

**Pró-Reitoria Acadêmica
Escola de Educação, Tecnologia e Comunicação
Curso de Análise e Desenvolvimento de Sistemas
Trabalho de Disciplina de Teste de Software**

Mundo Animal

**Autores: Daniel da Cunha Torres, Allan Barros de
Medeiros Miron, Nathanael Victor Paiva Magno.**

Orientador: Prof. Fabiano Oliveira de Carvalho

**Brasília - DF
2025**

**Daniel da Cunha Torres, Allan Barros de Medeiros Miron, Nathanael Victor Paiva
Magno.**

Mundo Animal

Documento apresentado ao Curso de graduação de Análise e Desenvolvimento de Sistemas da Universidade Católica de Brasília, como requisito parcial para obtenção da aprovação na disciplina de teste de software.

Orientador: Prof. Fabiano Oliveira de Carvalho

**Brasília
2025**

LISTA DE FIGURAS

Figura 1 - Diagrama de Casos de Uso de <i>Software</i>	14
Figura 2 - Tela de criação de novo projeto.	Erro! Indicador não definido.

SUMÁRIO

LISTA DE FIGURAS	3
1 INTRODUÇÃO.....	6
1.1 DIAGNÓSTICO DA EMPRESA / TEMA	7
2 OBJETIVOS	8
2.1 OBJETIVO GERAL	8
2.2 OBJETIVOS ESPECÍFICOS.....	8
3 PROPOSTA DO SISTEMA	9
3.1 DESCRIÇÃO DO SISTEMA PROPOSTO	9
3.2 RESULTADOS ESPERADOS	10
4 FERRAMENTAS UTILIZADAS.....	10
5 ANÁLISE DE NEGÓCIO	11
5.1 REGRAS DE NEGÓCIO.....	11
6 ANÁLISE DE REQUISITOS	12
6.1 REQUISITOS FUNCIONAIS	12
6.2 REQUISITOS NÃO-FUNCIONAIS.....	13
6.3 DIAGRAMA DE CASOS DE USO DA SOLUÇÃO	13
6.3.1 Visão Geral dos Casos de Uso e Atores.....	13
6.4 DIAGRAMA DE CLASSE.....	14
6.4.1 Visão Geral do digrama de Classe	14
6.4.2 Protótipos de Tela do Sistema	15
6.4.2.1 Histórias do Usuário	17
6.4.3 Planejamento de testes	18
6.4.4 Casos de Testes	19
6.4.5 Execução e Evidências dos Testes	25
6.4.6 Link do Repositório	31
7 CONCLUSÃO E LIÇÕES APRENDIDAS.....	32
REFERÊNCIAS	33

1 INTRODUÇÃO

A computação vem com papel essencial nas pequenas e grandes organizações comerciais, modernizando e automatizando com uma grande eficiência nos processos internos e externos. A adoção de tecnologias da informação, como sistemas informatizados de cadastro de clientes, controles de estoque, análise de finanças, permitindo gerenciar vendas, lucros, visão geral de clientes mais precisa e ágil. Assim, esses avanços estão cada vez mais contribuindo em decisões estratégicas, aumentando produtividade e reduzindo os erros operacionais.

Para solucionar esse problema, este projeto propõe o desenvolvimento de um sistema web focado no gerenciamento de cadastros de pets. A plataforma contará com dois perfis de acesso principais: Usuário (tutor) e Funcionário. O usuário poderá realizar seu próprio cadastro, fazer login e gerenciar as informações de seus animais de estimação, incluindo o cadastro, a atualização, a exclusão e a pesquisa de seus pets.

Por outro lado, o funcionário terá uma visão administrativa do sistema, com permissões para visualizar todos os usuários e seus respectivos pets, listar todos os animais cadastrados na plataforma, pesquisar por pets específicos e, quando necessário, atualizar ou remover registros de animais e usuários, garantindo a integridade e a organização dos dados. Seguindo esse raciocínio o sistema contará com um super usuário (root) que está definido como “admin”, podendo gerenciar os funcionários, usuários e pets do sistema.

Este projeto está estruturado em capítulos que acompanham cada etapa do desenvolvimento do sistema. Inicialmente, descreveremos os problemas enfrentados, analisaremos os recursos disponíveis e destacaremos as principais ferramentas que serão utilizadas, evidenciando sua eficiência. Em seguida, será apresentado o processo de elaboração do sistema, incluindo a definição dos requisitos, o planejamento dos testes e a implementação das funcionalidades. Por fim, serão demonstrados os resultados obtidos com a aplicação da solução proposta, assim como as conclusões e lições aprendidas ao longo do projeto.

1.1 DIAGNÓSTICO DA EMPRESA / TEMA

A gestão de informações de animais de estimação representa um desafio comum tanto para tutores individuais quanto para organizações. O controle manual desses dados, seja por meio de anotações em papel ou planilhas desorganizadas, frequentemente resulta na perda de informações importantes, dificuldade no acesso a históricos e ineficiência na gestão geral. Essa falta de um sistema centralizado impacta negativamente a capacidade de manter um registro preciso e acessível.

Para tutores, a dificuldade reside em consolidar o histórico de seus animais — como informações de saúde, vacinas e outros dados relevantes — em um único local. Para organizações que lidam com um volume maior de animais, a ausência de uma ferramenta digital adequada aumenta a probabilidade de erros e retrabalho, comprometendo a eficiência operacional.

Diante desse cenário, torna-se evidente a necessidade de um sistema digital que centralize as informações, facilite o controle dos cadastros e melhore a organização dos dados, atendendo às necessidades de gerenciamento tanto de usuários individuais quanto de entidades com múltiplos animais.

2 OBJETIVOS

Este capítulo tem como finalidade apresentar os objetivos do desenvolvimento do sistema "Mundo Animal", um site voltado para a gestão de serviços e cuidados com os pets. A proposta é oferecer uma solução completa que atenda tanto aos clientes quanto aos funcionários do pet shop, proporcionando praticidade, organização e melhor comunicação entre ambas as partes.

2.1 OBJETIVO GERAL

Desenvolver um sistema web para o gerenciamento de informações de animais de estimação, permitindo que usuários cadastrem e administrem seus pets e que funcionários autorizados gerenciem todos os dados de usuários e animais contidos na plataforma de forma centralizada e eficiente. E por final, o administrador do sistema que será capaz de gerenciar todos os dados produzidos pelos funcionários e usuários.

2.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral, o projeto se concentrará na implementação de um sistema de autenticação seguro com três níveis de acesso: Usuário, Funcionário e Admin. Os usuários terão autonomia para gerenciar o ciclo de vida completo de seus pets, incluindo as operações de criar, ler, atualizar, deletar (CRUD) e pesquisar seus próprios animais. O perfil de funcionário terá um escopo de atuação mais amplo, com permissões para visualizar todos os usuários e seus respectivos pets, além de poder buscar, atualizar e deletar qualquer pet existente na plataforma. No topo da hierarquia, o super usuário (admin) terá controle total e irrestrito sobre o sistema. Suas responsabilidades incluem o gerenciamento completo de todas as entidades: ele poderá criar, visualizar, editar e remover (CRUD) tanto usuários quanto funcionários, além de possuir todas as permissões de gerenciamento de pets, assegurando a administração global da plataforma.

3 PROPOSTA DO SISTEMA

A seguir será apresentada a proposta do sistema, visando detalhar os principais pontos a serem seguidos.

3.1 DESCRIÇÃO DO SISTEMA PROPOSTO

O sistema descrito neste documento irá atender às necessidades operacionais e comerciais do pet shop "Mundo Animal", com funcionalidades voltadas para a utilização estruturada em um modelo de acesso hierárquico com três perfis distintos: Usuário, Funcionário e Admin. A proposta do sistema é fornecer uma solução completa para o monitoramento de dados do pet shop, assegurando a segurança, a integridade dos dados e um controle de permissões bem definido em toda a aplicação.

Diferente de outras soluções existentes no mercado, o sistema do "Mundo Animal" se destaca por oferecer uma abordagem integrada que atende tanto os clientes, com foco em facilidade e autonomia, quanto os funcionários, com foco em organização e controle de processos. Os clientes irão realizar seu cadastro e login, o usuário entra em um ambiente pessoal e restrito, onde possui total autonomia para gerenciar as informações de seus próprios animais. Nesse painel, ele pode executar todo o ciclo de operações CRUD: cadastrar novos pets, visualizar a lista de seus animais, editar informações existentes e remover registros, além de poder realizar buscas dentro de sua própria base de dados.

Já os funcionários, que possui permissões administrativas para a manutenção dos dados de pets. Com seu acesso, um funcionário pode visualizar todos os usuários cadastrados e os animais associados a cada um deles. Além disso, ele tem a capacidade de gerenciar o ciclo de vida de qualquer pet no sistema, podendo editar informações para corrigir dados ou remover registros que não são mais válidos, garantindo a qualidade e a precisão da base de dados geral.

No topo da hierarquia encontra-se o **Admin**, o super usuário com controle total sobre a plataforma. Suas permissões transcendem o gerenciamento de pets e englobam a administração dos próprios acessos ao sistema. O Admin é o único perfil capaz de criar, visualizar, editar e remover contas de Usuários e Funcionários. Essa capacidade de gerenciamento de contas, somada ao controle completo sobre todos os pets, confere ao Admin a governança global do sistema, permitindo a gestão completa da estrutura e dos utilizadores da plataforma.

Assim, o "Mundo Animal" se apresenta como uma solução moderna e intuitiva, que une tecnologia e cuidado com os animais, promovendo uma experiência completa, segura e prática tanto para os tutores quanto para a equipe do pet shop.

3.2 RESULTADOS ESPERADOS

Com a implantação do Pet Shop – Mundo Animal, esperam-se os seguintes resultados:

- Centralização e organização das informações de pets;
- Autonomia para os usuários na gestão dos dados de seus próprios animais;
- Controle administrativo total para os funcionários sobre os dados da plataforma;
- Redução de erros operacionais através da digitalização do processo de cadastro;
- Acesso rápido e facilitado às informações através de ferramentas de busca;

4 FERRAMENTAS UTILIZADAS

Para o desenvolvimento do sistema do Pet Shop – Mundo Animal, serão utilizadas ferramentas modernas e amplamente adotadas no mercado, visando garantir eficiência, escalabilidade e uma boa experiência para o usuário final. A seguir, são descritas as tecnologias que compõem o projeto:

Frontend – React

A interface do sistema será desenvolvida com React, uma biblioteca JavaScript amplamente utilizada para construção de interfaces de usuário modernas e dinâmicas. A escolha do React se deve à sua flexibilidade, reutilização de componentes e à facilidade de integração com outras bibliotecas e ferramentas, o que contribui para uma navegação fluida e responsiva no sistema.

Backend – Node.js com Mongoose

A camada de servidor será construída com Node.js, que permite desenvolver aplicações rápidas e escaláveis utilizando JavaScript também no lado do servidor. Será utilizado o Mongoose como ferramenta de modelagem de dados, facilitando a interação entre o Node.js e o banco de dados MongoDB. Essa combinação oferece produtividade no desenvolvimento, além de uma integração eficiente com o banco de dados não relacional.

Banco de Dados – MongoDB

O armazenamento das informações será feito com o MongoDB, um banco de dados NoSQL que oferece grande flexibilidade para trabalhar com dados sem estrutura rígida. Sua escolha está relacionada à necessidade de armazenar dados diversos, como perfis de usuários, informações dos pets, agendamentos e histórico de serviços, de forma ágil e organizada.

Essas ferramentas foram escolhidas por proporcionarem um desenvolvimento rápido, robusto e compatível com as necessidades do sistema, além de serem tecnologias atuais e com ampla comunidade de suporte.

5 ANÁLISE DE NEGÓCIO

Este capítulo apresenta a análise do funcionamento do negócio no qual o sistema **"Mundo Animal"** será inserido. O objetivo é compreender como o pet shop opera atualmente, identificando e expressando as regras que orientam o uso das ferramentas envolvidas no projeto.

5.1 REGRAS DE NEGÓCIO

As regras de negócio representam as diretrizes que sustentam a existência e o funcionamento do sistema. A seguir, estão descritas as principais regras identificadas:

Número	Nome	Descrição	Setor
RN1	Autenticação de Usuário	O usuário deve estar autenticado para acessar a área de gerenciamento de pets.	Cliente
RN2	Gestão de Pets do Usuário	O usuário só pode visualizar, editar e excluir os pets que ele mesmo cadastrou.	Usuário
RN3	Autenticação de Funcionário	O funcionário deve estar autenticado para acessar a área administrativa.	Funcionário
RN4	Visualização de Dados	O funcionário pode visualizar todos os usuários e todos os pets cadastrados no sistema.	Funcionário
RN5	Gerenciamento de Pets pelo Funcionário	O funcionário pode editar ou excluir qualquer pet cadastrado no sistema.	Funcionário
RN6	Autenticação de Admin	O admin deve estar autenticado para acessar o painel de administração geral.	Admin
RN7	Admin autenticando Funcionário	O admin ele autentica o funcionário com credencial de funcionário	Admin
RN8	Gerenciamento de Contas	Apenas o admin pode criar, editar e remover contas de usuários e funcionários.	Admin
RN9	Controle Total do Admin	O admin possui todas as permissões dos funcionários e pode gerenciar todos os dados do sistema.	Admin

6 ANÁLISE DE REQUISITOS

Neste capítulo será descrito, através de diagramas e especificações, o processo do “Mundo Animal” em que o *software* em questão será inserido, sendo estes o diagrama do modelo de caso de uso de negócio, diagrama do modelo de classes do negócio, e, por fim, o diagrama de atividades.

6.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais descrevem o que o sistema deve fazer. Eles especificam as funcionalidades e os serviços que devem ser fornecidos aos usuários, detalhando as operações para cada perfil (Usuário, Funcionário e Admin).

Número	Requisitos Funcionais	RN
RF1	O sistema deve permitir que um novo usuário se cadastre através de janela de cadastro	RN01
RF2	O sistema deve permitir a autenticação (login) para os perfis de Usuário e Funcionário.	RN01, RN03, RN06
RF3	O sistema deve permitir que um usuário autenticado cadastre um ou mais pets.	RN02
RF4	O sistema deve permitir que um usuário autenticado visualize a lista de seus pets.	RN02
RF5	O sistema deve permitir que um usuário autenticado atualize as informações de seus pets.	RN02
RF6	O sistema deve permitir que um usuário autenticado exclua um de seus pets.	RN02
RF7	O sistema deve permitir que um usuário autenticado pesquise em sua lista de pets.	RN02
RF8	O sistema deve permitir que um funcionário autenticado visualize a lista de todos os usuários.	RN04
RF9	O sistema deve permitir que um funcionário autenticado visualize a lista de todos os pets.	RN04
RF10	O sistema deve permitir que um funcionário autenticado edite ou exclua qualquer pet do sistema.	RN05
RF11	O sistema deve permitir que um funcionário autenticado edite ou exclua qualquer usuário do sistema.	RN05
RF12	O sistema deve permitir que o admin autenticado crie, visualize, edite e remova contas de funcionários.	RN08, RN09
RF13	O sistema deve permitir que o admin autenticado crie, visualize, edite e remova contas de usuários.	RN08, RN09
RF14	O sistema deve garantir que o admin tenha acesso a todas as funcionalidades de um funcionário.	RN09, RN07

6.2 REQUISITOS NÃO-FUNCIONAIS

Os requisitos não-funcionais definem como o sistema deve operar, estabelecendo critérios de qualidade, desempenho, segurança e usabilidade, que são essenciais para a experiência do usuário e a robustez da aplicação.

Número	Requisitos Não-Funcionais	RF
RNF1	Desempenho de Resposta: O tempo de resposta para as operações críticas do sistema (login, consultas, salvamentos) deve ser, em média, inferior a 2 segundos.	Todos
RNF2	Criptografia de Senhas: Todas as senhas devem ser armazenadas no banco de dados usando um algoritmo de hashing forte (ex: bcrypt), impedindo o acesso em texto claro.	RF01, RF02, RF12, RF13
RNF3	Controle de Acesso por Papel (RBAC): O sistema deve garantir que cada perfil (Usuário, Funcionário, Admin) acesse apenas as funcionalidades e dados autorizados para sua função.	Todos
RNF4	Compatibilidade de Navegadores: A aplicação deve ser totalmente funcional e renderizar corretamente nas duas últimas versões estáveis dos principais navegadores (Chrome, Firefox, Edge, Safari).	Todos
RNF5	Manutenibilidade do Código: O código-fonte deve ser bem documentado e estruturado, seguindo boas práticas de desenvolvimento para facilitar futuras manutenções e evoluções.	Todos

6.3 DIAGRAMA DE CASOS DE USO DA SOLUÇÃO

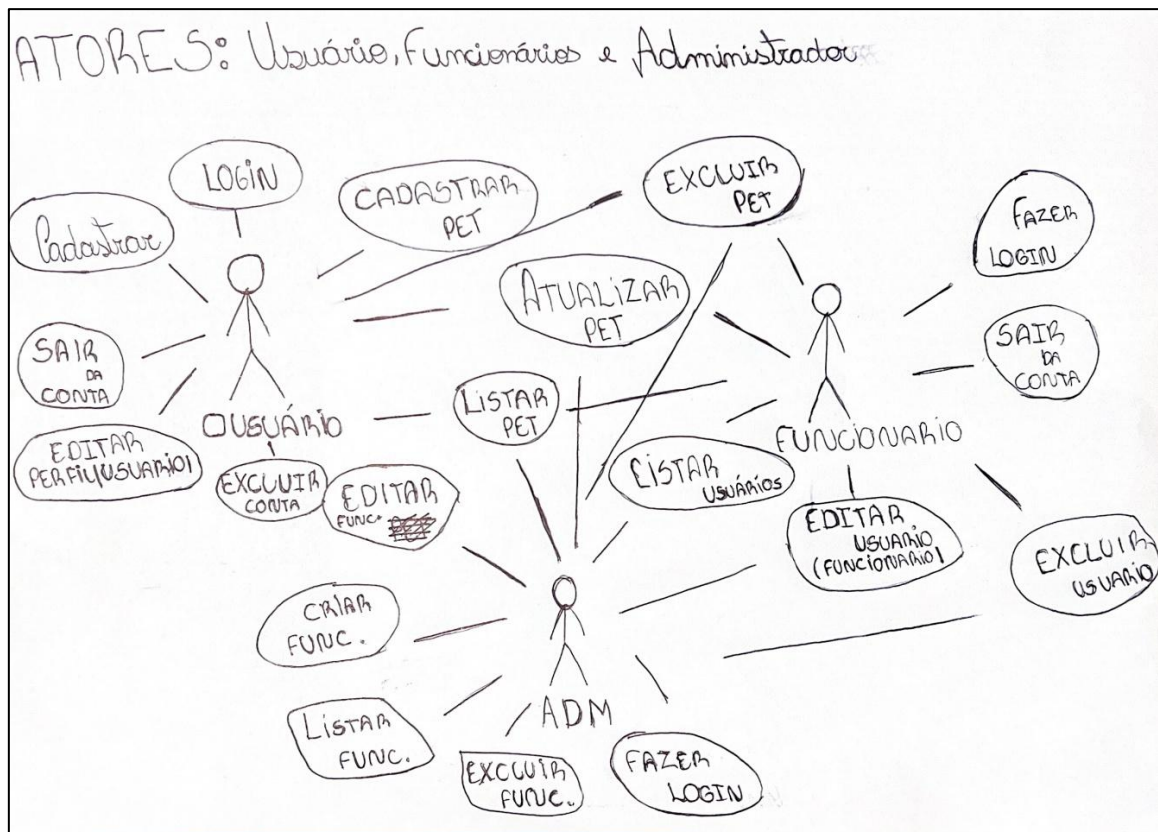
O Diagrama de Casos de Uso descreve as interações entre os atores (os perfis de usuário) e o sistema. Ele ilustra as diferentes funcionalidades que cada ator pode acessar, oferecendo uma visão geral do escopo e do comportamento do software.

Nesta seção serão definidos os modelos de casos de uso. Primeiramente será mostrada uma visão geral dos casos de uso que definem as funcionalidades do sistema, com seus respectivos atores.

6.3.1 Visão Geral dos Casos de Uso e Atores

A Figura 1 a seguir será apresentado o Diagrama de Casos de Uso de *Software* com a visão de cada ator do sistema o usuário, funcionário e o administrador, abrangendo assim todas as funcionalidades previstas para a implementação.

Figura 1 - Diagrama de Casos de Uso de *Software*.



Fonte: Elaboração própria, 2025.

6.4 DIAGRAMA DE CLASSE

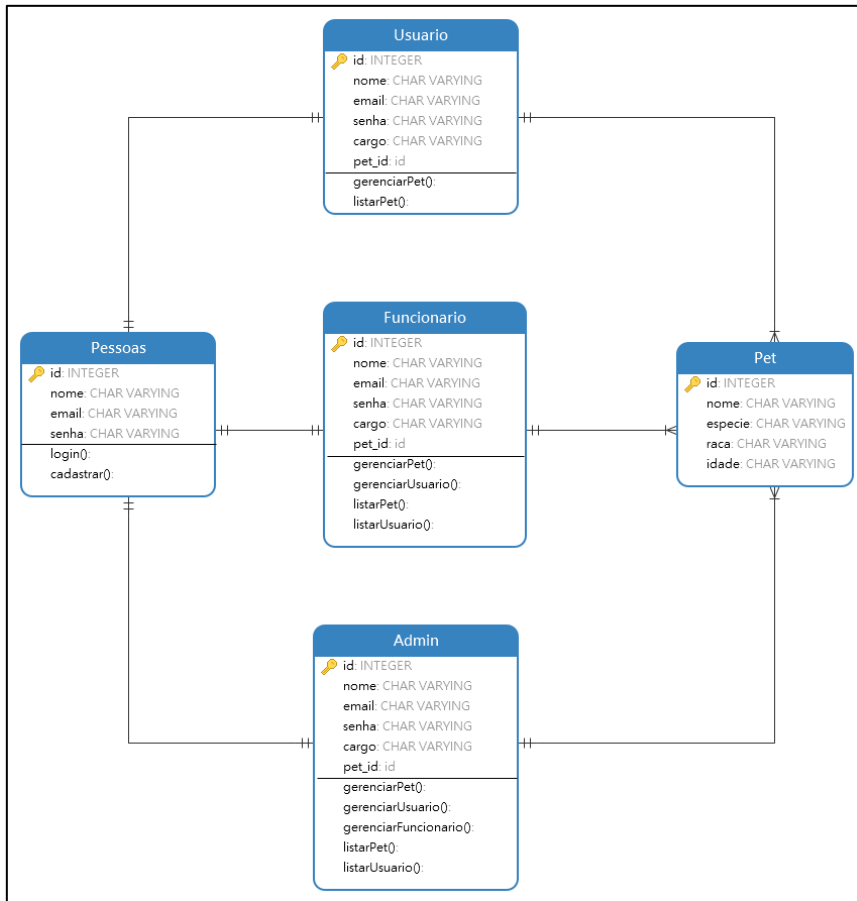
O Diagrama de Classe modela a estrutura estática do sistema. Ele exibe as classes que compõem o sistema, seus atributos e métodos, e os relacionamentos entre elas, servindo como um blueprint para a arquitetura do banco de dados e do código.

Nesta seção será mostrado o diagrama de classes do sistema. Primeiramente será mostrada uma visão geral do diagrama de classes do sistema do pet shop.

6.4.1 Visão Geral do diagrama de Classe

Aqui nessa figura está a exibição do diagrama de Classes:

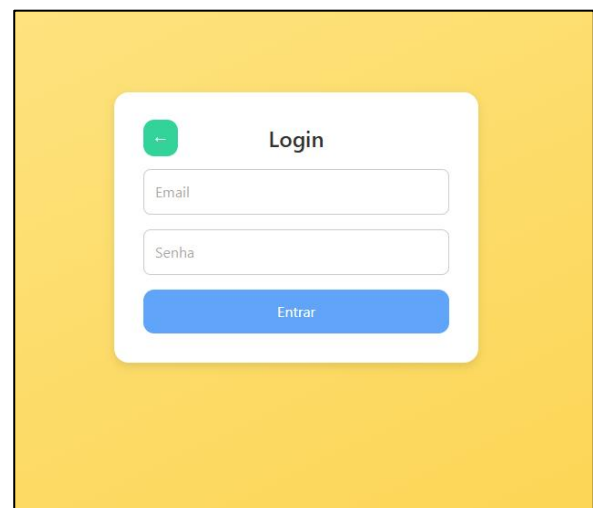
Figura 2 - Diagrama de Classes.



Fonte: Elaboração própria, 2025.

6.4.2 Protótipos de Tela do Sistema

Telas para login:



Telas de home para funcionário e usuário:

Painel

Usuários

Pets

Sair

Visão Cliente

Lista de Clientes

Pesquisar por nome ou e-mail...

☐ Cliente ☐ Admin ☐ Funcionário

ID do Cliente	Nome	E-mail	Cargo	Ações
6844e138e04ee3303f4037b7	Allan	allan@.com	cliente	<button>Alterar</button> <button>Excluir</button>
6844e140443be1449a2a6482	Allan	allan@a.com	cliente	<button>Alterar</button> <button>Excluir</button>
6844e14f3bcd901b7672786	Allan	allana@a.com	cliente	<button>Alterar</button> <button>Excluir</button>
6844e1eed15e2f53e0180bfd	Allan	allanaaaaa@a.com	cliente	<button>Alterar</button> <button>Excluir</button>
6844e223465222cf3427bd74	Allan	allanaaaaaaaaaa@a.com	funcionario	<button>Alterar</button> <button>Excluir</button>


MUNDO ANIMAL

Pesquisar por nome...

Olá, Perfil ADM ▼

Meus Pets

+ Registrar Novo Pet



Você ainda não possui pets registrados.

6.4.2.1.1 Histórias de Usuário

Campo	Conteúdo
Título	Login de Usuário
Descrição	O sistema deve permitir que usuários já cadastrados consigam acessar suas contas, com as credenciais corretas (e-mail e senha).
Critérios de Aceitação	1. Todos os dados são obrigatórios. 2. Deve haver uma validação para autenticar os dados informados.
Regras de Negócio	1. RN01- O usuário deve ser autenticado para poder passar para a tela home da sua sessão.
História de Usuário	Como usuário, eu quero logar no sistema de pet shop, para que eu possa acessar o gerenciamento dos meus pets.
Cenários BDD	Cenário 1: Login do usuário com suas credenciais.
	Dado usuário encontrado no banco de dados
	Quando eu tento logar em minha conta e-mail "tim@email.com" e senha "Senha123!" Então o usuário conseguirá acessar o sistema de forma autenticada.
Prioridade	Muito Alta
Tarefas Técnicas	- Criar input para preenchimento de dados. - Validar e-mail e o hash da senha no banco de dados. - Redirecionar para a página correta.
Dependências	Integração com o banco de dados confirmação do login.
Definição de Pronto	- Código implementado e revisado. - Testes unitários e de integração passando. - Cenários BDD automatizados e passando.

6.4.3 Planejamento de testes

Nesta seção, você define a estratégia geral.

Objetivo dos testes: Garantir a qualidade, funcionalidade e confiabilidade do sistema "Mundo Animal" em todas as suas camadas (componentes, API e fluxos de usuário), assegurando que os requisitos funcionais e as regras de negócio sejam atendidos.

Escopo do teste:

O que será testado:

Testes Unitários: Funções de lógica de negócio isoladas (ex: validação de dados), componentes de UI (React) e seus comportamentos individuais.

Testes de API: Todos os endpoints da API REST, incluindo operações de CRUD para usuários, pets e funcionários, autenticação (login/logout) e validação de dados de entrada/saída.

Testes E2E (End-to-End): Fluxos completos do usuário, como o processo de cadastro de um novo tutor, login, cadastro de um pet, edição e exclusão.

O que não será testado: Testes de carga/performance, testes de usabilidade com usuários finais e testes de compatibilidade em dispositivos móveis legados.

Ferramentas que serão utilizadas:

Testes Unitários: Vitest.

Testes de API: Swagger.

Testes E2E: Cypress.

Estratégia de testes: A abordagem será a da **Pirâmide de Testes**. Daremos foco a uma base sólida de testes unitários rápidos, seguida por uma camada de testes de API para garantir os contratos de serviço, e finalizando com alguns testes E2E para validar os fluxos críticos do sistema.

Papéis e responsabilidades:

Desenvolvedores backend: Responsáveis por criar os testes unitários para o código que desenvolvem.

Desenvolvedores frontend e API: Colaboram na criação e manutenção dos testes de API e E2E.

6.4.4 Casos de Testes

Testes Unitário:

Seguindo a analogia da equipe iremos começar pelo teste unitário, analisando de forma individual de cada funcionalidade.

Identificador: TU-01

Descrição: Verifica se a função *userService.create* usada para cadastro de usuário, dessa forma testaremos se retornará *sucess*.

Dados de Entrada:

Iremos dar os valores preenchendo os *inputs*.

Body: { name: 'Joao', email: 'joao@email.com', password: '123456' }

Saída esperada:

Iremos analisar se o status de saída foi igual a 201 *Created*.

Identificador: TU-02

Descrição: Verifica se a função *userService.create* usada para cadastro de usuário, diante isso deixaremos um campo vazio, para testarmos se retornará o erro indicando que todos os campos são obrigatórios.

Dados de Entrada:

Iremos dar somente esse valor para o *input* deixando os outros vazios

Body: { name: 'Joao' }

Saída esperada:

Iremos analisar se o status de saída foi igual a 400 *not found*.

Identificador: TU-03

Descrição: Verifica se a função *userService.findAll* usado para listar todos os usuários do banco de dados, assim iremos ver se retornará todos os usuários do banco de dados.

Dados de Entrada:

Usaremos para simular todos os usuários como:

```
[
  { id: 1, name: 'Joao', email: 'joao@email.com' },
  { id: 2, name: 'Maria', email: 'maria@email.com' }
]
```

Saída esperada:

Esperamos que a função seja capaz de retornar todos os usuários, e como resposta status 200 *Ok*.

Identificador: TU-04

Descrição: Verifica se a função *adminUpdateUser(req, res)* usado para atualizar cargo do usuário podendo ser executado somente por admin.

Dados de Entrada:

Dessa forma usaremos

body {role:'gerente'} (que não existe)

Saída esperada:

Iremos analisar se o status de saída foi igual a 400 *not found*, e o erro que o “cargo não existe”.

Identificador: TU-05

Descrição: Verifica se a função *userService.delete* usado para deletar usuários, se é capaz deletar o usuário com sucesso.

Dados de Entrada:

Dessa forma usaremos com requisição

{userId: 1, decodedId: 1, params: {id: 1}}

Saída esperada:

Dessa forma esperamos como status 200 *Ok*, e que seja capaz de deletar o usuário.

Identificador: TU-06

Descrição: Verifica se a função *userService.delete* usado para deletar usuários, se retorna um erro quando é dado os valores errados.

Dados de Entrada:

Dessa forma usaremos com requisição

{userId: 1, decodedId: 2, params: {id: 1}}

Saída esperada:

Dessa forma esperamos como status 500 *Internal Server Error*, e que não seja capaz de deletar o usuário.

Execução dos testes unitários:

Para analisarmos todos os testes unitários usaremos a biblioteca Vitest para a validação, eles foram executados dentro do arquivo `user.controller.test.js`. dentro da pasta teste.

Teste de API:

Após a execução do primeiro teste partiremos para o teste de API, neste testaremos autenticações e rotas.

Identificador: TAPI-01

Descrição: Verifica se a API **impede** a criação de um novo pet e retornando o erro 401 *Unauthorized* quando um token de autenticação inválido é fornecido.

Dados de Entrada:

Iremos fazer uma requisição POST para `/api/pet` passar o body `{ "user": "", "name": "Thor", "type": "Cachorro", "age": "10", "breed": "Pitbull" }` porém no lugar de informar o token na ferramenta iremos falar um aleatório como `"token: 1234"`

Saída esperada:

Iremos analisar se o status de saída foi igual a 401 *Unauthorized* quando um token de autenticação inválido é fornecido.

Identificador: TAPI-02

Descrição: Verifica se a API fornece os dados de todos os usuários do banco de dados.

Dados de Entrada:

Iremos fazer uma requisição GET para `/api/user`.

Saída esperada:

Assim esperamos como resposta status 200 *Ok*, e a lista de todos os usuários dentro do padrão JSON.

Identificador: TAPI-03

Descrição: Verifica se a API fornece os dados de todos os pets do banco de dados.

Dados de Entrada:

Iremos fazer uma requisição GET para `/api/pet`.

Saída esperada:

Assim esperamos como resposta status 200 *Ok*, e a lista de todos os pets dentro do padrão JSON.

Identificador: TAPI-04

Descrição: Verifica se a API fornece os dados de específico do usuário no banco de dados.

Dados de Entrada:

Iremos fazer uma requisição GET para `/api/user/{id}/`.

Saída esperada:

Assim esperamos como resposta status 200 *Ok*, e o formato json dos dados do usuário buscado.

Identificador: TAPI-05

Descrição: Verifica se a API consegue buscar um pet específico com um id invalido.

Dados de Entrada:

Iremos fazer uma requisição GET para `/api/pet/iderrado123/`.

Saída esperada:

Assim esperamos como resposta status 400 *bad request*, e não retornar nenhum pet em específico.

Execução dos testes de API:

Para a execução apresentada no próximo capítulo usamos a biblioteca swagger, e quando executada seremos encaminhados para sua rota no navegador. Assim sendo capaz de executar vários testes de API.

Teste de E2E:

Aqui será onde concluiremos os testes, juntando todas as informações adquiridas, para verificar se o sistema apresentará um resultado sofisticado como prometido na documentação. Dessa forma no identificador TE2E-01 será um fluxo completo pela aplicação.

Identificador: TE2E-01

Descrição: Simularemos um fluxo completo pelo sistema.

Dados de Entrada:

Ele vai usar a biblioteca Cypress, assim dentro do arquivo `pets.test.cy.js`, ele usará o código para impor os recursos dos seguintes testes.

...

Login do Cliente

- Email: nathan@gmail.com - E-mail válido de um usuário do tipo 'cliente'
- Senha: 123 - Senha correta correspondente ao e-mail acima

Edição de Pet

- Nome: Bolinho - Novo nome atribuído ao pet existente
- Espécie (type): Cachorro - Valor do campo <select>
- Raça (breed): Vira-lata - Novo valor para a raça do pet
- Idade (age): 4 - Nova idade do pet

Novo Pet

- Nome: Bolinha Editada - Nome do novo pet a ser registrado
- Espécie (type): Gato - Espécie do novo pet
- Raça (breed): Siamês - Raça do novo pet
- Idade (age): 2 - Idade do novo pet

Login do ADM

- Email: adm@gmail.com - E-mail de um usuário com permissão de administrador
- Senha: 123 - Senha do administrador

Edição de Usuário

- Nome atual: Nathan ou Allan - O teste verifica se é um desses e alterna para o outro

Pesquisa de Pet

- Nome pesquisado: Bolinha Editada - Nome do pet usado na barra de busca

Saída esperada:**Login do Cliente**

- Redirecionamento para a página de pets (/cliente/pets)

Edição do Pet

- SweetAlert com mensagem 'Pet atualizado com sucesso'

Exclusão do Pet

- SweetAlert com confirmação e pet removido da lista

Cadastro de Novo Pet

- SweetAlert com 'Pet cadastrado com sucesso' e pet visível na lista

Logout do Cliente

- Redirecionamento para tela de escolha de login (/)

Login do Administrador

- Redirecionamento para /admin/usuarios

Edição de Usuário (nome)

- SweetAlert com 'Usuário atualizado com sucesso' e campo #name alterado

Página recarregada (cy.reload)

- Alterações devem persistir e ser refletidas na UI

Busca por nome do pet

- Pet com nome 'Bolinha Editada' deve aparecer filtrado na lista

Visualização de cliente (.view)

- Exibe dados do cliente relacionado ao pet

Logout do Administrador

- Retorno para página inicial da aplicação

6.4.5 Execução e Evidências dos Testes

Testes Unitário:

TU-01:

```
1 it('deve criar um usuário com sucesso', async () => {
2   // Arrange
3   const req = {
4     body: { name: 'João', email: 'joao@email.com', password: '123456' }
5   }
6   const res = {
7     status: vi.fn().mockReturnThis(),
8     send: vi.fn()
9   }
10
11   userService.create.mockResolvedValue({ token: 'abc123' })
12
13   // Act
14   await registerUser(req, res)
15
16   // Assert
17   expect(res.status).toHaveBeenCalledWith(201)
18   expect(res.send).toHaveBeenCalledWith({ token: 'abc123' })
19 })
```

TU-02:

```
1 it('deve retornar erro quando faltam campos obrigatórios', async () => {
2   // Arrange
3   const req = {
4     body: { name: 'João' } // faltando email e password
5   }
6   const res = {
7     status: vi.fn().mockReturnThis(),
8     send: vi.fn()
9   }
10
11   // Act
12   await registerUser(req, res)
13
14   // Assert
15   expect(res.status).toHaveBeenCalledWith(400)
16   expect(res.send).toHaveBeenCalledWith({ message: "Todos campos requeridos!" })
17 })
```

TU-03:

```
1 it('deve listar todos os usuários', async () => {
2   // Arrange
3   const req = {}
4   const res = {
5     status: vi.fn().mockReturnThis(),
6     send: vi.fn()
7   }
8
9   const mockUsers = [
10    { id: 1, name: 'João', email: 'joao@email.com' },
11    { id: 2, name: 'Maria', email: 'maria@email.com' }
12  ]
13
14  userService.findAll.mockResolvedValue(mockUsers)
15
16  // Act
17  await findAllUsers(req, res)
18
19  // Assert
20  expect(res.status).toHaveBeenCalledWith(200)
21  expect(res.send).toHaveBeenCalledWith(mockUsers)
22 })
```

TU-04:

```
1 it('deve rejeitar cargo inválido', async () => {
2   // Arrange
3   const req = {
4     body: { role: 'gerente' } // cargo que não existe
5   }
6   const res = {
7     status: vi.fn().mockReturnThis(),
8     send: vi.fn()
9   }
10
11  // Act
12  await adminUpdateUser(req, res)
13
14  // Assert
15  expect(res.status).toHaveBeenCalledWith(400)
16  expect(res.send).toHaveBeenCalledWith({ message: "Cargo não existente!" })
17 })
```

TU-05:

```
1 it('deve deletar usuário com sucesso', async () => {
2   // Arrange
3   const req = {
4     userId: 1,
5     decodedId: 1, // mesmo usuário
6     params: { id: 1 }
7   }
8   const res = {
9     status: vi.fn().mockReturnThis(),
10    send: vi.fn()
11  }
12  userService.delete.mockResolvedValue({id: 1})
13
14  // Act
15  await deleteUser(req, res)
16
17  // Assert
18  expect(res.status).toHaveBeenCalledWith(200)
19  expect(res.send).toHaveBeenCalledWith({ message: "Usuario e seus pets deletados!" })
20 })
```

TU-06:

```
1 it('não deve deletar usuário com sucesso', async () => {
2   // Arrange
3   const req = {
4     userId: 1,
5     decodedId: 2, // usuário diferente
6     params: { id: 1 }
7   }
8   const res = {
9     status: vi.fn().mockReturnThis(),
10    send: vi.fn()
11  }
12  userService.delete.mockResolvedValue({id: 1})
13
14  // Act
15  await deleteUser(req, res)
16
17  // Assert
18  expect(res.status).toHaveBeenCalledWith(500)
19 })
```

Relatório dos testes Unitários:

```
DEV v3.2.2 E:/Projetos/PetsTesteSoftware/petshop-mundoanimal-api

stdout | tests/user.controller.test.js > User Controller - Testes Básicos > deve criar um
usuário com sucesso
Cargo: cliente

✓ tests/user.controller.test.js (6 tests) 10ms
  ✓ User Controller - Testes Básicos > deve criar um usuário com sucesso 6ms
  ✓ User Controller - Testes Básicos > deve retornar erro quando faltam campos obrigatóri
os 1ms
  ✓ User Controller - Testes Básicos > deve listar todos os usuários 1ms
  ✓ User Controller - Testes Básicos > deve rejeitar cargo inválido 0ms
  ✓ User Controller - Testes Básicos > deve deletar usuário com sucesso 1ms
  ✓ User Controller - Testes Básicos > não deve deletar usuário com sucesso 0ms

Test Files  1 passed (1)
Tests       6 passed (6)
Start at    01:37:01
Duration    352ms (transform 46ms, setup 0ms, collect 52ms, tests 10ms, environment 0ms,
prepare 105ms)

PASS Waiting for file changes...
press h to show help, press q to quit
```

Testes de API:

TAPI-01:

POST /api/pet/ Cria um novo pet

TAPI-02:

GET /api/user Busca todos usuarios

TAPI-03:

GET /api/pet Busca todos pets

TAPI-04:

GET /api/user/{id}/ Busca usuario pelo id

TAPI-05:

GET /api/pet/{id}/ Busca pet pelo id

Relatório dos testes de API:

TAPI-01:

Code	Details
401	Error: Unauthorized Response body <pre>Unauthorized</pre>

TAPI-02:

Code	Details
200	Response body <pre>[{ "id": "68463592f2d854b7537d316b", "name": "ADM", "email": "adm@gmail.com", "password": "\$2b\$10\$412.cQmCF9u4MJAQtVYxezdrn9Huf75nyUC0d5Wb00fN2SSAZtxS", "role": "admin", "createdAt": "2025-06-08T23:19:03.481Z", "__v": 0 }, { "id": "68465177dac2a5bfa0729b79", "name": "Nathan", "email": "nathan@gmail.com", "password": "\$2b\$10\$t7jH44Us.s/v504YKa84b..bLgEPj4EiY3qrAKefOLq9aZVjggvga", "role": "cliente", "createdAt": "2025-06-09T01:47:31.502Z", "__v": 0 }]</pre>

TAPI-03:

200	Response body <pre>[{ "id": "68463c13f2d854b7537d32ad", "name": "Daniel Baitola Viadinho de Merda", "type": "Gato", "age": 19, "breed": "Humano", "photo": "https://img.freepik.com/premium-vector/cartoon-cute-beagle-dog-with-speech-bubble_52569-2186.jpg?w=360", "user": { "id": "68463592f2d854b7537d316b", "name": "ADM", "email": "adm@gmail.com" }, "createdAt": "2025-06-08T23:19:03.468Z", "__v": 0 }, { "id": "684647e0dac2a5bfa07298ef", "name": "Feste", "type": "Gato", "age": 19, "breed": "Humano", "photo": "https://img.freepik.com/premium-vector/cartoon-cute-beagle-dog-with-speech-bubble_52569-2186.jpg?w=360", "user": { "id": "68463592f2d854b7537d316b", "name": "ADM", "email": "adm@gmail.com" } }]</pre>
-----	--

TAPI-04:

Code	Details
200	Response body <pre>{ "id": "68465177dac2a5bfa0729b79", "name": "Allan", "email": "nathan@gmail.com", "password": "\$2b\$10\$t7jH44Us.s/v504YKa84b..bLgEPj4EiY3qrAKefOLq9aZVjggvga", "role": "cliente", "createdAt": "2025-06-09T01:47:31.502Z", "__v": 0 }</pre>

TAPI-05:

Code	Details
400	Error: Bad Request Response body <pre>{ "message": "ID invalid!" }</pre>
	Response headers

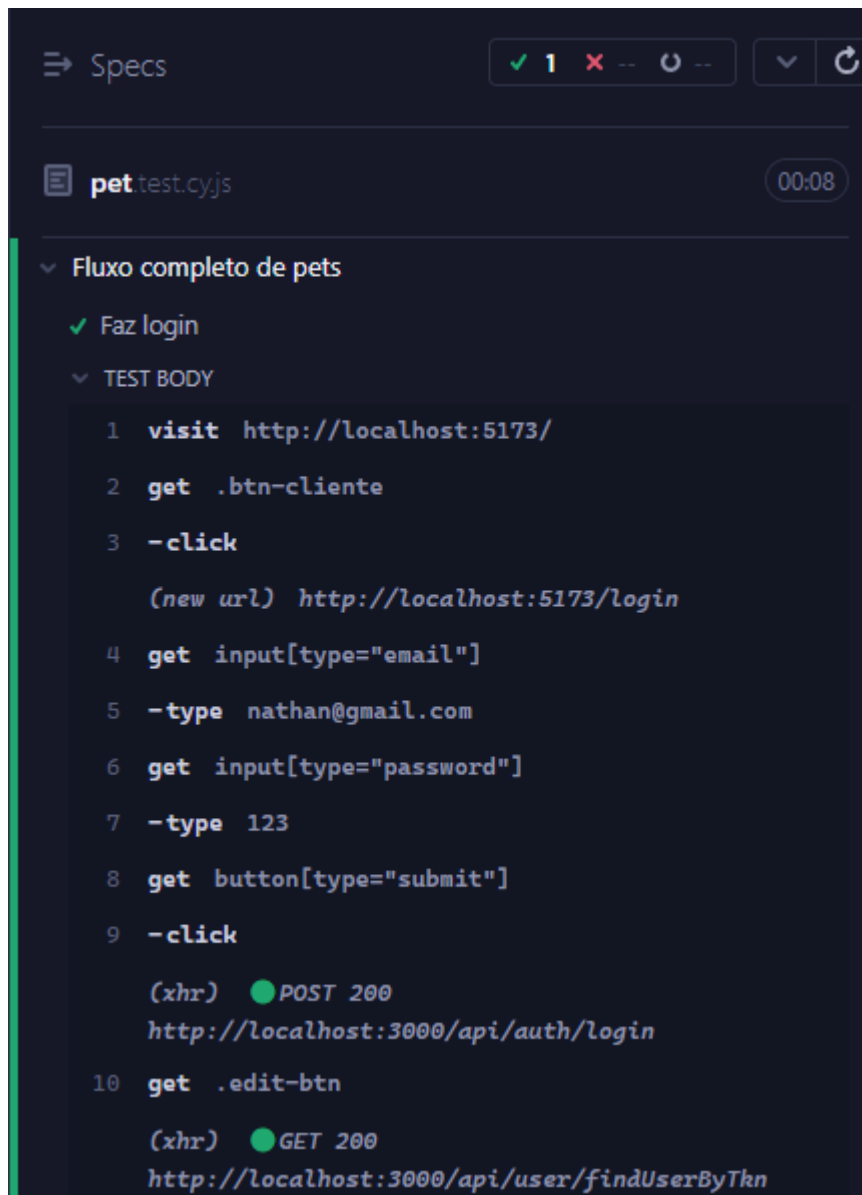
Testes de E2E:

```

1 describe('Fluxo completo de pets', () => {
2   const loginEmail = 'nathan@gmail.com';
3   const loginPassword = '123';
4
5   const loginEmailADM = 'adm@gmail.com';
6   const loginPasswordADM = '123';
7
8   const originalPet = {
9     name: 'Bolinho',
10    type: 'Cachorro',
11    breed: 'Vira-lata',
12    age: '4'
13  };
14  const updatedPet = {
15    name: 'Bolinha Editada',
16    type: 'Gato',
17    breed: 'Siames',
18    age: '2'
19  };
20
21  before(() => {
22    cy.on('window:confirm', () => true);
23  });
24
25  it('Faz login', () => {
26    cy.visit('http://localhost:5173/');
27
28    cy.get('.btn-cliente').click();
29    cy.get('input[type="email"]').type(loginEmail);
30    cy.get('input[type="password"]').type(loginPassword);
31    cy.get('button[type="submit"]').click();
32
33    cy.get('.edit-btn').click();
34    cy.get('#name').clear().type(originalPet.name);
35    cy.get('#type').select(originalPet.type);
36    cy.get('#breed').clear().type(originalPet.breed);
37    cy.get('#age').clear().type(originalPet.age);
38
39    cy.get('.submit-btn').click();
40    cy.get('.swal2-confirm').click();
41
42    cy.get('.delete-btn').click();
43    cy.get('.swal2-confirm').click();
44    cy.get('.swal2-confirm').click();
45
46    cy.get('.register-btn').click();
47    cy.get('#name').type(updatedPet.name);
48    cy.get('#type').select(updatedPet.type);
49    cy.get('#breed').type(updatedPet.breed);
50    cy.get('#age').type(updatedPet.age);
51
52    cy.get('.submit-btn').click();
53    cy.get('.swal2-confirm').click();
54
55    cy.get('.user-button').click();
56    cy.get('.logout').click();
57
58    cy.get('.btn-cliente').click();
59    cy.get('input[type="email"]').type(loginEmailADM);
60    cy.get('input[type="password"]').type(loginPasswordADM);
61    cy.get('button[type="submit"]').click();
62
63    cy.get(':nth-child(1) > input').click();
64    cy.get(':nth-child(1) > input').click();
65    cy.get('.filter-group > :nth-child(2) > input').click();
66    cy.get('.filter-group > :nth-child(2) > input').click();
67
68    cy.get('.btn-alterar').click();
69
70    cy.get('#name')
71      .invoke('val') // pega o valor atual do input
72      .then((currentValue) => {
73        if (currentValue === 'Nathan') {
74          cy.get('#name').clear().type('Allan');
75        } else {
76          cy.get('#name').clear().type('Nathan');
77        }
78      });
79
80    cy.get('.submit-btn').click();
81    cy.get('.swal2-confirm').click();
82
83    cy.reload();
84
85    cy.get('[href="/funcionario/pets"]').click();
86    cy.get('.search-input').click().type(updatedPet.name);
87
88    cy.get('.view-client-btn').click();
89    cy.get('.user-button').click();
90    cy.get('.logout').click();
91  });
92 });
93

```

Relatório de teste de E2E:



The screenshot shows the Cypress test runner interface. At the top, the 'Specs' tab is active, displaying a summary of test results: 1 passed (green checkmark), 0 failed (red X), and 0 pending (blue circle). Below this, the file 'pet.test.cy.js' is listed with a duration of 00:08. The test suite 'Fluxo completo de pets' is expanded, showing a single test 'Faz login' which has passed (green checkmark). The 'TEST BODY' for this test is visible, listing 10 steps:

- 1 **visit** `http://localhost:5173/`
- 2 **get** `.btn-cliente`
- 3 **-click**
`(new url) http://localhost:5173/login`
- 4 **get** `input[type="email"]`
- 5 **-type** `nathan@gmail.com`
- 6 **get** `input[type="password"]`
- 7 **-type** `123`
- 8 **get** `button[type="submit"]`
- 9 **-click**
`(xhr) POST 200 http://localhost:3000/api/auth/login`
- 10 **get** `.edit-btn`
`(xhr) GET 200 http://localhost:3000/api/user/findUserByTkn`

6.4.6 Link do Repositório

Link GitHub: <https://github.com/Danimell061/PetsTesteSoftware>

7 CONCLUSÃO E LIÇÕES APRENDIDAS

O desenvolvimento do sistema "Mundo Animal" representou um avanço significativo na aplicação de conceitos de engenharia de software. Ao longo do projeto, a equipe se dedicou a construir uma plataforma web robusta e funcional, que atendesse às necessidades de diferentes perfis de usuários, desde tutores de pets a administradores do sistema.

O objetivo geral de desenvolver um sistema web para o gerenciamento centralizado e eficiente de informações de animais de estimação foi plenamente alcançado. A plataforma desenvolvida permite que usuários (tutores) cadastrem e administrem os dados de seus pets, enquanto funcionários e um administrador com privilégios elevados gerenciam o sistema de forma mais ampla, garantindo a organização e a integridade dos dados.

Os objetivos específicos também foram atendidos com sucesso. Foi implementado um sistema de autenticação seguro com três níveis de acesso distintos (Usuário, Funcionário e Admin), conforme planejado. Usuários finais obtiveram a autonomia necessária para realizar o ciclo completo de operações CRUD (criar, ler, atualizar e deletar) em seus próprios animais. O perfil de Funcionário foi capacitado com permissões para visualizar e gerenciar todos os pets e usuários da plataforma. Por fim, o super usuário (Admin) recebeu controle total sobre todas as entidades do sistema, incluindo o gerenciamento de contas de usuários e funcionários, consolidando a hierarquia de permissões.

Os resultados esperados com a implantação do sistema foram em grande parte atingidos. A plataforma promove a centralização e organização das informações dos pets, superando os desafios do controle manual. Os usuários ganharam autonomia na gestão dos dados de seus animais, e a administração adquiriu um controle mais efetivo sobre os registros. A digitalização dos processos, aliada às ferramentas de testes, tende a reduzir erros operacionais e a facilitar o acesso rápido à informação.

Como principal lição aprendida, destaca-se a importância de uma análise de requisitos bem definida e da modelagem de negócios para o sucesso do desenvolvimento. A clara definição das regras de negócio e dos requisitos funcionais e não-funcionais serviu como um guia sólido durante toda a fase de implementação. A utilização de ferramentas modernas como React para o frontend, Node.js para o backend e MongoDB como banco de dados demonstrou ser uma escolha certa. O projeto reforçou a compreensão de que a construção de um software de qualidade vai além da codificação, envolvendo planejamento, documentação e testes sistemáticos para garantir que a solução final agregue valor real aos seus usuários.

REFERÊNCIAS

SWEETALERT2. SweetAlert2: a beautiful, responsive, customizable, accessible (WAI-ARIA) replacement for JavaScript's popup boxes. [S. l.], 2025. Disponível em: <https://sweetalert2.github.io/>. Acesso em: 8 jun. 2025.

BCRYPT. bcrypt. [S. l.]: npm, Inc., 2025. Disponível em: <https://www.npmjs.com/package/bcrypt>. Acesso em: 8 jun. 2025.

TAILWIND CSS. Tailwind CSS: Rapidly build modern websites without ever leaving your HTML. [S. l.]: Tailwind Labs, 2025. Disponível em: <https://tailwindcss.com/>. Acesso em: 8 jun. 2025.

AXIOS. Axios Docs. [S. l.], 2025. Disponível em: <https://axios-http.com/ptbr/docs/intro>. Acesso em: 8 jun. 2025.

JS-COOKIE. js-cookie. [S. l.]: npm, Inc., 2025. Disponível em: <https://www.npmjs.com/package/js-cookie>. Acesso em: 8 jun. 2025.

SWAGGER-UI-EXPRESS. swagger-ui-express. [S. l.]: npm, Inc., 2025. Disponível em: <https://www.npmjs.com/package/swagger-ui-express>. Acesso em: 8 jun. 2025.

REACT. Learn React. [S. l.]: Meta, 2024. Disponível em: <https://react.dev/learn>. Acesso em: 07 junho 2025.

CYPRESS. Cypress: JavaScript E2E Testing Framework. [S. l.]: Cypress.io, 2025. Disponível em: <https://www.cypress.io/>. Acesso em: 9 jun. 2025.

VITEST. Vitest: a Vite-native unit test framework. [S. l.], 2025. Disponível em: <https://vitest.dev/>. Acesso em: 9 jun. 2025.