UNIVERSIDADE PAULISTA

G937198 - MIKAEL FELIX DE BRITO

G9270A5 - DANIEL QUINTANA

R015EI4 - MATEUS RODRIGUES GARCIA

DESENVOLVIMENTO DE UM SISTEMA INTEGRADO PARA GESTÃO DE CHAMADOS E SUPORTE TÉCNICO COM ASSISTÊNCIA DE IA.

Assis - SP

UNIVERSIDADE PAULISTA

G937198 - MIKAEL FELIX DE BRITO

G9270A5 - DANIEL QUINTANA

R015EI4-MATEUS RODRIGUES GARCIA

DESENVOLVIMENTO DE UM SISTEMA INTEGRADO PARA GESTÃO DE CHAMADOS E SUPORTE TÉCNICO COM ASSISTÊNCIA DE IA.

Projeto Integrador Multidisciplinar do 4º Semestre do curso de Tecnologia em Análise e Desenvolvimento de Sistemas apresentado à Universidade Paulista – UNIP.
Orientadora: Prof. Fabiana Diniz

LISTA DE FIGURA

LISTA DE TABELAS

Sumário

Resumo

Este trabalho apresenta o desenvolvimento de três sistema integrado de gestão de chamados e suporte técnico baseado em inteligência artificial (IA), elaborado como parte do Projeto Integrador Multidisciplinar do curso de Tecnologia em Análise e Desenvolvimento de Sistemas. O objetivo principal é fornecer serviços para empresas que necessitam de suporte em TI, possibilitando a oferta de planos de atendimento por chamados, com o intuito de reduzir falhas de infraestrutura. Com o auxílio da inteligência artificial, busca-se aprimorar as descrições dos problemas e sugerir soluções automatizadas, visando otimizar o atendimento técnico, reduzir o tempo de resposta e diminuir a sobrecarga das equipes de TI. A metodologia adotada incluiu levantamento de requisitos, modelagem orientada a objetos com UML, desenvolvimento de banco de dados relacional com API para a comunicação do banco em PostgreSQL, implementação de algoritmos inteligentes com API do Ollama e criação de uma interface gráfica acessível e responsiva usando as linguagens de Html, Css, C#, Java. O projeto considerou ainda requisitos não funcionais, como segurança da informação e conformidade com a Lei Geral de Proteção de Dados (LGPD). Como resultado, foi desenvolvido três sistema funcionais com as funções automatizadas de atendimento e encaminhamento, além de módulos de relatórios e métricas. A proposta contribui com o avanço de soluções tecnológicas no suporte técnico, promovendo inovação, eficiência operacional e melhor experiência do usuário. A estrutura modular e escalável permite sua aplicação em diferentes contextos corporativos.

Palavras-chave: suporte técnico, inteligência artificial, chamados, automação, sistemas de informação, LGPD.

Abstract

This work presents the development of three integrated systems for ticket management and technical support based on Artificial Intelligence (AI), created as part of the Multidisciplinary Integrative Project in the Technology in Systems Analysis and Development program. The main objective is to provide services for companies that require IT support, enabling the offer of service plans based on ticket management to reduce infrastructure failures. With the aid of AI, the system aims to improve problem descriptions and suggest automated solutions, seeking to optimize technical support, reduce response time, and decrease the workload of IT teams. The adopted methodology included requirements gathering, object-oriented modeling with UML, development of a relational database with an API for communication in PostgreSQL, implementation of intelligent algorithms using the Ollama API, and creation of an accessible and responsive graphical interface using HTML, CSS, C#, and Java. The project also considered non-functional requirements such as information security and compliance with the General Data Protection Law (LGPD). As a result, three functional systems were developed with automated service and routing features, as well as reporting and metrics modules. The proposal contributes to the advancement of technological solutions in technical support, promoting innovation, operational efficiency, and an improved user experience. Its modular and scalable structure allows for application in different corporate contexts.

Keywords: technical support, artificial intelligence, tickets, automation, information systems, LGPD.

1. Introdução

O presente trabalho apresenta o desenvolvimento de três sistemas integrados de gestão de chamados e suporte técnico baseados em inteligência artificial (IA), elaborados como parte do Projeto Integrador Multidisciplinar do curso de Tecnologia em Análise e Desenvolvimento de Sistemas. A proposta surge diante da crescente demanda por soluções tecnológicas que melhorem a eficiência e a precisão no atendimento técnico. Muitas empresas enfrentam dificuldades em gerenciar seus chamados, ocasionando lentidão nos atendimentos, falhas de comunicação e retrabalho. Nesse contexto, a aplicação de IA mostra-se uma alternativa promissora para automatizar processos e aumentar a eficiência operacional (RUSSELL; NORVIG, 2022).

O objetivo principal deste projeto é desenvolver uma solução inteligente voltada para empresas que necessitam de suporte em TI, oferecendo planos de atendimento por chamados e reduzindo falhas de infraestrutura. Com o auxílio da inteligência artificial, busca-se aprimorar a descrição de problemas, sugerir soluções automáticas e otimizar a triagem de chamados, reduzindo o tempo de resposta e a sobrecarga das equipes técnicas. Segundo Pressman e Maxim (2021), a integração de técnicas de engenharia de software com recursos inteligentes permite o desenvolvimento de sistemas mais adaptáveis e eficientes, alinhados às necessidades reais do usuário.

O desenvolvimento contemplou três plataformas complementares que são a web, mobile e desktop com a integração por uma API e um banco de dados relacional. A plataforma web foi destinada à gestão administrativa; a versão mobile, ao registro e acompanhamento de chamados em campo e análises rápidas de métricas; e a desktop, ao uso técnico interno, com recursos avançados de interações entre o cliente e o tecnico. O projeto também considerou requisitos não funcionais, como segurança da informação e conformidade com a Lei Geral de Proteção de Dados (LGPD), conforme previsto pela legislação brasileira (BRASIL, 2018). Dessa forma, o sistema proposto contribui para o avanço de soluções inteligentes no suporte técnico, promovendo inovação, eficiência e melhor experiência para os usuários corporativos.

2. O Ciclo de Vida

O desenvolvimento de software é um processo complexo que segue um ciclo de vida composto por diversas etapas. Como destacam Pressman e Maxim (2016), “o ciclo de vida do software pode ser entendido como um modelo que descreve as fases pelas quais um software passa desde o nacimento até o fim”.

Dentro desse processo, a análise e o projeto são fases cruciais que precedem a codificação. Segundo Sommerville (2011, p. 89), “a análise de requisitos e o projeto do sistema estabelecem uma base sólida para a implementação, reduzindo riscos de falhas futuras”.

A correta atribuição de responsabilidades entre essas duas fases é determinada pela metodologia de desenvolvimento adotada pela equipe ou organização. Como ressalta Royce (1970, p. 2), “cada modelo de processo de software, desde a cascata até os ágeis, reflete diferentes estratégias de divisão e gerenciamento das responsabilidades de desenvolvimento”.

2.1. Classificação das Metodologias de Desenvolvimento

Uma metodologia de desenvolvimento de software consiste em um conjunto estruturado de práticas, processos e ferramentas utilizadas para planejar, gerenciar e controlar a criação de projetos de software. Segundo Pressman e Maxim (2016), o ciclo de vida de um software é composto por fases interligadas que orientam o processo desde a concepção até a manutenção, garantindo maior controle e qualidade no produto final.

As metodologias podem ser amplamente categorizadas em dois grupos principais: prescritivas (tradicionais), focadas em um planejamento detalhado e em etapas bem definidas e sequenciais, como as metodologias de Cascata, Incremental, Prototipação e Espiral e adequadas a projetos com requisitos claros, estáveis e bem definidos desde o início. Já as metodologias ágeis priorizam a flexibilidade, a colaboração contínua com o cliente e a entrega iterativa de software funcional, como o Scrum, Kanban e Extreme Programming (XP), sendo mais apropriadas para projetos com requisitos que mudam com frequência (Sommerville, 2019).

No contexto do projeto orientado a objetos, a fase de Análise concentra-se na modelagem do domínio do problema, buscando responder à pergunta “o quê?” o sistema deve fazer. Esta fase foca nas regras de negócio e na criação de um modelo conceitual, como o diagrama de classes de análise, abstraindo-se de detalhes de implementação. Por sua vez, a fase de Projeto dedica-se à modelagem da solução computacional, respondendo à pergunta “como?” o sistema será construído. O projeto complementa a análise, incorporando decisões sobre a arquitetura, a interface com o usuário, a persistência de dados e os mecanismos de segurança.

3. Projeto de Software

O projeto de software é uma etapa essencial no desenvolvimento de sistemas, pois define a estrutura, o funcionamento e as regras de negócio que orientarão sua construção. Nessa fase, são elaborados diagramas, modelos e especificações que permitem compreender a lógica do sistema e garantir que ele atenda aos requisitos definidos. Segundo Sommerville (2011), o projeto de software fornece a base para a implementação, assegurando que o sistema seja organizado, eficiente e mantenha a qualidade desejada.

Além disso, o projeto contribui para reduzir erros, facilitar a manutenção e melhorar a comunicação entre os membros da equipe de desenvolvimento. De acordo com Pressman (2016), um bom projeto de software atua como um mapa que orienta o processo de desenvolvimento, permitindo transformar as necessidades do usuário em uma solução técnica viável e consistente.

3.1. Arquitetura de Software

Primeiramente a arquitetura de software define a estrutura geral de um sistema, organizando seus componentes, relações e padrões de comunicação. Ela serve como base para o desenvolvimento, garantindo que o sistema seja coeso, escalável e de fácil manutenção.

Segundo Sommerville (2019), a arquitetura fornece uma visão de alto nível que orienta o design e facilita o entendimento do sistema. Para Pressman e Maxim (2021), uma boa arquitetura reduz a complexidade e promove o reuso de componentes, assegurando qualidade e consistência no projeto.

Em síntese, a arquitetura de software é o alicerce que orienta o desenvolvimento e sustenta o funcionamento do sistema.

Existindo as seguinte arquiteturas como mais ultilizadas no mercado hoje, o mercado favorece microserviços e arquiteturas em camadas, combinadas com soluções em nuvem, por oferecerem flexibilidade, escalabilidade e manutenção facilitada.

Arquitetura em Microserviços: o sistema é composto por vários serviços independentes, cada um com uma função específica e executando de forma autônoma. Segundo Newman (2015), a arquitetura de microserviços promove o desenvolvimento de aplicações como um conjunto de pequenos serviços, cada um executando em seu próprio processo e comunicando-se por meio de APIs leves. Essa abordagem permite maior escalabilidade, flexibilidade e facilidade na implementação de atualizações isoladas.

Arquitetura em Camadas: Arquitetura em Camadas é um modelo de organização de software que divide o sistema em níveis (camadas), cada um com responsabilidades específicas e bem definidas. Essa estrutura permite separar a lógica de apresentação, de negócio e de dados, facilitando o desenvolvimento, manutenção e evolução do sistema.

Geralmente, o **sistema é dividido em três ou mais camadas**, sendo:

Camada de Apresentação: responsável pela interação com o usuário (interface gráfica ou web);

Camada de Negócio: contém as regras e processos principais do sistema;

Camada de Dados: realiza o acesso, armazenamento e manipulação das informações no banco de dados.

O funcionamento ocorre de forma hierárquica, onde uma camada depende apenas da camada inferior, garantindo baixo acoplamento e alta coesão. Segundo Pressman e Maxim (2021), essa abordagem promove melhor organização e reutilização de código, além de facilitar testes e manutenção. Um exemplo visual do estilo de diagrama mais aplicável do modelo em camada na imagem 1.

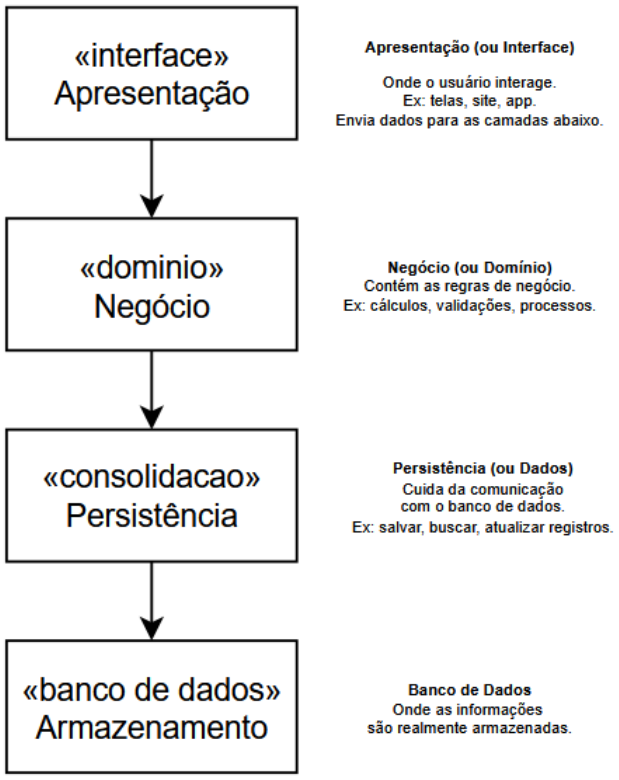

Figura 1: Diagrama de Arquitetura de camadas
Fonte: autoria própria (2025)

neste projeto, foi utilizada a arquitetura em camadas para estruturar o sistema de forma organizada e modular. Essa abordagem permitiu separar as responsabilidades em diferentes níveis, facilitando a manutenção, o reuso de componentes e futuras expansões.

O sistema foi dividido em três camadas principais espelhado no modelo de programação MVC:

Camada de Apresentação: responsável pela interação com o usuário, onde ocorre a entrada e exibição de informações por meio da interface.

Camada de Negócio: responsável pelas regras e lógicas do sistema, realizando o processamento das informações e a comunicação entre as demais camadas.

Camada de Dados: encarregada do acesso, armazenamento e manipulação das informações no banco de dados, garantindo segurança e integridade.

Segundo Pressman e Maxim (2021), a arquitetura em camadas promove a separação de responsabilidades, reduz o acoplamento entre os componentes e aumenta a clareza estrutural do sistema, facilitando a manutenção e evolução do software. Essa divisão proporcionou um melhor controle sobre as modificações,

evitando impactos em todo o sistema quando ajustes são feitos em partes específicas, além de tornar o desenvolvimento mais organizado e eficiente.

3.2. Ferramentas de Modelagem UML

Para apoiar o processo de projeto de software, diversas ferramentas e linguagens de modelagem são empregadas para criar representações visuais e estruturadas do sistema.

Entre as ferramentas mais conhecidas estão o Enterprise Architect, Rational Rose, Visual Paradigm, StarUML, Astah e a ferramenta online Draw.io.

Nesse processo de desenvolvimento dos diagramas de classes e layout, será utilizada a ferramenta online Draw.io para criar os diagramas visualmente. Por temos mais pratica e familiaridade na ultilização da ferramenta.

3.3. A Importância da UML

A Linguagem de Modelagem Unificada (UML) é o padrão da indústria para a modelagem de sistemas de software orientados a objetos. Como afirmam Booch, Rumbaugh e Jacobson, “a UML é uma linguagem de modelagem para especificar, visualizar, construir e documentar os artefatos de um sistema de software” (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 3).

É importante ressaltar que a UML é uma linguagem de notação visual, e não uma linguagem de programação ou uma metodologia de desenvolvimento. Ela pode ser integrada a diversos processos, como o RUP (Rational Unified Process), Scrum ou XP; Sendo uma prática indispensável para projetos de média e alta complexidade.

Segundo Larman, “a modelagem auxilia a compreender e comunicar ideias complexas, fornecendo diferentes perspectivas do sistema que se complementam” (LARMAN, 2007, p. 27).

Dado que a capacidade humana de gerenciar um grande volume de informações é limitada e que os sistemas de software estão em constante evolução, seja por mudanças na legislação, no mercado ou por novas solicitações de clientes, a modelagem torna-se fundamental para documentar o conhecimento sobre o sistema,

garantindo sua manutenibilidade e evolução, e evitando a dependência de indivíduos específicos.

3.4. Diagramas da UML

A UML fornece um conjunto rico de diagramas para capturar diferentes visões de um sistema. A seguir, são detalhados os diagramas para o projeto.

3.4.1. O Modelo de Objetos e o Diagrama de Classes

O modelo de objetos é o coração da abordagem orientada a objetos e busca capturar a estrutura estática de um sistema. O diagrama de classes é a principal ferramenta para representar este modelo. Como afirmam Booch, Rumbaugh e Jacobson, “os diagramas de classes são os principais blocos de construção da UML, mostrando a estrutura estática de classes, seus atributos, operações e os relacionamentos entre objetos” (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 125).

Objeto: é uma instância de uma classe, representando um conceito, uma abstração ou uma entidade do mundo real com limites e significados bem definidos. Todo objeto possui um estado (seus dados ou atributos) e um comportamento (suas ações ou métodos). Segundo Pressman e Maxim (2021), os objetos encapsulam dados e operações relacionadas, permitindo maior modularidade e reutilização no desenvolvimento de software.

Classe: funciona como um “modelo” que define os atributos e métodos comuns a todos os objetos de um mesmo tipo. Conforme Sommerville (2019), as classes descrevem a estrutura e o comportamento de conjuntos de objetos semelhantes, servindo como base para a criação e organização dos elementos de um sistema orientado a objetos.

3.4.1.1. Perspectivas do Diagrama de Classes

O **Diagrama de Classes:** o diagrama de classes é um artefato que evolui ao longo do ciclo de vida do desenvolvimento, podendo ser visto sob diferentes perspectivas. Segundo Booch, Rumbaugh e Jacobson (2006), o diagrama de classes é um dos

principais elementos da UML, pois descreve a estrutura estática de um sistema, mostrando as classes, seus atributos, operações e os relacionamentos entre elas.

Análise (Modelo Conceitual): representa as classes do domínio do problema, focando nos conceitos de negócio e seus relacionamentos, sem se preocupar com detalhes de implementação.

Projeto (Modelo de Especificação): adiciona detalhes técnicos ao modelo de análise, como tipos de dados para atributos e assinaturas completas de métodos, alinhando o modelo à tecnologia que será utilizada.

Implementação: corresponde à tradução final do diagrama de classes de projeto para o código-fonte em uma linguagem de programação específica.

Os relacionamentos no Diagrama de Classes podem ocorrer de diversas maneiras para formar a estrutura do sistema:

Associação: representa um relacionamento estrutural entre objetos de diferentes classes. A multiplicidade (ou cardinalidade) especifica quantos objetos de uma classe podem se relacionar com objetos da outra.

Agregação: é um tipo especial de associação que representa uma relação “todo-parte”, onde a parte pode existir independentemente do todo. É representada por um losango não preenchido.

Composição: é uma forma mais forte de agregação, onde a parte tem seu ciclo de vida atrelado ao do todo, ou seja, não pode existir sem ele. É representada por um losango preenchido.

Herança: permite que uma classe (subclasse) herde os atributos e métodos de outra classe (superclasse), promovendo o reuso de código e a criação de hierarquias de especialização.

Segundo Sommerville (2019), a modelagem por meio de diagramas, como o de classes, contribui para a compreensão e documentação da estrutura do sistema, facilitando a comunicação entre analistas e desenvolvedores.

Um exemplo prático do projeto para o diagrama de classes completo, que contempla suas classes, atributos, métodos e seus relacionamentos, pode ser observado na Figura 2.

3.4.1.2. Diagrama de Classe

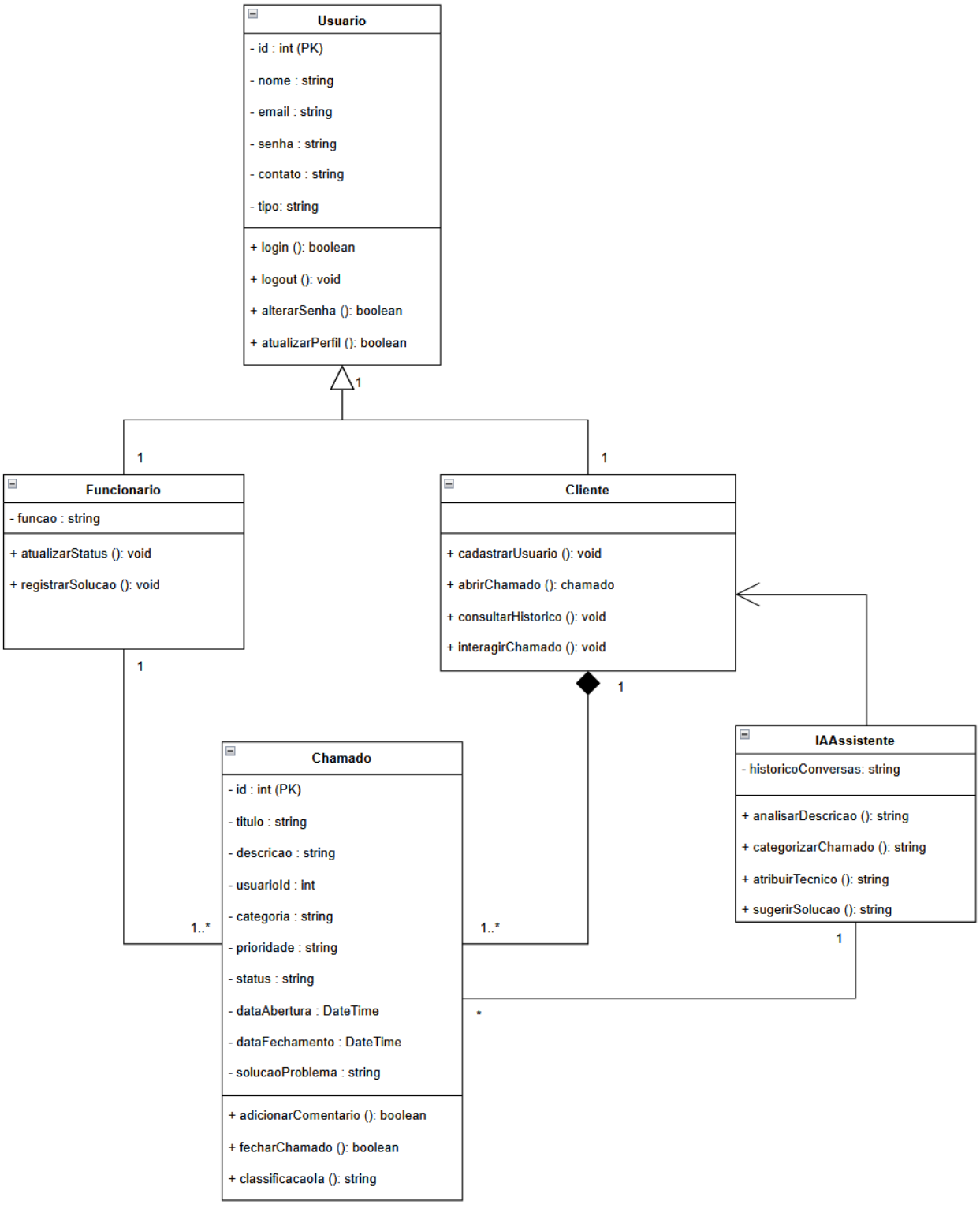

Figura 2: Diagrama de Classes
Fonte: Autoria própria (2025)

3.4.2. Diagrama de Estado de Navegação

Diagrama de Estado de Navegação: o diagrama de estado de navegação representa o fluxo e as transições entre diferentes telas, páginas ou estados do sistema, descrevendo como o usuário interage com a aplicação e como o sistema responde a essas ações. Ele é amplamente utilizado em sistemas com interface gráfica ou web, permitindo visualizar o comportamento dinâmico da navegação.

Esse diagrama contém:

Estados: representam as telas, páginas ou etapas do sistema acessadas pelo usuário;

Transições: indicam o caminho percorrido entre os estados, geralmente disparadas por eventos, cliques ou ações do usuário;

Eventos: são as ações que provocam a mudança de um estado para outro (como “Login”, “Salvar” ou “Voltar”);

Condições e ações: descrevem regras ou respostas específicas durante a transição entre estados.

Seu funcionamento consiste em mapear a navegação e o comportamento do sistema, mostrando como o usuário percorre as funcionalidades e como o sistema reage a cada interação.

De acordo com Booch, Rumbaugh e Jacobson (2005), os diagramas de estado permitem compreender o comportamento dinâmico de um sistema, ajudando a identificar possíveis inconsistências na navegação e aprimorar a experiência do usuário.

Aplicando esse diagrama no projeto podemos representar as telas e suas funcionalidades assim representado na figura 3 o diagrama de navegação fica nítido ver as funções do sistema.

3.4.2.1. Estado de navegação Web:

Na Figura 3 é apresentado o Diagrama de Navegação da versão Web do sistema HelpDesk ADS. Ele demonstra o fluxo de telas e as interações possíveis dentro da aplicação, destacando como o usuário navega entre as principais funcionalidades do sistema.

O fluxo inicia na tela Login, onde ocorre a autenticação do usuário. Após o acesso, o sistema direciona para a tela Gerenciar Chamados, que atua como painel principal de controle, possibilitando visualizar, filtrar, editar, excluir e acompanhar chamados.

A partir dessa tela central, o usuário pode navegar para Criar Chamado, responsável por registrar novos chamados, ou Editar Chamado, que permite modificar informações e atualizar o andamento de solicitações. O diagrama também inclui telas complementares como Gerenciar Usuários, Criar Usuário, Editar Usuário, Excluir Usuário, Sobre Nós e Conversa com IA, que ampliam as funcionalidades e a gestão administrativa da plataforma.

As setas do diagrama indicam as transições entre as telas, representando as ações do usuário, como cliques, confirmações e retornos, que provocam a mudança de estado dentro do sistema. Dessa forma, o diagrama evidencia a navegação clara, lógica e responsiva, reforçando a usabilidade e eficiência da versão Web do sistema HelpDesk ADS.

Figura 3: Diagrama de Estado de navegação Web

Fonte: Autoria própria (2025)

3.4.2.2. Estado de navegação do Desktop:

O diagrama de navegação apresentado foi desenvolvido no Draw.io e representa o fluxo de telas da versão desktop do sistema de chamados. Ele ilustra como o usuário interage com as principais funcionalidades do sistema, evidenciando as possíveis transições entre as telas.

O fluxo inicia na tela Login, onde o usuário realiza sua autenticação no sistema. Após o acesso, ele é direcionado à tela Gerenciar Chamado, que serve como painel principal, permitindo visualizar, filtrar e administrar os chamados existentes.

A partir dessa tela, o usuário pode seguir para Criar Chamado, onde é possível registrar um novo chamado, ou acessar Editar Chamado, para alterar informações e atualizar o status de um chamado em andamento.

As setas indicam o caminho da navegação entre as telas, mostrando de forma clara as ações possíveis de ida e retorno. Dessa forma, o diagrama evidencia o fluxo lógico e intuitivo de navegação do sistema, destacando a simplicidade e eficiência na interação entre as principais telas do ambiente desktop representado na figura 4.

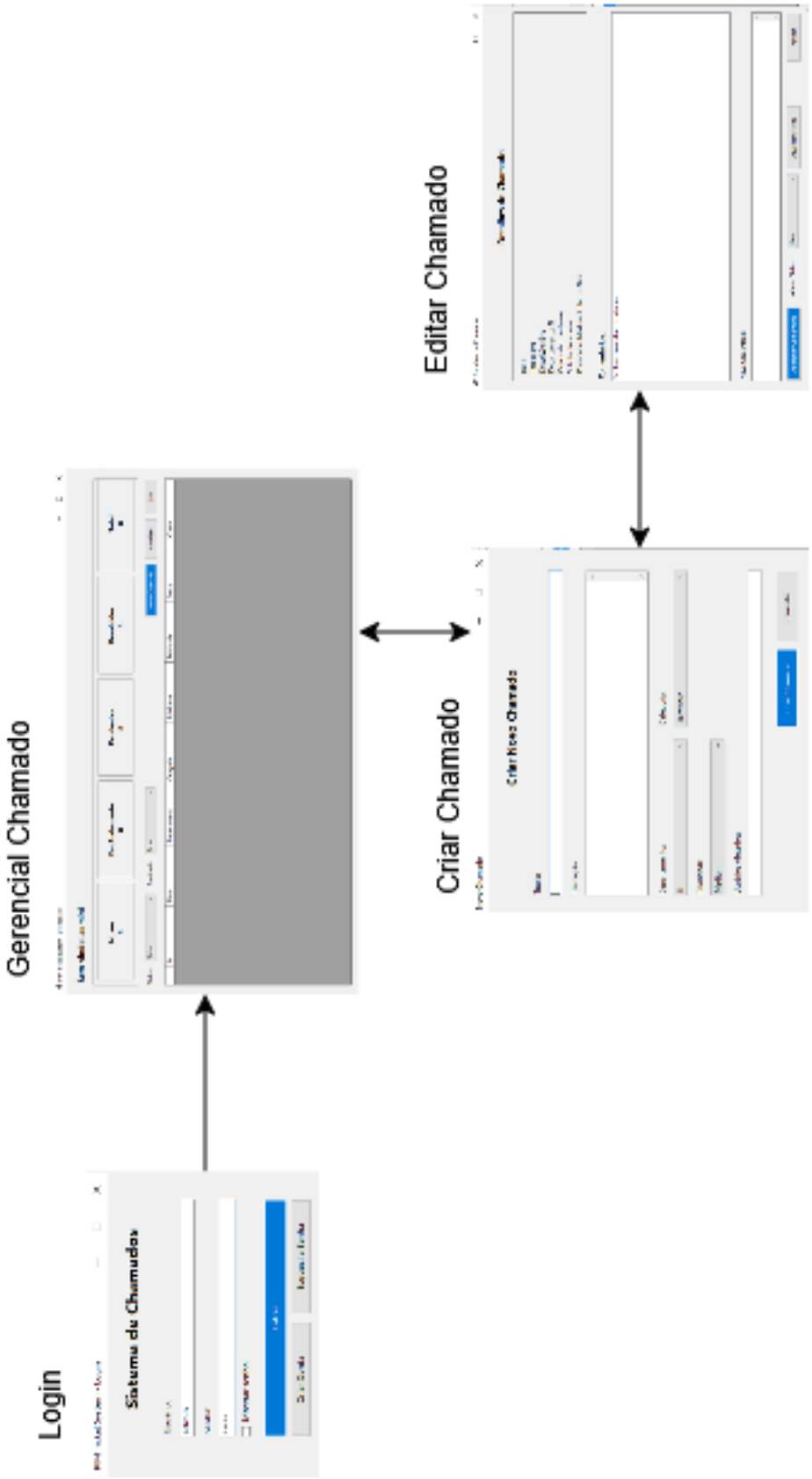

Figura 4: Diagrama de Estado de navegação Desktop
Fonte: Autoria própria (2025)

3.4.2.3. Estado de navegação do Mobile:

O diagrama de estados de navegação apresentado demonstra o fluxo de telas do aplicativo HelpDesk ADS em sua versão mobile, evidenciando como o usuário interage com o sistema e para quais telas ele pode ser direcionado de acordo com suas ações.

O fluxo inicia-se na tela de Login, que funciona como estado inicial do sistema. Após autenticação bem-sucedida, o usuário é encaminhado à área principal do aplicativo, que pode ser a tela de Chamados Usuário ou Chamados Admin, dependendo do perfil de acesso.

A partir dessas telas centrais, o usuário pode navegar para múltiplos estados/telas específicos, como:

Menus Chamados, Criar Chamado, Buscar Chamados, Histórico de Atividades, Gerenciamento de Tags, Troca de Tema, Conscientização e Diversidade, e Dashboard Analítico.

Cada seta apresentada no diagrama representa a transição entre telas, ocorrendo sempre por meio de uma ação do usuário, como seleção de menu lateral, botão de criação, filtro, seleção de item ou retorno para tela anterior.

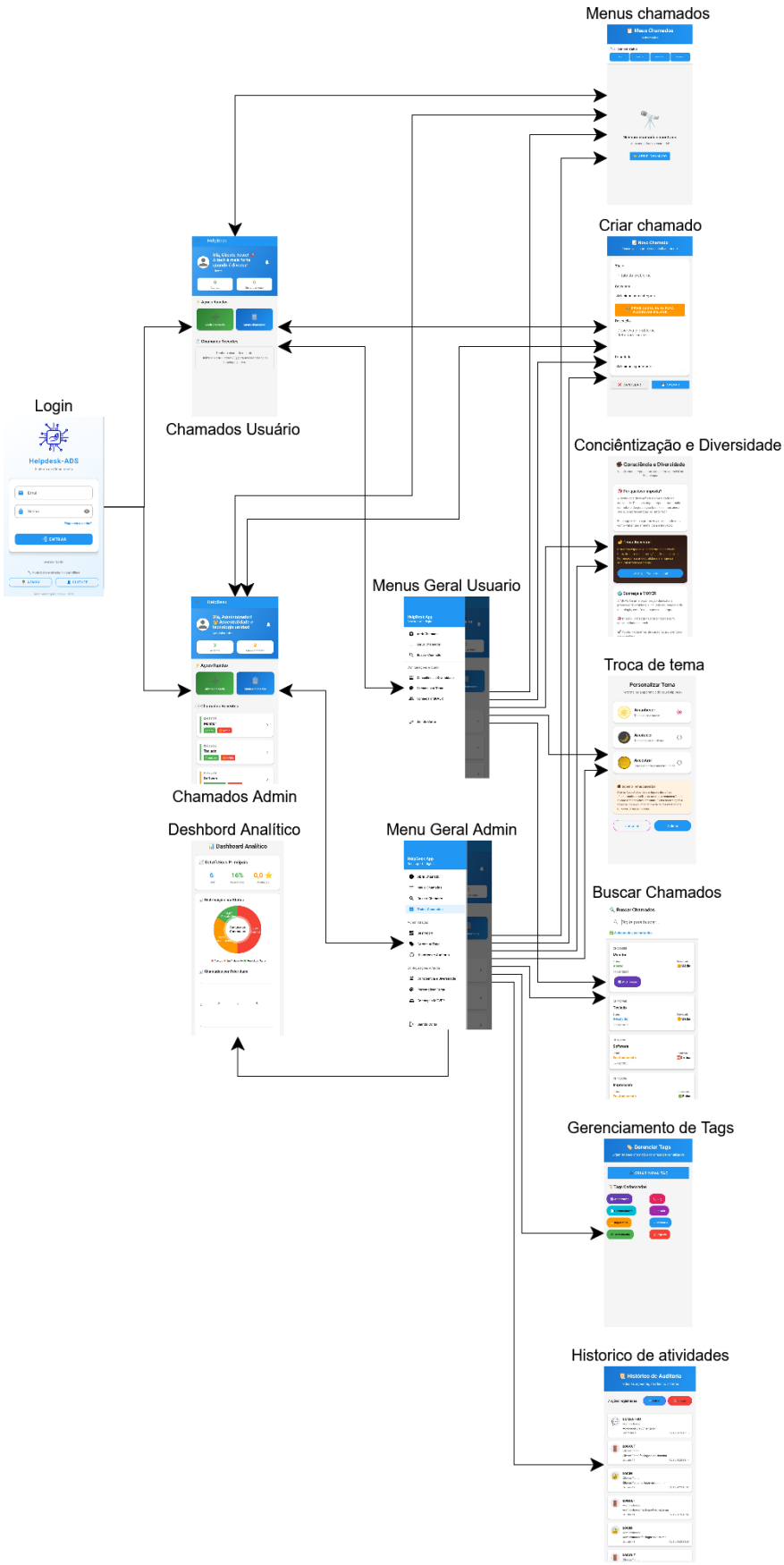

Figura 5: Diagrama de Estado de navegação Mobile
Fonte: Autoria própria (2025)

Assim, o diagrama evidencia o comportamento dinâmico do sistema e como cada funcionalidade está relacionada dentro da navegação geral. Ele permite identificar claramente quais caminhos o usuário pode percorrer dentro da aplicação e como as telas estão conectadas entre si.

Dessa forma, o diagrama serve como referência visual para compreensão da estrutura da navegação, apoiando o entendimento técnico da arquitetura de usabilidade do aplicativo HelpDesk ADS.

4. Fundamentação e Arquitetura (POO)

A Programação Orientada a Objetos (POO) não é apenas uma matéria, mas o alicerce para construir **sistemas que resistem ao tempo**. É a evolução do nosso raciocínio, aplicando pilares como Encapsulamento, Herança, Polimorfismo e Abstração para criar códigos robustos, escaláveis e, acima de tudo, fáceis de dar manutenção.

Autores renomados, como Deitel & Deitel (2016), nos mostram que a arquitetura orientada a objetos permite que modelemos as entidades do mundo real de maneira muito mais fiel. Na nossa realidade, utilizamos essa clareza para garantir a responsividade e a reutilização que Furgeri (2016) tanto valoriza.

No desenvolvimento do **HelpDeskApp**, esses conceitos de POO II foram aplicados de forma integral. Isso guiou desde o desenho inicial de cada classe até a separação clara de responsabilidades, garantindo que os módulos do sistema conversem entre si de maneira eficiente.

4.1. Tecnologias Utilizada

- **Mobile:** Java 8, arquitetura MVC, Android Studio, SQLite.
- **Web (Backend):** ASP.NET Core (C#), HTML5/CSS3, JavaScript.
- **Desktop:** C#, Windows Forms / WPF, Visual Studio.
- **Banco de Dados:** MS SQL Server (Web/Desktop), SQLite (Mobile).

4.2. Aplicações de POO no HelpDeskApp

A Coluna Vertebral: Arquitetura MVC (Model-View-Controller)

No aplicativo mobile, a separação de responsabilidades é palpável fazendo com que aumente a agilidade. A lógica funciona assim:

- **Modelos:** Representam o "o quê" do sistema (usuário, chamados, comentários), sendo as classes de dados puras (*Ex.: Usuario.java, Chamado.java*).
- **Views:** São as interfaces, o "rosto" do app (telas/Activities do Android), com seus respectivos layouts XML.
- **Controllers:** São as Activities e classes auxiliares que fazem a ponte, interligando as ações do usuário (View) aos dados (Modelos), garantindo que o fluxo seja lógico.

4.2.1. Organização do Fluxo (Exemplo no Código Java)

A figura 6 mostra uma classe que representa o modelo de dados de um chamado no sistema HelpDesk. Ela encapsula três propriedades principais:

- **id:** identificador único do chamado no banco de dados
- **descricao:** texto descritivo do problema ou solicitação
- **status:** situação atual do chamado (aberto, em andamento, fechado, etc.)

Os métodos getters e setters (não mostrados) permitem acesso controlado a esses atributos, seguindo o princípio do encapsulamento da POO. Esta classe é fundamental para a camada Model da arquitetura MVC, representando a estrutura dos dados que serão manipulados pelo sistema.

```
public class Chamado {
    private int id;
    private String descricao;
    private String status;
    // Getters/Setters
}
```

Figura 6: Organização do Fluxo

Fonte: Autoria própria (2025)

A figura 7 mostra uma classe que atua como Controller na arquitetura MVC. Ela herda de AppCompatActivity (classe base do Android para telas) e tem a responsabilidade de:

- Receber interações do usuário através da interface gráfica

- Utilizar o ChamadoDAO (Data Access Object) para realizar operações de banco de dados
- Atualizar a View com os dados obtidos do Model
- Coordenar o fluxo de informações entre a interface do usuário e a camada de dados

```
public class ChamadosActivity extends AppCompatActivity {
    // Usamos o ChamadoDAO para buscar/atualizar os dados
}
```

Figura 7: Organização do Fluxo

Fonte: Autoria própria (2025)

4.2.2. Desacoplamento de Dados com o Padrão DAO (Data Access Object)

Para que a lógica de negócio do aplicativo não dependa de qual banco de dados foram usados, foram criadas classes de persistência que isolam esse acesso.

Como mostra a figura 8 esta classe implementa o padrão DAO (Data Access Object), que isola toda a lógica de acesso ao banco de dados

```
public class ChamadoDAO {
    private SQLiteDatabase db;

    public ChamadoDAO(Context ctx) {
        HelpdeskDB helper = new HelpdeskDB(ctx);
        this.db = helper.getWritableDatabase();
    }

    public List<Chamado> listarChamados() {
        // Consulta ao banco, cria lista de Chamado
    }
    // Outros métodos: inserir, atualizar, excluir
}
```

Figura 8: Desacoplamento de Dados com o Padrão DAO

Fonte: Autoria própria (2025)

Essa abordagem, defendida por Deitel & Deitel (2016), nos deu flexibilidade; se precisarmos mudar a tecnologia do banco no futuro, o impacto nas Activities é mínimo.

4.2.3. Herança, Polimorfismo e Abstração em Ação

Utilizamos esses pilares em toda a hierarquia de usuários e na criação de *Adapters* para as listas dinâmicas:

- **Herança/Polimorfismo:** A hierarquia de usuários é um exemplo clássico.

Como mostra a figura 9 este código demonstra os conceitos de herança, abstração e polimorfismo:

Classe abstrata Usuario:

- Define um atributo comum 'nome' para todos os tipos de usuário
- Declara o método abstrato exibirPerfil(), forçando as subclasses a implementá-lo
- Não pode ser instanciada diretamente, servindo apenas como modelo

UsuarioComum e UsuarioTecnico:

- Herdam de Usuario, recebendo automaticamente o atributo 'nome'
- Implementam exibirPerfil() de maneira específica para cada tipo
- UsuarioComum pode ter uma tela de perfil básica
- UsuarioTecnico pode ter acesso a informações técnicas adicionais

Polimorfismo:

- Uma variável do tipo Usuario pode referenciar tanto UsuarioComum quanto UsuarioTecnico
- Ao chamar exibirPerfil(), o Java automaticamente executa a versão correta do método

```java
// Exemplo simples de herança
public abstract class Usuario {
    private String nome;
    public abstract void exibirPerfil();
}

public class UsuarioComum extends Usuario {
    @Override
    public void exibirPerfil() { /* Lógica específica */ }
}

public class UsuarioTecnico extends Usuario {
    @Override
    public void exibirPerfil() { /* Lógica específica */ }
}
```

Figura 9: Herança/Polimorfismo

Fonte: Autoria própria (2025)

O Polimorfismo nos permite chamar exibirPerfil() de forma dinâmica, dependendo de quem está logado – um ganho em elegância e manutenção.

- **Adapters:** Herdamos de RecyclerView.Adapter para gerenciar diferentes tipos de listagens de forma otimizada.

Como mostra a Figura 10 esta classe demonstra herança e polimorfismo no contexto de interfaces gráficas Android:

Herança de RecyclerView.Adapter:

- RecyclerView é o componente do Android para exibir listas eficientes
- Adapter é responsável por converter dados em elementos visuais
- ChamadoAdapter herda funcionalidades da classe base e adapta para chamados específicos

Tipo genérico <ViewHolder>:

- Define que cada item da lista terá uma estrutura ViewHolder
- ViewHolder mantém referências aos componentes visuais, otimizando performance

Override de métodos:

- **onCreateViewHolder():** cria a estrutura visual de cada item
- **onBindViewHolder():** preenche cada item com dados do chamado
- **getItemCount():** retorna quantos chamados existem na lista

Polimorfismo aplicado:

- Diferentes tipos de chamados podem ter layouts diferentes
- O adapter decide dinamicamente qual layout usar baseado no tipo/status

```
public class ChamadoAdapter extends RecyclerView.Adapter<ChamadoAdapter.ViewHolder:
    // Implementamos os métodos para lidar com diferentes views
}
```

Figura 10: Adapters

Fonte: Autoria própria (2025)

4.2.4. Abstração e Responsabilidade Única (SRP)

Para evitar as temidas classes "Deus" (que fazem de tudo), foi aplicado o **Princípio da Responsabilidade Única (SRP)**. Isso resultou na criação de Helpers (auxiliares) específicos.

Como mostra na Figura 11 esta classe exemplifica o princípio SRP (Single Responsibility Principle) - Princípio da Responsabilidade Única.

```java
public class NotificationHelper {
    public static void enviarNotificacao(Context ctx, String mensagem) {
        // Lógica de notificação push/local centralizada
    }
}
```

Figura 11: Helpers (auxiliares)

Fonte: Autoria própria (2025)

Classes como PDFHelper.java e NotificationHelper.java resolvem um domínio específico, simplificando o código principal.

4.2.5. Gestão de Sessão com Singleton

A Figura 12 mostra que para garantir que a sessão do usuário logado seja centralizada e única em todo o aplicativo, foi utilizado o padrão **Singleton**:

```java
public class SessionManager {
    private static SessionManager instancia;
    private Usuario usuarioLogado;

    private SessionManager() { } // Construtor privado

    public static SessionManager getInstance() {
        if (instancia == null) {
            instancia = new SessionManager();
        }
        return instancia;
    }
    // Métodos para get/set do usuário logado
}
```

Figura 12: Gestão de Sessão com Singleton

Fonte: Autoria própria (2025)

4.3. Benefícios e Alinhamento com o Mercado

A adoção desses padrões não é um luxo, mas uma necessidade. Eles nos entregam:

- Um **código limpo, seguro e com manutenção simplificada**;
- **Facilidade na criação de testes unitários**;
- O alinhamento com as **melhores práticas** exigidas pelo mercado de desenvolvimento Android e de sistemas corporativos.

5. Destaques Arquitetônicos e Padrões Avançados

O **HelpDesk ADS** nasceu com uma forte engenharia de software, incorporando padrões reconhecidos internacionalmente para garantir não apenas funcionalidade, mas também sustentabilidade.

5.1. Arquitetura Principal do HelpDesk ADS

- **MVC:** Garante a separação total de responsabilidades, facilitando expansão e manutenção.
- **DAO:** Isola a lógica de dados, permitindo a troca de tecnologia de banco de dados sem afetar as demais camadas.
- **Singleton:** Gerencia o estado global de forma segura (sessão de usuário, tema visual).
- **Design Inclusivo:** Estrutura visual moderna baseada em *Material Design Components*, incluindo temas com representatividade (como o tema *Ancestral*, que é afrocêntrico).

Conforme nosso README, o objetivo é ser um "sistema robusto e completo" com uma "Interface intuitiva" que valoriza a "representatividade étnico-racial".

5.2. Padrões Utiliizados no Código

- **Singleton** — Centralização de Sessão e Tema O SessionManager é o guardião das permissões. Assim que o usuário faz login, ele mantém o controle de quem está ativo, garantindo que as permissões sejam aplicadas corretamente em todas as Activities.

  Classes como UsuarioDAO e ChamadoDAO encapsulam a lógica de acesso ao banco. Isso maximiza a testabilidade e garante o desacoplamento das interfaces do usuário.
- **Adapters** — O Polimorfismo nas Listagens O uso intenso de RecyclerView (e seus *Adapters*) e CardView permite a exibição dinâmica de chamados e comentários, sendo um exemplo prático e eficiente de polimorfismo no *front-end* mobile.
- **Utilitários (Helpers) e Responsabilidade Única** Ao criarmos classes como PDFHelper, NotificationHelper e AuditoriaHelper, evitamos que a lógica de negócio principal seja poluída por tarefas secundárias, tornando cada módulo mais focado e reutilizável.

- **Multiplataforma — Planejamento para o Futuro** O código-fonte mobile em Java/Android Studio já foi planejado para se integrar com backends em ASP.NET Core (C#) e MS SQL Server, com a comunicação sendo feita via API RESTful – garantindo a interoperabilidade entre mobile, web e desktop.

5.3. Tendências e Recursos Especiais

- **IA e Automatização:** Já há um módulo para simular (ou planejar) respostas automáticas a FAQs, agilizando o atendimento com uso de técnicas de NLP (Processamento de Linguagem Natural).
- **Relatórios Profissionais:** Implementamos a exportação avançada de dados usando a biblioteca *iTextPDF*, uma referência no mercado Java para geração de documentos.

5.4. Tecnologias e Ferramentas Principais

- **IDE/Framework:** Android Studio; Material Design; Visual Studio Community.
- **Banco de Dados:** SQLite (Mobile); MS SQL Server (Planejado para backend).
- **Bibliotecas de Destaque:** RecyclerView, CardView (UI interativa), iTextPDF (PDF), MPAndroidChart (gráficos), Material Components (UX).
- **Controle de Versão:** Github.

6. Gerenciamento De Projetos De Software

O gerenciamento de projetos de software é uma área crítica no desenvolvimento de tecnologia, pois envolve alto grau de complexidade, mudanças constantes de requisitos e forte pressão por entregas rápidas e contínuas (PRESSMAN; MAXIM, 2021). Conforme o PMI (2021), gerenciar projetos significa aplicar conhecimento, habilidades, ferramentas e técnicas para atingir objetivos dentro de restrições como tempo, custo e qualidade. No contexto de software, esse desafio torna-se ainda maior, pois o produto é intangível, evolui com frequência e depende de equipes multidisciplinares que precisam trabalhar de maneira integrada e colaborativa (SOMMERVILLE, 2019).

Assim, o gerenciamento de projetos de software envolve planejar, organizar, liderar e controlar recursos para que o sistema seja entregue com valor, funcionando e atendendo às necessidades do usuário final, considerando que as mudanças são

inevitáveis e fazem parte do ciclo de desenvolvimento (PMI, 2021). Segundo Anderson (2010), metodologias ágeis como Scrum e Kanban surgiram justamente para lidar diretamente com essas variações, priorizando adaptação constante, ciclos curtos e entrega contínua de valor. Dessa forma, o gerenciamento de projetos torna-se um elemento essencial para garantir qualidade, previsibilidade e eficiência no desenvolvimento de soluções de software modernas.

6.1. Etapas do Gerenciamento de Projetos de Software

O gerenciamento de projetos de software segue um ciclo de vida que pode variar conforme a metodologia utilizada, mas geralmente envolve etapas bem definidas para garantir organização, previsibilidade e qualidade do produto final. Entre essas etapas estão: iniciação, com a definição de objetivos, escopo, stakeholders e riscos iniciais; planejamento, com elaboração de cronogramas, alocação de recursos, definição de marcos e análise de riscos; execução, onde o desenvolvimento do software ocorre de fato, incluindo comunicação, colaboração e solução de problemas; monitoramento e controle, voltado ao acompanhamento do progresso, qualidade e orçamento, ajustando planos quando necessário; e encerramento, que envolve a entrega do produto, avaliação dos resultados, lições aprendidas e formalização do término do projeto. Segundo o PMI (2021), essas fases formam a base do ciclo de vida de projetos, independentemente da abordagem adotada (ágil, híbrida ou tradicional).

6.2. Ferramentas de Suporte

Trello: Trello é uma ferramenta de gerenciamento de tarefas e projetos baseada no método Kanban, que utiliza quadros visuais (boards), listas e cartões (cards) para organizar atividades. É amplamente utilizada por equipes de software para rastrear progresso, priorizar tarefas e colaborar de forma intuitiva.

Funcionalidades Principais

- **Boards**: Representam projetos ou fluxos de trabalho. Um board pode ser configurado para um projeto de software, com listas como "A Fazer", "Em Progresso" e "Concluído".

- **Cards**: Cada card representa uma tarefa ou item de trabalho. Dentro de um card, é possível adicionar descrições, checklists, anexos, prazos e membros responsáveis.
- **Listas**: Agrupam cards em categorias ou estágios do fluxo de trabalho, permitindo uma visualização clara do status das tarefas.
- **Integrações**: Trello se integra a ferramentas como Slack, Jira e Google Drive, facilitando a comunicação e o compartilhamento de arquivos.
- **Power-Ups**: Extensões que adicionam funcionalidades, como calendários, automações (via Butler) e relatórios.

  Fonte: Site oficial do Trello (trello.com) e Atlassian Blog (atlassian.com/blog). Esses recursos fornecem informações sobre funcionalidades e casos de uso do Trello em projetos ágeis.

Grafana é uma plataforma de código aberto para monitoramento e visualização de dados, amplamente utilizada para criar dashboards interativos que ajudam a analisar métricas de desempenho em tempo real. Em projetos de software, é frequentemente empregada para monitorar sistemas, infraestrutura e até o progresso de equipes.

Funcionalidades Principais

- **Dashboards Personalizáveis**: Permite criar painéis visuais com gráficos, tabelas e alertas baseados em dados de diversas fontes.
- **Integração com Fontes de Dados**: Suporta bancos de dados como Prometheus, InfluxDB, MySQL e APIs, permitindo coletar métricas de servidores, aplicações ou ferramentas de CI/CD.
- **Alertas**: Configuração de notificações (via e-mail, Slack ou outros canais) quando métricas ultrapassam limites definidos, como alta latência em um sistema.
- **Exploração de Dados**: Ferramentas para análise detalhada, como zoom em gráficos e filtros temporais.

Miro é uma ferramenta online de colaboração visual que permite a criação de quadros digitais para brainstorming, planejamento e design de fluxos de trabalho. É

frequentemente usada por equipes de software para mapear processos, criar wireframes ou planejar sprints.

Funcionalidades Principais

- **Quadros Infinitos**: Espaço virtual ilimitado para adicionar post-its, diagramas, imagens e outros elementos visuais.
- **Colaboração em Tempo Real**: Vários usuários podem editar o mesmo quadro simultaneamente, com cursores visíveis e chat integrado.
- **Modelos Pré-definidos**: Templates para mapas mentais, retrospectivas ágeis, diagramas de fluxo e wireframes de UI/UX.
- **Integrações**: Conexão com ferramentas como Slack, Trello e Jira, permitindo incorporar cards ou notificações diretamente nos quadros.

Slack é uma plataforma de comunicação em equipe que centraliza mensagens, arquivos e notificações em canais organizados. É amplamente adotada em projetos de software para facilitar a colaboração entre desenvolvedores, gerentes e stakeholders.

Funcionalidades Principais

- **Canais**: Espaços para discussões temáticas (ex.: #desenvolvimento, #suporte), que podem ser públicos ou privados.
- **Mensagens Diretas e Grupos**: Comunicação individual ou em pequenos grupos para assuntos específicos.
- **Integrações**: Conexão com ferramentas como Trello, Jira, GitHub e Google Drive, permitindo notificações automáticas (ex.: novo commit no repositório ou tarefa concluída).
- **Bots e Automação**: Capacidade de criar bots personalizados ou usar apps para automatizar tarefas, como lembretes de reuniões.
- **Chamadas de Voz e Vídeo**: Funcionalidade para reuniões rápidas sem sair da plataforma.

6.3. Tabela de ferramentas:

Ferramenta	Foco Principal	Melhor Uso em Projetos de Software	Integração entre Si
Trello	Gestão de tarefas (Kanban)	Rastreamento de tarefas e sprints	Integra com Slack para notificações
Grafana	Monitoramento de desempenho	Análise de métricas de sistemas e equipes	Pode enviar alertas ao Slack
Miro	Colaboração visual	Brainstorming, retrospectivas e design de fluxos	Integra com Slack e Trello
Slack	Comunicação em equipe	Centralização de discussões e notificações	Integra com Trello, Miro e Grafana

Tabela 1: Ferramentas
Fonte: Ferramentas para Metologias ageis (2016)

6.4. Metodologias Ágeis

As metodologias ágeis surgiram para lidar com a necessidade de flexibilidade e entregas rápidas, baseando-se no **Manifesto Ágil** (2001), que prioriza indivíduos e interações, software funcional, colaboração com o cliente e resposta a mudanças.

- **Scrum**: Framework iterativo baseado em sprints (ciclos de 2-4 semanas). Possui papéis definidos como Product Owner (responsável pelo backlog), Scrum Master (facilitador) e Time de Desenvolvimento. Reuniões como Daily Scrum e Retrospectivas promovem transparência e melhoria contínua.
- **Kanban**: Método visual para gerenciar fluxos de trabalho, utilizando quadros com colunas (ex.: "A Fazer", "Em Progresso", "Concluído"). Foca em limitar trabalho em progresso (WIP) e melhorar continuamente.
- **Lean**: Filosofia que busca eliminar desperdícios (atividades que não agregam valor ao cliente) e maximizar valor. É frequentemente combinada com Kanban.
- **Kaiser** presume-se ser um framework estruturado para projetos de software, com foco em planejamento, controle de riscos e conformidade, possivelmente

adequado para setores regulados. Mais detalhes sobre o contexto são necessários para precisão.

O desenvolvimento do sistema HelpDesk, responsável pela abertura e gerenciamento de chamados, foi fortemente beneficiado pela aplicação das metodologias ágeis Kanban e Scrum. O Kanban contribuiu com a visualização clara do fluxo de tarefas, permitindo priorização contínua e controle do andamento das atividades (ANDERSON, 2010), enquanto o Scrum proporcionou ciclos curtos de entrega, foco em valor e acompanhamento incremental a cada sprint (SCHWABER; SUTHERLAND, 2020). A combinação dessas duas abordagens trouxe flexibilidade, organização e ritmo ao projeto, garantindo maior eficiência no desenvolvimento de funcionalidades e na adaptação contínua às necessidades dos usuários, conforme recomendado pela engenharia moderna de software (PRESSMAN, 2011).

6.4.1. Atividade no Slack

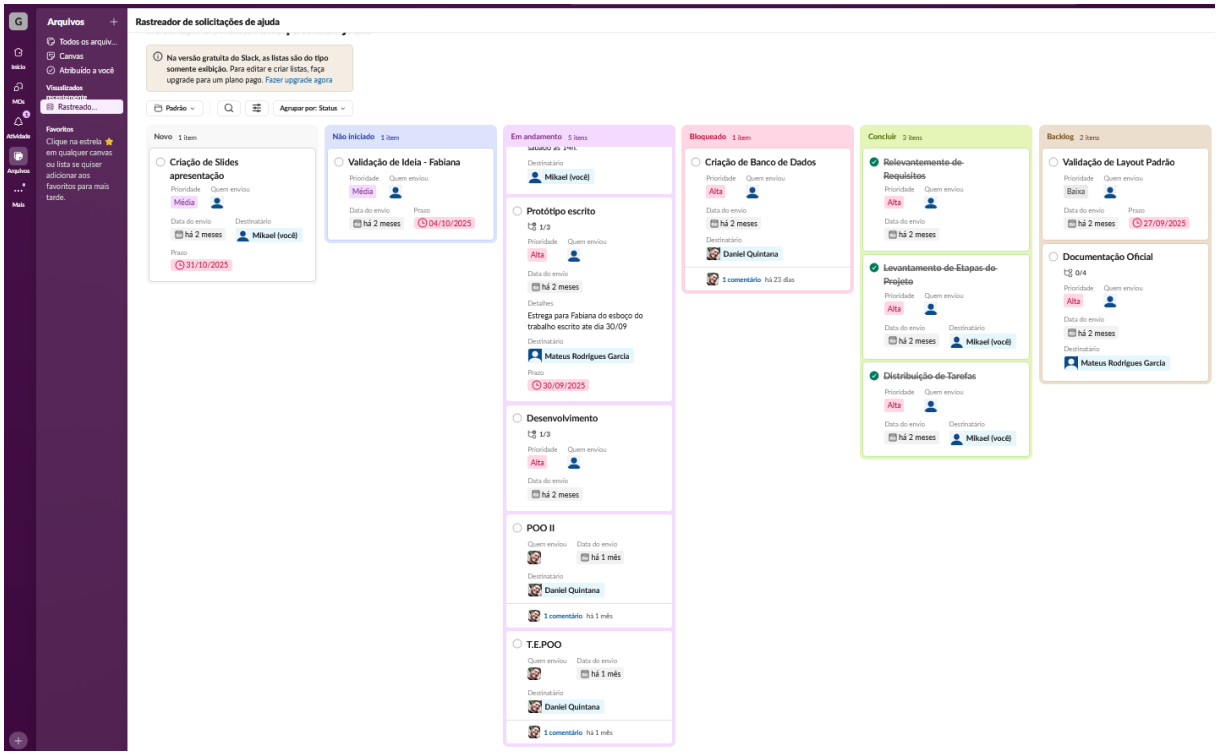

Figura 13: Slack Atividades

Fonte: autoria propria (2025)

Além disso, o HelpDesk foi projetado para múltiplas plataformas (desktop, web e mobile), o que gerou desafios como responsividade e usabilidade entre diferentes dispositivos. Entretanto, por meio de iterações frequentes, documentação contínua, testes estruturados e alinhamento constante com as partes interessadas, foi possível superar essas barreiras e entregar um sistema estável, intuitivo e eficiente, reforçando a importância de priorizar experiência do usuário, segurança da informação e clareza de comunicação durante todo o processo de desenvolvimento.

7. Desenvolvimento Web

O desenvolvimento web consolidou-se como uma das áreas mais relevantes da Tecnologia da Informação, possibilitando a criação de aplicações dinâmicas e interativas acessíveis diretamente por navegadores. Nesse contexto, o ASP.NET, framework da Microsoft, destaca-se por fornecer uma plataforma robusta e escalável para o desenvolvimento de aplicações web, como sites, sistemas corporativos e APIs. Segundo Esposito (2002), “o ASP.NET simplifica o desenvolvimento de aplicações web ao fornecer um modelo unificado de programação, integração com bancos de dados e suporte a múltiplos padrões de design”.

O ASP.NET é amplamente utilizado devido à sua integração nativa com o ecossistema Microsoft, suporte às linguagens C# e VB.NET, além de oferecer recursos de segurança, desempenho e manutenção facilitada (MICROSOFT, 2024). Essa combinação de fatores torna o ASP.NET uma escolha estratégica para empresas que buscam soluções confiáveis, escaláveis e de alto desempenho na web.

No desenvolvimento do front-end do sistema, foram construídas seis telas principais utilizando HTML, CSS e Bootstrap para a estilização das páginas (W3C, 2024; BOOTSTRAP, 2024), com o uso da IDE Visual Studio Community (MICROSOFT, 2024), aplicando o modelo de programação MVC (Model-View-Controller), que organiza a aplicação separando lógica, visualização e controle (FOWLER, 2003).

7.1. Tela de Usuario

A lógica central dos sistemas é dividida em dois pilares: a Gestão de Usuários e a Gestão de Chamados. As telas de Usuário estão concentradas no sistema Web, operando sob a lógica de CRUD (Create, Read, Update, Delete) e Controle de Acesso Baseado em Função (RBAC), permitindo que administradores gerenciem perfis como representado na imagem 14 e permissões, além de incluir um fluxo transacional (ConfirmacaoCompra) que sugere a integração com um módulo de vendas ou serviço. Por outro lado, as telas de Chamado representam o núcleo funcional de Helpdesk, sendo distribuídas entre o Mobile, que oferece uma interface simplificada para o usuário final abrir, listar e acompanhar suas solicitações, e o Web, que serve como a interface completa para o agente de suporte, permitindo a edição de status, prioridade e a gestão administrativa do ciclo de vida do chamado como mostra na figura 15.

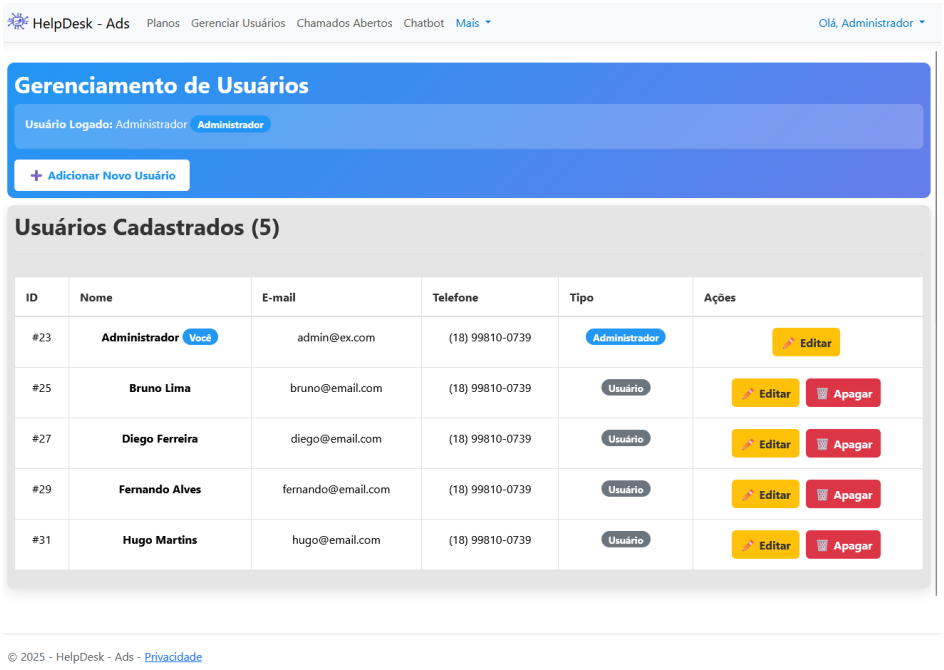

Figura 14: Tela de Gerenciamento de Usuários

Fonte: autoria propria (2025)

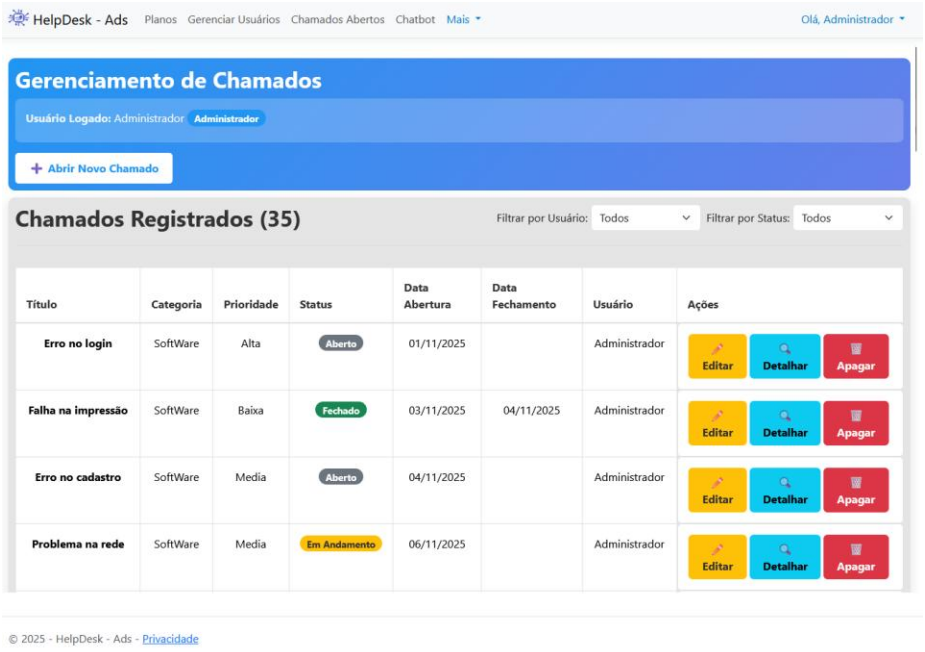

Figura 15: Tela de Gerenciamento de Chamados

Fonte: autoria propria (2025)

7.2. Associação de empreendedorismo

O empreendedorismo é uma área multidisciplinar voltada à identificação e gestão de oportunidades para criação de novos produtos, serviços ou negócios (SEBRAE, 2023). Envolve inovação, planejamento e a capacidade de assumir riscos calculados (DORNELAS, 2022). Nesse sentido, o empreendedor atua como agente transformador, convertendo ideias em valor econômico e social, contribuindo para o desenvolvimento de soluções e geração de empregos (SCHUMPETER, 1982).

Além de impulsionar o crescimento econômico, o empreendedorismo estimula criatividade, liderança e capacidade de adaptação em cenários de constantes mudanças (SEBRAE, 2023). Dessa forma, estudar essa disciplina é essencial para formar profissionais capazes de inovar e promover transformação social (DORNELAS, 2022).

Um exemplo prático é a aplicação do empreendedorismo digital na construção de soluções online como na figura 15 que demonstrando como a inovação e a oferta de serviços digitais podem gerar valor e novas oportunidades de negócio (SEBRAE, 2023).

7.2.1. Planos de Vendas

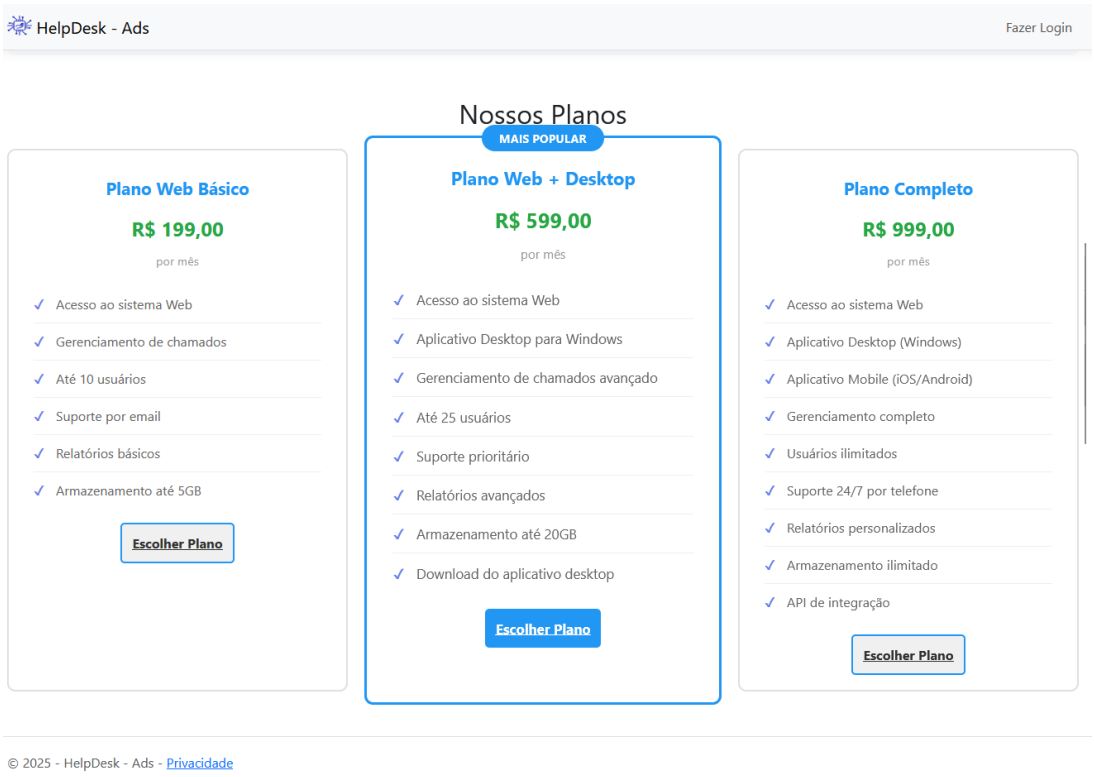

Figura 16: Planos de Venda

Fonte: autoria propria (2025)

Nesta etapa, foram utilizados HTML, CSS e C# para estruturar as interações e a visualização dos dados. Esses recursos permitem tanto o preenchimento quanto a exibição de informações na interface. Na Figura 14 é demonstrado o início da página principal representada pelo arquivo _Layout.cshtml. Esse arquivo inicia identificando o tipo de documento e, em seguida, define as referências e vínculos de estilo e scripts dentro da tag <head>. Já no <body> permanece todo o restante do conteúdo, ou seja, a parte que realmente é apresentada ao usuário final. Segundo MDN Web Docs (2024), a separação entre estrutura (HTML), estilo (CSS) e comportamento (JavaScript) contribui para uma manutenção mais organizada e uma melhor experiência de uso no desenvolvimento web moderno.

7.2.2. Tela de Codigo exemplo

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - HelpDesk - Ads</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/MyProject.styles.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/css/plans_and_auth.css" asp-append-version="true" />
</head>
<body>

    @{
    var usuarioLogadoId = Context.Session.GetInt32("UsuarioLogadoId");
    var usuarioLogadoNome = Context.Session.GetString("UsuarioLogadoNome");
}


@if (usuarioLogadoId.HasValue)
{
        // Se estiver logado, exibe o menu completo
    <header>
        <nav class="navbar navbar-expand-sm navbar-light bg-light border-bottom box-shadow">
            <div class="container">
```

Figura 17: Exemplo do Codigo

Fonte: autoria propria (2025)

8. Inclusão e Representatividade

8.1. O Brasil e o Contexto Afrodescendente

O Brasil abriga uma das maiores populações afrodescendentes do mundo, um legado histórico profundo de colonização, escravização e, principalmente, de enorme resistência e contribuição cultural. Nossa sociedade é marcada pela riqueza das influências africanas, indígenas e europeias, que moldam costumes, religião, linguagem e nossa vida diária (MUNANGA, 1999).

No entanto, essa riqueza convive com **desigualdades estruturais** persistentes. Dados do IBGE (2022) e estudos como os de Silva (2020) mostram que a população negra ainda é sub-representada nos espaços de poder, nos cargos de liderança e, crucialmente, nos setores de tecnologia e inovação. Esse padrão histórico exige uma postura ativa, com iniciativas que promovam a reparação, a representatividade e a valorização da identidade afrodescendente.

8.2. Inclusão e Representatividade na Tecnologia

A Tecnologia da Informação (TI), sendo um motor de desenvolvimento social, tem a responsabilidade de ser um ambiente justo e inclusivo. Contudo, relatórios de organizações como MOVER (2023) e PRETALAB (2022) continuam a apontar a baixa presença de pessoas negras e de outros grupos historicamente marginalizados na área, tanto nas universidades quanto nas empresas.

Essa realidade nos motiva a ir além do código. É fundamental criar iniciativas que potencializem o acesso, o pertencimento e a permanência de afrodescendentes em TI. Ações focadas em **valorização cultural, identidade positiva, combate ao racismo recreativo e institucional**, e a eliminação de estigmas são essenciais para transformar esse cenário.

8.3. O Papel do Design Inclusivo e das Mensagens Educativas

Ao desenvolver o **HelpDesk App** com temas visuais afrocentrados (como o *Ancestral*), mensagens de consciência racial e integração com informações sobre organizações como a MOVER, o projeto se propõe a ser mais do que um serviço técnico. Ele se torna um **espaço simbólico de afirmação identitária e pertencimento**.

O design inclusivo transcende a estética: ele garante que o usuário, em especial o historicamente marginalizado, se sinta representado e respeitado (SANTOS, 2021). A integração de conteúdos educativos, referências a datas importantes (Dia da Consciência Negra), frases de personalidades negras e links para organizações de apoio cria oportunidades de formação contínua, combate ao preconceito e ampliação das redes de suporte à diversidade em TI.

8.4. Referências Normativas e Legais

A legislação brasileira, como a Lei **10.639/03** (que obriga o ensino de História e Cultura Afro-Brasileira), aponta o caminho. Ela desafia a sociedade civil e o setor produtivo a garantirem ambientes mais plurais. Princípios de **equidade, respeito à diferença e justiça social** não devem ser opcionais; eles precisam permear a lógica de desenvolvimento de sistemas, plataformas e aplicativos corporativos, como um compromisso ético e social.

9. Gestão Da Qualidade

A Gestão da Qualidade é uma disciplina que busca assegurar que produtos, serviços e processos atendam a padrões estabelecidos, promovendo a satisfação do cliente e a melhoria contínua nas organizações (ISO, 2023). Essa área foi fortemente influenciada por importantes pensadores e pesquisadores que desenvolveram métodos, filosofias e ferramentas voltadas à melhoria de processos. Entre esses especialistas destacam-se William Edwards Deming, Joseph M. Juran, Philip B. Crosby, Armand V. Feigenbaum e Kaoru Ishikawa, cujas contribuições moldaram a evolução da qualidade nas empresas (ASQ, 2024).

Deming popularizou o Ciclo PDCA e seus 14 pontos de gestão, defendendo o uso de estatística para melhoria de processos (DEMING, 1986). Juran difundiu a Trilogia da Qualidade (planejamento, controle e melhoria) e enfatizou o Princípio de Pareto para priorização de problemas (JURAN, 1992). Crosby introduziu o conceito de “zero defeitos”, afirmando que qualidade é conformidade aos requisitos (CROSBY, 1979). Feigenbaum desenvolveu o Controle Total da Qualidade, defendendo que a qualidade é responsabilidade de toda a organização (FEIGENBAUM, 1991). Já Ishikawa criou o diagrama de causa e efeito e os círculos de qualidade, reforçando o envolvimento de colaboradores na resolução de problemas (ISHIKAWA, 1985).

Além disso, metodologias como TQM, PDCA e Seis Sigma complementam a Gestão da Qualidade e são amplamente utilizadas por organizações modernas, assim como normas internacionais como a ISO 9000, que estabelecem diretrizes formais de gestão de qualidade (ISO, 2023). Tais práticas proporcionam benefícios como redução de custos, aumento da satisfação do cliente, melhoria de desempenho e maior competitividade empresarial (HARVARD BUSINESS REVIEW, 2021).

No contexto de um sistema HelpDesk, esses princípios podem ser aplicados na análise de indicadores como tempo de resposta, resolução de chamados e satisfação do usuário, contribuindo para um processo de atendimento mais eficiente e com foco na melhoria contínua (ASQ, 2024).

10. Conclusão

O desenvolvimento do sistema integrado de HelpDesk apresentado neste Projeto Integrador Multidisciplinar evidenciou a importância da engenharia de software aplicada de forma estruturada, considerando desde a análise de requisitos, modelagem UML, definição arquitetural e implementação em múltiplas plataformas. A construção do software utilizando princípios da Programação Orientada a Objetos, arquitetura em camadas e padrões reconhecidos pelo mercado demonstrou que a aplicação correta dessas técnicas resulta em sistemas mais organizados, escaláveis, reutilizáveis e de manutenção simplificada (PRESSMAN; MAXIM, 2021).

A integração de recursos de Inteligência Artificial mostrou-se um diferencial significativo, permitindo que funcionalidades de sugestão, otimização e automatização de tarefas agregassem eficiência ao atendimento técnico, reduzindo carga operacional e tempo de resposta ao usuário. Segundo Russell e Norvig (2022), a IA possui papel estratégico na melhoria de processos, tomada de decisão e automação inteligente, o que evidencia a contribuição concreta desta tecnologia para sistemas modernos.

Além dos aspectos técnicos, o projeto reforçou a importância da gestão do desenvolvimento, abrangendo metodologias ágeis, ferramentas de gerenciamento colaborativo e indicadores de qualidade. Conforme o PMI (2021), a adoção de boas práticas de gerenciamento contribui para maior controle de escopo, eficiência de entregas e melhoria de resultados, elevando o nível de maturidade de projetos de software.

Por fim, o trabalho também destacou a relevância social da tecnologia, ao incorporar elementos de representatividade e inclusão, entendendo que soluções tecnológicas não devem apenas resolver problemas técnicos, mas também contribuir para uma construção digital mais justa, acessível e consciente. Como afirma Munanga (1999), reconhecer e valorizar a identidade afrodescendente é fundamental para combater desigualdades estruturais e ampliar espaços de participação.

Dessa forma, conclui-se que o sistema desenvolvido alcançou seus objetivos, comprovou sua viabilidade técnica e conceitual e demonstra potencial real de aplicação prática em ambientes corporativos, podendo evoluir para novas

funcionalidades, ampliação de módulos e integração com tecnologias avançadas. O HelpDesk apresentado configura-se, portanto, como uma solução inovadora, coerente com o cenário atual de transformação digital e alinhada às demandas do mercado contemporâneo.

11. Referências

ANDERSON, D. J. **Kanban: Successful Evolutionary Change for Your Technology Business**. Blue Hole Press, 2010.

BOOCH, Grady; RUMBAUGH, James; JACOBSON**, Ivar. UML – Guia do Usuário. 2.** ed. Rio de Janeiro: Elsevier, 2005.

DEITEL, Paul; DEITEL, Harvey. Java: **Como Programar. 10**. ed. São Paulo: Pearson, 2016.

ESPOSITO, Dino; SYME, Peter. **Professional ASP.NET MVC 5. 1**. ed. Wrox, 2018.
FURGERI, Sérgio. Lógica de Programação e Algoritmos com Java. São Paulo: Érica, 2016.

GRAFANA LABS**. Grafana Documentation.** Disponível em: https://grafana.com/docs/. Acesso em: 29 set. 2025.

IBGE. **Desigualdades Sociais por Cor ou Raça no Brasil. 2022**.
LARMAN, Craig. Utilizando UML e Padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

MARTHA, Luiz Fernando; PEREIRA, André. **Modelagem de Software Orientada a Objetos com UML.** PUC-Rio, 2021. Disponível em: https://www.tecgraf.puc-rio.br/ftp_pub/lfm/CIV2802-ModelagemOrientadaObjetos.pdf. Acesso em: 21 set. 2025.

MDN WEB DOCS**. HTML – Estrutura básica de um documento**. Mozilla Foundation, 2024. Disponível em: https://developer.mozilla.org/. Acesso em: 29 set. 2025.

MIRO. **Miro Help Center**. Disponível em: https://help.miro.com/. Acesso em: 29 set. 2025.

MOREIRA, Lorena Borges. **Análise e Projeto Orientado a Objeto Usando UML.** Uberlândia, 2000. Monografia (Bacharelado em Ciência da Computação) — Centro Universitário do Triângulo. Disponível em: https://pt.scribd.com/document/45295276/Analise-e-Projeto-Orientado-a-Objeto-Usando-UML. Acesso em: 21 set. 2025.

MOVER – **Movimento pela Equidade Racial.** Disponível em: https://mover.org.br/. Acesso em: 29 set. 2025.

MUNANGA, Kabengele**. Rediscutindo a mestiçagem no Brasil. Petrópolis: Vozes, 1999.**
PRESSMAN, Roger S.; MAXIM, Bruce R. Engenharia de Software: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

PRETALAB. **Presença de mulheres negras na tecnologia**. Relatório 2022.

PROJECT MANAGEMENT INSTITUTE – PMI. **A Guide to the Project Management Body of Knowledge (PMBOK Guide**). 7. ed., 2021.

ROYCE, Winston W**. Managing the Development of Large Software Systems.** Proceedings of IEEE WESCON, Los Angeles, 1970. Disponível em: https://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf. Acesso em: 21 set. 2025.

SANTOS, Joyce. **Design inclusivo e representatividade negra na tecnologia digital. UFBA, 2021.**

SCHWABER, K.; SUTHERLAND, J**. The Scrum Guide. 2020**. Disponível em: https://scrumguides.org. Acesso em: 29 set. 2025.

SILVA, Nádia. **O negro na tecnologia: Desafios e perspectivas. UnB, 2020.**

SILVEIRA, Paulo et al. Java: **Como Programar.** São Paulo: Novatec, 2015.
SLACK. Slack Help Center. Disponível em: https://slack.com/help. Acesso em: 29 set. 2025.

SOMMERVILLE, Ian. **Engenharia de Software. 9.** ed. São Paulo: Pearson Addison Wesley, 2011.

SOUZA NETO, Francisco. **Arquitetura e Design de Software**. São Paulo: Novatec.
TRELLO. Trello Help Center. Disponível em: https://support.atlassian.com/trello/. Acesso em: 29 set. 2025.

WOMACK, J. P.; JONES, D. T. **A Mentalidade Enxuta nas Empresas: Lean Thinking.** Rio de Janeiro: Campus, 2004.