INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ

CAMPUS TERESINA CENTRAL

CURSO TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

KAWÃ SOUSA DE LIMA

DESENVOLVIMENTO DE APLICAÇÃO MÓVEL INTEGRADA AO GLPI PARA SUPORTE DE TI

TERESINA - PIAUÍ

2026

KAWÃ SOUSA DE LIMA

DESENVOLVIMENTO DE APLICAÇÃO MÓVEL INTEGRADA AO GLPI PARA SUPORTE DE TI

Trabalho de Conclusão de Curso (relatório de software) apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina Central.
Orientador: Prof. Me. Francisco Marcelino Almeida de Araújo.

TERESINA - PIAUÍ

2026

AGRADECIMENTOS

Agradeço à minha mãe, Rita de Cássia, por me ensinar o poder da educação. Aos meus colegas e amigos do curso, em especial aos companheiros do LABIRAS, pela oportunidade de crescer juntos, como pessoa e como profissional. Aos educadores que passaram pela minha vida, em especial ao Me. Francisco Marcelino, por me mostrar onde o trabalho duro pode levar.

"Todas as pessoas verdadeiramente fortes são gentis."
(Arataka Reigen, personagem de Mob Psycho 100)

RESUMO

A crescente complexidade das infraestruturas de Tecnologia da Informação (TI) exige soluções de Gerenciamento de Serviços de TI (ITSM) que garantam agilidade no atendimento de incidentes. O sistema *open source* GLPi (Gestionnaire Libre de Parc Informatique), amplamente utilizado para gestão de ativos e chamados, embora robusto, apresenta limitações de usabilidade para técnicos em campo que dependem de sua interface web para operações rápidas. Este trabalho apresenta o desenvolvimento de uma aplicação móvel multiplataforma integrada ao GLPi, focada em otimizar o fluxo de atendimento da equipe de suporte e fornecer mobilidade. A solução foi arquitetada em três camadas, utilizando o *framework* Flutter (linguagem Dart) para o desenvolvimento cliente (Android e iOS), e um serviço *middleware* com Python/Django REST *Framework*, que atua como API *Gateway* seguro e orquestrador de notificações. Foi implementado um sistema de alertas instantâneos via Firebase Cloud Messaging (FCM) e funcionalidades cruciais, como a visualização de chamados recentes e a "autoatribuição"de *tickets*, eliminando o gargalo operacional da dependência de estações de trabalho fixas. O projeto demonstra a viabilidade técnica de modernizar sistemas ITSM legados através do desacoplamento de serviços e resulta em maior eficiência operacional para as equipes de TI.

Palavras-chave: GLPi; aplicação móvel; Flutter; notificações *push*; suporte de TI

ABSTRACT

The increasing complexity of Information Technology (IT) infrastructures demands IT Service Management (ITSM) solutions that ensure agility in incident response. The widely adopted open-source GLPi (Gestionnaire Libre de Parc Informatique) system, used for asset and *ticket* management, is robust but presents usability limitations for field technicians who rely on its web interface for quick operations. This work presents the development of a multiplatform mobile application integrated with GLPi, focused on optimizing the IT support team's workflow and providing mobility. The solution was architected in three layers, using the Flutter *framework* (Dart language) for client development (Android and iOS), and a *middleware* service built with Python/Django REST *Framework*, which acts as a secure API *Gateway* and notification orchestrator. An instant alert system via Firebase Cloud Messaging (FCM) was implemented alongside crucial features, such as viewing recent calls and *ticket* "self-assignment,"eliminating the operational bottleneck of relying on fixed workstations. The project demonstrates the technical feasibility of modernizing legacy ITSM systems through service decoupling, resulting in increased operational efficiency for IT teams.

Keywords: GLPi; mobile application; Flutter; *push* notifications; IT support

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AES	Advanced Encryption Standard
AOT	Ahead-of-Time
API	*Application Programming Interfaces*
APK	Android Package Kit
AWS	Amazon Web Services
DRF	Django REST *Framework*
EC2	Elastic Compute Cloud
FCM	Firebase Cloud Messaging
FIPS	Federal Information Processing Standards
GLPi	*Gestionnaire Libre de Parc Informatique*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IFPI	Instituto Federal de Educação, Ciência e Tecnologia do Piauí
IPA	iOS App Store Package
ITIL	Information Technology Infrastructure Library
ITSM	Information Technology Service Management
JIT	Just-in-Time
JSON	*JavaScript Object Notation*
LABIRAS	Laboratório de Robótica, Automação e Sistemas Inteligentes
MVVM	Model-View-ViewModel
RDS	Relational Database Service
REST	Representational State Transfer
SLA	Service Level Agreement (Acordo de Nível de Serviço)
SSL	Secure Sockets Layer
TI	Tecnologia da Informação
TMR	Tempo Médio de Resposta
UI	*User Interface*
URL	*Uniform Resource Locator*
UX	*User Experience*

SUMÁRIO

1 INTRODUÇÃO

No cenário corporativo contemporâneo, a Tecnologia da Informação (TI) consolidou-se não apenas como uma área de suporte operacional contingencial, mas como um pilar estratégico essencial para a continuidade, inovação e eficiência dos negócios. Com a crescente complexidade das infraestruturas computacionais e a exigência ininterrupta por alta disponibilidade de serviços, as equipes de suporte técnico enfrentam o desafio constante de solucionar incidentes e atender a solicitações de serviço de forma tempestiva. Tais atividades devem respeitar Acordos de Nível de Serviço (SLA - Service Level Agreement) cada vez mais rigorosos, alinhando-se às melhores práticas de Governança de TI e ao Gerenciamento de Serviços de TI (ITSM), cujos preceitos são amplamente difundidos por bibliotecas de infraestrutura consolidadas, a exemplo da ITIL 4 (AXELOS LIMITED, 2019). É neste contexto de alta exigência temporal e complexidade operacional que a mobilidade computacional surge como um fator crítico de sucesso, permitindo que a agilidade do atendimento técnico acompanhe o dinamismo imposto pelas operações empresariais modernas.

Os profissionais de suporte técnico frequentemente realizam atividades de campo, deslocando-se continuamente entre diferentes departamentos, filiais corporativas ou até mesmo clientes externos para executar manutenções preventivas e corretivas. Contudo, a dependência histórica de estações de trabalho fixas (*desktops*) para a consulta de inventário, atualização de status e encerramento de chamados gera um gargalo operacional crônico nas organizações. Essa limitação física inerente ao modelo tradicional de suporte resulta em lacunas de comunicação em tempo real e atrasos sistêmicos na atualização do estado dos atendimentos, impactando negativamente e diretamente os índices de SLA. A latência informacional gerada por este distanciamento impede que a equipe de suporte responda e registre prontamente novos eventos críticos, provocando a degradação sistêmica do tempo médio de resposta (TMR) e do tempo médio de resolução métricas fundamentais no ecossistema ITSM e afetando intrinsecamente a percepção de qualidade do serviço prestado ao usuário final (AXELOS LIMITED, 2019).

No ecossistema global de soluções para ITSM, o sistema GLPi (Gestionnaire Libre de Parc Informatique) consolidou-se como uma das plataformas de código aberto mais robustas e adotadas globalmente para a gestão centralizada de serviços, rastreamento de incidentes e inventário de ativos compatíveis com os processos ITIL (GLPI PROJECT, 2026). Todavia, sua interface nativa foi majoritariamente concebida e otimizada para o uso em navegadores web em ambientes *desktop*. Embora a plataforma web contemporânea possua características de

responsividade, a sua utilização através de navegadores em dispositivos móveis não proporciona a experiência de usuário (UX) ideal para operações dinâmicas em campo. Além da ergonomia de software prejudicada, o acesso via navegador carece de integrações vitais com o hardware do dispositivo do técnico, a exemplo das notificações *push* nativas, essenciais para alertas em tempo real. Evidencia-se, portanto, a premente necessidade de conceber soluções complementares focadas em mobilidade extrema para potencializar a atuação técnica em campo.

É fundamentado neste cenário desafiador que se justifica o desenvolvimento de uma aplicação móvel multiplataforma dedicada, desenhada especificamente para estender as funcionalidades primordiais do GLPi para o escopo móvel, colocando a gestão de incidentes diretamente na palma da mão do profissional de TI. A solução arquitetural proposta visa otimizar de ponta a ponta o fluxo de atendimento através de alertas instantâneos sobre novos chamados críticos e da inovadora funcionalidade de "autoatribuição", que permite a um técnico assumir a responsabilidade direta de um *ticket* imediatamente após sua abertura, independentemente de sua localização física ou da disponibilidade de um terminal fixo. Adicionalmente, a implementação planejada de um histórico *offline* de notificações atua como um mecanismo de contingência, garantindo que o profissional mantenha o acesso à informação temporal mesmo em ambientes confinados ou com conectividade de rede intermitente.

Para viabilizar tecnicamente esta arquitetura moderna e integrá-la ao sistema legado, optou-se por um modelo estrutural desacoplado em três camadas lógicas. A camada de apresentação, materializada no cliente móvel, utiliza o *framework* de código aberto Flutter e a linguagem de programação Dart, empregando o padrão de projeto de software MVVM (Model-View-ViewModel) para garantir uma segregação rígida entre a lógica de negócios e a interface, proporcionando alta fluidez e compilação nativa para os sistemas operacionais Android e iOS a partir de uma base de código unificada (GOSSMAN, 2005; GOOGLE, 2026c). O núcleo inteligente da solução é fundamentado em um serviço *middleware* desenvolvido com a linguagem Python e o *framework* Django REST (ENCODE, 2026). Esta camada intermediária atua principalmente como um API *Gateway* seguro e orquestrador de eventos, desacoplando o aplicativo móvel das complexidades da API nativa do sistema legado GLPi, e gerenciando o envio assíncrono de notificações transacionais por meio do serviço multiplataforma Firebase Cloud Messaging (GOOGLE, 2026b).

Um diferencial metodológico e crítico desta pesquisa é a implementação de uma lógica *multi-tenant* (*multi-time*), permitindo que o *middleware* orquestre múltiplos ambientes e instâncias independentes do GLPi de forma logicamente isolada, escalável e integralmente segura. A

integridade e a confidencialidade das credenciais sensíveis das instâncias geridas (compreendendo URLs de API, User Tokens e App Tokens) são resguardadas por uma sofisticada estratégia de unificação e serialização de dados, protegida no banco de dados relacional PostgreSQL (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2026) por meio da aplicação do padrão de criptografia simétrica de bloco AES-256 (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, 2001). Dessa forma, o presente trabalho monográfico não apenas explora os desafios da integração móvel com sistemas ITSM legados, mas demonstra empiricamente como a adoção de padrões arquiteturais contemporâneos, aliada à segurança criptográfica de nível governamental e à orquestração de serviços via API *Gateway*, pode transformar processos operacionais arcaicos, elevando substancialmente a eficiência, a segurança e a agilidade das equipes de suporte de Tecnologia da Informação.

1.1 OBJETIVOS

Os objetivos delineados para este projeto de software foram estruturados com o rigor necessário para garantir que o ciclo de desenvolvimento tecnológico resulte em uma solução arquiteturalmente robusta, resiliente e perfeitamente capaz de sanar as limitações de mobilidade identificadas na gestão de serviços de TI (ITSM) através do ecossistema GLPi.

1.1.1 Objetivo Geral

O objetivo primordial desta pesquisa aplicada consiste em desenvolver e validar uma aplicação móvel multiplataforma nativa, perfeitamente integrada ao sistema de gerenciamento de serviços de TI (GLPi), através da concepção de uma arquitetura orientada a serviços provida por um *middleware* (API *Gateway*). O propósito central é prover mobilidade irrestrita aos técnicos de suporte de campo, agilizar a comunicação de incidentes cibernéticos ou de infraestrutura por meio de notificações *push* em tempo real e, consequentemente, otimizar todo o fluxo de ciclo de vida e resolução de chamados técnicos.

1.1.2 Objetivos Específicos

Para assegurar que o objetivo geral seja atingido de maneira sistemática e mensurável, o escopo do projeto foi decomposto nos objetivos específicos detalhados e estruturados na Tabela de Metas Tecnológicas apresentada a seguir.

Tabela 1 – Metas Tecnológicas dos Objetivos Específicos

Eixo Tecnológico	**Descrição do Objetivo Específico**	**Impacto Operacional e Metodológico Esperado**
Camada de Apresentação (Mobile)	Construir a aplicação cliente multiplataforma empregando o *framework* Flutter e a linguagem Dart, sob a estrita observância do padrão de arquitetura de software MVVM (Model-View-ViewModel).	Garantir alto desempenho através de compilação nativa (AOT), fluidez de interface de usuário (UI) e o desacoplamento eficiente da lógica de estado em plataformas Android e iOS.
Camada de Aplicação (*Middleware*)	Projetar e desenvolver um serviço *middleware* escalável utilizando a linguagem Python e o Django REST *Framework*, atuando como um API *Gateway* centralizador e seguro.	Orquestrar a autenticação biométrica/tokenizada, simplificar as cargas úteis (*payloads*) JSON originadas do GLPi e proteger a comunicação direta com a API nativa do sistema legado.
Mensageria e Tempo Real	Implementar um sistema de mensageria assíncrona e alertas em tempo real através da integração estrutural com a infraestrutura do Firebase Cloud Messaging (FCM).	Assegurar a entrega garantida de notificações *push* sobre incidentes de alta severidade diretamente nos dispositivos móveis, operando ativamente mesmo com o software em segundo plano (*background*).
Mobilidade e Contingência	Desenvolver funcionalidades de operação remota críticas para a mitigação do tempo de resposta (SLA), com destaque para a "autoatribuição"remota de chamados e armazenamento local.	Eliminar a dependência de interfaces *desktop*, garantindo a persistência de um histórico local de notificações (via Shared Preferences) para consulta irrestrita em modo *offline*.
Segurança e Persistência	Assegurar o isolamento lógico de dados e a segurança da informação corporativa através da arquitetura *multi-tenant*, integrando um banco de dados relacional (PostgreSQL).	Proteger credenciais de API legadas em tabelas relacionais sob a encriptação de algoritmos de criptografia simétrica avançada (AES-256), inviabilizando o vazamento de chaves de acesso.
Validação Arquitetural	Validar a viabilidade técnica e acadêmica da modernização de sistemas ITSM legados através de métricas de testes de integração e segurança fim-a-fim.	Demonstrar empiricamente que o desacoplamento via *middleware* reduz a dependência de estações fixas, mitiga a latência informacional e otimiza os tempos de atendimento (TMR).

Fonte: Elaborado pelo autor (2026).

2 TECNOLOGIAS ENVOLVIDAS

A construção de uma solução que integra sistemas legados de ITSM com plataformas móveis modernas requer uma seleção criteriosa de tecnologias que garantam desempenho, escalabilidade e segurança. Este capítulo detalha o conjunto tecnológico (*stack*) selecionado para o desenvolvimento do projeto, abrangendo desde o *framework* de interface até a infraestrutura de backend e segurança.

2.1 FLUTTER E LINGUAGEM DART

Diferentemente de antigas abordagens baseadas em processos de visualizações *web* (WebViews) interconectadas, que impunham lentidão e caracterizavam o declínio dos primeiros aplicativos híbridos, a plataforma móvel deste trabalho utiliza o *framework* Flutter. Este conjunto de ferramentas emprega um motor de renderização gráfico próprio, construído na linguagem C++, operando sob o canvas nativo do dispositivo. Tal característica primária, organicamente associada à linguagem orientada a objetos Dart — que dispõe do diferencial de compilação dinâmica (JIT) e antecipada (AOT) —, confere à interface gráfica não somente uma coesão visual impecável independente do sistema operacional base (Android ou iOS), mas um teto de desempenho tido como equiparável a execuções puramente nativas (GOOGLE, 2026c; GOOGLE, 2026a).

- Desempenho e Renderização: Diferente de abordagens híbridas, o Flutter possui seu próprio motor de renderização (Skia/Impeller), conferindo desempenho superior e consistência visual entre plataformas (GOOGLE, 2026c).
- Agilidade no Desenvolvimento: A funcionalidade de *Hot Reload* permite a visualização instantânea de alterações no código, acelerando o ciclo de desenvolvimento.
- Arquitetura baseada em Widgets: Facilita a criação de interfaces de usuário (UI) ricas e customizáveis, aderindo aos princípios de design moderno.

2.2 PYTHON E DJANGO REST *FRAMEWORK*

A atribuição de mediar de forma integralmente segura a comunicação entre o aplicativo de campo e a API nativa do servidor corporativo recaiu sobre a camada de API *Gateway*, erguida sob as bases da linguagem Python e utilizando os potentes aceleradores oferecidos pelo Django REST *Framework* (DRF). A decisão técnica por integrar este pacote repousa fundamentalmente

em sua inerente e irretocável capacidade de abstrair as mais variadas e caóticas estruturas relacionais de banco de dados, facultando a serialização sofisticada de classes Python em estruturas padronizadas de objetos JSON de altíssima compactação de *bytes*, requisito primordial ao lidar com conectividade de rede restrita enfrentada em ambientes 4G ou 5G periféricos (ENCODE, 2026).

- Segurança e Escalabilidade: O Django é reconhecido por sua segurança robusta e escalabilidade, sendo ideal para atuar como um orquestrador central (ENCODE, 2026).
- API *Gateway*: O DRF atua como um API *Gateway* seguro, gerenciando a comunicação entre o aplicativo móvel e o GLPi, além de abstrair a complexidade da API nativa do sistema legado.
- Serialização: A ferramenta simplifica a conversão de objetos complexos do banco de dados em formatos leves, como JSON, facilitando o tráfego de dados em redes móveis.

2.3 FIREBASE CLOUD MESSAGING (FCM)

Para atender ao requisito de comunicação em tempo real, utilizou-se o Firebase Cloud Messaging (FCM) (GOOGLE, 2026b).

- Notificações *Push*: O FCM é uma solução multiplataforma que permite o envio confiável de notificações e mensagens de dados.
- Entrega em *Background*: Garante que a equipe de suporte receba alertas instantâneos sobre novos chamados críticos, mesmo quando o aplicativo não está em execução no primeiro plano.

2.4 BANCO DE DADOS POSTGRESQL E SEGURANÇA

A camada de persistência utiliza o PostgreSQL (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2026), um banco de dados relacional que garante o isolamento entre diferentes instâncias de suporte.

- Multi-tenancy: O banco armazena dados de times, usuários e dispositivos, permitindo que o *middleware* orquestre múltiplos ambientes de forma isolada através do identificador de time.

- Proteção de Secrets: Para garantir a segurança, as credenciais do GLPi (URL da API, User Token e App Token) são agrupadas em uma estrutura JSON serializada e protegidas por criptografia simétrica AES-256 (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, 2001). Essa estratégia garante que informações sensíveis nunca fiquem expostas no dispositivo móvel, permanecendo inteiramente no lado do servidor.

2.5 PADRÕES ARQUITETURAIS E PROTOCOLOS

- Padrão MVVM (Model-View-ViewModel): Utilizado na camada de apresentação para desacoplar a interface de usuário da lógica de negócios, facilitando a manutenção e a testabilidade do aplicativo (GOSSMAN, 2005).

- Arquitetura REST: A comunicação entre todos os componentes segue o estilo arquitetural REST, utilizando métodos HTTP (GET, POST, etc.) para garantir interoperabilidade e padronização.

- HTTPS/SSL: Toda a comunicação entre o aplicativo, o *middleware* e o GLPi é obrigatoriamente criptografada via protocolo HTTPS/SSL, assegurando a integridade e privacidade dos dados.

- Padrão ITIL: O sistema integra-se ao GLPi, que é baseado nas melhores práticas da biblioteca ITIL para gestão de incidentes e ativos (AXELOS LIMITED, 2019).

3 MODELAGEM DO PROJETO

Este capítulo apresenta a especificação técnica da solução desenvolvida, detalhando os requisitos, as interações entre os atores e a estrutura arquitetural. A modelagem seguiu uma abordagem iterativa. Para orientar o desenvolvimento da aplicação móvel integrada ao GLPi (GLPI PROJECT, 2026), foi realizado um levantamento detalhado das necessidades operacionais. Os requisitos foram divididos entre funcionais e não funcionais, garantindo que o sistema atenda não apenas às expectativas do usuário, mas também aos padrões de qualidade técnica e segurança exigidos pelo ambiente corporativo.

3.1 LEVANTAMENTO DE REQUISITOS

Os requisitos funcionais descrevem as ações que o sistema deve ser capaz de executar.

Tabela 2 – Requisitos Funcionais

ID	Descrição do Requisito	Prioridade
RF001	Autenticação de Usuário: O técnico deve realizar *login* com credenciais do GLPi, validadas pelo *middleware*.	Alta
RF002	Visualização de Chamados: Listar chamados recentes (título, data, status) via API do GLPi.	Alta
RF003	Notificações *Push*: Notificar o técnico em tempo real sobre eventos críticos, mesmo em segundo plano.	Alta
RF004	Autoatribuição: Permitir que o técnico assuma a responsabilidade técnica de um *ticket* ("Atribuir a mim").	Média
RF005	Histórico *Offline*: Armazenar notificações localmente para consulta sem conexão à internet.	Média
RF006	Gerenciamento de Dispositivos: O *middleware* deve registrar e atualizar tokens FCM vinculados a usuários e times.	Alta

Fonte: Elaborado pelo autor (2026).

Os requisitos não funcionais definem os critérios de qualidade e restrições do sistema.

Tabela 3 – Requisitos Não Funcionais

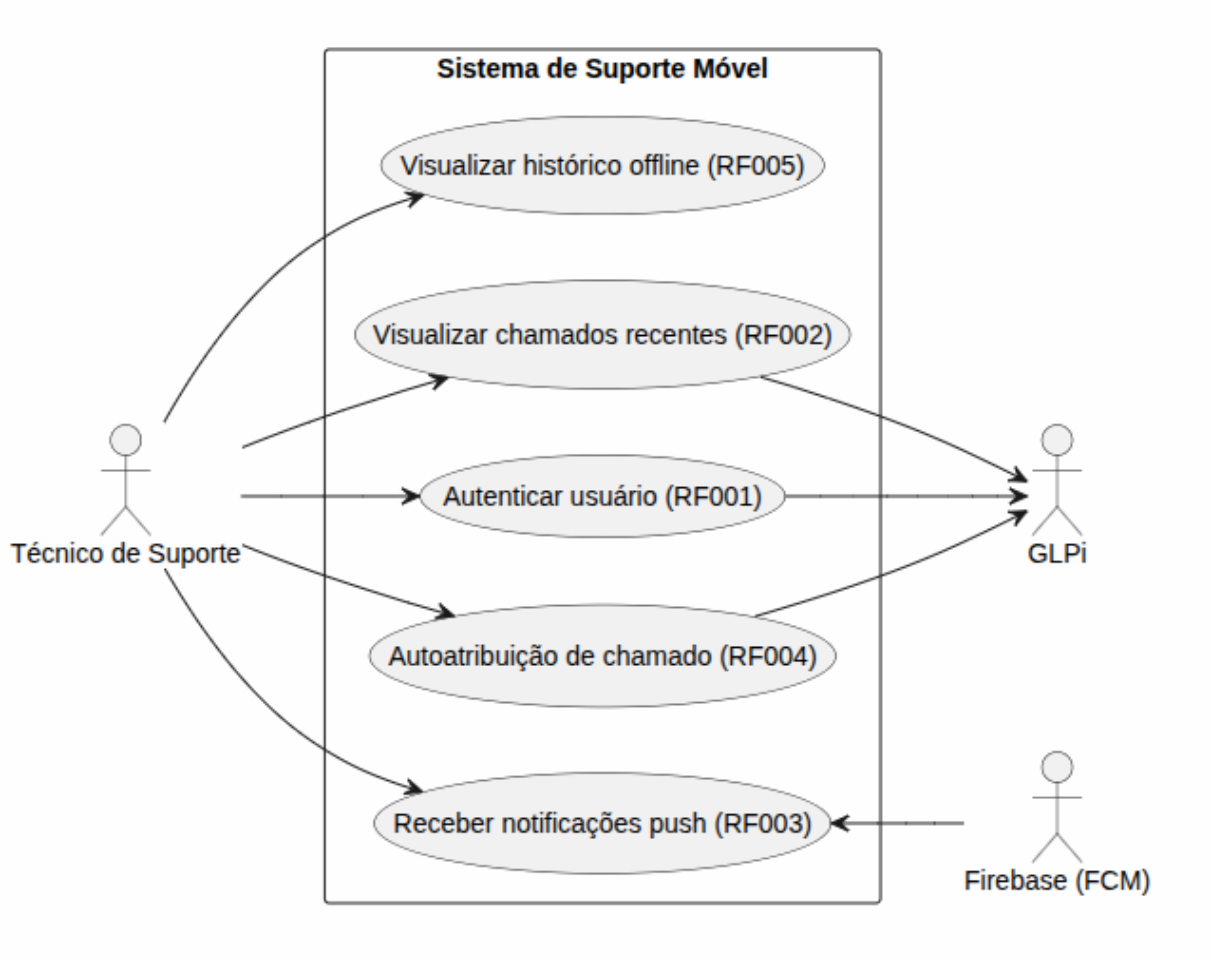
ID	Descrição do Requisito	Prioridade
RNF001	Compatibilidade Multiplataforma: Uso de base de código única Flutter para Android e iOS.	Alta
RNF002	Segurança na Comunicação: Toda troca de dados deve ser criptografada via HTTPS/SSL.	Alta
RNF003	Desempenho de Rede: Minimização do tráfego através de JSON leve para redes móveis.	Alta
RNF004	Escalabilidade: O *middleware* deve suportar múltiplos dispositivos sem sobrecarregar o servidor GLPi.	Média

Fonte: Elaborado pelo autor (2026).

3.2 DIAGRAMAS DE CASOS DE USO

A modelagem ilustra as interações entre o Técnico de Suporte, o Sistema GLPi e o Firebase (FCM).

Figura 1 – Diagrama de Casos de Uso



Fonte: Elaborado pelo autor (2026).

3.3 DIAGRAMAS DE CLASSE

A modelagem reflete o padrão MVVM (GOSSMAN, 2005) no aplicativo e a lógica de API *Gateway* no *middleware*. O diagrama de classes é essencial para visualizar o desacoplamento entre a interface e a lógica de negócios. Ele está dividido entre os dois componentes principais do projeto: o aplicativo móvel e o *middleware*.

3.3.1 Camada Mobile (Flutter/MVVM)

No aplicativo, a estrutura segue o padrão Model-View-ViewModel (GOSSMAN, 2005) para garantir testabilidade e manutenção facilitada:

- Models (TicketModel, NotificationModel): Representam os dados brutos consumidos da API, com métodos de serialização para converter o JSON recebido do *middleware* em objetos Dart.
- ViewModels (AuthViewModel, TicketViewModel): Atuam como intermediários, gerenciando o estado da interface (carregamento, listas) e executando as ações solicitadas pelo técnico, como a "autoatribuição".
- Serviços (NotificationStorage): Responsável pela persistência local (Shared Preferences), permitindo que as notificações sejam consultadas em modo *offline*.

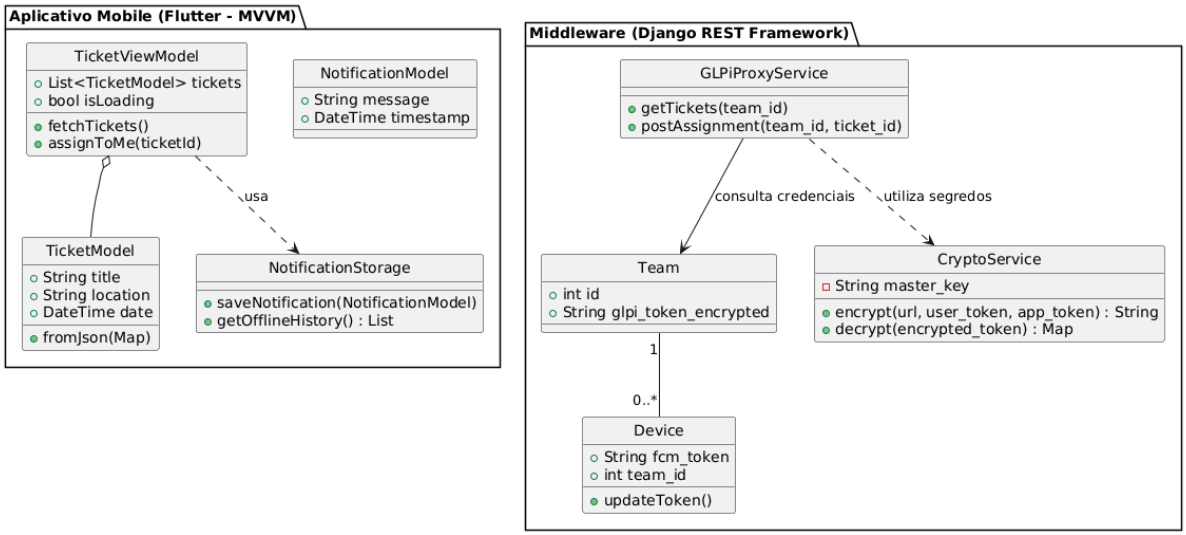
3.3.2 Camada *Middleware* (Django REST *Framework*)

No backend, as classes organizam a segurança e a orquestração com o GLPi:

- Models de Dados (Team, AuthUser, Device): Definem a estrutura de persistência no banco de dados PostgreSQL (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2026). A classe Team armazena o identificador único (cd_team) e as credenciais criptografadas de cada equipe, enquanto a classe Device gerencia os tokens FCM vinculados aos usuários para o envio de alertas.
- Serviços de Segurança (CryptoService): Este componente centraliza a lógica de criptografia simétrica AES-256. Ele é responsável por unificar as secrets do GLPi (URL, User Token e App Token) em uma estrutura serializada antes da gravação no banco, garantindo que informações críticas permaneçam inacessíveis sem a chave mestra do servidor.
- Serializers: Classes fundamentais do Django REST *Framework* (ENCODE, 2026) que convertem objetos complexos do banco de dados em um formato JSON leve, assegurando um tráfego de dados eficiente para redes móveis 4G/5G.
- GLPiProxyService: Atua como o orquestrador principal na lógica de *gateway*, sendo responsável por processar as consultas de chamados e enviar comandos de atualização, como a autoatribuição, diretamente para a API nativa do GLPi.

3.3.3 Diagrama

Figura 2 – Diagrama de Classe



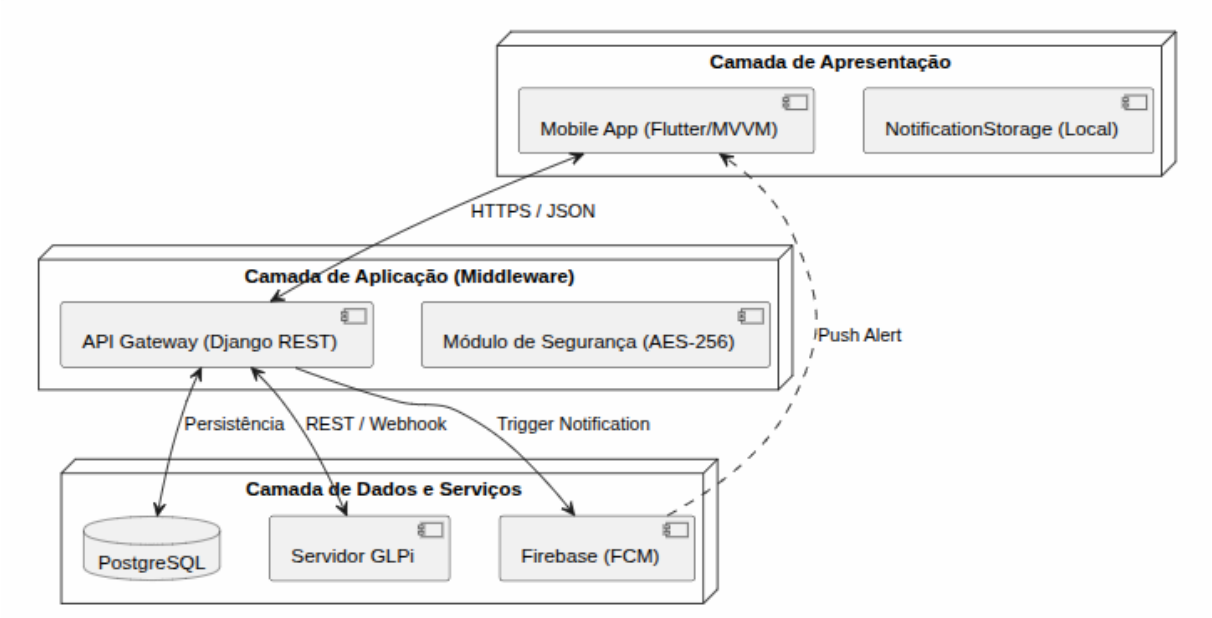
Fonte: Elaborado pelo autor (2026).

3.4 ARQUITETURA DO SISTEMA

3.4.1 Visão em Três Camadas

A solução adota uma Arquitetura em Três Camadas, utilizando um *middleware* como API *Gateway* seguro para desacoplar a aplicação móvel do sistema legado.

Figura 3 – Visão em Três Camadas

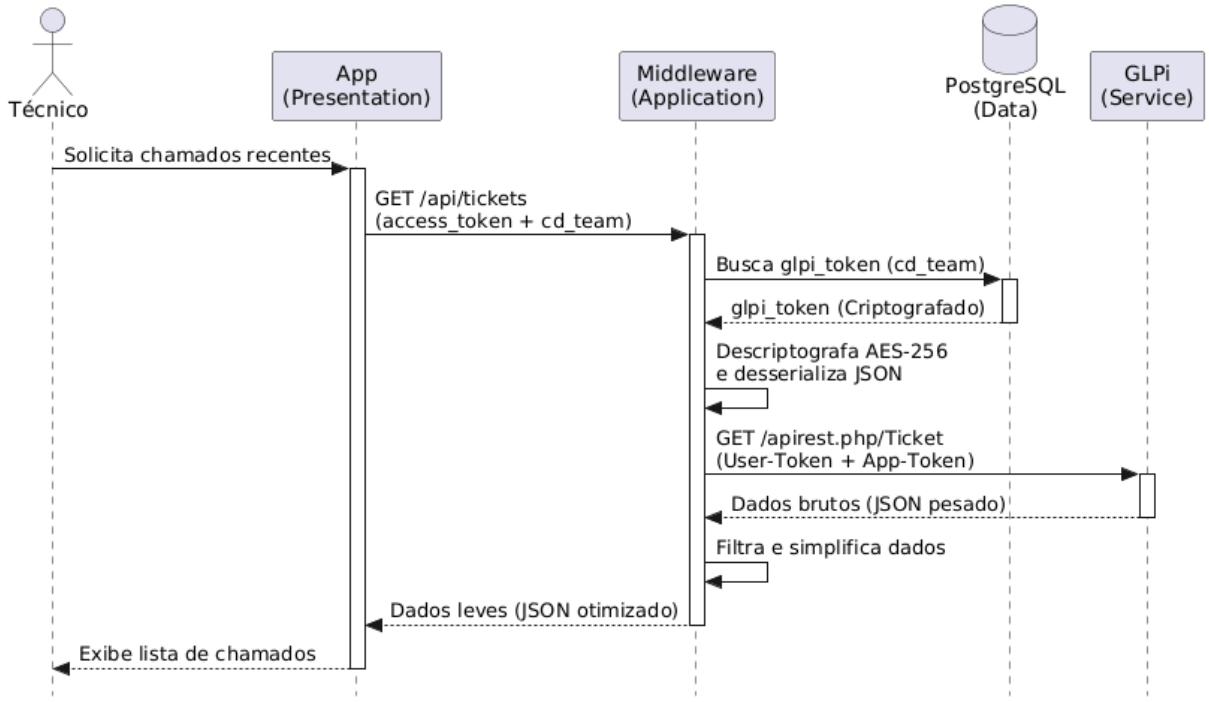


Fonte: Elaborado pelo autor (2026).

- Camada de Apresentação (Mobile): Composta pelo aplicativo desenvolvido em Flutter utilizando o padrão MVVM (Model-View-ViewModel). Esta camada foca na experiência do técnico, lidando apenas com a lógica de interface e consumindo dados leves em formato JSON.
- Camada de Aplicação (*Middleware*): Implementada com Django REST *Framework* (ENCODE, 2026), funciona como um API *Gateway* seguro e orquestrador de mensagens. Esta camada é o núcleo inteligente que valida permissões, gerencia notificações *push* e protege o acesso às instâncias do GLPi.
- Camada de Dados e Serviços: Compreende o banco de dados central PostgreSQL (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2026), o sistema legado GLPi e o serviço de mensageria Firebase Cloud Messaging (FCM).

O diagrama de sequência abaixo ilustra a interação entre essas camadas durante uma operação de consulta:

Figura 4 – Diagrama de Sequência



Fonte: Elaborado pelo autor (2026).

3.4.2 Segurança e Proteção de Credenciais

A salvaguarda da confidencialidade cibernética de organizações exige que a integridade das credenciais primárias de acesso às instâncias GLPi jamais possa ser confiada ou fixada

(*hardcoded*) nas memórias persistentes dos dispositivos móveis da equipe técnica, considerando sua altíssima e comprovada vulnerabilidade à manipulação física e técnicas de engenharia reversa de aplicativos (decompiling). Mediante este risco premente, a lógica central do API *Gateway* determina o armazenamento blindado de todos os tokens de sessão e rotas de APIs num banco de dados relacional hospedado na retaguarda corporativa. Este registro, antes de persistido no disco rígido do PostgreSQL, encontra-se hermeticamente encriptado sob a rigidez do algoritmo Advanced Encryption Standard com variação de blocos chaves de 256 bits (AES-256). Ratificado mundialmente e padronizado em agências de segurança global, o processo atende às mais elevadas normas do processamento da informação civil, chancelado em suas matrizes matemáticas diretamente pelo governo americano sob a norma FIPS 197 (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, 2001).

O trecho de código a seguir demonstra a implementação do serviço de criptografia em Python/Django, centralizando a lógica de segurança:

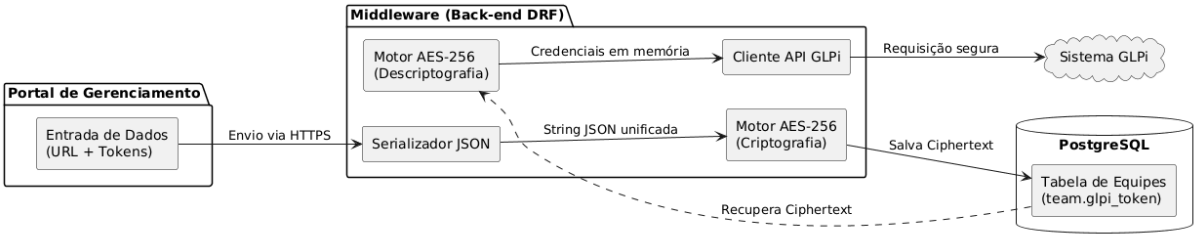
Código-fonte 1 – Implementação do *CryptoService*

```python
import json
from cryptography.fernet import Fernet
from django.conf import settings

class CryptoService:
    _fernet_instance = None

    @staticmethod
    def _get_fernet():
        if CryptoService._fernet_instance is not None:
            return CryptoService._fernet_instance

        master_key = settings.MASTER_KEY
        if isinstance(master_key, str):
            master_key = master_key.encode('utf-8')

        CryptoService._fernet_instance = Fernet(master_key)
        return CryptoService._fernet_instance
```

```
19
20      @staticmethod
21      def encrypt_credentials(url: str, user_token: str,
            app_token: str) -> bytes:
22          data = {'url': url, 'user_token': user_token, '
                app_token': app_token}
23          f = CryptoService._get_fernet()
24          return f.encrypt(json.dumps(data).encode('utf-8'))
25
26      @staticmethod
27      def get_decrypted_credentials(team) -> dict:
28          if not team.glpi_token:
29              return {'url': '', 'user_token': '', 'app_token
                    ': ''}
30          f = CryptoService._get_fernet()
31          decrypted_json = f.decrypt(team.glpi_token).decode(
                'utf-8')
32          return json.loads(decrypted_json)
```

O fluxo de segurança para o armazenamento e recuperação das credenciais é detalhado a seguir:

Figura 5 – Fluxo de Segurança para Armazenamento



Fonte: Elaborado pelo autor (2026).

Um diferencial crítico deste design foi a robustez na proteção da informação. A estratégia de unificação e serialização de segredos, protegida por criptografia simétrica AES-256 no banco de dados PostgreSQL, assegurou que as credenciais de acesso ao GLPi nunca ficassem expostas no lado do cliente. A arquitetura *multi-time* provou-se escalável, permitindo que o *middleware*

orquestre múltiplos ambientes de suporte de forma isolada e segura.

3.4.3 Lógica *Multi-tenant* (*Multi-time*)

Para suportar múltiplas equipes de forma isolada, o sistema utiliza uma lógica *multi-time* (*multi-tenant*). Cada equipe possui um identificador único (cd_team), que vincula usuários e dispositivos às suas respectivas configurações de GLPi. O *Middleware* orquestra as requisições de modo que um técnico de um time jamais acesse notificações ou dados de outro, garantindo o isolamento lógico dentro de um banco de dados centralizado.

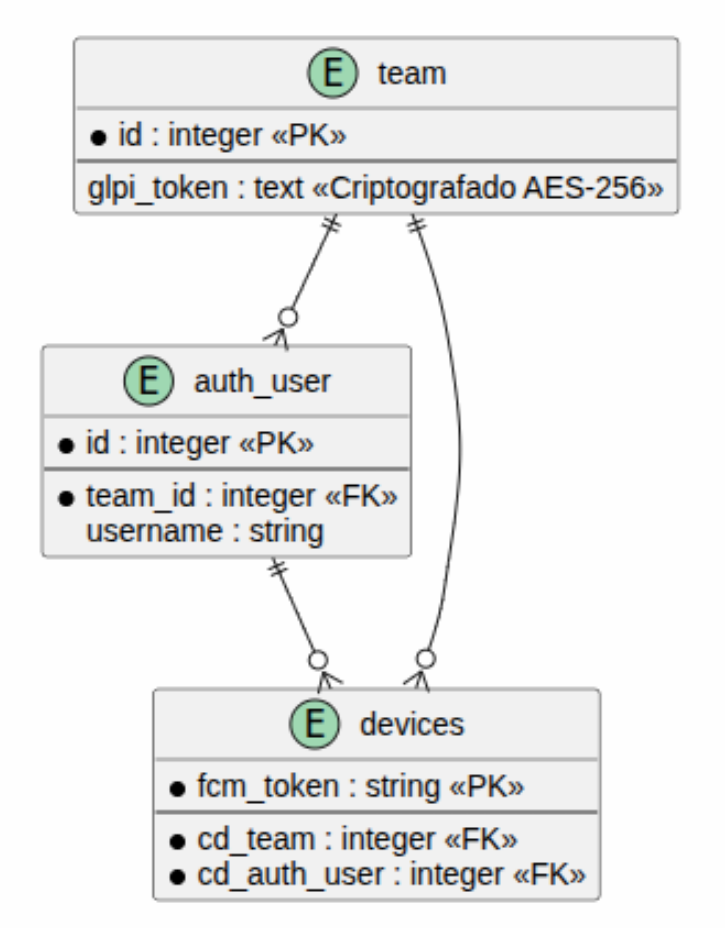
3.4.4 Comunicação e Integridade

A proteção das informações sensíveis baseia-se na centralização da gestão de segredos no *middleware*, que funciona como um proxy seguro entre o cliente e o sistema legado. Por meio da estratégia de unificação e serialização, o sistema agrupa os parâmetros de acesso em uma estrutura protegida por criptografia simétrica AES-256 no banco de dados. Este design assegura que as credenciais de cada equipe permaneçam isoladas e residam exclusivamente no lado do servidor, eliminando riscos de exposição de tokens no dispositivo móvel e viabilizando a operação escalável em ambientes *multi-time*.

3.5 DIAGRAMA ENTIDADES-RELACIONAMENTO

O diagrama a seguir exibe a estrutura relacional do banco de dados PostgreSQL (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2026), modelada para suportar a lógica *multi-tenant*. Ele descreve as tabelas **Team**, **AuthUser** e **Device**, detalhando os tipos de dados e os relacionamentos de chave estrangeira que garantem a integridade referencial.

Figura 6 – Diagrama Entidade Relacionamento



Fonte: Elaborado pelo autor (2026).

3.6 INTERFACE

Este capítulo detalha o Design de Interface (UI/UX) do aplicativo móvel, focando na experiência do técnico de campo. A interface foi projetada para ser intuitiva, priorizando a clareza e a rapidez de acesso à informação, garantindo que o profissional possa gerenciar chamados com o mínimo de cliques possíveis.

3.6.1 Descrição das Telas

O aplicativo é composto por três interfaces principais e componentes de interação rápida, todos construídos utilizando a arquitetura baseada em widgets do Flutter.

1. Tela de *Login*:

   - Propósito: Ponto de entrada para autenticação e vinculação ao time.
   - Elementos: Contém campos para "Usuário", "Senha"e o código da equipe (cd_team).

- Diferenciais: Inclui *feedback* visual de carregamento durante a validação das credenciais pelo *middleware* e um ícone de escudo que reforça a identidade visual do sistema de suporte.

Figura 7 – Interface da Tela de Login

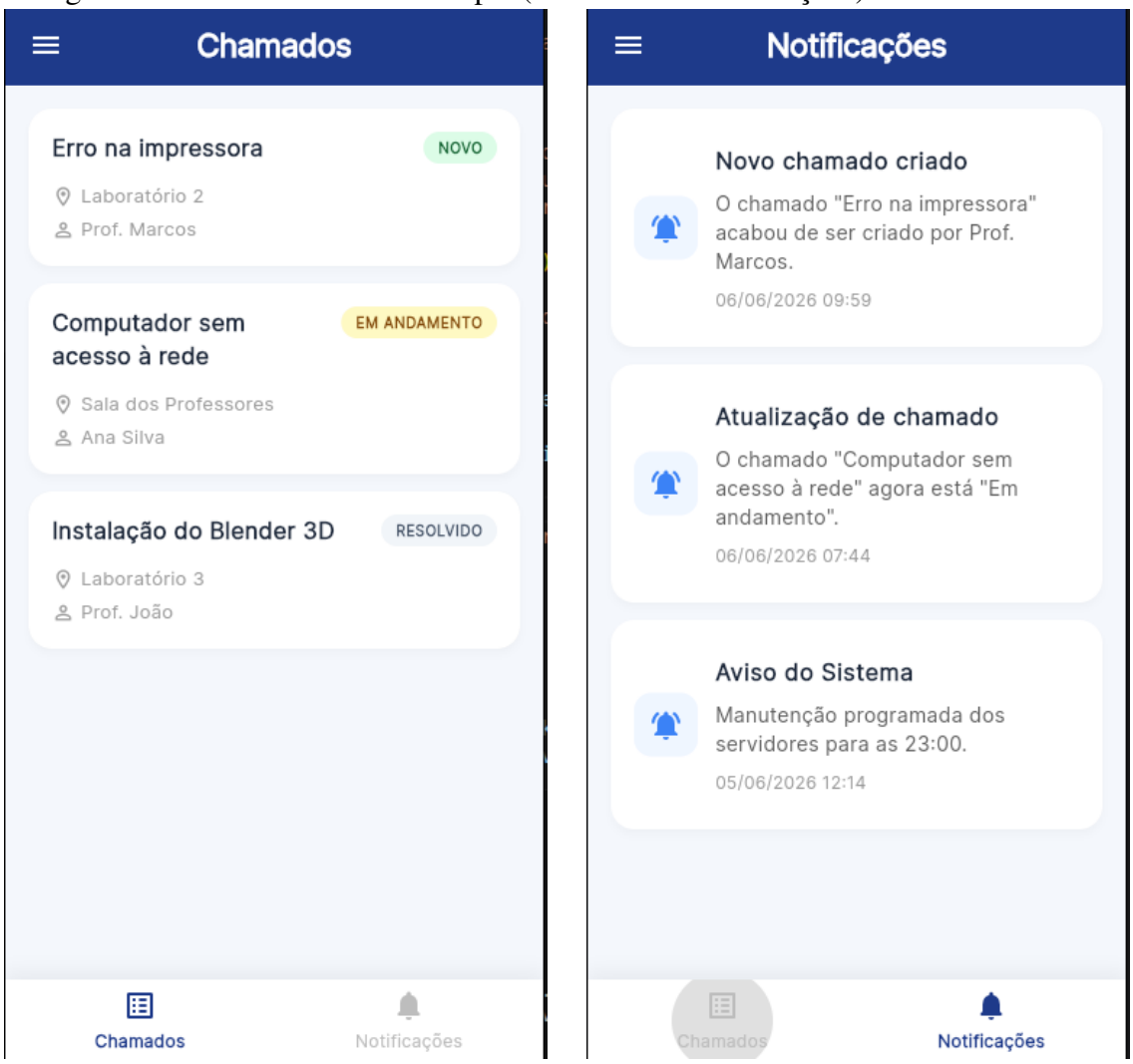
Fonte: Elaborado pelo autor (2026).

2. Tela Principal (Home):

- Propósito: Centralizar o fluxo de trabalho diário do técnico.
- Estrutura: Organizada em Abas (Tabs) para alternância rápida entre duas visões principais:
  - Aba de Notificações: Exibe o histórico de alertas recebidos via Firebase Cloud Messaging (FCM). Esta tela consome dados do armazenamento local (NotificationStorage), permitindo a consulta *offline*.

- Aba de Últimos Chamados: Lista os chamados mais recentes abertos ou atualizados no GLPi. Utiliza Cards para resumir informações essenciais como título, localização (ex: "Salão Principal"), solicitante e horário.

Figura 8 – Interface da Tela Principal (Chamados e Notificações)



Fonte: Elaborado pelo autor (2026).

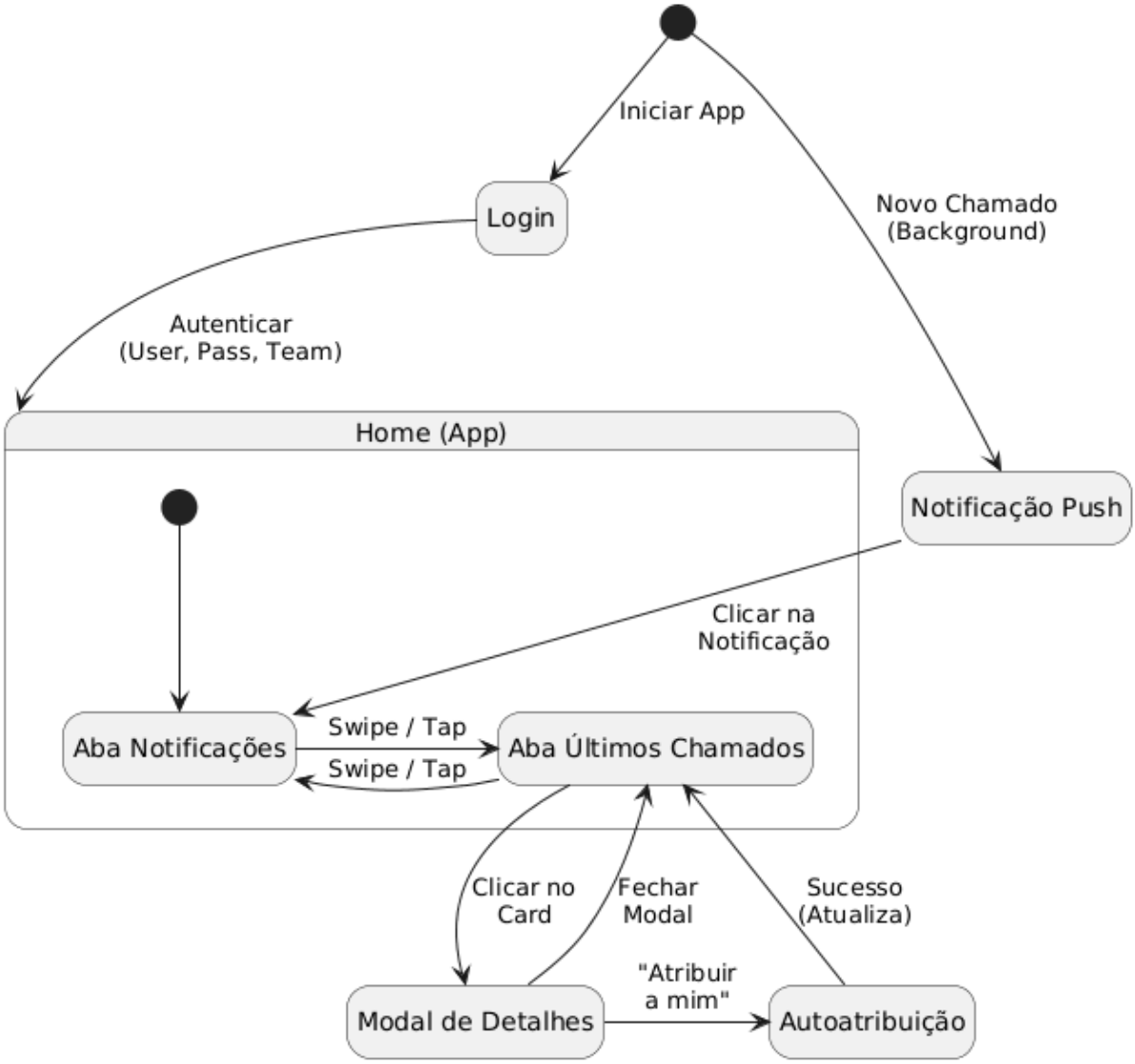
3. Modal de Detalhes (Dialog):

- Propósito: Exibir informações completas de um *ticket* sem trocar de tela, mantendo o contexto da lista.
- Elementos: Um Dialog sobreposto que apresenta a descrição detalhada do problema e a data/hora da abertura.
- Ação Principal: Contém o botão de destaque “Atribuir a mim”, que dispara a lógica de autoatribuição no *middleware*.

3.6.2 Fluxo de Navegação do Usuário

O fluxo de navegação foi simplificado para reduzir a carga cognitiva do técnico em situações de urgência. O diagrama abaixo utiliza PlantUML para ilustrar as transições entre estados e telas:

Figura 9 – Fluxo de Navegação de Usuário



Fonte: Elaborado pelo autor (2026).

3.6.3 Justificativa do Design (UI/UX)

- Uso de Cards: Selecionados para resumir informações, permitindo que o técnico escaneie visualmente a lista de chamados de forma rápida.
- Dialogs para Detalhes: A escolha por modais em vez de novas telas evita que o usuário se perca na navegação profunda, permitindo retornar à lista principal instantaneamente após

ler os detalhes ou realizar uma atribuição.

- Navegação por Abas: Proporciona uma separação clara entre o histórico de eventos (Notificações) e o estado atual do sistema (Chamados), ambos acessíveis com um único toque.
- JSON Leve: Embora invisível para o usuário, a interface é otimizada pelo *middleware* para carregar apenas os dados estritamente necessários, garantindo que as telas carreguem rapidamente mesmo em redes móveis 4G/5G.

4 SOFTWARE

4.1 IMPLANTAÇÃO

A estrutura de implantação foi projetada para suportar a lógica *multi-tenant* e a proteção rigorosa de dados sensíveis, seguindo os requisitos do modelo institucional.

4.1.1 Infraestrutura de Nuvem e Conteinerização (AWS & Docker)

O ambiente de execução da camada de aplicação e dados é sustentado pela infraestrutura da Amazon Web Services (AWS) (AMAZON WEB SERVICES, 2026).

- Hospedagem do *Middleware*: O serviço desenvolvido em Django REST *Framework* é implantado em uma instância Amazon EC2, utilizando Docker (DOCKER INC., 2026) para garantir a consistência do ambiente entre as fases de desenvolvimento e produção.
- Servidor Web e Proxy: O Nginx atua como servidor web e proxy reverso, sendo responsável por gerenciar as requisições criptografadas via HTTPS/SSL e encaminhá-las para o container do *middleware*.
- Banco de Dados Gerenciado: A camada de persistência, que utiliza o PostgreSQL, é alocada no serviço Amazon RDS. Esta escolha garante alta disponibilidade, backups automáticos e o isolamento lógico necessário para orquestrar múltiplos ambientes de suporte de forma independente.
- Segurança de Ambiente: Como medida crítica de segurança, a chave mestra de criptografia reside exclusivamente nas variáveis de ambiente da instância EC2. Isso assegura que, mesmo em caso de um acesso indevido ao banco de dados, os segredos das equipes não possam ser descriptografados sem o acesso ao servidor de aplicação.

4.1.2 Integração com o Sistema Legado (GLPi)

A comunicação entre o sistema legado e o novo ecossistema móvel é estabelecida através de configurações específicas no GLPi.

- Configuração de *Webhooks*: Devem ser ativados os *webhooks* nas instâncias do GLPi, apontando-os para o endpoint público gerenciado pelo Nginx na AWS.

- Sincronização de Eventos: Este fluxo permite que o *middleware* atue como um API *Gateway*, recebendo alertas instantâneos de novos chamados e orquestrando o disparo de notificações via Firebase Cloud Messaging (FCM) para os dispositivos móveis.

4.1.3 Empacotamento e Distribuição de Artefatos (Cloudflare R2)

Face às burocracias, taxas e complexidades operacionais inerentes à publicação e aprovação rigorosa exigida por lojas virtuais de varejo massivo (Google Play Store e Apple App Store), o setor de operações elegeu mecanismos laterais e automatizados para a entrega corporativa dos artefatos digitais terminados (instaladores .APK e .IPA) às equipes de campo homologadas. Para essa finalidade distributiva foi provisionada infraestrutura isolada no ecossistema de armazenamento orientado a objetos compatível com os protocolos nativos de AWS S3, materializado através do produto Cloudflare R2. Essa robusta malha de distribuição de conteúdo, operando globalmente e na extremidade das redes corporativas (*edge computing*), confere disponibilidade praticamente infalível aos artefatos compactados da aplicação móvel, eliminando perigosamente os abusivos custos decorrentes de tarifas regressivas atreladas a picos intensos de uso da largura de banda (CLOUDFLARE, 2026).

4.1.4 Configuração Inicial de Times e Secrets

O processo de habilitação de uma nova equipe de suporte segue um fluxo administrativo rigoroso no portal.

- Portal Administrativo: O líder da equipe utiliza a interface web para cadastrar as credenciais (secrets) do GLPi, incluindo a URL da API, o User Token e o App Token.

- Processamento de Segurança: O portal envia esses dados via HTTPS para o *middleware*, onde são unificados em uma estrutura JSON serializada e protegidos por criptografia simétrica AES-256 antes de serem salvos no Amazon RDS.

- Ativação do Técnico: Após o cadastro, o sistema gera o identificador único cd_team. Este código deve ser inserido pelo técnico no momento do *login* no aplicativo para vincular seu dispositivo ao ambiente correto da sua respectiva equipe.

4.2 TESTES

A estratégia de testes foi concebida para validar tanto a operabilidade funcional quanto a integridade estrutural do sistema de suporte móvel.

4.2.1 Definição do Tipo de Teste

A metodologia adotada foca em dois pilares fundamentais:

- Testes de Integração: Sendo o foco principal do projeto, estes testes visam garantir a comunicação fluida entre as três camadas distintas: o Aplicativo Flutter, o *Middleware* Django e a API nativa do GLPi. O objetivo é certificar que o fluxo de dados e o processamento de requisições ocorram sem interrupções entre os componentes.
- Testes de Segurança: São essenciais para validar se o *middleware* está operando corretamente como um API *Gateway* seguro. Estes testes verificam a eficácia da criptografia dos segredos e a proteção contra acessos indevidos entre diferentes equipes.

4.2.2 Cenários de Testes de Integração

Os testes de integração buscam validar os resultados práticos das funcionalidades centrais descritas nos requisitos funcionais:

- Autenticação (RF001): Consiste em validar se o *login* realizado com as credenciais do GLPi e o código único da equipe (cd_team) vincula o técnico corretamente à instância específica de sua empresa.
- Recebimento de Notificações *Push* (RF003): Verifica se a criação de um novo chamado no sistema legado dispara um alerta instantâneo no dispositivo móvel através do Firebase Cloud Messaging (FCM) (GOOGLE, 2026b), mesmo com o aplicativo em segundo plano.
- Autoatribuição de Chamado (RF004): Testa se a ação "Atribuir a mim", executada via interface móvel, reflete a mudança imediata do técnico responsável no banco de dados original do GLPi.
- Persistência e Histórico *Offline* (RF005): Valida se as notificações armazenadas localmente pela classe NotificationStorage permanecem acessíveis para consulta quando o técnico está sem conexão à rede de dados.

4.2.3 Cenários de Testes de Segurança

Focados nos requisitos não funcionais, estes cenários garantem a proteção da informação e o isolamento de dados:

- Criptografia de Secrets: Garante que, no ato do cadastro de uma nova equipe, as credenciais de API sejam serializadas em JSON e protegidas por criptografia simétrica AES-256 no banco de dados central.
- Comunicação HTTPS (RNF002): Verifica se toda a troca de dados entre as camadas é obrigatoriamente realizada via protocolo HTTPS/SSL, impedindo a interceptação de informações em texto claro.
- Isolamento *Multi-tenant*: Assegura que um técnico autenticado em uma equipe jamais consiga acessar chamados, notificações ou dados de configuração pertencentes a outro time configurado no *middleware*.

4.2.4 Tabela de Resultados

A tabela a seguir consolida os cenários projetados, detalhando o comportamento esperado para cada funcionalidade validada.

Ao completar estas validações, a lacuna de implementação é preenchida por uma verificação técnica robusta que comprova a viabilidade da modernização do sistema ITSM legado através do desacoplamento de serviços.

Tabela 4 – Resultados dos Testes

Caso de Teste	**Resultado Esperado**	**Resultado Obtido**
CT01: Autenticação via cd_team	O sistema deve validar o técnico e vinculá-lo exclusivamente ao ambiente da sua equipe no GLPi.	Autenticação finalizada com sucesso (Código HTTP 200). O aplicativo validou as credenciais via *Middleware*, vinculando o aparelho apenas ao banco de dados da equipe correta.
CT02: Recebimento de *Push* (Real-time)	O dispositivo deve receber a notificação segundos após a abertura de um *ticket* no GLPi.	Notificação capturada perfeitamente em segundo plano. Baixa latência registrada entre a criação do *ticket* e o alerta do Firebase na tela do aparelho.
CT03: Autoatribuição Remota	O chamado no GLPi deve ser atualizado com o nome do técnico que realizou a ação pelo aplicativo.	Ação concluída com sucesso. O clique disparou um POST e a API do GLPi acatou a ordem, registrando o técnico de campo como responsável instantaneamente.
CT04: Consulta *Offline*	O histórico de notificações deve estar disponível para leitura mesmo sem conexão de dados ativa.	Com o celular em Modo Avião, o *app* não travou e conseguiu renderizar 100% das notificações antigas salvas na memória local persistente do dispositivo.
CT05: Criptografia AES-256	As credenciais no banco de dados PostgreSQL devem estar em formato ilegível (ciphertext).	Inspeção no PostgreSQL confirmou que a coluna armazena apenas textos ilegíveis (ciphertext), protegendo as senhas com eficácia sem o acesso à chave mestra.
CT06: Tráfego HTTPS/SSL	Todas as requisições capturadas devem apresentar criptografia de transporte ativa.	A verificação da rede provou que a comunicação entre o aplicativo e a AWS utiliza túneis criptografados HTTPS, impedindo interceptações em texto claro.
CT07: Isolamento de Dados	Tentativas de acesso a *tickets* de outros times devem ser bloqueadas pelo *middleware*.	A manipulação intencional para acessar outra equipe foi barrada instantaneamente pelo *Middleware* (Erro 403), provando o isolamento da estrutura *multi-tenant*.

Fonte: Elaborado pelo autor (2026).

5 CONSIDERAÇÕES FINAIS

O presente Trabalho de Conclusão de Curso atingiu seu objetivo geral de desenvolver e arquitetar uma aplicação móvel multiplataforma integrada ao sistema GLPi, focada em otimizar o fluxo de atendimento das equipes de suporte de TI. A solução demonstrou ser uma ferramenta eficaz para estender as capacidades de gerenciamento de serviços (ITSM) para o ambiente móvel, superando a dependência histórica de estações de trabalho fixas.

5.1 SÍNTESE DOS RESULTADOS

A implementação técnica confirmou a eficiência das escolhas arquiteturais adotadas:

- Camada de Apresentação: O uso do *framework* Flutter e da linguagem Dart garantiu a compatibilidade nativa com Android e iOS a partir de uma única base de código, oferecendo uma experiência de usuário ágil e intuitiva. O padrão MVVM permitiu o desacoplamento necessário para garantir a testabilidade e a manutenção facilitada do aplicativo.
- Camada de Aplicação (*Middleware*): O serviço desenvolvido com Django REST *Framework* cumpriu seu papel essencial como API *Gateway* seguro, realizando o tratamento de dados pesados do sistema legado e entregando um JSON leve para o dispositivo móvel.
- Funcionalidades de Impacto: A implementação bem-sucedida de notificações *push* via Firebase Cloud Messaging (FCM) garantiu que os técnicos recebessem alertas em tempo real sobre chamados críticos. Além disso, a funcionalidade de "autoatribuição"de chamados proveu a agilidade necessária para que a equipe assumisse responsabilidades em campo, eliminando gargalos operacionais.

5.2 TRABALHOS FUTUROS

Embora a solução atual resolva os principais problemas de mobilidade e comunicação imediata, o design foi concebido para permitir evoluções incrementais. Como trabalhos futuros, sugerem-se os seguintes avanços:

- Ciclo de Vida Completo do *Ticket*: Expandir as funcionalidades para incluir o gerenciamento total do chamado diretamente no aplicativo, permitindo a inclusão de comentários,

a atualização de status (ex: de "Pendente"para "Solucionado") e o encerramento formal do *ticket*.

- Gestão de Mídias: Implementar o upload de mídias, como fotos de evidências técnicas ou documentos, vinculando-os diretamente ao chamado através da câmera ou galeria do dispositivo móvel.
- Expansão de Atributos de Autenticação: Graças à lógica de serialização no *middleware*, o sistema pode ser atualizado para suportar novos parâmetros de autenticação ou configurações específicas de diferentes versões do GLPi sem a necessidade de alterar a estrutura física das tabelas no banco de dados.
- Otimização de *Webhooks*: Refinar os gatilhos de eventos no GLPi para suportar notificações ainda mais segmentadas por categorias de urgência ou impacto.

Por fim, conclui-se que o presente projeto atinge seu objetivo ao atestar a viabilidade técnica de estender as capacidades de sistemas de gestão legados, como o GLPi, por meio da adoção de arquiteturas baseadas em microsserviços e APIs. O desacoplamento proporcionado pelo *middleware* e o desenvolvimento da interface móvel resultaram em um produto de software funcional que resolve o gargalo da dependência de estações fixas. Dessa forma, a solução desenvolvida promove uma maior eficiência operacional para as equipes de campo e, por consequência, eleva a qualidade dos serviços de TI entregues aos usuários finais.

REFERÊNCIAS

AMAZON WEB SERVICES. **Amazon Elastic Compute Cloud (Amazon EC2) Documentation**. Seattle, WA: AWS, 2026. Disponível em: https://docs.aws.amazon.com/ec2/. Acesso em: 10 maio 2026.

AXELOS LIMITED. **ITIL Foundation: ITIL 4 Edition**. London: TSO, 2019. 221 p.

CLOUDFLARE. **Cloudflare R2 Documentation**. San Francisco, CA: Cloudflare, Inc., 2026. Disponível em: https://developers.cloudflare.com/r2/. Acesso em: 10 maio 2026.

DOCKER INC. **Docker Documentation**. San Francisco, CA: Docker Inc., 2026. Disponível em: https://docs.docker.com/. Acesso em: 10 maio 2026.

ENCODE. **Django REST framework: Web APIs for Django, made easy**. Encode OSS, 2026. Disponível em: https://www.django-rest-framework.org/. Acesso em: 10 maio 2026.

GLPI PROJECT. **GLPi: Gestionnaire Libre de Parc Informatique**. Paris: Teclib, 2026. Disponível em: https://github.com/glpi-project/glpi. Acesso em: 10 maio 2026.

GOOGLE. **Dart Programming Language Specification**. Mountain View, CA: Google Developers, 2026. Disponível em: https://dart.dev/resources/language/spec. Acesso em: 10 maio 2026.

GOOGLE. **Firebase Cloud Messaging Documentation**. Mountain View, CA: Google Developers, 2026. Disponível em: https://firebase.google.com/docs/cloud-messaging. Acesso em: 10 maio 2026.

GOOGLE. **Flutter: Build apps for any screen**. Mountain View, CA: Google Developers, 2026. Disponível em: https://flutter.dev/. Acesso em: 10 maio 2026.

GOSSMAN, J. **Introduction to Model/View/ViewModel pattern for building WPF apps**. MSDN Blogs, Microsoft, 2005. Disponível em: https://learn.microsoft.com/en-us/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps. Acesso em: 10 maio 2026.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **FIPS PUB 197: Advanced Encryption Standard (AES)**. Gaithersburg, MD: U.S. Department of Commerce, 2001. Atualizado em 2023. Disponível em: https://doi.org/10.6028/NIST.FIPS.197-upd1. Acesso em: 10 maio 2026.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL 16 Documentation**. PostgreSQL Global Development Group, 2026. Disponível em: https://www.postgresql.org/docs/. Acesso em: 10 maio 2026.

ANEXO A – DECLARAÇÃO DE ENTREGA

Ministério da Educação
Instituto Federal de Educação, Ciência e Tecnologia do Piauí
IFPI - CAMPUS TERESINA CENTRAL
Praça da Liberdade, 1597, Centro, TERESINA / PI, CEP 64.000-040
Fone: Site: www.ifpi.edu.br

DECLARAÇÃO 17/2026 - CCADS/DIAS/DENS/DG-TERCENT/CATCE/IFPI

TERESINA, 7 de junho de 2026.

Declaro, para os devidos fins, que o aluno do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, Kawã Sousa de Lima, matrícula 2024111TADS0019, entregou à coordenação do referido curso o código-fonte do relatório de software (Trabalho de Conclusão de Curso).

ALINE MONTENEGRO LEAL

Coordenadora do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Documento assinado eletronicamente por:

- **Aline Montenegro Leal, COORDENADOR(A) DE CURSO - FUC0001 - CCADS-IFPI - CAMPUS TERESINA CENTRAL**, em 07/06/2026 15:21:35.

Este documento foi emitido pelo SUAP em 07/06/2026. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpi.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 457156
Código de Autenticação: 77913ca73c

Fonte: Instituto Federal do Piauí (2026).