UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

RAFAEL SEDOR OLIVEIRA DEDA

UTFPETS: UMA APLICAÇÃO WEB PARA GERENCIAMENTO DE REFEIÇÕES E CONTROLE NUTRICIONAL DE PETS

GUARAPUAVA

2025

RAFAEL SEDOR OLIVEIRA DEDA

UTFPETS: UMA APLICAÇÃO WEB PARA GERENCIAMENTO DE REFEIÇÕES E CONTROLE NUTRICIONAL DE PETS

UTFPets: A Web Application for Meal Management and Nutritional Control of Pets

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

Orientador: Prof. Dr. Roni Fabio Banaszewski

GUARAPUAVA

2025

RESUMO

A alimentação adequada de animais de estimação é essencial para garantir saúde e bem-estar, especialmente em um contexto de crescimento do número de animais domésticos. Entretanto, diversos tutores enfrentam dificuldades ao gerenciar uma dieta equilibrada, em razão de rotinas agitadas e de lacunas no conhecimento técnico, o que pode acarretar problemas de saúde como obesidade e outras doenças crônicas. O presente trabalho descreve o desenvolvimento de um aplicativo denominado UTFPets, que permite registrar dados sobre a saúde e alimentação dos animais, receber lembretes de atividades e compartilhar informações com outros cuidadores ou estabelecimentos especializados. Entre os principais recursos, destacam-se o cadastro de usuário e do perfil do animal, lembretes para atividades essenciais, controle alimentar e seções educativas sobre segurança alimentar e nutrição. Além disso, a aplicação inclui o envio de notificações para garantir a entrega eficaz de lembretes e informações. Com isso, há a intenção de que o UTFPets simplifique o controle nutricional e promova a saúde dos animais, oferecendo aos tutores mais conscientização sobre os cuidados necessários e melhorando a qualidade de vida dos pets e a rotina dos cuidadores.

Palavras-chave: gestão de pets; nutrição animal; controle alimentar; aplicativo para pets; tecnologia em saúde animal.

ABSTRACT

Proper nutrition for pets is essential to ensure their health and well-being, especially in the context of an increasing number of domestic animals. However, many pet owners face challenges in managing a balanced diet due to busy routines and a lack of technical knowledge, resulting in health issues such as obesity and other chronic diseases. This project aims to develop an application called UTFPets. UTFPets will allow pet owners to record important health and nutrition data about their pets, receive activity reminders, and share information with other caregivers or establishments, such as pet hotels. Among its main features, the app will include user registration, animal profiles, reminders for essential activities, dietary control, and educational information about food safety and nutrition. The proposal also involves the use of technologies like Firebase Cloud Messaging (FCM), enabling push notifications to ensure that users receive relevant reminders and information conveniently. UTFPets is expected to simplify nutritional management and promote pet health by providing users with greater control and awareness of pet care, thereby improving the quality of life for pets and the daily routines of their caregivers.

Keywords: pet management; animal nutrition; dietary control; pet app; health technology for pets.

LISTA DE FIGURAS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	Application Programming Interface (Interface de Programação de Aplicações)
CDN	Content Delivery Network (Rede de Distribuição de Conteúdo)
CI/CD	Continuous Integration/Continuous Deployment (Integração Contínua/Implantação Contínua)
CRUD	Create, Read, Update, Delete (Criar, Ler, Atualizar e Excluir)
CSS	Cascading Style Sheets (Folhas de Estilo em Cascata)
E2E	End-to-End (Extremo a Extremo)
FCM	Firebase Cloud Messaging (Mensageria em Nuvem do Firebase)
GCP	Google Cloud Platform (Plataforma de Nuvem do Google)
GDPR	General Data Protection Regulation (Regulamento Geral de Proteção de Dados)
HTML	HyperText Markup Language (Linguagem de Marcação de Hipertexto)
HTTP	Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)
HTTPS	Hypertext Transfer Protocol Secure (Protocolo de Transferência de Hipertexto Seguro)
IAM	Identity and Access Management (Gerenciamento de Identidade e Acesso)
JSON	JavaScript Object Notation (Notação de Objetos JavaScript)
JWT	JSON Web Token (Token Web JSON)
LGPD	Lei Geral de Proteção de Dados
MVC	Model-View-Controller (Modelo-Visão-Controlador)
MVP	Minimum Viable Product (Produto Mínimo Viável)
ORM	Object-Relational Mapping (Mapeamento Objeto-Relacional)
PHP	PHP: Hypertext Preprocessor (Pré-processador de Hipertexto PHP)
PWA	Progressive Web App (Aplicativo Web Progressivo)

RBAC	Role-Based Access Control (Controle de Acesso Baseado em Funções)
REST	Representational State Transfer (Transferência de Estado Representacional)
SPA	Single Page Application (Aplicação de Página Única)
SQL	Structured Query Language (Linguagem de Consulta Estruturada)
SSL	Secure Sockets Layer (Camada de Soquetes Seguros)
UI	User Interface (Interface do Usuário)
URL	Uniform Resource Locator (Localizador Uniforme de Recursos)
UX	User Experience (Experiência do Usuário)
VM	Virtual Machine (Máquina Virtual)
WCAG	Web Content Accessibility Guidelines (Diretrizes de Acessibilidade para Conteúdo Web)

SUMÁRIO

1 INTRODUÇÃO

A alimentação adequada é essencial para a saúde e o bem-estar de animais de estimação, especialmente em um momento em que cada vez mais pessoas estão acolhendo pets em suas casas e os tratando como parte da família. Uma nutrição balanceada não apenas previne doenças crônicas, como obesidade e diabetes, mas também contribui para que os animais tenham uma vida mais longa e saudável (FASCETTI; DELANEY, 2012). Entretanto, a obesidade é uma das condições nutricionais mais comuns entre cães e gatos, resultando em complicações que podem incluir problemas metabólicos e aumento nos custos com cuidados veterinários (LAFLAMME; FLAMMER; HANSEN, 2008; GERMAN, 2022).

Estudos recentes reforçam a relevância do controle alimentar de animais de estimação. A obesidade figura entre as principais doenças nutricionais em cães e gatos, frequentemente associada a dietas inadequadas e à falta de rotina nos cuidados diários (GERMAN, 2022; CHANDLER, 2022). De acordo com a Association for Pet Obesity Prevention, aproximadamente 59% dos cães e 61% dos gatos nos Estados Unidos e Europa apresentam excesso de peso, evidenciando um problema de saúde recorrente (Association for Pet Obesity Prevention, 2024). Além disso, a ausência de comunicação entre múltiplos cuidadores e o registro desorganizado de informações podem agravar esses quadros, reduzindo a qualidade de vida dos animais (KOGAN; HELLYER, 2023). Nesse contexto, soluções tecnológicas despontam como instrumentos valiosos para organizar e monitorar rotinas de alimentação e medicação, promovendo práticas mais consistentes de cuidado e bem-estar animal.

Apesar da crescente conscientização sobre a importância da nutrição animal, muitos tutores enfrentam dificuldades práticas no manejo diário. A rotina corrida, somada à falta de comunicação entre múltiplos cuidadores em um mesmo lar, pode resultar em alimentação descontrolada, duplicação de porções ou esquecimento de medicações (KOGAN; HELLYER, 2023). Essas falhas comprometem a saúde dos pets e evidenciam a necessidade de ferramentas que auxiliem na organização e coordenação dos cuidados.

Nesse contexto, a tecnologia apresenta-se como aliada para centralizar informações, automatizar lembretes e facilitar o compartilhamento de responsabilidades. A proposta deste trabalho é desenvolver uma aplicação web progressiva que permita registrar dados de saúde dos animais, gerenciar rotinas alimentares, receber notificações automáticas e compartilhar informações com cuidadores autorizados, contribuindo para uma gestão mais eficiente e colaborativa dos cuidados com pets.

1.1 Objetivo geral

Desenvolver uma aplicação web e progressiva Progressive Web App (Aplicativo Web Progressivo) (PWA) que centralize e facilite a gestão dos cuidados e informações de animais de estimação, possibilitando o registro de dados importantes, o envio de lembretes para atividades

rotineiras e o compartilhamento de informações com outros cuidadores ou estabelecimentos especializados, como hotéis para pets.

1.2 Objetivos específicos

Para alcançar o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

- Implementar um sistema de cadastro de usuário e perfil dos pets, permitindo o registro de informações detalhadas de cada animal e a criação de uma base personalizada para acompanhamento de saúde e bem-estar;
- Desenvolver funcionalidades de lembretes e notificações, de modo a auxiliar os tutores na manutenção de um plano de cuidado consistente, incluindo horários de alimentação, medicação e consultas;
- Criar um módulo de controle alimentar, no qual seja possível monitorar a dieta dos pets, controlar porções e frequência de refeições, promovendo uma nutrição equilibrada e individualizada;
- Implementar uma funcionalidade de compartilhamento de informações com controle granular de permissões, viabilizando a colaboração entre diferentes cuidadores e garantindo que múltiplos responsáveis possam seguir a mesma rotina alimentar e de cuidado;
- Desenvolver um painel administrativo para gestão do sistema, incluindo controle de usuários, auditoria de ações e monitoramento de operações críticas.

1.3 Justificativa

O crescimento da população de animais de estimação no Brasil e no mundo tem sido exponencial nas últimas décadas. Segundo o Instituto Pet Brasil (Instituto Pet Brasil, 2022), o país possui cerca de 149,6 milhões de animais de estimação, sendo o terceiro maior mercado pet do mundo.

Esse aumento reflete mudanças sociais importantes, nas quais os pets deixaram de ser vistos apenas como animais de companhia para se tornarem membros efetivos das famílias.

Contudo, o crescimento na população de pets não foi acompanhado, na mesma proporção, pelo conhecimento técnico necessário para oferecer cuidados adequados, especialmente no que diz respeito à nutrição. Estudos como os de German (2022) e Laflamme, Flammer e Hansen (2008) demonstram que a obesidade é uma das condições nutricionais mais prevalentes em cães e gatos, estando diretamente relacionada a problemas metabólicos, cardiovasculares e ortopédicos.

A falta de controle sobre as porções, a frequência inadequada de alimentação e o desconhecimento sobre as necessidades nutricionais específicas de cada animal contribuem significativamente para esse cenário.

A rotina cada vez mais agitada dos tutores dificulta o acompanhamento sistemático da alimentação e medicação de seus pets. Muitas vezes, a ausência de um registro organizado leva ao esquecimento de horários importantes e até mesmo à duplicação de refeições quando mais de uma pessoa cuida do mesmo animal.

Essa falta de coordenação pode resultar em consequências graves para a saúde do pet, comprometendo seu bem-estar e qualidade de vida.

Além disso, a colaboração entre múltiplos cuidadores representa um desafio adicional. Em muitos lares, diferentes membros da família ou até mesmo profissionais, como *dog walkers* e *pet sitters*, compartilham a responsabilidade pelos cuidados dos animais. A ausência de uma plataforma centralizada para registro e compartilhamento de informações pode levar a inconsistências na rotina do pet, comprometendo seu bem-estar.

Diante desse contexto, a tecnologia surge como uma aliada importante. Aplicativos móveis e sistemas web têm se mostrado eficazes na organização de rotinas, no registro de informações e na promoção de boas práticas em diversas áreas da saúde (KOGAN; HELLYER, 2023). No entanto, conforme evidenciado na revisão de trabalhos relacionados, as soluções existentes no mercado geralmente focam em aspectos isolados do cuidado pet, como controle de vacinas ou planejamento nutricional, sem oferecer uma abordagem integrada e colaborativa.

O UTFPets justifica-se, portanto, pela necessidade de preencher essa lacuna, oferecendo uma plataforma que integre o gerenciamento alimentar, lembretes automatizados e a possibilidade de compartilhamento de informações entre cuidadores com controle granular de permissões. Ao centralizar essas funcionalidades em um único aplicativo, espera-se facilitar a rotina dos tutores, reduzir erros no manejo dos pets e, consequentemente, promover uma gestão mais eficiente dos cuidados diários.

Além dos benefícios diretos para os tutores e seus pets, o UTFPets também tem relevância acadêmica e profissional. O projeto aplica conceitos modernos de engenharia de software, metodologias ágeis, arquitetura de sistemas web e boas práticas de desenvolvimento, contribuindo para a formação técnica do desenvolvedor e oferecendo um exemplo prático de como a tecnologia pode resolver problemas reais e complexos da sociedade contemporânea.

Por fim, a justificativa do UTFPets reside na busca por promover o bem-estar animal através da tecnologia, capacitando os tutores com ferramentas que facilitem a tomada de decisões informadas sobre a saúde e nutrição de seus pets, e estabelecendo uma base sólida para futuras expansões que possam incluir inteligência artificial, telemedicina veterinária e integração com dispositivos inteligentes.

1.4 Estrutura do trabalho

Este trabalho está organizado em sete capítulos:

- No Capítulo 1, é apresentada a introdução do projeto, destacando a motivação, os objetivos e a justificativa para o desenvolvimento do UTFPets, bem como a estrutura deste trabalho.
- No Capítulo 2, são analisados sistemas similares existentes, realizando-se uma comparação entre as soluções disponíveis, suas funcionalidades, vantagens e limitações.
- No Capítulo 3, é apresentado o referencial teórico que sustenta o projeto, abordando metodologias ágeis de desenvolvimento de software (Scrum, Kanban), testes automatizados, ferramentas de controle de versão e as tecnologias utilizadas no desenvolvimento de aplicações web modernas.
- No Capítulo 4, são descritos os materiais e métodos empregados, incluindo as ferramentas, tecnologias e o processo metodológico utilizado no desenvolvimento do aplicativo.
- No Capítulo 5, são apresentados os artefatos de análise e projeto do sistema, tais como funcionalidades prioritárias, prototipação de telas e modelagem do banco de dados.
- No Capítulo 6, são apresentados os resultados do desenvolvimento do sistema.
- Por fim, no Capítulo 7, apresenta-se a conclusão do trabalho, juntamente com sugestões para evoluções futuras.

2 TRABALHOS RELACIONADOS

Este capítulo apresenta alguns aplicativos existentes no mercado que oferecem funcionalidades para o gerenciamento da saúde e da alimentação de animais de estimação. O objetivo é avaliar em que medida tais soluções atendem às demandas de tutores e como o aplicativo proposto (UTFPets) se diferencia em termos de recursos e abrangência.

Nos últimos anos, surgiram diversas soluções tecnológicas para auxiliar no controle de vacinas, agendamento de consultas, planejamento nutricional e outras práticas relacionadas ao bem-estar dos pets. Entre elas, destacam-se Petzillas, FuncionalPet, PetDesk e Pet Diet Designer, descritos a seguir.

2.1 Petzillas

O Petzillas, ilustrado na Figura 1, é um aplicativo gratuito disponível para *Android* e *iOS*, focado em fornecer aos tutores recursos para acompanhamento de vacinas, medicamentos, vermífugos, antipulgas, consultas veterinárias, controle de peso e higiene. Um de seus pontos fortes é a possibilidade de compartilhar dados com outros cuidadores, facilitando a colaboração em lares com múltiplas pessoas responsáveis pelo mesmo animal.

Figura 1 – Página inicial do site do aplicativo Petzillas

2.2 FuncionalPet

O FuncionalPet, apresentado na Figura 2, é um software voltado principalmente para clínicas veterinárias, com ênfase na elaboração de dietas personalizadas para cães e gatos. Profissionais podem usar a ferramenta para gerar planos nutricionais e acompanhar a evolução do animal. Por outro lado, a versão atual não oferece recursos de lembretes diários ou de compartilhamento de tarefas entre diversos cuidadores.

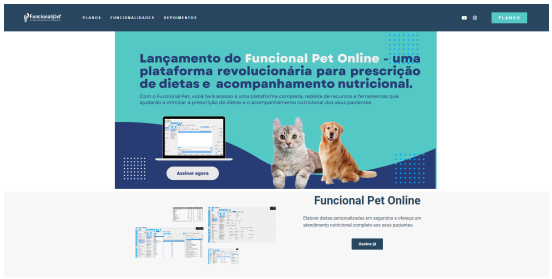


Figura 2 – Página inicial do site do aplicativo FuncionalPet

2.3 PetDesk

O PetDesk, ilustrado na Figura 3, destaca-se por auxiliar no gerenciamento de compromissos dos animais, como consultas veterinárias, lembretes de medicação e controle de vacinas. É uma ferramenta útil para quem necessita de mais organização no acompanhamento da saúde do pet. Entretanto, não contempla um módulo nutricional ou o controle de porções diárias, limitando-se ao gerenciamento de eventos de saúde.

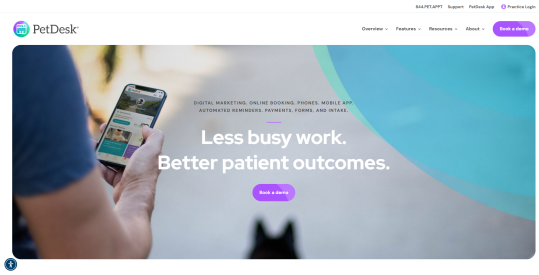


Figura 3 – Página inicial do site do aplicativo PetDesk

2.4 Pet Diet Designer

O Pet Diet Designer, ilustrado na Figura 4, é focado na alimentação de pets, com planejamento de dietas, porções e monitoramento do consumo de água. Embora ofereça informações nutricionais e dicas para manter a saúde dos animais, não prevê o registro de cuidados médicos ou colaboração entre diferentes cuidadores.

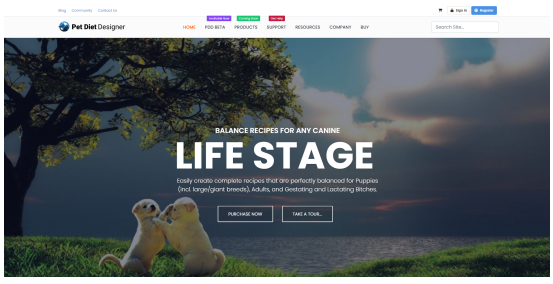


Figura 4 – Página inicial do site do aplicativo Pet Diet Designer

2.5 Estudo comparativo

O Quadro 1 apresenta uma comparação entre os aplicativos analisados e o UTFPets.

Quadro 1 – Comparação de funcionalidades entre aplicativos de gerenciamento pet

Funcionalidade	Petzillas	FuncionalPet	PetDesk	Pet Diet Designer	UTFPets
Gerenciamento de Perfil	Sim	Sim	Sim	Sim	Sim
Controle Alimentar	Parcial	-	-	Sim	Sim
Lembretes e Notificações	Sim	-	Sim	Parcial	Sim
Planejamento Nutricional	-	Sim	-	Sim	Parcial
Compartilhamento Colaborativo	Sim	-	-	-	**Completo**
Histórico de Saúde	Sim	Sim	Sim	-	Sim
Conteúdo Educativo	Parcial	-	Parcial	Sim	Parcial
Multiplataforma	iOS/Android	Web	iOS/Android	iOS/Android	**Web/PWA**
Funcionalidade Offline	-	-	-	-	**Sim**

Fonte: Autoria própria (2025).

Com base na análise comparativa, o UTFPets apresenta os seguintes diferenciais em relação às soluções existentes:

1. **Compartilhamento Colaborativo Avançado**: Enquanto o Petzillas oferece compartilhamento básico, o UTFPets implementa um sistema de compartilhamento com três níveis de permissões (owner, editor, viewer), permitindo controle granular sobre quem pode visualizar, editar ou gerenciar as informações dos pets. Além disso, suporta compartilhamento de locations inteiras, compartilhando automaticamente todos os pets vinculados.

2. **Progressive Web App (PWA)**: O UTFPets é o único entre os analisados que funciona como PWA, permitindo instalação em dispositivos sem necessidade de app stores, funcionamento offline e notificações push mesmo com o aplicativo fechado.

3. **Sistema de Lembretes Inteligentes**: Lembretes com recorrência avançada, incluindo dias da semana específicos, janelas de horário e suporte a múltiplos timezones.

4. **Hierarquia Espacial**: Organização de pets por locations (casas, apartamentos), com suporte a timezone por local, facilitando o gerenciamento de pets em diferentes localizações.

5. **Notificações Multicanal**: Sistema completo de notificações via banco de dados, e-mail e push notifications, com histórico e controle de leitura.

6. **Painel Administrativo**: Sistema de auditoria e gerenciamento para administradores, permitindo supervisão e controle do sistema.

7. **Open Source e Extensível**: Código aberto que permite futuras integrações e extensões, incluindo potencial para inteligência artificial e telemedicina veterinária.

3 REFERENCIAL TEÓRICO

O presente capítulo apresenta os principais fundamentos e tecnologias que embasam o desenvolvimento de aplicações web modernas, com foco em metodologias ágeis, arquitetura de software, ferramentas de controle de versão e serviços em nuvem utilizados no desenvolvimento do projeto.

3.1 Processo de desenvolvimento

O desenvolvimento de sistemas de software modernos demanda metodologias que permitam flexibilidade, adaptação rápida a mudanças e entrega contínua de valor. Nesse contexto, as metodologias ágeis têm se destacado como abordagens eficazes para conduzir projetos de software de forma iterativa e incremental (SCHWABER; SUTHERLAND, 2017).

3.1.1 Scrum

O Scrum é um *framework* ágil amplamente utilizado para o gerenciamento de projetos de desenvolvimento de software. Segundo Schwaber e Sutherland (2017), o Scrum organiza o trabalho em ciclos curtos chamados *sprints*, que geralmente duram de uma a quatro semanas. Cada sprint tem como objetivo entregar um incremento funcional do produto, permitindo *feedback* rápido e ajustes contínuos.

O Scrum define três papéis principais: o *Product Owner*, responsável por priorizar o backlog do produto e definir os requisitos; o *Scrum Master*, que facilita o processo e remove impedimentos; e o Time de Desenvolvimento, que implementa as funcionalidades.

Entre os artefatos do Scrum, destacam-se o *Product Backlog*, que contém todas as funcionalidades desejadas priorizadas; o *Sprint Backlog*, que lista as tarefas selecionadas para a *sprint* atual; e o Incremento, que é a versão funcional do produto ao final de cada sprint. Além disso, o Scrum prevê cerimônias como o planejamento da sprint, reuniões diárias (*daily standups*), revisão da *sprint* e retrospectiva, que promovem transparência, inspeção e adaptação contínuas.

3.1.2 Kanban

O *Kanban* é outro método ágil que complementa o Scrum ao fornecer uma visualização clara do fluxo de trabalho. Segundo Anderson (2010), o *Kanban* utiliza um quadro visual dividido em colunas que representam os estágios do processo de desenvolvimento, como “A Fazer”, “Em Progresso”, “Em Teste” e “Concluído”. Cada tarefa é representada por um cartão que se move pelas colunas conforme avança no processo.

Uma das principais vantagens do Kanban é a identificação de gargalos no fluxo de trabalho. Ao limitar o número de tarefas em progresso simultaneamente (*Work in Progress* - WIP), a equipe consegue focar em concluir tarefas antes de iniciar novas, aumentando a eficiência e reduzindo o tempo de ciclo.

3.1.3 Testes Automatizados

Os testes automatizados consistem na utilização de ferramentas e frameworks que executam automaticamente rotinas de verificação sobre o código-fonte, comparando o resultado obtido com o resultado esperado (MESZAROS, 2007). Essa abordagem contribui para o princípio de integração contínua, uma prática amplamente difundida em metodologias ágeis, na qual o código é constantemente validado a cada atualização do repositório.

De acordo com Fewster e Graham (2012), os testes automatizados podem ser classificados em diferentes níveis de granularidade. Os *testes unitários* validam o comportamento de funções, classes ou componentes isolados, garantindo que pequenas partes do sistema funcionem corretamente. Já os *testes de integração* verificam a comunicação entre módulos distintos, enquanto os *testes de sistema* analisam o comportamento do software como um todo. Em um nível mais amplo, os *testes de aceitação* avaliam se o sistema atende aos requisitos definidos pelo cliente ou usuário final.

Além disso, o uso de testes *end-to-end* (E2E) tem se tornado cada vez mais comum em aplicações web modernas. Esses testes simulam o comportamento do usuário final, interagindo com a interface gráfica e percorrendo fluxos reais do sistema, o que permite avaliar a experiência de uso e identificar falhas que poderiam passar despercebidas em testes isolados (GAROUSI; FELDERER; MäNTYLä, 2018).

3.2 Ferramentas de desenvolvimento

O desenvolvimento de aplicações web modernas demanda o uso de um conjunto integrado de ferramentas que auxiliam na gestão do projeto, no controle de versão, na colaboração entre desenvolvedores e na implantação contínua das soluções em ambiente de produção.

3.2.1 Git, GitFlow e GitHub

O Git é um sistema de controle de versão distribuído amplamente utilizado no desenvolvimento de software. Ele permite que desenvolvedores gerenciem mudanças no código-fonte de forma eficiente, criando ramificações (branches) para desenvolver funcionalidades isoladamente e mesclando (merge) essas mudanças de volta ao código principal quando estão prontas.

O GitFlow é uma estratégia de branching que define um fluxo de trabalho estruturado para projetos Git. Segundo Driessen (2010), o GitFlow organiza o desenvolvimento em branches principais (`main` e `develop`) e branches auxiliares para funcionalidades (`feature`), correções (`hotfix`) e releases.

O GitHub, por sua vez, é uma plataforma de hospedagem de repositórios Git que oferece recursos adicionais como pull requests, issues, actions (CI/CD) e colaboração em equipe.

3.2.2 Google Cloud Platform (GCP)

A Google Cloud Platform é uma plataforma de computação em nuvem que oferece serviços de infraestrutura, armazenamento e banco de dados com alta disponibilidade e escalabilidade. A plataforma oferece monitoramento integrado, logs centralizados e escalabilidade sob demanda, facilitando a manutenção e operação da aplicação em produção.

3.2.3 Monorepo

Monorepo é uma estratégia de organização de código onde múltiplos projetos relacionados são mantidos em um único repositório de versionamento. Diferentemente da abordagem tradicional de múltiplos repositórios (multirepo), onde cada componente ou serviço possui seu próprio repositório, o Monorepo centraliza todo o código-fonte em uma estrutura unificada.

As principais vantagens do Monorepo incluem:

- **Versionamento atômico**: Mudanças que afetam múltiplos componentes podem ser commitadas de forma atômica, garantindo consistência entre frontend, backend e outras partes do sistema;
- **Refatoração simplificada**: Alterações que afetam múltiplos projetos podem ser realizadas em um único commit, facilitando a manutenção e evolução do código;
- **Compartilhamento de código**: Bibliotecas, tipos, interfaces e utilitários podem ser facilmente compartilhados entre diferentes partes do projeto;
- **Testes integrados**: Facilita a execução de testes end-to-end e de integração que envolvem múltiplos componentes do sistema;
- **Configuração centralizada**: Scripts de build, deploy, Continuous Integration/Continuous Deployment (Integração Contínua/Implantação Contínua) (CI/CD) e configurações de ambiente podem ser mantidos em um único local.

No contexto do UTFPets, a arquitetura Monorepo consolidou o backend Laravel, frontend Angular, testes E2E, configurações nginx e scripts de automação em uma estrutura organizada.

Essa decisão baseou-se na necessidade de manter consistência entre as interfaces e Application Programming Interface (Interface de Programação de Aplicações) (API)s, além de simplificar o processo de CI/CD e facilitar o desenvolvimento local através do Docker Compose.

3.3 Tecnologias do lado cliente

3.3.1 Angular

O Angular é um framework desenvolvido pelo Google para a construção de aplicações web dinâmicas e Single Page Applications (Single Page Application (Aplicação de Página Única) (SPA)s). Ele utiliza TypeScript, uma linguagem que adiciona tipagem estática ao JavaScript, aumentando a segurança e facilitando a manutenção do código.

O Angular adota uma arquitetura baseada em componentes, onde cada componente encapsula sua lógica, template HyperText Markup Language (Linguagem de Marcação de Hipertexto) (HTML) e estilos Cascading Style Sheets (Folhas de Estilo em Cascata) (CSS). Essa modularidade facilita a reutilização de código e a organização do projeto. Além disso, o Angular oferece recursos poderosos como injeção de dependências, roteamento, formulários reativos, gerenciamento de estado e comunicação com APIs através do módulo Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto) (HTTP) Client.

3.3.2 Progressive Web App (PWA)

Progressive Web App (PWA) é uma abordagem de desenvolvimento que combina o melhor das aplicações web e nativas, oferecendo experiências de usuário ricas e confiáveis mesmo em condições de conectividade limitada (RUSSELL; FIRTMAN, 2018). Os PWAs utilizam tecnologias como Service Workers, Web App Manifest e Hypertext Transfer Protocol Secure (Protocolo de Transferência de Hipertexto Seguro) (HTTPS) para habilitar funcionalidades como instalação na tela inicial, notificações push e funcionamento offline.

Os Service Workers são scripts que rodam em background, interceptando requisições de rede e permitindo o cache de recursos. Isso possibilita que o aplicativo carregue rapidamente e funcione mesmo quando o dispositivo está offline. O Web App Manifest é um arquivo JavaScript Object Notation (Notação de Objetos JavaScript) (JSON) que define metadados da aplicação, como nome, ícones, cores e orientação, permitindo que o PWA seja instalado como um aplicativo nativo.

3.3.3 Bibliotecas e Ferramentas Complementares

Além do Angular e PWA, o frontend do UTFPets utilizou diversas bibliotecas complementares:

- **RxJS 7.8**: Biblioteca para programação reativa, permitindo o gerenciamento eficiente de eventos assíncronos e streams de dados através de Observables;
- **TailwindCSS 3.4**: Framework CSS utilitário que facilita a criação de interfaces responsivas e modernas através de classes utilitárias, reduzindo a necessidade de CSS customizado;
- **Angular Material 17**: Biblioteca de componentes User Interface (Interface do Usuário) (UI) baseada no Material Design, oferecendo elementos prontos como botões, formulários, diálogos, tabelas e cards;
- **Angular CDK (Component Dev Kit)**: Conjunto de ferramentas para criar componentes customizados com funcionalidades avançadas como drag-and-drop, overlays e acessibilidade.

3.4 Tecnologias do lado servidor

3.4.1 Docker

O Docker é uma plataforma de conteinerização que permite empacotar aplicações e suas dependências em contêineres leves e portáteis (MERKEL, 2014). Um contêiner é uma unidade de software que inclui tudo o que é necessário para executar a aplicação: código, runtime, bibliotecas e configurações do sistema. A principal vantagem do Docker é garantir que a aplicação funcione de forma consistente em qualquer ambiente, eliminando problemas do tipo “funciona na minha máquina”.

O Docker Compose foi utilizado para orquestrar esses contêineres através de arquivos distintos: `docker-compose.local.yml` para desenvolvimento e `docker-compose.yml` para produção, definindo suas configurações, variáveis de ambiente, volumes e redes. Isso simplificou o processo de desenvolvimento e deploy, garantindo consistência entre ambientes.

3.4.2 Laravel

O Laravel é um framework PHP: Hypertext Preprocessor (Pré-processador de Hipertexto PHP) (PHP) moderno e elegante para o desenvolvimento de aplicações web, conhecido

por sua sintaxe expressiva e ferramentas poderosas. Ele segue o padrão arquitetural Model-View-Controller (Modelo-Visão-Controlador) (MVC) (Model-View-Controller) e oferece funcionalidades como roteamento, middlewares, Eloquent Object-Relational Mapping (Mapeamento Objeto-Relacional) (ORM) para manipulação de banco de dados, sistema de autenticação, filas, notificações e muito mais.

3.4.3 PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados relacional open-source conhecido por sua confiabilidade, robustez e conformidade com padrões Structured Query Language (Linguagem de Consulta Estruturada) (SQL). Ele oferece suporte a transações ACID, índices avançados, triggers, views, stored procedures e tipos de dados complexos, como JSON e arrays.

No contexto do Laravel, o conceito de migrations é implementado através do sistema nativo que permite criar, modificar e reverter estruturas de tabelas de forma controlada e versionada.

Cada migration representa uma mudança específica no banco de dados, como a criação de uma tabela, adição de uma coluna ou criação de índices. As migrations são executadas sequencialmente, garantindo que todos os ambientes (desenvolvimento, teste e produção) possuam o mesmo esquema de banco de dados.

3.4.4 Cloudinary

O Cloudinary é um serviço em nuvem especializado no gerenciamento, otimização e entrega de mídia (imagens e vídeos). Ele oferece funcionalidades como upload de arquivos, transformação de imagens (redimensionamento, recorte, aplicação de filtros), compressão automática, conversão de formatos e entrega através de Content Delivery Network (Rede de Distribuição de Conteúdo) (CDN) (Content Delivery Network).

3.4.5 JSON Web Token (JWT)

JSON Web Token (Token Web JSON) (JWT) é um padrão aberto (RFC 7519) para criação de tokens de acesso baseados em JSON, amplamente utilizado para autenticação e autorização em aplicações web e APIs Representational State Transfer (Transferência de Estado Representacional) (REST)ful (JONES; BRADLEY; SAKIMURA, 2015). Um JWT é composto por três partes: header, payload e signature. O payload contém informações sobre o usuário (claims), como ID e roles, e a signature garante a integridade do token.

3.4.6 Firebase Cloud Messaging (FCM)

O Firebase Cloud Messaging é um serviço do Google Firebase que permite enviar notificações push de forma confiável para dispositivos móveis e navegadores web. O Firebase Cloud Messaging (Mensageria em Nuvem do Firebase) (FCM) gerencia toda a infraestrutura de entrega de mensagens, incluindo retry automático, priorização e suporte a múltiplas plataformas.

3.5 Considerações

Este capítulo apresentou os fundamentos teóricos e tecnológicos que sustentam o desenvolvimento de aplicações web modernas, abordando metodologias ágeis, testes automatizados, ferramentas de controle de versão e tecnologias utilizadas no lado cliente e servidor. Esses conceitos formam a base técnica que orienta as decisões de implementação e garantem qualidade, escalabilidade e manutenção eficiente do sistema.

4 MATERIAIS E MÉTODOS

O presente capítulo descreve os recursos tecnológicos e metodológicos empregados no desenvolvimento do UTFPets, abordando tanto as ferramentas utilizadas quanto o processo adotado para organização das tarefas, versionamento do código e validação do sistema. Inicialmente, são apresentados os *materiais* (ferramentas, linguagens e infraestrutura). Em seguida, são detalhados os *métodos*, que compreendem a aplicação de práticas ágeis e o fluxo de desenvolvimento do projeto.

4.1 Materiais

As tecnologias empregadas no UTFPets foram escolhidas com base em critérios de robustez, escalabilidade e facilidade de manutenção. Conforme discutido no Capítulo 3, o projeto foi estruturado a partir de princípios de desenvolvimento web moderno, arquitetura em camadas e conteinerização.

O **frontend** foi implementado com o **framework Angular 17**, utilizado para construir uma interface web responsiva e modular. A estilização foi desenvolvida com **Angular Material** e **TailwindCSS**, possibilitando um design limpo e adaptável a diferentes dispositivos.

O **backend** foi desenvolvido em **Laravel 12.x** com **PHP 8.2**, framework escolhido por sua produtividade, segurança e integração nativa com banco de dados relacionais. O **PostgreSQL** foi adotado como sistema gerenciador de banco de dados pela confiabilidade e suporte a consultas complexas.

Para gerenciamento de mídia e otimização de imagens, foi utilizado o serviço **Cloudinary**. A padronização do ambiente de desenvolvimento e produção foi garantida por meio da conteinerização com **Docker**, que possibilitou a execução isolada dos componentes da aplicação.

O controle de versão foi realizado com **Git** e hospedado no **GitHub**, que também serviu como plataforma de integração e entrega contínua (*CI/CD*) por meio do **GitHub Actions**. A aplicação foi implantada na **Google Cloud Platform (GCP)**, utilizando os serviços *Compute Engine* e *Cloud SQL*, com autenticação gerenciada pelo *Cloud IAM* e certificados HTTPS fornecidos pelo **Let's Encrypt**.

Por fim, adotou-se uma arquitetura **monorepo**, centralizando frontend, backend e scripts de implantação em um único repositório. Essa estrutura facilitou o controle de versões, os testes integrados e o processo de deploy.

4.2 Métodos

O desenvolvimento do UTFPets foi conduzido com base em metodologias ágeis, adaptadas ao contexto de um projeto acadêmico individual supervisionado por um orientador. Essa abordagem favoreceu entregas incrementais e acompanhamento contínuo da evolução do sistema.

4.2.1 Aplicação do Scrum e Kanban

O **Scrum** serviu como estrutura principal para o gerenciamento do projeto, enquanto o **Kanban** foi utilizado de forma complementar para o controle visual das tarefas. O professor orientador atuou como *Product Owner* e *Scrum Master*, responsável pela priorização de funcionalidades e pela remoção de impedimentos. O autor desempenhou o papel de desenvolvedor, executando as etapas de planejamento, implementação e testes.

As atividades foram organizadas em *sprints* de duas semanas, com planejamento no início e revisão ao final de cada ciclo. Nessas reuniões, as entregas eram apresentadas ao orientador, que fornecia feedbacks e sugestões de aprimoramento. O quadro Kanban, integrado ao GitHub, era composto pelas colunas *To Do*, *Doing*, *Testing* e *Done*, refletindo o andamento das tarefas.

4.2.2 Planejamento e controle de tarefas

As funcionalidades foram descritas na forma de **Histórias de Usuário**, permitindo priorização de acordo com a necessidade dos tutores de pets e clareza nos objetivos de cada incremento. Um exemplo de história utilizada é:

> *"Como owner, quero cadastrar um novo pet para acompanhar suas refeições e manter o controle nutricional."*

Cada história foi desdobrada em tarefas menores (*issues*) vinculadas às *branches* correspondentes no repositório Git. Periodicamente, as tarefas concluídas eram revisadas e integradas à *branch* principal.

4.2.3 Integração contínua e versionamento

O projeto utilizou o fluxo de trabalho **GitFlow**, definindo ramificações específicas para funcionalidades, correções e versões estáveis. A automação de integração e entrega contínua (*CI/CD*) foi realizada via **GitHub Actions**, responsável por executar testes, gerar builds e pu-

blicar novas versões no ambiente de testes ou produção na GCP. Essa prática reduziu falhas manuais e aumentou a confiabilidade das entregas.

4.2.4 Ambientes e testes automatizados

O ambiente de desenvolvimento foi padronizado com **Docker**, garantindo consistência entre as máquinas e simplificando a implantação. Foram definidos dois ambientes principais: o ambiente de desenvolvimento, utilizado para testes e ajustes contínuos, e o ambiente de produção, implantado em contêineres na GCP.

Para garantir a qualidade do código, foram aplicados testes automatizados em múltiplos níveis. No **backend**, utilizaram-se **PHPUnit** para testes de unidade e integração. No **frontend**, o **Playwright** foi empregado para testes *end-to-end*, simulando interações reais de usuários. A execução automática desses testes foi integrada ao pipeline de CI/CD, assegurando que cada alteração passasse por verificação antes da liberação.

4.3 Considerações

O capítulo apresentou os principais recursos tecnológicos e metodológicos utilizados no projeto UTFPets, destacando o uso de ferramentas modernas e práticas ágeis adaptadas ao contexto acadêmico. A combinação de metodologias ágeis, integração contínua e conteinerização garantiu um desenvolvimento organizado, seguro e escalável.

No capítulo seguinte, são descritos os artefatos de análise e projeto do sistema, incluindo modelagem de dados, diagramas e principais decisões de arquitetura.

5 ANÁLISE E PROJETO DO SISTEMA

Este capítulo apresenta a concepção do UTFPets de forma estruturada, abordando a priorização de requisitos por meio do método MoSCoW, a arquitetura conceitual (usuários, locais e animais), as funcionalidades essenciais, a modelagem do banco de dados e os protótipos de interface (*wireframes*). Ao final, faz-se referência ao Apêndice A, que detalha a estrutura completa das tabelas do banco de dados.

5.1 Implementação do método MoSCoW

Para priorizar as funcionalidades do UTFPets, foi aplicado o método **MoSCoW**, que classifica os requisitos em quatro categorias:

- **Must Have (Essencial)**: Funcionalidades obrigatórias para o funcionamento do Minimum Viable Product (Produto Mínimo Viável) (MVP), garantindo que a aplicação seja viável para o primeiro lançamento.
- **Should Have (Importante)**: Funcionalidades que melhoram a experiência do usuário ou adicionam valor ao sistema, mas não são imprescindíveis na versão inicial.
- **Could Have (Desejável)**: Recursos adicionais que podem ser incluídos no futuro, ampliando as capacidades do sistema.
- **Won't Have (Não serão implementadas agora)**: Funcionalidades que, embora relevantes, não estão no escopo atual e podem ser consideradas em atualizações posteriores.

Este capítulo foca nas funcionalidades **Must Have**, descritas ao longo das seções seguintes, de modo a viabilizar o lançamento inicial do UTFPets.

5.2 Estrutura do aplicativo

O UTFPets organiza os dados de maneira hierárquica para facilitar o gerenciamento colaborativo. A estrutura conceitual é composta por:

- **Usuário**: Cada usuário pode cadastrar e gerenciar múltiplos pets, definindo permissões para cuidadores ou estabelecimentos.
- **Local**: Representa o ambiente onde os pets estão localizados (ex.: residência, clínica veterinária). Um usuário pode administrar vários locais.
- **Pet**: Cada animal possui um perfil individual (nome, espécie, raça, idade, peso, histórico alimentar).

- **Informações**: Incluem registros de alimentação, lembretes médicos e atividades, proporcionando um acompanhamento diário do pet.

Essa organização possibilita ao usuário manejar diversos pets e conceder níveis de acesso diferenciados, garantindo uma experiência colaborativa no cuidado dos animais.

5.3 Papéis dos usuários e permissões

O UTFPets implementa um sistema de controle de acesso baseado em papéis (Role-Based Access Control (Controle de Acesso Baseado em Funções) (RBAC) - Role-Based Access Control), permitindo que diferentes usuários tenham níveis distintos de permissões sobre os pets e suas informações. Foram definidos quatro papéis principais, cada um com responsabilidades e permissões específicas.

5.3.1 Owner (Proprietário)

O papel de Owner é atribuído ao usuário que cadastrou originalmente o pet no sistema. Este papel possui controle total sobre todas as funcionalidades relacionadas ao pet. As permissões do Owner são:

- Visualizar todas as informações do pet (perfil completo, histórico, estatísticas);
- Editar dados do pet (nome, foto, peso, notas);
- Deletar o pet do sistema (soft delete);
- Criar, editar e deletar refeições;
- Criar, editar e deletar lembretes;
- Gerenciar compartilhamento (convidar, aceitar, revogar acesso);
- Alterar papéis de outros usuários compartilhados;
- Transferir ownership (funcionalidade futura).

5.3.2 Editor

O papel de Editor permite que o usuário contribua ativamente com o cuidado do pet, registrando informações e acompanhando a rotina, mas sem permissão para alterar dados essenciais do pet ou gerenciar compartilhamentos. As permissões do Editor são:

- Visualizar todas as informações do pet;

- Criar, editar e deletar refeições;
- Visualizar lembretes;
- Registrar tarefas concluídas;
- **NÃO PODE**: editar dados do pet, deletar o pet, gerenciar compartilhamento.

Casos de uso típicos:

- Familiares que participam da alimentação do pet;
- Dog walkers e pet sitters;
- Profissionais de hotéis para pets.

5.3.3 Viewer (Visualizador)

O papel de Viewer oferece acesso somente leitura às informações do pet, ideal para situações onde o usuário precisa apenas acompanhar o pet sem interferir na rotina estabelecida. As permissões do Viewer são:

- Visualizar informações do pet (perfil, fotos);
- Visualizar histórico de refeições;
- Visualizar lembretes;
- **NÃO PODE**: criar, editar ou deletar qualquer informação.

Casos de uso típicos:

- Familiares que querem acompanhar, mas não participam diretamente dos cuidados;
- Veterinários para consulta do histórico;
- Membros da família em viagens.

5.3.4 Administrador do Sistema

Além dos papéis relacionados aos pets, existe o papel de Administrador do Sistema, com permissões especiais para gestão global do UTFPets. As permissões do Administrador são:

- Visualizar todos os usuários cadastrados;

- Alterar permissões de usuários (promover/rebaixar administradores);
- Visualizar, editar e deletar todos os pets do sistema (para fins de moderação e manutenção);
- Criar, editar e deletar refeições e lembretes de qualquer pet;
- Acessar logs de auditoria completos;
- Gerenciar configurações globais do sistema.

Importante: As permissões administrativas destinam-se exclusivamente à gestão e moderação do sistema, não substituindo o papel de Owner na administração cotidiana dos pets.

5.3.5 Matriz de permissões

O Quadro 2 apresenta uma visão consolidada das permissões por papel.

Quadro 2 – Matriz de permissões por papel de usuário

Ação	Owner	Editor	Viewer	Admin
Visualizar pet	Sim	Sim	Sim	Sim
Editar pet	Sim	-	-	Sim
Deletar pet	Sim	-	-	Sim
Criar/Editar refeição	Sim	Sim	-	Sim
Criar/Editar lembrete	Sim	-	-	Sim
Gerenciar compartilhamento	Sim	-	-	-
Acessar audit log	-	-	-	Sim
Gerenciar usuários	-	-	-	Sim

Fonte: Autoria própria (2025).

5.4 Histórias de Usuário prioritárias (Must Have)

As histórias de usuário apresentadas a seguir representam as funcionalidades **Must Have** do sistema UTFPets, isto é, aquelas consideradas essenciais para o funcionamento mínimo viável do produto. Essas histórias contemplam os quatro papéis do sistema — **Owner** (proprietário), **Editor** (cuidador com permissão de edição), **Viewer** (cuidador com permissão apenas de visualização) e **Administrador do Sistema** (gestão global) — abrangendo as operações fundamentais para o acompanhamento nutricional, o cuidado colaborativo e a gestão segura dos dados dos pets.

5.4.1 Histórias de Usuário - Owner (Proprietário)

- **HU001**: Como owner, quero cadastrar um perfil detalhado do meu pet — incluindo nome, idade, raça, porte e restrições alimentares — para personalizar o acompanhamento nutricional.
- **HU002**: Como owner, quero registrar e monitorar as refeições do meu pet, anotando horários e quantidades, para assegurar uma dieta equilibrada e regular.
- **HU003**: Como owner, quero receber lembretes automáticos sobre alimentação e medicação do meu pet, para manter uma rotina consistente de cuidados.
- **HU004**: Como owner, quero compartilhar as informações do meu pet com cuidadores autorizados (editors ou viewers), para facilitar o cuidado colaborativo quando eu não estiver presente.
- **HU005**: Como owner, quero gerenciar as permissões de acesso dos cuidadores, definindo quais dados podem ser visualizados ou editados, para garantir a segurança e o controle das informações.

5.4.2 Histórias de Usuário - Editor (cuidador com edição)

- **HU006**: Como editor, quero visualizar o perfil completo do pet, incluindo horários e detalhes de alimentação, para manter o cuidado padronizado conforme as orientações do owner.
- **HU007**: Como editor, quero registrar refeições e marcar tarefas como concluídas — como alimentar ou medicar o pet — para que o owner possa acompanhar as atividades realizadas.

5.4.3 Histórias de Usuário - Viewer (cuidador com visualização)

- **HU008**: Como viewer, quero visualizar o perfil do pet e o histórico de refeições, para acompanhar a rotina do animal sem poder realizar alterações.
- **HU009**: Como viewer, quero visualizar lembretes e notificações relacionados ao pet, para estar informado sobre os cuidados necessários.

5.4.4 Histórias de Usuário - Administrador do Sistema

- **HU010**: Como administrador, quero garantir a segurança e a privacidade dos dados dos usuários, aplicando autenticação e criptografia, para proteger as informações sensíveis do sistema.
- **HU011**: Como administrador, quero gerenciar o cadastro de usuários e pets, mantendo a integridade e a consistência da base de dados.
- **HU012**: Como administrador, quero acessar logs de auditoria do sistema, para rastrear ações importantes e garantir a conformidade com políticas de segurança.

Essas histórias de usuário descrevem as principais funcionalidades do sistema sob a perspectiva de cada perfil, orientando o planejamento e o desenvolvimento incremental do UTF-Pets.

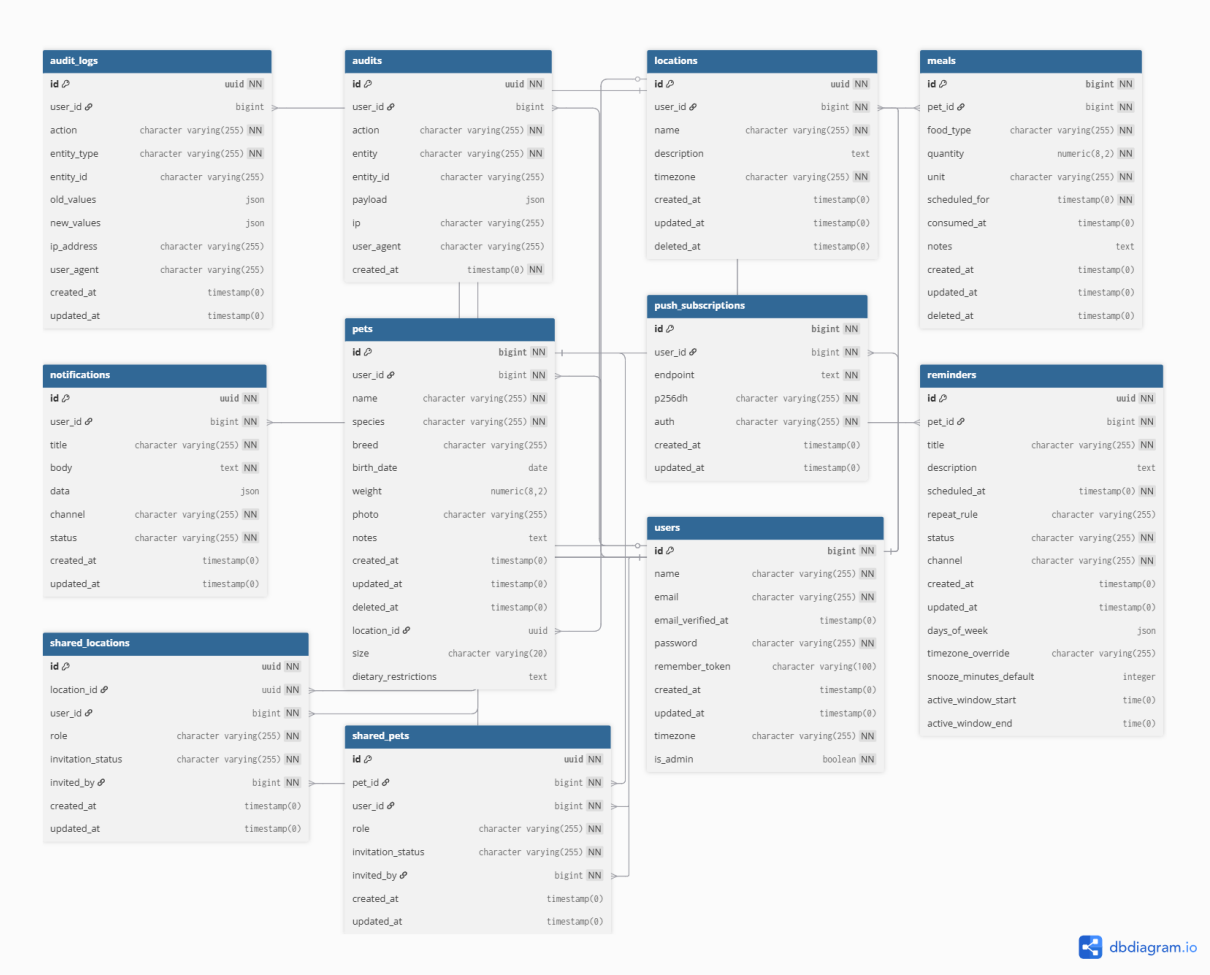
5.5 Modelagem do Banco de Dados

O banco de dados do sistema foi projetado utilizando o **PostgreSQL**, seguindo os princípios de normalização e integridade referencial. A modelagem buscou garantir desempenho, consistência e facilidade de manutenção, refletindo os principais relacionamentos entre usuários, pets, locais, lembretes e notificações.

5.5.1 Visão Geral do Modelo Relacional

A Figura 5 apresenta o diagrama entidade-relacionamento (ER) completo do UTFPets, contemplando as principais entidades do sistema e suas associações. O modelo é composto por 11 tabelas principais (users, locations, pets, meals, reminders, shared_pets, shared_locations, notifications, push_subscriptions, audit_logs e a tabela de migrations do Laravel). As 10 primeiras são tabelas de domínio da aplicação, enquanto migrations é uma tabela técnica do framework Laravel para versionamento do esquema do banco de dados.

Figura 5 – Diagrama entidade-relacionamento do banco de dados



Fonte: Autoria própria (2025).

O modelo contempla as seguintes 11 tabelas principais:

- **Users**: armazena as informações dos usuários e credenciais de acesso;
- **Locations**: representa os locais físicos onde os pets residem;
- **Pets**: contém os dados dos animais, como espécie, raça e peso;
- **Meals**: registra o histórico de refeições, horários e quantidades;
- **Reminders**: gerencia lembretes e notificações recorrentes;
- **SharedPets**: controla o compartilhamento de pets entre usuários;
- **SharedLocations**: controla o compartilhamento de locations entre usuários;
- **Notifications**: armazena mensagens e alertas enviados aos usuários;
- **PushSubscriptions**: vincula os dispositivos aos usuários para envio de notificações *push*;

- **AuditLogs**: mantém o registro detalhado de auditoria com valores antigos e novos de alterações;
- **Migrations**: tabela técnica do Laravel para controle de versionamento do esquema do banco de dados.

Os relacionamentos principais são:

- Um **usuário** pode gerenciar vários **pets** e **locations**;
- Cada **pet** pertence a um usuário e opcionalmente a uma location;
- Cada **pet** possui várias **refeições** e **lembretes**;
- As tabelas **SharedPets** e **SharedLocations** permitem colaboração entre usuários;
- Um **usuário** pode receber múltiplas **notificações** e possuir várias **assinaturas push**;
- Todas as entidades críticas possuem rastreabilidade via **AuditLogs**.

Essa estrutura garante a consistência das operações e a extensibilidade do sistema, permitindo a adição de novas funcionalidades sem impacto significativo na base de dados. O detalhamento completo das tabelas, incluindo atributos, chaves e restrições, encontra-se no **Apêndice A**.

5.5.2 Evolução da modelagem durante o desenvolvimento

Durante o processo de desenvolvimento, a modelagem do banco de dados passou por uma evolução arquitetural importante que ampliou significativamente as capacidades do sistema. Inicialmente, a modelagem conceitual previa que os pets fossem vinculados diretamente aos usuários através de uma relação simples entre as tabelas `users` e `pets`. No entanto, durante a implementação das funcionalidades essenciais, identificou-se a oportunidade de introduzir o conceito de **Locations** (locais físicos), criando uma hierarquia organizacional mais flexível e realista.

A introdução da tabela `locations` trouxe os seguintes benefícios:

- **Organização espacial**: Permite que usuários gerenciem pets distribuídos em diferentes locais físicos (residências, clínicas veterinárias, hotéis para pets);
- **Compartilhamento hierárquico**: Possibilita compartilhar uma location inteira (via `shared_locations`), propagando automaticamente o acesso a todos os pets vinculados àquele local;

- **Configuração de timezone**: Cada location possui seu próprio fuso horário (`timezone`), garantindo que lembretes sejam disparados nos horários corretos, independentemente da localização física dos pets;
- **Escalabilidade**: Facilita a expansão futura do sistema para cenários profissionais (clínicas, pet shops) onde múltiplos animais residem no mesmo local.

Esta mudança arquitetural demandou ajustes incrementais na estrutura do banco de dados:

- Adição da tabela `locations` com relacionamento $1 \rightarrow *$ com `users`;
- Inclusão do campo `location_id` (nullable) na tabela `pets`, permitindo que pets sejam opcionalmente associados a uma location;
- Criação da tabela `shared_locations` para suportar compartilhamento de locais completos;
- Ajustes nas policies de autorização para verificar permissões tanto por pet individual quanto por location.

Essa refatoração foi realizada de forma controlada através do sistema de migrations do Laravel, garantindo versionamento e rastreabilidade das alterações. O resultado final é uma modelagem mais robusta e adequada aos casos de uso reais identificados durante o desenvolvimento, mantendo a integridade dos dados e a performance do sistema.

5.6 Protótipos das telas

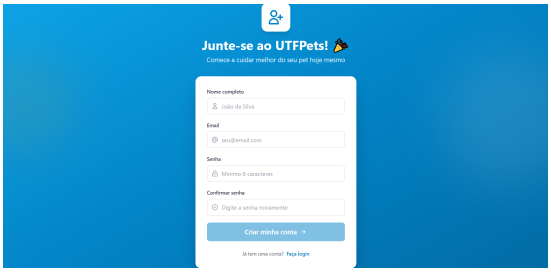
O design de interface do UTFPets foi concebido seguindo princípios de User Experience (User Experience (Experiência do Usuário) (UX)) e User Interface (UI), priorizando usabilidade, acessibilidade e consistência visual. Esta seção apresenta os protótipos implementados das principais telas do sistema.

5.6.1 Telas de autenticação

As telas de login e cadastro são os pontos de entrada dos usuários no UTFPets, desenvolvidas para serem simples, intuitivas e transmitirem confiança. A Figura 6 apresenta ambas as interfaces lado a lado.

Figura 6 – Telas de autenticação do UTFPets

Tela de login



Tela de cadastro

Fonte: Autoria própria (2025).

Elementos principais da tela de login:

- Logo do UTFPets centralizado;
- Campo de e-mail com validação de formato;
- Campo de senha com opção de visualizar/ocultar;
- Botão “Entrar” com destaque visual;
- Links para recuperação de senha e criação de conta.

5.6.2 Dashboard (Página Inicial)

Após o login, o usuário é direcionado ao Dashboard, mostrado na Figura 7, que apresenta uma visão geral dos pets e atividades recentes.

Figura 7 – Dashboard do UTFPets

Fonte: Autoria própria (2025).

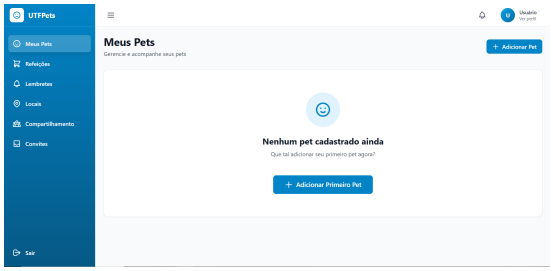
Componentes do Dashboard:

- Cards dos pets cadastrados com fotos;
- Lembretes do dia em destaque;
- Resumo de refeições recentes;
- Botão flutuante para ações rápidas;
- Menu de navegação inferior.

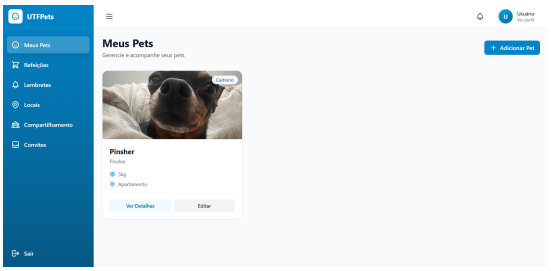
5.6.3 Gerenciamento de *pets*

A Figura 8 apresenta as interfaces de listagem e cadastro de pets, permitindo navegação rápida e registro completo de informações com upload de foto via Cloudinary.

Figura 8 – Sistema de gerenciamento de pets



Listagem de pets



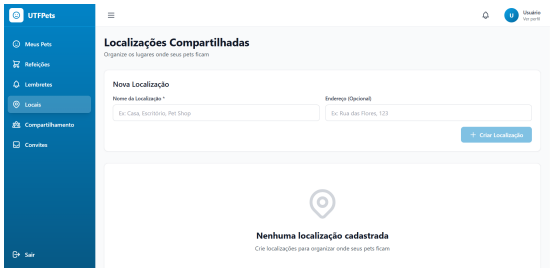
Cadastro de pet

Fonte: Autoria própria (2025).

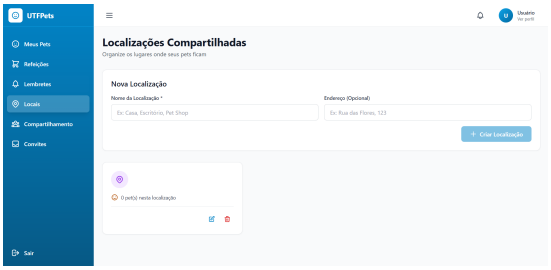
5.6.4 Gerenciamento de locais

O sistema de gerenciamento de locais onde os pets residem está ilustrado na Figura 9, que apresenta tanto a listagem quanto o cadastro de locations.

Figura 9 – Sistema de gerenciamento de locations



Listagem de locations



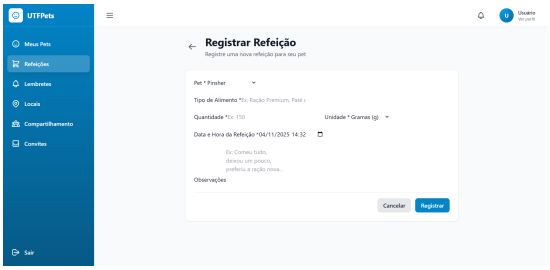
Cadastro de location

Fonte: Autoria própria (2025).

5.6.5 Gerenciamento de refeições

A Figura 10 apresenta a interface para registro e acompanhamento das refeições dos pets, com histórico e estatísticas.

Figura 10 – Página de gerenciamento de refeições

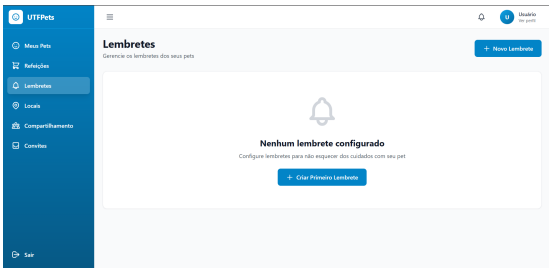


Fonte: Autoria própria (2025).

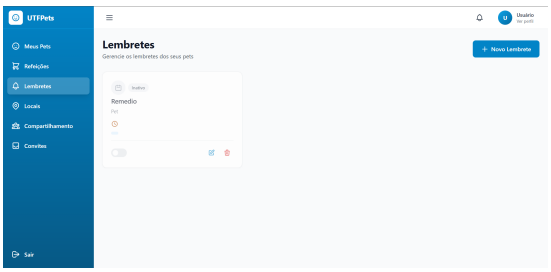
5.6.6 Sistema de lembretes

O sistema de lembretes com recorrência avançada e notificações automáticas está apresentado na Figura 11.

Figura 11 – Sistema de gerenciamento de lembretes



Listagem de lembretes



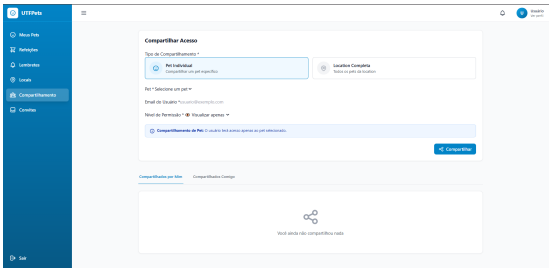
Formulário de criação

Fonte: Autoria própria (2025).

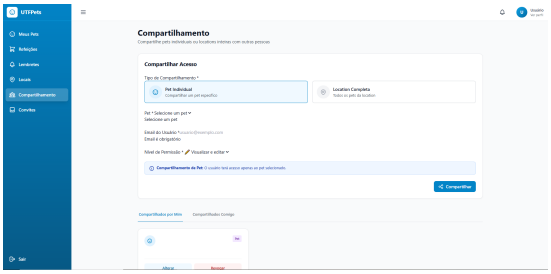
5.6.7 Sistema de compartilhamento

O sistema completo de compartilhamento colaborativo com controle de permissões está apresentado na Figura 12, que mostra a página de gerenciamento e o formulário de compartilhamento, além da Figura 13 que apresenta a gestão de convites pendentes.

Figura 12 – Sistema de gerenciamento de compartilhamentos



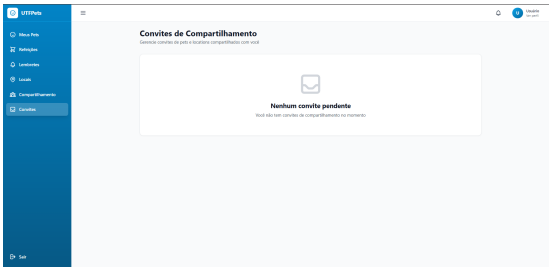
Página de gerenciamento



Formulário de compartilhamento

Fonte: Autoria própria (2025).

Figura 13 – Página de convites pendentes

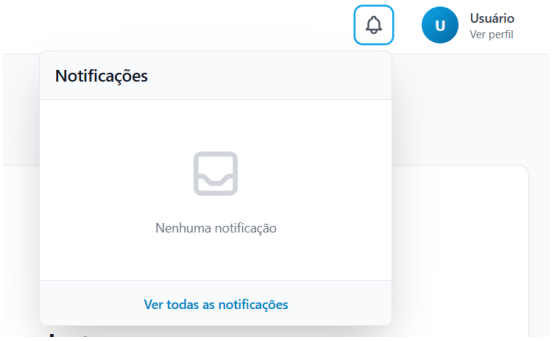


Fonte: Autoria própria (2025).

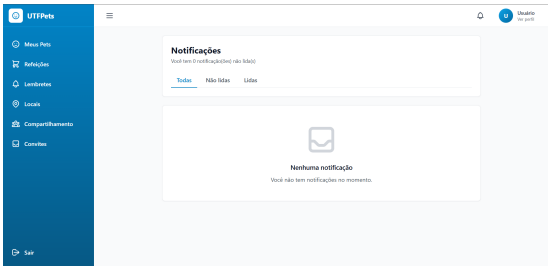
5.6.8 Sistema de notificações

A Figura 14 apresenta o sistema completo de notificações com histórico e controle de leitura, mostrando tanto o modal de notificações quanto a página completa.

Figura 14 – Sistema de notificações do UTFPets



Modal de notificações



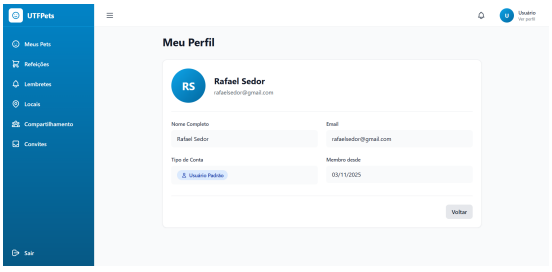
Página completa de notificações

Fonte: Autoria própria (2025).

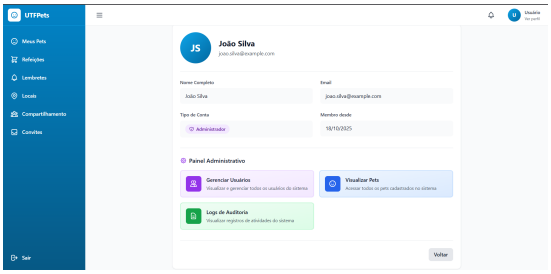
5.6.9 Painel administrativo

O painel administrativo completo para gestão do sistema está ilustrado nas figuras a seguir, apresentando as diferentes interfaces e permissões disponíveis para usuários comuns e administradores.

Figura 15 – Comparação entre perfis de usuário



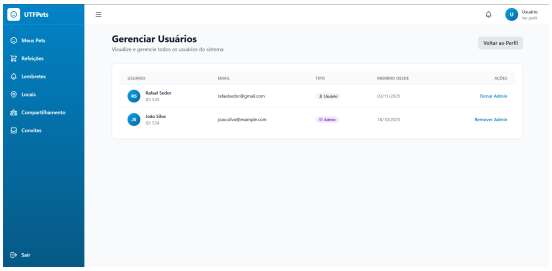
Perfil de usuário padrão



Perfil com permissões administrativas

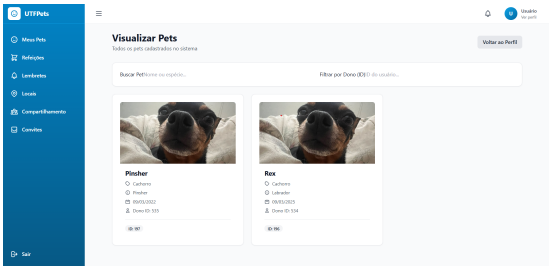
Fonte: Autoria própria (2025).

Figura 16 – Painel de gerenciamento de usuários (Admin)



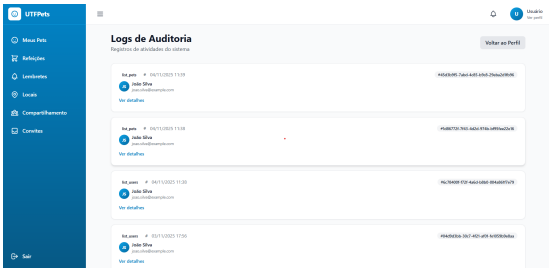
Fonte: Autoria própria (2025).

Figura 17 – Visualização de todos os pets (Admin)



Fonte: Autoria própria (2025).

Figura 18 – Painel de auditoria e logs do sistema (Admin)



Fonte: Autoria própria (2025).

5.6.10 Considerações sobre o design

O design das interfaces seguiu os seguintes princípios:

- **Consistência Visual**: Uso padronizado de cores, tipografia (Roboto) e componentes do Angular Material e TailwindCSS;
- **Responsividade**: Layouts adaptativos que funcionam em dispositivos móveis, tablets e desktops;
- **Acessibilidade**: Contraste adequado (Web Content Accessibility Guidelines (Diretrizes de Acessibilidade para Conteúdo Web) (WCAG) 2.1), tamanhos de toque adequados (44x44px mínimo), navegação por teclado;

- **Feedback Visual**: Animações suaves e estados de loading claros para melhor experiência do usuário;
- **Progressive Enhancement**: Funcionalidades que degradam graciosamente em navegadores mais antigos.

Todas as telas foram desenvolvidas como Single Page Application (SPA) utilizando Angular 17, proporcionando transições fluidas e experiência próxima a aplicativos nativos.

5.7 Considerações

A análise e o projeto do UTFPets abordaram o método **MoSCoW** para priorizar as funcionalidades, estabelecendo as características fundamentais (*Must Have*) que compõem a primeira versão do sistema. Também foi definida a arquitetura hierárquica (usuário, local, animal), o modelo de banco de dados em PostgreSQL e a concepção das telas principais por meio de *wireframes*.

6 DESENVOLVIMENTO DO SISTEMA

Este capítulo apresenta o processo de desenvolvimento do UTFPets, realizado entre maio e novembro de 2025, totalizando 100 commits distribuídos ao longo de sete meses. O desenvolvimento caracterizou-se por ciclos iterativos e incrementais, organizados em três fases principais: estruturação inicial, implementação das funcionalidades essenciais e transição para ambiente de produção.

Embora os conceitos de Scrum e sprints tenham sido apresentados como fundamentação metodológica nos capítulos anteriores, o desenvolvimento adotou uma abordagem adaptada ao contexto acadêmico individual, com iterações mais flexíveis e organizadas por conjuntos funcionais. A fase inicial (maio de 2025) concentrou-se na validação arquitetural, seguida por um período de suspensão das atividades devido a compromissos acadêmicos, e retomada em outubro de 2025 com foco no desenvolvimento das funcionalidades prioritárias. Por essa razão, a narrativa deste capítulo está organizada por fases funcionais, refletindo a dinâmica real de implementação do sistema.

6.1 Arquitetura geral do sistema

Antes de detalhar o desenvolvimento das funcionalidades, esta seção apresenta uma visão geral da arquitetura do UTFPets e a forma como os principais componentes se integram.

6.1.1 Camadas e organização do projeto

O UTFPets adota uma arquitetura em camadas baseada no padrão MVC, na qual o **Laravel 12.x** implementa a camada de controle e lógica de negócio, enquanto o **Angular 17** compõe a camada de apresentação. A comunicação entre as camadas é feita por meio de uma API RESTful que troca dados em formato JSON.

O projeto foi estruturado em um **monorepo**, reunindo em um único repositório Git o código do *backend*, do *frontend*, os arquivos de configuração do Nginx, scripts de deploy e testes End-to-End (Extremo a Extremo) (E2E). Essa organização facilita o versionamento atômico, os testes integrados e o deploy sincronizado de todas as partes do sistema.

6.1.2 Backend e persistência de dados

O *backend* é composto por 11 *controllers* e 10 *models* organizados em Laravel, responsáveis por implementar autenticação, cadastro de usuários, gerenciamento de pets, refeições, lembretes, notificações, compartilhamentos e operações administrativas. A persistência de dados é realizada em um banco **PostgreSQL**, cujo esquema é versionado por meio de 17 *migra-*

tions, que criam e mantêm tabelas como `users`, `pets`, `meals`, `reminders`, `locations`, `notifications`, `shared_pets`, `shared_locations` e `audit_logs`.

Camadas adicionais de segurança e organização incluem **middlewares** para autenticação e autorização, **policies** para controle de acesso a recursos específicos e **jobs** para processamento assíncrono de lembretes e envio de notificações.

6.1.3 Autenticação, notificações e mídia

A autenticação dos usuários é implementada com **JSON Web Tokens (JWT)**, utilizando o pacote `php-open-source-saver/jwt-auth`. Após o login, o token é armazenado no cliente e enviado no cabeçalho `Authorization` em cada requisição, sendo validado por middlewares no backend.

As notificações *push* são gerenciadas por meio do **Firebase Cloud Messaging (FCM)**, integrado ao PWA. Eventos relevantes (como lembretes próximos ou compartilhamentos aceitos) disparam notificações através de *events* e *listeners*, que enviam mensagens para os dispositivos inscritos.

As imagens dos pets são armazenadas e otimizadas pelo serviço **Cloudinary**, que realiza compressão e entrega via CDN, reduzindo o tempo de carregamento das telas e o consumo de banda.

6.1.4 Infraestrutura e conteinerização

Em ambiente de produção, o UTFPets é executado em contêineres **Docker** hospedados na **Google Cloud Platform (GCP)**. A aplicação é dividida, em produção, em cinco contêineres principais: backend Laravel, frontend Angular (build estático), Nginx (proxy reverso), Cloud SQL Proxy (conexão segura com o banco gerenciado) e Certbot (obtenção e renovação automática de certificados Secure Sockets Layer (Camada de Soquetes Seguros) (SSL)/TLS).

O acesso externo é realizado via Nginx, que recebe as requisições em HTTPS, encaminha-as ao backend e serve os arquivos estáticos do frontend. O banco de dados é fornecido pelo serviço gerenciado **Cloud SQL**, com backups automáticos e alta disponibilidade.

6.1.5 Fluxo de requisições

De forma geral, cada requisição enviada pela interface Angular percorre o seguinte caminho: é recebida pelo servidor Nginx em HTTPS, encaminhada ao backend Laravel, validada por *middlewares* de autenticação JWT e autorização, processada pelo *controller* correspondente e, por fim, tem seus dados persistidos ou consultados no PostgreSQL por meio do Eloquent ORM. O resultado é então retornado ao cliente em formato JSON.

Com essa visão geral da arquitetura consolidada, as seções seguintes descrevem como o sistema foi construído ao longo das *sprints*, bem como as principais decisões tomadas em cada etapa.

6.2 Fase inicial: estrutura base e prototipação

6.2.1 Contexto e planejamento

Ao iniciar o desenvolvimento em maio de 2025, a primeira fase concentrou-se em estabelecer os alicerces tecnológicos do projeto. O objetivo principal era validar as escolhas arquiteturais e criar um protótipo funcional que demonstrasse a viabilidade técnica da solução proposta, correspondendo à implementação parcial das histórias de usuário HU001 (cadastro de pets) e HU010 (autenticação e segurança).

6.2.2 Atividades realizadas

A fase inicial concentrou-se em três frentes principais de trabalho. Primeiramente, foi criado o repositório GitHub "TCC_UTFPets_API", estabelecendo a estrutura inicial tanto do projeto Laravel para backend quanto do projeto Angular para frontend, além da configuração básica do .gitignore para controle de versionamento adequado.

No backend, foi realizada a instalação do Laravel 12 com PHP 8.2, seguida pela definição das primeiras migrations (users e pets), estruturação inicial de Controllers e Models, e configuração das rotas API fundamentais. Paralelamente, o frontend foi inicializado com Angular 17, estabelecendo a estrutura básica de componentes, configuração de roteamento e implementação das primeiras telas (login e cadastro).

As funcionalidades implementadas nesta etapa incluíram o sistema básico de autenticação, o Create, Read, Update, Delete (Criar, Ler, Atualizar e Excluir) (CRUD) inicial de pets e as primeiras rotas de API, formando a base sobre a qual o restante do sistema seria construído.

6.2.3 Resultados da fase inicial

Ao final dessa primeira fase em maio de 2025, obteve-se um protótipo funcional com autenticação básica e listagem de pets, validando a arquitetura proposta. A estrutura base do projeto estava definida, incluindo a primeira versão do README documentado. Esse resultado forneceu confiança para avançar com a implementação das funcionalidades essenciais do sistema.

Após a conclusão dessa etapa inicial em maio, as atividades de desenvolvimento foram suspensas até outubro devido a compromissos acadêmicos. Em outubro, o projeto foi retomado

com foco na implementação das funcionalidades essenciais (*Must Have*) definidas pelo método MoSCoW.

6.3 Fase principal: implementação das funcionalidades essenciais

6.3.1 Contexto da retomada

Com a retomada do desenvolvimento em outubro de 2025, esta fase concentrou-se na implementação do núcleo funcional do sistema. O objetivo era desenvolver as funcionalidades categorizadas como *Must Have*, essenciais para viabilizar o MVP. Esse período registrou aproximadamente 40 commits e representou a consolidação das principais capacidades da aplicação, implementando as histórias de usuário HU002 a HU007.

6.3.2 Narrativa do desenvolvimento

O desenvolvimento neste período seguiu uma sequência lógica de prioridades. Inicialmente, consolidou-se o **sistema de autenticação com JWT** (HU010), implementando o pacote php-open-source-saver/jwt-auth versão 2.2. O AuthController foi estendido para suportar registro, login, logout e renovação de tokens, enquanto no *frontend* foram criados AuthGuard e AuthInterceptor para gerenciar o fluxo de autenticação. Esse trabalho garantiu que todas as requisições subsequentes fossem protegidas e vinculadas ao usuário autenticado, conforme ilustrado nas telas de login e cadastro apresentadas nas Figuras 6a e 6b.

Com a autenticação estabelecida, a próxima etapa concentrou-se no **sistema de gerenciamento de pets** (HU001), implementando o CRUD completo através do PetController. O Model Pet foi estruturado com relacionamentos complexos (User, Location, Meals, SharedPets, Reminders), permitindo a navegação entre entidades. A integração com Cloudinary possibilitou o upload e otimização automática de fotos, enquanto o *soft delete* garantiu que registros pudessem ser recuperados caso necessário. No *frontend*, foram desenvolvidos componentes modulares (pet-list, pet-form, pet-detail, pet-card) seguindo a arquitetura de *Standalone Components* do Angular 17, conforme mostrado na Figura 8a e 8b.

O endpoint `POST /api/pets` exemplifica o padrão de interação entre frontend e backend adotado no UTFPets. A Listagem 1 apresenta uma requisição típica para cadastro de pet.

Listagem 1 – Requisição HTTP para cadastro de pet

```
POST https://api.utfpets.online/api/pets
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGc...
Content-Type: application/json
```

```
{
  "name": "Rex",
  "species": "dog",
  "breed": "Labrador",
  "birth_date": "2020-05-15",
  "weight": 30.5,
  "gender": "male",
  "location_id": "550e8400-e29b-41d4-a716-446655440000"
}
```

Fonte: Autoria própria (2025).

Ao receber a requisição, o *middleware* de autenticação verifica o token JWT, e o *controller* responsável (`PetController@store`) realiza a validação dos dados, checa as permissões do usuário sobre a *location* informada e, em seguida, cria o registro no banco PostgreSQL. Caso uma foto seja enviada, o arquivo é enviado ao Cloudinary, que retorna uma Uniform Resource Locator (Localizador Uniforme de Recursos) (URL) pública para exibição. A Listagem 2 mostra a resposta retornada em caso de sucesso.

Listagem 2 – Resposta HTTP de sucesso ao cadastrar pet

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "name": "Rex",
  "species": "dog",
  "breed": "Labrador",
  "birth_date": "2020-05-15",
  "weight": 30.5,
  "gender": "male",
  "photo_url": "https://res.cloudinary.com/...",
  "location_id": "550e8400-e29b-41d4-a716-446655440000",
  "created_at": "2025-11-05T14:30:00.000000Z",
  "updated_at": "2025-11-05T14:30:00.000000Z"
}
```

Fonte: Autoria própria (2025)..

Esse exemplo ilustra o padrão RESTful adotado na API do UTFPets e a forma como as camadas de autenticação, validação, persistência e integração com serviços externos (Cloudinary) se articulam em uma única operação.

Na sequência, foi desenvolvido o **sistema de refeições** (HU002), criando o *MealController* com funcionalidades de agendamento (`scheduled_for`) e controle de consumação (`consumed_at`). O sistema permite registrar diferentes tipos de alimento com quantidades e unidades específicas, além de oferecer filtros por pet e período temporal. Essa funcionalidade permite aos owners monitorar a dieta de seus pets de forma detalhada, conforme ilustrado na Figura 10.

Em paralelo, implementou-se o **sistema de compartilhamento colaborativo** (HU004, HU005, HU008 e HU009), um dos diferenciais do UTFPets. O *SharedPetController* gerencia convites com três estados (pending, accepted, revoked) e três papéis distintos (owner, editor, viewer), cada um com permissões granulares definidas na Quadro 2. O sistema valida as permissões em cada operação através de Policies, garantindo que usuários somente executem ações autorizadas. No *frontend*, o share-dialog facilita o envio de convites, enquanto o share-list exibe os compartilhamentos ativos, conforme mostrado nas Figura 12a, Figura 12b e Figura 13.

Paralelamente ao desenvolvimento das funcionalidades, foi estabelecida uma **cultura de testes automatizados**. Ao final desse ciclo, o sistema contava com mais de 88 testes implementados em PHPUnit, cobrindo cenários de autenticação, gerenciamento de pets, refeições e compartilhamento. Os *feature tests* simulam fluxos completos de usuário, enquanto as *factories* geram dados de teste consistentes. A estratégia *RefreshDatabase* garante isolamento entre testes, eliminando efeitos colaterais.

6.3.3 Entregas e aprendizados

Ao final dessa fase, a parte principal do sistema estava totalmente funcional, com upload de imagens otimizadas, compartilhamento colaborativo operacional e cobertura robusta de testes. A documentação da API foi consolidada utilizando Swagger/OpenAPI, facilitando o entendimento dos *endpoints* e acelerando a integração *frontend-backend*.

Um aprendizado importante desse período foi a necessidade de refatorações incrementais. Inicialmente, os pets eram vinculados diretamente aos usuários, mas identificou-se a oportunidade de introduzir o conceito de **Locations**, criando uma hierarquia mais flexível. Essa mudança arquitetural, embora tenha demandado ajustes no código existente, ampliou significativamente as possibilidades de uso do sistema.

6.4 Fase avançada: funcionalidades diferenciadas e PWA

6.4.1 Contexto e motivação

Com o núcleo funcional consolidado, esta fase voltou-se para funcionalidades avançadas que diferenciariam o UTFPets de soluções similares. Este período registrou aproximadamente 36 commits e teve como principais entregas: a formalização do conceito de *Locations*, sistema de lembretes com recorrência complexa (HU003), notificações multicanal, transformação em PWA e painel administrativo (HU011 e HU012).

6.4.2 Narrativa do desenvolvimento

Nesta fase, formalizou-se o **sistema de Locations**, transformando o conceito inicial em uma funcionalidade completa. O *LocationController* foi implementado com CRUD completo, enquanto o *Model Location* estabeleceu relacionamentos com Users e Pets. Uma inovação importante foi o SharedLocationController, que permite compartilhar locations inteiras, propagando automaticamente o acesso a todos os pets vinculados. Cada *location* possui seu próprio *timezone*, garantindo que lembretes sejam disparados nos horários corretos independentemente da localização física. No *frontend*, os componentes *location-list*, *location-form* e *location-detail* oferecem interface intuitiva para gerenciamento espacial, conforme ilustrado na Figura 9a e Figura 9b.

Em seguida, desenvolveu-se o **sistema de lembretes com recorrência avançada** (HU003), uma das funcionalidades mais complexas do UTFPets. O ReminderController expõe 7 *endpoints* que suportam regras de repetição (none, daily, weekly, custom) através do enum RepeatRule. O sistema utiliza *jobs* assíncronos com idempotência, executados por um *scheduler* Laravel que verifica lembretes pendentes a cada minuto. Customizações avançadas incluem dias da semana específicos (`days_of_week`), janelas de horário ativas (`active_window`), substituição de *timezone* (`timezone_override`) e tempo de adiamento (snooze). As ações de snooze e complete permitem aos usuários gerenciar lembretes de forma flexível, conforme mostrado na Figura 11a e Figura 11b.

O **sistema de notificações multicanal** foi implementado em paralelo, criando o NotificationController com suporte a três canais (db, email, push) através do enum NotificationChannel. O sistema gerencia estados de notificação (queued, sent, failed, read) e oferece histórico com paginação, contador de mensagens não lidas e marcação individual ou em lote, conforme apresentado na Figura 14a e Figura 14b. A arquitetura baseada em Events e Listeners permite que notificações sejam disparadas automaticamente em resposta a eventos do sistema, promovendo desacoplamento e manutenibilidade.

A transformação do UTFPets em um **PWA** completo foi realizada na sequência. Utilizando o pacote @angular/service-worker do Angular, foi configurado um Service Worker que intercepta requisições de rede e implementa cache inteligente de assets e respostas de API. O manifest.webmanifest define metadados que permitem instalação em dispositivos móveis e desktop. A integração com Firebase Cloud Messaging habilita notificações push mesmo com o aplicativo fechado, aumentando significativamente o engajamento dos usuários. O modo offline garante que funcionalidades básicas permaneçam acessíveis sem conexão à internet.

Complementando as funcionalidades essenciais, implementou-se o **painel administrativo** (HU011 e HU012), criando o AdminController protegido por middleware específico. O painel permite listar e gerenciar usuários, alterar permissões (`is_admin` flag), visualizar todos os pets cadastrados no sistema e acessar logs de auditoria. No *frontend*, os componentes admin-users e admin-dashboard oferecem interface dedicada para administradores, conforme ilustrado na Figura 15a, Figura 15b, Figura 16, Figura 17 e Figura 18.

Complementando as funcionalidades administrativas, criou-se a infraestrutura de **auditoria**, com a tabela `audit_logs` registrando ações críticas (created, updated, deleted, viewed). O sistema armazena alterações em formato JSON, incluindo valores antigos e novos, além de capturar IP address e user agent. Embora a trait Auditable tenha sido estruturada para futura implementação automática, a base para rastreabilidade completa estava estabelecida.

6.4.3 Entregas e reflexões

Ao final dessa fase, o UTFPets havia evoluído significativamente. As funcionalidades avançadas estavam totalmente operacionais: PWA com suporte offline, lembretes com recorrência complexa, notificações multicanal e painel administrativo. A documentação técnica foi expandida para 17 arquivos, cobrindo arquitetura, API e procedimentos de deploy.

Um desafio importante desse período foi o balanceamento entre complexidade técnica e usabilidade. O sistema de lembretes, por exemplo, oferece configurações avançadas (timezone override, active windows) sem comprometer a simplicidade para usuários básicos que desejam apenas agendar lembretes simples. Essa abordagem de "potência progressiva"tornou-se um princípio de design do projeto.

6.5 Fase final: deploy, produção e refinamentos

6.5.1 Contexto da transição para produção

A fase final do desenvolvimento (final de outubro e novembro de 2025) marcou a transição do UTFPets do ambiente de desenvolvimento para produção. Este período registrou 36 commits, concentrados em infraestrutura, automação de deploy, otimizações de performance e

refinamentos de interface. O objetivo era disponibilizar o sistema de forma confiável, segura e com experiência de usuário polida.

6.5.2 Narrativa do desenvolvimento

Nesta fase final, o foco principal foi a **infraestrutura de produção no Google Cloud Platform**. Foi provisionada uma Virtual Machine (Máquina Virtual) (VM) e2-small no Compute Engine (Debian 12, região southamerica-east1-b), escolhida por sua proximidade geográfica e custo-benefício adequado para o projeto. O Cloud SQL PostgreSQL foi configurado como serviço gerenciado, eliminando preocupações com backups, atualizações e alta disponibilidade. O Cloud SQL Proxy foi implementado para conexão segura entre a aplicação e o banco de dados, utilizando Service Accounts e políticas Identity and Access Management (Gerenciamento de Identidade e Acesso) (IAM) restritivas. Docker e Docker Compose foram instalados na VM, permitindo orquestração de 5 containers: app (backend Laravel), cloud-sql-proxy, nginx (proxy reverso), certbot (certificados SSL) e frontend (build Angular).

Em paralelo, implementou-se o **pipeline CI/CD com GitHub Actions**, criando o workflow deploy-vm.yml com *trigger* automático em push para a branch main. O pipeline executa: backup automático do banco de dados, deploy via gcloud scp, build de containers Docker, execução de migrations e otimizações Laravel, e health check automático. Um script PowerShell foi desenvolvido para sincronizar *secrets* entre o ambiente local e o GitHub, facilitando a manutenção de variáveis de ambiente sensíveis.

A configuração de **HTTPS com Let's Encrypt** foi concluída utilizando Certbot containerizado para obtenção e renovação automática de certificados SSL/TLS. O Nginx foi configurado para servir tráfego HTTPS e redirecionar automaticamente requisições HTTP. Três domínios foram estabelecidos: https://utfpets.online (frontend), https://api.utfpets.online (backend) e https://api.utfpets.online/swagger (documentação interativa).

Foram realizadas **otimizações de performance** fundamentais. No backend, foram aplicados cache de configuração Laravel e otimização de queries com índices estratégicos. No frontend, a compilação foi configurada em modo produção (Angular build –prod) com lazy loading de módulos, reduzindo significativamente o tempo de carregamento inicial. O Nginx foi ajustado para compressão gzip, diminuindo o tamanho dos assets transferidos.

Paralelamente, a interface passou por um **redesign completo com TailwindCSS**, adotando componentes modernos e responsivos. O Pet Form foi redesenhado com validações visuais claras, enquanto o Pet List recebeu melhorias de debugging e usabilidade. Um design system consistente foi estabelecido, garantindo uniformidade visual em toda a aplicação.

A reestruturação do projeto em **arquitetura Monorepo** consolidou backend, frontend, testes E2E, configurações nginx e scripts de automação em uma estrutura organizada (backend/, frontend/, tests/e2e/, nginx/, scripts/). O README foi completamente reescrito, incluindo

Listagem 3 – Estrutura do repositório monorepo do UTFPets

```
TCC_UTFPets_API/
 backend/
    app/
       Http/
          Controllers/
          Middleware/
       Models/
       Enums/
    database/
       migrations/
    tests/                  (88+ testes PHPUnit)
 frontend/
    src/
       app/
          auth/
          pets/
          meals/
          reminders/
          ...             (demais features)
       assets/
    angular.json
 nginx/
    nginx.conf
 tests/
    e2e/                    (Testes Playwright)
 docker-compose.yml
```

Fonte: Autoria própria (2025)..

instruções de instalação, desenvolvimento, deploy e arquitetura do sistema. A Listagem 3 apresenta um resumo da organização final.

Os **testes E2E com Playwright** foram configurados nessa fase final, criando a estrutura /tests/e2e/ com cenários que simulam fluxos completos de usuário. Esses testes foram integrados ao pipeline CI/CD, garantindo que cada deploy somente ocorra após validação automática dos fluxos críticos.

A fase final demandou **32 commits de ajustes incrementais**, incluindo correções de configuração nginx, refinamentos no health check, ajustes no setup de SSL, melhorias no service worker do PWA, correções de timeout e permissões, e otimizações no pipeline CI/CD. Esse processo iterativo de refinamento demonstrou a importância de testes em ambiente de produção real e a necessidade de monitoramento contínuo.

6.5.3 Entregas e lições do deploy

Ao final dessa fase de deploy, o UTFPets estava em produção em https://utfpets.online, com API documentada em https://api.utfpets.online/swagger. O HTTPS estava configurado com certificados válidos, o deploy automático funcionava perfeitamente via GitHub Actions, a performance estava otimizada (tempos de resposta < 200ms) e a interface apresentava design moderno e responsivo.

O processo de deploy revelou lições valiosas. A configuração inicial do CI/CD demandou múltiplas iterações (32 commits de ajustes), evidenciando que automação de infraestrutura é complexa mas fundamental. A escolha por Monorepo simplificou o versionamento atômico, permitindo que mudanças no backend e frontend fossem deployadas sincronizadamente. O uso de containers Docker garantiu paridade entre ambientes de desenvolvimento e produção, eliminando o clássico problema "funciona na minha máquina".

6.6 Considerações Finais do Desenvolvimento

O desenvolvimento do UTFPets foi concluído com sucesso em **7 meses** (Maio-Novembro 2025), resultando em uma aplicação web completa e robusta com as seguintes métricas:

6.6.1 Estatísticas finais

O desenvolvimento ocorreu no período de maio a novembro de 2025, totalizando 100 commits distribuídos ao longo dos sete meses. A distribuição temporal dos commits reflete a dinâmica de trabalho adotada: 4 commits em maio de 2025 concentraram-se na inicialização do projeto, 76 commits em outubro de 2025 representaram o desenvolvimento principal das funcionalidades essenciais, e 32 commits em novembro de 2025 foram dedicados ao deploy e ajustes finais em ambiente de produção.

6.6.2 Métricas quantitativas do sistema

A Tabela 1 apresenta um resumo quantitativo dos principais componentes desenvolvidos no UTFPets, evidenciando a complexidade e abrangência do sistema.

Tabela 1 – Métricas quantitativas do sistema UTFPets

Componente	Quantidade	Descrição
Backend		
Controllers	11	AuthController, PetController, MealController, ReminderController, NotificationController, SharedPetController, SharedLocationController, LocationController, AdminController, UserController, PushSubscriptionController
Models	10	User, Pet, Meal, Reminder, Notification, SharedPet, SharedLocation, Location, PushSubscription, AuditLog
Migrations	17	Estrutura completa do banco de dados com suporte a UUID
Enums	7	RepeatRule, ReminderStatus, SharedPetRole, InvitationStatus, NotificationChannel, NotificationStatus
Testes Automatizados	88+	Testes unitários e de integração com PHPUnit
Banco de Dados		
Tabelas	11	users, pets, meals, reminders, notifications, shared_pets, shared_locations, locations, push_subscriptions, audit_logs, audit
API RESTful		
Endpoints	45+	Distribuídos entre autenticação, pets, refeições, lembretes, notificações, compartilhamento e administração
Frontend		
Features	8	auth, pets, meals, reminders, locations, sharing, notifications, admin
Componentes Angular	40+	Componentes standalone organizados por feature
Infraestrutura		
Containers Docker	5	app, cloud-sql-proxy, nginx, certbot, frontend
Documentação		
Arquivos Técnicos	17	Documentação detalhada de arquitetura, API e deployment
Swagger/OpenAPI	1	Documentação interativa completa da API

Essas métricas demonstram a robustez e complexidade do sistema desenvolvido, que implementa funcionalidades avançadas mantendo código limpo, testável e bem documentado.

6.6.3 Arquitetura e código

A arquitetura final do sistema compreende, no backend, 11 Controllers, 10 Models, 17 Migrations e 7 Enums que estruturam toda a lógica de negócio. O frontend foi organizado em 8 features principais com múltiplos componentes reutilizáveis. A qualidade do código é assegurada por mais de 88 testes automatizados implementados em PHPUnit. A documentação técnica conta com 17 arquivos detalhados que cobrem arquitetura, deployment e boas práticas. A API RESTful completa está documentada com Swagger, facilitando a integração e manutenção do sistema.

6.6.4 Tecnologias utilizadas

O stack tecnológico adotado combina ferramentas modernas e consolidadas no mercado. No backend, utilizou-se Laravel 12 com PHP 8.2 e PostgreSQL como sistema gerenciador de banco de dados. O frontend foi desenvolvido em Angular 17, estilizado com TailwindCSS e Angular Material para componentes visuais. A infraestrutura de produção baseia-se em Docker para containerização, Google Cloud Platform (Plataforma de Nuvem do Google) (GCP) para hospedagem, Nginx como servidor web e proxy reverso, e Let's Encrypt para certificados SSL/TLS. Serviços complementares incluem Cloudinary para otimização de imagens, Firebase FCM para notificações push e GitHub Actions para automação de CI/CD.

6.6.5 Funcionalidades implementadas

Todas as 12 histórias de usuário apresentadas (HU001 a HU012) foram completamente implementadas na versão atual do UTFPets. Não houve alterações, remoções ou adição de novas histórias durante o processo de desenvolvimento, garantindo que todos os requisitos essenciais definidos na fase de análise foram atendidos. O sistema implementado contempla integralmente:

- **HU001 a HU005 (Owner)**: Gerenciamento completo de pets, refeições, lembretes e compartilhamento com controle de permissões;
- **HU006 e HU007 (Editor)**: Visualização e registro de atividades com permissões de edição controladas;
- **HU008 e HU009 (Viewer)**: Acesso somente leitura para acompanhamento sem interferência;
- **HU010 a HU012 (Administrador)**: Segurança, gestão de usuários e auditoria completa do sistema.

O sistema implementa um conjunto completo de funcionalidades que atendem aos objetivos propostos. A autenticação JWT garante segurança no acesso aos recursos. O CRUD de pets inclui upload otimizado de imagens via Cloudinary. O sistema de refeições mantém histórico detalhado de alimentação com quantidades e horários. O compartilhamento colaborativo suporta tanto pets individuais quanto locations inteiras, com três níveis de permissões (owner, editor, viewer). Os lembretes oferecem recorrência avançada com suporte a diferentes timezones. O sistema de notificações opera em três canais (banco de dados, email e push). A aplicação funciona como PWA completo com suporte a modo offline. O painel administrativo permite gestão de usuários e auditoria detalhada do sistema. Por fim, o deploy automatizado com CI/CD garante entregas rápidas e confiáveis.

6.6.6 Cobertura dos objetivos específicos

As histórias de usuário implementadas cobrem integralmente os cinco objetivos específicos estabelecidos no Capítulo 1:

- **Objetivo 1 - Sistema de cadastro**: Coberto por HU001 (cadastro de perfil detalhado dos pets);
- **Objetivo 2 - Lembretes e notificações**: Coberto por HU003 (lembretes automáticos de alimentação e medicação);
- **Objetivo 3 - Controle alimentar**: Coberto por HU002 (registro e monitoramento de refeições com controle de porções);
- **Objetivo 4 - Compartilhamento com permissões**: Coberto por HU004, HU005, HU006, HU007, HU008 e HU009 (sistema completo de compartilhamento com três níveis de permissões);
- **Objetivo 5 - Painel administrativo**: Coberto por HU010, HU011 e HU012 (segurança, gestão de usuários e auditoria).

Esta correspondência direta entre objetivos e histórias garante que o sistema desenvolvido atende plenamente aos propósitos estabelecidos para este trabalho.

6.6.7 Desafios superados

1. Arquitetura Monorepo: Decisão de manter frontend e backend em um único repositório simplificou versionamento atômico e testes integrados.

2. Deploy Automatizado: Configuração completa de CI/CD com GitHub Actions, incluindo 32 commits de ajustes até alcançar estabilidade total.

3. PWA e Offline Mode: Implementação de Service Workers e cache inteligente permitindo funcionamento offline.

4. Sistema de Lembretes Complexo: Jobs assíncronos com idempotência e suporte a recorrência avançada (dias da semana, janelas de horário, timezone override).

5. Compartilhamento Flexível: Sistema dual de compartilhamento (pets individuais + locations inteiras) com controle granular de permissões.

6.6.8 Lições aprendidas

O processo de desenvolvimento proporcionou aprendizados valiosos sobre engenharia de software moderna. A adoção de metodologia ágil com entregas incrementais permitiu validação contínua das funcionalidades desenvolvidas. Os testes automatizados provaram ser essenciais para realizar refatorações com segurança e confiança. A manutenção de documentação técnica detalhada facilitou significativamente a manutenção e evolução do código ao longo do projeto. Uma arquitetura bem planejada desde o início mostrou-se fundamental para facilitar a adição de features avançadas sem necessidade de reescritas estruturais. Por fim, a implementação de CI/CD demonstrou reduzir significativamente tanto o tempo de deploy quanto erros humanos durante o processo de publicação.

O projeto demonstrou com sucesso a aplicação de conceitos modernos de engenharia de software, resultando em uma solução robusta, escalável e pronta para uso em produção.

7 CONCLUSÃO

O desenvolvimento do UTFPets representou uma jornada de aprendizado técnico, metodológico e pessoal, resultando em uma aplicação web funcional voltada ao apoio de tutores no controle de alimentação e cuidados de seus animais de estimação. A solução integra, em uma única plataforma, recursos de cadastro de pets, registro de refeições, lembretes inteligentes e colaboração entre múltiplos cuidadores.

Conforme detalhado no Capítulo 6, todas as histórias de usuário definidas na fase de análise foram implementadas, cobrindo integralmente os objetivos específicos apresentados na Introdução. O sistema oferece:

- gerenciamento completo de perfis de pets, incluindo informações nutricionais relevantes;
- registro estruturado de refeições, permitindo acompanhar horários e quantidades;
- lembretes configuráveis para alimentação, medicação e outras rotinas;
- compartilhamento com controle granular de permissões entre tutores e cuidadores;
- um painel administrativo para gestão global de usuários, pets e auditoria.

Dessa forma, o UTFPets atende ao propósito central do trabalho: disponibilizar uma ferramenta capaz de organizar e tornar mais seguro o cuidado diário com os animais.

Entre os principais desafios enfrentados destacam-se a modelagem de um sistema de lembretes recorrentes com múltiplos fusos horários, a implementação do modo offline via PWA, o desenho do modelo de compartilhamento com diferentes papéis de usuário e a configuração do pipeline de deploy automatizado em nuvem. A superação desses pontos exigiu estudo aprofundado, experimentação e refatorações sucessivas, consolidando o amadurecimento do autor em engenharia de software.

Como contribuições, o UTFPets oferece uma solução integrada que reúne funcionalidades até então dispersas em aplicativos distintos, com foco em colaboração entre cuidadores e controle detalhado da rotina alimentar dos pets. Para além do produto em si, o projeto demonstra, de forma prática, a aplicação de conceitos modernos de desenvolvimento web, testes, segurança, conteinerização e operação em nuvem.

O UTFPets encontra-se operacional em ambiente de produção e pronto para ser utilizado por tutores e cuidadores, cumprindo os objetivos propostos e abrindo espaço para novas evoluções. A experiência de desenvolvimento consolidou conhecimentos adquiridos ao longo do curso e reforçou a importância de boas práticas de projeto, testes e automação.

Com a base técnica estabelecida e os requisitos iniciais atendidos, o sistema está preparado para incorporar melhorias mais avançadas, como recursos de inteligência artificial, in-

tegrações com dispositivos IoT e serviços de telemedicina veterinária, que são discutidos na seção de *Trabalhos futuros*.

7.1 Trabalhos futuros

O UTFPets foi desenvolvido como um MVP, focado nas funcionalidades *Must Have* definidas pelo método MoSCoW. As histórias de usuário prioritárias foram implementadas e atendem aos objetivos específicos deste trabalho, mas há um conjunto de funcionalidades classificadas como *Should Have*, *Could Have* e *Won't Have* que compõem um roadmap consistente para a evolução da plataforma.

No grupo das **funcionalidades importantes (*Should Have*)**, destacam-se:

- **Conteúdo educativo:** criação de uma área com materiais sobre nutrição e cuidados gerais, curados por especialistas, para apoiar decisões dos tutores.
- **Personalização avançada de lembretes:** embora o sistema já ofereça diferentes regras de recorrência e fusos horários, futuros aprimoramentos poderão incluir presets por tipo de atividade, perfis de lembrete por pet e preferências globais de notificação.
- **Gráficos de evolução do peso:** utilização dos dados já armazenados no banco para gerar visualizações que auxiliem na avaliação do estado corporal e na prevenção de obesidade ou desnutrição.

Entre as **funcionalidades desejáveis (*Could Have*)**, podem ser citadas:

- **Histórico analítico:** expansão das telas de refeições e atividades com filtros avançados, relatórios por período e exportação de dados.
- **Integração com calendários externos:** sincronização de lembretes com o calendário do dispositivo ou serviços como Google Calendar, centralizando compromissos do pet na agenda do usuário.
- **Sugestões automáticas de dieta:** geração de recomendações de porções e tipos de alimento considerando espécie, idade, porte e histórico registrado.

As **funcionalidades de longo prazo (*Won't Have* na versão atual)** envolvem mudanças mais profundas e uso intensivo de tecnologias emergentes. Entre elas, destacam-se:

- módulos de dietas personalizadas elaboradas por veterinários diretamente no aplicativo;
- integração com serviços de telemedicina veterinária e prontuário eletrônico;
- sistemas de recomendação de produtos com apoio de inteligência artificial;

- integração com dispositivos IoT (comedouros inteligentes, coleiras com GPS, balanças conectadas) para registro automático de dados.

Além das funcionalidades orientadas ao usuário, há espaço para **melhorias técnicas** relevantes, como a migração gradual para uma arquitetura de microserviços, adoção de cache distribuído e CDN global, fortalecimento de mecanismos de segurança (autenticação em duas etapas, auditorias periódicas) e internacionalização da plataforma com suporte a múltiplos idiomas, moedas e regulamentações (Lei Geral de Proteção de Dados (LGPD), General Data Protection Regulation (Regulamento Geral de Proteção de Dados) (GDPR), entre outras).

Como **próximos passos imediatos**, recomenda-se a realização de testes beta com tutores reais, a coleta sistemática de feedback e métricas de uso, e a priorização das funcionalidades futuras com base nas necessidades observadas. A partir dessa validação, o UTFPets poderá evoluir de um MVP acadêmico para uma solução de mercado robusta, ampliando seu impacto na saúde e bem-estar dos animais de estimação.

REFERÊNCIAS

ANDERSON, D. J. **Kanban: Successful evolutionary change for your technology business**. Seattle, WA: Blue Hole Press, 2010.

Association for Pet Obesity Prevention. **2024 Pet Obesity Prevalence Survey**. 2024. https://www.petobesityprevention.org/2024-survey. Acessado em: 15 jan. 2025.

CHANDLER, M. Nutrition for weight management in cats and dogs. **UK Vet: Companion Animal**, v. 27, n. 5, p. 234–242, 2022.

DRIESSEN, V. **A successful Git branching model**. 2010. https://nvie.com/posts/a-successful-git-branching-model/. Accessed: 2023-08-01.

FASCETTI, A.; DELANEY, S. **Applied Veterinary Clinical Nutrition**. [*S.l.*]: Wiley-Blackwell, 2012.

FEWSTER, M.; GRAHAM, D. **Experiences of Test Automation: Case Studies of Software Test Automation**. Boston: Addison-Wesley, 2012. ISBN 978-0321754068.

GAROUSI, V.; FELDERER, M.; MäNTYLä, M. V. The need for multivocal literature reviews in software engineering: Complementing systematic literature reviews with grey literature. **Information and Software Technology**, v. 94, p. 62–81, 2018. ISSN 0950-5849.

GERMAN, A. J. Updates on obesity management for dogs and cats. **Veterinary Clinics of North America: Small Animal Practice**, v. 52, n. 5, p. 1053–1070, 2022.

Instituto Pet Brasil. **Censo Pet: 149,6 milhões de animais de estimação no Brasil**. 2022. http://institutopetbrasil.com/. Acessado em: 15 jan. 2025.

JONES, M.; BRADLEY, J.; SAKIMURA, N. **JSON Web Token (JWT)**. [*S.l.*], 2015. Internet Engineering Task Force.

KOGAN, L. R.; HELLYER, P. W. Access to veterinary care in the us: A preliminary study of pet owner experiences. **Animals**, v. 13, n. 2, p. 340, 2023.

LAFLAMME, D. P.; FLAMMER, S. A.; HANSEN, B. D. Obesity in dogs and cats: A metabolic and endocrine disorder. **Topics in Companion Animal Medicine**, v. 23, n. 3, p. 126–131, 2008.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. **Linux Journal**, v. 2014, n. 239, p. 2, 2014.

MESZAROS, G. **xUnit Test Patterns: Refactoring Test Code**. Boston: Addison-Wesley Professional, 2007. ISBN 978-0131495050.

RUSSELL, A.; FIRTMAN, M. **Progressive Web Apps**. Sebastopol, CA: O'Reilly Media, 2018.

SCHWABER, K.; SUTHERLAND, J. **The Scrum Guide: The Definitive Guide to Scrum**. 2017. https://scrumguides.org/scrum-guide.html. Accessed: 2023-08-01.

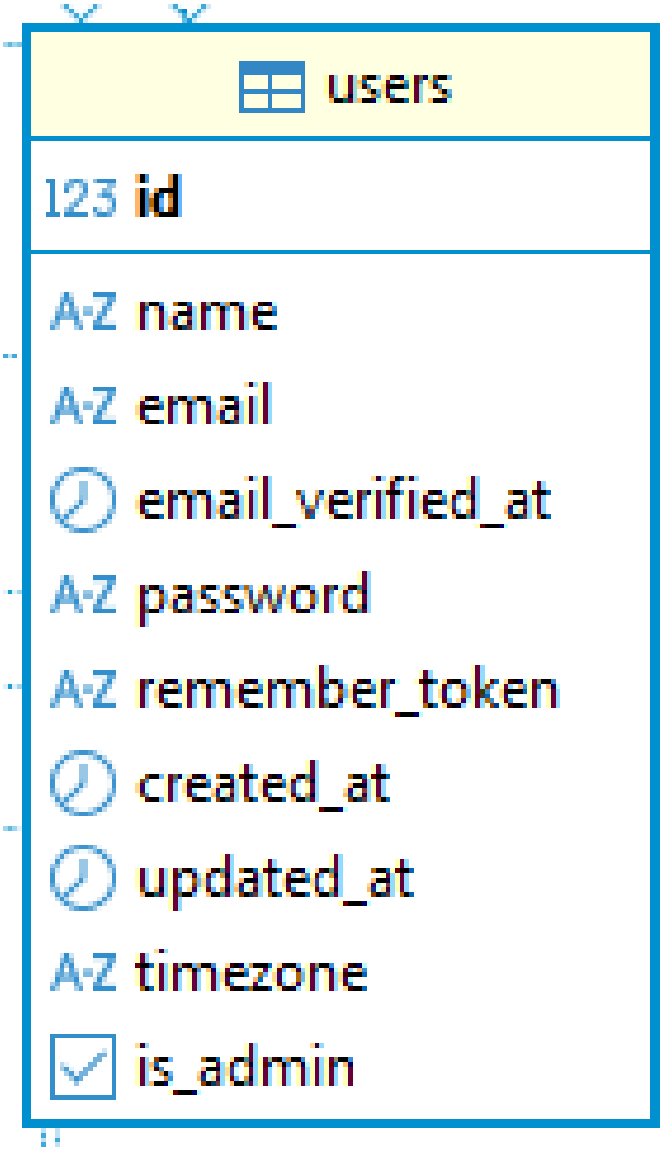
APÊNDICE A – Estruturas das Tabelas do Banco de Dados

Este apêndice apresenta o detalhamento completo do esquema relacional do banco de dados do UTFPets, desenvolvido em **PostgreSQL**. As tabelas foram implementadas por meio de *migrations* no framework Laravel, garantindo versionamento, integridade referencial e reprodutibilidade do ambiente de dados.

A.1 Tabela Users (Usuários)

A tabela **Users** armazena as informações dos usuários e credenciais de acesso ao sistema.

Figura 19 – Estrutura da tabela *Users*



Fonte: Autoria própria (2025).

Campos principais:

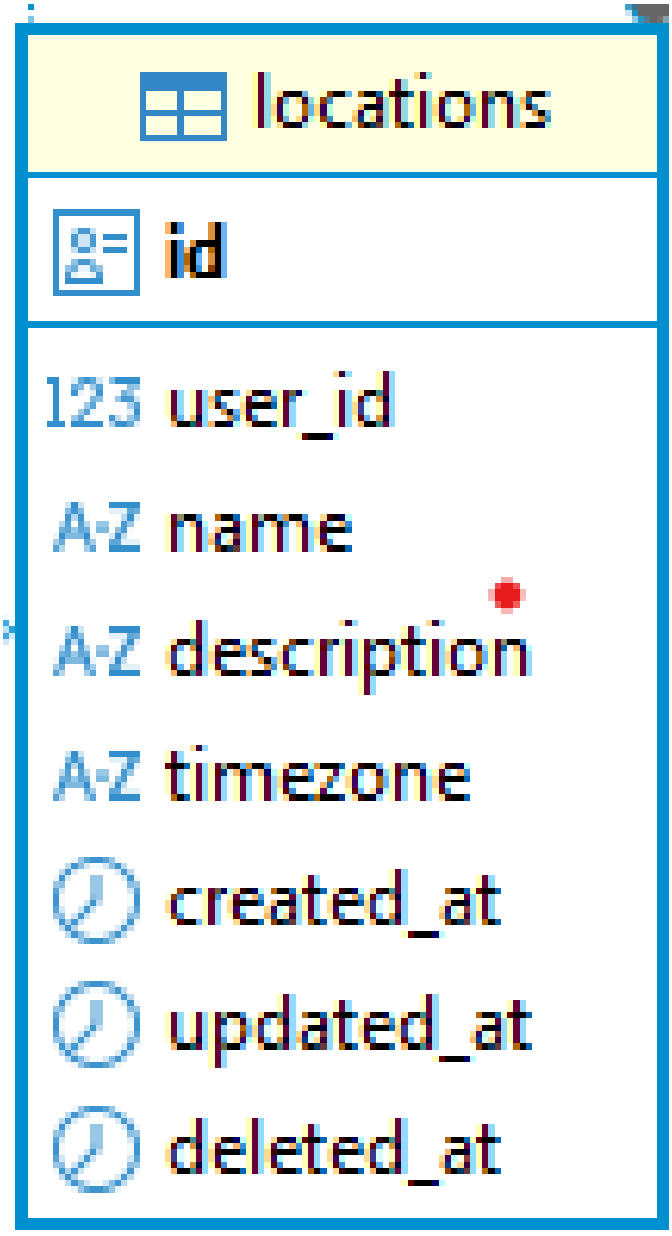
- `id`: identificador único (PK);

- `name`: nome completo do usuário;
- `email`: endereço de e-mail único;
- `password`: senha criptografada (*bcrypt*);
- `timezone`: fuso horário preferencial;
- `is_admin`: define se o usuário possui privilégios administrativos;
- `created_at`, `updated_at`: timestamps de auditoria.

A.2 Tabela Locations (Locais)

Representa os locais físicos onde os pets residem, podendo ser compartilhados entre usuários.

Figura 20 – Estrutura da tabela *Locations*



Fonte: Autoria própria (2025).

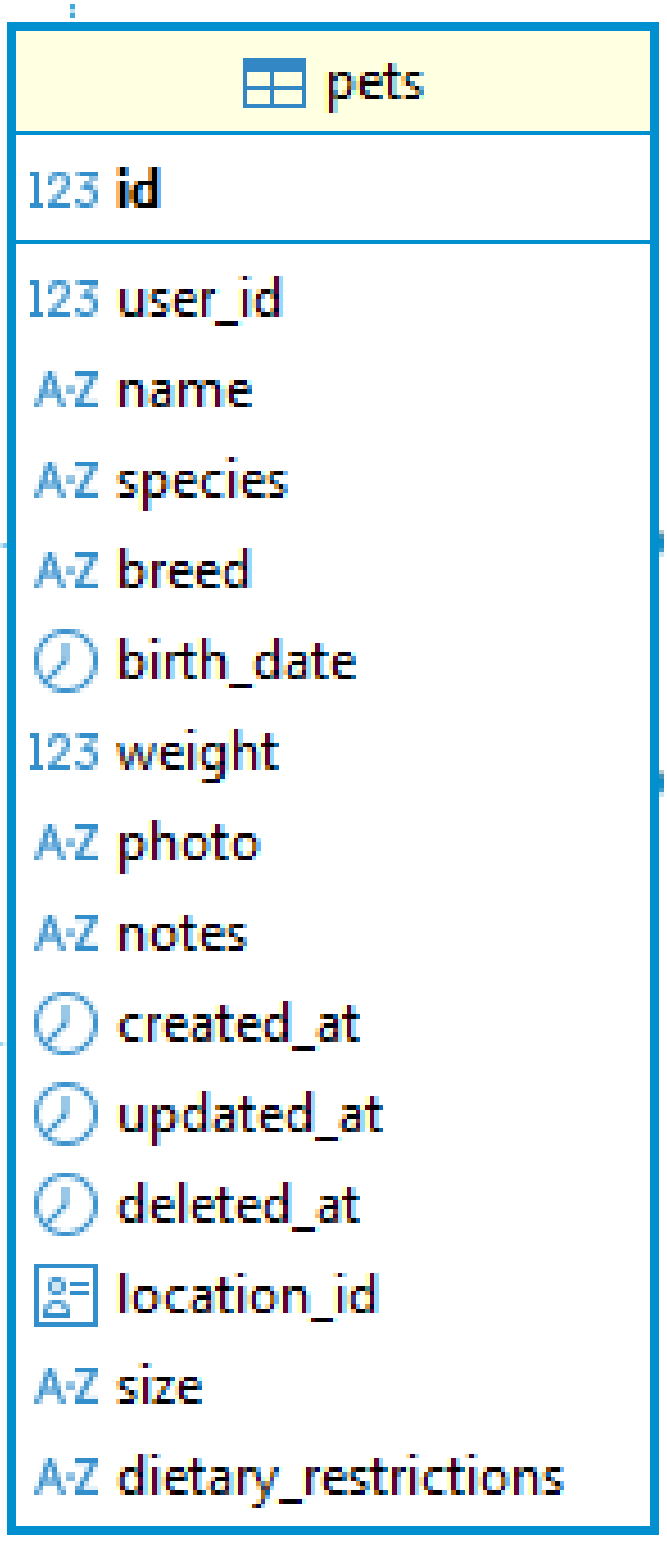
Campos principais:

- `id`: identificador único (UUID, PK);
- `user_id`: referência ao proprietário (FK → users);
- `name`: nome do local (único por usuário);
- `description`: descrição opcional;
- `timezone`: fuso horário específico;
- `deleted_at`: campo de *soft delete*.

A.3 Tabela Pets

Contém os dados dos animais cadastrados pelos usuários, vinculando-os a um tutor e, opcionalmente, a uma localização.

Figura 21 – Estrutura da tabela *Pets*



Fonte: Autoria própria (2025).

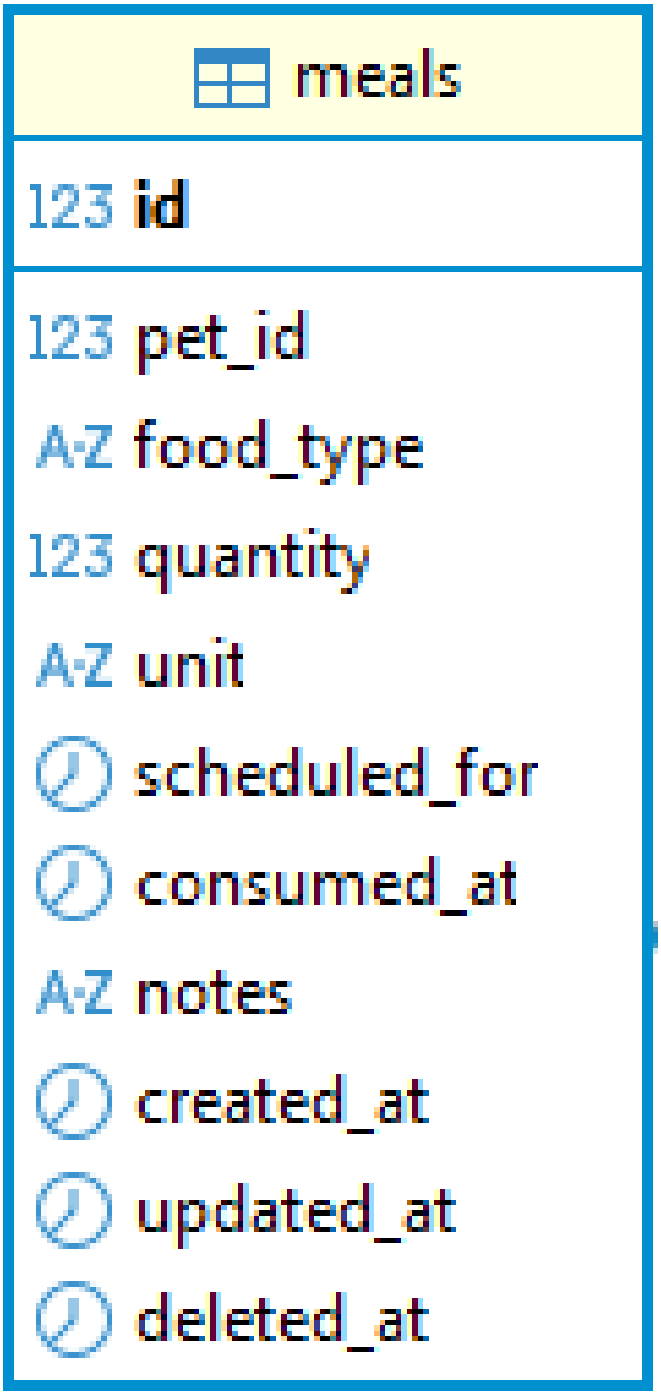
Campos principais:

- `id`: identificador único (PK);
- `user_id`: referência ao tutor (FK → users);
- `location_id`: referência ao local (FK → locations);
- `name`: nome do pet;
- `species`, `breed`: espécie e raça;
- `birth_date`, `weight`: dados biológicos;
- `photo`: URL da imagem no Cloudinary;
- `notes`: observações;
- `deleted_at`: *soft delete*.

A.4 Tabela Meals (Refeições)

Registra o histórico de refeições de cada pet, permitindo o controle de horários, tipos e quantidades de alimento.

Figura 22 – Estrutura da tabela *Meals*



Fonte: Autoria própria (2025).

Campos principais:

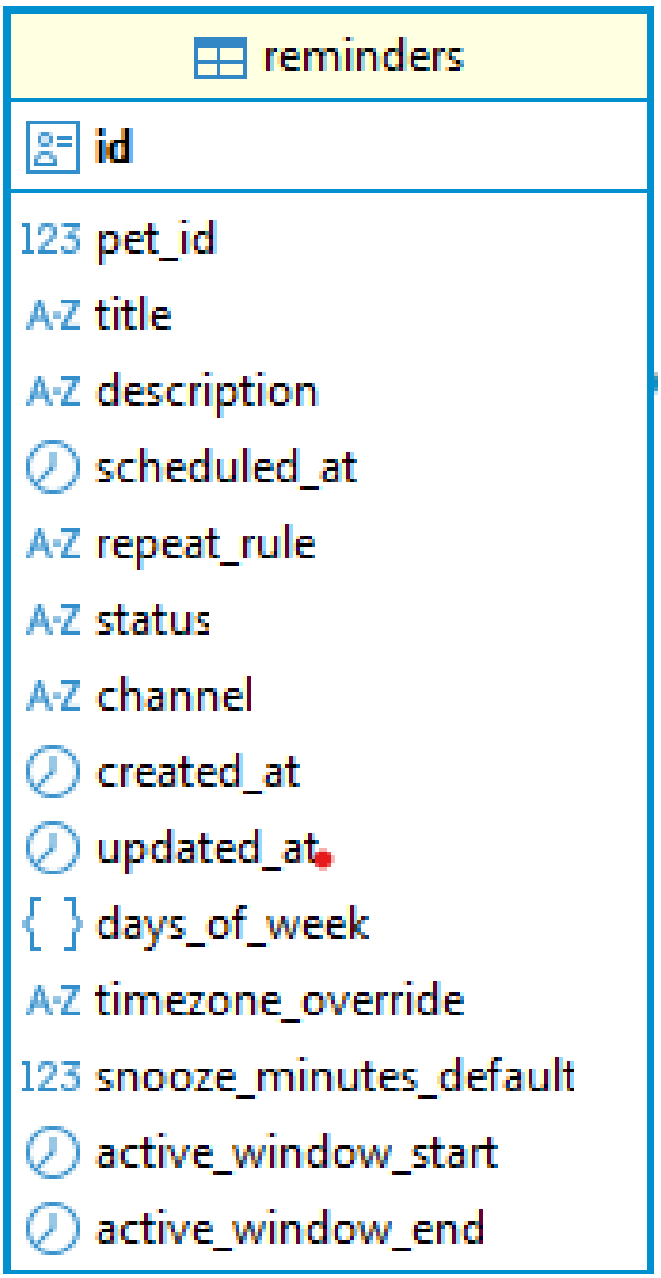
- `id`: identificador único (PK);
- `pet_id`: referência ao pet (FK → pets);
- `food_type`: tipo de alimento;
- `quantity`: quantidade servida;

- `unit`: unidade de medida;
- `scheduled_for`, `consumed_at`: horários de agendamento e consumo;
- `notes`, `deleted_at`: observações e exclusão lógica.

A.5 Tabela Reminders (Lembretes)

Gerencia os lembretes configurados pelos usuários, com suporte a recorrência e notificações automáticas.

Figura 23 – Estrutura da tabela *Reminders*



Fonte: Autoria própria (2025).

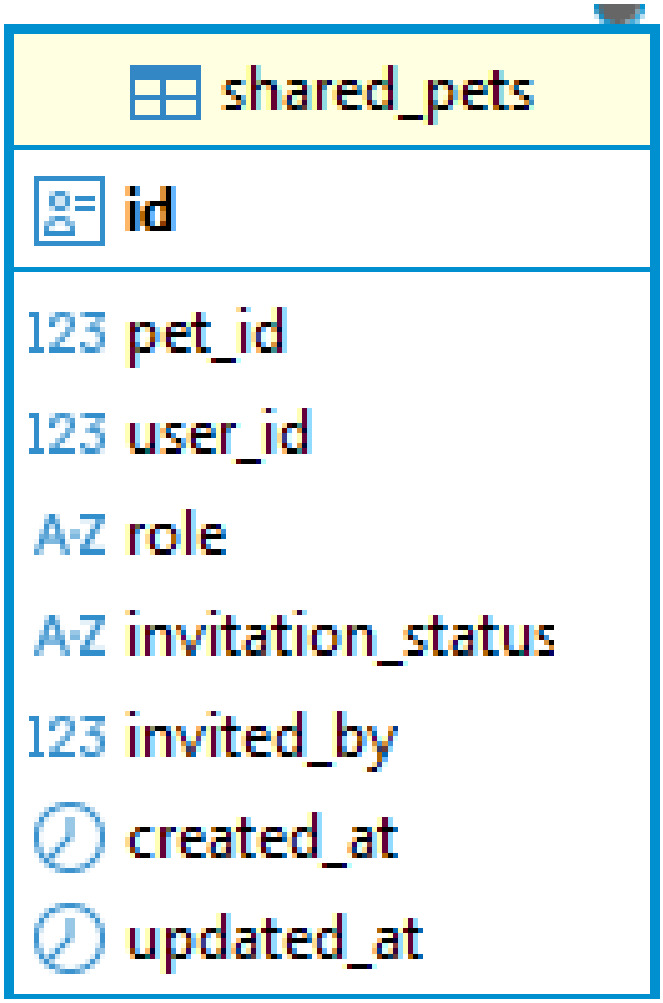
Campos principais:

- `id`: identificador único (UUID, PK);
- `pet_id`: referência ao pet (FK → pets);
- `title`, `description`: título e descrição do lembrete;
- `scheduled_at`: data e hora agendadas;
- `repeat_rule`, `status`, `channel`: controle de recorrência e status;
- `days_of_week`, `timezone_override`: personalizações avançadas;
- `active_window_start`, `active_window_end`: janelas de horário válidas.

A.6 Tabelas de Compartilhamento

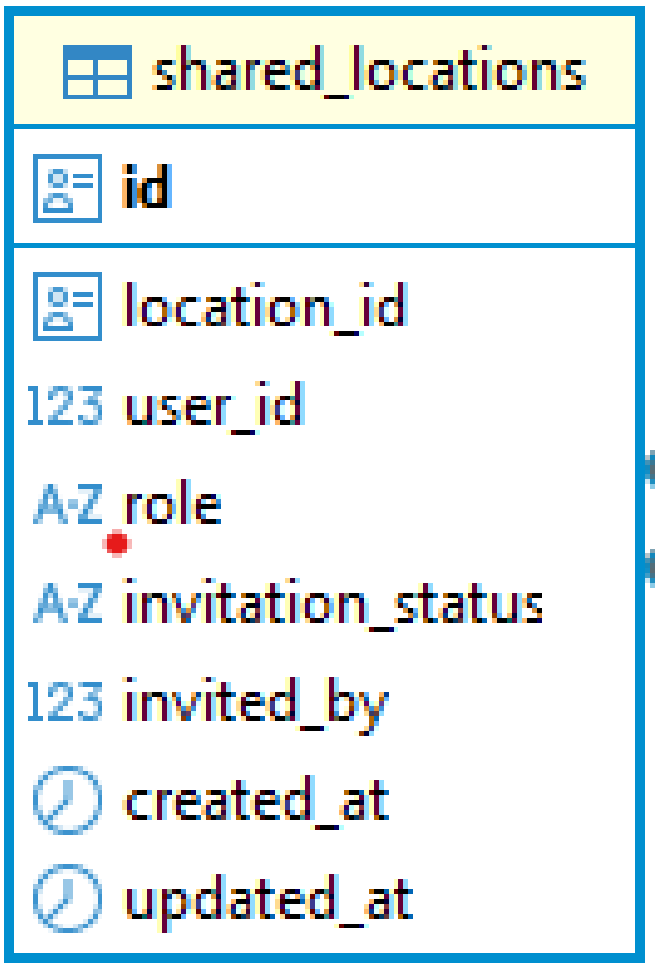
As tabelas **SharedPets** e **SharedLocations** permitem o compartilhamento colaborativo de pets e locais entre diferentes usuários, com papéis e permissões distintas.

Figura 24 – Estrutura da tabela *SharedPets*



Fonte: Autoria própria (2025).

Figura 25 – Estrutura da tabela *SharedLocations*



Fonte: Autoria própria (2025).

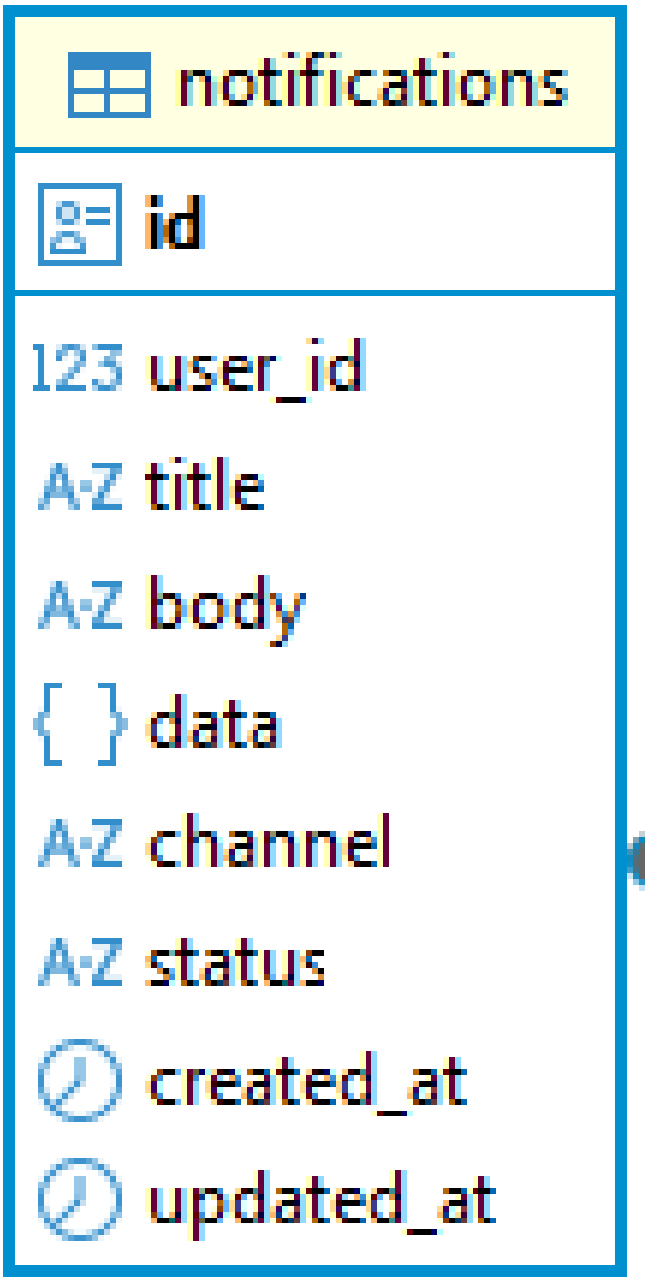
Ambas incluem os campos:

- `id`: identificador único (UUID, PK);
- `user_id`: usuário convidado (FK → users);
- `role`: papel do usuário (owner, editor, viewer);
- `invitation_status`: status do convite (pending, accepted, revoked);
- `invited_by`: referência ao usuário que realizou o convite.

A.7 Tabela Notifications (Notificações)

Registra o histórico de notificações enviadas aos usuários via banco, e-mail ou push.

Figura 26 – Estrutura da tabela *Notifications*



Fonte: Autoria própria (2025).

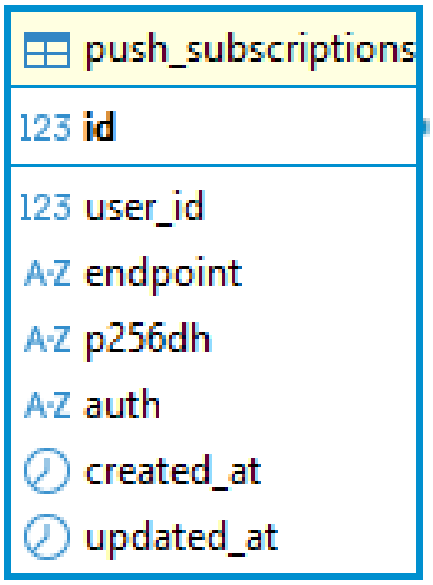
Campos principais:

- `id`: identificador único (UUID, PK);
- `user_id`: destinatário (FK → users);
- `title`, `body`: conteúdo da notificação;
- `data`: informações adicionais (JSON);
- `channel`: canal de envio;
- `status`: estado da notificação (queued, sent, failed, read).

A.8 Tabela PushSubscriptions (Assinaturas Push)

Gerencia as assinaturas de notificações push vinculadas a cada usuário do PWA.

Figura 27 – Estrutura da tabela *PushSubscriptions*



Fonte: Autoria própria (2025).

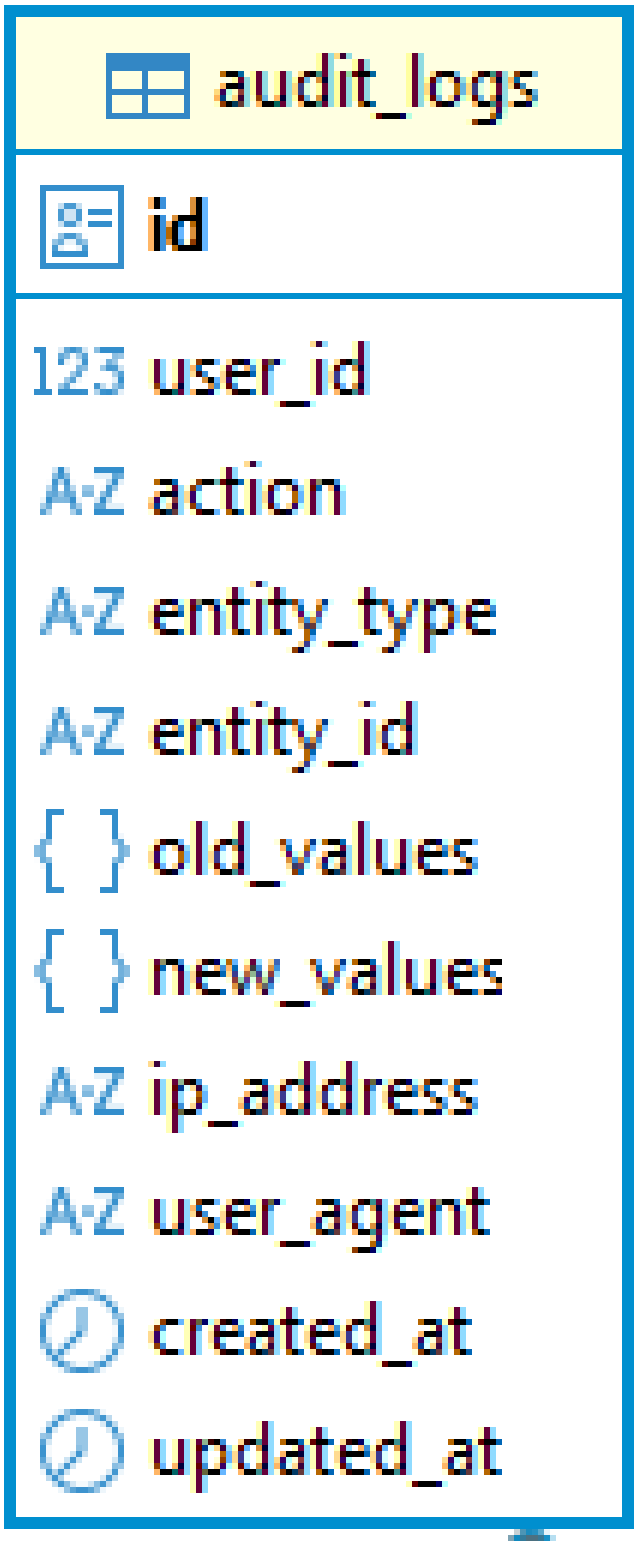
Campos principais:

- `id`: identificador único (PK);
- `user_id`: referência ao usuário (FK → users);
- `endpoint`: URL do endpoint de notificação;
- `public_key`, `auth_token`: dados de autenticação criptográfica.

A.9 Tabela AuditLogs (Logs de Auditoria)

Mantém o histórico de ações executadas pelos usuários, garantindo rastreabilidade completa.

Figura 28 – Estrutura da tabela *AuditLogs*



Fonte: Autoria própria (2025).

Campos principais:

- `id`: identificador único (PK);
- `user_id`: autor da ação (FK → users);

- `action`: tipo da ação (created, updated, deleted, viewed);
- `model_type`, `model_id`: referência ao registro afetado;
- `changes`: objeto JSON com valores antigos e novos;
- `ip_address`, `user_agent`: dados de origem da requisição.

A.10 Resumo dos Relacionamentos

O modelo relacional do UTFPets implementa os seguintes relacionamentos principais:

- **User** $1 \rightarrow *$ **Pets**;
- **User** $1 \rightarrow *$ **Locations**;
- **Pet** $1 \rightarrow *$ **Meals** e **Reminders**;
- **Pet** $1 \rightarrow *$ **SharedPets**;
- **Location** $1 \rightarrow *$ **SharedLocations**;
- **User** $1 \rightarrow *$ **Notifications** e **PushSubscriptions**;
- **Todas as entidades críticas** são registradas em **AuditLogs**.