UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO MULTIDISCIPLINAR DE PAU DOS FERROS
CURSO DE BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

ANTÔNIO ROMAILSON FERNANDES COSTA

PROJETO DE ARQUITETURA DE UM SISTEMA PARA ADOÇÃO DE ANIMAIS RESGATADOS PELO GRUPO UNIDOS PELAS PATINHAS EM PAU DOS FERROS/RN E REGIÃO

PAU DOS FERROS - RN
2024

ANTÔNIO ROMAILSON FERNANDES COSTA

PROJETO DE ARQUITETURA DE UM SISTEMA PARA ADOÇÃO DE ANIMAIS RESGATADOS PELO GRUPO UNIDOS PELAS PATINHAS EM PAU DOS FERROS/RN E REGIÃO

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito parcial para obtenção do título de Bacharel em Tecnologia da Informação.

Orientador: Prof. Dr. Reudismam Rolim de Sousa

PAU DOS FERROS - RN

2024

C837p Costa, Antônio Romailson Fernandes.
Projeto de arquitetura de um sistema para adoção de animais resgatados pelo grupo unidos pelas patinhas em Pau do Ferros/RN e região / Antônio Romailson Fernandes Costa. - 2024.
56 f. : il.

Orientador: Reudismam Rolim de Sousa.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Tecnologia da Informação, 2024.

1. Adoção de animais. 2. Arquitetura de sistema. 3. Linguagem UML. 4. Grupo unidos pelas patinhas.. I. Sousa, Reudismam Rolim de, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automáto em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

ANTÔNIO ROMAILSON FERNANDES COSTA

PROJETO DE ARQUITETURA DE UM SISTEMA PARA ADOÇÃO DE ANIMAIS RESGATADOS PELO GRUPO UNIDOS PELAS PATINHAS EM PAU DOS FERROS/RN E REGIÃO

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito parcial para obtenção do título de Bacharel em Tecnologia da Informação.

Defendida em: **27/03/2025**.

BANCA EXAMINADORA

_____
Reudismam Rolim de Sousa, Prof. Dr. - (UFERSA)
Presidente

_____
Hortência Pessoa Rêgo Gomes, Profa. Me. - (UFERSA)
Membro Examinador

_____
Huliane Medeiros da Silva, Profa. Dra. - (UFERSA)
Membro Examinador

Dedico este trabalho à minha família, especialmente à minha mãe, Antonia Irineuda, meu pai, Antonio Francisco, minha namorada, Deyse Daiane e aos meus irmãos, Maria Aparecida, Francisca Adriana, José Roberto, Robério Fernandes e Emanoel Rodrigo. Eles nunca duvidaram da minha capacidade, sempre me apoiaram e estiveram ao meu lado nas fases mais desafiadoras desta jornada e continuam a fazer isso.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer a Deus, a principal base de princípios ao qual cresci e fui ensinado a respeitar e clamar em momentos de dificuldade; agradecer por ser o primeiro de minha família a se formar em uma universidade federal a uma longa distância de casa e que desde a minha chegada eu nunca me senti só de verdade.

De maneira a faltar palavras, sou grato pelos meus pais, que desde o início nunca disseram nenhuma palavra de desânimo ou fraqueza; a minha mãe Antônia Irineuda, que diante de dificuldades financeiras, problemas de saúde, e situações diversas mostraram que eu tenho a base forte fundamentada no verdadeiro valor de uma família. Foram nesses momentos meus conselheiros, cuidadores e sempre me motivaram a seguir em frente.

Agradeço aos meus irmãos que sempre estiveram dispostos a me ajudar sem pedir nada em troca, sempre me apoiando em tudo que eu precisei, seja me dando conselhos.

Agradeço aqui, especialmente, a minha namorada, pelo incentivo contínuo, pelas palavras de apoio que nunca me deixou desanimar em momentos difíceis. Logo, sou muito grato a ela: Deyse Daiane.

Agora neste agradecimento não conterá laço de sangue, mas um forte laço de amizade, respeito, companheirismo, que cresceu em meio a dificuldades e desafios, mas que até hoje nos deu frutos; aqui cito Emerson Igor, Maria Eduarda, Giselia Kaline. Todos eles de maneira direta fizeram por mim coisas que nem todo amigo, colega ou parceiro faz; cada um do seu jeito, do seu próprio modo e com suas próprias capacidades.

Agradeço aos meus professores, mas tem alguns que o meu agradecimento transcende os muros da universidade, principalmente ao Prof.Dr. Reudismam, seu exemplo como pessoa, carisma e dedicação serão lembrados e ficarão gravados na minha essência de universitário.

Agradeço a todo corpo docente da UFERSA Campus Pau dos Ferros. Obrigado!

Agradeço ao meu orientador, o Prof. Dr. Reudismam, que sempre esteve à disposição para me ouvir e me orientar em cada passo dado dentro da universidade e me guiou para a entrega deste trabalho de conclusão de curso.

Agradecemos aos grupos LIS — Laboratório de Inovações em Software — e LISA — Laboratório de Inovações em Software e Automação — pelo apoio neste trabalho, e à UFERSA pelo financiamento, por meio da Pró-Reitoria de Pesquisa e Pós-Graduação (PROPPG), através do Edital PROPPG Nº 22/2024 e PROPPG Nº 21/2024.

Por fim, agradeço à minha banca examinadora. Obrigado!

RESUMO

O aumento significativo no abandono de animais errantes tem exigido ações mais eficazes das ONGs que atuam nessa área. O grupo Unidos pelas Patinhas, atuante em Pau dos Ferros/RN, enfrenta desafios crescentes na gestão de informações relacionadas aos processos de adoção, monitoramento e bem-estar dos animais resgatados, o que demanda ações para lidar com esses dilemas. Buscando minimizar os desafios, Santos et al. (2012) apresentou um app *mobile* para adoção de animais resgatados pelo grupo Unidos pelas Patinhas. O aplicativo encontra-se com a parte *front-end* desenvolvida. No entanto, é necessário o desenvolvimento da parte *back-end* para lidar com as necessidades do grupo. Neste contexto, um dos elementos importantes para isso é o desenvolvimento da arquitetura do ambiente. Neste sentido, este trabalho propõe o desenvolvimento de uma arquitetura de sistema que auxilie no gerenciamento desses processos, visando otimizar a adoção de animais e fortalecer a transparência das ações do grupo. A arquitetura é organizada em termos de visões arquitetônicas, que representam os módulos, os componentes e suas conexões e detalhes de alocação. Para avaliar a arquitetura, foi utilizada uma abordagem baseada em cenários, buscando demonstrar que os elementos projetados atendem às especificações do ambiente proposto. Os resultados demonstram que as visões implementadas atendem aos cenários analisados.

Palavras-chave: adoção de animais; arquitetura de sistema; linguagem UML; grupo unidos pelas patinhas.

ABSTRACT

The significant increase in the abandonment of stray animals has required more effective actions from NGOs working in this area. The Unidos pelas Patinhas group, operating in Pau dos Ferros/RN, faces increasing challenges in managing information related to the adoption, monitoring, and welfare processes of rescued animals, which demands actions to deal with these dilemmas. Seeking to minimize the challenges, Santos et al. (2012) presented a mobile app for the adoption of animals rescued by the Unidos pelas Patinhas group. The application has its front-end part developed. However, the back-end part needs to be developed to deal with the group's needs. In this context, one of the important elements for this is the development of the environment's architecture. In this sense, this work proposes the development of a system architecture that helps in the management of these processes, aiming to optimize the adoption of animals and strengthen the transparency of the group's actions. The architecture is organized in terms of architectural views, which represent the modules, components, and their connections and allocation details. To evaluate the architecture, a scenario-based approach was used, seeking to demonstrate that the designed elements meet the specifications of the proposed environment. The results demonstrate that the implemented views meet the analyzed scenarios.

Keywords: Animal Adoption, System Architecture, Language UML, United by Paws group.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

UML	*Unified Modeling Language*
SRP	*Princípio da Responsabilidade Única*
OCP	*Princípio Aberto/Fechado*
LSP	*Princípio da Substituição de Liskov*
ISP	*Princípio da Segregação de Interface*
DIP	*Princípio da Inversão de Dependência*
ATAM	*Architecture Tradeoff Analysis Method*

SUMÁRIO

1 INTRODUÇÃO

A adoção de animais abandonados é um tema de grande relevância social no Brasil. Muitos municípios enfrentam dificuldades no controle populacional de cães e gatos errantes, o que gera impactos significativos na saúde pública e no bem-estar animal. A superpopulação desses animais contribui para a disseminação de zoonoses, como a leishmaniose e a raiva, além de representar um risco para a segurança urbana (Silva et al., 2018). De acordo com a Organização Mundial da Saúde (OMS), estima-se que existam mais de 30 milhões de animais abandonados no país, sendo aproximadamente 20 milhões de cães e 10 milhões de gatos (Jusbrasil, 2023). Neste contexto, as Organizações Não Governamentais (ONGs) e protetores independentes desempenham um papel crucial, oferecendo cuidados e promovendo adoções.

Entretanto, esses voluntários enfrentam desafios para gerenciar seus processos internos, como a divulgação de campanhas, organização de dados dos animais e interação com adotantes. Essa lacuna pode ser minimizada por meio de soluções tecnológicas que otimizem essas tarefas, promovam maior eficiência e ampliem o alcance das campanhas de adoção, conectando os protetores a um número maior de possíveis adotantes. De acordo com Tozzi, Anderle e Nogueira (2018), o uso de tecnologias digitais tem se mostrado uma ferramenta essencial para a divulgação de eventos de adoção e para a localização de animais perdidos ou disponíveis para adoção, simplificando o processo e promovendo adoções mais responsáveis.

Para auxiliar a minimizar essa problemática, Santos et al. (2012) desenvolveram um aplicativo para o referido grupo. O aplicativo focou na parte *front-end* do ambiente; no entanto, para atender às necessidades da aplicação, é preciso o desenvolvimento do *back-end* para tratar as funcionalidades e regras de negócio requeridas para o sistema. Neste aspecto, um dos elementos principais para guiar o desenvolvimento do *back-end* da aplicação é a arquitetura do software.

Diante desse contexto, a proposta deste trabalho é realizar uma análise e desenvolvimento de uma arquitetura de um sistema para a gestão de adoção de animais, baseada em um estudo de caso do grupo Unidos pelas Patinhas, com atuação em Pau dos Ferros/RN e região. O objetivo é facilitar a organização das informações e proporcionar uma experiência simplificada tanto para os administradores quanto para os adotantes.

1.1 GRUPO UNIDOS PELAS PATINHAS

O grupo Unidos pelas Patinhas é uma organização que atua na cidade de Pau dos

Ferros/RN e região, desempenhando um papel fundamental na proteção e adoção de animais resgatados e abandonados. Seu trabalho envolve o resgate, cuidado e encaminhamento responsável desses animais para novos lares. Além disso, o grupo realiza campanhas de conscientização sobre posse responsável e castração, buscando reduzir o número de animais em situação de rua e melhorar a qualidade de vida dos resgatados. Atualmente, usam a ferramenta “instagram” para fazer o controle das adoções e doações. A entidade enfrenta desafios relacionados à organização dos dados dos animais, ao gerenciamento de adoções e à comunicação com possíveis adotantes, tornando essencial a implementação de um sistema que otimize esses processos.

1.2 JUSTIFICATIVA

A adoção de animais abandonados é um tema de grande relevância social no Brasil. Muitos municípios enfrentam dificuldades no controle populacional de cães e gatos em situação de rua, o que gera impactos significativos na saúde pública e no bem-estar animal. A superpopulação desses animais contribui para a disseminação de zoonoses, como a leishmaniose e a raiva, além de representar um risco para a segurança urbana (Silva et al., 2018). Para enfrentar esse desafio, foi proposto o desenvolvimento de um sistema de gestão adaptado às necessidades do grupo Unidos pelas Patinhas, atuante em Pau dos Ferros/RN e região, conforme apontado por Santos et al. (2022). Essa iniciativa visa otimizar processos internos, ampliar o alcance de campanhas de adoção e promover o bem-estar animal de maneira eficiente. A seguir, destacam-se os principais benefícios dessa implementação:

- **Centralização de Informações e Cadastro de Animais:** O sistema permitirá que o grupo centralize o cadastro de animais resgatados, registrando informações relevantes sobre sua saúde, histórico e estado atual. Essa funcionalidade elimina a necessidade de métodos manuais, tornando o gerenciamento mais organizado e acessível.
- **Eficiência na Gestão de Campanhas e Processos Internos:** Com o uso de ferramentas digitais, o grupo poderá gerenciar suas campanhas de adoção de forma integrada, ampliando o alcance e a eficácia das estratégias de divulgação. Além disso, a automação de tarefas como o cadastro de adotantes e o acompanhamento de processos de adoção proporcionam maior eficiência.
- **Transparência e Engajamento com a Sociedade:** O sistema também fortalecerá a transparência das ações do grupo, permitindo que adotantes e

colaboradores acompanhem os processos de adoção e o impacto das campanhas realizadas. Essa abordagem incentiva o engajamento social e aumenta a credibilidade do grupo.

Portanto, o desenvolvimento de um sistema de gestão para o grupo Unidos pelas Patinhas é essencial para promover a eficiência e modernização de seus processos internos. A iniciativa não só atenderá às demandas específicas do grupo, como também contribuirá para o bem-estar animal e o impacto social na região, oferecendo uma solução tecnológica adaptada e sustentável.

1.3 OBJETIVOS

Este projeto visa desenvolver a arquitetura de software do grupo Unidos pelas Patinhas com o intuito de otimizar o gerenciamento de campanhas de adoção e promover uma comunicação mais eficiente com potenciais adotantes. A plataforma busca oferecer uma experiência centralizada e acessível, permitindo o cadastro e acompanhamento de animais resgatados, a divulgação de campanhas e a interação com adotantes de forma prática e transparente. Além disso, o sistema incluirá funcionalidades que facilitem o registro de adotantes e o acompanhamento dos processos de adoção, criando um ambiente que promova o bem estar animal e fortaleça o impacto social da organização.

Para alcançar o objetivo geral descrito anteriormente, foram definidos os seguintes objetivos específicos:

- Realizar levantamento dos requisitos funcionais e não funcionais para o sistema, incluindo cadastro de animais, gerenciamento de adoção e campanhas.
- Elaborar documentos técnicos utilizando UML, como diagramas de casos de uso, classes e sequência, para descrever as funcionalidades e a estrutura do sistema.
- Desenvolver protótipos de interface amigável, priorizando a acessibilidade e a usabilidade para diferentes tipos de usuários.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado da seguinte forma: no Capítulo 1, é apresentado a introdução do trabalho, mostrando o problema relevante enfrentado pelo grupo Unidos Pelas Patinhas, a justificativa do porque o trabalho é importante e os objetivos para alcançar os resultados positivos, no Capítulo 2, é apresentado o embasamento teórico e alguns trabalhos relacionados a esta pesquisa; no Capítulo 3 é apresentada a proposta de solução e os resultados alcançados; no Capítulo 4, são expostas as conclusões obtidas.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão discutidos alguns conceitos e termos importantes para o entendimento deste trabalho.

2.1 ABANDONO DE ANIMAIS

A relação entre humanos e animais, que existe há milênios, tem se tornado ainda mais próxima devido a mudanças na estrutura social, como a redução no tamanho das famílias e a busca por companhia e afeto (Moutinho; Serra; Valente, 2019). No entanto, o problema do abandono de animais continua crescendo e traz impactos diretos no bem-estar animal e na saúde pública. Dados apresentados por Santos et al. (2022) revelam que apenas uma pequena parcela dos animais resgatados consegue um novo lar, o que evidencia a necessidade de estratégias mais eficazes para campanhas de adoção.

A Organização Mundial da Saúde (OMS) estimou que, em 2022, cerca de 30 milhões de animais estavam em situação de abandono no Brasil, com 10 milhões de gatos e 20 milhões de cães. Isso representa não apenas um sofrimento para os animais, mas também riscos à saúde pública, uma vez que esses animais podem transmitir zoonoses como leishmaniose e raiva (Organização Mundial da Saúde, 2022).

Neste contexto, as ONGs e os protetores independentes, como o Unidos pelas Patinhas, desempenham um papel crucial no enfrentamento desse problema. Elas resgatam, cuidam e promovem a adoção responsável, além de conscientizar a sociedade sobre a importância de práticas como a castração e o não abandono. No entanto, essas organizações frequentemente enfrentam desafios relacionados à falta de recursos e à gestão de dados sobre os animais resgatados e adotados. Isso reforça a necessidade de sistemas tecnológicos que auxiliem na otimização dessas atividades, como o proposto neste trabalho.

2.2 ARQUITETURA DE SOFTWARE

A arquitetura de software é um elemento fundamental no desenvolvimento de sistemas computacionais. Ela define a estrutura organizacional do sistema, especificando os componentes principais, suas interações, e as restrições tecnológicas que afetam o *design*. Segundo Bass, Clements e Kazman (2012), a arquitetura de *software* é responsável por garantir que o sistema atenda aos seus requisitos de qualidade, como desempenho, escalabilidade e segurança.

Uma boa arquitetura de *software* facilita a comunicação entre as partes interessadas, permite a reutilização de componentes e simplifica a manutenção ao longo do ciclo de vida do sistema. Estilos arquiteturais, como arquiteturas em camadas, microsserviços e arquitetura orientada a eventos, são amplamente utilizados para atender a requisitos específicos de diferentes tipos de sistemas. Com base no livro de Bass, Clements e Kazman (2021), a arquitetura em camadas organiza o aplicativo em grupos de funcionalidades relacionadas, promovendo a separação de responsabilidades e facilitando a manutenção. Segundo Richards e Ford (2020), a arquitetura de microsserviços permite que os aplicativos sejam compostos por serviços independentes, cada um responsável por uma funcionalidade específica, facilitando a escalabilidade e a implantação independente. Além disso, de acordo com Fowler (2002), a arquitetura orientada a eventos é projetada para detectar eventos significativos e responder a eles de forma assíncrona, tornando os sistemas mais responsivos e escaláveis.

Para sistemas voltados à gestão, como o proposto neste trabalho, uma abordagem arquitetural em camadas é comumente empregada, dividindo o sistema em uma camada de apresentação (*front-end*), uma camada de lógica de negócios (*back-end*) e uma camada de dados (banco de dados). Cada camada tem responsabilidades bem definidas, permitindo maior modularidade e facilidade de manutenção (Sommerville, 2011).

2.3 PRINCÍPIOS ARQUITETURAIS

Bass, Clements e Kazman (2012) destacam que a arquitetura de *software* deve ser orientada pelas qualidades desejadas do sistema, como desempenho, segurança e usabilidade. Esses atributos de qualidade são cruciais nas decisões de *design* e estrutura do sistema, influenciando diretamente a forma como o *software* é construído e mantido. Além disso, os autores enfatizam a importância dos estilos arquitetônicos, como camadas e microsserviços, que apresentam vantagens e desvantagens específicas. A escolha adequada do estilo arquitetônico não só facilita a manutenção e a evolução do sistema, mas também garante que ele atenda aos requisitos técnicos e agregue valor estratégico à organização. Assim, os princípios arquiteturais fornecem uma base sólida para o desenvolvimento de sistemas robustos, escaláveis e eficientes.

Estilos Arquitetônicos Adotados

Para facilitar a evolução e manutenção do sistema, foi escolhido a arquitetura em camadas, que organiza o sistema em módulos com funções bem definidas:

1. Camada de Apresentação: Gerencia a interação com o usuário, como interfaces gráficas e validações básicas.
2. Camada de Negócios: Centraliza a lógica principal do sistema, incluindo funcionalidades como cadastro e busca de animais.
3. Camada de Persistência: lida com o armazenamento e recuperação de dados, seguindo práticas que garantem consistência e segurança.

Aplicando os princípios SOLID

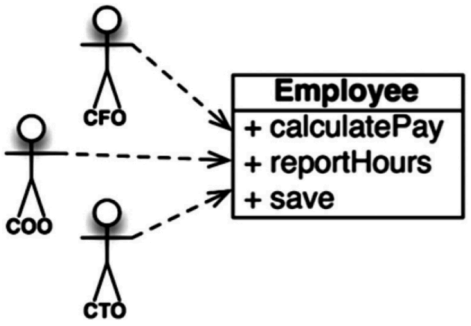
Os princípios SOLID, descritos por Martin (2019), foram aplicados para garantir um *design* flexível, modular e preparado para futuras expansões. A seguir, apresenta-se cada um dos princípios e exemplos práticos de sua aplicação.

Princípio da Responsabilidade Única (SRP): Uma classe deve ter apenas um motivo para mudar, ou seja, deve possuir uma única responsabilidade. Esse princípio evita que uma classe tenha múltiplas funções, o que pode dificultar sua manutenção e evolução.

Exemplo prático do SRP:

Para exemplificar, na Figura 1 é apresentada a classe *Employee*, que possui três métodos:

Figura 1: A classe *Employee*

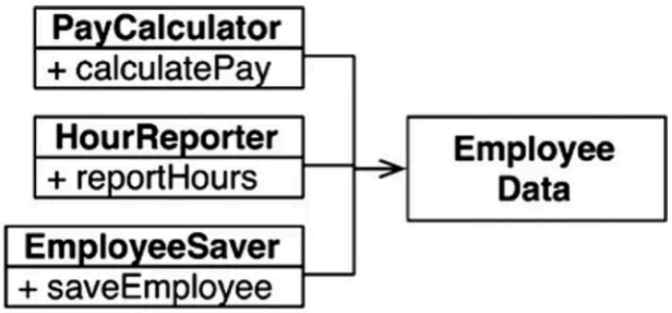


Fonte: Martin (2019)

Conforme Martin (2019), essa classe viola o SRP porque possui métodos associados a diferentes atores. Por exemplo, *calculatePay()* está relacionado ao Setor de Contabilidade, *reportHours()* ao Setor de Recursos Humanos e *save()* associado à Administração de Banco de Dados.

Para resolver esse problema, pode-se separar os diferentes métodos em classes específicas, relacionadas a cada setor, conforme apresentado na Figura 2. Todas essas classes podem fazer uso da estrutura *EmployeeData,* que armazena os dados do empregado, para realizar as suas operações.

Figura 2: As três classes não sabem uma das outras



Fonte: Martin (2019)

Princípio Aberto/Fechado (OCP): Entidades de *software* devem ser abertas para extensão, mas fechadas para modificações. Isso significa que devemos projetar nosso código de forma que novas funcionalidades possam ser adicionadas sem alterar o código existente.

Exemplo prático do OCP:

Martin (2019) utiliza como exemplo do princípio do Aberto/Fechado (OCP), um ambiente que oferece um resumo financeiro, que pode ser exibido na *Web* ou impresso. Ao ser impresso, o relatório deve destacar os números negativos entre parênteses. No resumo para a Web, os dados devem ser apresentados de forma padrão e os números negativos destacados em vermelho

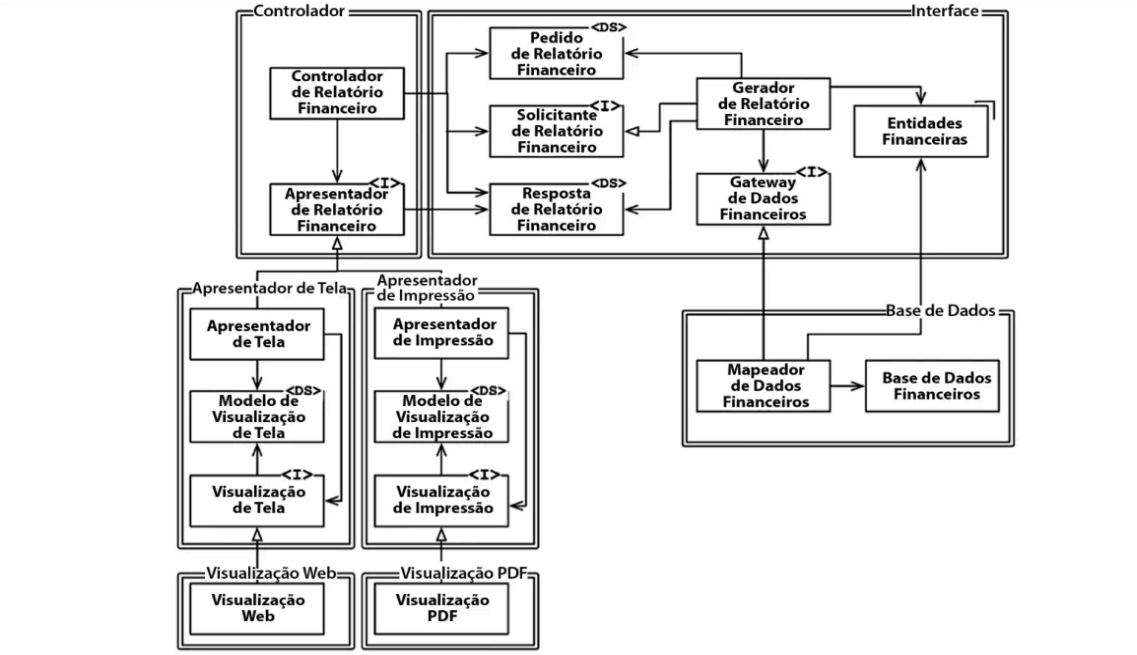
O procedimento pode ser organizado em diferentes responsabilidades: 1) fazer a análise financeira, 2) gerar dados para o relatório e 3) exibir relatório em diferentes formatos (e.g., *Web* e impresso), conforme pode ser visualizado na Figura 3.

Figura 3: Aplicando o SRP

Fonte: Martin (2019)

Uma vez que as responsabilidades foram separadas, o código pode ser dividido em classes específicas, conforme pode ser visto na Figura 4. Considerando que as partes responsáveis pelos resumos financeiros são separados em diferentes classes/componentes, o código pode ser estendido para os apresentar em novos formatos, sem afetar o código original.

Figura 4: Particionando os processos em classes e separando as classes em componentes



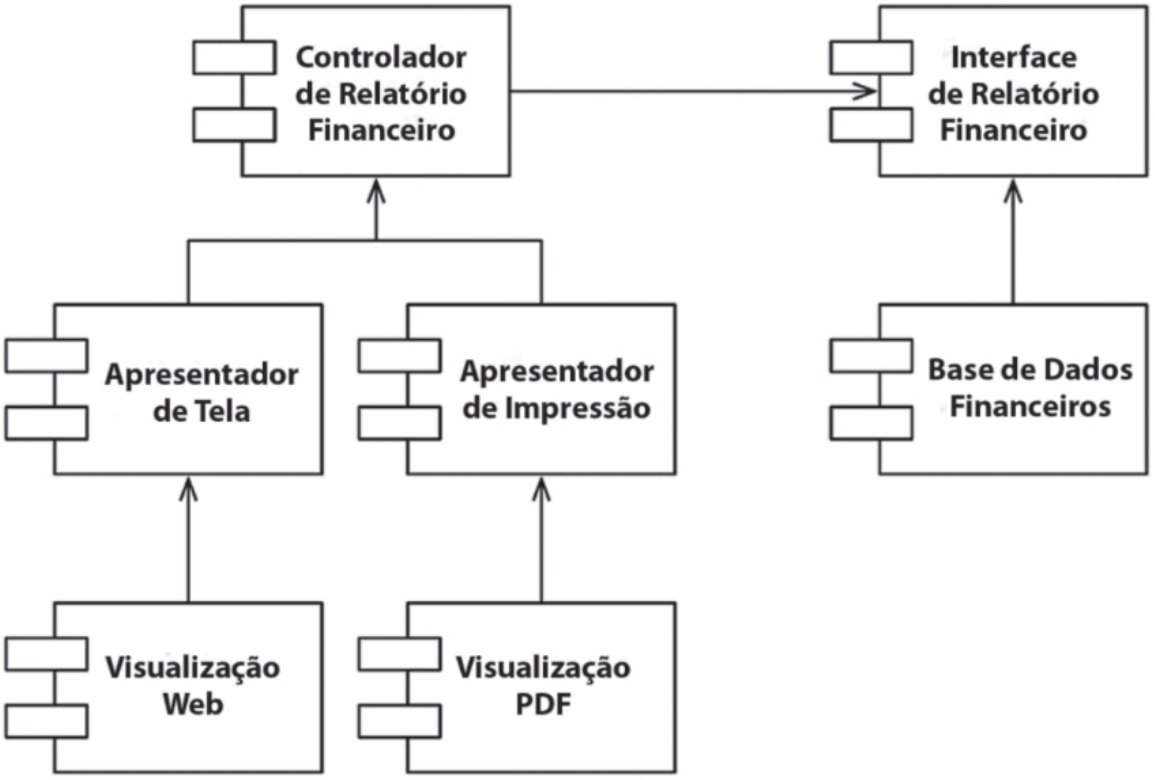
Fonte: Martin (2019)

Na Figura 4, as classes marcadas com <I> são interfaces; por sua vez, as marcadas com <DS> são estruturas de dados.

Nesta visualização, a dependência entre as classes é representada pelas setas de uma classe A para uma classe B, indicando que a classe A possui uma referência para a classe B. No entanto, a classe B, não possui nenhuma referência (desconhece) a classe A. Por exemplo, a classe Controlador de Relatório Financeiro, conhece a Estrutura de Dados Pedido de Relatório Financeiro (o inverso não é verdadeiro).

Os relacionamentos na imagem mostram que as linhas duplas são cruzadas em apenas um sentido (são unidirecionais), implicando para a direção das classes que não devem ser alteradas pela extensão do código. Na Figura 5, é possível ver esses relacionamentos em formato de componentes, trazendo uma configuração que isola as partes mais estáveis, o que impede que mudanças em partes menos críticas afetem as partes principais do código.

Figura 5: Os relacionamentos dos componentes são unidirecionais



Fonte: Martin (2019)

Da forma como o código está organizado, se houver mudanças na base de dados, no controlador, nos apresentadores ou nas visualizações, não afetará quem depende desses

elementos, uma vez que a dependência é estabelecida por meio de uma interface, permitindo que o sistema se torne estável e fácil de expandir.

Princípio da Substituição de Liskov (LSP):

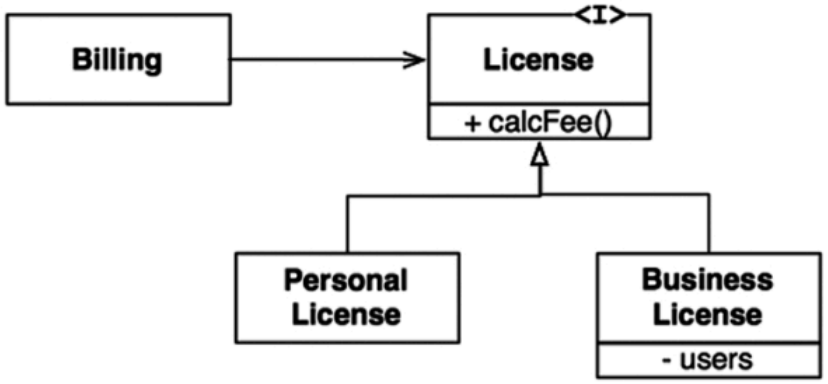
Objetos de uma classe derivada devem poder ser substituídos por objetos da classe base sem afetar a corretude do programa.

Exemplo prático do LSP

Martin (2019) exemplifica o LSP pela implementação de uma funcionalidade para o uso de uma licença (Figura 6). Nele é apresentada a classe *License*, que calcula a taxa para o uso do sistema, pelo uso do método *calcFee()*.

Na aplicação exemplo, há dois tipos de licenças: *PersonalLicense* e *BusinessLicense.* Apesar da implementação para o cálculo da taxa divergir entre os tipos de licenças, elas devem ser usadas de forma a não requerer modificação no *software* para tratar um tipo específico de licença. Neste sentido, a classe *License* pode ser substituída por qualquer um dos subtipos de licenças, sem impactar a aplicação.

Figura 6: *License* e suas derivadas, de acordo com o LSP



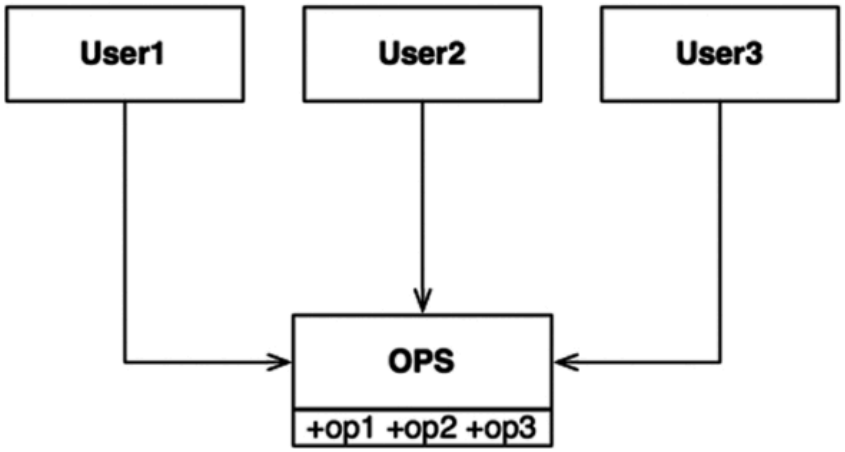
Fonte: Martin (2019)

Princípio da Segregação de Interfaces (ISP): Uma classe não deve ser forçada a implementar interfaces que não utiliza.

Exemplo prático do ISP:

Matin (2019) usa como exemplo do ISP um ambiente com três usuários, que dependem de uma classe que implementa três operações (*op1()*, *op2()* e *op3()*). No exemplo, *User1* utiliza apenas *op1()*, *User2* utiliza apenas o *op2()*, e assim por diante. Nesta implementação, todos os usuários são obrigados a depender das implementações dos três métodos, apesar de usarem apenas um deles, conforme pode ser visto na Figura 7.

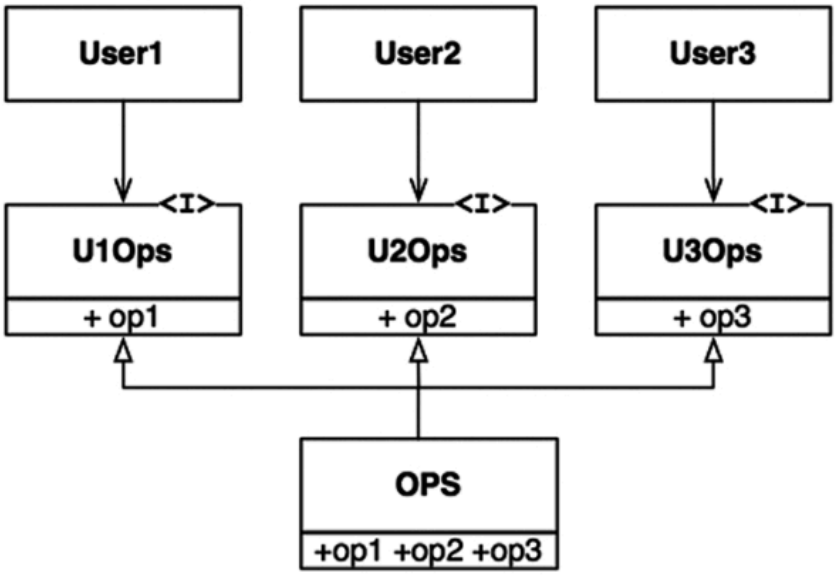
Figura 7: Princípio da Segregação de Interface



Fonte: Martin (2019).

Aplicando o ISP, as operações são divididas em interfaces específicas, fazendo com que os usuários dependam apenas das operações associadas a eles (Figura 8), evitando dependência desnecessárias no *software*.

Figura 8: Operações segregadas



Fonte: Martin (2019)

Princípio da Inversão de Dependência (DIP): Módulos de alto nível não devem depender de módulos de baixo nível; ambos devem depender de abstrações. Na maioria das linguagens orientadas a objetos, como Java, usamos uma Abstract Factory (*Fábrica Abstrata*) para lidar com essa dependência indesejada.

Exemplo prático do DIP:

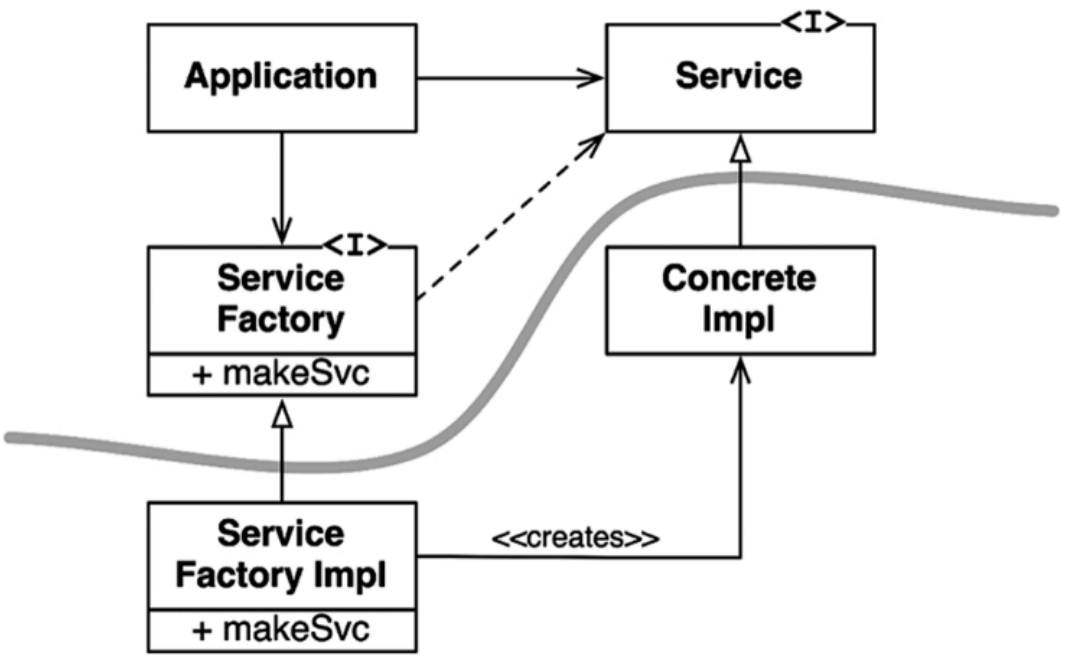
Martin (2019) utiliza como exemplo do DIP, uma classe *Application*, que precisa criar uma implementação concreta (*ConcreteImpl*). Uma vez que criar uma classe diretamente deixa o código em desacordo com o princípio, a implementação concreta é criada por uma interface *ServiceFactory*. Ademais, ao invés de lidar diretamente com a classe *ConcreteImpl*, a classe *Application* possui uma dependência da classe *Service*, conforme pode ser visto na Figura 9.

Essa organização do código divide o sistema em dois componentes, sendo um deles um componente concreto e o outro um componente abstrato, facilitando a manutenção do sistema, uma vez que as alterações nas classes concretas não afetam diretamente as classes que as empregam.

Conforme pode ser visto na Figura 9, o DIP é quebrado em algum momento. Por

exemplo, a classe *ServiceFactoryImpl* não está de acordo com o princípio porque cria a classe *ConcreteImpl*. Nos sistemas, uma forma comum de organização é a criação de uma classe *Main*, que cria uma instância da fábrica concreta e deixa uma instância global disponível do tipo *ServiceFactory*. Dessa forma, quando alguma parte do código precisar de um objeto concreto pode pegá-lo diretamente a partir dessa variável global (Martin, 2019).

Figura 9: Uso do padrão *Abstract Factory* para lidar com a dependência



Fonte: Martin (2019)

2.4 VISÕES ARQUITETURAIS

Bass, Clements e Kazman (2012) apresenta as visões arquiteturais para representar diferentes aspectos da arquitetura. As visões possibilitam visualizar a arquitetura sob diferentes perspectivas. Nesta modelagem, a arquitetura é dividida em três visões. Visão de módulo, que apresenta a organização estrutural dos elementos de *software*; a visão componente & conector, que organizar os elementos estruturais em componentes reutilizáveis, permitindo que se tenha uma visão desses elementos em tempo de execução; e, a visão de alocação, que permite visualizar como o *software* será implantado e que recursos serão alocados a eles.

VISÃO DE MÓDULO

A visão de módulos permite representar o sistema em termos de estruturas de código ou unidades de dados, como destacado por Bass, Clements e Kazman (2012). Essa abordagem facilita a compreensão das responsabilidades de cada elemento do código (classes, camadas, entre outros), enfatizando como o código está organizado de forma estática, sem considerar os relacionamentos em tempo de execução. Com isso, é possível responder a diversas questões, como: quais são as funcionalidades de cada módulo, quais unidades de código os módulos dependem e interagem, e se há relações de herança entre essas unidades (Bass, Clements e Kazman, 2012).

VISÃO COMPONENTE & CONECTOR

A visão de componente & conector permite representar o ambiente do sistema em termos de componentes, que correspondem a elementos com comportamentos observáveis em tempo de execução (como serviços, clientes, servidores, etc.), e conectores, que descrevem as interações entre esses elementos (como troca de mensagens, chamadas de métodos, etc.) (Bass, Clements e Kazman, 2012). Essa perspectiva facilita responder a questões importantes, como: existem dados compartilhados entre os componentes? Quais componentes podem executar em paralelo? Como os dados fluem ao longo do sistema? (Bass, Clements e Kazman, 2012).

VISÃO DE ALOCAÇÃO

A visão de alocação possibilita visualizar como o sistema está integrado a outras estruturas, que não são de *software*, tais como equipes de desenvolvimento, sistema de arquivo, entre outro…, permitindo visualizar como o sistema está distribuído e quem são os responsáveis pelo desenvolvimento/manutenção de cada elemento (Bass, Clements e Kazman, 2012).

2.5 UNIFIED MODELING LANGUAGE (UML)

A *Unified Modeling Language* (UML) é uma linguagem de modelagem amplamente utilizada na engenharia de *software* para especificar, visualizar, desenvolver e documentar os artefatos de um sistema. Criada no final da década de 1990 por Grady Booch, Ivar Jacobson e James Rumbaugh, a UML tornou-se um padrão mantido pelo *Object Management Group* (OMG), sendo amplamente reconhecida como referência no *design* e desenvolvimento de sistemas (Booch Et Al., 2005; Larman, 2004; Fowler, 2004).

A Linguagem de Modelagem Unificada (UML) oferece um conjunto de elementos gráficos que facilitam a representação de diferentes aspectos de um sistema, contribuindo para uma comunicação mais eficiente entre equipes técnicas e de negócios. Conforme destacado por Guedes (2010), a UML é ensinada por meio da apresentação de seus diversos diagramas, detalhando o propósito e a aplicação de cada um deles, bem como os elementos que os compõem, suas funções e como podem ser aplicados. Ela é composta por diversos diagramas, que representam diferentes aspectos do sistema, como diagramas de classes, de casos de uso, de sequência, entre outros. A adoção da UML no desenvolvimento de sistemas proporciona benefícios significativos, como a melhoria da compreensão do sistema, a identificação precoce de problemas e a facilitação da manutenção e evolução do *software* ao longo do tempo (Miro, 2025). Além disso, a UML é compatível com diversas metodologias de desenvolvimento, como o desenvolvimento ágil e o modelo em cascata, tornando-se uma ferramenta versátil e amplamente aceita na indústria de *software*.

Segundo Booch, Rumbaugh e Jacobson (2005), a UML oferece um conjunto de diagramas que podem ser utilizados para representar tanto a estrutura estática quanto o comportamento dinâmico de um sistema. Entre os diagramas mais utilizados estão:

- **Diagrama de Casos de Uso:** Representa as interações entre os autores (usuários ou outros sistemas) e as funcionalidades do sistema.
- **Diagrama de Classes:** Descreve a estrutura estática do sistema, mostrando as classes, seus atributos, métodos, e os relacionamentos entre elas.
- **Diagrama de Sequência:** Mostra a ordem das interações entre os objetos no sistema.
- **Diagrama de Componentes:** Representa a organização dos componentes de *software* e suas dependências.

- **Diagrama de Implementação:** Ilustra como o sistema será fisicamente implantado em *hardware* e redes.

3 PROTÓTIPO DO AMBIENTE

Neste capítulo, é apresentada a proposta de solução deste trabalho, inicialmente destacando que o protótipo foi desenvolvido por Santos et al, ao qual realizou o levantamento de requisitos, a especificação e a prototipação do ambiente. A Seção 3.1 aborda o levantamento de requisitos necessários para o desenvolvimento do sistema. Na Seção 3.2, são apresentados os diagramas de classes, casos de uso, sequência e componentes, com o objetivo de facilitar o desenvolvimento do *software*.

3.1 LEVANTAMENTO DE REQUISITOS

Os requisitos de um sistema são a base fundamental para a documentação e desenvolvimento de *software*, desempenhando um papel crucial ao garantir que as necessidades e expectativas dos usuários sejam devidamente atendidas (Sommerville, 2011). De acordo com Sommerville (2011), os requisitos devem ser descritos de maneira clara e precisa para que possam orientar todas as etapas do ciclo de vida de desenvolvimento do sistema, desde o *design* até a implementação e manutenção.

ATORES DO SISTEMA

Os atores envolvidos no uso da plataforma podem ser vistos no Quadro 1, 2 e 3.

Quadro 1 - Administrador

Papel	Recursos
Gerenciar o Sistema	O administrador pode cadastrar, atualizar, listar e excluir dados de usuários, *pets* e solicitações de adoção. Também pode alterar o *status* dos *pets* e registrar *feedbacks*.

Fonte: Santos et al. (2022)

Quadro 2 - Adotante

Papel	Recursos
Realizar solicitações	O adotante pode acessar o sistema, criar um perfil, consultar a lista de *pets* disponíveis e enviar solicitações de adoção.

Fonte: Santos et al. (2022)

Quadro 3 - Doador

Papel	Recursos
Acompanhar	O doador pode acompanhar as solicitações de adoção dos animais que ele

Solicitações	cadastrou no sistema, verificando os *status* e atualizações relacionadas a essas solicitações.

Fonte: Santos et al. (2022)

REQUISITOS FUNCIONAIS

Na Figura 10, é apresentado o diagrama de casos de uso, que ilustra as principais interações entre os usuários do sistema de adoção de animais e suas funcionalidades. O sistema possui três atores principais: o Adotante, o Doador e o Administrador, cada um com acesso a funcionalidades específicas.

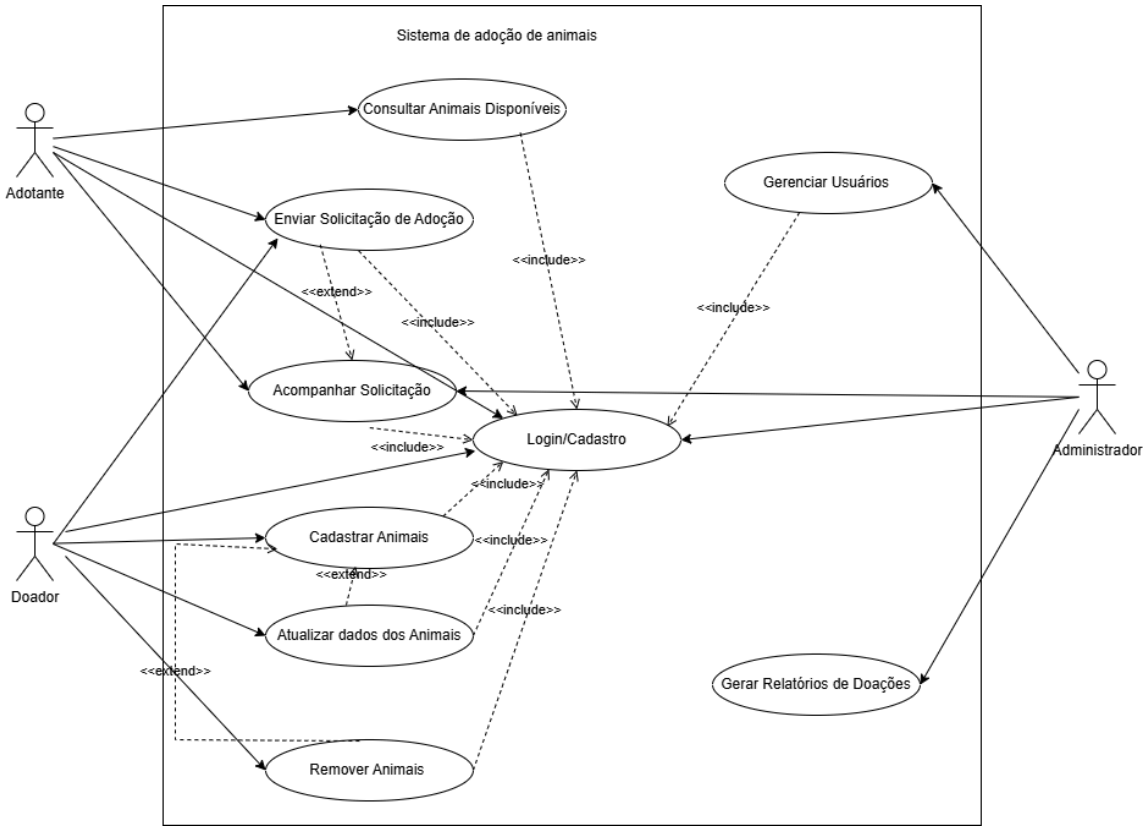
O **Adotante** pode realizar o *login* em seu perfil, funcionalidade que inclui a possibilidade de recuperar a senha em caso de esquecimento. Após estar autenticado no sistema, o adotante tem acesso às funcionalidades de consulta de animais disponíveis, envio de solicitações de adoção e acompanhamento do *status* dessas solicitações. Essas ações são interligadas, com o envio de uma solicitação de adoção se estendendo ao acompanhamento de sua evolução, permitindo um fluxo de interação contínuo e eficiente.

O **Doador** também desempenha um papel essencial no sistema. Ele pode realizar o *login* para acessar funcionalidades como o cadastro de novos animais, atualização de informações dos animais já cadastrados e a remoção de registros de animais, caso necessário. Essa interação permite que o sistema esteja constantemente atualizado com as informações dos animais disponíveis para adoção.

O **Administrador**, por sua vez, possui acesso a funcionalidades de gerenciamento que abrangem todo o sistema. Ele pode cadastrar novos animais, listar os *pets* disponíveis para adoção e alterar informações ou o *status* de um *pet*, indicando se está disponível ou adotado. Além disso, o administrador tem a capacidade de gerenciar usuários e gerar relatórios de doações, funcionalidades essenciais para a organização e a transparência do sistema. Também é possível que o administrador envie notificações aos usuários, ação que se estende a outras operações para garantir que os adotantes e doadores estejam sempre atualizados sobre mudanças relevantes no sistema.

Este diagrama foi desenvolvido utilizando o *software draw.io*, que possibilita uma representação clara e detalhada das interações entre os atores e o sistema. Ele oferece uma visão abrangente das funcionalidades, evidenciando os fluxos de ações e os relacionamentos entre as operações. Dessa forma, promove uma experiência organizada e eficiente tanto para os adotantes quanto para os doadores e administradores.

Figura 10 - Diagrama de Caso de Uso



Fonte: Adaptado de Santos et al. (2022)

Os requisitos funcionais para o sistema mobile de adoção de animais estão apresentados no quadro 4:

Quadro 4 - Requisitos Funcionais

Requisito	Descrição	Atores	Prioridade
[RF001] Cadastrar Administrador	O sistema deve permitir o cadastro dos usuários administradores, solicitando o seguinte dados: nome completo, cadastro de pessoa física (CPF), data de nascimento, *e-mail* e senha.	Administrador	Essencial
[RF002] Realizar *Login*	O sistema deve permitir que o administrador consiga alterar os dados que foram informados no momento do cadastro.	Administrador	Essencial
			(continua)

Requisito	Descrição	Atores	Prioridade
[RF003] Inativar Administrador	O sistema deve permitir a exclusão do cadastro do administrador.	Administrador	Importante
[RF004] Entrar no Perfil	O sistema deve permitir que o usuário seja ele administrador ou adotador, acesse seu perfil através de um *login*, informando o *email* e senha inseridos no momento do cadastro.	Administrador, Adotante	Essencial
[RF005] Consultar Dicas	O sistema deve permitir ao usuário realizar a recuperação de senha, informando o email do seu cadastro.	Administrador, Adotante	Essencial
[RF006] Cadastrar *Pets*	O sistema deve permitir que o usuário administrador realize o cadastro de todos os animais disponíveis para adoção, inserindo nome, raça, idade, espécie, cor da pelagem, sexo, vacinas em dia, porte e endereço onde fica o animal.	Administrador	Essencial
[RF007] Atualizar *Pets*	O sistema deve permitir que o usuário administrador consiga alterar os dados dos *pets* que foram informados no momento do cadastro.	Administrador	Importante
[RF008] Listar *Pets*	O sistema deve permitir a exibição de uma lista com todos os *pets* cadastrados no banco de dados da aplicação exibindo todos os seus dados.	Administrador, Adotantes	Essencial
[RF009] Excluir *Pets*	O sistema deve permitir que o usuário administrador exclua os *pets* cadastrados na aplicação.	Administrador	Importante
[RF010] Cadastrar Adotante	O sistema deve permitir o cadastro dos usuários adotantes, solicitando os seguintes dados: nome completo, cadastro de pessoa física(CPF), data de nascimento, *e-mail*, senha e telefone.	Administrador	Essencial

(continua)

Requisito	Descrição	Atores	Prioridade
[RF011] Atualizar Adotante	O sistema deve permitir que o adotante consiga alterar os dados que foram informados no momento do cadastro.	Administrador	Essencial
[RF012] Excluir Adotante	O sistema deve permitir a exclusão do cadastro do adotante.	Administrador	Importante
[RF013] Exibir Informações para Resgate	O sistema deve permitir a exibição de informações para a realização dos primeiros socorros nos *pets* encontrados em situação de vulnerabilidade.	Administrador	Desejável
[RF014] Buscar *Pets*	O sistema deve permitir a busca por algum *pet*, pelo sexo, nome, cor, tipo de pets e em seguida exibir o resultado da busca contendo as informações do *pet*.	Adotante	Importante
[RF015] Solicitar Adoção	O sistema deve permitir que o usuário adotante solicite por adoção dos *pets* inserindo apenas dados básicos como: se deseja um filhote ou adulto, a idade, nome, telefone e endereço de quem deseja adotar e qual a utilidade do *pet*.	Adotante	Importante
[RF016] Listar Solicitações	O sistema deve permitir a exibição de uma lista com todas as solicitações cadastrados no banco de dados da aplicação, exibindo todos os seus dados para o usuário administrador.	Administrador	Essencial
[RF017] Visualizar Adoções	O sistema deve permitir que o administrador visualize as doações recebidas, bem como o tipo de doação que foi realizada.	Administrador	Essencial
[RF018] Alterar *Status*	O administrador irá atualizar o *status* dos *pets* (disponível, em tratamento, adotado) e em seguida exibirá essa informação para os usuários.	Administrador	Essencial

(continua)

Requisito	Descrição	Atores	Prioridade
[RF019] Receber Feedback	O sistema deve enviar *e-mail* para usuário respondendo se a solicitação foi atendida.	Administrador	Desejável
[RF020] Acompanhar Solicitação	O sistema deve permitir que o usuário adotante acompanhe suas solicitações de adoção realizadas.	Adotante	Importante
[RF021] Atualizar *Status* de Solicitação	O sistema deve permitir que o usuário administrador altere o status das solicitação de adoção para visualizada, pendente e aprovada.	Administrador	Essencial
[RF022] Notificar	O sistema deve enviar mensagem ao usuário adotante, sempre que houver atualização no *status* de sua solicitação de adoção, informando o estado o qual a mesma se encontra a ser movimentada.	Adotante	Essencial
[RF023] Notificar Doador	O sistema deve enviar mensagem ao doador sempre que estiver faltando 48 horas para a realização da sua doação.	Administrador, Doador	Desejável
[RF024] Cadastrar *Feedback* da Adoção	O sistema deve permitir que o administrador cadastre um *feedback* após fazer o acompanhamento das adoções.	Administrador	Desejável
[RF025] Gerar Relatórios *Pets*	O sistema deve permitir aos administradores gerar um relatório acerca dos animais adotados, disponíveis e castrados.	Administrador	Importante
[RF026] Gerar Relatórios de Solicitações	O sistema deve permitir aos administradores gerar um relatório acerca das solicitações de adoções realizadas, a fim de poderem visualizar e ter um controle maior sobre as mesmas.	Administrador	Importante

Fonte: Santos et al. (2022)

REQUISITOS NÃO-FUNCIONAIS

No quadro 5 são elencados os requisitos não funcionais do ambiente.

Quadro 5 - Requisitos Não Funcionais

Requisito	Descrição	Categoria
[RNF001] Autenticação do Usuário	O sistema deve autenticar os usuários para a realização das principais operações.	Segurança
[RNF002] Interface Agradável	O sistema deve ter uma interface amigável, com cores agradáveis para a visão do usuário.	Usabilidade

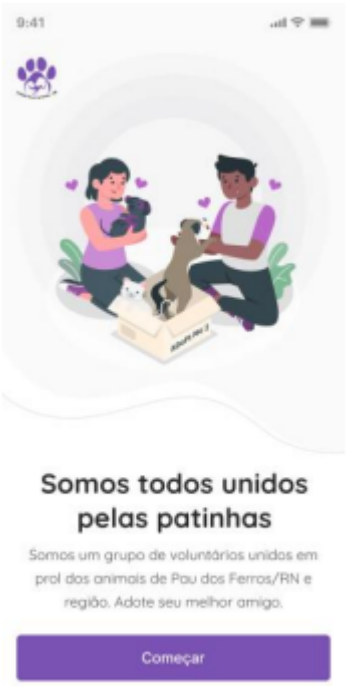
Fonte: Santos et al. (2022)

3.2 PROTÓTIPO

O protótipo apresentado abaixo visa ilustrar a interface e a interação do usuário com o sistema de adoção. As telas foram desenvolvidas por Santos et al. (2022) e projetadas para garantir acessibilidade e facilidade de uso, atendendo aos requisitos funcionais previamente descritos.

Na Figura 11, é apresentada a tela inicial do aplicativo, que serve como porta de entrada para os usuários. Esta tela destaca os valores e a missão do sistema, criando uma interface amigável e atrativa.

Figura 11 - Tela inicial do aplicativo



Fonte: Santos et al. (2022)

Por sua vez, na Figura 12, é apresentada a tela de login. Nessa tela, o usuário deve fornecer seu *e-mail* e senha para acessar o sistema. Caso ainda não possua uma conta, ele poderá realizar o cadastro por meio da tela exibida na Figura 13, que apresenta o formulário necessário para esse procedimento.

Figura 12 - Tela de *login*

Fonte: Santos et al. (2022)

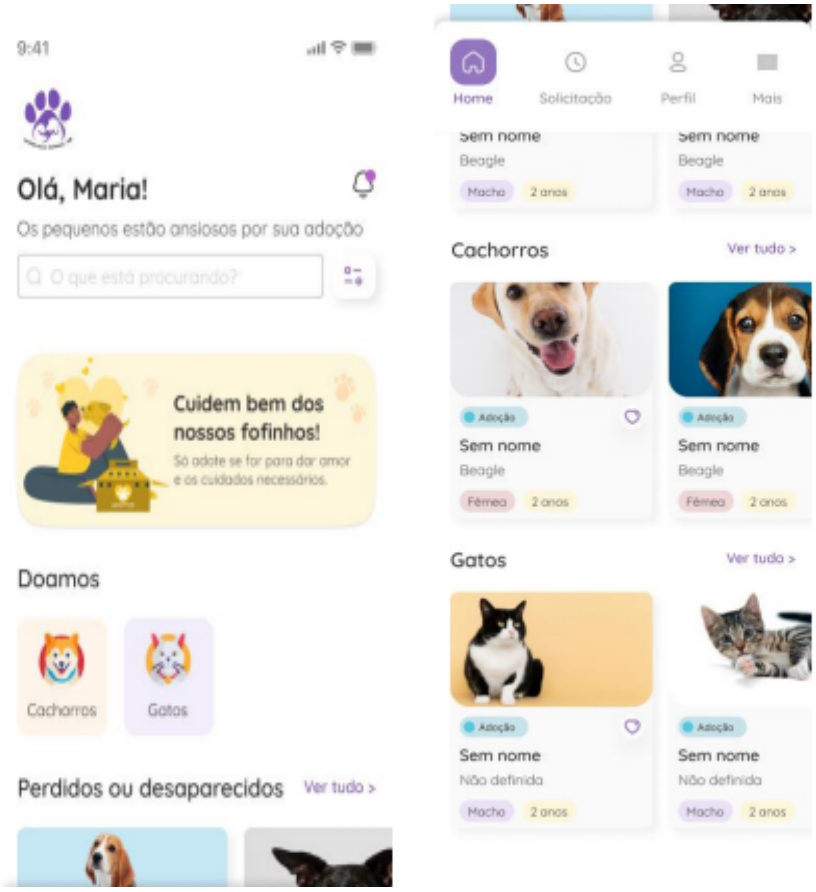
Figura 13 - Tela de cadastro

Fonte: Santos et al. (2022)

Na Figura 14, é apresentada a tela principal do ambiente. Nessa interface, o usuário pode navegar pelas diferentes funcionalidades do sistema. A tela inicial exibe opções como busca de animais disponíveis para adoção, divididas em categorias como “Cachorros” e “Gatos”, além de seções com informações sobre animais perdidos ou desaparecidos. O usuário pode interagir com os ícones de categoria para filtrar os animais de acordo com seu interesse.

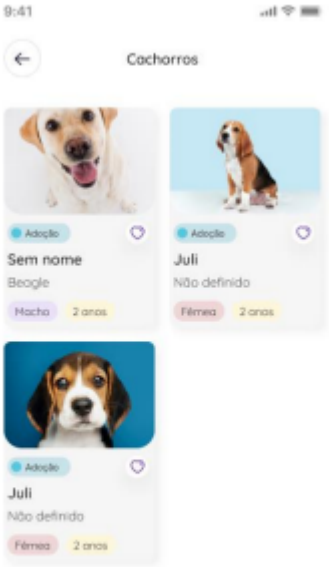
Ao selecionar uma categoria específica, como "Cachorros", o sistema apresenta uma listagem detalhada dos animais disponíveis, conforme ilustrado na Figura 15. Cada *card* exibe informações importantes sobre o animal, como nome, raça, idade e *status* de adoção, permitindo ao usuário explorar as opções antes de tomar uma decisão.

Figura 14 - Tela principal



Fonte: Santos et al. (2022)

Figura 15 - Lista de animais da categoria cachorros



Fonte: Santos et al. (2022)

Na Figura 16, é apresentada a tela de perfil de um animal de estimação. Nessa tela, o usuário pode visualizar uma imagem do animal, assim como informações detalhadas, incluindo nome, status (perdido ou disponível para adoção) e outras características. Caso o animal esteja perdido, a tela exibirá botões de contato, como “Ligar” e “WhatsApp”, permitindo que o usuário entre em contato diretamente com o responsável ou com o grupo. Além disso, há uma descrição que detalha informações sobre a localização onde o animal foi visto pela última vez, bem como orientações sobre como proceder caso o usuário tenha informações relevantes.

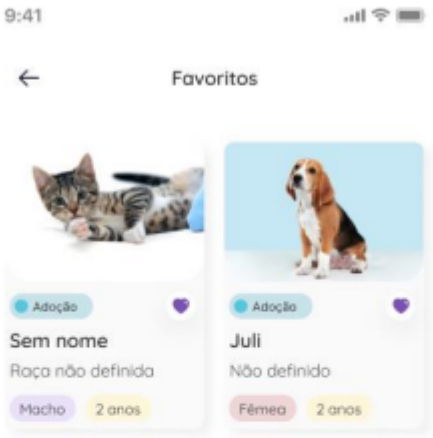
Se o animal estiver disponível para adoção, o botão "Adotar" estará habilitado. Ao clicar nesse botão, o usuário será direcionado para o procedimento de adoção, onde poderá fornecer seus dados e realizar o processo necessário para a adoção do *pet*.

Figura 16 - Perfil de um animal de estimação

Fonte: Santos et al. (2022)

Pelo aplicativo também é possível favoritar os *pets* de interesse, como pode ser visto na Figura 17. Além disso, ele pode buscar algum *pet* dentre a lista de animais favoritos.

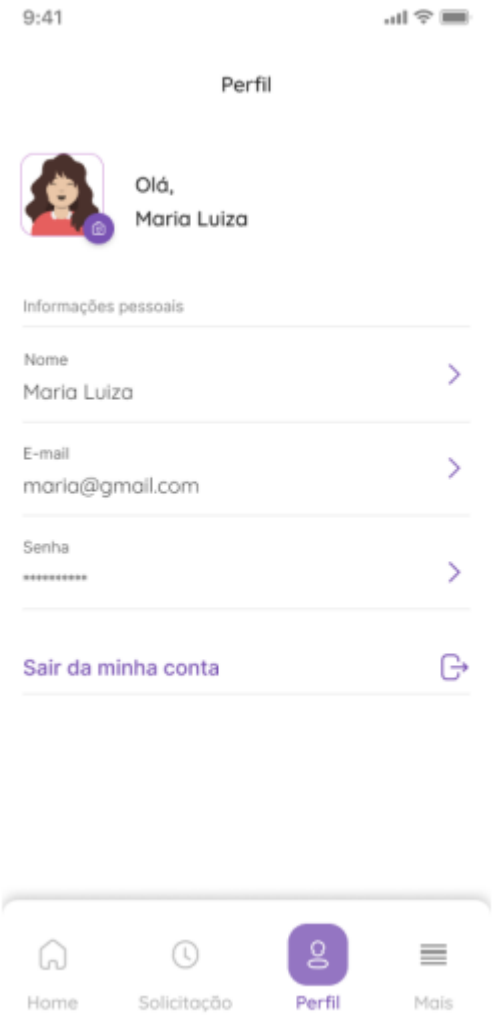
Figura 17 - Lista de favoritos



Fonte: Santos et al. (2022)

Além disso, os usuários do aplicativo terão a opção de editar suas informações pessoais por meio da tela de perfil, apresentada na Figura 18.

Figura 18 - Tela de perfil



Fonte: Santos et al. (2022)

Na Figura 19 pode ser vista as telas do processo de realizar uma adoção. O processo começa com a coleta de dados para contato, onde o usuário deve preencher campos obrigatórios, como telefone, endereço e cidade, necessários para viabilizar a comunicação e os trâmites do processo.

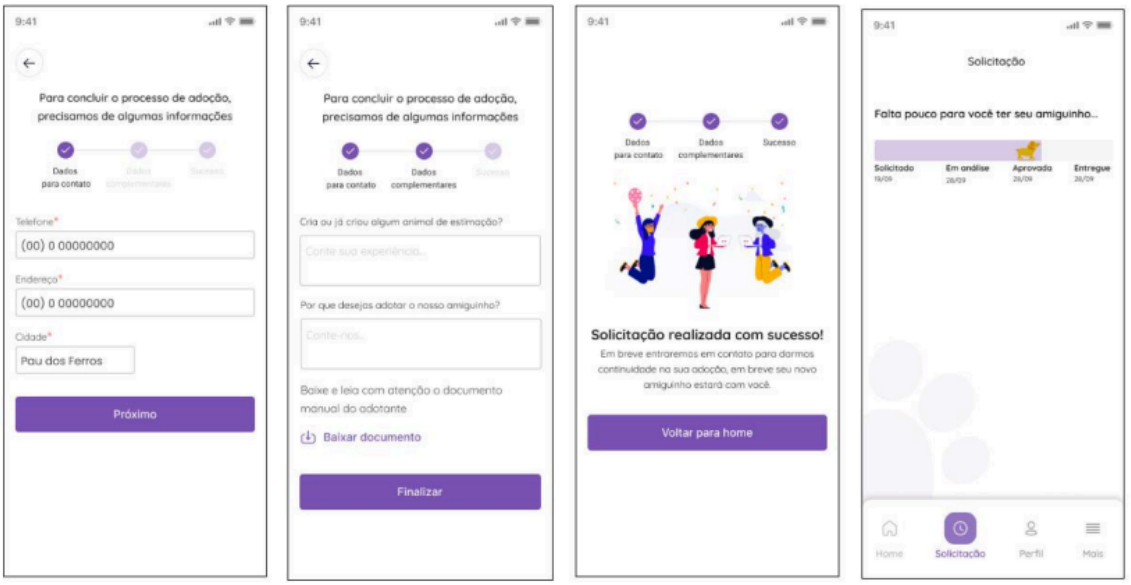
Na sequência, a tela de informações complementares solicita detalhes adicionais, como a experiência do solicitante com animais de estimação e o motivo da adoção. Além disso, é exigida a leitura e aceitação dos termos do manual do adotante, que pode ser baixado diretamente no aplicativo. Após o preenchimento das informações, o usuário finaliza esta etapa clicando em "Finalizar".

Com a conclusão das etapas anteriores, o aplicativo exibe uma mensagem de sucesso, informando que a solicitação foi enviada e que a equipe responsável entrará em contato para dar continuidade ao processo.

Por fim, o solicitante pode acompanhar o *status* da adoção por meio de uma tela dedicada. O processo é dividido em quatro etapas:

1. **Solicitado:** Indica que a solicitação foi registrada.
2. **Em análise:** Mostra que a equipe está avaliando as informações enviadas.
3. **Aprovado:** Confirma que a solicitação foi aceita.
4. **Entregue:** Indica que o processo foi concluído e o animal entregue ao novo adotante.

Figura 19 - Sequência de etapas para se realizar uma adoção



Fonte: Santos et al. (2022)

3.3 PROPOSTA DE ARQUITETURA

Para modelar a arquitetura do ambiente foi utilizada a divisão em visões arquiteturais proposta por Bass, Clements e Kazman (2012).

3.3.1 Visão de Módulo

No contexto deste trabalho, a visão de módulos foi representada por meio de um diagrama de classes. Na Figura 20, apresenta-se o diagrama de classes desenvolvido para o sistema proposto, detalhando a estrutura estática e as responsabilidades de cada módulo. Além de representar a organização das classes, o diagrama reflete a aplicação de princípios de *design* de *software*, especificamente, os princípios SOLID, que desempenham um papel fundamental, pois orientam a organização das classes e a forma como interagem entre si. A seguir, são destacados os principais princípios aplicados no modelo:

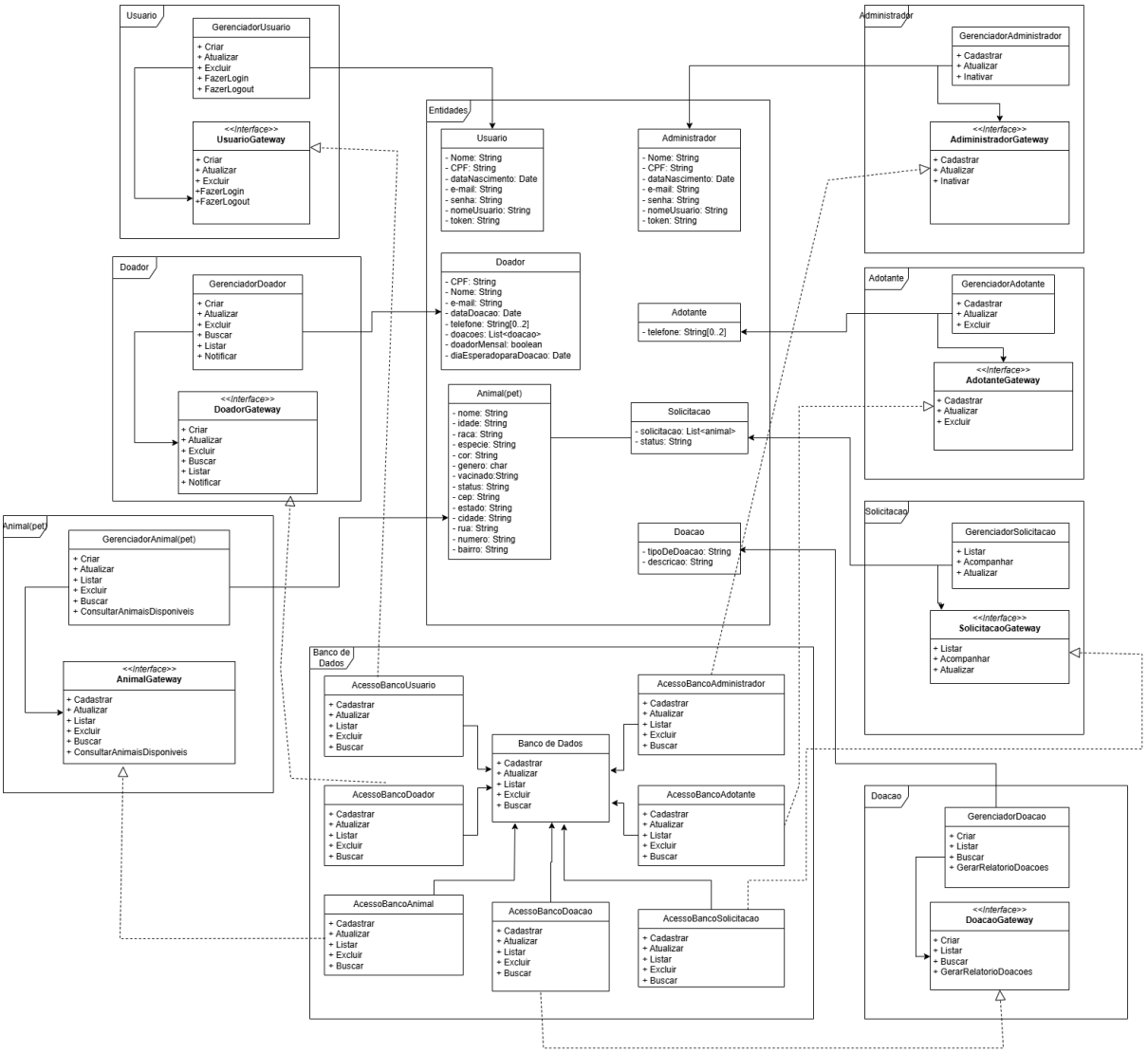
- **Princípio da Responsabilidade Única:** Cada classe no diagrama possui uma responsabilidade bem definida, evitando a sobrecarga de funções em uma única entidade. Por exemplo, as classes *GerenciadorUsuario*, *GerenciadorDoador*, *GerenciadorAdotante* e *GerenciadorAnimal* encapsulam operações específicas de gerenciamento, garantindo a separação de preocupações e facilitando a manutenção do código.
- **Princípio Aberto/Fechado:** O uso de interfaces, como *UsuarioGateway*, *DoadorGateway* e *AdotanteGateway*, permite a extensão do sistema sem a necessidade de modificações no código existente. Dessa forma, é possível adicionar novas implementações sem comprometer a estrutura já estabelecida.
- **Princípio da Substituição de Liskov:** A hierarquia entre as classes *Usuário*, *Doador*, *Administrador* e *Adotante* indica que subclasses podem substituir suas classes bases sem alterar o comportamento esperado pelo sistema. Isso garante a consistência das operações e evita falhas decorrentes de implementações incompatíveis.
- **Princípio da Segregação de Interfaces:** O diagrama demonstra a criação de interfaces especializadas, assegurando que cada classe implemente apenas os métodos necessários. Por exemplo, as interfaces *Gateway* são definidas de

maneira específica para cada entidade, evitando que módulos sejam obrigados a implementar funcionalidades desnecessárias.

- **Princípio da Inversão de Dependência:** A dependência de abstrações, evidenciada pelo uso de interfaces como *SolicitacaoGateway*, *DoacaoGateway* e *AnimalGateway*, reduz o acoplamento entre os módulos do sistema. Dessa forma, a implementação de novas funcionalidades ou a substituição de componentes pode ser realizada sem impacto significativo na arquitetura geral.

A aplicação desses princípios no projeto reforça a qualidade do *design* da arquitetura, tornando o sistema mais escalável, sustentável e alinhado às boas práticas de engenharia de *software*. Além de facilitar a manutenção e evolução do código, esses princípios contribuem para um desenvolvimento mais organizado e eficiente, atendendo melhor às necessidades do grupo e dos usuários envolvidos no processo de adoção de animais.

Figura 20 - Diagrama de Classes



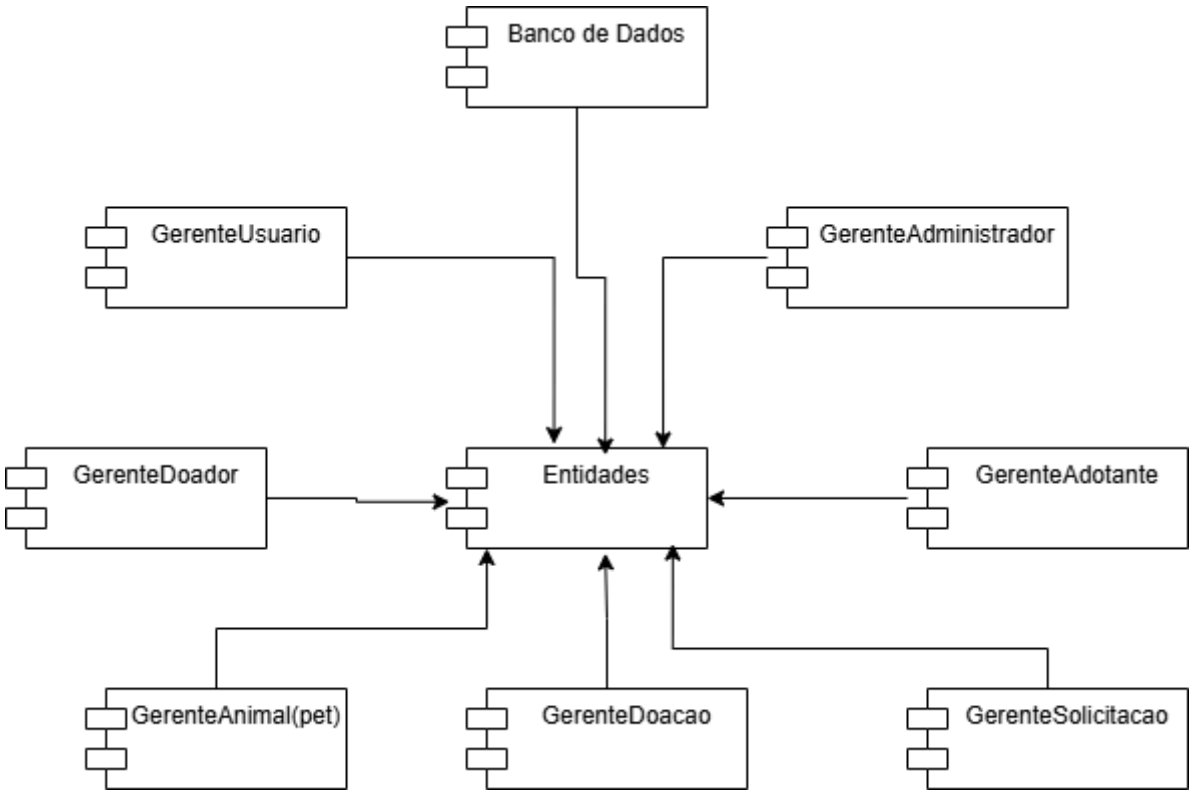
Fonte: Autoria própria (2024)

3.3.2 Visão Componente e Conector

No sistema proposto, os componentes foram definidos para incluir gerenciadores responsáveis pelos principais atores do sistema (por exemplo, usuário, Administrador, Adotante, Doacao). Essa organização segue os princípios de Coesão de Componentes de Martin (2019), como o Princípio do Fechamento Comum, que recomenda agrupar em um único componente as classes que mudam pelas mesmas razões e separar em componentes distintos aquelas que não compartilham esse comportamento.

Uma das formas de representar a visão de componente & conector é por meio de diagramas de componentes. Na Figura 21, apresenta-se o diagrama de componentes para o ambiente proposto, destacando os elementos e suas interações no sistema.

Figura 21 - Diagrama de Componentes

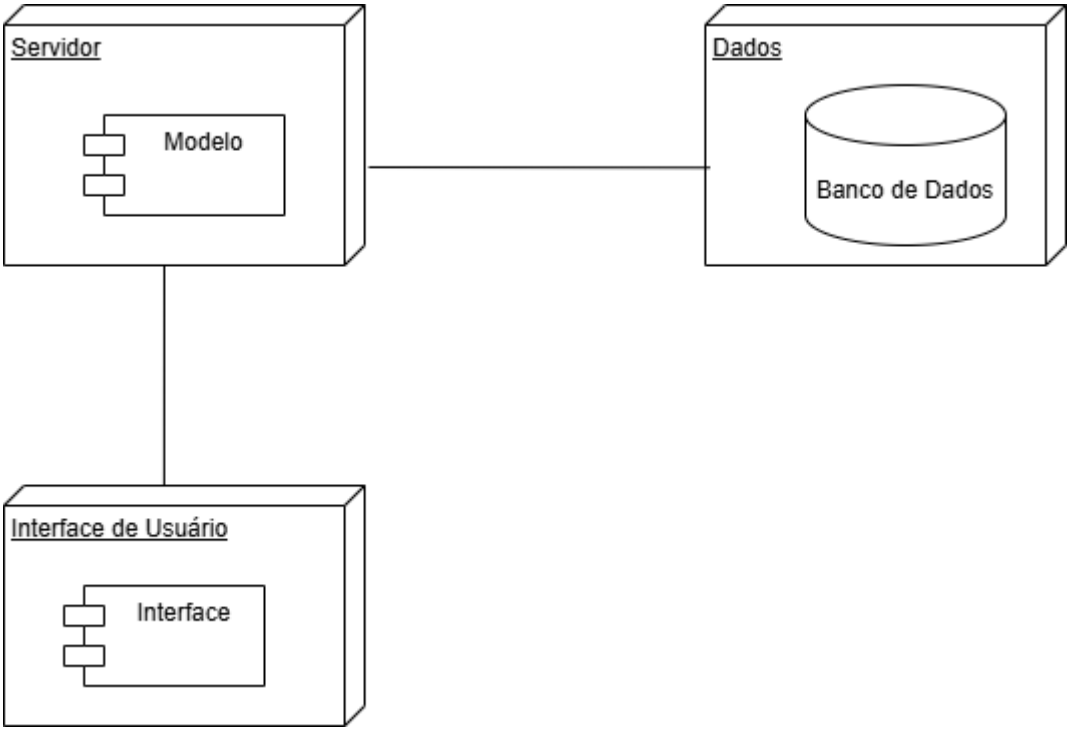


Fonte: Autoria própria(2024)

3.3.3 Visão De Alocação

Uma das formas de representar uma visão de alocação é por meio de diagramas de implantação. Na Figura 22 pode ser visto o diagrama de implantação para o sistema proposto.

Figura 22 - Diagrama de Implantação



Fonte: Autoria própria(2024)

O sistema foi estruturado para permitir que o usuário o acesse por meio de um navegador *Web*, representado no diagrama pelo bloco “interface de usuário”. Essa interface é responsável por possibilitar a interação direta do usuário com o sistema, oferecendo funcionalidades como *login*, consulta de animais disponíveis para adoção e acompanhamento de solicitações.

A Interface de Usuário se comunica com o Servidor, representado no diagrama como o núcleo da lógica de negócios do sistema. No servidor, estão implementados módulos que gerenciam usuários, animais, solicitações e doações, além de realizar as operações necessárias para atender às interações iniciadas pelos usuários.

O Servidor acessa o Banco de Dados, que armazena todas as informações essenciais do sistema, como usuários (adotantes, doadores e administradores), dados dos animais,

histórico de solicitações e registros de doações. Essa estrutura permite um fluxo claro entre as camadas do sistema, garantindo organização e eficiência.

No que diz respeito à flexibilidade do sistema, ele foi projetado de forma a permitir modificações e ampliações de maneira eficiente. Por exemplo, caso seja necessário adicionar uma nova funcionalidade, como o suporte a novos tipos de doação, seria possível criar classes específicas para representar as entidades relacionadas, além de um gerenciador para lidar com as operações dessa funcionalidade e as classes associadas ao banco de dados.

A adoção de princípios arquiteturais, como os fundamentos do SOLID, contribui para que o sistema seja escalável, compreensível e capaz de ser modificado sem altos custos de manutenção, mantendo a qualidade do código e a facilidade de implementação de novas funcionalidades.

4 AVALIAÇÃO

Para avaliar a arquitetura proposta foi utilizada uma avaliação baseada em cenários de uso, uma abordagem comum empregada em avaliações arquiteturais, a exemplo do método *Architecture Tradeoff Analysis Method* — ATAM (Bass, Clements e Kazman, 2012). O cenário de uso da arquitetura é modelado com o uso de diagramas de sequência da UML.

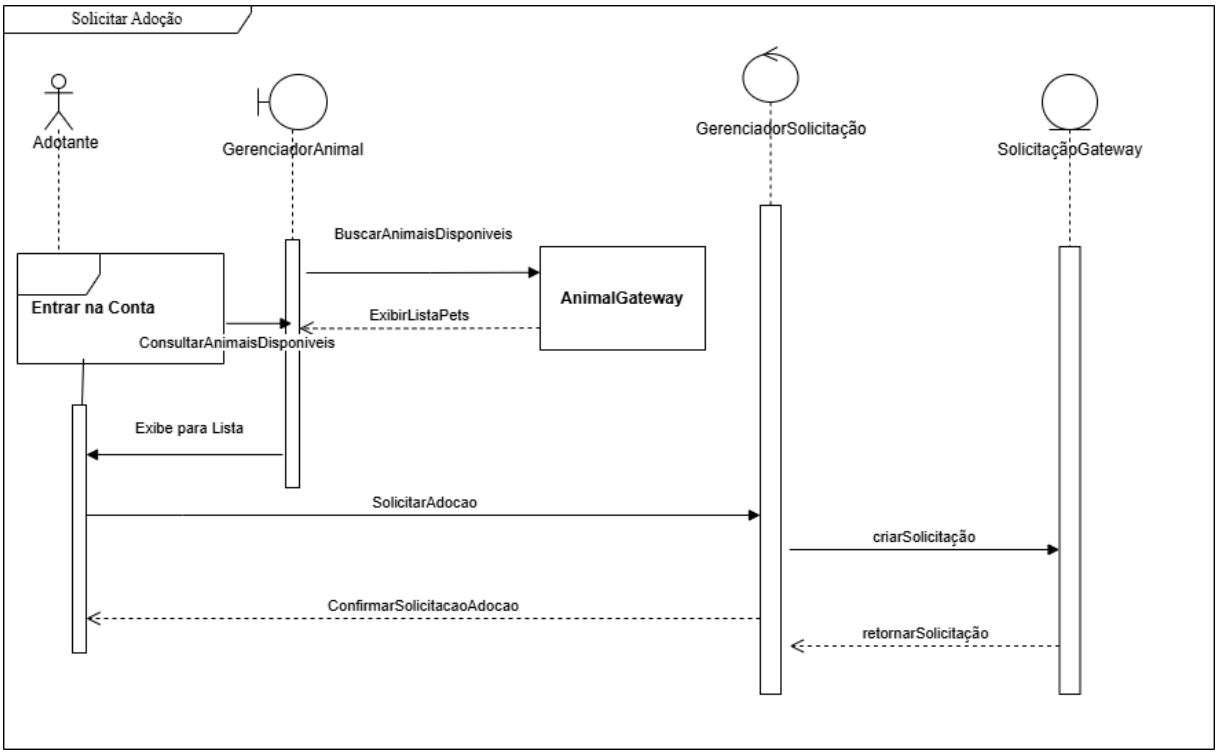
4.1 AVALIAÇÃO BASEADA EM CENÁRIOS DE USO E INTERAÇÃO COM O SISTEMA

A Figura 23 apresenta o diagrama de sequência modelado no *Draw.io*, representando detalhadamente o processo de solicitação de adoção dentro do sistema. O fluxo tem início quando o adotante acessa a plataforma e realiza *login*. Após autenticação, ele solicita a lista de animais disponíveis para adoção. Essa solicitação é enviada ao *GerenciadorAnimal*, que, por sua vez, encaminha a busca de animais ao *AnimalGateway* por meio do método *BuscarAnimaisDisponiveis.* O *AnimalGateway* busca os dados dos *pets* no banco de dados e retorna uma lista contendo os animais disponíveis. Essa lista é então exibida na interface do adotante.

Ao visualizar a lista, o adotante pode selecionar um animal e clicar na opção de *SolicitarAdoção*. Essa ação gera um comando que é enviado ao *GerenciadorSolicitacao,* o qual processa a solicitação e aciona o *SolicitacaoGateway* por meio do método *CriarSolicitação.* O *SolicitaçãoGateway*, por sua vez, registra a solicitação no banco de dados e retorna uma confirmação ao *GerenciadorSolicitação* por meio do método *retornarSolicitação*.

Após receber a confirmação, o *GerenciadorSolicitação* repassa a resposta ao sistema, que exibe ao adotante a mensagem informando que a solicitação de adoção foi registrada com sucesso. Dessa forma, o diagrama evidencia todas as interações necessárias para que um adotante possa visualizar e solicitar a adoção de um animal por meio do sistema.

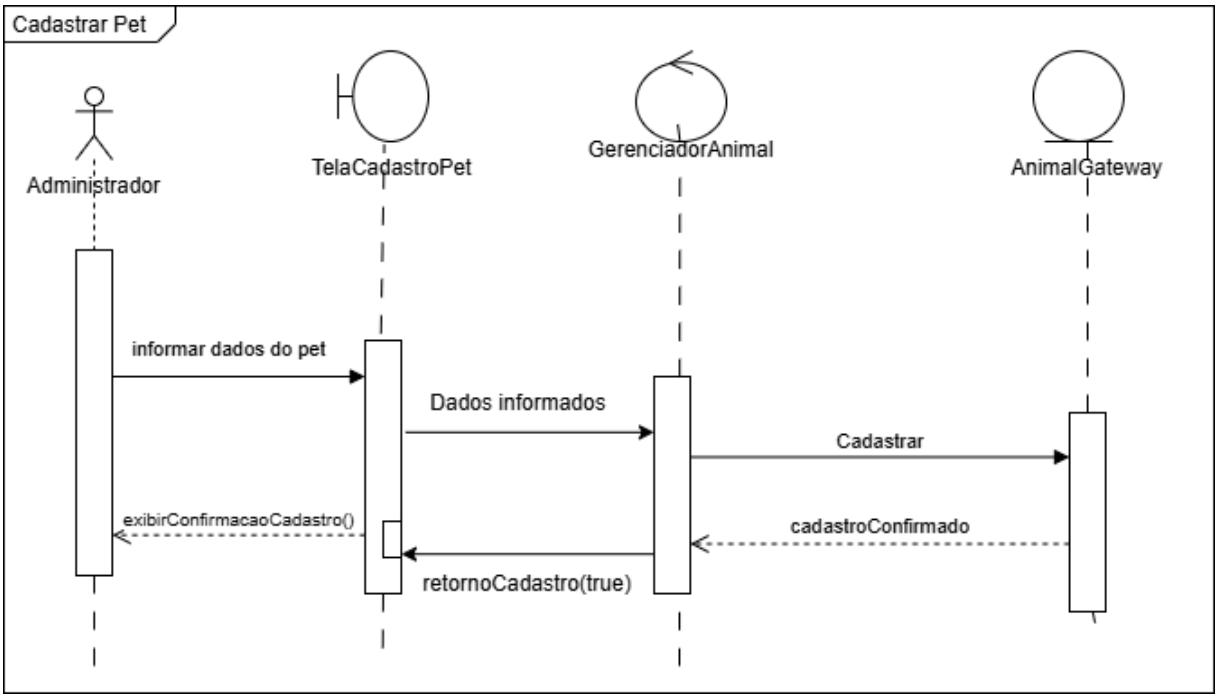
Figura 23 - diagrama de sequência do cadastro de uma solicitação



Fonte: Adaptado de Santos et al. (2022)

O Diagrama de Sequência da Figura 24, modelado usando a ferramenta *Draw.io*, mostra como acontece o processo de cadastro de um *pet* no sistema. Tudo começa quando o administrador informa os dados do *pet* na *TelaCadastroPet.* Esses dados são então enviados para o *GerenciadorAnimal*, que tem a função de processar a solicitação e encaminhá-la ao *AnimalGateway*, responsável por registrar o *pet* no sistema. O *AnimalGateway* executa o método *cadastrar*, salvando o novo pet na base de dados. Depois que o cadastro é concluído com sucesso, o sistema retorna uma confirmação (*cadastroConfirmado*) para o *GerenciadorAnimal*, que repassa essa resposta para a interface. Por fim, a *TelaCadastroPet* exibe uma mensagem para o administrador, confirmando que o *pet* foi cadastrado corretamente. Esse fluxo garante que o processo de cadastro seja organizado, eficiente e siga corretamente a estrutura do sistema.

Figura 24 - diagrama de sequência do cadastro de um pet

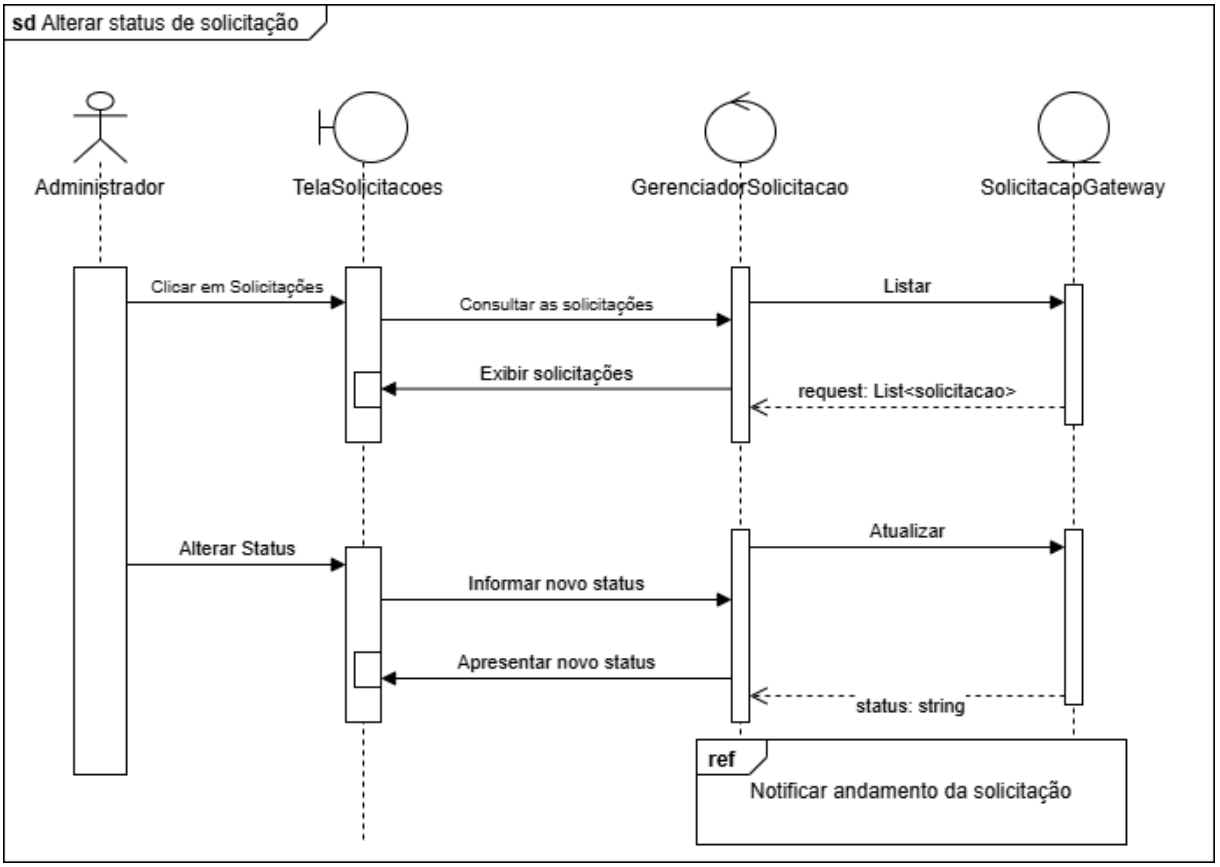


Fonte: Adaptado de Santos et al. (2022)

No Diagrama de Sequência da Figura 25, modelado no *Draw.io*, mostra como o administrador interage com o sistema para alterar o *status* de uma solicitação. Primeiro, o administrador acessa a seção de solicitações na interface e clica para visualizá-las. A *TelaSolicitacoes* então solicita a lista de solicitações ao *GerenciadorSolicitacao*, que busca esses dados no sistema e os retorna para exibição.

Depois de visualizar as solicitações, o administrador escolhe uma e altera seu *status*. Essa atualização é enviada pela *TelaSolicitacoes* para o *GerenciadorSolicitacao*, que processa a mudança e repassa a atualização para o *SolicitacaoGateway*, responsável por registrar a modificação no sistema. Após a confirmação da alteração, o novo *status* é atualizado na interface e o sistema notifica o andamento da solicitação.

Figura 25 - Diagrama de sequência da alteração do status de uma solicitação.



Fonte: Adaptado de Santos et al. (2022)

5 CONSIDERAÇÕES FINAIS

Neste trabalho, desenvolvemos uma proposta de arquitetura para um sistema voltado à adoção de animais resgatados pelo grupo Unidos pelas Patinhas. A modelagem foi elaborada considerando boas práticas de desenvolvimento e diferentes visões arquiteturais, buscando criar uma solução eficiente, escalável e de fácil manutenção. A proposta tem como objetivo centralizar informações sobre os animais, otimizar o processo de adoção e aumentar a transparência das ações da, facilitando a interação entre os adotantes e o grupo.

A avaliação da arquitetura foi realizada através de cenários de uso e interação com os atores do sistema, garantindo que os requisitos definidos fossem atendidos. Além disso, a modelagem com diagramas UML permitiu uma visão clara da estrutura do sistema e seus componentes, favorecendo uma implementação mais organizada e consistente.

Embora o sistema ainda não tenha sido completamente desenvolvido, os resultados obtidos até esta etapa demonstram um grande potencial para melhorar os processos do grupo. A expectativa é que, com a futura implementação, a plataforma torne a adoção mais ágil, eficiente e acessível, impactando positivamente tanto os adotantes quanto os animais resgatados.

5.1 TRABALHOS FUTUROS

A implementação de um sistema de adoção de animais para o Unidos pelas Patinhas representa uma excelente oportunidade para otimizar o processo de adoção e garantir mais eficiência na gestão dos animais resgatados. Segundo Silva et al. (2018), a falta de um sistema organizado pode dificultar a adoção responsável, resultando na permanência prolongada dos animais nos abrigos. Além disso, estudos como o de Silva (2017) apontam que plataformas digitais voltadas à adoção de animais podem tornar o processo mais ágil e transparente, conectando adotantes e ONGs de forma eficiente.

Além de permitir o cadastro detalhado dos animais disponíveis para adoção, o sistema poderá incluir funcionalidades que incentivem a adoção responsável, como questionários para perfis de adotantes e acompanhamento pós-adoção. Como destacam Silva, Silva e Silva (2016), ações de conscientização são fundamentais para promover a adoção responsável e reduzir o abandono. Essas iniciativas permitem que potenciais adotantes compreendam melhor as responsabilidades envolvidas, contribuindo para um processo mais seguro e eficaz.

Outra melhoria planejada é a inclusão de diferentes perfis de usuário, como administradores, voluntários e veterinários, possibilitando uma gestão mais eficiente. Com essa estrutura, será possível registrar o histórico de saúde dos animais, acompanhar vacinas e castrações, além de facilitar a comunicação entre o grupo e os adotantes. Conforme destaca Silva (2017), sistemas que oferecem interfaces acessíveis para múltiplos usuários aumentam significativamente a eficiência e o engajamento das partes envolvidas.

Com essas futuras implementações, a plataforma se tornará uma ferramenta essencial no combate ao abandono e na promoção da adoção responsável, auxiliando diretamente no bem-estar animal.

REFERÊNCIAS

BASS, L.; CLEMENTS, P.; KAZMAN, R. Software architecture in practice. 3. ed. Boston, MA. USA.: Addison-Wesley, 2012.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. The unified modeling language user guide. 2. ed. Boston, MA. USA.: Addison-Wesley, 2005.

CASA DO DESENVOLVEDOR, C. Princípios SOLID: Entendendo o que são e como podem revolucionar seu código. Casa do Desenvolvedor, 2023. Disponível em: <https://blog.casadodesenvolvedor.com.br/principios-solid-o-que-sao/>. Acesso em: 9 mar. 2025.

FOWLER, Martin. UML distilled*:* a brief guide to the standard object modeling language. 3. ed. Boston: Addison-Wesley, 2004.

GUEDES, G. T. P. UML 2: Uma Abordagem Prática. 3. ed. São Paulo, SP: Novatec, 2010.

JUSBRASIL. Brasil tem 30 milhões de animais abandonados. JUSBRASIL, 2013. Disponível em: <https://www.jusbrasil.com.br/noticias/brasil-tem-30-milhoes-de-animais-abandonados/100681698>. Acesso em: 3 mar. 2025.

LARMAN, Craig. Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development. 3. ed. Upper Saddle River: Prentice Hall, 2004.

MARTIN, Robert C. Agile Software Development, Principles, Patterns, and Practices. 1. ed. Londres, Reino Unido: Pearson Education, 2023.

MARTIN, Robert C. Arquitetura limpa: O guia do artesão para estrutura e design de software. Trad. Daniel F. de Souza. Rio de Janeiro: Alta Books, 2019.

MIRO. O guia definitivo para diagramas UML. Miro, [s.d.]. Disponível em: <https://miro.com/pt/diagrama/o-que-e-uml/>. Acesso em: 9 mar. 2025.

MOUTINHO, F. F. B.; SERRA, C. M. B.; VALENTE, L. C. M. Situação pós-adoção dos animais adotados junto a uma ONG de proteção animal no estado do Rio de Janeiro. Revista de Ciências Agrárias, v. 62, n. 1, p. 1–7, 2019.

OMG. About the Unified Modeling Language Specification Version 2.5.1. OMG, 2017. Disponível em: <https://www.omg.org/spec/UML>. Acesso em: 9 mar. 2025.

RICHARDS, Mark; FORD, Neal. Fundamentals of Software Architecture: An Engineering Approach. 1. ed. Sebastopol, Califórnia, EUA: O'Reilly Media, 2020.

SANTOS, J. D. S.; SOUZA, D. M. O.; SILVA, F. L. A. DA. SOUSA, R. R. DE Desenvolvimento de app mobile para adoção de animais recuperados pela ONG Unidos pelas Patinhas - Pau dos Ferros/RN. Conjecturas, v. 22, n. 16, p. 704–720, 2022.

SÃO PAULO, G. DO E. Coordenadoria de Defesa e Saúde Animal. Governo do Estado de São Paulo, [s.d.]. Disponível em: <https://www.saude.sp.gov.br/coordenadoria-de-defesa-e-saude-animal/homepage/destaques/saude-animal-x-saude-publica>. Acesso em: 9 mar. 2025.

SILVA, A. S.; SILVA, M. S.; SILVA, R. S. As mídias sociais como promotoras da adoção de cães. In: Congresso Brasileiro de Extensão Universitária, 7., 2016, Ouro Preto. Anais [...]. Ouro Preto: UFOP, 2016. p. 1-5.

SILVA, R. Pet Adoto: Projeto de interface de aplicativo para adoção de animais. 2017. Trabalho de Conclusão de Curso (Graduação em Desenho Industrial) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2017.

SILVA, A. S.; SILVA, M. S.; SILVA, R. S. Adopet: um sistema de gerenciamento de adoção de animais. In: Congresso de Tecnologia da Informação e Comunicação do IFC, 5., 2018, Araquari. **Anais** [...]. Araquari: IFC, 2018. p. 1-10.

SILVA, M. C. P. B. *Controle populacional de cães e gatos: desafios e estratégias.* Universidade Federal da Paraíba, 2018. Disponível em: repositorio.ufpb.br

SOMMERVILLE, I. Software engineering. 9. ed. Boston, MA. USA.: Pearson, 2011.

TOZZI, T.; ANDERLE, D. F.; NOGUEIRA, R. R.. Levantamento de Tecnologias para ONGs de Proteção Animal para apoio ao resgate de animais domésticos acoplados ao ciclo de vida de um sistema Web. In: Workshop de Trabalhos de Iniciação Científica - Simpósio Brasileiro De Sistemas Multimídia e Web (Webmedia), 24., 2018, Salvador. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2018 . p. 81-84.