pelo *back-end* da aplicação, gerenciando as requisições e respostas solicitadas pelo front-end. O Django também realiza o mapeamento entre o modelo de classes e o banco de dados SQLite, mantendo as estruturas de dados em seu código.

2.2.4. SQLite

O SQLite é um sistema de gerenciamento de banco de dados (SGBD) embutido de código aberto baseado em uma arquitetura independente e sem a utilização de processos de um servidor, armazenando os dados em um único arquivo. Dessa forma, o armazenamento é compacto e simplificado, sendo utilizado principalmente por *websites* com pouco tráfego (SQLite, 2023). O SQLite foi selecionado por conta de sua simplicidade e facilidade de uso, já que inicialmente o *software* projetado não requer a utilização de um SGBD com conexão a um servidor e uma grande transição de dados.

2.2.5. Github

O *GitHub* é uma plataforma que realiza a hospedagem de repositórios de código-fonte utilizando *Git* e que fornece aos desenvolvedores o controle de versão de seus projetos. O *GitHub* possibilita a colaboração direta durante o processo de desenvolvimento, organizando o trabalho nos repositórios onde podem ser definidos requisitos ou direção para os integrantes da equipe (GitHub, 2023).

É usado um repositório no GitHub para este trabalho, visando um controle mais eficiente do projeto com as funções de edição simultânea, versionamento e registro de alterações.

2.2.6. Visual Studio Code

O *Visual Studio Code* é um editor de código gratuito, disponível para *Windows*, *macOS* e *Linux*. O VSCode possui suporte interno para JavaScript, TypeScript e Node.js, juntamente com extensões para outras linguagens como Java, Python e C# (Visual Studio Code, 2023). O editor é usado para a implementação e desenvolvimento da aplicação, escolhido principalmente por seu suporte a Python e Vue, e também devido à série de extensões que possui.

2.2.7. Bootstrap

O *Bootstrap* é um *framework* para a linguagem CSS de código-aberto, utilizado para customização de *websites* responsivos. O *Bootstrap* foi criado com a intenção de ser um guia com ferramentas de estilos para os desenvolvedores de um projeto (Bootstrap, 2023). No caso deste trabalho, o *Bootstrap* foi utilizado por conta de sua integração de estilos ao código, facilitando a implementação do front-end de forma ágil e intuitiva.

2.2.8. Biblioteca Axios

A biblioteca Axios é um cliente HTTP baseado em *Promises* responsável por gerenciar os métodos *GET*, *POST*, *PUT* e *DELETE* no navegador e no *node.js*. O *Axios* é isomórfico, ou seja, pode utilizar o mesmo código tanto no navegador com métodos *XMLHttpsRequests*, como no servidor *node.js* com o módulo http (Axios, 2023).

O *Promise* é um *proxy* para representar a possível conclusão ou falha de um valor ainda não conhecido no momento da criação do objeto, além de apresentar o resultado da atividade. Desta forma, o *Promise* permite que ações assíncronas retornem seus valores como métodos síncronos, prometendo a entrega do valor em um momento futuro ao invés de retornar de forma direta (Mozilla, 2023c).

Um *Promise* pode estar no estado *pending* quando ainda não houve a conclusão ou falha da operação, *fulfilled* quando é concluida com sucesso e *rejected* em caso de rejeição (Mozilla, 2023c).

2.2.9. Lighthouse

O *Lighthouse* é uma ferramenta oriunda do *Google* que audita sites para verificar seu desempenho, acessibilidade e questões sobre otimizações de busca, além de verificar se o site possui todos os pré-requisitos de um *PWA*. A ferramenta também disponibiliza várias sugestões de melhorias com base nas auditorias realizadas (Google, 2023). O *Lighthouse* foi escolhido por apresentar a informação sobre se o site cumpre ou não os pré-requisitos de um *PWA*.

2.2.10. Vercel

O *Vercel* é uma plataforma que realiza o *deploy* de uma aplicação, mais voltada ao *front-end*. A plataforma conta com alguns planos pagos e também o plano gratuito, o qual já permite o *deploy* funcional do *front-end* da aplicação. Para a utilização do *Vercel* é necessário ter uma conta *Github* (Vercel, 2023).

O *Vercel* foi escolhido pela facilidade na realização do *deploy* e também por possuir um plano gratuito.

3. Desenvolvimento

Esta seção aborda a modelagem utilizada para embasar a implementação. A seção 3.1 traz a modelagem do sistema, requisitos funcionais e a prototipação das interfaces. A seção 3.2 aborda a implementação propriamente dita, descrevendo a metodologia utilizada durante a construção do projeto e o código desenvolvido.

3.1. Modelagem do sistema

Esta seção apresenta os requisitos funcionais identificados para a construção da aplicação e também o modelo relacional do banco de dados.

3.1.1. Requisitos funcionais

A engenharia de requisitos estabelece uma base sólida para a construção do projeto, pois sem a definição correta dos requisitos, o software resultante tem grande probabilidade de não atender as necessidades esperadas (Pressman e Maxim, 2016).

Em função disto, o trabalho foi iniciado com o levantamento de requisitos funcionais para identificar os principais itens que devem estar presentes na aplicação. Os requisitos foram levantados considerando o estudo feito no caderno de campo usado no programa de assistência técnica e gerencial do SENAR (Serviço Nacional de Aprendizagem Rural). Além disto, foram realizadas conversas informais pelo grupo com pecuaristas da região visando complementar e esclarecer as informações obtidas junto ao caderno de campo

O Quadro 2 contém os requisitos funcionais identificados para o sistema, com título e descrição.

Quadro 2: Requisitos Funcionais

ID	Nome	Descrição
RF1	Gerenciar Produtor	Adicionar, remover, atualizar e visualizar Produtor, com os campos nome, cpf, email, user, password e telefone
RF2	Gerenciar Propriedade	Adicionar, remover, atualizar e visualizar Propriedade, com os campos nome, id_Produtor, tamanhoAreaProducao, endereco, latitude e longitude
RF3	Gerenciar Animal	Adicionar, remover, atualizar e visualizar Animal, com os campos id_Raca, observacoesRaca, sexo, peso, data_Nascimento, data_Morte, observacoes, rfid, id_Propriedade, id_animal_Mae, animal_Pai, numeroGestacoes, id_Lote e brinco
RF4	Gerenciar Vendas	Adicionar, remover, atualizar e visualizar registros das vendas dos Animais, com os campos finalidadeAnimal, valorAnimal, data_Venda, peso_Animal, precoAtualArrobaKg, dataRecebimento, observacao e id_Animal
RF5	Gerenciar Inseminações	Adicionar, remover, atualizar e visualizar Inseminações, com os campos dataInseminacao, horaCio, tecnica, diaGestacao, id_Vaca, identificadorTouro e id_Inseminador
RF6	Gerenciar Ocorrências	Adicionar, remover, atualizar e visualizar Ocorrências, com os campos ocorrencia e id_Animal
RF7	Gerenciar Movimentações	Adicionar, remover, atualizar e visualizar Movimentações, com os campos motivo, id_Animal, id_lote_origem, id_lote_destino e data
RF8	Gerenciar Suplementações	Adicionar, remover, atualizar e visualizar Suplementações, com os campos kgConsumo, dataAplicacao, id_Lote e id_Produto
RF9	Visualizar Despesas	Visualizar Despesas, com a soma dos campos valorCompra (comprar_Animal), valor (outras_despesas), valorTotal (compra_produto_sanitario), valorTotal (compra_produto_alimenticio), dataCompra (comprarAnimal), dataCompra(compra_produto_sanitario), dataCompra (compra_produto_alimenticio) e dataDespesa(outras_despesas)
RF10	Visualizar Receitas	Visualizar Receitas, com os campos valorAnimal (vendas), dataRecebimento e data_Venda (vendas)
RF11	Gerenciar Veterinários	Adicionar, remover, atualizar e visualizar Veterinários, com os campos nome, telefone, email e crmv
RF12	Gerenciar prod. Sanitários	Adicionar, remover, atualizar e visualizar Produtos Sanitários, com os campos nome e lote
RF13	Gerenciar prod. Alimentícios	Adicionar, remover, atualizar e visualizar Produtos Alimentícios, com os campos nome e tipo
RF14	Gerenciar Pesagens	Adicionar, remover, atualizar e visualizar Pesagens com os campos id_Animal, peso e dataPesagem
RF15	Gerenciar Raças	Adicionar, remover, atualizar e visualizar Raças, com o campo nome
RF16	Gerenciar Lotes	Adicionar, remover, atualizar e visualizar Lotes, com os campos nome, tipoCultivo e id_Propriedade
RF17	Gerenciar Compras de Produtos	Adicionar, remover, atualizar e visualizar Compras de Produtos, com os campos id_Produto, valorUnitario, qtd_Comprada, validade, dataCompra e descricao
RF18	Gerenciar Outras Despesas	Adicionar, remover, atualizar e visualizar Outras Despesas, com os campos dataDespesa, motivoGasto, valor e id_Propriedade
RF19	Gerenciar Compra de Animal	Adicionar, remover, atualizar e visualizar Compra de Animal, com os campos dataCompra, valorCompra, observacoes e id_Animal

3.1.2. Diagrama Relacional do Banco de Dados

Um banco de dados relacional é composto por tabelas, chaves, domínios, valores vazios e restrições de integridade. No contexto de um modelo relacional, as tabelas também são conhecidas como relações e são definidas como conjuntos não ordenados de tuplas ou linhas compostas de valores de atributos. As chaves são utilizadas para identificação das linhas e geralmente são divididas entre chaves primárias, estrangeiras e alternativas (Pichetti et al., 2020).

Para melhor descrever a relação entre as entidades presentes durante o armazenamento de dados, a Figura 1 apresenta um diagrama relacional do sistema proposto.

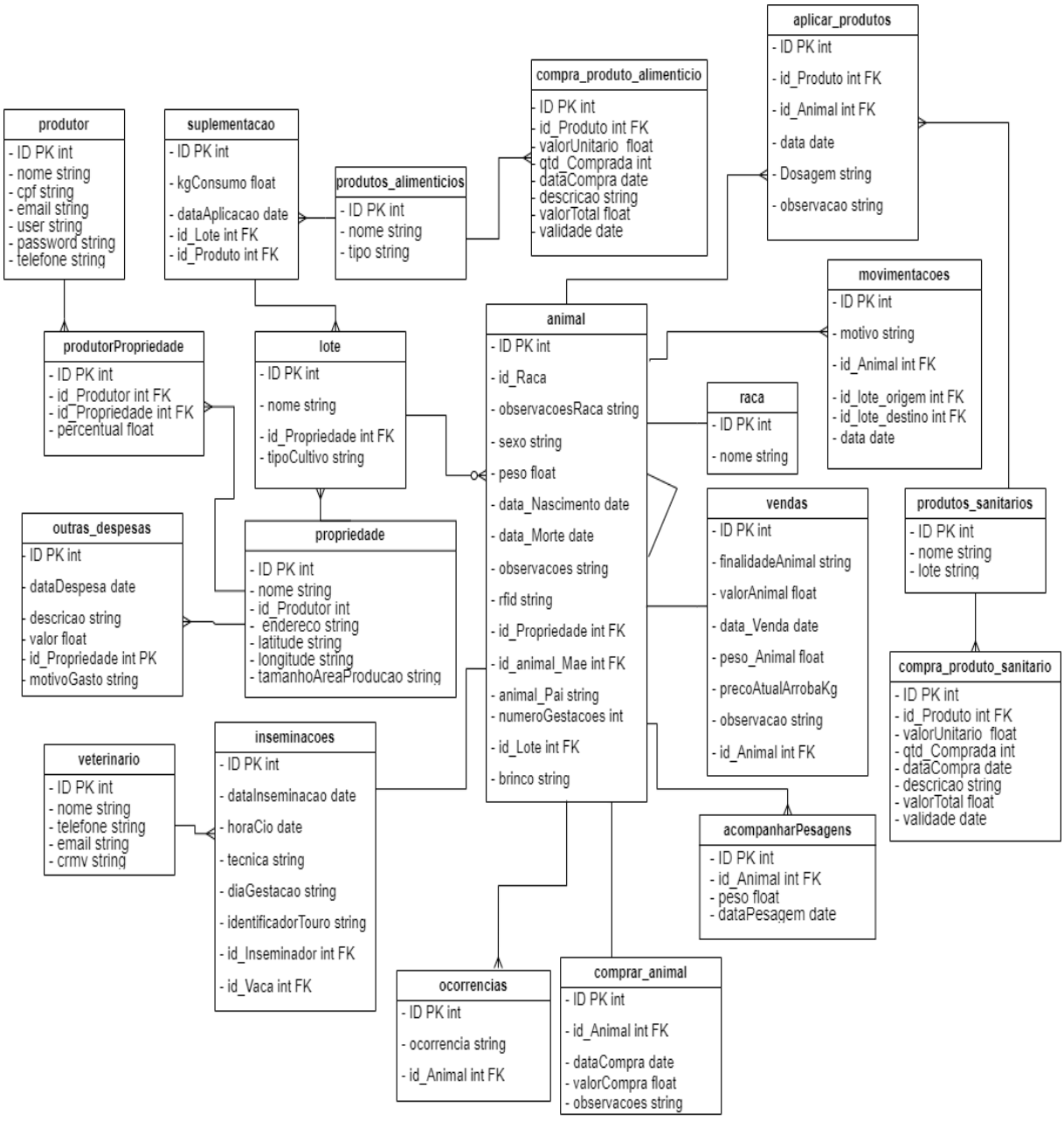

Figura 1. Diagrama Relacional do Banco de Dados

3.2. Implementação da PWA

Esta seção aborda as interfaces e códigos utilizados para a construção do sistema proposto. A seção 3.2.1 apresenta as interfaces resultantes do desenvolvimento e suas respectivas funções. A seção 3.2.2 evidencia de forma detalhada a lógica estruturada para a implementação do *front-end*, demonstrando os códigos relacionados à interface.

3.2.1. Front-end: Interfaces

As interfaces foram elaboradas de maneira intuitiva e eficiente ao usuário final, mantendo essas características não somente para o acesso de uma máquina *desktop*, mas também para dispositivos móveis, por conta da utilização da tecnologia *PWA*. Tais interfaces permitem o cadastro, edição, exclusão e visualização dos dados, como mostra a Figura 2 que apresenta o cadastro de propriedades rurais.

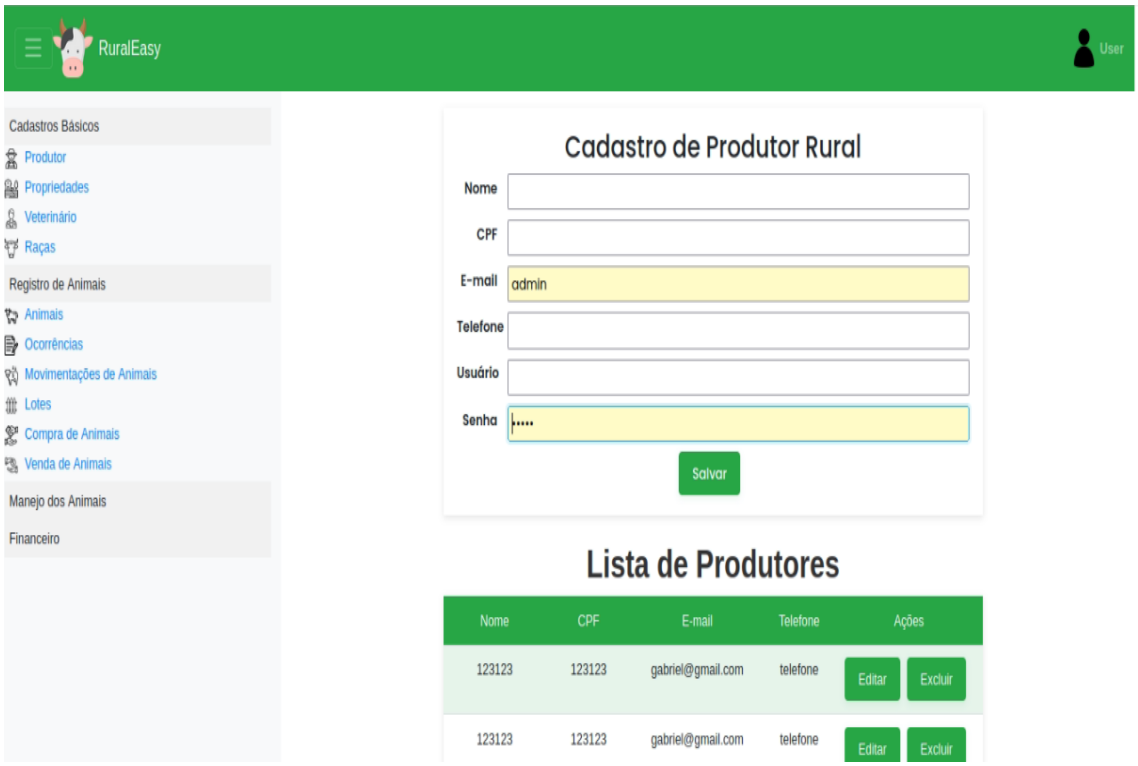

Figura 2. Interface de Produtor Rural

Esta interface possui uma área de cadastro e outra de visualização dos dados cadastrados. A área de cadastro, na parte superior da tela, é composta por um formulário contendo os campos necessários para inserção das informações da propriedade no sistema. Os campos são *Nome*, *Produtor*, *Endereço*, *Latitude* e *Longitude*. Ao preencher as informações e clicar no botão *Salvar*, é realizado o armazenamento dos dados no banco do sistema. O botão de *Salvar* também realiza um atualização na página, a reexibindo com a nova informação cadastrada.

Além dos cadastros e visualizações das propriedades, o usuário também pode

acessar ao menu da aplicação no canto superior esquerdo, clicando no ícone aparente, conforme Figura 3.

Figura 3. Menu da aplicação

O menu da aplicação é dividido em quatro conjuntos de opções para acesso. As opções são *Cadastros Básicos*, *Registros de Animais*, *Manejo dos Animais* e *Financeiro*. As demais interfaces de cadastros do sistema podem ser acessadas através deste menu. O menu interage de maneira responsiva, expandindo e contraindo conforme ações de clique do usuário.

Caso o sistema seja acessado por dispositivos móveis, as interfaces apresentadas se ajustam de acordo com a resolução do dispositivo, conforme Figura 4. A otimização garante que todos os campos sejam acessíveis, mantendo todas as funcionalidades da aplicação. Este comportamento da aplicação foi desenvolvido utilizando propriedades e métodos tanto do *Vue* como do *Bootstrap*.

Figura 4. Tela otimizada para dispositivo móvel

Além das interfaces de cadastro e listagem dos dados armazenados na aplicação, também foram implementados relatórios financeiros para auxiliar o produtor na administração de sua propriedade e transações. Esses relatórios incluem o acompanhamento das despesas e da lucratividade das receitas.

A interface *Despesas* exibe um relatório detalhado dos gastos realizados na propriedade, abrangendo a aquisição de animais, compra de insumos e outras despesas. Conforme mostrado na Figura 5, os dados são apresentados em forma de tabela, registrando a

data da despesa, seu valor e observações identificadoras. Além disso, é feito o cálculo do total das despesas, exibido acima da tabela.

Figura 5. Tela de resumo das despesas

No que diz respeito à interface *Receitas*, é apresentado um demonstrativo das receitas geradas pelas vendas de animais. A tabela inclui informações como a finalidade, valor e data da venda, a data de recebimento da receita, o peso do animal na ocasião da venda, o valor por arroba, observações adicionais e o identificador do brinco do animal vendido, como mostrado na Figura 6.

Além disso, o relatório na interface *Receitas* permite visualizar o valor total disponível em caixa, que reflete os recursos gerados pelas vendas e ajustados após as despesas serem subtraídas. Esse valor é atualizado após cada transação, seja uma venda ou despesa.

Figura 6. Tela de resumo das receitas

3.2.2. Front-end: Implementação

Para a elaboração do *front-end*, foram utilizadas em conjunto as tecnologias *Vue.js* e *Bootstrap*. As interfaces recebem as informações submetidas pelos usuários, interagem com a API e apresentam os resultados do processamento realizado por ela. A construção de uma interface de cadastro começa pela criação dos campos a serem preenchidos pelo usuário. O código apresentado no Quadro 3 descreve a criação do campo *Nome* do Cadastro de Propriedades.

Quadro 3: Cadastro de propriedade

```
1. <div class="container">
2.   <h2>Cadastro de Propriedade Rural</h2>
3.  <form @submit.prevent="submitForm">
4.  <div>
5.      <label for="nomeProp">Nome:</label>
6.      <input type="text" id="nomeProp" v-model="propriedade.nomeProp" required>
7.  </div>
8. <div>
```

Para auxiliar nas atividades do usuário, foi idealizado um campo para a pré-seleção das entidades já armazenadas nas tabelas da base de dados. Dessa forma, em caso de necessidade de edição, exclusão ou inserção, o usuário não precisa digitar manualmente uma entidade e pode seguir apenas com a escolha do elemento desejado através de um registro na caixa de seleção. Essa funcionalidade é essencial nas interfaces de cadastro de entidades que possuem vínculos com outras tabelas através de chaves estrangeiras, assim evitando divergências nos dados e perda de associação entre as entidades.

Como exemplo desta função, o Quadro 4 apresenta o código implementado para o funcionamento dessa propriedade do sistema na interface de Cadastro de Propriedade. Na linha 3, é instanciada a caixa de seleção com base no atributo Produtor da entidade Propriedade. A informação desse atributo é vinculada a um *v-model*, que é uma diretiva do *Vue* que permite a ligação entre a seleção do formulário e a instância *Vue* criada para a Propriedade.

Nas linhas 4 e 5, são inseridas as informações dos produtores através de uma lista de produtores enviadas através da API do *back-end* e o atributo de nome do produtor é apresentado ao usuário para seleção.

Quadro 4: Seleção dos produtores

```
1.  <div>
2.      <label for="produtor">Produtor:</label>
3.          <select id="produtor" v-model="propriedade.produtor" required>
4.          <option value="">Selecione um produtor</option>
5.          <option v-for="produtor in produtores" :value="produtor.id" :key="produtor.
    id">{{ produtor.nomeProd }}</option>
6.          </select>
7.  </div>
```

As *interfaces* construídas para a visualização das entidades possuem laços de repetição que apresentam os campos já cadastrados e suas descrições. Essas interfaces também permitem a edição e exclusão dos dados através dos botões *Editar* e *Excluir*.

No quadro 5, nas linhas 11 e 12, é demonstrado o laço de repetição criado para a visualização dos dados da entidade Propriedade. Nas linhas 15 e 16, estão presentes os botões *Editar* e *Excluir*.

Quadro 5: Laço de repetição e botões de exclusão e edição

```
1.  <div class="table">
2.      <h1>Lista de Propriedades Rurais</h1>
3.          <table>
4.              <thead>
5.                  <tr>
6.                      <th>Nome</th>
7.                      [...]
8.                  </tr>
9.              </thead>
10.             <tbody>
11.                 <tr v-for="propriedade in propriedades" :key="propriedade.id">
12.                     <td>{{ propriedade.nomeProp }}</td>
13.                     [...]
14.                     <td>
15;     <button @click.prevent="editPropriedade(propriedade)">Editar</button>
16.     <button @click.prevent="confirmDelete(propriedade)">Excluir</button>
17.                     </td>
18.                 </tr>
19.             </tbody>
20.         </table>
21. </div>
```

3.3. API Back-end

A API foi idealizada para comunicar-se com o *front-end* através dos métodos implementados com o *framework* Django. Esses métodos são responsáveis pelas ações de atualização, inserção, remoção e visualização das informações no banco de dados SQLite. Também é possível serializar os dados e validá-los, para garantir a segurança e facilidade da troca de informações.

No processo de comunicação entre *front-end* e *back-end*, algumas bibliotecas podem ser utilizadas para ajudar na comunicação, como a já citada *Axios*. O *Axios* realiza o controle dos métodos de chamada da API *back-end*, como o *put* para as alterações, o *get* para a visualização, o *delete* para a exclusão e o *post* para as inserções.

Como apresentado no Quadro 6, em uma ação de adição de um cadastro de uma propriedade, o fluxo da API começa com o usuário interagindo com os formulários do *front-end*. A partir da linha 7, é possível observar os campos esperados como resposta da inserção ou atualização dos dados.

A URL da requisição feita, visível nas linhas 3 e 19 do Quadro 6, (*axios.post, axios.put*) é interpretada pelo Django através das definições da *urls.py* implementada, conforme Quadro 7. Presente nas demais interfaces do aplicativo, a *urls.py* é responsável por armazenar as rotas da aplicação e direcionar a requisição gerada pelo *front-end* para a classe implementada na *views.py*, conforme exemplo do Quadro 7.

No exemplo do Quadro 8, pode-se observar o método que realiza a requisição HTTP para a inserção da entidade na base de dados, além de dar um retorno ao usuário, visível nas linhas 6, 7 e 8.

A classe em *views.py* interage com o banco de dados através da *models.py*. Na *models.py*, estão presentes os campos de atributos das entidades da aplicação, conforme

idealizado durante o Diagrama Relacional do Banco de Dados. No Quadro 9, pode-se observar o exemplo da entidade de Propriedade.

Quadro 6: Envio do formulário

```
1.  submitForm() {
2.  if (this.isEdit) {
3.  axios.put(`http://localhost:8000/propriedades/editar/${this.propriedade.id}/`, this.
    propriedade)
4.  .then(response => {
5.      this.propriedades = response.data;
6.      this.isEdit = false;
7.      this.propriedade = {
8.      nomeProp: '',
9.      produtor: this.produtor.id,
10.     endereco: '',
11.     latitude: '',
12.     longitude: ''
13.         };
14.     })
15.     .catch(error => {
16.         console.log(error);
17.     });
18.     } else {
19.         axios.post('http://localhost:8000/propriedades/', this.propriedade)
20.         .then(response => {
21.             this.propriedades = response.data;
22.             this.propriedade = {
23.                 nomeProp: '',
24.                 produtor: this.produtor.id,
25.                 endereco: '',
26.                 latitude: '',
27.                 longitude: ''
28.                 };
29.             })
30.         .catch(error => {
31.             console.log(error);
32.         });
33.     }
34. setTimeout(function() {
35.     location.reload();
36.     }, 1000);
37.  },
```

Quadro 7: *urls.py*

```
1.  path('propriedades/', PropriedadeRuralList.as_view()),
2.  path('propriedades/propriedade/<int:id_propriedade>/',views.mostra_propriedade,name
    ='mostra_propriedade'),
3.  path('propriedades/editar/<int:pk>/', views.editar_propriedade, name='
    editar_propriedade'),
4.  path('propriedades/deletepropriedade/<int:propriedade_id>/', delete_propriedade,
    name='delete_propriedade'),
```

Quadro 8: *views.py*

```
1. def adicionar_propriedade(request):
2.     if request.method == 'POST':
3.         serializer = PropriedadeRuralSerializer(data=request.data)
4.         if serializer.is_valid():
5.             serializer.save()
6.             return JsonResponse({'mensagem': 'Propriedade adicionada com sucesso.'},
    status=201)
7.             return JsonResponse(serializer.errors, status=400)
8.             return JsonResponse({'mensagem': 'Metodo nao permitido.'}, status=405)
```

Quadro 9: *models.py*

```
1.  class PropriedadeRural (models.Model):
2.      nomeProp = models.CharField(max_length=100)
3.      produtor = models.ForeignKey(ProdutorRural, on_delete=models.CASCADE, blank=True
    , null=True)
4.      endereco = models.CharField(max_length=100, default='endere o')
5.      latitude = models.CharField(max_length=100, default='latitude')
6.      longitude = models.CharField(max_length=100, default='longitude')
7.
8.      def __str__(self):
9.          return self.nomeProp
```

O formulário desenvolvido no *front-end* segue as parametrizações que foram atribuídas durante a criação do *forms.py*. O *forms.py* descreve um formulário e o seu funcionamento, definindo como o formulário deve ser apresentado e quais dados devem ser recepcionados por parte da API. Para exemplificar o funcionamento desta classe no Django, é apresentado o *forms.py* para a entidade de Propriedade no Quadro 10.

Quadro 10: *forms.py*

```
1.  class PropriedadeRuralForm(forms.ModelForm):
2.      class Meta:
3.          model = PropriedadeRural
4.          fields = ['nomeProp', 'produtor', 'endereco', 'latitude', 'longitude']
```

É necessário que haja a padronização da troca de informações entre o *front-end* e o *back-end* durante o processo de inserção das informações no banco de dados. Com este propósito, foi necessário o desenvolvimento de um serializador que transformasse arquivos *JSON* enviados pelas interfaces do *Vue* em objetos *Django* e vice-e-versa. Como exemplo de um serializador construído, o Quadro 11 apresenta a *serializers.py* feita para o objeto da classe Propriedade.

Quadro 11: *serializers.py*

```
1.      class PropriedadeRuralSerializer(serializers.ModelSerializer):
    class Meta:
        model = PropriedadeRural
        fields = ['id', 'nomeProp', 'produtor', 'endereco', 'latitude', 'longitude']
```

Após a serialização dos dados gerados, uma resposta é enviada ao *front-end*, que exibe as novas informações inseridas através da interface do usuário. Para a exibição das informações, foi utilizado um método *GET*, juntamente com a implementação para a exibição no *front-end*. No Quadro 12, como exemplo, é apresentado o método *GET* criado para a Propriedade e o seu respectivo retorno ao *front-end* no Quadro 13.

Quadro 12: Método *GET*

```
1.  mounted() {
2.      axios.get('http://localhost:8000/propriedades/')
3.      .then(response => {
4.          this.propriedades = response.data
5.       })
6.       .catch(error => {
7.          console.log(error)
8.       })}
```

Quadro 13: Retorno ao *front-end*

```
1.  <tr v-for="propriedade in propriedades" :key="propriedade.id">
2.      <td>{{ propriedade.nomeProp }}</td>
3.      <td>{{ propriedade.endereco }}</td>
4.      <td>{{ propriedade.latitude }}</td>
5.      <td>{{ propriedade.longitude }}</td>
6.  </tr>
```

A criação das demais interfaces do sistema foram implementadas seguindo as estruturas apresentadas nesta seção, padronizando a lógica de desenvolvimento dos componentes tanto no *front-end* como no *back-end*. Os demais códigos estão disponíveis no repositório *GitHub*[1] do trabalho.

3.4. Testes

Segundo Pressman e Maxim (2021), os testes de *software* são atividades que podem ser planejadas antecipadamente e são executadas sistematicamente com o objetivo de revelar erros cometidos pelos desenvolvedores durante o projeto e construção de um sistema. Os testes são realizados inicialmente de forma unitária em nível de componente e devem progredir para os testes de integração do sistema computacional como um todo.

O processo de desenvolvimento de um *software* pode ser representado por um espiral, ilustrado na Figura 7, iniciando-se com definição do objetivo pela engenharia de sistemas, passando para a análise de requisitos e então avançando para o projeto e codificação em seu interior (Pressman e Maxim, 2021).

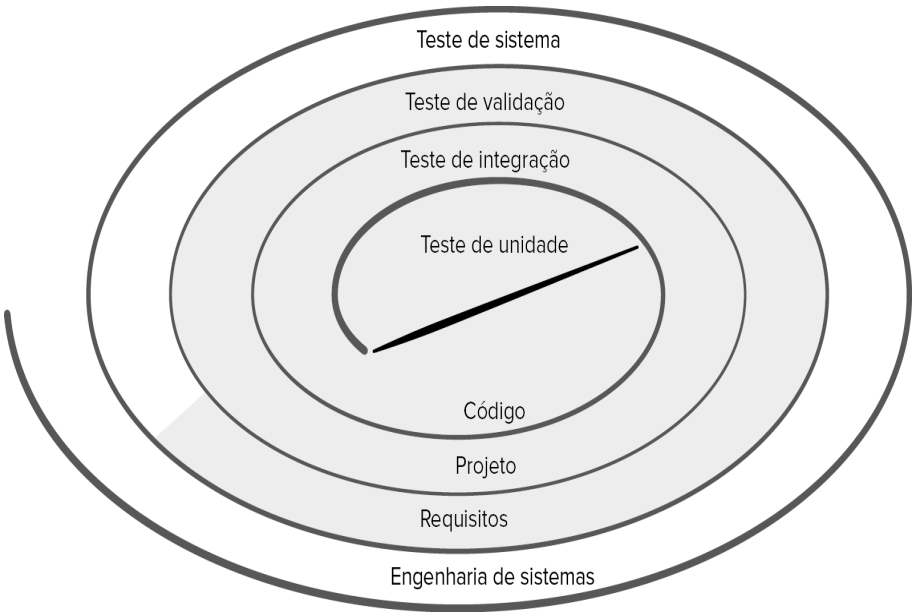

Figura 7. Estratégia de testes
Fonte: Pressman e Maxim (2021)

O conceito do espiral também pode ser usado para a estratégia de testes de um *software*. Os testes de unidade concentram-se no centro do espiral e são responsáveis

[1] https://github.com/TCC-PWA-IFSC/RuralEasy

por cada componente da aplicação. Em seguida, são realizados os testes de integração com foco na arquitetura e consistência da construção do projeto. Nos testes de validação, são validados se os requisitos elaborados no projeto são atendidos no sistema construído. Por fim, os testes de sistema verificam o *software* em sua totalidade (Pressman e Maxim, 2021).

Para o *RuralEasy*, foram utilizados apenas os testes de sistema, os testes de integração e os testes de validação, não sendo utilizado os testes de unidade pois foi avaliado que os testes planejados já englobariam as necessidades de validação do trabalho. Com os testes de sistema, foram validadas as funções básicas do sistema, como o registro e visualização de informações. Os testes foram baseados na usabilidade comum do usuário, apenas registrando, excluindo e editando informações. Ainda durante esse processo, foi possível confirmar que ambas as camadas de *back-end* e *front-end* estavam trabalhando em harmonia, conforme evidenciado pelos testes de integração.

Após a verificação do funcionamento das funcionalidades do sistema, foram realizados os testes de validação, onde constatou-se que todas seções implementadas atendiam os requisitos definidos durante a modelagem do trabalho.

3.4.1. Lighthouse

Como complemento aos testes, foi utilizada a ferramenta *Lighthouse* para auditar questões de desempenho, acessibilidade, práticas recomendadas, motor de otimização de busca (*SEO*) e também para validar se a aplicação estava cumprindo os critérios para ser considerada uma *PWA*.

Para que a ferramenta funcionasse, foi necessário realizar o *deploy* do *front-end* da aplicação utilizando a plataforma *Vercel*. O *deploy* foi realizado de maneira automática pela própria plataforma, necessitando apenas da conta *Github*.

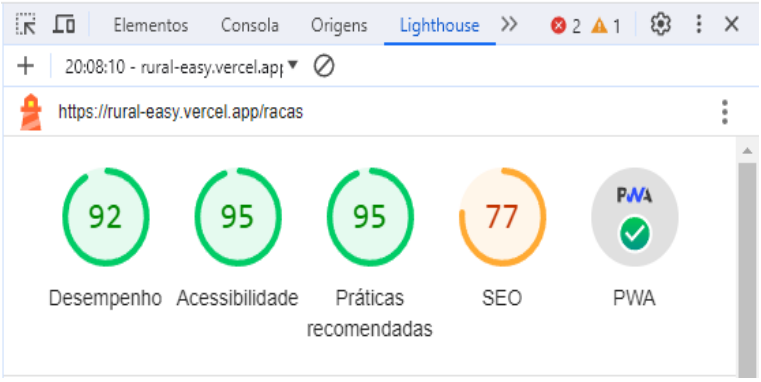

Figura 8. Auditoria realizada pelo *Lighthouse*

Conforme Figura 8, percebe-se que o sistema alcançou bons resultados em relação ao desempenho, a acessibilidade, as práticas recomendadas e aos requisitos para um *PWA*. Porém ele obteve resultados medianos em relação ao *SEO*. Foram recomendadas a inclusão de uma meta descrição, para que seja mais fácil a compreensão do conteúdo da aplicação e, conforme Figura 9, que haja uma melhor formatação do arquivo *robots.txt*,

arquivo responsável por configurar os acessos e permissões de softwares de rastreio e pesquisa. Ressalta-se que as melhorias relacionadas a parte do *SEO* não são, atualmente, prioritárias para a aplicação, uma vez que não há a necessidade de que seja possível pesquisar pela aplicação através de um motor de busca.

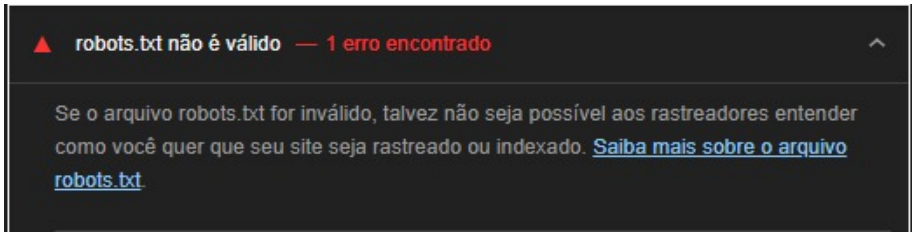

Figura 9. Observação sobre o *robots.txt*

4. Considerações finais

O agronegócio tem um papel fundamental na economia brasileira, contribuindo fortemente para o PIB e empregando famílias que tiram seu sustento desse mercado. De livros e lápis, a cadernetas e canetas, a gestão desse setor necessita cada vez mais de uma renovação que promova eficiência. Para atender esta demanda, este trabalho se propôs a desenvolver um sistema web gratuito voltado aos pequenos produtores. Para este propósito, foram utilizadas tecnologias como *Vue*, *Django*, *Python* e *Progressive Web Application*.

Primeiramente, foram necessários estudos relativos as tecnologias empregadas no desenvolvimento do projeto. Após a realização dos estudos, foram concentrados esforços na construção das interfaces utilizando *Vue* e os conceitos de *PWA*, até então aprendidos. Com o intuito de tornar as interfaces familiares ao usuário final que as utilizaria, foram elaborados requisitos funcionais com base no caderno de campo do SENAR (Serviço Nacional de Aprendizagem Rural). As interfaces foram modeladas e ficaram de acordo com os protótipos realizados no início do projeto. Foram registradas apenas algumas dificuldades em relação a estilização, as quais foram resolvidas.

A terceira etapa do desenvolvimento teve como objetivo a modelagem e implementação de uma *API REST* para o *back-end*, utilizando *Python* e *Django*. A primeira parte desse desenvolvimento foi a criação do diagrama relacional do banco de dados, baseado nos requisitos funcionais. O desenvolvimento da *API REST* foi realizada de maneira segmentada, implementando cada *model* de maneira completa. Para cada *model* foi definida uma tabela no banco de dados, além dos métodos HTTP de cada modelo para a sua comunicação com o *front-end*. Além disso, foram implementadas as lógicas para cada um dos processos de cadastro, edição, exclusão e visualização dos dados.

Por fim, foram realizados os testes de integração, sistema e validação para constatar se todas as funcionalidades do sistema estavam satisfazendo os requisitos previamente definidos, além de confirmar se as principais funcionalidades da aplicação estavam funcionais. Além dos testes supracitados, foi realizado o *deploy* do front-end da aplicação através da ferramenta Vercel, para avaliar as propriedades PWA do *software* com o auxílio da extensão *Lighthouse*, comprovando que o sistema desenvolvido era uma aplicação *web* progressiva. Apesar do sucesso no *deploy* do *front-end*, não houve a disponibilização da

API desenvolvida, por conta de dificuldades encontradas pelos autores para seguir com o procedimento, havendo uma limitação de ferramentas gratuitas para esse fim no mercado e também dificuldades técnicas em relação a um *deploy* alternativo pela própria máquina.

Ainda que o objetivo principal do trabalho tenha sido alcançado, existem melhorias a serem feitas. O *deploy* completo e funcional da aplicação é uma delas, conforme referenciado no parágrafo acima. A funcionalidade de *login* no sistema utilizando os campos já cadastrados para os produtores é outra atividade a ser desenvolvida futuramente. Além disso, a aplicação atual é limitada apenas a produtores de gados de corte, porém ela poderia agregar funcionalidades extras para produtores de leite ou quaisquer criação de outros animais. O controle de estoque foi uma funcionalidade avaliada durante o desenvolvimento da aplicação, todavia não foi implementada, o que é algo a ser considerado em trabalhos futuros. Por fim, há também a integração do RFID com a aplicação, uma funcionalidade que não foi implementada durante o projeto e que auxiliará na fácil identificação do animal.

Referências

Axios (2023). Getting Started. Disponível em: https://axios-http.com/ptbr/docs/intro. Acesso em: 19 de set. de 2023.

Bootstrap (2023). Get started with bootstrap. Disponível em: https://getbootstrap.com/docs/5.3/getting-started/introduction. Acesso em: 21 de abr. de 2023.

Carvalho, T. G. (2019). Análise e desenvolvimento de um sistema offline para auxílio gerencial da pecuária de corte e fluxo de caixa simples para pequenas propriedades rurais. Orientador: Luciana Recart Cardoso. TCC (Formação Tecnóloga em Análise e Desenvolvimento de Sistemas). Instituto Federal Goiano – Campus Iporá. Iporá.

Centro de Inteligência da Carne Bovina (2020). Gestão pecuária e desafios futuros. Disponível em: https://www.infoteca.cnptia.embrapa.br/infoteca/bitstream /doc/1127142/1/Boletim-CiCarne-32.pdf. Acesso em: 9 de mar. de 2023.

Confederação da Agricultura e Pecuária (2021). Panorama do agro. Disponível em: https://www.cnabrasil.org.br/cna/panorama-do-agro. Acesso em: 8 de mar. de 2023.

Costa, E. G., Klein, A. Z., e Vieira, L. M. (2014). Análise da utilização de tecnologias da informação móveis e sem fio (tims) na cadeia bovina: um estudo de caso no estado de goiás. Disponível em: https://www.scielo.br/j/read/a/6656MWQP6B k4fdwfkR474Xm/?lang=pt. Acesso em: 9 de mar. de 2023.

Embrapa (2022). A organização da produção agropecuária aumenta a competitividade. Disponível em: https://www.embrapa.br/busca-de-noticias/-/noticia/75579830/artigo—a-organizacao-da-producao-agropecuaria-aumenta-a-competitividade. Acesso em: 9 de mar. de 2023.

Fundação Getúlio Vargas (2018). Setor de carnes no brasil. Disponível em: https://gvagro.fgv.br/sites/gvagro.fgv.br/files/u115/03_Setor_Carnes_Brasil_PT.pdf. Acesso em: 8 de mar. de 2023.

Fundação Getúlio Vargas (2019). Agroindústria brasileira: Retrato do setor e projeções para 2019. Disponível em: https://gvagro.fgv.br/sites/gvagro.fgv.br/files/u115/2019-04-08_agroindustria_fgv_PT.pdf. Acesso em: 8 de mar. de 2023.

GitHub (2023). Github docs. Disponível em: https://docs.github.com/pt. Acesso em: 19 de abr. de 2023.

Google (2023). Lighthouse - Overview. Disponível em: https://developer.chrome.com/docs/lighthouse/overview/. Acesso em: 25 de set. de 2023.

IBGE (2021). Produção agropecuária. Disponível em: https://www.ibge.gov.br/explica/producao-agropecuaria/. Acesso em: 8 de mar. de 2023.

iRancho (2023). A nova gestão da sua fazenda começa aqui! Disponível em: https://www.irancho.com.br. Acesso em: 20 de mar. de 2023.

Jetbov (2023). Controle do gado e gestão da fazenda sem complicação. Disponível em: https://jetbov.com Acesso em: 20 de mar. de 2023.

Microsoft (2023). Overview of progressive web apps (pwas). Disponível em: https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/. Acesso em: 14 de mar. de 2023.

Mozilla (2023a). Introduction to progressive web apps. Disponível em: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Introduction. Acesso em: 18 de abr. de 2023.

Mozilla (2023b). Introdução ao django. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Django/Introduction. Acesso em: 21 de abr. de 2023.

Mozilla (2023c). Promise. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise. Acesso em: 19 de set. de 2023.

Pichetti, R. F. V., Cortes, E. S., e Paixão, V. S. M. (2020). *Banco de Dados*. Grupo A. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9786556900186/pageid/0. Acesso em: 20 de abr. de 2023.

Pressman, R. e Maxim, B. (2016). *Engenharia de Software*. Grupo A. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9788580555349/pageid/0. Acesso em: 16 de abr. de 2023.

Pressman, R. e Maxim, B. (2021). *Engenharia de Software*. Grupo A. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9786558040118/epubcfi/6/2Acesso em: 02 de nov. de 2023.

Rauta, E. A. P. (2016). Desenvolvimento de um sistema para gerenciamento bovino. Orientador: Razer Anthom Nizer Rojas Montanõ. Monografia (Pós-Graduação em Engenharia de Software). Universidade Federal do Paraná. Curitiba.

Rodrigues, A. M. M., Oliveira, C. A. F., e Malerba, G. H. (2021). Deskpec: Software de gestão pecuária. Orientador: Camila Brandão Fantozzi. TCC (Formação Técnica em Desenvolvimento de Sistemas) - Centro Paula Souza ETEC Philadelpho Gouvêa Netto. São José do Rio Preto.

Silva, E. L. e Menezes, E. M. (2005). Metodologia da pesquisa e elaboração de dissertação. Disponível em: https://tccbiblio.paginas.ufsc.br/file/2010/09/024_Metodologia_de_pesquisa_e_ elaboracao_ de_teses _e_dissertacoes1.pdf. Acesso em: 9 de mar. de 2023.

Sociedade Nacional da Agricultura (2020). Agronegócio: A força da economia brasileira. Disponível em: https://www.sna.agr.br/agronegocio/. Acesso em: 8 de mar. de 2023.

SQLite (2023). About sqlite. Disponível em: https://www.sqlite.org/about.html. Acesso em: 19 de abr. de 2023.

Tandel, S. S. e Jamadar, A. (2018). Impact of progressive web apps on web app development. Disponível em: https://www.researchgate.net/profile/Sayali-Tandel-2/publication/330834334_Impact_of_Progressive_Web_Apps_on_Web_App_Development/links/5c5605d3a6fdccd6b5dde018/Impact-of-Progressive-Web-Apps-on-Web-App-Development.pdf. Acesso em: 18 de abr. de 2023.

Vercel (2023). O que é o Vercel. Disponível em: https://vercel.com/blog/what-is-vercel. Acesso em: 25 de set. de 2023.
Visual Studio Code (2023). Getting started. Disponível em: https://code.visualstudio.com/docs. Acesso em: 20 de abr. de 2023.
Vue (2023). Whats is vue? Disponível em: https://vuejs.org/guide/introduction.html. Acesso em: 18 de abr. de 2023.