

**Adryan Thiago de Oliveira Francisco**

**Felipe Pacheco Bianchini**

**Guilherme Henrique de Oliveira**

**Wendell Alves Nascimento**

**GRANJA TECH**

**Gestão e monitoramento de granjas**

**Projeto Integrador**

Jales

2025

**Lista de Figuras**

[Figura 1 - AgroSui 9](#_Toc208869170)

[Figura 2 – AgroPostura 10](#_Toc208869171)

[Figura 3 – Aegro 11](#_Toc208869172)

[Figura 4 - Diagrama de classe 17](#_Toc208869173)

[Figura 5 – Atores do Sistema 27](#_Toc208869174)

[Figura 6 - Diagrama Geral do Financeiro – Visão do Financeiro 35](#_Toc208869175)

[Figura 7 - Diagrama Geral do Produtor – Visão do Produtor 36](#_Toc208869176)

[Figura 8 — Diagrama Geral do Administrador – Visão do Administrador 37](#_Toc208869177)

[Figura 9 – Diagrama de Caso de Uso Individual – Administrador Lista Lotes 38](#_Toc208869178)

[Figura 10 – Diagrama de Caso de Uso Individual - Produtor Consumo Ração 41](#_Toc208869179)

[Figura 11 – Diagrama de Caso de Uso Individual - Produtor Eventos Sanitários 44](#_Toc208869180)

[Figura 12 - Diagrama de Caso de Uso Individual - Financeiro Transações Financeiras 47](#_Toc208869181)

[Figura 13 – Persona 1 54](#_Toc208869182)

[Figura 14 – Persona 2 54](#_Toc208869183)

[Figura 15 – Wireframe - Tela Login 55](#_Toc208869184)

[Figura 16 – Wireframe – Tela Dashboard 56](#_Toc208869185)

[Figura 17 – Wireframe – Tela Cadastro de Usuário 57](#_Toc208869186)

[Figura 18 – Wireframe - Tela Granjas 58](#_Toc208869187)

[Figura 19 - Wireframe - Tela Lotes 59](#_Toc208869188)

[Figura 20 – Protótipo Tela – Cadastrar Usuário 60](#_Toc208869189)

[Figura 21 – Protótipo Tela – Login do usuário 61](#_Toc208869190)

[Figura 22 – Protótipo Tela – Dashboard 62](#_Toc208869191)

[Figura 23 – Protótipo Tela – Granja 63](#_Toc208869192)

[Figura 24 – Protótipo Tela – Lotes 64](#_Toc208869193)

[**Figura 25 -** Mapeamento do Objeto Relacional 69](#_Toc208869194)

[Figura 26 - Diagrama de Componentes da Arquitetura do Back-End 82](#_Toc208869195)

[Figura 27 - Diagrama de Componentes da Arquitetura do Front-End (Web) 84](#_Toc208869196)

[Figura 28 - Diagrama de Componentes da Arquitetura do Front-End (Mobile) 87](#_Toc208869197)

[Figura 29 - Diagrama de Implantação 91](#_Toc208869198)

Lista de Quadros

[Quadro 1 - Requisitos Funcionais 12](#_Toc208848652)

[Quadro 2 - Requisitos Não Funcionais 13](#_Toc208848653)

[Quadro 3 - Descrição Classe Perfil 18](#_Toc208848654)

[Quadro 4 - Descrição Classe Usuario 18](#_Toc208848655)

[Quadro 5 - Descrição Classe Granja 19](#_Toc208848656)

[Quadro 6 - Descrição Classe Lote 19](#_Toc208848657)

[Quadro 7 - Descrição Classe Produto 20](#_Toc208848658)

[Quadro 8 - Descrição Classe TransacaoFinanceira 20](#_Toc208848659)

[Quadro 9 - Descrição Classe Sensor 20](#_Toc208848660)

[Quadro 10 – Descrição Classe LeituraSensor 21](#_Toc208848661)

[Quadro 11 – Descrição Classe ConsumoAgua 21](#_Toc208848662)

[Quadro 12 – Descrição Classe ConsumoRacao 22](#_Toc208848663)

[Quadro 13 – Descrição Classe EventoSanitario 22](#_Toc208848664)

[Quadro 14 - Descrição Classe MedicaoQualidadeAr 23](#_Toc208848665)

[Quadro 15 - Descrição Classe PesagemSemanal 23](#_Toc208848666)

[Quadro 16 - Descrição Classe RegistroAbate 24](#_Toc208848667)

[Quadro 17 - Descrição Classe RegistroMortalidade 25](#_Toc208848668)

[Quadro 18 - Descrição Classe LogAuditoria 25](#_Toc208848669)

[Quadro 19 - Descrição Classe FinanceiroProdutor 25](#_Toc208848670)

[Quadro 20 - Definição dos Atores 26](#_Toc208848671)

[Quadro 21 - Lista de Mensagens 28](#_Toc208848672)

[Quadro 22 - Casos de Uso – Administrador 29](#_Toc208848673)

[Quadro 23 - Casos de Uso – Produtor 30](#_Toc208848674)

[Quadro 24 - Casos de Uso – Financeiro 32](#_Toc208848675)

[Quadro 25 - Casos de uso – Sistema 32](#_Toc208848676)

[Quadro 26 - Documentação - Administrador Lista Lotes 38](#_Toc208848677)

[Quadro 27 - Documentação - Administrador Cadastrar Lote 38](#_Toc208848678)

[Quadro 28 - Documentação - Administrador Alterar Lote 39](#_Toc208848679)

[Quadro 29 - Documentação - Administrador Encerrar Lote 39](#_Toc208848680)

[Quadro 30 - Documentação - Administrador Excluir Lote 40](#_Toc208848681)

[Quadro 31 - Documentação - Produtor Registrar Consumo de Ração 41](#_Toc208848682)

[Quadro 32 - Documentação – Produtor Listar Consumo de Ração 41](#_Toc208848683)

[Quadro 33 - Documentação - Produtor Alterar Consumo de Ração 42](#_Toc208848684)

[Quadro 34 - Documentação - Produtor Excluir Consumo de Ração 42](#_Toc208848685)

[Quadro 35 - Documentação – Produtor Aplicar Filtros 43](#_Toc208848686)

[Quadro 36 - Documentação - Produtor Listar Eventos Sanitários 44](#_Toc208848687)

[Quadro 37 - Documentação - Produtor Registrar Evento Sanitário 44](#_Toc208848688)

[Quadro 38 - Documentação - Produtor Alterar Evento Sanitário 45](#_Toc208848689)

[Quadro 39 - Documentação - Produtor Excluir Evento Sanitário 45](#_Toc208848690)

[Quadro 40 - Documentação - Produtor Aplicar Filtros (Eventos Sanitários) 46](#_Toc208848691)

[Quadro 41 - Documentação - Financeiro Listar Transações Financeiras 47](#_Toc208848692)

[Quadro 42 - Documentação - Financeiro Aplicar Filtros 47](#_Toc208848693)

[Quadro 43 - Documentação - Financeiro Cadastrar Entrada/Saída 48](#_Toc208848694)

[Quadro 44 - Documentação - Financeiro Alterar Entrada/Saída 48](#_Toc208848695)

[Quadro 45 - Documentação - Financeiro Excluir Entrada/Saída 49](#_Toc208848696)

[Quadro 46 - Documentação - Financeiro Calcular Lucro 49](#_Toc208848697)

[Quadro 47 - Documentação -Financeiro Gerar Relatório Financeiro 50](#_Toc208848698)

[Quadro 48 – Cenário – Registro de Mortalidade de Aves pelo Produtor 52](#_Toc208848699)

[Quadro 49 – Cenário – Análise de Dashboards pelo Responsável Financeiro da Granja 52](#_Toc208848700)

[Quadro 50 - Tabela de Logs de Auditoria 70](#_Toc208848701)

[Quadro 51 - Tabela de Perfis 70](#_Toc208848702)

[Quadro 52 - Tabela de Usuários 70](#_Toc208848703)

[**Quadro 53 -** Relacionamento Financeiro/Produtor 71](#_Toc208848704)

[**Quadro 54 -** Tabela de Granjas 71](#_Toc208848705)

[**Quadro 55 -** Tabela de Lotes 71](#_Toc208848706)

[**Quadro 56** - Tabela de Produtos 72](#_Toc208848707)

[**Quadro 57 -** Tabela de Transações Financeiras 72](#_Toc208848708)

[**Quadro 58 -** Tabela de Sensores 72](#_Toc208848709)

[**Quadro 59 -** Tabela de Leitura de Sensores 73](#_Toc208848710)

[Quadro 60 - Tabela de Consumo de Água 73](#_Toc208848711)

[Quadro 61 - Tabela de Consumo de Ração 73](#_Toc208848712)

[Quadro 62 - Tabela de Eventos Sanitários 74](#_Toc208848713)

[Quadro 63 - Tabelas de Medições de Qualidade de Ar 74](#_Toc208848714)

[Quadro 64 - Tabela de Pesagens Semanais 75](#_Toc208848715)

[Quadro 65 - Tabela de Registros de Abate 75](#_Toc208848716)

[Quadro 66 - Tabela de Registro de Mortalidade 76](#_Toc208848717)

SUMÁRIO

[1 INTRODUÇÃO 5](#_Toc208494024)

[2 LEVANTAMENTO DE REQUISITOS DE SOFTWARE 7](#_Toc208494025)

[2.1 Descrição dos objetivos do sistema 7](#_Toc208494026)

[2.2 Descrição do sistema atual 8](#_Toc208494027)

[2.3 Análise de Sistemas Existentes 9](#_Toc208494028)

[2.4 Descrição dos principais problema 11](#_Toc208494029)

[2.5 Descrição dos requisitos funcionais 12](#_Toc208494030)

[2.6 Descrição dos requisitos não funcionais 13](#_Toc208494031)

[3 VISÃO DE CASO DE USO – UML 15](#_Toc208494032)

[3.1 Diagrama de Classes 15](#_Toc208494033)

[3.2 Dicionário de classes 17](#_Toc208494034)

[3.3 Definição dos Atores 26](#_Toc208494035)

[3.4 Lista de Casos de Uso 28](#_Toc208494038)

[3.5 Diagramas de Casos de Uso 33](#_Toc208494039)

[3.5.1 Diagrama de Casos de uso individuais 35](#_Toc208494040)

[3.6. Diagrama de Sequência 54](#_Toc208494041)

[3.7. Diagrama de Comunicação 55](#_Toc208494042)

[3.8. Diagrama de Atividade 55](#_Toc208494043)

[3.9 Diagrama de Máquina de Estados 55](#_Toc208494044)

[4 DEFINIÇÃO DA INTERFACE COM O USUÁRIO (UX) 56](#_Toc208494045)

[4.1 Descrição de cenário 56](#_Toc208494046)

[4.2 Descrição de personas 58](#_Toc208494047)

[4.3 Esboços de tela (wireframes) 60](#_Toc208494048)

[4.4 Protótipos de tela 64](#_Toc208494049)

[4.5 Acessibilidade 68](#_Toc208494052)

[5 BANCO DE DADOS 71](#_Toc208494053)

[5.1 Modelo Entidade Relacionamento 72](#_Toc208494054)

[5.2 Script das tabelas 72](#_Toc208494055)

[5.3 Mapeamento Objeto Relacional – ORM 76](#_Toc208494056)

[6 ARQUITETURA DE SOFTWARE 78](#_Toc208494057)

[6.1 Arquitetura de desenvolvimento 78](#_Toc208494058)

[6.1.1 Back-End..........................................................................................................................78](#_Toc208494059)

[6.1.2 Front-End - Web 78](#_Toc208494060)

[6.1.3 Front-End - Mobile 78](#_Toc208494061)

[6.1.4 Serviços de Mensageria 78](#_Toc208494062)

[6.2 Segurança da informação 79](#_Toc208494063)

[6.3 Implantação .......................................................................................................................79](#_Toc208494064)

[7 CONCLUSÃO 80](#_Toc208494065)

[8 REFERÊNCIAS 81](#_Toc208494066)

# INTRODUÇÃO

No âmbito da produção agropecuária, o gerenciamento eficiente das atividades produtivas e sanitárias de granjas se apresenta como uma ferramenta essencial, sobretudo para pequenos e médios produtores que necessitam monitorar, registrar, analisar e controlar seus processos de forma eficaz. Essas soluções desempenham um papel fundamental na modernização da gestão rural, permitindo maior visibilidade sobre indicadores produtivos, melhorando a tomada de decisões e estabelecendo um controle sólido da produção.

De acordo com a EMBRAPA (2024), “a adoção de tecnologias digitais na avicultura contribui diretamente para a eficiência produtiva, a sustentabilidade e a competitividade do setor”. Nesse contexto, torna-se evidente que a ausência de um sistema de gestão integrado pode representar um risco significativo para a rentabilidade e continuidade das atividades produtivas.

A legislação brasileira, em especial a Lei nº 13.288/2016 (Brasil, 2016), que dispõe sobre a defesa agropecuária, destaca a importância do monitoramento e da rastreabilidade na produção animal, assegurando padrões de qualidade e biossegurança. Assim, o acompanhamento adequado das condições ambientais, sanitárias e produtivas de uma granja se mostra indispensável para atender às exigências legais, de mercado e de sustentabilidade.

O controle das atividades de uma granja envolve aspectos como o registro de lotes, a alimentação das aves, o acompanhamento de mortalidade, a gestão sanitária, o controle financeiro e a análise de indicadores de desempenho. No entanto, a execução manual desses processos, muitas vezes por meio de anotações físicas ou planilhas, torna-se suscetível a erros, perda de informações e ineficiência no uso dos recursos.

Em um cenário no qual diversos produtores enfrentam o desafio de manter a eficiência e a sustentabilidade de suas granjas, torna-se crucial adotar soluções tecnológicas que integrem dados e centralizem informações estratégicas. A ausência desse controle pode resultar em perdas econômicas, impactos ambientais e dificuldades na gestão zootécnica.

Com o objetivo de auxiliar os produtores na superação desses desafios, o presente trabalho propõe o desenvolvimento do GranjaTech, um sistema web voltado para a gestão e monitoramento de granjas. O software oferecerá recursos como registro e controle de lotes, consumo de insumos, acompanhamento sanitário, indicadores produtivos, gestão financeira e monitoramento ambiental integrado a sensores. Dessa forma, busca-se promover maior eficiência, transparência e sustentabilidade na administração das propriedades avícolas.

# LEVANTAMENTO DE REQUISITOS DE SOFTWARE

O **Levantamento de Requisitos de Software** é a etapa inicial do processo de desenvolvimento em que são identificadas, coletadas, documentadas e validadas as necessidades e expectativas dos usuários e demais partes interessadas em relação ao sistema. Essa fase visa compreender profundamente o problema a ser resolvido, o contexto do cliente, as restrições envolvidas e as funcionalidades desejadas.

Os requisitos são classificados, geralmente, em **funcionais** (o que o sistema deve fazer, como cadastro de usuários ou geração de relatórios) e **não funcionais** (como o sistema deve se comportar, incluindo desempenho, usabilidade, segurança e disponibilidade). Além disso, essa fase pode envolver a descrição do sistema atual, análise de sistemas existentes, entrevistas, questionários, observações e prototipação.

Um levantamento de requisitos bem executado é essencial para garantir que o software atenda aos objetivos do projeto, evitando retrabalho, falhas de comunicação e funcionalidades desnecessárias. Ele também serve como base para todas as etapas posteriores de modelagem, projeto, testes e validação do sistema.

## Descrição dos objetivos do sistema

Atualmente, a gestão das atividades dentro das granjas é conduzida de maneira majoritariamente manual, sendo realizada, em sua maioria, por meio de registros em cadernos, anotações avulsas e planilhas eletrônicas, como o Microsoft Excel. Esse modelo, embora simples e de baixo custo, apresenta uma série de limitações que impactam diretamente a eficiência, a confiabilidade e a segurança dos dados registrados.

O acompanhamento das condições ambientais, como temperatura e umidade dos galpões, bem como o controle de consumo de ração, registros de mortalidade, entrada e saída de lotes, ocorre de forma visual e manual, exigindo que o produtor esteja constantemente presente na propriedade para realizar as verificações necessárias. Essa dependência da presença física do gestor, além de aumentar a carga de trabalho, compromete a agilidade na identificação de problemas críticos, como variações bruscas de temperatura, falhas nos sistemas de alimentação ou surgimento de doenças nos animais.

Ademais, o modelo atual não oferece mecanismos para armazenamento histórico dos dados, nem tampouco para análise preditiva, dificultando a realização de planejamentos estratégicos baseados em informações consistentes. A ausência de integração entre os diversos setores da granja, como a gestão produtiva, sanitária e financeira, resulta na fragmentação das informações, o que, por sua vez, prejudica tanto a eficiência operacional quanto a capacidade de avaliar a rentabilidade e a sustentabilidade do negócio.

Observa-se, ainda, que o controle financeiro é igualmente afetado, uma vez que a utilização de planilhas dispersas e registros manuais dificulta o acompanhamento preciso dos custos fixos e variáveis, bem como a correta apuração das receitas, lucros e indicadores econômicos da granja. Tal cenário gera insegurança na tomada de decisões, além de elevar o risco de perdas operacionais, desperdícios de recursos e, consequentemente, impactos negativos sobre a viabilidade econômica do empreendimento.

Diante desse contexto, torna-se evidente a necessidade de uma solução tecnológica que permita centralizar, automatizar e qualificar os processos de monitoramento, controle e gestão dentro das granjas. A proposta do sistema Granja Tech surge, portanto, como uma resposta a essa demanda, oferecendo uma plataforma que possibilita não apenas a digitalização dos processos, mas também a transformação dos dados em informações estratégicas, capazes de elevar significativamente os níveis de eficiência, produtividade e sustentabilidade das propriedades rurais.

## Descrição do sistema atual

Atualmente, é realizada de forma manual ou com o uso de planilhas simples, o que pode levar a erros humanos, falta de padronização e dificuldades na tomada de decisão. O monitoramento de temperatura, umidade, consumo de ração e condição dos animais é feito por inspeção visual, tornando o processo menos eficiente e sujeito a falhas.

O sistema atual, portanto, apresenta limitações significativas que comprometem a eficiência e a sustentabilidade da produção agropecuária.

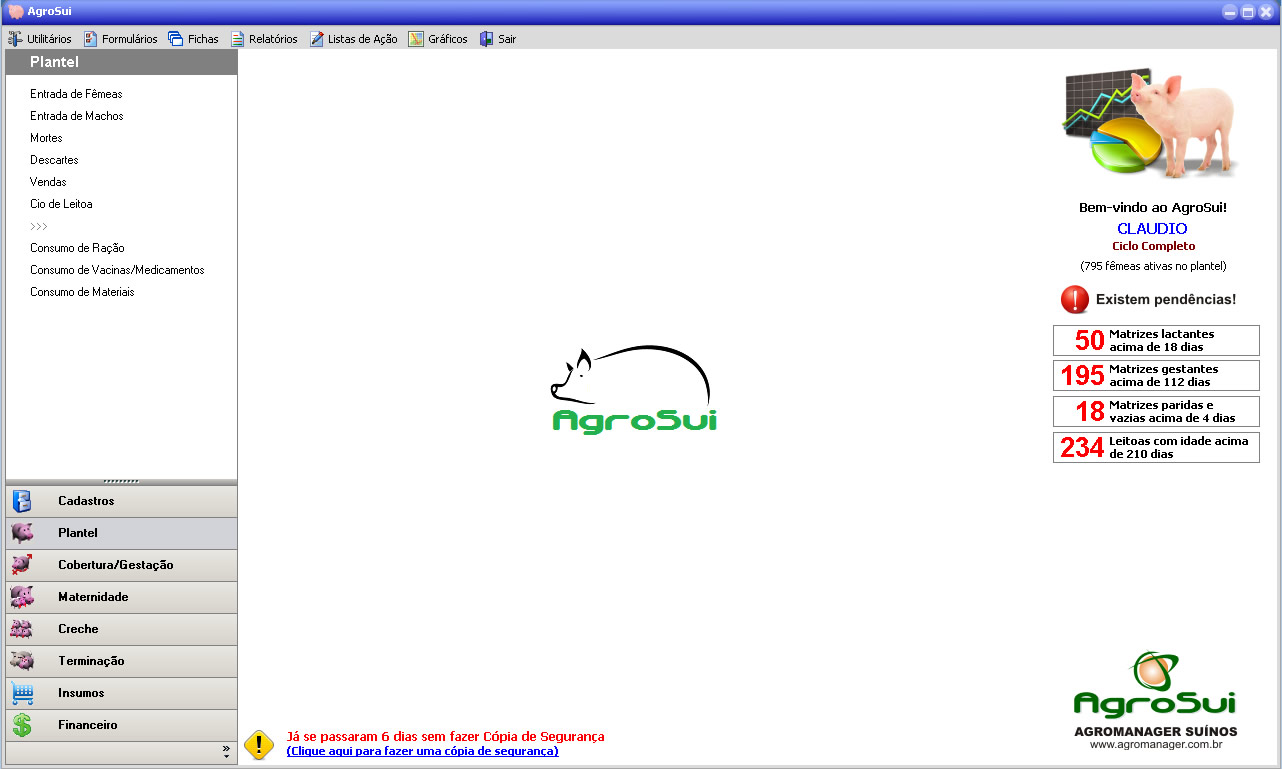
## Análise de Sistemas Existentes

Um sistema de monitoramento e gestão de granjas é uma solução tecnológica projetada para otimizar a produção agropecuária através do acompanhamento contínuo de diversos parâmetros ambientais e biológicos. Esse sistema visa coletar dados precisos e fornecer informações para auxiliar os produtores na tomada de decisão, buscando aumentar a eficiência, reduzir custos e melhorar a qualidade.

A implementação de sistemas de gestão e monitoramento é crucial para o setor agropecuária na atualidade, pois permite um controle mais preciso dos processos de produção. O monitoramento da temperatura, umidade, consumo de ração, saúde dos animais e outros fatores possibilita a identificação de problemas em tempo real, a otimização do uso de recursos e a prevenção de perdas (backup). Ao adotar essas tecnologias, os produtores podem aumentar a produtividade, reduzir o impacto ambiental e garantir a sustentabilidade de suas operações.

O AgroSui é um sistema de gestão específico para suinocultura, com funcionalidades para controle de produção, reprodução, alimentação, sanidade, custos e resultados. Permite o acompanhamento individualizado dos animais, o planejamento da produção e a análise de indicadores de desempenho.

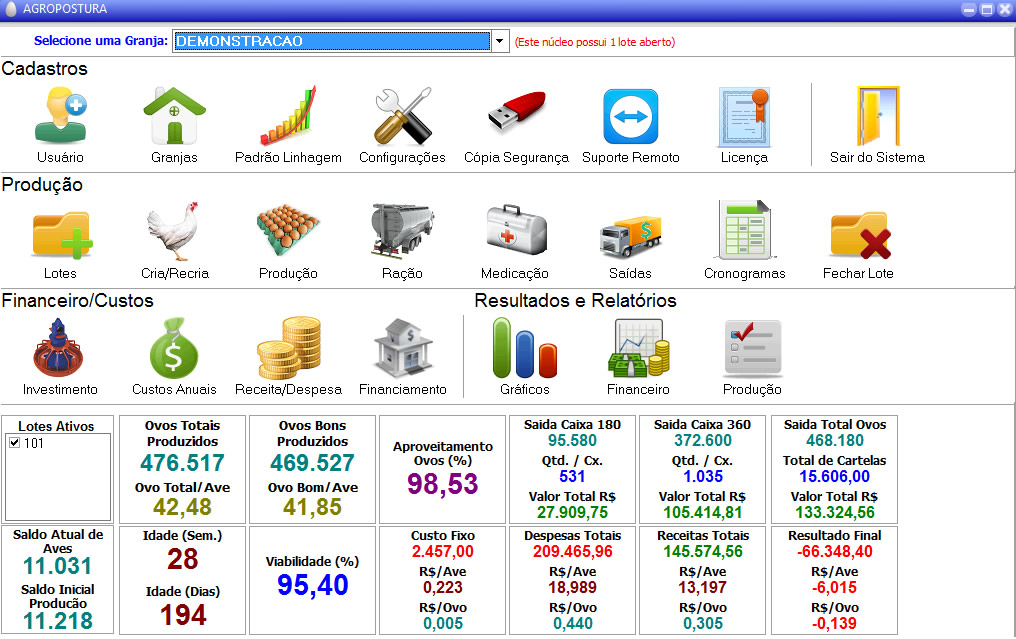
Figura 1 - AgroSui



**Fonte:** AGROSUI, 2025.

O AGROPOSTURA é um software para produtores de ovos comerciais, abrange as fases de cria/recria e produção de ovos. Permite o controle dos processos diários da granja bem como todos os custos envolvidos na produção de ovos. Conhecer o quanto se gasta para saber o quanto se ganha, é um dos benefícios da utilização deste sistema.

Figura 2 – AgroPostura



**Fonte:** AGROPOSTURA, 2025.

Figura 3 – Aegro

Fonte: AEGRO, 2025

O Aegro é uma plataforma de gestão agrícola que oferece diversas funcionalidades, incluindo o monitoramento da produção, o controle de estoque, o gerenciamento financeiro e a emissão de relatórios. Integrado com sensores e drones, permite monitoramento em tempo real das condições ambientais e da saúde das culturas.

## Descrição dos principais problema

A gestão manual ou baseada em planilhas nas granjas apresenta diversos desafios que comprometem a eficiência e a produtividade do negócio. A ausência de um monitoramento em tempo real faz com que os produtores dependam de inspeções manuais para detectar problemas como doenças nos animais ou falhas no fornecimento de alimentos, o que pode atrasar ações corretivas e impactar negativamente a produção. Além disso, o registro manual de informações aumenta as chances de erros humanos, como digitação incorreta, perda de dados e inconsistências no controle da produção.

Outro grande problema enfrentado é a falta de integração dos dados. Com informações espalhadas em diversas planilhas e anotações, torna-se difícil realizar análises precisas do desempenho da granja, no aspecto financeiro, a ausência de um sistema integrado prejudica o controle de custos e receitas, dificultando a previsão de gastos e investimentos e tornando a gestão econômica da granja menos eficiente.

Diante desses desafios, o Granja Tech surge como uma solução para eliminar essas deficiências, promovendo a automação e digitalização dos processos de gestão da granja, garantindo maior controle, precisão e eficiência na administração da produção agropecuária.

## Descrição dos requisitos funcionais

A Descrição dos Requisitos Funcionais detalha o que o sistema deve fazer, especificando todas as funcionalidades e comportamentos esperados, como permitir o cadastro de usuários ou gerar relatórios de vendas. Essa especificação é crucial para guiar o desenvolvimento do software, pois estabelece um contrato claro sobre as capacidades do produto final, alinha as expectativas de todas as partes interessadas (desenvolvedores, clientes e usuários) e serve de base para os testes, garantindo que o sistema atenda precisamente às necessidades e objetivos do projeto (PRESSMAN, 2010; SOMMERVILLE, 2011).

Quadro 1 - Requisitos Funcionais

|  |  |  |
| --- | --- | --- |
| N° | Requisitos Funcionais | Descrição |
| 1 | Cadastro de  Usuários | O sistema deve ser capaz de cadastrar usuários com suas  informações. |
| 2 | Cadastro de  granjas e núcleos | Cadastrar diferentes unidades de produção (granja, gado, etc..) |
| 3 | Gerenciamento de  lotes | Registro e monitoramento de lotes ativos e o histórico deles |
| 4 | Monitoramento de  mortalidade | Controle de taxas de mortalidade, possíveis causas e a prevenção futura |
| 5 | Controle de ração | Registro e monitoramento de consumo diário da ração por lote |
| 6 | Monitoramento de retirada das aves | O registro e monitoramento da saída de aves |
| 7 | Fechamento de lote | Realiza um cálculo da produtividade final ao encerrar uma granja |
| 8 | Administração de  Medicamento e vacinas | Registro do histórico de vacinas e medicações aplicadas ao decorrer da vida do animal |
| 9 | Coleta de dados ambientais | Realiza a coleta da temperatura, umidade e outros fatores ambientais que afetam a produção |
| 10 | Suporte remoto e atualizações | Permitir suporte técnico remoto, backup e atualizações do sistema |
| 11 | Gestão financeira | O controle de custos fixos e variáveis, a receita das despesas gerais da granja e relatórios financeiros detalhados. |
| 12 | Relatórios e dashboards | Indicadores técnicos de desempenho, análises financeiras e produtivas e viabilidade em % do lote e do negócio. |
| 13 | Backup e segurança | O sistema deve gerar backups para evitar a perda de dados e ter a segurança dos dados do usuário para evitar vazamentos |

**Fonte**: Elaborada pelos autores.

## Descrição dos requisitos não funcionais

A Descrição dos Requisitos Não Funcionais define como o sistema deve se comportar, abrangendo atributos de qualidade como desempenho, usabilidade, segurança e manutenibilidade, em vez das funcionalidades diretas; ela é fundamental para garantir a qualidade, a robustez e a aceitação do software, influenciando as decisões arquiteturais e tecnológicas e servindo como base para a avaliação da eficácia do sistema (PRESSMAN, 2010; SOMMERVILLE, 2011).

Quadro 2 - Requisitos Não Funcionais

|  |  |  |
| --- | --- | --- |
| N° | Requisitos Não Funcionais | Descrição |
| 1 | Usabilidade | O sistema deve possuir uma interface intuitiva e de fácil navegação, permitindo que usuários com pouca experiência em tecnologia possam utilizá-lo sem dificuldades. |
| 2 | Desempenho | O software deve responder rapidamente às interações do usuário e processar grandes volumes de dados sem comprometer sua performance. |
| 3 | Segurança | O sistema deve garantir a proteção dos dados dos usuários por meio de criptografia, autenticação de usuários e backups periódicos |
| 4 | Escalabilidade | O sistema deve ser capaz de crescer conforme o número de usuários e dados aumenta, sem comprometer sua funcionalidade. |
| 5 | Compatibilidade | O software deve ser acessível em diferentes dispositivos, incluindo computadores, tablets e smartphones, e ser compatível com os principais navegadores web. |
| 6 | Disponibilidade | O sistema deve garantir alta disponibilidade, minimizando o tempo de inatividade e permitindo o acesso 24/7. |
| 7 | Manutenção | O software deve ser modular e de fácil manutenção, permitindo atualizações e melhorias sem impactar a usabilidade. |
| 8 | Conformidade | O sistema deve estar em conformidade com legislações e padrões do setor agropecuário, garantindo a adequação às normas vigentes. |

**Fonte**: Elaborada pelos autores.

# VISÃO DE CASO DE USO – UML

A **Visão de Caso de Uso – UML** tem como objetivo apresentar, de maneira estruturada e visual, as funcionalidades do sistema sob a perspectiva dos usuários (atores). Por meio dos **casos de uso**, são descritas as interações entre os atores e o sistema, evidenciando os comportamentos esperados e os limites do sistema (escopo). Esta visão é fundamental para compreender o que o sistema deve fazer, servindo de base para a modelagem, análise, projeto e testes.

Os casos de uso são representados através de **diagramas UML**, que ilustram graficamente os relacionamentos entre os atores e os casos de uso, além de complementar com descrições textuais de fluxos normais e alternativos. Esta abordagem auxilia na validação dos requisitos funcionais e na comunicação entre desenvolvedores, analistas e partes interessadas.

O uso da UML segue padrões estabelecidos por autores clássicos da engenharia de software, como Pressman e Sommerville, garantindo uma documentação clara, compreensível e alinhada às boas práticas da modelagem orientada a objetos.

## 3.1 Diagrama de Classes

O diagrama de classe é uma representação do banco de dados em forma de tabela em UML, ele descreve toda a estrutura de um sistema, apresentando classes, atributos e operações entre as relações dos objetos. Pode-se dizer que o diagrama de classe é uma das partes mais importantes pois, como já foi citado acima, ele é responsável por descrever a estrutura de um sistema e apresentar classes, atributos e outros. (GUEDES, 2011).

O sistema GranjaTech é composto por diversas classes que representam as principais entidades e processos envolvidos na gestão de uma granja. A classe Granja centraliza as informações sobre a propriedade, armazenando dados como código, nome e localização, sendo fundamental para estruturar o gerenciamento dos lotes e da produção. Associada a ela, a classe **Lote** registra informações detalhadas sobre os grupos de aves, incluindo a quantidade inicial, quantidade atual, datas de entrada e saída e status, permitindo um controle eficiente do ciclo produtivo.

A produção é monitorada por classes específicas: ConsumoAgua e ConsumoRacao registram o consumo diário de água e ração; PesagemSemanal acompanha a evolução do peso médio e ganho semanal; RegistroMortalidade documenta perdas de animais e suas causas; RegistroAbate organiza as informações da saída final para o abate; EventoSanitario controla vacinas, medicamentos e tratamentos; e MedicaoQualidadeAr registra parâmetros ambientais, como amônia, CO₂, oxigênio, luminosidade, temperatura e umidade.

Para viabilizar o monitoramento automatizado, a classe **Sensor** cadastra os dispositivos utilizados, enquanto a classe LeituraSensor armazena os dados coletados, garantindo acompanhamento em tempo real das condições produtivas.

No campo administrativo, o sistema é estruturado pela classe Usuario, que contém atributos como código, nome, e-mail, senha e perfil, garantindo o controle de acesso. A classe Perfil define os tipos de usuário e suas permissões. A segurança das informações é reforçada pela classe LogAuditoria, que registra ações realizadas no sistema. O controle de insumos é feito pela classe Produto, e a gestão financeira é contemplada pela classe TransacaoFinanceira, que organiza custos e receitas associadas aos lotes e usuários. A classe FinanceiroProdutor estabelece a relação entre os usuários com perfil financeiro e os produtores da granja.

Dessa forma, o sistema busca modernizar a administração das propriedades avícolas, integrando aspectos produtivos, sanitários, ambientais e financeiros, reduzindo erros e aumentando a eficiência por meio da tecnologia.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 4 - Diagrama de classe

**Fonte:** Elaborado pelos autores

## 3.2 Dicionário de classes

O Dicionário de Classes é um artefato de documentação essencial no desenvolvimento de software, que consiste na formalização detalhada de todas as classes que compõem a arquitetura do sistema. Ele visa catalogar e descrever cada classe, especificando seu nome, responsabilidades, atributos (com seus respectivos tipos de dados), métodos (operações e comportamentos) e os relacionamentos com outras classes (como associações, heranças e agregações). Essa documentação é crucial para promover a clareza e a consistência na compreensão da estrutura do software, facilitando a comunicação entre os membros da equipe e servindo como uma referência vital para o desenvolvimento, manutenção e futuras expansões do projeto (GAMMA et al., 1995; BOOCH; RUMBAUGH; JACOBSON, 1999).

A classe Perfil é responsável por definir os diferentes tipos de perfis de acesso dentro do sistema, como administrador, produtor e financeiro. Essa classe armazena informações básicas sobre cada perfil, garantindo que os usuários tenham permissões adequadas às suas funções.  
Exemplo de uso: Diferenciar as permissões de um administrador, que pode cadastrar usuários, de um produtor, que pode registrar mortalidade ou consumo.

Quadro 3 - Descrição Classe Perfil

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | |  | | --- | | Identificador do perfil. |  |  | | --- | |  | |
| Nome | string | Nome do perfil de acesso. |

**Fonte:** Elaborado pelos autores

A classe Usuario gerencia as informações dos usuários do sistema, incluindo código, nome, e-mail e senha, além do perfil ao qual está vinculado. Essa classe garante o controle de autenticação e autorização no acesso às funcionalidades.  
Exemplo de uso: Um usuário com perfil de administrador acessa o módulo de auditoria, enquanto um usuário produtor acessa apenas as funções de lote e consumo.

Quadro 4 - Descrição Classe Usuario

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | |  | | --- | | Identificador do usuário. |  |  | | --- | |  | |
| Codigo | string | Código interno. |
| Nome | string | |  | | --- | | Nome completo. |  |  | | --- | |  | |
| Email | string | E-mail de acesso. |
| senhaHash | string | |  | | --- | | Hash da senha. |  |  | | --- | |  | |
| PerfilId | int | FK para Perfil. |

**Fonte:** Elaborado pelos autores

A classe Granja centraliza os dados sobre a propriedade, armazenando código, nome e localização. Está associada a lotes, produtos e sensores, servindo como estrutura de organização das informações produtivas.  
Exemplo de uso: Registrar uma nova granja vinculada a um usuário administrador, permitindo que seus lotes e sensores sejam monitorados.

Quadro 5 - Descrição Classe Granja

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | |  | | --- | | Identificador da granja. |  |  | | --- | |  | |
| Codigo | string | Código da granja. |
| Nome | int | |  | | --- | | Nome da granja. |  |  | | --- | |  | |
| Localizacao | float | |  | | --- | |  |  |  | | --- | | Endereço/localização. | |
| UsuarioId | float | Responsável (FK Usuario). |

**Fonte:** Elaborado pelos autores

A classe Lote é o núcleo do sistema, representando os grupos de aves presentes na granja. Contém informações sobre código, identificador, datas de entrada e saída, quantidade inicial e atual de aves, além do status do lote.  
Exemplo de uso: Acompanhar a evolução de um lote desde a entrada dos pintinhos até o encerramento, integrando consumo, pesagens, eventos sanitários e abates.

Quadro 6 - Descrição Classe Lote

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | |  | | --- | | Identificador do lote. |  |  | | --- | |  | |
| Codigo | String | Código do lote. |
| Identificador | String | Identificador legível. |
| DataEntrada | datetime | Entrada das aves. |
| DataSaida | datetime? | Saída/encerramento. |
| QuantidadeAvesInicial | Int | Qtde inicial. |
| QuantidadeAvesAtual | Int | Qtde atual. |
| Status | String | Situação do lote. |
| GranjaId | int | FK para Granja |

**Fonte:** Elaborado pelos autores

A classe Produto registra os insumos utilizados na produção, como rações, medicamentos e materiais diversos, controlando nome, tipo, quantidade e unidade de medida.  
Exemplo de uso: Manter controle do estoque de ração para verificar a disponibilidade e planejar novas compras.

Quadro 7 - Descrição Classe Produto

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | |  | | --- | |  |  |  | | --- | | Identificador. | |
| Nome | int | Nome do produto/insumo. |
| Tipo | String | Categoria do produto. |
| Quantidade | decimal | Quantidade em estoque. |
| Unidade de Medida | String | Unidade (kg, un, L) |
| GranjaId | Int | FK para Granja |

**Fonte:** Elaborado pelos autores

A classe TransacaoFinanceira gerencia receitas e despesas associadas a usuários e, opcionalmente, a lotes específicos. Registra descrições, valores, tipo (receita ou despesa) e data.  
Exemplo de uso: Registrar o custo da compra de ração vinculado a um lote ou lançar a receita de venda das aves abatidas.

Quadro 8 - Descrição Classe TransacaoFinanceira

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | Código identificador da granja. |
| Descricao | String | Nome da granja. |
| Valor | String | Localização geográfica da granja. |
| Tipo | String | Nome ou número do galpão da granja. |
| Data | Datetime | Data de lançamento |
| UsuarioId | Int | Autor (FK Usuario) |
| LoteId | Int? | Lote relacionado (Opcional |

**Fonte:** Elaborado pelos autores

A classe Sensor representa os dispositivos de monitoramento instalados na granja, armazenando tipo (temperatura, umidade, etc.) e identificador único.  
Exemplo de uso: Cadastrar um sensor de temperatura em determinado galpão.

Quadro 9 - Descrição Classe Sensor

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | Identificador do sensor. |
| Tipo | string | Tipo (temp., umid., etc.). |
| IdentificadorUnico | string | |  | | --- | |  |  |  | | --- | | ID físico/lógico. | |
| GranjaId | int | FK para Granja |

**Fonte:** Elaborado pelos autores

A classe LeituraSensor registra os dados coletados pelos sensores, armazenando valor e momento da medição.  
Exemplo de uso: Registrar automaticamente leituras de temperatura e umidade do ambiente em tempo real.

Quadro 10 – Descrição Classe LeituraSensor

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | Identificador. |
| Valor | Decimal | |  | | --- | |  |  |  | | --- | | Valor lido. | |
| Timestamp | Datetime | Momento da leitura. |
| SensorId | int | FK para Sensor. |

**Fonte:** Elaborado pelos autores

A classe ConsumoAgua controla o registro diário de água consumida pelos lotes, incluindo quantidade em litros, aves vivas, temperatura ambiente e observações.  
Exemplo de uso: Identificar alterações anormais no consumo de água que possam indicar problemas de saúde nas aves.

Quadro 11 – Descrição Classe ConsumoAgua

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | int | Identificador. |
| LoteId | int | FK para Lote. |
| Data | Datetime | Data do registro. |
| QuantidadeLitros | Decimal | Litros consumidos. |
| AvesVivas | Int | |  | | --- | |  |  |  | | --- | | Aves vivas no dia. | |
| TemperaturaAmbiente | Decimal | Temperatura ambiente. |
| Observacoes | String | Observações. |
| DataCriacao | Datetime | Carimbo de criação. |

**Fonte:** Elaborado pelos autores

A classe ConsumoRacao registra diariamente a quantidade de ração consumida por lote, tipo de ração, aves vivas e observações.  
Exemplo de uso: Calcular o consumo médio de ração por ave em um período.

Quadro 12 – Descrição Classe ConsumoRacao

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | Int | Identificador. |
| LoteId | Int | FK para Lote. |
| Data | Datetime | Data do registro. |
| QuantidadeKg | Decimal | Kg consumidos. |
| TipoRacao | String | Tipo de ração. |
| AvesVivas | Int | Aves vivas no dia. |
| Observacoes | String¿ | Observações |
| DataCriacao | Datetime | Carimbo de criação |

**Fonte:** Elaborado pelos autores

A classe EventoSanitario gerencia eventos relacionados à saúde das aves, como vacinas, medicamentos e tratamentos. Armazena produto, dosagem, via de aplicação, sintomas e responsável.  
Exemplo de uso: Registrar a aplicação de uma vacina em determinado lote e controlar o período de carência.

Quadro 13 – Descrição Classe EventoSanitario

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | Int | Identificador. |
| LoteId | int | FK para Lote. |
| Data | Datetime | Data do evento. |
| TipoEvento | String | Vacina, medicação, etc. |
| Produto | String | Produto aplicado. |
| LoteProduto | String | Lote do produto. |
| Dosagem | String | Dosagem. |
| ViaAdministracao | String | Via de aplicação. |
| AvesTratadas | Int | Quantidade tratada. |
| DuracaoTratamentoDias | Int | Duração. |
| PeriodoCarenciaDias | Int | Carência. |
| ResponsavelAplicacao | String | Responsável. |
| Sintomas | String | Sintomas observados. |
| Observacoes | String | Observações. |
| Custo | Decimal | Custo do evento. |
| DataCriacao | datetime | Carimbo de criação. |

**Fonte:** Elaborado pelos autores

A classe MedicaoQualidadeAr monitora a qualidade do ar dentro dos galpões, registrando parâmetros como amônia, CO₂, oxigênio, luminosidade, temperatura e umidade.  
Exemplo de uso: Identificar níveis elevados de amônia que podem afetar a saúde das aves e acionar medidas corretivas.

Quadro 14 - Descrição Classe MedicaoQualidadeAr

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | Int | Identificador. |
| LoteId | Int | FK para Lote. |
| DataHora | Datetime | Momento da medição. |
| NH3\_ppm | Decimal | Amônia. |
| CO2\_ppm | Decimal | Dióxido de carbono. |
| O2\_percentual | Decimal | Oxigênio. |
| VelocidadeAr\_ms | Decimal | Velocidade do ar. |
| Luminosidade\_lux | Decimal | Luminosidade. |
| TemperaturaAr | Decimal | Temperatura. |
| UmidadeRelativa | Decimal | Umidade. |
| LocalMedicao | String | Local interno. |
| EquipamentoMedicao | String | Equipamento. |
| Observacoes | String | Observações. |
| DataCriacao | datetime | Carimbo de criação. |

**Fonte:** Elaborado pelos autores

A classe PesagemSemanal registra o acompanhamento do peso médio das aves semanalmente, incluindo idade, peso mínimo, máximo, desvio padrão e ganho semanal.  
Exemplo de uso: Comparar o ganho de peso semanal de um lote em relação a padrões esperados.

Quadro 15 - Descrição Classe PesagemSemanal

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | Int | Identificador |
| LoteId | Int | FK para Lote |
| DataPesagem | Datetime | Data da pesagem |
| IdadeDias | Int | Idade das aves |
| SemanaVida | Int | Semana de vida |
| PesoMedioGramas | Decimal | Peso médio |
| QuantidadeAmostrada | Int | Nº de aves na amostra |
| PesoMinimo | Decimal | Mínimo |
| PesoMaximo | Decimal | Máximo |
| DesvioPadrao | Decimal | Desvio padrão |
| CoeficienteVariacao | Decimal | CV |
| GanhoSemanal | Decimal | Ganho de peso |
| Observacoes | String | Observações |
| DataCriacao | datetime | Carimbo de criação |

**Fonte:** Elaborado pelos autores

A classe RegistroAbate armazena informações sobre a saída final de aves para abate, como quantidade enviada, peso total, aves condenadas, destino e valor recebido.  
Exemplo de uso: Registrar a venda de um lote para um frigorífico, calculando o lucro obtido.

Quadro 16 - Descrição Classe RegistroAbate

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | Int | Identificador |
| LoteId | Int | FK para Lote |
| DataAbate | Datetime | Data do abate |
| DataAbatePrevista | Datetime | Previsão |
| IdadeAbateDias | Int | Idade ao abate |
| QuantidadeEnviada | Int | Aves enviadas |
| PesoVivoTotalKg | Decimal | Peso vivo total |
| PesoCarcacaTotalKg | Decimal | Peso carcaça |
| AvesCondenadas | Int | Nº condenadas |
| MotivoCondenacoes | String | Motivos |
| PesoCondenadoKg | Decimal | Peso condenado |
| FrigorificoDestino | String | Destino |
| Transportadora | String | Transportadora |
| ValorPorKg | Decimal | Preço por kg |
| ValorTotalRecebido | Decimal | Valor total |
| Observacoes | String | Observações |
| DataCriacao | datetime | Carimbo de criação |

**Fonte:** Elaborado pelos autores

A classe RegistroMortalidade documenta as perdas de aves em um lote, registrando data, quantidade, motivo, setor, responsável e observações.  
Exemplo de uso: Acompanhar as causas de mortalidade em diferentes setores para identificar padrões e prevenir ocorrências futuras.

Quadro 17 - Descrição Classe RegistroMortalidade

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | Int | Identificador |
| LoteId | Int | FK para Lote |
| Data | Datetime | Data do evento |
| Quantidade | Int | Qtde de aves mortas |
| Motivo | String | Causa principal |
| Setor | String | Setor/galpão |
| ResponsavelRegistro | String | Responsável |
| Observacoes | string | Observações |

**Fonte:** Elaborado pelos autores

A classe LogAuditoria Armazena informações sobre as ações realizadas por usuários, como criação, edição ou exclusão de registros, garantindo rastreabilidade.  
Exemplo de uso: Verificar quem registrou uma determinada transação financeira e em qual horário.

Quadro 18 - Descrição Classe LogAuditoria

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| Id | Int | Identificador do log |
| TimeStamp | Datetime | Momento da ação |
| UsuarioId | Int | Usuário (FK) |
| UsuarioEmail | String | E-mail do usuário |
| Acao | String | Ação executada |
| Detalhes | string | Detalhes do evento |

**Fonte:** Elaborado pelos autores

A classe FinanceiroProdutor relaciona usuários com função financeira aos produtores, estabelecendo vínculos de responsabilidade entre diferentes perfis.  
Exemplo de uso: Associar um usuário do setor financeiro responsável pelo acompanhamento de transações a um produtor específico.

Quadro 19 - Descrição Classe FinanceiroProdutor

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Tipo** | **Descrição** |
| FinanceiroId | Int | Usuário financeiro (FK) |
| ProdutorId | int | Usuário produtor (FK) |

**Fonte:** Elaborado pelos autores

## 3.3 Definição dos Atores

A Definição dos Atores é o processo de identificar e descrever todas as entidades, humanas ou sistemas externos, que interagem diretamente com o software, representando seus papéis e objetivos ao usar o sistema. Isso é fundamental para compreender o escopo do projeto, mapear as funcionalidades necessárias sob a perspectiva de quem as utilizará e garantir que o sistema atenda às necessidades de todos os seus usuários (PRESSMAN, 2010; SOMMERVILLE, 2011).

Quadro 20 - Definição dos Atores

|  |  |
| --- | --- |
| **Ator** | **Descrição** |
| **Administrador** | Gerencia o sistema como um todo. Acessa todos os módulos (usuários, perfis, granjas, lotes, sensores, consumos, pesagens, sanitário, mortalidade, abates, financeiro e relatórios). Realiza cadastros, edições, exclusões e auditoria. |
| **Sistema** | Plataforma web e APIs que armazenam dados, executam regras de negócio, integram sensores e geram relatórios/dashboards. Provê autenticação, autorização e trilhas de auditoria. |
| **Usuário** | Qualquer pessoa autenticada que interage com o sistema. Recebe permissões conforme o Perfil associado (Administrador, Produtor, Financeiro). |
| **Produtor** | Responsável operacional da granja. Registra e consulta lotes, consumo de água/ração, pesagens, eventos sanitários, mortalidade, medições ambientais e abates. Visualiza indicadores produtivos. |
| **Financeiro** | Responsável por lançamentos e análises financeiras. Registra receitas e despesas (por usuário e opcionalmente por lote), emite relatórios, acompanha custos e lucratividade, e cruza dados produtivos x financeiros. |

**Fonte:** Elaborada pelos autores.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 5 – Atores do Sistema

Fonte: Elaborada pelos autores.

O ator Usuário (Figura 5) representa coletivamente todas as pessoas que interagem com o sistema GranjaTech. Ele abrange diferentes papéis, como Administrador, Produtor e Financeiro, reunindo as interações possíveis com o sistema em um nível mais geral. A partir dele, são definidas as especializações que permitem separar permissões e responsabilidades de acordo com as funções desempenhadas dentro da granja.

O ator Administrador (Figura 5) desempenha o papel central na gestão do sistema. Suas responsabilidades incluem o cadastro e manutenção de usuários, perfis, granjas e lotes, além do acesso total a todos os módulos disponíveis, como consumo, pesagens, registros sanitários, mortalidade, abates e financeiro. O Administrador é responsável por configurar o sistema, supervisionar relatórios e garantir que a operação esteja de acordo com os objetivos da granja, assegurando eficiência, segurança e integridade dos dados.

O ator Produtor (Figura 5) representa os responsáveis operacionais pela granja, com permissões voltadas ao registro e acompanhamento das atividades produtivas. Suas funções incluem cadastrar e consultar dados de lotes, registrar consumos de água e ração, efetuar pesagens semanais, lançar eventos sanitários e mortalidade, além de acompanhar os registros de abate. O Produtor atua diretamente na rotina diária da criação, garantindo que as informações da produção estejam atualizadas no sistema.

O ator Financeiro (Figura 5) corresponde aos usuários encarregados do controle econômico da granja. Ele gerencia as transações financeiras relacionadas a custos, despesas e receitas, gera relatórios de viabilidade econômica e acompanha a lucratividade por lote ou por período. Sua atuação é restrita ao módulo financeiro, mas fundamental para auxiliar na tomada de decisão estratégica, permitindo que a gestão produtiva seja acompanhada sob a ótica de custos e resultados econômicos.

Por fim, o ator Sistema (Figura 5) refere-se à própria aplicação do GranjaTech. Ele não representa um usuário humano, mas sim o conjunto de funcionalidades e processos automatizados, responsáveis por armazenar os dados, executar regras de negócio, integrar sensores ambientais e gerar relatórios e dashboards. O Sistema serve de interface para que os diferentes atores possam interagir, garantindo a consistência das operações e a integração entre os módulos produtivos, sanitários, financeiros e administrativos.



## 3.4 Lista de Casos de Uso

A Lista de Casos de Uso apresenta de forma estruturada todas as interações possíveis entre os atores identificados e o sistema proposto. Cada item da lista representa uma funcionalidade ou comportamento relevante que o sistema deve oferecer a seus usuários, conforme os requisitos levantados. Essa listagem é fundamental para o entendimento das capacidades do sistema, servindo como base para a modelagem detalhada dos fluxos de operação, elaboração dos diagramas de casos de uso e planejamento dos testes de validação. Além disso, permite visualizar quais operações cada ator pode executar, como cadastrar usuários, gerar relatórios ou consultar informações operacionais e financeiras da granja.

A construção dessa lista segue os princípios da Engenharia de Requisitos e da modelagem orientada a objetos, conforme descrito em metodologias consagradas na literatura da engenharia de software (PRESSMAN, 2010; SOMMERVILLE, 2011).

Quadro 21 - Lista de Mensagens

| **Nº da Mensagem** | **Conteúdo da Mensagem** |
| --- | --- |
| Msg 1 | Operação realizada com sucesso. |
| Msg 2 | Registro excluído com sucesso. |
| Msg 3 | Dados atualizados com sucesso. |
| Msg 4 | Dados exibidos com sucesso. |
| Msg 5 | Login efetuado com sucesso. |
| Msg 6 | Erro: campos obrigatórios em branco. |
| Msg 7 | Erro ao conectar com o banco de dados. |
| Msg 8 | Registro já existente no sistema. |
| Msg 9 | Acesso concedido com sucesso. |
| Msg 10 | Relatório gerado com sucesso. |
| Msg 11 | Senha redefinida com sucesso. |
| Msg 12 | Permissão atribuída com sucesso. |
| Msg 13 | Solicitação registrada com sucesso. |
| Msg 14 | Histórico exibido com sucesso. |
| Msg 15 | Estimativa de produtividade realizada com sucesso |
| Msg 16 | Consumo de ração registrado com sucesso |
| Msg 17 | Consumo de água registrado com sucesso |
| Msg 18 | Mortalidade registrada com sucesso |
| Msg 19 | Evento sanitário registrado com sucesso |
| Msg 20 | Pesagem semanal registrada com sucesso |
| Msg 21 | Abate registrado com sucesso |
| Msg 22 | Sensor cadastrado com sucesso |
| Msg 23 | Leitura de sensor salva com sucesso |
| Msg 24 | Transação financeira registrada com sucesso |
| Msg 25 | Erro: operação não permitida para este perfil |

**Fonte**: Elaborada pelos autores.

Quadro 22 - Casos de Uso – Administrador

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nº** | **Descrição** | **Entrada** | **Caso de Uso** | **Saída** |
| 1 | Administrador Cadastra Usuário | Dados Usuário | Cadastrar Usuário | Msg 1 |
| 2 | Administrador atualiza usuário | Dados Usuário | Atualizar Usuário | Msg 3 |
| 3 | Administrador cadastra perfil | ID Usuário | Excluir Usuário | Msg 2 |
| 4 | Administrador Lista Usuários | Dados Perfil | Cadastrar Perfil | Msg 1 |
| 5 | Administrador atualiza perfil | Dados Perfil | Atualizar Perfil | Msg 3 |
| 6 | Administrador exclui perfil | ID Perfil | Excluir Perfil | Msg 2 |
| 7 | Administrador cadastra granja | Dados Granja | Cadastrar Granja | Msg 1 |
| 8 | Administrador atualiza granja | Dados Granja | Atualizar Granja | Msg 3 |
| 9 | Administrador exclui granja | |  | | --- | | ID Granja |  |  | | --- | |  | | Excluir Granja | Msg 2 |
| 10 | Administrador cadastra lote | Dados Lote | Cadastrar Lote | Msg 1 |
| 11 | Administrador atualiza lote | Dados Lote | Atualizar Lote | Msg 3 |
| 12 | Administrador encerra lote | |  | | --- | | ID Lote |  |  | | --- | |  | | Encerrar Lote | Msg 15 |
| 13 | Administrador exclui lote | ID Lote | Excluir Lote | Msg 2 |
| 14 | Administrador consulta relatórios gerais | Parâmetros Relatório | Consultar Relatório | Msg 10 |
| 15 | Administrador consulta auditoria | Dados Log | Consultar Auditoria | Msg 4 |

**Fonte**: Elaborada pelos autores.

Quadro 23 - Casos de Uso – Produtor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nº** | **Descrição do Caso de Uso** | **Entrada** | **Caso de Uso** | **Saída** |
| 16 | Produtor registra consumo de água | Dados Consumo Água | Registrar Consumo Água | Msg16 |
| 17 | Produtor atualiza consumo de água | Dados Consumo Água | Atualizar Consumo Água | Msg3 |
| 18 | Produtor exclui consumo de água | ID Consumo Água | Excluir Consumo Água | Msg2 |
| 19 | Produtor registra consumo de ração | Dados Consumo Ração | Registrar Consumo Ração | Msg17 |
| 20 | Produtor atualiza consumo de ração | Dados Consumo Ração | Atualizar Consumo Ração | Msg3 |
| 21 | Produtor exclui consumo de ração | ID Consumo Ração | Excluir Consumo Ração | Msg2 |
| 22 | Produtor registra mortalidade | Dados Mortalidade | Registrar Mortalidade | Msg18 |
| 23 | Produtor atualiza mortalidade | Dados Mortalidade | Atualizar Mortalidade | Msg3 |
| 24 | Produtor exclui mortalidade | ID Mortalidade | Excluir Mortalidade | Msg2 |
| 25 | Produtor registra evento sanitário | Dados Evento Sanitário | Registrar Evento Sanitário | Msg19 |
| 26 | Produtor atualiza evento sanitário | Dados Evento Sanitário | Atualizar Evento Sanitário | Msg3 |
| 27 | Produtor exclui evento sanitário | ID Evento | Excluir Evento Sanitário | Msg2 |
| 28 | Produtor registra pesagem semanal | Dados Pesagem | Registrar Pesagem | Msg20 |
| 29 | Produtor atualiza pesagem | Dados Pesagem | Atualizar Pesagem | Msg3 |
| 30 | Produtor exclui pesagem | ID Pesagem | Excluir Pesagem | Msg2 |
| 31 | Produtor registra abate | Dados Abate | Registrar Abate | Msg21 |
| 32 | Produtor atualiza abate | Dados Abate | Atualizar Abate | Msg3 |
| 33 | Produtor exclui abate | ID Abate | Excluir Abate | Msg2 |
| 34 | Produtor consulta indicadores produtivos | Parâmetros Indicadores | Consultar Indicadores | Msg4 |

**Fonte**: Elaborada pelos autores.

Quadro 24 - Casos de Uso – Financeiro

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nº** | **Descrição** | **Entrada** | **Caso de Uso** | **Saída** |
| 35 | Financeiro cadastra entrada/saída | Dados Financeiros | Cadastrar Entrada e Saída | Msg24 |
| 36 | Financeiro atualiza entrada/saída | Dados Financeiros | Atualizar Entrada e Saída | Msg3 |
| 37 | Financeiro exclui entrada/saída | ID Registro Financeiro | Excluir Entrada e Saída | Msg2 |
| 38 | Financeiro consulta receita total | Dados Receita | Consultar Receita Total | Msg4 |
| 39 | Financeiro calcula custo fixo | Dados Custos Fixos | Calcular Custo Fixo | Msg15 |
| 40 | Financeiro calcula lucro | Dados Receita e Custos | Calcular Lucro | Msg15 |
| 41 | Financeiro consulta valor do lote | ID Lote | Consultar Valor do Lote | Msg4 |
| 42 | Financeiro gera relatório | Período, Tipo | Gerar Relatório Financeiro | Msg10 |
| 43 | Financeiro consulta histórico | Intervalo de Data | Consultar Histórico Financeiro | Msg14 |
| 44 | Financeiro estima receita futura | Dados Anteriores | Estimar Receita Futura | Msg15 |

**Fonte**: Elaborada pelos autores.

Quadro 25 - Casos de uso – Sistema

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nº** | **Descrição** | **Entrada** | **Caso de Uso** | **Saída** |
| 45 | Sistema valida login | Credenciais | Validar Login | Msg5 |
| 46 | Sistema registra log de auditoria | Dados Ação | Registrar Auditoria | Msg1 |
| 47 | Sistema integra sensor | Dados Sensor | Integrar Sensor | Msg22 |
| 48 | Sistema registra leitura automática | Dados Leitura | Registrar Leitura | Msg23 |
| 49 | Sistema gera relatório automático | Parâmetros Predefinidos | Gerar Relatório Automático | Msg10 |
| 50 | Sistema realiza backup | Dados Sistema | Realizar Backup | Msg1 |
| 45 | Sistema valida login | Credenciais | Validar Login | Msg5 |
| 46 | Sistema registra log de auditoria | Dados Ação | Registrar Auditoria | Msg1 |
| 47 | Sistema integra sensor | Dados Sensor | Integrar Sensor | Msg22 |
| 48 | Sistema registra leitura automática | Dados Leitura | Registrar Leitura | Msg23 |

**Fonte**: Elaborada pelos autores.

## 3.5 Diagramas de Casos de Uso

O diagrama de Caso de Uso é um dos mais abstratos, flexíveis e informais da UML, utilizado para representar uma visão abrangente do sistema. Seu propósito é identificar os atores que interagem com o software, juntamente com suas respectivas funcionalidades. Este diagrama é amplamente empregado durante as fases de levantamento e análise de requisitos, mas sua aplicação se estende ao longo de todo o processo de modelagem, podendo inclusive servir como base para a elaboração de outros diagramas (GUEDES, 2018).

Nas subseções seguintes, serão apresentados os diagramas de caso de uso do GranjaTech, organizados de acordo com os diferentes atores que interagem com o sistema. Cada diagrama ilustra as interações e responsabilidades de um ator específico, detalhando como cada um contribui para o funcionamento geral do sistema de gestão de granjas.

O Administrador é o ator responsável por gerenciar o sistema de forma global. Suas atribuições incluem o cadastro e manutenção de usuários, perfis, granjas e lotes, além da definição de permissões e do monitoramento das atividades realizadas por meio da auditoria e dos relatórios. Esse ator possui acesso completo às funcionalidades e é essencial para a configuração e supervisão do ambiente.

O Produtor é o ator que realiza as tarefas operacionais dentro do sistema. Suas responsabilidades englobam o registro e a atualização de informações sobre o consumo de água e ração, a mortalidade das aves, os eventos sanitários, as pesagens semanais e os abates. Além disso, o produtor pode consultar indicadores produtivos, o que permite acompanhar a eficiência da criação e adotar medidas corretivas quando necessário.

O Financeiro é o ator responsável pelo controle econômico da granja. Ele registra entradas e saídas financeiras, consulta receitas, calcula custos fixos e variáveis, além de gerar relatórios e estimar a receita futura. Essas funcionalidades permitem ao financeiro acompanhar a viabilidade econômica da produção e apoiar decisões estratégicas relacionadas à lucratividade.

O Sistema representa a própria aplicação do GranjaTech. Embora não seja um usuário humano, esse ator é responsável por funcionalidades automáticas, como validar logins, registrar auditorias, integrar sensores, armazenar leituras ambientais e gerar relatórios de forma automática. Também garante a execução de rotinas de backup, assegurando a integridade e disponibilidade dos dados.

Dessa forma, os diagramas de caso de uso do GranjaTech evidenciam a interação entre atores e funcionalidades, oferecendo uma visão clara e organizada do papel de cada perfil e de como esses elementos se articulam para garantir o funcionamento eficiente da gestão da granja.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 6 - Diagrama Geral do Financeiro – Visão do Financeiro

**Fonte:** Elaborada pelos autores.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 7 - Diagrama Geral do Produtor – Visão do Produtor

**Fonte:** Elaborada pelos autores

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 8 — Diagrama Geral do Administrador – Visão do Administrador

**Fonte:** Elaborada pelos autores

## 3.5.1 Diagrama de Casos de uso individuais

Os diagramas de casos de uso são fundamentais para modelagem de estrutura de um sistema específico, que são associados aos usuários do software para suprir as funcionalidades de cada um no sistema desenvolvido. Entre os diagramas, o de classe de uso é o mais informal e geral da UML, mesmo assim acaba sendo um dos mais usados como ferramenta de consulta durante o processo de modelagem. (GUEDES, 2011).

Na figura 9, o ator Administrador interage com o sistema para listar lotes. A partir da listagem, o ator pode realizar outras ações, como cadastrar novo lote, alterar dados do lote, encerrar lote e excluir lote. O caso “Listar Lotes” também envolve a validação de login e o carregamento de dados do banco.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 9 – Diagrama de Caso de Uso Individual – Administrador Lista Lotes

**Fonte:** Elaborado pelos autores

Quadro 26 - Documentação - Administrador Lista Lotes

|  |  |
| --- | --- |
| Nome do caso de uso | Listar Lotes |
| Ator principal | Administrador |
| Resumo | Permite visualizar a lista de lotes com filtros e paginação. |
| Pré-condições | Administrador autenticado no sistema. Deve existir ao menos um lote cadastrado. |
| Pós-condições | Lista de lotes exibida. Sistema apresenta 'Dados exibidos com sucesso' (Msg4). |
| Fluxo Normal | 1. Administrador acessa a tela de Lotes. 2. Informa filtros opcionais (período, status, granja). 3. Sistema consulta o banco e exibe a lista. 4. Administrador pode selecionar um lote para ações relacionadas. |
| Fluxo alternativo | 1. Sem resultados para os filtros. 2. Sistema informa 'Nenhum lote encontrado'. |

**Fonte**: Elaborada pelos autores.

Quadro 27 - Documentação - Administrador Cadastrar Lote

|  |  |
| --- | --- |
| Nome do caso de uso | Cadastrar Lote |
| Ator principal | Administrador |
| Resumo | Registra um novo lote na granja. |
| Pré-condições | Administrador autenticado. Granja existente. |
| Pós-condições | Lote criado e disponível para consultas e operações. Sistema apresenta 'Operação realizada com sucesso' (Msg1). |
| Fluxo Normal | 1. Administrador aciona 'Novo Lote'. 2. Informa dados obrigatórios (código, identificador, granja, data de entrada, quantidade inicial). 3. Sistema valida campos obrigatórios. 4. Sistema grava o lote. 5. Sistema confirma o cadastro. |
| Fluxo alternativo | 1. Campos obrigatórios ausentes ou duplicidade de código. 2. Sistema informa erro (Msg6/Msg8) e mantém na tela para correção. |

**Fonte**: Elaborada pelos autores.

Quadro 28 - Documentação - Administrador Alterar Lote

|  |  |
| --- | --- |
| Nome do caso de uso | Alterar Lote |
| Ator principal | Administrador |
| Resumo | Atualiza dados de um lote existente. |
| Pré-condições | Administrador autenticado. Lote existente e selecionado. |
| Pós-condições | Dados do lote atualizados. Sistema apresenta 'Dados atualizados com sucesso' (Msg3). |
| Fluxo Normal | 1. Administrador seleciona um lote da lista. 2. Aciona 'Editar'. 3. Altera os campos desejados. 4. Sistema valida e salva as alterações. 5. Sistema confirma a atualização. |
| Fluxo alternativo | 1. Lote inexistente ou inválido. 2. Sistema informa erro e retorna à listagem. |

**Fonte**: Elaborada pelos autores.

Quadro 29 - Documentação - Administrador Encerrar Lote

|  |  |
| --- | --- |
| Nome do caso de uso | Encerrar Lote |
| Ator principal | Administrador |
| Resumo | Finaliza o ciclo do lote, registrando data de saída e status final. |
| Pré-condições | Administrador autenticado. Lote existente, ativo e selecionado. |
| Pós-condições | Lote marcado como encerrado com data de saída registrada. Sistema apresenta 'Dados atualizados com sucesso' (Msg3). |
| Fluxo Normal | 1. Administrador seleciona um lote ativo. 2. Aciona 'Encerrar Lote'. 3. Informa data de saída e confirma. 4. Sistema valida pendências básicas e salva encerramento. 5. Sistema confirma o encerramento. |
| Fluxo alternativo | 1. Lote já encerrado ou com dados inconsistentes. 2. Sistema informa impedimento e mantém na tela. |

**Fonte**: Elaborada pelos autores.

Quadro 30 - Documentação - Administrador Excluir Lote

|  |  |
| --- | --- |
| Nome do caso de uso | Excluir Lote |
| Ator principal | Administrador |
| Resumo | Remove um lote do sistema (conforme regra de negócio). |
| Pré-condições | Administrador autenticado. Lote existente e selecionado. |
| Pós-condições | Lote removido. Sistema apresenta 'Registro excluído com sucesso' (Msg2). |
| Fluxo Normal | 1. Administrador seleciona o lote. 2. Aciona 'Excluir'. 3. Sistema solicita confirmação. 4. Administrador confirma exclusão. 5. Sistema remove o registro e confirma a operação. |
| Fluxo alternativo | 1. Administrador cancela a confirmação. 2. Sistema não exclui e retorna à tela anterior. |

**Fonte**: Elaborada pelos autores.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 10 – Diagrama de Caso de Uso Individual - Produtor Consumo Ração

**Fonte**: Elaborada pelos autores.

Quadro 31 - Documentação - Produtor Registrar Consumo de Ração

|  |  |
| --- | --- |
| Nome do caso de uso | Registrar Consumo de Ração |
| Ator principal | Produtor |
| Resumo | Registra um novo consumo de ração para um lote. |
| Pré-condições | Produtor autenticado; lote existente e ativo. |
| Pós-condições | Registro de consumo salvo. Sistema apresenta 'Consumo de ração registrado com sucesso' (Msg17). |
| Fluxo Normal | 1. Produtor seleciona um lote. 2. Acessa opção 'Novo Consumo de Ração'. 3. Informa data, quantidade, tipo de ração e aves vivas. 4. Sistema valida dados obrigatórios. 5. Sistema salva registro e confirma operação. |
| Fluxo alternativo | 1. Campos obrigatórios não preenchidos. 2. Sistema exibe erro (Msg6). |

**Fonte**: Elaborada pelos autores.

Quadro 32 - Documentação – Produtor Listar Consumo de Ração

|  |  |
| --- | --- |
| Nome do caso de uso | Listar Consumo de Ração |
| Ator principal | Produtor |
| Resumo | Permite ao produtor visualizar os registros de consumo de ração por lote e período. |
| Pré-condições | Produtor autenticado no sistema; deve existir lote ativo. |
| Pós-condições | Lista de registros de consumo exibida. Sistema apresenta 'Dados exibidos com sucesso' (Msg4). |
| Fluxo Normal | 1. Produtor acessa a tela de Consumo de Ração. 2. Informa filtros (período e lote). 3. Sistema consulta e exibe registros de consumo. 4. Produtor pode selecionar um registro para manutenção. |
| Fluxo alternativo | 1. Se não existirem registros, sistema exibe 'Nenhum consumo encontrado'. |

**Fonte**: Elaborada pelos autores.

Quadro 33 - Documentação - Produtor Alterar Consumo de Ração

|  |  |
| --- | --- |
| Nome do caso de uso | Alterar Consumo de Ração |
| Ator principal | Produtor |
| Resumo | Permite ao produtor modificar um registro de consumo existente. |
| Pré-condições | Produtor autenticado; registro existente selecionado. |
| Pós-condições | Registro atualizado. Sistema apresenta 'Dados atualizados com sucesso' (Msg3). |
| Fluxo Normal | 1. Produtor seleciona registro na lista. 2. Acessa opção 'Alterar Consumo'. 3. Modifica dados necessários. 4. Sistema valida e grava alterações. 5. Sistema confirma atualização. |
| Fluxo alternativo | 1. Registro inexistente ou inválido. 2. Sistema exibe erro e mantém na tela. |

**Fonte**: Elaborada pelos autores.

Quadro 34 - Documentação - Produtor Excluir Consumo de Ração

|  |  |
| --- | --- |
| Nome do caso de uso | Excluir Consumo de Ração |
| Ator principal | Produtor |
| Resumo | Permite excluir um registro de consumo de ração. |
| Pré-condições | Produtor autenticado; registro existente selecionado. |
| Pós-condições | Registro excluído. Sistema apresenta 'Registro excluído com sucesso' (Msg2). |
| Fluxo Normal | 1. Produtor seleciona registro da lista. 2. Acessa opção 'Excluir'. 3. Sistema solicita confirmação. 4. Produtor confirma exclusão. 5. Sistema remove o registro e confirma operação. |
| Fluxo alternativo | 1. Produtor cancela a exclusão. 2. Sistema mantém registro e retorna à tela anterior. |

**Fonte**: Elaborada pelos autores.

Quadro 35 - Documentação – Produtor Aplicar Filtros

|  |  |
| --- | --- |
| Nome do caso de uso | Aplicar Filtros |
| Ator principal | Produtor |
| Resumo | Permite ao produtor aplicar filtros para visualizar registros de consumo de ração de forma segmentada (por período, lote ou tipo de ração). |
| Pré-condições | Produtor autenticado no sistema. Deve existir ao menos um lote cadastrado. |
| Pós-condições | Sistema apresenta apenas os registros que correspondem aos filtros aplicados. |
| Fluxo Normal | 1. Produtor acessa a tela de consumo de ração. 2. Informa parâmetros de filtro (período, lote, tipo de ração). 3. Sistema processa os filtros e exibe apenas os registros correspondentes. |
| Fluxo alternativo | 1. Caso nenhum registro atenda aos filtros, o sistema exibe 'Nenhum registro encontrado'. |

**Fonte**: Elaborada pelos autores.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 11 – Diagrama de Caso de Uso Individual - Produtor Eventos Sanitários

**Fonte:** Elaborado pelos autores

Quadro 36 - Documentação - Produtor Listar Eventos Sanitários

|  |  |
| --- | --- |
| Nome do caso de uso | Listar Eventos Sanitários |
| Ator principal | Produtor |
| Resumo | Permite ao produtor visualizar os eventos sanitários (vacinas, medicamentos e tratamentos) de um lote. |
| Pré-condições | Produtor autenticado no sistema; lote ativo selecionado. |
| Pós-condições | Lista de eventos exibida. Sistema apresenta 'Dados exibidos com sucesso' (Msg4). |
| Fluxo Normal | 1. Produtor acessa a tela de eventos sanitários. 2. Sistema solicita filtro por lote/período. 3. Sistema exibe lista de eventos cadastrados. |
| Fluxo alternativo | 1. Caso não existam registros, o sistema exibe 'Nenhum evento encontrado'. |

**Fonte**: Elaborada pelos autores.

Quadro 37 - Documentação - Produtor Registrar Evento Sanitário

|  |  |
| --- | --- |
| Nome do caso de uso | Registrar Evento Sanitário |
| Ator principal | Produtor |
| Resumo | Registra novo evento sanitário associado a um lote. |
| Pré-condições | Produtor autenticado; lote ativo selecionado. |
| Pós-condições | Evento registrado e disponível para consulta. Sistema apresenta 'Evento sanitário registrado com sucesso' (Msg19). |
| Fluxo Normal | 1. Produtor seleciona o lote. 2. Acessa a opção 'Novo Evento Sanitário'. 3. Informa dados obrigatórios (tipo de evento, produto, dose, via de aplicação, data). 4. Sistema valida dados. 5. Sistema salva e confirma o registro. |
| Fluxo alternativo | 1. Dados incompletos ou inválidos. 2. Sistema exibe erro e mantém na tela. |

**Fonte**: Elaborada pelos autores.

Quadro 38 - Documentação - Produtor Alterar Evento Sanitário

|  |  |
| --- | --- |
| Nome do caso de uso | Alterar Evento Sanitário |
| Ator principal | Produtor |
| Resumo | Permite modificar dados de um evento sanitário já existente. |
| Pré-condições | Produtor autenticado; evento existente selecionado. |
| Pós-condições | Evento atualizado. Sistema apresenta 'Dados atualizados com sucesso' (Msg3). |
| Fluxo Normal | 1. Produtor seleciona o evento na lista. 2. Acessa a opção 'Alterar Evento'. 3. Modifica os campos necessários. 4. Sistema valida e salva alterações. 5. Sistema confirma a atualização. |
| Fluxo alternativo | 1. Evento inexistente ou inválido. 2. Sistema informa erro e retorna à tela anterior. |

**Fonte**: Elaborada pelos autores.

Quadro 39 - Documentação - Produtor Excluir Evento Sanitário

|  |  |
| --- | --- |
| Nome do caso de uso | Excluir Evento Sanitário |
| Ator principal | Produtor |
| Resumo | Permite remover um evento sanitário já registrado. |
| Pré-condições | Produtor autenticado; evento existente selecionado. |
| Pós-condições | Evento removido. Sistema apresenta 'Registro excluído com sucesso' (Msg2). |
| Fluxo Normal | 1. Produtor seleciona o evento da lista. 2. Acessa opção 'Excluir'. 3. Sistema solicita confirmação. 4. Produtor confirma exclusão. 5. Sistema remove registro e confirma operação. |
| Fluxo alternativo | 1. Produtor cancela a exclusão. 2. Sistema mantém o evento e retorna à tela anterior. |

**Fonte**: Elaborada pelos autores.

Quadro 40 - Documentação - Produtor Aplicar Filtros (Eventos Sanitários)

|  |  |
| --- | --- |
| Nome do caso de uso | Aplicar Filtros (Eventos Sanitários) |
| Ator principal | Produtor |
| Resumo | Permite aplicar filtros para refinar a listagem de eventos sanitários por período, tipo de evento ou lote. |
| Pré-condições | Produtor autenticado no sistema; lote ativo selecionado. |
| Pós-condições | Sistema apresenta apenas registros que atendem aos filtros. |
| Fluxo Normal | 1. Produtor acessa a tela de eventos sanitários. 2. Informa filtros (período, lote, tipo de evento). 3. Sistema processa filtros e exibe resultados. |
| Fluxo alternativo | 1. Caso não existam registros correspondentes, o sistema exibe 'Nenhum evento encontrado'. |

**Fonte**: Elaborada pelos autores.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 12 - Diagrama de Caso de Uso Individual - Financeiro Transações Financeiras

**Fonte**: Elaborada pelos autores.

Quadro 41 - Documentação - Financeiro Listar Transações Financeiras

|  |  |
| --- | --- |
| Nome do caso de uso | Listar Transações Financeiras |
| Ator principal | Financeiro |
| Resumo | Permite ao financeiro visualizar todas as transações financeiras de entradas e saídas da granja, com possibilidade de aplicar filtros. |
| Pré-condições | Financeiro autenticado no sistema. Devem existir registros financeiros cadastrados. |
| Pós-condições | Lista de transações financeiras exibida. Sistema apresenta 'Dados exibidos com sucesso' (Msg4). |
| Fluxo Normal | 1. Financeiro acessa a tela de transações financeiras. 2. Sistema solicita filtros (período, tipo, lote). 3. Sistema exibe lista de transações encontradas. |
| Fluxo alternativo | 1. Caso não existam registros, o sistema exibe mensagem 'Nenhuma transação encontrada'. |

**Fonte**: Elaborada pelos autores.

Quadro 42 - Documentação - Financeiro Aplicar Filtros

|  |  |
| --- | --- |
| Nome do caso de uso | Aplicar Filtros (Transações Financeiras) |
| Ator principal | Financeiro |
| Resumo | Permite aplicar filtros sobre as transações financeiras para refinar resultados (por período, tipo de transação ou lote). |
| Pré-condições | Financeiro autenticado; transações cadastradas. |
| Pós-condições | Sistema exibe apenas registros correspondentes aos filtros aplicados. |
| Fluxo Normal | 1. Financeiro acessa a tela de transações financeiras. 2. Informa filtros desejados. 3. Sistema processa filtros e exibe resultados. |
| Fluxo alternativo | 1. Caso não haja registros correspondentes, sistema apresenta mensagem 'Nenhuma transação encontrada'. |

**Fonte**: Elaborada pelos autores.

Quadro 43 - Documentação - Financeiro Cadastrar Entrada/Saída

|  |  |
| --- | --- |
| Nome do caso de uso | Cadastrar Entrada/Saída |
| Ator principal | Financeiro |
| Resumo | Permite registrar uma nova transação financeira, seja entrada (receita) ou saída (despesa). |
| Pré-condições | Financeiro autenticado; lote existente (quando aplicável). |
| Pós-condições | Registro salvo. Sistema apresenta 'Transação financeira registrada com sucesso' (Msg24). |
| Fluxo Normal | 1. Financeiro acessa a opção 'Nova Transação'. 2. Informa dados obrigatórios (descrição, valor, tipo, data, lote). 3. Sistema valida campos obrigatórios. 4. Sistema salva registro. 5. Sistema confirma cadastro. |
| Fluxo alternativo | 1. Campos obrigatórios não preenchidos ou valor inválido. 2. Sistema exibe erro (Msg6) e mantém usuário na tela. |

**Fonte**: Elaborada pelos autores.

Quadro 44 - Documentação - Financeiro Alterar Entrada/Saída

|  |  |
| --- | --- |
| Nome do caso de uso | Alterar Entrada/Saída |
| Ator principal | Financeiro |
| Resumo | Permite modificar dados de uma transação financeira existente. |
| Pré-condições | Financeiro autenticado; transação selecionada existente. |
| Pós-condições | Transação atualizada. Sistema apresenta 'Dados atualizados com sucesso' (Msg3). |
| Fluxo Normal | 1. Financeiro seleciona uma transação. 2. Acessa a opção 'Alterar'. 3. Modifica os campos desejados. 4. Sistema valida alterações. 5. Sistema confirma atualização. |
| Fluxo alternativo | 1. Transação inexistente ou inválida. 2. Sistema exibe erro e retorna à tela anterior. |

**Fonte**: Elaborada pelos autores.

Quadro 45 - Documentação - Financeiro Excluir Entrada/Saída

|  |  |
| --- | --- |
| Nome do caso de uso | Excluir Entrada/Saída |
| Ator principal | Financeiro |
| Resumo | Permite remover uma transação financeira registrada. |
| Pré-condições | Financeiro autenticado; transação existente selecionada. |
| Pós-condições | Registro excluído. Sistema apresenta 'Registro excluído com sucesso' (Msg2). |
| Fluxo Normal | 1. Financeiro seleciona uma transação da lista. 2. Acessa a opção 'Excluir'. 3. Sistema solicita confirmação. 4. Financeiro confirma. 5. Sistema exclui o registro e confirma a operação. |
| Fluxo alternativo | 1. Financeiro cancela exclusão. 2. Sistema não exclui e retorna à tela anterior. |

**Fonte**: Elaborada pelos autores.

Quadro 46 - Documentação - Financeiro Calcular Lucro

|  |  |
| --- | --- |
| Nome do caso de uso | Calcular Lucro |
| Ator principal | Financeiro |
| Resumo | Permite calcular o lucro de um lote ou período com base nas receitas e despesas cadastradas. |
| Pré-condições | Financeiro autenticado; registros de receitas e despesas existentes. |
| Pós-condições | Sistema apresenta cálculo do lucro. Sistema exibe 'Estimativa realizada com sucesso' (Msg15). |
| Fluxo Normal | 1. Financeiro acessa a opção 'Calcular Lucro'. 2. Informa parâmetros (lote/período). 3. Sistema realiza cálculo considerando receitas – despesas. 4. Sistema apresenta resultado. |
| Fluxo alternativo | 1. Falta de registros financeiros. 2. Sistema informa impossibilidade de cálculo. |

**Fonte**: Elaborada pelos autores.

Quadro 47 - Documentação -Financeiro Gerar Relatório Financeiro

|  |  |
| --- | --- |
| Nome do caso de uso | Gerar Relatório Financeiro |
| Ator principal | Financeiro |
| Resumo | Gera relatório detalhado das transações financeiras, incluindo entradas, saídas, saldo e lucro. |
| Pré-condições | Financeiro autenticado; registros financeiros disponíveis. |
| Pós-condições | Relatório gerado e exibido em tela. Sistema apresenta 'Relatório gerado com sucesso' (Msg10). |
| Fluxo Normal | 1. Financeiro acessa a opção 'Gerar Relatório'. 2. Informa parâmetros (período, tipo de transação, lote). 3. Sistema compila dados. 4. Sistema apresenta relatório. |
| Fluxo alternativo | 1. Ausência de registros no período informado. 2. Sistema apresenta relatório vazio com aviso. |

**Fonte**: Elaborada pelos autores.

## 3.6. Diagrama de Sequência

*(2 principais: cadastro, consulta, relatório, agendamento, controle, etc..)*

*Inserir o diagrama e abaixo colocar a documentação do mesmo.*

## 3.7. Diagrama de Comunicação

*(1 diagrama relevante do projeto)*

## 3.8. Diagrama de Atividade

*(1 diagrama de uma ação específica do sistema)*

## 3.9 Diagrama de Máquina de Estados

# 4 DEFINIÇÃO DA INTERFACE COM O USUÁRIO (UX)

Nesta seção, apresenta-se o processo de pesquisa e desenvolvimento aplicado ao planejamento da interface do sistema Granja Tech, com foco na Experiência do Usuário (UX). O conceito de UX abrange as percepções, emoções e respostas de um usuário ao interagir com um sistema, seja durante a utilização, visualização ou execução de qualquer ação dentro da plataforma. De acordo com Rogers, Sharp e Preece (2013), a experiência do usuário refere-se diretamente à qualidade da interação, tendo como pilares a satisfação, o conforto, a eficiência e a facilidade durante o uso.

Este conceito não se limita apenas a produtos físicos, aplicando-se também a ambientes digitais, como softwares, aplicativos e sistemas web. A definição de UX considera, portanto, não apenas a estética visual da interface, mas também sua usabilidade, acessibilidade, navegabilidade e a clareza na apresentação das informações (Unger; Chandler, 2009).

Com base nesses princípios, o desenvolvimento da interface do Granja Tech buscou compreender profundamente as necessidades, expectativas e limitações dos usuários, que são, em sua maioria, produtores rurais, colaboradores das granjas e responsáveis pela gestão financeira, produtiva e sanitária. A adoção de boas práticas de UX visa garantir que o sistema proporcione uma navegação intuitiva, com informações organizadas de forma clara e acessível, mesmo para usuários com pouca familiaridade com tecnologia.

## Descrição de cenário

No desenvolvimento de sistemas orientados pela experiência do usuário, os cenários desempenham um papel fundamental na modelagem das interações e na construção das funcionalidades. Segundo Rogers, Sharp e Preece (2013), cenários consistem em representações narrativas que descrevem atividades e tarefas dos usuários em determinado contexto, permitindo visualizar como eles interagem com a plataforma e quais são suas necessidades durante esse processo.

Para o desenvolvimento do Granja Tech, foram elaborados dois cenários que representam situações comuns no cotidiano de uma granja, abordando, respectivamente, o processo de registro de mortalidade de aves e a análise dos dashboards de desempenho zootécnico e financeiro.

O Quadro 48 descreve o cenário em que um produtor, devidamente autenticado no sistema, realiza o registro da mortalidade de um lote específico. Este processo é essencial para manter o controle sanitário e acompanhar os indicadores de desempenho da produção. O cenário ilustra a experiência do usuário, desde o acesso à interface, preenchimento dos dados, até a validação e confirmação do registro, demonstrando a eficiência e praticidade proporcionadas pelo sistema.

Quadro 48 – Cenário – Registro de Mortalidade de Aves pelo Produtor

Em uma manhã de trabalho na granja, o produtor acessa o sistema Granja Tech por meio de seu notebook, devidamente autenticado. Durante sua rotina de vistoria, ele observa que algumas aves do lote atual vieram a óbito. Com o intuito de manter o controle sanitário atualizado, o produtor abre o módulo de gestão de lotes e seleciona o lote correspondente. Na tela, ele localiza a opção "Registrar Mortalidade" e preenche os campos solicitados, inserindo informações como a quantidade de aves, a data e, se necessário, uma breve descrição da causa aparente. Após revisar os dados, clica em “Salvar”. O sistema valida as informações, atualiza automaticamente os indicadores do lote e gera uma mensagem de confirmação, permitindo que o produtor retome sua rotina com a segurança de que aquele evento foi devidamente registrado no histórico da granja.

**Fonte**: Elaborada pelos autores.

O Quadro 49 apresenta o cenário no qual um colaborador da área administrativa da granja acessa o dashboard do sistema para realizar uma análise detalhada dos dados produtivos e financeiros. Este cenário demonstra como a interface do Granja Tech fornece informações visuais e analíticas essenciais para a tomada de decisões, contribuindo diretamente para a gestão eficiente do negócio.

Quadro 49 – Cenário – Análise de Dashboards pelo Responsável Financeiro da Granja

No ambiente administrativo da granja, o responsável financeiro, devidamente autenticado no sistema, acessa o módulo de dashboards do Granja Tech. Na tela inicial, são exibidos gráficos com a visão geral da produção, destacando informações como taxa de mortalidade, consumo diário de ração, peso médio das aves e alertas sobre variações ambientais. O dashboard também apresenta indicadores financeiros, como custos fixos, variáveis, receitas e lucro operacional de cada lote ativo. O colaborador utiliza os filtros disponíveis para visualizar dados específicos, como desempenho por período ou por unidade de produção, e analisa se os custos estão dentro dos parâmetros esperados. Com base nessas informações, gera um relatório consolidado que servirá de base para o planejamento das próximas ações na granja, como ajustes na alimentação, intervenções sanitárias ou revisão dos custos operacionais.

**Fonte**: Elaborada pelos autores.

## Descrição de personas

No processo de desenvolvimento de sistemas, o uso de personas é uma prática importante para assegurar que as necessidades e características do público-alvo sejam contempladas. Segundo Unger e Chandler (2009), personas são representações detalhadas de usuários-alvo que ajudam a compreender de maneira prática quem utiliza o sistema e como ele é utilizado. Elas são criadas a partir de dados reais sobre os usuários e são úteis para orientar decisões de design e funcionalidades do sistema. Para a criação de personas efetivas, é necessário identificar o público e reunir informações suficientes que revelem o comportamento, as motivações e as expectativas do usuário.

As personas apresentadas para o projeto Granja Tech foram criadas com base em informações detalhadas sobre os usuários potenciais. Essas personas incluem dados como foto, nome, idade, localização e ocupação, além de objetivos específicos de uso do sistema (Unger; Chandler, 2009). Tais dados foram utilizados para representar com precisão o perfil dos usuários e para guiar o desenvolvimento do sistema de acordo com as necessidades específicas de cada persona.

A Figura 13 apresenta a persona “Carlos Eduardo da Silva”, um produtor avícola experiente que gerencia uma granja de pequeno porte. Com mais de 25 anos de atuação, Carlos busca uma ferramenta tecnológica que permita um controle mais eficiente da produção, minimizando desperdícios e otimizando o manejo dos lotes de aves.

Figura 13 – Persona 1

**Interface gráfica do usuário, Texto, Aplicativo
Fonte:** Elaborada pelos autores.

A Figura 14 apresenta a persona “Patrícia Mendes Rocha”, uma técnica agropecuária que presta consultoria a produtores rurais, oferecendo suporte técnico relacionado à gestão e produtividade das granjas. Com formação em Zootecnia, Patrícia visa utilizar o sistema Granja Tech para gerar relatórios detalhados e apoiar seus clientes na tomada de decisões estratégicas.

Figura 14 – Persona 2

**Interface gráfica do usuário, Texto, Aplicativo
Fonte:** Elaborada pelos autores.

## Esboços de tela (wireframes)

Wireframes são representações visuais simplificadas, que permitem estruturar a disposição dos elementos de uma interface, sem a necessidade de detalhamento visual ou gráfico. Trata-se de um protótipo de baixa fidelidade, que foca na organização e hierarquia de informações, apresentando aspectos como a navegação entre páginas, os campos de formulários, botões de ação e seções de conteúdo (Unger; Chandler, 2009).

Normalmente, os wireframes utilizam tons de preto, branco e cinza, dispensando elementos gráficos elaborados ou cores. Essa abordagem minimalista facilita o entendimento da funcionalidade e do fluxo das telas, antes que recursos visuais mais complexos sejam incorporados. Além disso, permite um alinhamento mais ágil entre os desenvolvedores, designers e stakeholders, evitando retrabalhos futuros (Unger; Chandler, 2009).

Para o desenvolvimento do sistema Granja Tech, foram elaborados os wireframes das principais telas do sistema: tela de login, tela principal (dashboard), tela de lotes, tela de granjas e a tela de cadastro de usuário. Essas telas foram priorizadas por serem essenciais para o fluxo inicial da plataforma, garantindo que o usuário consiga acessar o sistema, visualizar informações relevantes da granja e gerenciar os dados fundamentais de sua produção.

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.Figura 15 – Wireframe - Tela Login

**Fonte:** Elaborada pelos autores.

Na Figura 15, apresenta-se o wireframe da tela de login do sistema GranjaTech. A interface adota uma estrutura simples e objetiva, centralizando os elementos essenciais para o primeiro contato do usuário com a aplicação.

No centro da tela, encontram-se dois campos de entrada, destinados à inserção de e-mail e senha, necessários para a autenticação. Logo abaixo, está localizado o botão Entrar, que permite o acesso imediato ao sistema.

Na parte superior, o nome GranjaTech é exibido em destaque, reforçando a identidade visual da aplicação. A disposição dos elementos busca assegurar equilíbrio, clareza e funcionalidade, proporcionando uma navegação intuitiva e eficiente já na etapa inicial de uso.

Figura 16 – Wireframe – Tela Dashboard

**Fonte:** Elaborada pelos autores.

Na Figura 16, apresenta-se o wireframe da tela Dashboard do sistema GranjaTech. A interface adota layout simples e organizado, reunindo em um só ambiente os principais acessos e informações iniciais. À esquerda, encontra-se o menu lateral, com atalhos para Dashboard, Granjas, Lotes, Estoque, Avicultura Pro, Consumo, Pesagens, Sanitário e Sensores, garantindo acesso rápido às funcionalidades.

Na parte superior, a barra de navegação exibe o nome do sistema, o usuário logado e recursos de acessibilidade, como modo claro/escuro e ajuste de fonte. Ao centro, destaca-se o Resumo Mensal, em formato gráfico, apresentando entradas e saídas de forma visual, permitindo acompanhar rapidamente dados essenciais da granja.

A composição prioriza clareza, objetividade e navegação intuitiva, mantendo a identidade visual do sistema e oferecendo eficiência já no painel inicial.

Interface gráfica do usuário, Site

O conteúdo gerado por IA pode estar incorreto.Figura 17 – Wireframe – Tela Cadastro de Usuário

**Fonte:** Elaborada pelos autores.

Na Figura 17, apresenta-se o wireframe da tela de Usuários do sistema GranjaTech. Diferente do acesso inicial, o cadastro de novos usuários é realizado exclusivamente pelo administrador, garantindo maior controle e segurança sobre os perfis que terão acesso à plataforma.

À esquerda, mantém-se o menu lateral de navegação, com atalhos para os principais módulos do sistema. Na parte superior, a barra de navegação exibe o nome do sistema, o usuário logado e recursos de acessibilidade, como modo claro/escuro e ajuste de fonte.

No centro da interface, há uma tabela destinada à listagem de usuários cadastrados, organizada por Nome, Email e Permissão. Acima dela, encontra-se o botão Criar novo usuário, que abre o formulário de cadastro. Esse formulário apresenta campos obrigatórios para Nome, Email, Senha e Perfil, seguidos dos botões Cadastrar e Cancelar, que permitem confirmar ou desistir da operação.

A tela foi projetada para facilitar a gestão de usuários pelo administrador, mantendo clareza na organização das informações e garantindo uma navegação simples e eficiente.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.Figura 18 – Wireframe - Tela Granjas

**Fonte:** Elaborada pelos autores.

Na Figura 18, apresenta-se o wireframe da tela de Granjas do sistema GranjaTech, destinada ao gerenciamento das propriedades cadastradas. A interface mantém a estrutura de navegação das demais telas, com menu lateral à esquerda e barra superior exibindo o nome do sistema, o usuário logado e os recursos de acessibilidade (modo claro/escuro e ajuste de fonte).

No centro, encontra-se uma tabela organizada com colunas para Código, Nome, Localização, Dono e Ações, permitindo ao administrador visualizar de forma clara e estruturada todas as granjas cadastradas. No canto superior direito, está disponível o botão Nova granja, que abre o formulário de cadastro para inclusão de uma nova unidade produtiva e somente um usuário Produtor ou Administrador podem registrar granjas.

Essa tela foi projetada para facilitar a gestão das granjas pelo administrador, reunindo em um único ambiente funções de consulta, cadastro e manutenção, assegurando simplicidade e eficiência no controle das propriedades.

Interface gráfica do usuário, Texto, Aplicativo, Site

O conteúdo gerado por IA pode estar incorreto.Figura 19 - Wireframe - Tela Lotes

**Fonte:** Elaborada pelos autores.

Na Figura 19, apresenta-se o wireframe da tela de Lotes do sistema GranjaTech, destinada ao gerenciamento dos lotes de produção. A interface segue o padrão das demais telas, com menu lateral à esquerda contendo os módulos principais e barra superior exibindo recursos de acessibilidade (modo claro/escuro e ajuste de fonte), além da identificação do usuário logado, onde, só usuários Produtor e Administrador podem registrar novos lotes.

No centro, há uma tabela organizada em colunas de Código, Identificador, Quantidade de Aves, Data de Entrada, Data de Saída, Status e Granja, possibilitando ao administrador visualizar e controlar todos os lotes cadastrados. No canto superior direito, encontra-se o botão Novo lote, utilizado para registrar um novo grupo de aves na granja.

A tela foi estruturada para oferecer praticidade e clareza no acompanhamento da produção, permitindo o controle completo dos lotes de forma simples e eficiente.

## Protótipos de tela

Segundo Rosa (2024), "a finalidade principal de um protótipo é validar hipóteses com uma versão mais simples". Em outras palavras, a criação de protótipos permite testar ideias e verificar, na prática, o que funciona e o que pode ser aprimorado. O wireframe busca estruturar o layout e a navegação, enquanto o protótipo tem como objetivo simular de forma mais realista a experiência do usuário, possibilitando uma compreensão mais profunda de suas interações com o sistema (Rosa, 2024).

No projeto do sistema GranjaTech, os protótipos foram desenvolvidos com o auxílio da ferramenta Figma, permitindo a criação de interfaces interativas com alta fidelidade. Esses protótipos incorporam elementos visuais completos, como cores, ícones e estruturas de navegação, simulando a aparência e o funcionamento do sistema final.

A seguir, são apresentados os protótipos das principais telas do sistema, anteriormente demonstradas nos wireframes, acompanhados de uma breve descrição de suas funcionalidades e dos conceitos utilizados em seu desenvolvimento.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.Figura 20 – Protótipo Tela – Cadastrar Usuário

**Fonte:** Elaborada pelos autores.

Na Figura 20, apresenta-se o protótipo da tela de Cadastro de Usuários do sistema GranjaTech, destinada à administração de contas e permissões. A interface mantém o padrão visual do sistema, com menu lateral à esquerda reunindo os principais módulos e barra superior contendo as informações do usuário logado.

No centro da tela, destaca-se o formulário de novo usuário, exibido em um card branco com campos para Nome, Email, Senha e Perfil, organizados de forma simples e clara. Na parte inferior, dois botões de ação permitem Cancelar ou Criar o registro, assegurando flexibilidade ao administrador.

Essa tela foi projetada para oferecer ao administrador um fluxo prático e objetivo de gerenciamento de usuários, garantindo consistência, segurança e usabilidade no controle de acessos ao sistema.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.Figura 21 – Protótipo Tela – Login do usuário

**Fonte:** Elaborada pelos autores.

Na Figura 21, apresenta-se o protótipo da tela de Login do sistema GranjaTech, desenvolvida para garantir um acesso simples e seguro. A interface possui fundo em tom verde, reforçando a identidade visual da aplicação e transmitindo relação com o setor agropecuário.

No centro, um card branco destaca os elementos principais: o logotipo do sistema, seguido do nome GranjaTech e da descrição *“*Sistema de Gestão Agropecuária*”*. Logo abaixo, são exibidos os campos obrigatórios para Email e Senha, organizados de forma clara e objetiva.

Na parte inferior, encontra-se o botão Entrar, que permite ao usuário acessar o sistema após a autenticação. O design prioriza minimalismo, contraste adequado e usabilidade, assegurando uma navegação fluida e intuitiva já no primeiro contato do usuário com a plataforma.

Interface gráfica do usuário, Texto, Aplicativo

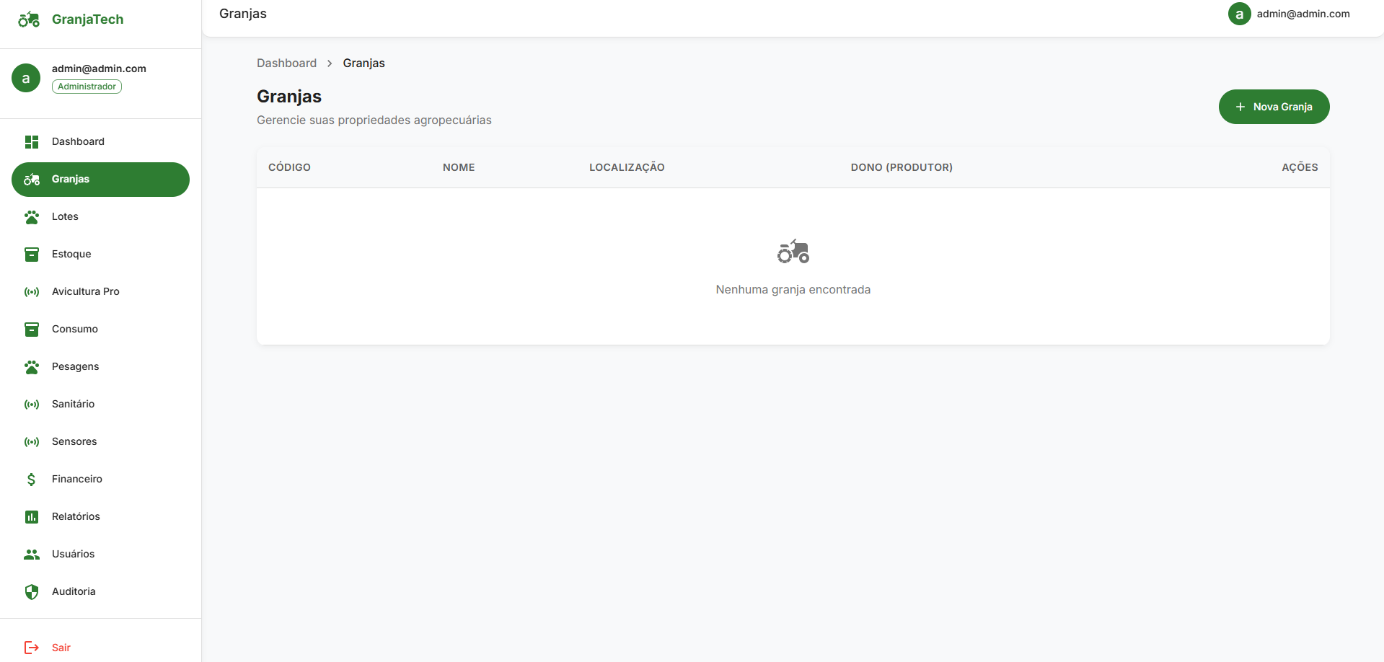
O conteúdo gerado por IA pode estar incorreto.Figura 22 – Protótipo Tela – Dashboard

**Fonte:** Elaborada pelos autores.

Na Figura 22, apresenta-se o protótipo da tela Dashboard do sistema GranjaTech, desenvolvida para fornecer uma visão geral das operações. A interface mantém o menu lateral à esquerda, reunindo os módulos principais, como Granjas, Lotes, Estoque, Consumo, Financeiro e Auditoria, além de exibir o perfil do usuário logado.

No centro, o destaque é o Resumo Mensal, representado em gráfico, que organiza as informações de Entradas e Saídas de forma visual e simplificada. Esse recurso auxilia o administrador na análise rápida do desempenho da granja, permitindo acompanhamento de resultados sem necessidade de relatórios extensos.

A composição prioriza clareza e usabilidade, garantindo ao usuário uma experiência intuitiva, com dados acessíveis logo na tela inicial do sistema.

Figura 23 – Protótipo Tela – Granja

**Fonte:** Elaborada pelos autores.

Na Figura 23, apresenta-se o protótipo da tela de Granjas do sistema GranjaTech, utilizada para o gerenciamento das propriedades agropecuárias. A interface mantém a padronização visual das demais telas, com menu lateral à esquerda, contendo os módulos principais, e a barra superior exibindo informações do usuário logado.

No centro, encontra-se uma tabela estruturada com colunas de Código, Nome, Localização, Dono (Produtor) e Ações, destinada a listar as granjas cadastradas. No canto superior direito, está disponível o botão Nova Granja, que permite ao administrador incluir uma nova propriedade no sistema.

A tela foi projetada para oferecer praticidade na consulta e no registro de granjas, assegurando clareza na organização dos dados e consistência visual em relação ao restante da aplicação.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.Figura 24 – Protótipo Tela – Lotes

**Fonte:** Elaborada pelos autores.

Na Figura 24, apresenta-se o protótipo da tela de Lotes do sistema GranjaTech, destinada ao gerenciamento dos lotes de produção. A interface segue o mesmo padrão visual das demais telas, com menu lateral à esquerda reunindo os módulos principais e barra superior exibindo informações do usuário logado.

No centro, está disposta uma tabela organizada em colunas de Código, Identificador, Quantidade de Aves, Data de Entrada, Data de Saída, Status, Granja e Ações, que concentra as informações essenciais para acompanhamento dos lotes. No canto superior direito, o botão Novo Lote possibilita o cadastro de novos registros.

A tela foi projetada para oferecer clareza e praticidade no controle dos lotes, permitindo ao administrador gerenciar a produção de forma eficiente e padronizada.



## Acessibilidade

A acessibilidade web refere-se ao desenvolvimento de aplicações digitais que possam ser utilizadas por todas as pessoas, incluindo aquelas com deficiências visuais, motoras, auditivas ou cognitivas. No contexto do sistema GranjaTech, a acessibilidade foi considerada um requisito fundamental desde as fases iniciais de projeto, visando garantir que a plataforma seja inclusiva e atenda aos padrões internacionais e nacionais de acessibilidade.

As diretrizes internacionais de acessibilidade, conhecidas como WCAG (Web Content Accessibility Guidelines), estabelecidas pelo W3C, foram utilizadas como referência principal no desenvolvimento do sistema. Além disso, a Lei Brasileira de Inclusão da Pessoa com Deficiência (Lei nº 13.146/2015), que determina a obrigatoriedade da acessibilidade em serviços públicos e privados, também foi considerada para assegurar a conformidade legal da aplicação.

No que diz respeito à implementação prática, diversos recursos de acessibilidade foram incorporados ao sistema GranjaTech. Um dos aspectos mais importantes foi a garantia de contraste adequado entre cores de fundo e texto, especialmente nas telas de cadastro e login. Nessas interfaces, foi utilizado um fundo escuro com elementos textuais claros e botões em cores vivas, como verde e laranja, proporcionando melhor legibilidade para usuários com baixa visão.

Outro recurso significativo foi a utilização de ícones representativos nas interfaces, particularmente na tela de estoque. Esses ícones, posicionados estrategicamente no menu lateral e nas ações principais do sistema, facilitam a navegação para usuários com dificuldades cognitivas ou menor familiaridade com ambientes digitais. A padronização visual desses elementos contribui para uma experiência mais intuitiva e acessível.

Os botões de ação foram projetados com tamanhos amplos e cores bem definidas, seguindo um padrão consistente em todo o sistema. Por exemplo, na tela de controle de estoque, foram utilizadas as cores vermelha para ações de exclusão, azul para edição e verde para confirmação. Essa padronização cromática auxilia usuários com limitações motoras ou visuais na identificação rápida das funções disponíveis.

O sistema também incorporou mecanismos de ajuste visual, disponíveis na barra superior das telas principais. Esses controles permitem ao usuário aumentar ou diminuir o tamanho da fonte, além de alternar entre os modos claro e escuro de exibição. Tais funcionalidades são particularmente úteis para pessoas com sensibilidade visual ou necessidades específicas de leitura.

Na tela inicial do sistema, os dados são apresentados tanto em formato tabular quanto por meio de representações gráficas, como gráficos de linha e de pizza. Essa abordagem oferece alternativas visuais para a interpretação das informações, beneficiando usuários que apresentam dificuldades na compreensão de tabelas ou preferem análises mais visuais.

Em síntese, a implementação desses recursos no sistema GranjaTech teve como objetivo principal garantir que a plataforma seja acessível ao maior número possível de usuários, independentemente de suas capacidades físicas ou cognitivas. A adoção dessas práticas não apenas atende às exigências legais e normativas, mas também reflete o compromisso do projeto com a inclusão digital e a igualdade de acesso à tecnologia.

# BANCO DE DADOS

O banco de dados constitui um dos elementos centrais do sistema, sendo responsável pelo armazenamento, organização e manutenção da integridade das informações. No âmbito do projeto Granja Tech, a escolha pelo PostgreSQL foi fundamentada na necessidade de garantir confiabilidade, escalabilidade e suporte a operações complexas, características indispensáveis para a gestão e o monitoramento das granjas (DATE, 2019).

O sistema desenvolvido utiliza o PostgreSQL como Sistema Gerenciador de Banco de Dados (SGBD). Trata-se de uma solução relacional de código aberto amplamente reconhecida no meio acadêmico e corporativo por sua robustez, flexibilidade e conformidade com os padrões SQL (SILBERSCHATZ; KORTH; SUDARSHAN, 2020). Para a administração do banco foi adotada a ferramenta pgAdmin, que disponibiliza uma interface gráfica intuitiva para criação de tabelas, execução de consultas e monitoramento das estruturas e do desempenho do banco (DE OLIVEIRA; LIMA, 2021).

A adoção do PostgreSQL possibilita diversas vantagens para o projeto. Em primeiro lugar, sua confiabilidade e robustez asseguram consistência nas operações realizadas (HEUSER, 2009). Além disso, por tratar-se de um SGBD escalável, o PostgreSQL permite manipular grandes volumes de dados sem comprometer a eficiência do sistema (SANTOS; PEREIRA, 2022). Outro aspecto relevante refere-se ao suporte às propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), fundamentais para a integridade das transações (SILBERSCHATZ; KORTH; SUDARSHAN, 2020).

Ressalta-se ainda sua capacidade de extensibilidade, que possibilita a criação de funções e tipos de dados personalizados, além de recursos avançados de segurança, responsáveis por assegurar o controle de acesso, permissões diferenciadas e criptografia de dados (MACHADO; ABREU, 2018).

Dessa forma, a utilização do PostgreSQL mostrou-se adequada às demandas da Granja Tech, viabilizando o armazenamento e a gestão de informações relacionadas à produção, ao controle financeiro e aos dados coletados pelos sensores, de maneira segura, eficiente e padronizada (PRESSMAN, 2010; SOMMERVILLE, 2011).

## Modelo Entidade Relacionamento

O Modelo Entidade-Relacionamento (MER), segundo Silberschatz, Korth e Sudarshan (2020), é uma técnica que representa graficamente as entidades, seus atributos e os relacionamentos que estruturam logicamente um banco de dados. A Figura 23 apresenta o mapeamento do objeto relacional do sistema, evidenciando como as tabelas se conectam para preservar consistência e integridade dos dados, além de servir de base para a implementação do modelo físico (SILBERSCHATZ; KORTH; SUDARSHAN, 2020).

Linha do tempo

O conteúdo gerado por IA pode estar incorreto.**Figura 25 -** Mapeamento do Objeto Relacional

**Fonte:** Elaborado pelos autores

## Script das tabelas

De acordo com Date (2019), os scripts de banco de dados são instruções escritas em SQL (Structured Query Language), cuja função principal é definir a estrutura das tabelas, os relacionamentos e as restrições de integridade de um banco relacional. Esses scripts possibilitam a criação e manutenção organizada dos dados, assegurando consistência e padronização no desenvolvimento de sistemas.

No contexto do projeto, os scripts foram gerados automaticamente pelo **Entity Framework Core**, ferramenta de mapeamento objeto-relacional utilizada em conjunto com o PostgreSQL. Segundo a Microsoft (2024), o Entity Framework é um recurso que permite aos desenvolvedores do ambiente **.NET** manipular dados relacionais por meio de objetos do domínio, eliminando a necessidade de escrever grande parte do código SQL manualmente.

A seguir, serão apresentados os scripts responsáveis pela definição das tabelas, relacionamentos e demais estruturas do banco de dados do sistema **Granja Tech**, que garantem a integridade e o correto funcionamento da aplicação.

Quadro 50 - Tabela de Logs de Auditoria

|  |
| --- |
| CREATE TABLE "LogsAuditoria" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Timestamp" timestamp with time zone NOT NULL,  "UsuarioId" integer NOT NULL,  "UsuarioEmail" text NOT NULL,  "Acao" text NOT NULL,  "Detalhes" text NOT NULL,  CONSTRAINT "PK\_LogsAuditoria" PRIMARY KEY ("Id")  ); |

**Fonte**: Elaborado pelos autores

Quadro 51 - Tabela de Perfis

|  |
| --- |
| CREATE TABLE "Perfis" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Nome" text NOT NULL,  CONSTRAINT "PK\_Perfis" PRIMARY KEY ("Id")  ); |

**Fonte:** Elaborado pelos autores

Quadro 52 - Tabela de Usuários

|  |
| --- |
| CREATE TABLE "Usuarios" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Codigo" text NOT NULL,  "Nome" text NOT NULL,  "Email" text NOT NULL,  "SenhaHash" text NOT NULL,  "PerfilId" integer NOT NULL,  CONSTRAINT "PK\_Usuarios" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_Usuarios\_Perfis\_PerfilId" FOREIGN KEY ("PerfilId") REFERENCES "Perfis" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

**Quadro 53 -** Relacionamento Financeiro/Produtor

|  |
| --- |
| CREATE TABLE "FinanceiroProdutor" (  "FinanceiroId" integer NOT NULL,  "ProdutorId" integer NOT NULL,  CONSTRAINT "PK\_FinanceiroProdutor" PRIMARY KEY ("FinanceiroId", "ProdutorId"),  CONSTRAINT "FK\_FinanceiroProdutor\_Usuarios\_FinanceiroId" FOREIGN KEY ("FinanceiroId") REFERENCES "Usuarios" ("Id") ON DELETE CASCADE,  CONSTRAINT "FK\_FinanceiroProdutor\_Usuarios\_ProdutorId" FOREIGN KEY ("ProdutorId") REFERENCES "Usuarios" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

**Quadro 54 -** Tabela de Granjas

|  |
| --- |
| CREATE TABLE "Granjas" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Codigo" text NOT NULL,  "Nome" text NOT NULL,  "Localizacao" text,  "UsuarioId" integer NOT NULL,  CONSTRAINT "PK\_Granjas" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_Granjas\_Usuarios\_UsuarioId" FOREIGN KEY ("UsuarioId") REFERENCES "Usuarios" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

**Quadro 55 -** Tabela de Lotes

|  |
| --- |
| CREATE TABLE "Lotes" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Codigo" text NOT NULL,  "Identificador" text NOT NULL,  "DataEntrada" timestamp with time zone NOT NULL,  "DataSaida" timestamp with time zone,  "QuantidadeAvesInicial" integer NOT NULL,  "GranjaId" integer NOT NULL,  CONSTRAINT "PK\_Lotes" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_Lotes\_Granjas\_GranjaId" FOREIGN KEY ("GranjaId") REFERENCES "Granjas" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

**Quadro 56** - Tabela de Produtos

|  |
| --- |
| CREATE TABLE "Produtos" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Nome" text NOT NULL,  "Tipo" text NOT NULL,  "Quantidade" numeric NOT NULL,  "UnidadeDeMedida" text NOT NULL,  "GranjaId" integer NOT NULL,  CONSTRAINT "PK\_Produtos" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_Produtos\_Granjas\_GranjaId" FOREIGN KEY ("GranjaId") REFERENCES "Granjas" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

**Quadro 57 -** Tabela de Transações Financeiras

|  |
| --- |
| CREATE TABLE "TransacoesFinanceiras" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Descricao" text NOT NULL,  "Valor" numeric NOT NULL,  "Tipo" text NOT NULL,  "Data" timestamp with time zone NOT NULL,  "TimestampCriacao" timestamp with time zone NOT NULL,  "UsuarioId" integer NOT NULL,  "LoteId" integer,  CONSTRAINT "PK\_TransacoesFinanceiras" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_TransacoesFinanceiras\_Lotes\_LoteId" FOREIGN KEY ("LoteId") REFERENCES "Lotes" ("Id") ON DELETE CASCADE,  CONSTRAINT "FK\_TransacoesFinanceiras\_Usuarios\_UsuarioId" FOREIGN KEY ("UsuarioId") REFERENCES "Usuarios" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

**Quadro 58 -** Tabela de Sensores

|  |
| --- |
| CREATE TABLE "Sensores" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Tipo" text NOT NULL,  "IdentificadorUnico" text NOT NULL,  "GranjaId" integer NOT NULL,  CONSTRAINT "PK\_Sensores" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_Sensores\_Granjas\_GranjaId" FOREIGN KEY ("GranjaId") REFERENCES "Granjas" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

**Quadro 59 -** Tabela de Leitura de Sensores

|  |
| --- |
| CREATE TABLE "LeiturasSensores" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "Valor" numeric NOT NULL,  "Timestamp" timestamp with time zone NOT NULL,  "SensorId" integer NOT NULL,  CONSTRAINT "PK\_LeiturasSensores" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_LeiturasSensores\_Sensores\_SensorId" FOREIGN KEY ("SensorId") REFERENCES "Sensores" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

Quadro 60 - Tabela de Consumo de Água

|  |
| --- |
| CREATE TABLE "ConsumosAgua" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "LoteId" integer NOT NULL,  "Data" timestamp with time zone NOT NULL,  "QuantidadeLitros" numeric(10,3) NOT NULL,  "AvesVivas" integer NOT NULL,  "TemperaturaAmbiente" numeric,  "Observacoes" character varying(500),  "DataCriacao" timestamp with time zone NOT NULL,  CONSTRAINT "PK\_ConsumosAgua" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_ConsumosAgua\_Lotes\_LoteId" FOREIGN KEY ("LoteId") REFERENCES "Lotes" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

Quadro 61 - Tabela de Consumo de Ração

|  |
| --- |
| CREATE TABLE "ConsumosRacao" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "LoteId" integer NOT NULL,  "Data" timestamp with time zone NOT NULL,  "QuantidadeKg" numeric(10,3) NOT NULL,  "TipoRacao" character varying(50) NOT NULL,  "AvesVivas" integer NOT NULL,  "Observacoes" character varying(500),  "DataCriacao" timestamp with time zone NOT NULL,  CONSTRAINT "PK\_ConsumosRacao" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_ConsumosRacao\_Lotes\_LoteId" FOREIGN KEY ("LoteId") REFERENCES "Lotes" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

Quadro 62 - Tabela de Eventos Sanitários

|  |
| --- |
| CREATE TABLE "EventosSanitarios" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "LoteId" integer NOT NULL,  "Data" timestamp with time zone NOT NULL,  "TipoEvento" character varying(50) NOT NULL,  "Produto" character varying(200) NOT NULL,  "LoteProduto" character varying(100),  "Dosagem" character varying(100),  "ViaAdministracao" character varying(50),  "AvesTratadas" integer,  "DuracaoTratamentoDias" integer,  "PeriodoCarenciaDias" integer,  "ResponsavelAplicacao" character varying(200),  "Sintomas" character varying(1000),  "Observacoes" character varying(1000),  "Custo" numeric,  "DataCriacao" timestamp with time zone NOT NULL,  CONSTRAINT "PK\_EventosSanitarios" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_EventosSanitarios\_Lotes\_LoteId" FOREIGN KEY ("LoteId") REFERENCES "Lotes" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

Quadro 63 - Tabelas de Medições de Qualidade de Ar

|  |
| --- |
| CREATE TABLE "MedicoesQualidadeAr" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "LoteId" integer NOT NULL,  "DataHora" timestamp with time zone NOT NULL,  "NH3\_ppm" numeric(6,2),  "CO2\_ppm" numeric(8,2),  "O2\_percentual" numeric,  "VelocidadeAr\_ms" numeric,  "Luminosidade\_lux" numeric,  "TemperaturaAr" numeric,  "UmidadeRelativa" numeric,  "LocalMedicao" character varying(100),  "EquipamentoMedicao" character varying(200),  "Observacoes" character varying(500),  "DataCriacao" timestamp with time zone NOT NULL,  CONSTRAINT "PK\_MedicoesQualidadeAr" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_MedicoesQualidadeAr\_Lotes\_LoteId" FOREIGN KEY ("LoteId") REFERENCES "Lotes" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

Quadro 64 - Tabela de Pesagens Semanais

|  |
| --- |
| CREATE TABLE "PesagensSemanais" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "LoteId" integer NOT NULL,  "DataPesagem" timestamp with time zone NOT NULL,  "IdadeDias" integer NOT NULL,  "SemanaVida" integer NOT NULL,  "PesoMedioGramas" numeric(8,2) NOT NULL,  "QuantidadeAmostrada" integer NOT NULL,  "PesoMinimo" numeric,  "PesoMaximo" numeric,  "DesvioPadrao" numeric,  "CoeficienteVariacao" numeric,  "GanhoSemanal" numeric,  "Observacoes" character varying(500),  "DataCriacao" timestamp with time zone NOT NULL,  CONSTRAINT "PK\_PesagensSemanais" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_PesagensSemanais\_Lotes\_LoteId" FOREIGN KEY ("LoteId") REFERENCES "Lotes" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

Quadro 65 - Tabela de Registros de Abate

|  |
| --- |
| CREATE TABLE "RegistrosAbate" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "LoteId" integer NOT NULL,  "DataAbate" timestamp with time zone NOT NULL,  "DataAbatePrevista" timestamp with time zone,  "IdadeAbateDias" integer NOT NULL,  "QuantidadeEnviada" integer NOT NULL,  "PesoVivoTotalKg" numeric(10,3) NOT NULL,  "PesoCarcacaTotalKg" numeric,  "AvesCondenadas" integer,  "MotivoCondenacoes" character varying(1000),  "PesoCondenadoKg" numeric,  "FrigorificoDestino" character varying(200),  "Transportadora" character varying(200),  "ValorPorKg" numeric,  "ValorTotalRecebido" numeric,  "Observacoes" character varying(1000),  "DataCriacao" timestamp with time zone NOT NULL,  CONSTRAINT "PK\_RegistrosAbate" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_RegistrosAbate\_Lotes\_LoteId" FOREIGN KEY ("LoteId") REFERENCES "Lotes" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

Quadro 66 - Tabela de Registro de Mortalidade

|  |
| --- |
| CREATE TABLE "RegistrosMortalidade" (  "Id" integer GENERATED BY DEFAULT AS IDENTITY,  "LoteId" integer NOT NULL,  "Data" timestamp with time zone NOT NULL,  "Quantidade" integer NOT NULL, -- era QuantidadeMortas  "Motivo" character varying(200), -- era CausaPrincipal  "Setor" character varying(100),  "Observacoes" character varying(1000),  "ResponsavelRegistro" character varying(200),  "DataCriacao" timestamp with time zone NOT NULL,  CONSTRAINT "PK\_RegistrosMortalidade" PRIMARY KEY ("Id"),  CONSTRAINT "FK\_RegistrosMortalidade\_Lotes\_LoteId" FOREIGN KEY ("LoteId") REFERENCES "Lotes" ("Id") ON DELETE CASCADE  ); |

**Fonte:** Elaborado pelos autores

## Mapeamento Objeto Relacional – ORM

O Mapeamento Objeto-Relacional, conhecido pela sigla ORM (*Object-Relational Mapping*), é uma técnica de programação que converte dados entre um banco de dados relacional e a estrutura de objetos de uma aplicação. Em essência, um ORM atua como uma camada de tradução, permitindo que o desenvolvedor manipule os dados do banco (tabelas, registros e colunas) como se fossem objetos nativos da linguagem de programação utilizada, como C#, Java ou Python. A principal vantagem dessa abordagem é abstrair a complexidade da linguagem SQL, aumentando a produtividade e reduzindo a incidência de erros, pois o desenvolvedor pode focar na lógica de negócios da aplicação em vez de escrever consultas SQL repetitivas (CASA DO DESENVOLVEDOR, 2024).

No contexto de um projeto desenvolvido com a tecnologia .NET, como o Granja Tech, o framework ORM mais utilizado é o Entity Framework Core (EF Core). Ele é a ponte que conecta a aplicação ao banco de dados, que no caso do projeto é o PostgreSQL. O EF Core implementa o padrão conhecido como *Code-First*, onde o desenvolvedor primeiro define as classes de modelo em C# (as "entidades"), e a partir delas, o framework tem a capacidade de gerar e atualizar automaticamente o esquema do banco de dados (ARGHANDABI, 2019). Isso significa que as tabelas, colunas, tipos de dados e relacionamentos são criados com base na estrutura das classes C#, garantindo que o banco de dados esteja sempre sincronizado com o código da aplicação.

# ARQUITETURA DE SOFTWARE

A arquitetura de software pode ser compreendida como a estrutura fundamental que organiza os componentes de um sistema e define como eles interagem entre si. Ela atua como um guia para o desenvolvimento, manutenção e evolução do software, conectando os requisitos levantados com as soluções técnicas que irão sustentá-lo. Segundo Bass, Clements e Kazman (2016), a arquitetura de software é responsável por lidar com a complexidade de sistemas modernos, assegurando modularidade, escalabilidade e qualidade.

No caso do GranjaTech, a arquitetura é o alicerce que permite integrar em uma única plataforma diferentes áreas de gestão da granja, como controle produtivo, sanitário, ambiental e financeiro. Ao definir claramente as responsabilidades de cada módulo e suas interfaces de comunicação, garante-se maior robustez, clareza no desenvolvimento e facilidade de manutenção futura. Como destaca Sommerville (2018), uma arquitetura bem planejada reduz riscos, aumenta a confiabilidade e simplifica o ciclo de vida do software.

## 6.1 Arquitetura de desenvolvimento

A arquitetura de desenvolvimento do GranjaTech foi projetada para garantir modularidade, escalabilidade e manutenibilidade, dividindo o sistema em duas partes principais: o back-end (servidor) e o front-end (cliente). Essa abordagem desacoplada permite que as equipes de desenvolvimento trabalhem de forma independente e que a aplicação possa se adaptar a diferentes plataformas no futuro.

O back-end foi desenvolvido em C# com o framework .NET, utilizando a arquitetura ASP.NET Core Web API para construir serviços RESTful. Essa API é responsável por toda a lógica de negócio, processamento de dados e comunicação com o banco de dados. Para o armazenamento e a gestão dos dados, foi empregado o PostgreSQL, um sistema de gerenciamento de banco de dados relacional de código aberto, robusto e confiável, administrado através da ferramenta pgAdmin (POSTGRESQL, 2024). A interação com o banco de dados na aplicação é realizada pelo Entity Framework Core, um mapeador objeto-relacional (ORM) que abstrai a complexidade das consultas. Estruturalmente, o back-end adota os princípios da Clean Architecture (Arquitetura Limpa) e do Domain-Driven Design (DDD). Essa escolha organiza o código em camadas distintas (Domain, Application, Infrastructure e API), isolando as regras de negócio de detalhes de implementação, o que facilita os testes e a manutenção do sistema (EVANS, 2004; MARTIN, 2019).

O processo de design da interface do usuário foi iniciado com a prototipação de alta fidelidade na ferramenta Figma, onde as telas e o fluxo de navegação foram desenhados e validados antes do desenvolvimento (FIGMA, 2024).

Posteriormente, o front-end do GranjaTech foi construído como uma Single-Page Application (SPA) utilizando React, uma biblioteca JavaScript para a criação de interfaces de usuário. Essa abordagem permite uma experiência mais fluida e dinâmica, pois as atualizações de conteúdo ocorrem sem a necessidade de recarregar a página inteira. A interface foi desenvolvida com componentes reutilizáveis da biblioteca Material-UI, garantindo um design consistente e responsivo. A comunicação entre o front-end e o back-end é realizada por meio de requisições HTTP à API RESTful.

Para o controle de versão de todo o código-fonte, tanto do back-end quanto do front-end, foi utilizada a plataforma GitHub, que permite o trabalho colaborativo e o rastreamento de todas as alterações realizadas no projeto (GITHUB, 2024).

## 6.1.1 Back-End

O back-end do GranjaTech foi desenvolvido em C#, linguagem amplamente utilizada no ecossistema .NET. Schildt (2009) descreve o C# como uma das linguagens mais importantes da Microsoft, ressaltando sua relevância para sistemas modernos e de alta performance. Skeet (2019) complementa que, por ser uma linguagem orientada a objetos, ela é adequada para o desenvolvimento de aplicações robustas e modulares, o que se alinha às necessidades de um sistema complexo de gestão como o GranjaTech.

A arquitetura do back-end foi estruturada em camadas, cada uma com responsabilidades específicas, seguindo boas práticas de modularidade. Fowler (2012) destaca que a separação de responsabilidades em camadas contribui para a manutenibilidade e escalabilidade do sistema. No GranjaTech, a camada de Controllers atua como ponto de entrada das requisições enviadas pela interface do usuário, processando as solicitações, validando dados e encaminhando para a lógica de negócio. Essa abordagem está em consonância com o padrão MVC descrito por Freeman e Robson (2020), que promove organização e eficiência no processamento de dados.

A camada Service concentra a lógica de negócios, onde são aplicadas as regras que garantem a consistência e a integridade das operações do sistema. Martin (2017) ressalta que a separação entre lógica de negócios e persistência reduz o acoplamento entre os componentes e aumenta a flexibilidade do software, algo fundamental no GranjaTech para lidar com funcionalidades como registro de lotes, controle de consumo de ração e água, eventos sanitários e gestão financeira.

Já a camada Repository é responsável pelo acesso e manipulação dos dados armazenados, funcionando como intermediária entre a lógica de negócios e o banco de dados. Evans (2004) observa que a utilização de repositórios fornece abstrações importantes que simplificam a lógica de negócio e favorecem a testabilidade. Essa camada foi implementada com o uso do Entity Framework Core, garantindo o mapeamento objeto-relacional (ORM) e a persistência em PostgreSQL, banco de dados escolhido pela sua confiabilidade e escalabilidade.

Outra camada relevante é a de Modelos (Models/Entities), que define as classes e estruturas de dados centrais do sistema, como Usuario, Granja, Lote, ConsumoRacao, EventoSanitario e TransacaoFinanceira. Lerman (2019) destaca que o uso de contextos no Entity Framework simplifica o mapeamento entre essas entidades e suas respectivas tabelas, reduzindo a complexidade de configuração e otimizando a comunicação com o banco.

Além disso, a aplicação utiliza DTOs (Data Transfer Objects) para transferir dados entre camadas de forma mais segura e eficiente. Lima (2023) aponta que esse padrão ajuda a validar informações e evitar a exposição desnecessária de entidades internas. Complementarmente, os arquivos de configuração como Program.cs são responsáveis por inicializar os serviços, definir dependências e configurar a injeção de dependência, etapa considerada essencial por Proise (2021) para evitar falhas em tempo de execução.

Um exemplo prático é o endpoint POST /api/lote, que permite o cadastro de um novo lote. Esse fluxo inicia no Controller, passa pela Service, que aplica as regras de negócio, e segue até a Repository, onde ocorre a persistência no banco de dados. A resposta é retornada ao usuário no formato JSON, assegurando integração com o front-end.

Com essa organização, o GranjaTech apresenta uma arquitetura modular e escalável, que favorece a manutenção e evolução contínua do sistema, garantindo robustez e eficiência na gestão integrada de granjas, em conformidade com os princípios defendidos por autores clássicos da engenharia de software.

A Figura 26 exibe o diagrama de componentes da arquitetura do back-end do sistema GranjaTech, desenvolvido sobre a plataforma .NET. A estrutura adota os princípios da Arquitetura Limpa (Clean Architecture), que organiza o software em camadas de responsabilidades, com um fluxo de dependências direcionado para o núcleo do negócio, garantindo baixo acoplamento e alta manutenibilidade (MARTIN, 2019).

O sistema é dividido em quatro componentes principais. O componente Domain Layer representa o núcleo da aplicação, contendo as entidades de negócio (Lote, Usuario, etc.) e as regras de domínio mais críticas, sendo totalmente independente de tecnologias externas. A camada Application Layer orquestra os casos de uso do sistema, contendo a lógica de aplicação e as interfaces dos serviços (ILoteService), e depende exclusivamente da camada de Domínio.

A camada API Layer serve como ponto de entrada para as requisições externas, expondo os endpoints da API RESTful através dos Controllers. Este componente depende da Application Layer para invocar as operações de negócio, atuando como uma camada de apresentação que traduz as requisições HTTP em chamadas de serviço. Por fim, a Infrastructure Layer implementa os detalhes técnicos e as dependências de frameworks externos. Nela se encontram os repositórios de dados que implementam as interfaces da camada de Aplicação, o contexto do banco de dados (GranjaTechDbContext) e a comunicação com serviços externos.

As dependências externas são representadas pelos componentes Entity Framework Core, o framework de mapeamento objeto-relacional (ORM) utilizado para a persistência de dados, e o Banco de Dados PostgreSQL, onde as informações são efetivamente armazenadas. Conforme a regra de dependência da Arquitetura Limpa, a Infrastructure Layer depende das abstrações da Application Layer, invertendo o controle e garantindo que o núcleo do sistema permaneça isolado de detalhes de implementação.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 26 - Diagrama de Componentes da Arquitetura do Back-End

**Fonte**: Elaborado pelos autores

## 6.1.2 Front-End – Web

O front-end do GranjaTech foi desenvolvido com base em uma arquitetura modular e escalável, garantindo clareza na organização do código e facilidade de manutenção. Meloni (2018) destaca que o front-end é a camada da aplicação que mantém contato direto com o usuário, sendo responsável por exibir as informações processadas no back-end e assegurar uma interação eficiente e intuitiva.

Para a implementação dessa camada foi utilizada a biblioteca React, reconhecida por permitir o desenvolvimento baseado em componentes reutilizáveis, que se combinam para formar interfaces completas (REACT, 2024). Essa abordagem possibilita maior modularidade, reduz o acoplamento e facilita a manutenção incremental do sistema. Além disso, o uso do Figma na etapa de prototipagem viabilizou a criação e validação prévia das telas, assegurando alinhamento com os requisitos funcionais do sistema (FIGMA, 2024).

A estrutura do projeto foi organizada a partir do diretório raiz src, dividido em pastas de acordo com responsabilidades específicas. O diretório assets é destinado ao armazenamento de recursos estáticos, como imagens e arquivos de estilo. O diretório components concentra os elementos reutilizáveis da interface, a exemplo de botões, formulários e ícones. As páginas completas do sistema, como Login, Dashboard, Lotes e Financeiro, encontram-se no diretório pages, permitindo a centralização das funcionalidades. A navegação entre essas páginas é gerenciada pelo diretório routes, implementado com o uso da biblioteca React Router, que possibilita a definição de rotas, carregamento de dados e execução de mutações de acordo com os parâmetros de URL (REACT ROUTER, 2024). Já a comunicação com o back-end é realizada a partir do diretório services, no qual se utiliza o Axios, cliente HTTP baseado em promessas que facilita a integração com APIs tanto em navegadores quanto no ambiente Node.js (AXIOS, 2024).

Um princípio fundamental na construção do front-end do GranjaTech é a componentização. Conforme enfatiza a documentação oficial do React (2024), a criação de componentes independentes permite maior reutilização e reduz os riscos de falhas ao realizar alterações. Na tela de Dashboard (Figura 22), por exemplo, é possível observar a aplicação dessa abordagem. A interface é formada por três componentes principais: a Navbar, localizada na parte superior, responsável por disponibilizar acesso a notificações, preferências de acessibilidade e informações de perfil; a Sidebar, posicionada à esquerda, que organiza a navegação entre os módulos do sistema, como Lotes, Financeiro, Sanitário e Relatórios; e o Content, área central da tela, que exibe dinamicamente as informações específicas de cada página, conforme a rota selecionada.

Essa organização modular e baseada em componentes assegura maior clareza no código, promove a reutilização de elementos visuais e facilita a manutenção. Além disso, garante que o sistema possa ser expandido de maneira contínua, sem comprometer a estabilidade das funcionalidades já existentes, atendendo, assim, aos princípios de escalabilidade e evolução contínua defendidos pela literatura especializada em desenvolvimento de software.

A Figura 27 ilustra o diagrama de componentes que descreve a arquitetura do front-end web do sistema GranjaTech, desenvolvido como uma Single-Page Application (SPA) com a biblioteca React. O diagrama define os limites da aplicação, seus módulos internos e as dependências de bibliotecas externas, seguindo a modelagem padrão da UML.

O componente principal, Front-End Application (Web), encapsula a estrutura lógica do código-fonte, que é organizada em componentes modulares com responsabilidades distintas. O componente Pages (Views) representa as telas principais da aplicação, como o Dashboard e a gestão de Lotes. Para construir suas interfaces, este componente depende do UI Components, que agrupa elementos visuais reutilizáveis, como barras de navegação, botões e contêineres de página.

A lógica de navegação entre as diferentes páginas é gerenciada pelo componente de Routing, que utiliza a biblioteca externa React Router. A gestão do estado global da aplicação, como os dados de autenticação do usuário, é centralizada no componente State Management (Context). As interações com o back-end são abstraídas pelo componente API Services, que utiliza a biblioteca Axios para realizar requisições HTTP à API do sistema.

As dependências entre os componentes internos demonstram um fluxo de dados coeso: as Pages consomem funcionalidades dos UI Components, do State Management e dos API Services para compor a experiência do usuário. Toda a estrutura é fundamentada na biblioteca React, enquanto a consistência visual é mantida pela biblioteca de componentes Material-UI. Essa arquitetura componentizada promove a reutilização de código, facilita a manutenção e permite a escalabilidade da aplicação de forma organizada.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 27 - Diagrama de Componentes da Arquitetura do Front-End (Web)

**Fonte**: Elaborada pelos autores.

## 6.1.3 Front-End – Mobile

Compreendendo a necessidade de mobilidade na gestão de uma granja, onde o produtor frequentemente precisa registrar dados diretamente do campo, o desenvolvimento de um front-end mobile para o GranjaTech foi uma decisão estratégica. A aplicação mobile foi concebida para oferecer as funcionalidades essenciais do sistema em um formato acessível e otimizado para dispositivos como smartphones e tablets, garantindo que o registro de informações seja ágil e possa ser realizado em tempo real, mesmo offline.

A arquitetura de desenvolvimento do front-end mobile foi estruturada utilizando React Native, um framework que permite a criação de aplicações nativas para Android e iOS a partir de uma base de código unificada em JavaScript e React (REACT NATIVE, 2024). Essa escolha se mostrou vantajosa por permitir o reaproveitamento de parte da lógica e da componentização já desenvolvida para a versão web, acelerando o ciclo de desenvolvimento e garantindo consistência entre as plataformas. A organização da arquitetura foi dividida em camadas de software com responsabilidades bem definidas, visando a modularidade e a manutenibilidade.

A Camada de Apresentação (UI Layer) é responsável por toda a interface com o usuário, contendo as telas (screens) e os componentes visuais. Para sua implementação, foi utilizada a biblioteca React Native Paper, um conjunto de componentes baseados no Material Design que garante uma interface intuitiva, responsiva e visualmente consistente com a identidade do GranjaTech (REACT NATIVE PAPER, 2024). Para gerenciar o fluxo de telas na aplicação, a Camada de Navegação (Navigation Layer) foi implementada com a biblioteca React Navigation, que estrutura a navegação em pilha (stack) e abas (tabs), permitindo que o produtor transite de forma fluida entre a tela de lotes, o registro de mortalidade e o formulário de consumo de ração.

O estado global da aplicação, como informações do usuário autenticado e dados de lotes ativos, é gerenciado na Camada de Gerenciamento de Estado (State Management Layer) por meio do Context API nativo do React. Essa abordagem permite o compartilhamento de dados entre diferentes componentes sem a complexidade de bibliotecas externas. Por fim, a Camada de Serviços (Service Layer) realiza a comunicação com o back-end utilizando Axios para as requisições HTTP à API RESTful. Esta camada também implementa uma lógica de cache local e sincronização, permitindo que o produtor registre dados mesmo sem conexão com a internet, os quais são sincronizados automaticamente quando a conexão é restabelecida.

A componentização da interface, um princípio fundamental do React, foi amplamente aplicada no desenvolvimento mobile. Em vez de construir telas monolíticas, a interface foi dividida em componentes menores e reutilizáveis. Um exemplo claro pode ser observado na tela de Registro de Mortalidade, que é composta por um componente de cabeçalho (HeaderComponent), um seletor de lotes (LotSelectorComponent), campos de entrada específicos para dados numéricos (NumericInputComponent) e datas (DatePickerComponent), e um botão padronizado para submissão do formulário (SubmitButtonComponent). Essa abordagem não apenas padroniza a interface e o código, mas também facilita a manutenção e a testabilidade de cada parte da aplicação de forma isolada.

A Figura 28 apresenta o diagrama de componentes que ilustra a arquitetura de software adotada para o front-end mobile da aplicação GranjaTech. Este diagrama modela a organização estrutural do sistema, detalhando seus principais componentes, os artefatos que os implementam e as relações de dependência entre eles, conforme os padrões da UML (Linguagem de Modelagem Unificada).

O componente principal, estereotipado como <<Application>>, representa a Aplicação Mobile como um todo, desenvolvida em React Native. Ele encapsula quatro camadas internas principais, que organizam as responsabilidades do software de forma modular. A Camada de Apresentação (UI) é responsável por toda a interface visual e interação com o usuário, contendo os pacotes de screens (telas) e components (componentes reutilizáveis). A Camada de Navegação gerencia o fluxo entre as telas, orquestrando a experiência do usuário. A Camada de Gerenciamento de Estado centraliza e distribui os dados globais da aplicação, como o estado de autenticação e as informações dos lotes ativos. Por fim, a Camada de Serviços abstrai a comunicação com o back-end, realizando as requisições à API e gerenciando a lógica de sincronização de dados.

O diagrama também evidencia as dependências externas do sistema, modeladas como componentes com o estereótipo <<Library>>. A Camada de Navegação depende da biblioteca React Navigation para sua funcionalidade, enquanto a Camada de Apresentação (UI) utiliza a React Native Paper para a construção dos elementos visuais. A Camada de Serviços, por sua vez, depende da biblioteca Axios para a comunicação HTTP.

As relações de dependência, representadas por setas tracejadas, demonstram o fluxo de controle e dados entre as camadas. A Camada de Apresentação (UI) depende tanto da Camada de Gerenciamento de Estado para obter dados quanto da Camada de Serviços para executar operações no servidor. A Camada de Navegação depende da Camada de UI para renderizar as telas correspondentes a cada rota. Essa arquitetura modular e desacoplada garante a manutenibilidade, a testabilidade e a escalabilidade do software, facilitando futuras evoluções no sistema.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 28 - Diagrama de Componentes da Arquitetura do Front-End (Mobile)

**Fonte**: Elaborado pelos autores

## 6.1.4 Serviços de Mensageria

Serviços de Mensageria, também conhecidos como Message-Oriented Middleware (MOM), constituem um padrão de arquitetura que viabiliza a comunicação assíncrona e desacoplada entre componentes de software. Diferente de uma comunicação síncrona, na qual um serviço invoca outro e aguarda uma resposta, os componentes trocam informações através de mensagens enviadas a um intermediário, o message broker. Este broker armazena as mensagens em filas ou tópicos, garantindo sua entrega confiável aos serviços consumidores, mesmo que estes estejam temporariamente indisponíveis (BASS; CLEMENTS; KAZMAN, 2012). Essa abordagem é fundamental para construir sistemas resilientes e escaláveis.

Após uma análise da arquitetura implementada no projeto GranjaTech, constatou-se que, para o escopo atual, um serviço de mensageria não era necessário, motivo pelo qual não foi utilizado. A comunicação do sistema foi estruturada em um modelo síncrono, baseado no fluxo de requisição e resposta do protocolo HTTP, característico de uma API RESTful. Tal abordagem mostra-se eficaz e adequada, uma vez que as operações de negócio do sistema, como cadastros e consultas, possuem um tempo de execução baixo e não demandam processamento assíncrono.

No entanto, a integração de um serviço de mensageria representa uma análise relevante para a evolução futura do GranjaTech, especialmente para otimizar funcionalidades que podem se tornar computacionalmente intensivas. A geração de relatórios complexos, por exemplo, poderia ser delegada a um serviço em segundo plano através de uma mensagem, liberando a interface do usuário imediatamente e notificando-o apenas quando o processamento for concluído. Da mesma forma, em um cenário de expansão com múltiplos sensores (IoT), um message broker como o RabbitMQ poderia gerenciar o alto fluxo de dados de leitura, enfileirando-os para serem processados de forma controlada e evitando a sobrecarga da API principal. A implementação de alertas e notificações também seria beneficiada, permitindo que eventos críticos, como variações de temperatura, disparem ações em diversos serviços de forma desacoplada. Portanto, a adoção de uma arquitetura de mensageria é uma melhoria necessária e recomendada para garantir a escalabilidade e a resiliência do sistema em futuras versões.

## 6.2 Segurança da informação

A segurança da informação é um pilar essencial na arquitetura de qualquer sistema de software, sendo responsável por proteger os dados contra acessos não autorizados, alterações indevidas e indisponibilidade. Seus conceitos fundamentais são baseados na tríade da Confidencialidade, Integridade e Disponibilidade (CIA). A confidencialidade assegura que as informações sejam acessíveis apenas por usuários autorizados; a integridade garante que os dados permaneçam exatos e consistentes, não sendo modificados sem permissão; e a disponibilidade certifica que os dados e serviços do sistema estejam acessíveis aos usuários legítimos sempre que necessário (SOMMERVILLE, 2011).

No projeto GranjaTech, a segurança foi implementada em múltiplas camadas para proteger os dados sensíveis da produção e financeiros. O principal mecanismo de controle de acesso é o JSON Web Token (JWT), um padrão aberto para a transmissão segura de informações entre partes como um objeto JSON. O processo de autenticação tem início quando o usuário insere suas credenciais (e-mail e senha) na tela de login. O back-end, através do AuthController, valida essas credenciais e, em caso de sucesso, gera um JWT assinado digitalmente. Este token contém as "declarações" (claims) do usuário, como seu identificador e perfil de acesso (Administrador, Produtor, Financeiro), e possui um tempo de expiração definido para mitigar riscos de uso indevido.

A segurança na comunicação entre o front-end e o back-end é garantida pelo uso do protocolo HTTPS, que criptografa todo o tráfego de dados. Após a autenticação, o token JWT gerado é armazenado de forma segura no cliente (front-end). A cada requisição subsequente para endpoints protegidos da API, o token é enviado no cabeçalho HTTP Authorization no formato Bearer {token}. O back-end, por sua vez, possui um middleware de autenticação configurado para interceptar todas as requisições, validar a assinatura e a data de expiração do JWT. Se o token for válido, o acesso ao recurso solicitado é concedido; caso contrário, a requisição é rejeitada com um status de não autorizado. Esse mecanismo assegura que apenas usuários autenticados e com as permissões adequadas possam interagir com os dados do sistema.

Adicionalmente, foi implementada a auditoria das operações do software, um recurso crucial para a rastreabilidade e a segurança. O sistema registra automaticamente as ações relevantes executadas pelos usuários em uma tabela dedicada (LogsAuditoria). Conforme verificado na entidade LogAuditoria, cada registro armazena informações vitais como o identificador do usuário que realizou a ação, o seu e-mail, a operação efetuada (ex: "Cadastro de Lote"), um carimbo de data e hora (timestamp) e detalhes adicionais sobre o evento. Essa trilha de auditoria permite monitorar o uso do sistema, identificar atividades suspeitas e serve como um registro histórico para fins de conformidade e investigações de segurança, garantindo a integridade e a responsabilidade sobre as operações realizadas na plataforma GranjaTech.

## 6.3 Implantação

A fase de implantação define a arquitetura física e a estratégia de distribuição dos componentes de software que compõem o sistema GranjaTech. Este planejamento é essencial para garantir que a aplicação opere de maneira eficiente, segura e escalável em um ambiente de produção. A abordagem adotada considera as necessidades de servidores para cada parte da aplicação, o uso de tecnologias de contêineres para padronização e a possibilidade de hospedagem em serviços de nuvem.

A infraestrutura necessária para a operação do GranjaTech requer um ambiente com, no mínimo, três servidores distintos para garantir o desacoplamento e a segurança entre as camadas. O primeiro é um Servidor Web, responsável por hospedar e servir a aplicação front-end (React) aos usuários. O segundo é um Servidor de Aplicação, onde o back-end (.NET) é executado, processando toda a lógica de negócio e as requisições da API. Por fim, um Servidor de Banco de Dados é necessário para hospedar a instância do PostgreSQL, garantindo que o armazenamento e a gestão dos dados ocorram de forma isolada e otimizada.

Para padronizar os ambientes de desenvolvimento e produção, garantindo que o software se comporte de maneira consistente independentemente de onde seja executado, o projeto adota a tecnologia de dockerização. A utilização de contêineres Docker permite empacotar as aplicações de front-end e back-end com todas as suas dependências e configurações, eliminando problemas de compatibilidade (DOCKER, 2024). Conforme o Dockerfile presente no projeto, a aplicação back-end utiliza uma abordagem de multi-stage build: primeiramente, uma imagem do SDK do .NET é usada para compilar a aplicação; em seguida, os artefatos compilados são copiados para uma imagem de runtime do ASP.NET, que é mais leve e otimizada para produção. Este processo resulta em um contêiner final menor e mais seguro. Uma abordagem similar é aplicada ao front-end, onde a aplicação React é compilada e servida por um servidor web leve, como o Nginx, dentro de seu próprio contêiner.

O diagrama de implantação, apresentado na Figura 29, modela a arquitetura física do sistema, descrevendo a disposição dos servidores e os protocolos de comunicação entre eles.

Diagrama

O conteúdo gerado por IA pode estar incorreto.Figura 29 - Diagrama de Implantação

**Fonte**: Elaborado pelos autores.

Conforme o diagrama, o usuário final interage com o sistema a partir de seu hardware (computador ou dispositivo móvel) através do protocolo HTTPS. A requisição passa por um Servidor de Firewall, que garante a segurança da rede, e é direcionada ao Servidor Web, que entrega a aplicação front-end. A interface, por sua vez, comunica-se com o Servidor de Aplicação através de uma API RESTful, também via HTTPS. O Servidor de Aplicação processa a lógica de negócio e interage com o Servidor de Banco de Dados através de uma conexão segura para persistir e consultar os dados.

Considerando as necessidades de escalabilidade, disponibilidade e manutenção, a implantação do GranjaTech em um serviço de nuvem, como Microsoft Azure ou Amazon Web Services (AWS), é a abordagem mais recomendada. Plataformas de nuvem oferecem serviços gerenciados que simplificam a infraestrutura, como o Azure App Service ou o AWS Elastic Beanstalk para hospedar as aplicações de front-end e back-end, e o Azure Database for PostgreSQL ou Amazon RDS for PostgreSQL para o banco de dados. A utilização desses serviços permite o escalonamento automático dos recursos conforme a demanda, além de oferecer soluções integradas de backup, monitoramento e segurança, reduzindo a complexidade de gerenciamento da infraestrutura e permitindo que a equipe foque na evolução do software.

# CONCLUSÃO

O desenvolvimento do sistema GranjaTech, conforme detalhado neste documento, partiu da identificação de um desafio central no setor agropecuário: a dependência de métodos manuais e descentralizados para a gestão de granjas, que resultam em ineficiências operacionais, suscetibilidade a erros e dificuldades na tomada de decisão estratégica. A análise dos requisitos e a modelagem do sistema demonstraram a necessidade de uma solução integrada que centralizasse o controle produtivo, sanitário e financeiro, oferecendo aos produtores uma ferramenta robusta para otimizar seus processos.

A arquitetura de software adotada, com um back-end estruturado em Arquitetura Limpa e um front-end modular desenvolvido em React, provou ser uma base sólida e escalável para a aplicação. A implementação de funcionalidades essenciais, como o gerenciamento de lotes, o controle de consumo de insumos, o registro de eventos sanitários e o acompanhamento de transações financeiras, atende diretamente às necessidades levantadas, substituindo planilhas e anotações manuais por um fluxo de dados digital, seguro e consistente. A incorporação de recursos como dashboards e relatórios consolida as informações, transformando dados operacionais em indicadores de desempenho que apoiam uma gestão mais assertiva e baseada em evidências.

As tecnologias empregadas, como .NET para o back-end, React para o front-end, e PostgreSQL para o banco de dados, alinhadas à estratégia de implantação com contêineres Docker, não apenas asseguram a performance e a segurança do sistema, mas também garantem sua manutenibilidade e capacidade de evolução futura. A prototipação e o design da interface, focados na experiência do usuário (UX), resultaram em uma plataforma intuitiva, acessível até mesmo para produtores com pouca familiaridade tecnológica, o que é crucial para a adesão e o sucesso da ferramenta no campo.

Diante do exposto, conclui-se que o desenvolvimento do software GranjaTech é plenamente viável. A solução proposta não apenas resolve os problemas práticos identificados, mas também entrega um valor agregado significativo ao modernizar a gestão avícola, promover a eficiência e a sustentabilidade, e fornecer aos pequenos e médios produtores uma ferramenta competitiva para o mercado atual. O projeto cumpre seus objetivos ao entregar um sistema funcional, escalável e alinhado às demandas do setor, confirmando sua relevância e potencial de impacto positivo.

# REFERÊNCIAS

AEGRO. Plataforma de gestão agrícola. 2025. Disponível em: <https://www.aegro.com.br>. Acesso em: 25 ago. 2025.

AGROPOSTURA. Software para gestão de granjas de postura. 2025. Disponível em: <https://www.agropostura.com.br>. Acesso em: 25 ago. 2025.

AGROSUI. Sistema de gestão para suinocultura. 2025. Disponível em: <https://www.agrosui.com.br>. Acesso em: 25 ago. 2025.

ARGHANDABI, H. ASP.NET Core, Entity Framework Core with PostgreSQL Code First. Faun, 10 abr. 2019. Disponível em: <https://faun.pub/asp-net-core-entity-framework-core-with-postgresql-code-first-4f01f5c5e54b>. Acesso em: 24 ago. 2025.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML: guia do usuário. Trad. Cristina Schiel. Rio de Janeiro: Campus, 1999.

BRASIL. Lei nº 13.146, de 6 de julho de 2015. Institui a Lei Brasileira de Inclusão da Pessoa com Deficiência (Estatuto da Pessoa com Deficiência). Diário Oficial da União, Brasília, DF, 7 jul. 2015. Disponível em: <http://www.planalto.gov.br/ccivil\_03/\_ato2015-2018/2015/lei/l13146.htm>. Acesso em: 2025.

CASA DO DESENVOLVEDOR. Desvendando o mundo do ORM: mapeamento objeto-relacional. Blog Casa do Desenvolvedor, 8 mar. 2024. Disponível em: <https://blog.casadodesenvolvedor.com.br/desvendando-o-mundo-do-orm>. Acesso em: 24 ago. 2025.

DATE, C. J. Introdução a sistemas de banco de dados. 8. ed. Rio de Janeiro: Elsevier, 2019.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Padrões de projeto: soluções reutilizáveis de software orientado a objetos. Trad. André Lúcio V. da Costa. Porto Alegre: Bookman, 1995.

GUEDES, G. T. A. UML 2: uma abordagem prática. 2. ed. São Paulo: Novatec Editora, 2011.

HEUSER, C. A. Projeto de banco de dados. 6. ed. Porto Alegre: Bookman, 2009.

MACHADO, F. B.; ABREU, M. C. Banco de dados: projeto e implementação. 4. ed. São Paulo: LTC, 2018.

MICROSOFT. Criando e configurando um modelo – EF Core. Microsoft Learn, 9 jun. 2025. Disponível em: <https://learn.microsoft.com/ef/core/modeling/>. Acesso em: 24 ago. 2025.

OLIVEIRA, M. de; LIMA, R. Administração de bancos de dados com PostgreSQL. São Paulo: Novatec, 2021.

PRESSMAN, R. S. Engenharia de software: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2010.

ROGERS, Y.; SHARP, H.; PREECE, J. Design de interação: além da interação humano-computador. 3. ed. Porto Alegre: Bookman, 2013.

ROSA, A. Protótipo: entenda o que é, tipos, exemplos e como fazer na prática. 2024. Disponível em: <https://softdesign.com.br/blog/prototipo-baixa-e-alta-fidelidade/#Qual-a-diferenca-entreum-wireframe-e-prototipo>. Acesso em: 24 ago. 2025.

SANTOS, L.; PEREIRA, J. PostgreSQL aplicado: práticas e técnicas para ambientes corporativos. Belo Horizonte: Ciência Moderna, 2022.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Sistemas de banco de dados. 7. ed. São Paulo: McGraw-Hill, 2020.

SOMMERVILLE, I. Engenharia de software. 9. ed. São Paulo: Pearson Addison Wesley, 2011.

UNGER, R.; CHANDLER, C. A project guide to UX design: for user experience designers in the field or in the making. 1. ed. Berkeley: New Riders, 2009.

WORLD WIDE WEB CONSORTIUM. Web Content Accessibility Guidelines (WCAG) 2.1. 2018. Disponível em: <https://www.w3.org/TR/WCAG21/>. Acesso em: 2025

BASS, L.; CLEMENTS, P.; KAZMAN, R. Software Architecture in Practice. 3. ed. Boston: Addison-Wesley Professional, 2012.

EVANS, E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley Professional, 2004.

FIGMA. Figma: The Collaborative Interface Design Tool. 2024. Disponível em: <https://www.figma.com>. Acesso em: 12 set. 2025.

GITHUB. GitHub: Where the world builds software. 2024. Disponível em: <https://github.com>. Acesso em: 12 set. 2025.

MARTIN, R. C. Arquitetura Limpa: O Guia do Artesão para Estrutura e Design de Software. Rio de Janeiro: Alta Books, 2019.

MICROSOFT. ASP.NET Core documentation. Microsoft Docs, 2024. Disponível em: [https://docs.microsoft.com/pt-br/aspnet/core/](https://www.google.com/search?q=https://docs.microsoft.com/pt-br/aspnet/core/&authuser=2). Acesso em: 12 set. 2025.

POSTGRESQL. PostgreSQL: The World's Most Advanced Open Source Relational Database. 2024. Disponível em: <https://www.postgresql.org>. Acesso em: 12 set. 2025.

REACT. Documentação Oficial do React. 2024. Disponível em: <https://react.dev/>. Acesso em: 12 set. 2025.

FOWLER, Martin. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley Professional, 2012.

KEELING, Michael. Design It!: From Programmer to Software Architect. Raleigh: Pragmatic Bookshelf, 2017.

BARBOSA, J. Padrões de Projeto para Acesso a Dados: Repository Pattern. São Paulo: Novatec, 2021.

EVANS, E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2004.

FOWLER, M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2012.

FREEMAN, E.; ROBSON, E. Head First Design Patterns. 2. ed. Sebastopol: O’Reilly Media, 2020.

LERMAN, J. Programming Entity Framework: Code First. Sebastopol: O’Reilly Media, 2019.

LIMA, P. Arquitetura e Padrões de Projeto com DTO. Rio de Janeiro: Ciência Moderna, 2023.

MARTIN, R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston: Prentice Hall, 2017.

MICROSOFT. Documentação oficial do C# e .NET. Disponível em: https://learn.microsoft.com/dotnet/

AXIOS. Axios Documentation. 2024. Disponível em: <https://axios-http.com>. Acesso em: 15 set. 2025.

REACT ROUTER. React Router Documentation. 2024. Disponível em: <https://reactrouter.com>. Acesso em: 15 set. 2025.

DOCKER. Docker overview. 2024. Disponível em: <https://docs.docker.com/get-started/overview/>. Acesso em: 12 set. 2025.

TOTVS. Experiência do usuário: o que é e qual sua importância. 2021. Disponível em: <https://www.totvs.com>. Acesso em: 15 set. 2025.

REACT NATIVE. Documentação Oficial do React Native. 2024. Disponível em: <https://reactnative.dev/>. Acesso em: 15 set. 2025.

REACT NATIVE PAPER. Documentação Oficial do React Native Paper. 2024. Disponível em: <https://reactnativepaper.com/>. Acesso em: 15 set. 2025.