UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

FERNANDO DOS SANTOS PAETZOLD

INTEGRAÇÃO DE UM SISTEMA LEGADO DE REGISTRO DE INCIDENTES COM UMA SOLUÇÃO DE UNIDADE DE RESPOSTA AUDÍVEL DE CÓDIGO ABERTO

TOLEDO

2024

FERNANDO DOS SANTOS PAETZOLD

INTEGRAÇÃO DE UM SISTEMA LEGADO DE REGISTRO DE INCIDENTES COM UMA SOLUÇÃO DE UNIDADE DE RESPOSTA AUDÍVEL DE CÓDIGO ABERTO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.
Orientador: Prof. Me. Marcelo Alexandre da Cruz Ismael

TOLEDO

2024

FERNANDO DOS SANTOS PAETZOLD

INTEGRAÇÃO DE UM SISTEMA LEGADO DE REGISTRO DE INCIDENTES COM UMA SOLUÇÃO DE UNIDADE DE RESPOSTA AUDÍVEL DE CÓDIGO ABERTO

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

Data de aprovação: 05/09/2024

_____

Marcelo Alexandre da Cruz Ismael
Mestre, Universidade Tecnologica Federal do Paraná (UTFPR), 2016

_____

Fabio Alexandre Spanhol
Doutor, Universidade Federal do Paraná (UFPR), 2018

_____

Edson Tavares de Camargo
Doutor, Universidade Federal do Paraná (UFPR), 2017

TOLEDO

2024

Dedico este trabalho aos meu orientador Marcelo Ismael que esteve presente para me auxiliar no desenvolvimento do projeto.

AGRADECIMENTOS

Agradeço imensamente a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho. Em especial, gostaria de manifestar minha profunda gratidão ao professor Me. Marcelo Alexandre da Cruz Ismael, pela orientação segura e paciente durante todo o processo. Sua expertise e dedicação foram fundamentais para o sucesso desta pesquisa.

Aos meus colegas de curso, agradeço a amizade, companheirismo e troca de conhecimentos. A cada um dos professores que tive a oportunidade de ter, meu mais sincero agradecimento pelos ensinamentos valiosos e pela inspiração.

À Universidade Tecnológica Federal do Paraná, agradeço a estrutura e recursos disponibilizados, que foram essenciais para a conclusão deste trabalho. A minha família, meu porto seguro, agradeço pelo apoio incondicional, pela confiança e pelo amor que me impulsionaram a seguir em frente.

Tenho certeza de que muitos outros nomes poderiam ser citados aqui, mas gostaria de registrar minha gratidão a todos que, de alguma forma, fizeram parte desta jornada. Muito obrigado a todos!

A primeira regra de qualquer tecnologia utilizada nos negócios é que a automação aplicada a uma operação eficiente aumentará a eficiência. A segunda é que a automação aplicada a uma operação ineficiente aumentará a ineficiência.

(BILL GATES, 1995).

RESUMO

O Departamento de Tecnologia da Informação (TI) da Prefeitura Municipal de Santa Helena, Paraná, enfrenta um alto volume diário de solicitações telefônicas para resolução de incidentes. Para otimizar a gestão dessas demandas e reduzir a carga de trabalho da equipe de suporte, este trabalho propõe a implementação de uma solução de Unidade de Resposta Audível (URA) integrada ao sistema de chamados Web existente. A URA foi desenvolvida utilizando o sistema de telefonia VoIP FreePBX, permitindo a automação da abertura de chamados sem a necessidade de intervenção direta da equipe de suporte. É esperado que a integração da URA com o sistema de chamados proporcionará maior eficiência no atendimento, agilizando o registro e o acompanhamento das solicitações.

Palavras-chave: URA, VOIP, FREEPBX, API REST, SDK

ABSTRACT

The Information Technology (IT) Department of the City Hall of Santa Helena, Paraná, handles a high volume of daily telephone communications for incident resolution. To optimize the management of these demands and reduce the workload of the support team, this paper proposes the implementation of an Audio Response Unit (IVR) solution integrated with the existing Web call system. The IVR was developed using the FreePBX VoIP telephony system, allowing the automation of call opening without the need for direct intervention by the support team. It is expected that the integration of the IVR with the call system will provide greater efficiency in service, speeding up the registration and monitoring of transfers.

Keywords: IVR, VOIP, FREEPBX, API REST, SDK

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AMP	*Asterisk Management Portal*
API	*Application Protocol Interface*
AWS	*Amazon Web Services*
CGI	*Common Gateway Interface*
CPU	*Central Processing Unit*
CTI	*Computer and telephony integration*
HTML	*Hypertext Markup Language*
IVR	*Interactive Voice Response*
PHP	*Hypertext Preprocessor*
PR	Paraná
REST	*Representational State Transfer*
SDK	*Software Development Kit*
SLA	*Service Level Agreement*
SQL	*Structure Query Language*
TCC	Trabalho de conclusão de curso
TI	Tecnologia da Informação
URA	Unidade de Resposta Audível

SUMÁRIO

1 INTRODUÇÃO

A Prefeitura Municipal de Santa Helena, situada no oeste do Paraná, conta com um quadro de aproximadamente mil e quinhentos funcionários, distribuídos entre os distritos do município. O município atende tanto servidores concursados quanto terceirizados, como professores, médicos, dentistas, gestores e profissionais de serviços gerais, que desempenham suas atividades sociais, administrativas e burocráticas com o apoio da infraestrutura de Tecnologia da Informação (TI). A equipe de TI é composta por seis profissionais responsáveis não apenas pela manutenção dessa infraestrutura, mas também pelo suporte técnico aos usuários, solucionando problemas e esclarecendo dúvidas que surgem no cotidiano de trabalho.

Por se tratar de órgão público, com setores sensíveis como saúde, educação e segurança, a continuidade dos serviços de TI é fundamental. Atualmente, o registro de incidentes é realizado por meio de contato telefônico, sendo que o técnico que atende a chamada efetua o registro manual da solicitação no sistema de chamados Web. Essa abordagem, embora funcional, demanda tempo dos técnicos e pode ocasionar atrasos na resolução dos problemas, especialmente em momentos de alta demanda.

Visando otimizar o processo de atendimento e aprimorar o atendimento ao usuário, este trabalho propõe o desenvolvimento e a implementação de uma solução de Unidade de Resposta Audível (URA), integrada ao sistema de chamados Web já existente. A URA permitirá que os usuários registrem suas solicitações de forma autônoma, sem a necessidade de intervenção direta de um técnico, agilizando o processo e assegurando que os chamados sejam atendidos dentro dos prazos estabelecidos. A expectativa que essa solução contribua para a redução de reclamações referente ao atendimento prestado pelo departamento e agilizar abertura dos incidentes para que seja possível ser resolvido o mais rápido possível.

1.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver um fluxo de URA (Unidade de Resposta Audível) utilizando a ferramenta *de código aberto FreePBX*, baseada em *Asterisk*, e integrá-lo a um sistema de chamados Web existente. Essa solução visa oferecer aos usuários a opção de realizar a abertura automatizada de chamados e/ou incidentes por meio de um ramal telefônico, sem a necessidade de intervenção direta da equipe de TI.

Para alcançar esse objetivo, será realizada a integração da solução de telefonia com a API REST do sistema atual de registro de incidentes, permitindo a comunicação e o fluxo de informações entre os dois canais de atendimento.

1.2 Justificativa

Devido ao crescimento da demanda de serviços prestados pelo departamento de TI, surge a necessidade de facilitar o atendimento aos demais servidores públicos garantindo que todas as solicitações de atendimento de incidentes sejam registradas e atendidas. Outro ponto importante é ter a gestão desses atendimentos para fins de auditoria, análise das ocorrências, controle de satisfação do solicitante, definir quais são os serviços mais utilizados e o nível de criticidade deles. Garantido a qualidade dos serviços prestados pelo departamento de TI.

A integração entre o sistema de chamados e a central telefônica busca melhorar o atendimento ao usuário de várias maneiras:

Eficiência: Com automação de abertura de chamados, a equipe de TI pode se designar maior parte de tempo na resolução de problemas, em vez de direcionar tempo registrando e classificando chamados. Isso pode levar a uma resolução mais rápida dos problemas.

Acessibilidade: Os usuários podem abrir chamados diretamente pelo telefone dentro ou fora do horário de atendimento da equipe de TI, sem a necessidade de treinamento específico ou conhecimento técnico. Isso torna o sistema mais acessível para todos os usuários.

Rastreabilidade: Todos os chamados são registrados e classificados automaticamente no sistema atual de chamados Web, o que facilita na classificação para atendimento deles.

Redução do volume de chamadas: Ao permitir que os usuários façam abertura de chamados por meio da URA, o volume de chamadas diretas para a equipe de TI pode ser consideravelmente reduzido.

1.3 Problema e Hipótese

A Prefeitura Municipal de Santa Helena, com uma equipe de TI reduzida, enfrenta dificuldades na gestão de solicitações de suporte, principalmente devido à dependência do atendimento telefônico. Essa dependência causa atrasos, perda de informações, falta de registro de novas solicitações e sobrecarga da equipe técnica. A situação é agravada quanto toda equipe está em campo atendendo solicitações anteriores sendo impossível realizar atendimento telefônico.

A implementação de uma URA para abertura de chamados centralizará as solicitações, agilizando o atendimento, otimizando o trabalho da equipe de TI e melhorando a experiência do usuário disponibilizando mais um canal de atendimento.

2 REFERENCIAL TEÓRICO

Com base neste referencial teórico, na seção 2.1 será explorada a tecnologia FreePBX, uma interface web utilizada para gerenciar o Asterisk, que é um servidor de voz sobre IP e telefonia de código aberto. Serão apresentados os principais módulos do Asterisk e as funcionalidades do FreePBX, relevantes para a implementação da URA, como a criação de menus de navegação, reconhecimento de voz e integração com outras aplicações, como a criação de menus de navegação, reconhecimento de voz e integração com outras aplicações.

Em seguida, a seção 2.2, a linguagem de programação PHP, utilizada para o desenvolvimento dos scripts da URA e scripts das APIs Rest. São destacadas suas características, como a flexibilidade e a facilidade de integração com as aplicações, e sua adequação para a criação de scripts no lado do servidor.

Na seção 2.3, será apresentado o sistema de gerenciamento de banco de dados MySQL, essencial para o armazenamento e organização de informações como dados dos chamados ou dados de usuários. Também são abordados seus principais recursos e vantagens, como o suporte a múltiplas camadas e a alta performance.

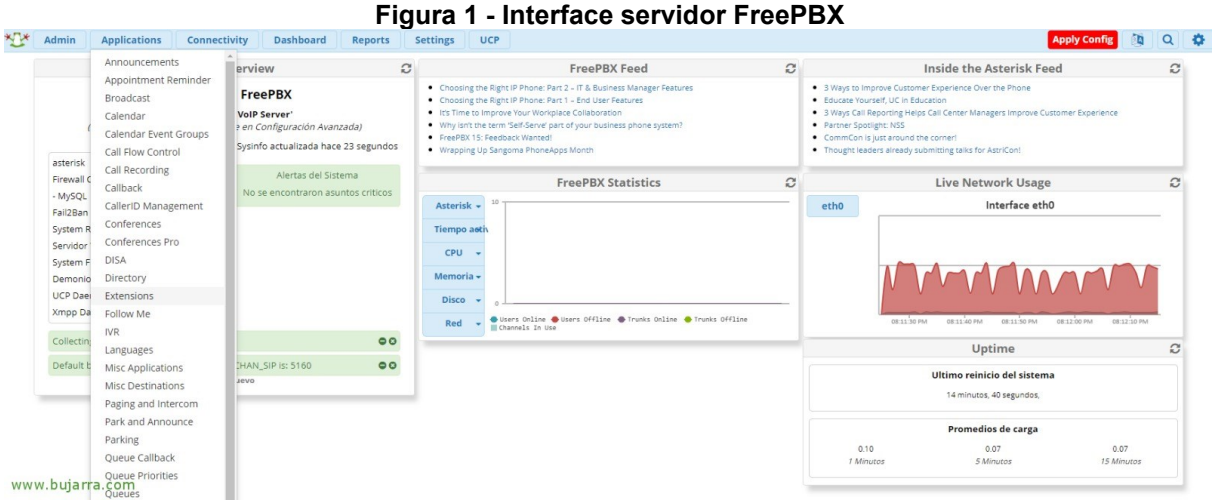
A seção 2.4 explorará o conceito de APIs REST (*Representational State Transfer*), fundamentais para a integração da URA com o sistema de chamados existente. Serão explicados os princípios de *design* do REST e como eles garantem uma comunicação eficiente e escalável entre diferentes sistemas.

Na seção 2.5, será apresentada a plataforma de nuvem *Amazon Web Services* (AWS), que oferece os serviços necessários para conclusão da solução URA, incluindo armazenamento de dados, computação, análise, rede, ferramentas de desenvolvimento e segurança. A AWS será utilizada para receber informações do servidor FreePBX, analisar essas informações e designar para o serviço necessário e retornar informação solicitada para a solução de URA, segurança e alto desempenho.

Na seção 2.6, será discutido o conceito de CTI (*Computer Telephony Integration*), que permite a integração entre sistemas de telefonia e computação. São abordadas as vantagens do uso de CTI em centrais de atendimento, como a identificação e roteamento de chamadas.

2.1 FreePBX - *Asterisk*

Figura 1 - Interface servidor FreePBX



Fonte: Autoria própria (2024)

O FreePBX consiste em um *software* de código aberto voltado para realizar o gerenciamento das chamadas telefônicas de uma empresa. Ou seja, “é uma interface gráfica de usuário (GUI) de código aberto baseada na Web que gerencia *Asterisk*, um servidor de voz sobre IP e telefonia” (Madsen; Meggelen; Bryant, 2013). Segundo os autores, o FreePBX permite configurar e gerenciar diversos aspectos de um sistema Asterik “sem tocar em um único arquivo de configuração” (Madsen; Meggelen; Bryant, 2013, p. 800).

Figura 2 - Comunicação com servidor asterisk

Fonte: Autoria própria (2024)

O desenvolvimento do *Asterisk* ocorreu em 1999 com Mark Spencer, da empresa Digium, que atualmente é parte da *Sangoma Technologies Corporation*. Inicialmente, foi concebido como um *Private Branch Exchange* (PBX privado), gerando condições para a conexão de chamadas em uma organização e dela com a rede telefônica pública (Lopes, 2015).

A princípio foi denominado como *Asterisk Management* Portal (AMP), só mais tarde passou a ser definido como FreePBX, tendo como objetivo fornecer uma interface amigável visando realizar a configuração e gerenciamento do *Asterisk*. O que foi de extrema importância, pois o *Asterisk* exige uma configuração complexa, necessitando conhecimentos em linha de comando e edição de arquivos (Madsen; Meggelen; Bryant, 2013).

De acordo com Menezes (2019), o funcionamento do *Asterisk* ocorre em quatro módulos principais:

a) Protocolo: estabelece a comunicação com o sistema, usando protocolos SIP ou *Inter Asterik Exchange* (IAX);

b) Canal de comunicação: estabelece diálogo entre os ambientes internos e externos de *hardwares* ou *softwares*;

c) CODES: estabelece a forma como o áudio é digitalizado;

d) Aplicação: estabelece o tratamento das chamadas, realizando as funcionalidades do *Asterik* tanto para receber quanto para fazer chamadas.

Com base nesses módulos, Lopes (2015) considera que as principais funcionalidades do *FreePBX - Asterisk* são as seguintes:

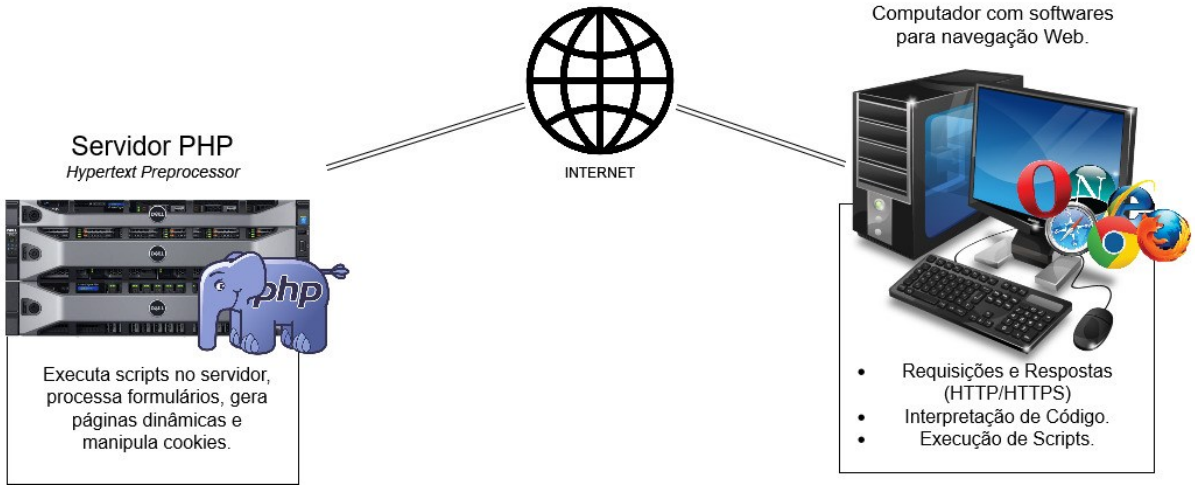
- **Correio de Voz**: Armazena as mensagens de voz, que podem ser enviadas para os usuários via correio eletrônico.
- **Unidade de Resposta Audível (URA):** Realiza o atendimento eletrônico das chamadas, permitindo a integração com outras aplicações e bancos de dados. Também possibilita criar menus de navegação e reconhecimento de voz.
- **Distribuidor Automático de Chamadas (DAC):** Distribui as chamadas de entradas no dispositivo (grupo ou serviço), através de algoritmos de distribuição (maior tempo livre ou menor tempo acumulado) aos agentes que estão ligados no respetivo serviço. Em *Call Centers* facilita o gerenciamento de filas, as prioridades das chamadas e direcionamento das chamadas para agentes específicos.
- **Conferência de áudio:** Cria salas de conferência que possibilitam vários usuários conversarem simultaneamente.

- **Discador automático:** Aplicação que gera chamadas a partir de uma base de dados (contendo número de telefones), direcionando a mesma para determinado ramal ou agente.
- **Servidor de música de espera:** Vários formatos de arquivos podem ser reproduzidos pelo *Asterisk*, de forma síncrona ou assíncrona. Essas reproduções normalmente ocorrem ao colocar uma chamada corrente em espera, ou até mesmo antes do atendimento da chamada (quem origina a chamado pode ouvir a música enquanto aguarda o atendimento).
- **Registro detalhado das ligações:** Relatórios completos sobre as ligações (duração, origem, destino, custo etc.).
- **Ramal local e remoto:** Independência geográfica, Dois ramais podem estar a geograficamente distantes, porém podem fazer parte do mesmo servidor (desde que exista acesso ao servidor via rede IP).
- **Interoperabilidade com diferentes padrões de VoIP:** Suporta praticamente todos os protocolos utilizados atualmente em telefonia e VoIP, portanto a migração e interligação com sistemas híbridos é altamente facilitada.

Além desses aspectos mencionados, é importante salientar que o *FreePBX* - *Asterisk* pode ser integrado a sistemas como CRM ou de e-mail, permitindo uma comunicação unificada, além de facilitar a configuração de chamadas de conferência, facilitando a comunicação simultânea de várias pessoas. Por possuir uma arquitetura modular facilita para que as empresas possam adicionar novas funcionalidades conforme a sua demanda. Em relação aos aspectos de segurança, conta com recursos para proteger e evitar acesso não autorizado, bem como de fraude, como *firewalls*, além de configurações de segurança para todas as extensões (Lopes, 2015).

2.2 PHP

Figura 3 - Como Funciona o PHP



Fonte: Autoria própria (2024)

PHP é um acrônimo para *Hipertext Preprocessor*, sendo uma linguagem de *script open source* de amplo uso, que é muito utilizada para o desenvolvimento Web, podendo ser encaixada em um HTML. Isso significa que o PHP é focado na execução de *scripts* do lado do servidor, desempenhando funções semelhantes às de um programa CGI (*Common Gateway Interface*), permitindo a criação de conteúdos dinâmicos e interativos para páginas Web, isso é:

a) Coletar dados de formulários;

b) Criar páginas com conteúdo dinâmico;

c) Receber e enviar *cookies*.

O PHP oferece uma ampla gama de aplicações, que em sua maioria estão nos sistemas operacionais, como Linux, Microsoft Windows, algumas variantes do Unix, macOS, RISC OS, dentre outros (Manual PHP, 2023).

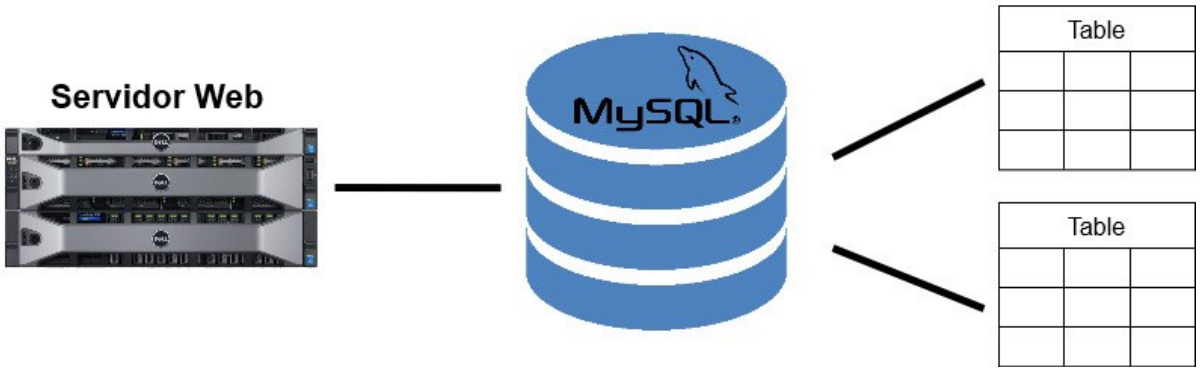
Existem áreas principais para o uso dos scripts PHP nesse tipo de linguagem, que são as seguintes:

a) Scripts no lado do servidor *(server-side*): tradicional e principal campo de atuação do PHP.

b) *Scripts* de linha de comando: Você pode fazer um script PHP para executá-lo sem um servidor ou navegador (Manual PHP, 2023).

2.3 Sistema Gerenciador de Banco de Dados MySQL

Figura 4 - Como funciona gerenciador de Banco de Dados Mysql



Fonte: Autoria própria (2024)

O MySQL é um sistema de gerenciamento de banco de dados relacional de código aberto, utilizando a linguagem *Structure Query Language* (SQL) ou Linguagem de Consulta Estruturada, que é desenvolvido, distribuído e suportado pela Oracle *Corporation*. Um banco de dados é uma coleção estruturada de dados. Pode ser qualquer coisa, desde uma simples lista de compras até uma galeria de fotos ou a vasta quantidade de informações em uma rede corporativa (Oracle 2023).

Essa estrutura relacional, aliada à linguagem SQL, permite armazenar e recuperar os dados de forma eficiente, possibilitando que os usuários acessem um grande volume de informações de maneira assertiva e organizada.

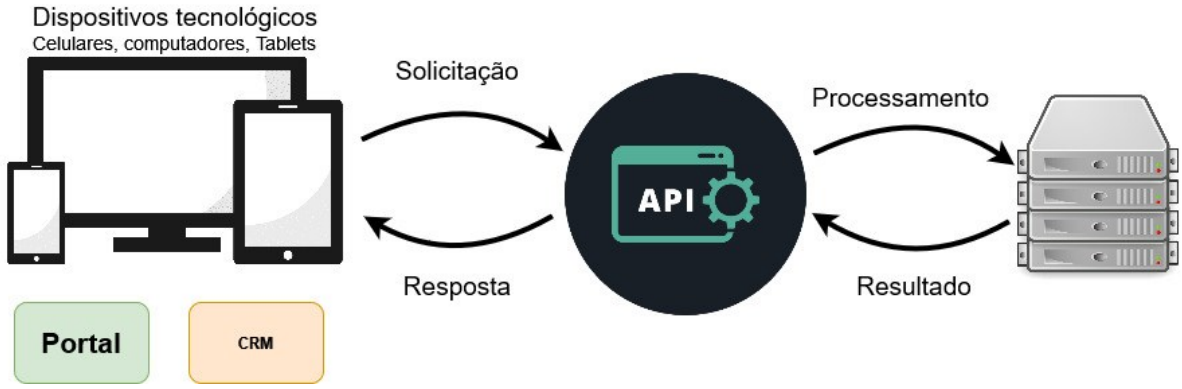
Os principais recursos do *software* de banco de dados MySQL, aplicados a todas as versões, são os seguintes:

- Usa *design* de servidor multicamadas com módulos independentes.
- Projetado para ser totalmente *multithread* usando *threads* de *kernel*, para usar facilmente várias CPUs, se estiverem disponíveis.
- Fornece mecanismos de armazenamento transacionais e não transacionais.
- O tipo nativo é o (InnoDB) com compactação de índice.
- Projetado para tornar relativamente fácil adicionar outros mecanismos de armazenamento. Isso é útil se você deseja fornecer uma interface SQL para um banco de dados interno.
- Usa um sistema de alocação de memória baseado em *thread* muito rápido.
- Executa junções muito rápidas usando uma junção de *loop* aninhada otimizada.

- Implementa tabelas *hash* na memória, que são usadas como tabelas temporárias.
- Implementa funções SQL usando uma biblioteca de classes altamente otimizada que deve ser mais rápida possível. Geralmente não há nenhuma alocação de memória após a inicialização da consulta.
- Fornece o servidor como um programa separado para uso em um ambiente de rede cliente/servidor (Oracle 2023).

2.4 API REST

Figura 5 - Como funciona uma API



Fonte: Autoria própria (2024)

Uma interface de programação de aplicativos ou API consiste em alguns critérios usados para especificar como os aplicativos ou dispositivos devem estabelecer comunicação e integração entre si. Nesse sentido, é preciso esclarecer sobre a API de REST ou APIs de *RESTful*, que é uma API adequada aos princípios de *design* do estilo de arquitetura do *Representational State Transfer* (REST) (IBM, 2023).

É interessante destacar que as APIs de *REST* são desenvolvidas com quase todas as linguagens de programação e uma grande variedade de formatos de dados, tendo como exigência que respeitem os princípios de *design* de *REST* ou restrições de arquitetura, que são:

- Interface uniforme: todas as solicitações da API para o mesmo recurso devem ser iguais, não importa a origem da solicitação.

- Desacoplamento do cliente-servidor: os aplicativos cliente e servidor devem ser completamente independentes um do outro.
- Sem estado definido: não possuem estado definido, o que significa que cada solicitação precisa incluir todas as informações necessárias para processá-lo.
- Capacidade de armazenamento em cache: quando possível, os recursos devem ser armazenados em cache pelo cliente ou servidor.
- Arquitetura de sistema em camadas: as chamadas e respostas passam por diferentes camadas.
- Código sob demanda (opcional): enviam recursos estáticos, mas em certos casos, as respostas também podem conter código executável (como applets Java) (IBM, 2023).

Portanto, uma API *REST* é um meio que permite estabelecer uma interoperabilidade nas internets para sistemas de computadores, uma vez que facilita a comunicação entre diferentes sistemas com a troca de dados de forma simples e estruturada.

2.5 AWS

AWS é a sigla para *Amazon Web Services*, que é a plataforma de nuvem de serviços da *Amazon*, que é a mais abrangente do mundo. Isso porque oferece mais de 200 serviços completos de datacenters no mundo inteiro, que incluem:

- Armazenamento de dados;
- Computação;
- Rede;
- Ferramentas de desenvolvedor;
- Ferramentas de gerenciamento;
- IoT;
- Segurança e aplicações empresariais (AWS, 2023).

Uma das vantagens oferecidas por esses serviços é facilitar a migração das operações da empresa para a nuvem, gerando a diminuição dos custos e aumento de escalabilidade, além de segurança e alto desempenho (AWS, 2023).

Em termos gerais, tanto as empresas quanto os desenvolvedores fazem uso da AWS para executar aplicações web, hospedar *websites*, armazenar e processar dados, e fazer a gestão de várias funções de negócios e processos de TI.

2.6 *Computer and telephony integration* (CTI)

Computer Telephony Integration ou CTI é uma tecnologia específica para promover a integração entre os sistemas de telefonia e os sistemas de computação, visto que o seu objetivo está voltado para duas ações específicas: melhorar a eficiência das comunicações e a funcionalidade dos serviços de telefonia (Blokdyk, 2018).

Desse modo, o aplicativo CTI pode ser um *software*, um computador com interface para o sistema telefônico ou um conjunto de diferentes equipamentos e sistemas integrados. Um exemplo bem conhecido é um *software* como o Skype que, instalado em um computador pessoal, passa a funcionar como interface telefônica, com agenda telefônica, registro de chamadas efetuadas e não atendidas, além de *pop-ups* de janelas com informações de quem lhe está a ligar (Blokdyk, 2023)

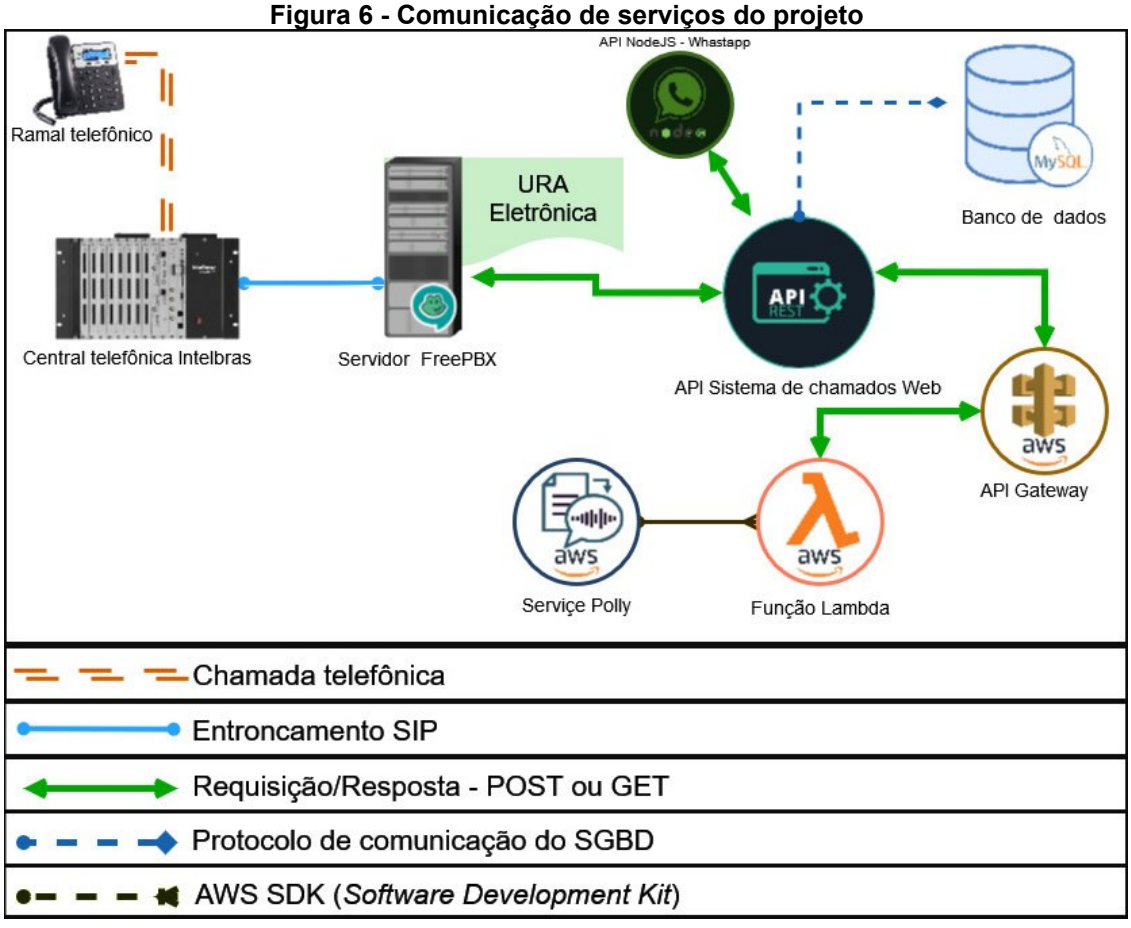
As principais vantagens em uma solução do tipo CTI estão relacionadas com a aumento de flexibilidade e escalabilidade, uma significativa redução de custos operacionais, aumento de produtividade do agente de atendimento, emissão de relatórios de produtividade e desempenho e monitoramento e acompanhamento da jornada do cliente (Blokdyk, 2018).

Para as centrais de atendimento, os principais recursos de um CTI estão relacionados com a identificação e roteamento de chamadas para o atendente e o registro de chamadas (Blokdyk, 2018).

3 DESENVOLVIMENTO DO PROJETO

Este capítulo detalha o desenvolvimento da solução URA, abordando as tecnologias e ferramentas utilizadas para conclusão da solução proposta. Cada uma dessas tecnologias desempenha um papel crucial na funcionalidade da solução proposta. A seção 3.1 oferece uma visão geral do projeto, descrevendo as possíveis interações do usuário ao realizar uma chamada para a URA eletrônica. Na seção 3.2, serão apresentadas as funcionalidades da solução URA, fornecendo a base para a elaboração dos diagramas de casos de uso detalhados na seção 3.3. Dando continuidade, na sequência a seção 3.4 abrange todos os serviços utilizados e suas respectivas configurações. Uma vez que os serviços estão configurados e em funcionamento, são realizados os testes de comunicação, conforme descrito na seção 3.5. Por fim, para assegurar a qualidade da solução proposta, a seção 3.6 apresenta os resultados da avaliação realizada com um seleto grupo de usuários.

Figura 6 - Comunicação de serviços do projeto



Fonte: Autoria própria (2024)

A Figura 6 ilustra a comunicação entre os serviços da aplicação. Inicialmente, quando o usuário liga para o ramal designado ao departamento de informática, a central telefônica direciona a chamada para o servidor *FreePBX*, que, por sua vez, apresenta a URA eletrônica. Ao selecionar uma opção na URA, o servidor requisita informações à API do sistema de chamados Web, que se comunica com os demais serviços, como o banco de dados utilizado para buscar e salvar informações, enviar mensagens ao usuário ou converter texto em áudio utilizando os serviços da AWS. A seção 3.1 aprofunda a compreensão sobre a comunicação entre os serviços da solução proposta.

3.1 Visão geral da solução

A solução URA visa oferecer aos usuários uma maneira simplificada de abrir chamados de suporte ao setor de TI, por chamada telefônica. O processo, seguirá os seguintes passos:

1° - O usuário disca para um ramal determinado utilizando seu telefone fixo, celular ou ramal telefônico.

2° - O atendimento é recebido pela central telefônica e encaminhado ao servidor FreePBX.

3° - Uma rota de entrada, configurada nesse servidor, encaminha a chamada para uma URA, denominada como URA PRINCIPAL que apresenta ao usuário as opções de:

1. Abrir um chamado
2. Consultar o status de um chamado já aberto anteriormente
3. Falar com um atendente

Caso o usuário opte pela opção “1”, o sistema lhe requisita, primeiramente, seu código de cadastro. O usuário informa digitando o código pelo teclado do telefone. Essa informação é validada através de uma requisição para API do servidor de chamados, que por sua vez consulta o banco de dados e realiza a verificação.

Caso o código digitado seja inválido, um *prompt* de voz alerta sobre o problema e solicita as informações novamente.

Uma vez validado o usuário, o texto contendo o nome dele é enviado para o serviço de *Text to Speech* da AWS, conhecido como Polly, utilizando para isso o serviço de *API GATEWAY* da mesma operadora de nuvem pública.

Esse serviço gera um arquivo de áudio que contém o nome do usuário e o envia de volta ao servidor requisitante, que por sua vez o armazena para que seja executado para que o usuário o ouça e confirme ou decline do atendimento.

Em paralelo a isso, é feita uma requisição a API do WhatsApp que gera um *token* numérico e realiza o envio dessa informação para o celular do usuário (obtido por consulta no cadastro) utilizando o aplicativo de mensagens WhatsApp.

Na próxima etapa, um *prompt* de voz solicita que o usuário digite os números do código recebido. Este processo foi adotado para evitar que algum agente inadvertido, malicioso ou mal-intencionado possa abrir chamados em nome de terceiros.

Se os números digitados forem corretos, um *prompt* de áudio com o nome do usuário é executado. O usuário deve confirmar sua identidade. Caso confirme, o usuário é encaminhado para outra etapa da URA. Caso decline, a ligação é encerrada.

No caso de confirmação, são apresentadas as opções de abertura de chamados, onde o requisitante escolhe para que tipo de serviço necessita apoio técnico.

1 - Sistemas
2 - Computador
3 - Impressoras
4 - Liberação de acessos
5 - Projetor multimídia
6 - Ramal telefônico
7 - Repetir os itens do menu

Cada uma dessas opções apresenta submenus com opções para melhor especificar a necessidade do usuário.

Uma vez finalizado o procedimento, o chamado é criado no sistema e um número de protocolo é gerado. Esse número é encaminhado ao serviço Polly da AWS, a fim de que possa ser ditado ao usuário para que, discricionariamente, seja anotado para consultas posteriores de status.

Caso a opção escolhida pelo usuário seja a “2” do menu principal, para consulta de chamados, o fluxo é o seguinte:

a) A URA requisita ao usuário seu número de cadastro

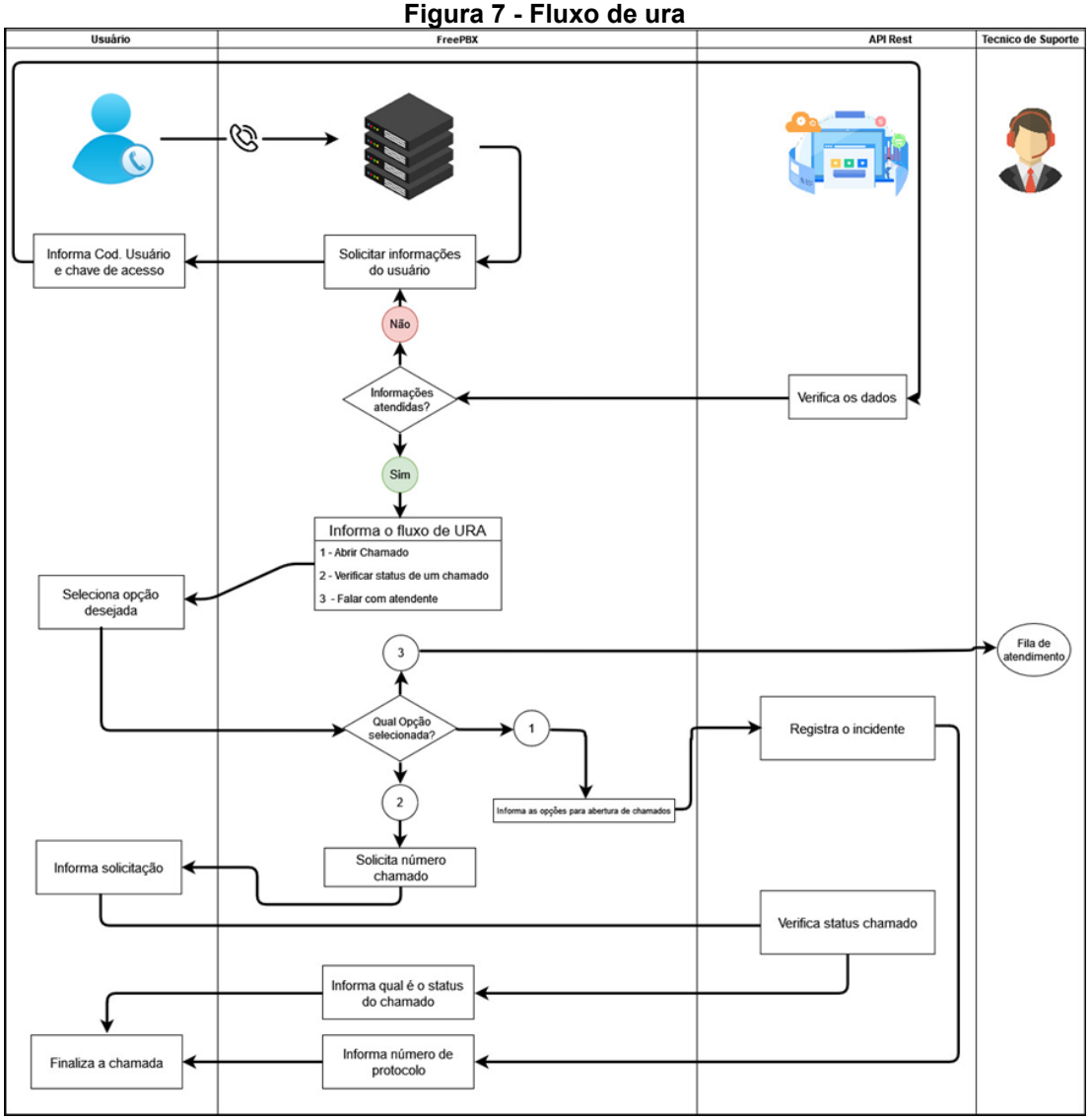
b) A URA requisita o número do chamado

c) Caso as informações estejam corretas, um áudio é executado, informando ao usuário o status do chamado em questão. Esse áudio também é elaborado pelo serviço de *Text to Speech* da AWS.

Finalmente, caso a opção escolhida seja a “3” do menu principal, o usuário tem sua chamada encaminhada a um técnico de atendimento do departamento de T.I.

A Figura 7 proporciona uma visão geral, destacando como funciona o fluxo de uma chamada entre o usuário e serviço de atendimento automatizado.

Figura 7 - Fluxo de ura



Fonte: Autoria própria (2024)

3.2 Modelagem da solução URA

A modelagem da solução URA apresentada inclui a integração entre a central telefônica Intelbras e o sistema de chamados Web, utilizando o FreePBX com *Asterisk*, baseado nessa perspectiva modular e escalável, visando facilitar a compreensão, implementação e manutenção. Foi elaborado o levantamento dos requisitos necessário para entender como a solução URA deve se comportar quando um usuário realizar uma chamada telefônica. Os diagramas de casos de uso e fluxogramas ilustram os principais componentes, interações e fluxos de informações do sistema como demonstrado nas subseções a seguir.

3.2.1 Requisitos funcionais

- **RF01** – O usuário poderá realizar abertura de chamados.
- **RF02** – O usuário poderá realizar consultas sobre um chamado.
- **RF03** – O usuário poderá solicitar atendimento com um técnico.
- **RF04** – A solução URA deverá consultar informações sobre chamados realizados.
- **RF05** – A solução URA deverá realizar abertura de chamados.

3.2.2 Requisitos não funcionais

- **RNF01** – A resposta das consultas com API's não devem exceder 15 segundos.
- **RNF02** – O código do sistema deve seguir as boas práticas de programação e ser bem documentado.

3.2.3 Regras de negócio

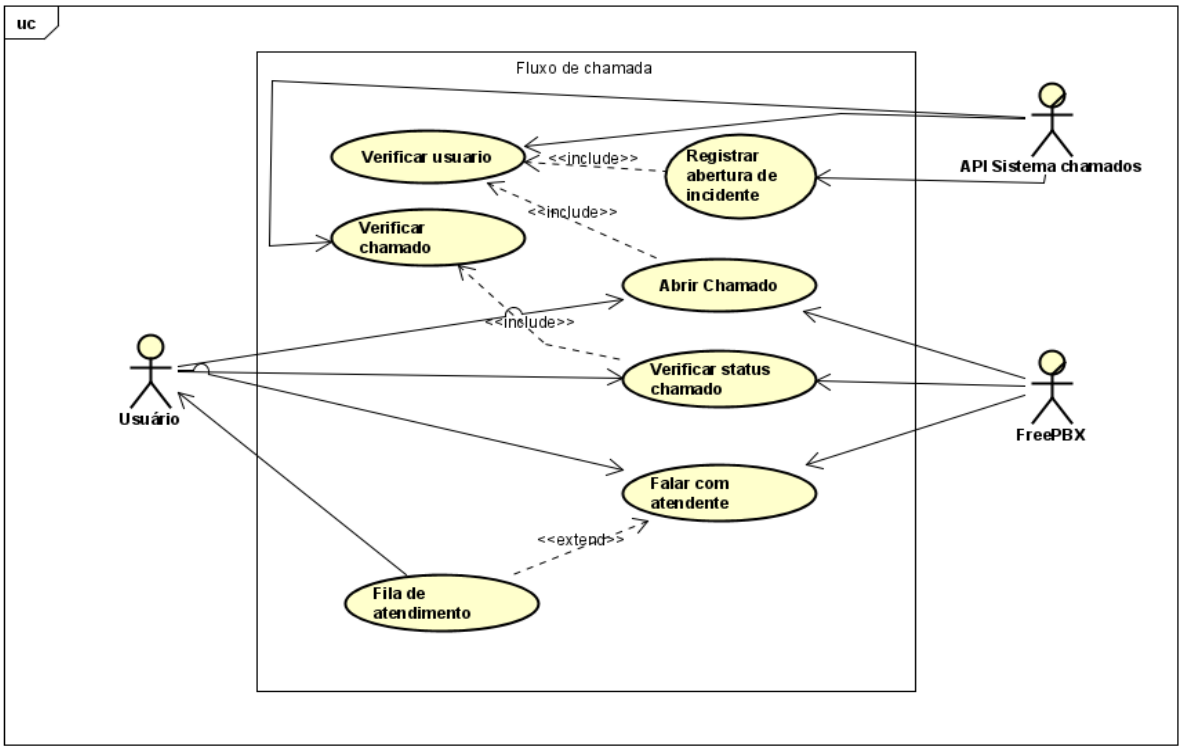
- **RN01** – O código de acesso deve ser corresponder com o código de usuário informado.
- **RN02** – Para falar com atendente o usuário deve ligar durante o horário de expediente.
- **RN03** – Todas as solicitações de chamados devem ser registradas no banco de dados do servidor.

3.3 Diagramas de casos de uso

Os diagramas de casos de uso a seguir desempenham um papel fundamental na compreensão da solução proposta, ilustrando de forma clara e concisa as interações entre os diferentes componentes do sistema e o usuário. Essa representação visual facilita a compreensão do fluxo de trabalho, auxiliando no desenvolvimento e na validação da solução.

3.3.1 Diagrama fluxo de chamada

Figura 8 - Diagrama de casos de uso fluxo de chamada



Fonte: Autoria própria (2024)

A Figura 8, ilustra a interação do sistema com os seus usuários e componentes ao realizar uma chamada telefônica para o departamento de TI. Entre essa interação é possível, destacar o usuário do sistema, API sistema de chamados e o *FreePBX*. As atividades de cada um deles são apresentadas da seguinte forma:

Usuário:

O usuário, ao efetuar a chamada telefônica para o ramal do departamento de I, é direcionado para a URA, onde lhe são apresentadas diversas opções de interação com o sistema.

O usuário pode optar por abrir um chamado, o que inicia o processo de registro de um problema ou solicitação.

Caso já possua um chamado em aberto, pode escolher a opção de verificar o status de chamado para acompanhar o andamento do atendimento.

Em algum momento durante a interação, o usuário pode ser transferido para um atendente, a fim de obter suporte em tempo real.

FreePBX:

O sistema telefônico VoIP gerencia as chamadas, responsável por verificar o usuário, através de um processo de autenticação para garantir que a pessoa tenha permissão para acessar os serviços.

Também verifica o chamado caso o usuário já tenha um aberto, buscando informações na API de chamados.

Quando necessário interage com a API do Sistema de Chamados para registrar a abertura de um novo incidente ou buscar informações sobre um chamado existente.

API Sistema Chamados:

O *webservice* responsável que permite a comunicação entre o *FreePBX* e o sistema onde os chamados são registrados e gerenciados.

Recebe informações do *FreePBX* para registrar a abertura de um incidente quando um novo chamado é iniciado.

3.4 Metodologia de Implementação

Após a compreensão do comportamento e das funcionalidades desejadas para a solução, procedeu-se à etapa de configuração e desenvolvimento dos ambientes e aplicações. As próximas subseções detalham como cada um desses componentes foi configurado e desenvolvido para realizar o entroncamento SIP da central telefônica com o servidor *FreePBX*, a integração entre a o servidor de telefonia VoIP com o sistema de chamados e os serviços em nuvem.

3.4.1 Central telefônica Intelbras

Inicialmente, foi configurado na central telefônica Intelbras um ramal específico para o departamento de informática. Este ramal, protegido por senha, estabelece uma conexão segura com o servidor *FreePBX*, permitindo o encaminhamento das ligações para o sistema de telefonia via internet (VoIP).

3.4.2 FreePBX

No servidor *FreePBX*, a configuração do tronco SIP estabeleceu a comunicação entre a central telefônica e o servidor, possibilitando o encaminhamento das chamadas para o servidor VoIP.

A modelagem dos casos de uso citados na seção 3.2, permitiu compreender o fluxo de chamadas e definir a estrutura da URA, com isso foi criado uma URA para cada tipo de serviço que o departamento de TI atende. Desta forma é possível categorizar os serviços e os usuários podem navegar entre os menus de serviços de uma forma mais fácil, sem ter a necessidade de ter que ouvir todas as inúmeras opções para selecionar o serviço que corresponde com a solicitação. Como demonstrado na tabela 1, estão listados todos as URAs e os respectivos serviços

Tabela 1 - Uras e serviços

URA	Serviços
URA Saudação	Abrir chamado
	Verificar o status de um chamado
	Falar com um atendente
Sistemas	Atualização do portal
	Problemas diversos
Computador	Formatação computador
	Computador não liga
	Sem conexão com a impressora
	Problema de conexão com a Internet
	Instalação de Software
Impressoras	Problemas diversos
	Instalar impressora
	Configurar Impressora
	Troca de suprimentos
Acessos	Privilégio - IPM
	Liberar acesso ao WIFI
	Liberar acesso a sites
	Liberar acessos a pastas do servidor
Projetor	Reserva de projetor
Ramal	Problema com ramal

Fonte: Autoria própria (2024)

3.4.2.1 PHP AGI

O servidor *FreePBX* utiliza a biblioteca AGI para estabelecer a integração com a API do sistema de chamados. Essa integração é realizada através de scripts PHP que controlam e manipulam as chamadas telefônicas em tempo real.

No contexto desta implementação, os scripts PHP que utilizam a biblioteca AGI desempenham as seguintes funções:

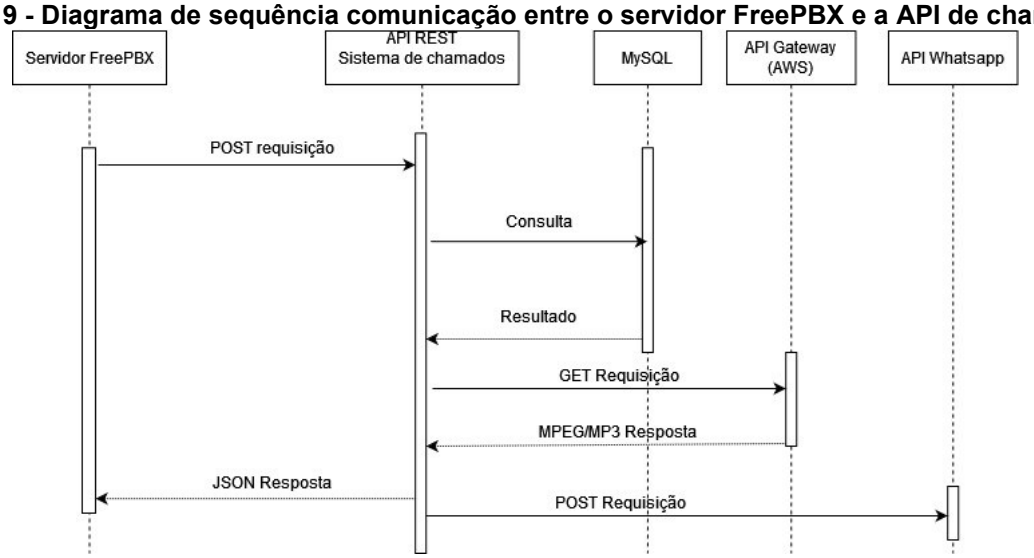
- **Comunicação com a API:** Os scripts PHP realizam requisições à API do sistema de chamados, enviando e recebendo informações relevantes para o atendimento da chamada.
- **Validação de Informações:** Os scripts PHP verificam se as informações inseridas pelo usuário durante a chamada, como o código de usuário, estão corretas e em conformidade com os requisitos do sistema.
- **Direcionamento para a URA:** Com base nas informações coletadas e validadas, os scripts PHP direcionam o usuário para a URA (Unidade de Resposta Audível) correspondente à sua solicitação, utilizando o teclado numérico do telefone.
- **Abertura de Chamados**: Quando o usuário solicita a abertura de um chamado, os scripts PHP enviam uma requisição para a API de chamados, incluindo informações como o código do usuário e o serviço solicitado

A biblioteca AGI facilita a comunicação entre o *FreePBX* e a API de chamados, permitindo que *scripts* PHP personalizados controlem o fluxo das chamadas, validem informações, direcionem usuários para as URAs corretas e abram chamados no sistema Web de forma automatizada.

3.4.3 Desenvolvimento API Rest sistema de chamados

Para simplificar o desenvolvimento, optou-se por centralizar os webservices na API principal do sistema de chamados. Como demonstrado na Figura 9, a API atua como um ponto de entrada, recebendo as requisições do *FreePBX* e delegando-as aos serviços apropriados, como o banco de dados (MySQL) para consultas de informações e a API do WhatsApp para interações com usuários. Adicionalmente, a API pode integrar-se com serviços da AWS.

Figura 9 - Diagrama de sequência comunicação entre o servidor FreePBX e a API de chamados



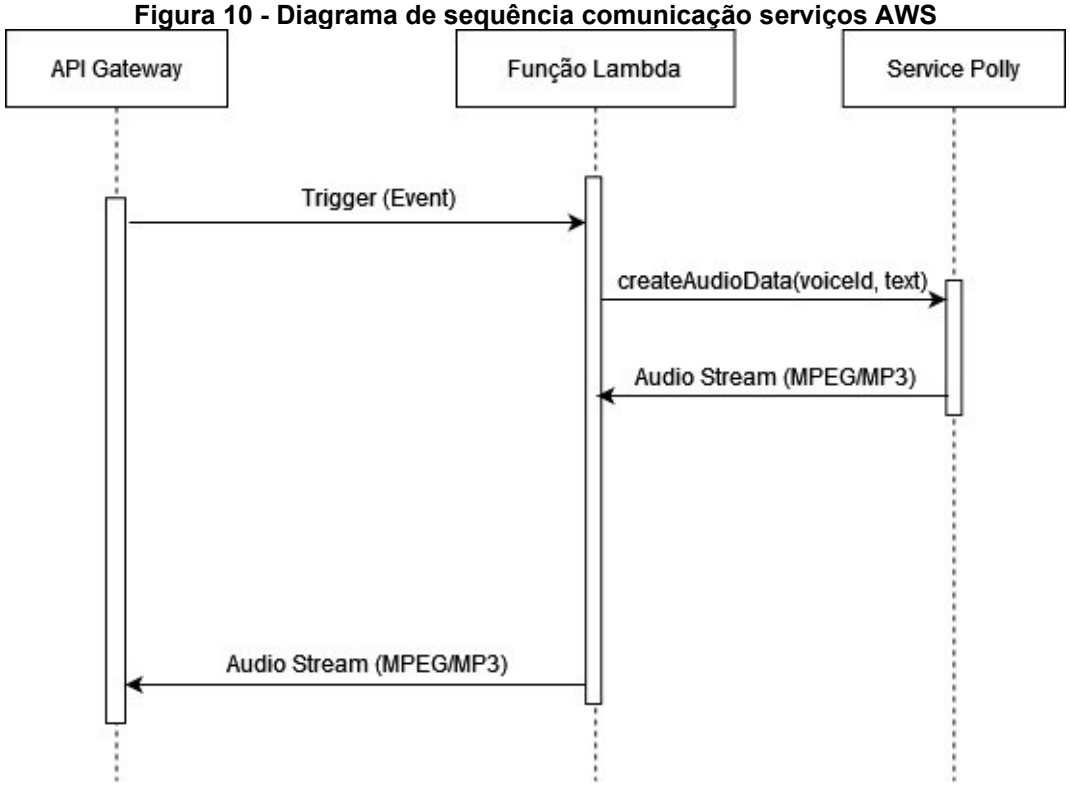
Fonte: Autoria própria (2024)

A API do sistema de chamados foi desenvolvida utilizando a linguagem PHP e lida com requisições *HTTP POST* e *GET*. Para aprimorar a segurança, implementou-se um processo de autenticação em duas etapas: primeiro, é necessário autenticar com um *token* de segurança; em seguida, o endereço IP (*host*) da máquina que realiza a requisição deve estar em uma lista de IPs permitidos. Após a autenticação bem-sucedida, é possível acessar todos os *endpoints* da API, como verificar usuário, solicitar código de acesso e validar códigos de acesso. A API retorna dados no formato JSON para facilitar o consumo por outras aplicações.

3.4.4 Serviços da AWS

Para converter os textos obtidos do banco de dados em áudio, durante o fluxo da chamada e conforme as informações solicitadas à API do sistema de chamados, é utilizado os serviços *da Amazon Web Services* (AWS). Como demonstrado na Figura 10 a arquitetura permite que aplicações externas utilizem a API *Gateway* para solicitar a conversão de texto em áudio de forma simples e eficiente, aproveitando a capacidade do serviço Polly e a flexibilidade da função Lambda.

Figura 10 - Diagrama de sequência comunicação serviços AWS



Fonte: Autoria própria (2024)

3.4.4.1 API Gateway

O API *Gateway* é o serviço responsável por atuar como ponto de entrada para APIs em uma arquitetura de microserviços, permite receber informações e se comunicar com outros serviços da AWS. O API *Gateway* foi configurado para chamar uma função Lambda, fornecendo as informações necessárias para sua execução. Então dispara uma *trigger(*evento*)* para função lambda, passando o serviço que deseja utilizar que no caso é converter o texto em áudio, e aguarda a execução da *trigger*.

3.4.4.2 Função Lambda

Após ser acionada, a função Lambda estabelece comunicação com o serviço Polly utilizando o SDK (*Software Development Kit*) da AWS.

Dentro da função Lambda, dois métodos principais são executados:

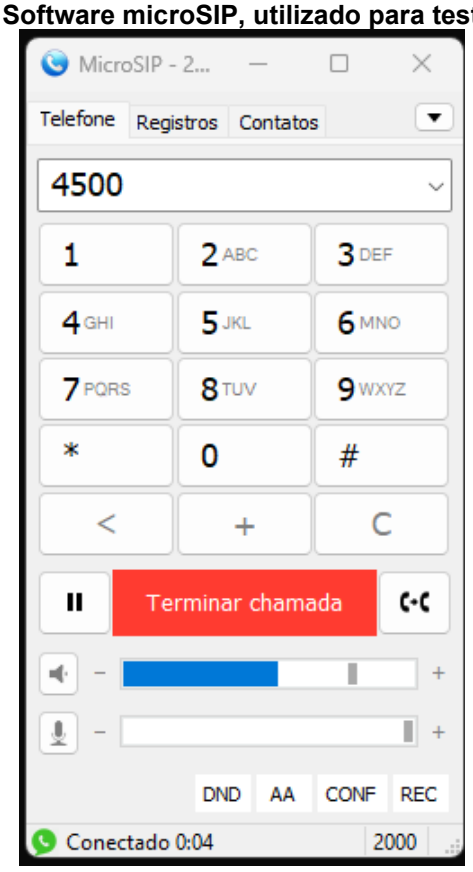
1. ***getQueryParameter***: Este método captura os parâmetros enviados pela API Gateway, como o texto a ser convertido em áudio e a voz desejada. Esses parâmetros são essenciais para a interação com o Polly.

2. ***createAudioData***: Este método assíncrono recebe duas variáveis: o identificador da voz (por exemplo, "José" para português brasileiro) e o texto a ser convertido. Com essas informações, utiliza o SDK para chamar o serviço *Polly*, solicitando a conversão do texto em áudio. Após a conversão, o áudio gerado é retornado para a API Gateway, que pode então enviá-lo ao usuário que fez a requisição original.

3.5 Testes de integração

Para a realização dos testes de integração, utilizamos o software MicroSIP como demonstrado na Figura 11. Com ele, foi verificado se as chamadas estavam sendo efetivadas sem interferências, se os menus URA estavam funcionando corretamente e os retornos das APIs estavam fornecendo as informações corretas de acordo com as expectativas e se a comunicação entre o servidor FreePBX e a API de chamados estava dentro das conformidades.

Figura 11 - Software microSIP, utilizado para testes



Fonte: Autoria própria (2024)

Para garantir a qualidade e o funcionamento correto entre o servidor VoIP e API de chamados, foi utilizado o *software* Postman para realizar testes em todos os *endpoints* como apresentado na Figura 12.

Figura 12 – Tela do *software* Postman, utilizado para teste api de chamados

Fonte: Autoria própria (2024)

Ao longo da fase testes alguns problemas foram detectados que precisavam ser corrigidos, como:

- **Problemas de volume na troca de áudio**: Em alguns momentos, o volume do áudio reproduzido durante as chamadas apresentava variações indesejadas, prejudicando a experiência do usuário.
- **Falhas na comunicação com a API**: Em determinados cenários, a comunicação entre o sistema FreePBX e a API de chamados não ocorria de forma consistente, resultando em erros ou interrupções no fluxo da chamada.

Após identificar os problemas, foi preciso trabalhar nas correções necessárias para serem realizados novos testes, até que a solução URA funcione da forma esperada.

3.6 Avaliação do uso da ferramenta

Com o objetivo de avaliar a aceitação da novidade e identificar potenciais áreas de aprimoramento, realizou-se uma pesquisa com os usuários que utilizaram a URA para registrar incidentes.

3.6.1 Metodologia da Pesquisa

A pesquisa foi conduzida com base em uma amostra aleatória de 50 incidentes registrados através da URA, selecionados a partir dos últimos 100 registros no sistema de chamados Web. Os usuários correspondentes a esses incidentes foram contatados por e-mail e convidados a responder um formulário de pesquisa.

3.6.2 Resultados da Pesquisa

A Tabela 2 apresenta o resultado da pesquisa realizada com o propósito de avaliar a recepção do novo canal de atendimento disponibilizado pelo departamento de informática. Os resultados revelam que a maioria dos usuários (56%) considerou a ferramenta "Boa", enquanto 24% a classificaram como "Muito boa".

Tabela 2 – Avaliação do novo canal de atendimento

Como você avalia sua satisfação geral com a ferramenta?	**Registros**	**%**
Não opinou	7	14
Ruim	0	0,0
Médio	3	6
Bom	28	56
Muito bom	12	24

Fonte: Autoria própria (2024)

A Tabela 3 apresenta os resultados de uma pesquisa elaborada com o objetivo de identificar quais alterações devem ser feitas futuramente para garantir a qualidade do serviço prestado pela ferramenta. Através de uma pergunta simples, buscou-se entender quais tipos de melhorias os usuários gostariam de ver implementadas. As quatro áreas mais citadas, demonstradas na Tabela 3, evidenciam os pontos com maior potencial de aprimoramento. A "Identificação de usuário" foi a área mais mencionada (30%), seguida por "Forma de atendimento" e "Serviços apresentados na URA", ambas com 14% das respostas.

Tabela 3 - Pesquisa melhorias na ferramenta

Quais áreas você acredita que a ferramenta precisa de melhorias?	**Registros**	**%**
Não opinou	21	42
Identificação de usuário	15	30
Forma de atendimento	7	14
Serviços apresentados na URA	7	14

Fonte: Autoria própria (2024)

3.6.3 Análise dos Resultados

Os resultados da pesquisa indicam uma recepção positiva da ferramenta por parte dos usuários, com a maioria expressando satisfação com sua utilização. No entanto, as sugestões de melhoria apontam áreas onde a ferramenta deve ser aprimorada para otimizar ainda mais a experiência do usuário.

3.7 Trabalhos futuros

Com objetivo de aperfeiçoar a solução apresentada. Novas funcionalidades a URA podem ser adicionadas em trabalhos futuros, como por exemplo:

- **Identificação por dispositivo:** Desenvolver a identificação automática do usuário através do ramal utilizado na ligação para o departamento de TI, agilizando o processo de atendimento.
- **Feedback em Tempo Real:** Ao realizar uma consulta sobre o status de um chamado, oferecer ao usuário se deseja participar de uma breve pesquisa de satisfação, permitindo a coleta de feedbacks para a melhoria contínua do serviço.
- **Gravação de chamadas:** Gravar as chamadas direcionadas aos atendentes, garantindo maior segurança e possibilitando a análise posterior do atendimento para fins de treinamento e controle de qualidade.
- **Acesso a informações:** Disponibilizar informações úteis na URA, como horários de atendimento, tutoriais e guias de utilização de recursos de TI.

4 CONCLUSÃO

Este projeto demonstrou que, com planejamento e criatividade, é possível integrar diferentes serviços de forma econômica para solucionar problemas que, tradicionalmente, exigiriam investimentos significativos em soluções de terceiros. Tais soluções, muitas vezes, não se adaptam às necessidades específicas de demandas altamente personalizadas, como foi o caso em questão.

Soluções de integração entre sistemas Web e telefonia (CTI) costumam ser relativamente caros, as vezes sendo necessário adquirir licenças, hardwares e suporte oferecidos pelos fabricantes. A escolha por tecnologias de código aberto como Linux, Asterisk, PHP e MySQL resultou em economia substancial em termos de licenciamento e *hardware*.

A flexibilidade do Asterisk permite ser utilizado até mesmo em ambientes de grande escala, substituindo equipamentos de grandes fabricantes sem comprometer a confiabilidade e a capacidade de expansão do sistema.

Foi optado utilizar serviços da *Amazon Web Services* (AWS) por sua natureza *serverless*, que possibilita o escalonamento automático para atender a demandas de qualquer porte, com custos extremamente reduzidos. Funções Lambda, API *Gateway* e *Polly* oferecem limites de uso gratuito, e mesmo após excedidos, os custos permanecem baixos em relação aos benefícios proporcionados. Em nosso caso, a demanda projetada se mantém dentro dos limites de gratuidade de todos os serviços utilizados.

O problema inicial, relacionado ao tempo excessivo gasto na abertura de chamados, foi solucionado de forma eficaz e ágil, resultando em uma melhoria significativa no tempo de resposta a incidentes pelo setor de TI.

As pesquisas de satisfação realizadas indicam uma aceitação gradual das melhorias implementadas, à medida que os usuários se familiarizam com a nova ferramenta. As áreas com menor aprovação receberão atenção especial, com estudos e ações para aprimorar a experiência do usuário e atender às suas expectativas.

O *feedback* coletado através da pesquisa foi crucial para identificar pontos de melhorias e direcionar o desenvolvimento para atualizações da URA. Continuaremos buscando aprimoramentos para otimizar a experiência do usuário e a eficiência do atendimento de incidentes.

Este projeto comprova a viabilidade da implementação de uma URA para o atendimento de chamados de TI, utilizando tecnologias *de código aberto* e serviços em nuvem. A solução desenvolvida automatiza o processo de registro e encaminhamento de chamados, proporcionando agilidade, eficiência e redução de custos. Acreditamos que este trabalho inspire futuras pesquisas e desenvolvimentos na área de atendimento automatizado, impulsionando a inovação e a melhoria contínua dos serviços de TI. A busca por soluções cada vez mais eficientes e personalizadas para o atendimento ao usuário é um desafio constante, e o projeto aqui apresentado é um passo significativo nessa direção.

5 REFERÊNCIAS

ASTERISK. Disponível em: <http://www.asterisk.org/>. Acesso em: 15 Jul 2013.

Rodrigues, Ada Priscila Machado Felipe; José Fabiano da Serra Costa; Monique de Menezes. 2008. “Avaliação Da Escolha de Unidade de Resposta Audível (URA) Através Do Método de Análise Hierárquica (AHP).”: 161. https://revista.feb.unesp.br/index.php/gepros/article/viewFile/474/194.

AWS. Amazon Web Services, 2023. Disponível em: https://aws.amazon.com/pt/what-is-aws/. Acesso em: 07 nov. 2023.

GABI, Caio Fernandes; LUCENA, Daiana Correia de; FRANCO, Joyce Maia; DIAS, Michel Coura. Uso do Asterik como ferramenta de auxílio no ensino prático de telefonia. In: XLI CONGRESSO BRASILEIRO DE EDUCAÇÃO EM ENGENHARIA, Gramado, RS, COBENGI 2013, Educação na Era do Conhecimento. Anais... Disponível em: https://turing.pro.br/anais/COBENGE-2013/pdf/118522_1.pdf. Acesso em: 10 nov. 2023.

IBM. O que é uma API de REST? 2023. Disponível em: https://www.ibm.com/br-pt/topics/rest-apis. Acesso em: 07 nov. 2023.

KUNTZ, Ricardo Pletsch. Central de Atendimento e Sistema de Chamados baseado no Framework ITIL. Trabalho de Conclusão de Curso (Monografia de Sistemas de Informação) – Departamento de Ciências Exatas, Universidade Regional do Noroeste do Estado do Rio Grande do Sul – UNIJUÍ, 2011. Disponível:https://bibliodigital.unijui.edu.br:8443/server/api/core/bitstreams/c45f4c27-f468-4c89-8eba-73c0200ba990/content. Acesso em: 10 nov. 2023.

LOPES, Ridson Robel. Tecnologia VOIP com Asterik. Trabalho de Conclusão de Curso (Monografia Licenciatura em Informática de Gestão) – Departamento de Engenharia e Recursos do Mar, Universidade do Mindelo, 2015. Disponível em: https://core.ac.uk/download/pdf/38683019.pdf. Acesso em: 10 nov. 2023.

MADSEN, Leif; MEGGELEN, Jim Van; BRYANT, Russel. Asterisk: the definitive guide. 4th

MEGGELEN, Jim Van; SMITH, Jared; MADSEN, Leif. Asterisk: The Future of Telephony. 2th edition. EUA: O'Reilly Media, agosto 2007. ISBN 9780596510480.

MENEZES, Adauto Cavalcante et al. Técnicas para análise de desempenho de plataformas computacionais genéricas para uso de sistemas VoIP Open Source. Aracaju: IFS, 2019. Disponível em: http://www.ifs.edu.br/images/EDIFS/ebooks/2019.2/E-Book_-_T%C3%A9cnicas_para_an%C3%A1lise_de_desempenho_em_plataformas_computacionais_gen%C3%A9ricas_para_uso_de_sistemas_VoIP_Open_Source.pdf. Acesso em: 10 nov. 2023.

NOBRE, Pedro Sávio de Oliveira. VOIPBOX: uma solução de gerenciamento VoIP para Universidade Federal do Ceará Campus Quixadá. Trabalho de Conclusão de Curso (Monografia em Redes de Computadores) – Universidade do Ceará, Quixadá, 2016. Disponível em: https://repositorio.ufc.br/bitstream/riufc/24783/1/2016_tcc_psdeonobre.pdf. Acesso em: 10 nov. 2023.

ORACLE. Manual de Referência do MySQL 8.0. Disponível em: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. Acesso em: 10 nov. 2023.

PHP. Manual do PHP, 2023. Disponível em: https://www.php.net/manual/pt_BR/intro-whatcando.php. Acesso em: 07 nov. 2023.

6 APÊNDICE A – APRESENTAÇÃO DETALHADA DAS TELAS E TRECHOS DE CÓDIGO

6.1 Scripts desenvolvidos em PHP para integração

Os scripts desenvolvidos em PHP foram essenciais para realizar a integração entre os serviços da solução proposta. As subseções a seguir apresentam trechos de códigos desenvolvidos para a concretização da solução.

6.1.1 Integração com API sistema de chamados Web

Nesta seção, serão apresentados os scripts em PHP desenvolvidos para viabilizar a integração entre a central telefônica e o sistema de chamados Web. O script principal, “serviceChamado.php”, estabelece a comunicação com a API do sistema de chamados, enquanto outros scripts complementares realizam verificações e ações específicas, como autenticação de usuário, criação de chamados e remoção de arquivos temporários de áudio. A seguir, serão detalhadas as funcionalidades e a estrutura de cada um desses scripts, oferecendo uma visão abrangente do papel crucial que desempenham na solução proposta.

1. **serviceChamado**: A Figura 13 apresenta um script PHP chamado "serviceChamado.php" que define uma classe chamada apiChamado. Essa classe é responsável por interagir com API do sistema de chamados Web, através de requisições HTTP utilizando a biblioteca cURL.

Figura 13 - Script serviceChamado.php

```php
<?php
class apiChamado
{
    protected $conexao;
    protected $hash;
    protected $url;

    function __construct()
    {
        $this->url = "https://atendesh.santahelena.pr.gov.br/api";
        $this->hash = "[redacted]";
        $this->conexao = curl_init();
        curl_setopt($this->conexao, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($this->conexao, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($this->conexao, CURLOPT_COOKIEFILE, $this->url);
        curl_setopt($this->conexao, CURLOPT_URL, $this->url . '/autenticar_sessao/' . $this->hash);
        curl_exec($this->conexao);
    }

    public function verifyUserBoolean($cod_usuario)
    {

        return $this->curl('/verificar_usuario', ['usuario' => $cod_usuario])  ? 1 : 0;
    }

    public function newCall($user, $service)
    {
        return $this->curl('/novo_chamado', ['usuario' => $user, "servico" => $service]);
    }

    public function getUsuario($user)
    {
        return $this->curl('/get_user_to_speech', ['usuario' => $user]);
    }
```

Fonte: Autoria própria (2024)

A classe possui três propriedades protegidas: $conexao, $hash e $url. O construtor da classe inicializa a URL base da API e um hash de autenticação, além de configurar uma conexão cURL para ser reutilizada nas requisições.

A classe também possui três métodos públicos:

verifyUserBoolean($cod_usuario): Verifica se um usuário existe na API com base no código fornecido e retorna 1 (verdadeiro) ou 0 (falso).

newCall($user, $service): Realiza uma chamada à API para criar um novo chamado, passando o usuário e o serviço como parâmetros.

getUsuario($user): Faz uma requisição à API para obter informações sobre um usuário, possivelmente para gerar uma mensagem de voz.

O método curl é o método auxiliar privado que encapsula a lógica de realizar as requisições HTTPs usando cURL, incluindo a configuração de cabeçalhos e o tratamento da resposta da API.

Figura 14 - Continuação script serviceChamado.php

```
30
31      public function getUsuario($user)
32      {
33          return $this->curl('/get_user_to_speech', ['usuario' => $user]);
34      }
35
36      public function verificaChamado($cod_user, $cod_chamado)
37      {
38          return $this->curl('/verificar_chamado', ['usuario' => $cod_user, 'chamado' => $cod_chamado]);
39      }
40
41      public function verificar_codigo($user, $codigo)
42      {
43          return $this->curl('/verificar_codigo', ['user' => $user, 'codigo' => $codigo]) ? 1 : 0;
44      }
45
46      public function removerAudio($name_audio)
47      {
48          return $this->curl('/remover_audio', ['audio' => $name_audio]);
49
50      }
51
52      protected function curl($path, $dados)
53      {
54          curl_setopt($this->conexao, CURLOPT_URL, $this->url . $path);
55          curl_setopt($this->conexao, CURLOPT_POSTFIELDS, json_encode($dados));
56          $temp = curl_exec($this->conexao);
57          return json_decode($temp);
58      }
59  }
60
```

Fonte: Autoria própria (2024)

2. **verifyCodigo**: O script como apresentado na Figura 15 inclui arquivos de configuração do FreePBX e importa a classe "apiChamado" e a biblioteca "phpagi". Sendo assim recebe o código do usuário e o código de acesso como argumentos de linha de comando. Em seguida, cria uma instância da classe apiChamado e chama o método verificar_codigo para validar o código do usuário e o código de acesso. O resultado da verificação é armazenado como uma variável no ambiente AGI usando $AGI->set_variable. Finalmente, a sessão da API é encerrada com $service->exitSession().

Figura 15 - Estrutura do código verifyCodigo.php

```php
#!/usr/bin/env php
<?php

include '/etc/freepbx.conf';
include "serviceChamado.php";
require_once "phpagi.php";
$AGI = new AGI();
$cod_user = $argv[1];
$codigo = $argv[2];
$service = new apiChamado();
$retorno = $service->verificar_codigo($cod_user, $codigo);
var_dump($retorno->status);
$AGI->set_variable('cod_ver', $retorno);
$service->exitSession();


?>
```

Fonte: Autoria própria (2024)

3. verifyUser: A Figura 16 demonstra o script PHP chamado "verifyUser.php" responsável por verificar a existência de um usuário. Ele inclui arquivos de configuração do FreePBX, importa a classe "apiChamado" e a biblioteca "phpagi". O script recebe o código do usuário como argumento da linha de comando e cria uma instância da classe apiChamado. Em seguida, chama o método getUsuario para obter informações sobre o usuário. A URL e o nome do áudio associados ao

usuário são extraídos do retorno da API e armazenados como variáveis no ambiente AGI. Finalmente, a sessão da API é encerrada.

Figura 16 - Estrutura do código verifyUser.php

```php
<?php

include '/etc/freepbx.conf';
include "serviceChamado.php";
require_once "phpagi.php";
$AGI = new AGI();
$cod_user=$argv[1];
$service = new apiChamado();
$retorno = $service->getUsuario($cod_user);


$audio_url = $retorno->url;
$audio_name = $retorno->nome_audio;

$AGI->set_variable('audio_url_name', $audio_name);

$AGI->set_variable('audio_name', $retorno->nome_audio);
$service->exitSession();


?>

```

Fonte: Autoria própria (2024)

4. verificarUsuarioChamado: A Figura 17 apresenta o script PHP chamado "verificarUsuarioChamado.php". Ele inclui arquivos de configuração do FreePBX, importa a classe "apiChamado" e a biblioteca "phpagi". O script recebe o código do usuário como argumento da linha de comando, cria uma instância da classe apiChamado e verifica se o usuário existe usando o método verifyUserBoolean. Se o usuário existir, busca os detalhes do usuário com o método getUsuario, define variáveis no ambiente AGI com o código do usuário, a URL de áudio e o nome do áudio associados ao usuário, e finalmente encerra a sessão da API.

Figura 17 - Estrutura do código verificarUsuarioChamado.php

```php
<?php

include '/etc/freepbx.conf';
require_once "phpagi.php";
include "serviceChamado.php";
$cod_user=$argv[1];
$AGI = new AGI();
$service = new apiChamado();
$retorno = $service->verifyUserBoolean($cod_user);
$AGI->set_variable('RETORNO', $retorno);
$usuario = $service->getUsuario($cod_user);
$AGI->set_variable('USER_TEMP',trim($cod_user));
$AGI->set_variable('AUDIO_USER', $usuario->url);
$AGI->set_variable('audio_name', $usuario->nome_audio);



?>
```

Fonte: Autoria própria (2024)

5. verificarHorario: Este *script* verifica o horário de atendimento provido pelo departamento de informática. Se estiver dentro dos horários de atendimento, direciona a chamada telefônica para a fila de atendimento. Caso contrário, exibe a mensagem que está fora do horário de atendimento e designa a chamada para abertura de chamados pela URA.

Figura 18 - Estrutura do código verificarHorario.php

```php
<?php

include '/etc/freepbx.conf';
require_once "phpagi.php";
$AGI = new AGI();
//Hora do servidor
$hora = date("H:i:s");
$AGI->verbose("Hora do servidor: ".$hora);

//horario de funcionamento
$horarios = array(
    "07:30:00",
    "11:30:00",
    "13:30:00",
    "17:30:00"
);

//verifica se a hora atual esta dentro do horario de funcionamento
if($hora >= $horarios[0] && $hora <= $horarios[1] || $hora >= $horarios[2] && $hora <= $horarios[3]){
    $AGI->verbose("Dentro do horario de funcionamento");
    //joga pra fila de atendimento ramal 9000
    $AGI->exec("Playback", "custom/aguarde_atendimento");
    $AGI->exec("Queue", "9000");

}else{
    $AGI->verbose("Fora do horario de funcionamento");
    //mensagem de horario de funcionamento
    $AGI->exec("Playback", "custom/horarios_atendimento");
    $AGI->exec("Playback", "custom/direcionar_ura_chamado");
    $AGI->exec("Wait", "2");
    $AGI->exec("Goto", "cadastrado,c,1");
}
```

Fonte: Autoria própria (2024)

6. novoChamado: Este *script* como apresentado na Figura 19 é responsável por abrir novos chamados no sistema. Ele inclui arquivos de configuração do FreePBX, importa a classe "apiChamado" e a biblioteca "phpagi". O *script* recebe o código do usuário e o código do serviço como argumentos da linha de comando. Em seguida, define esses valores como variáveis no ambiente AGI. Uma instância da classe apiChamado é criada e o método *newCall* é invocado para registrar um novo chamado, passando o código do usuário e o código do serviço. A URL e o nome do áudio associados ao chamado são armazenados como variáveis no ambiente AGI. Finalmente, a sessão da API é encerrada.

Figura 19 - Estrutura do codigoNovoChamado.php

```php
<?php

include '/etc/freepbx.conf';
include "serviceChamado.php";
require_once "phpagi.php";
$AGI = new AGI();
$AGI->verbose("++++++++++++++++++++++++++++");
$cod_user=$argv[1];
$cod_servico=$argv[2];
$AGI->set_variable('variavel1', $cod_user);
$AGI->set_variable('variavel2', $cod_servico);
$service = new apiChamado();
$retorno = $service->newCall($cod_user, $cod_servico);


$AGI->set_variable('audio_name_chamado', $retorno->url);
$AGI->set_variable('audio_name', $retorno->nome_audio);

$service->exitSession();


?>
```

Fonte: Autoria própria (2024)

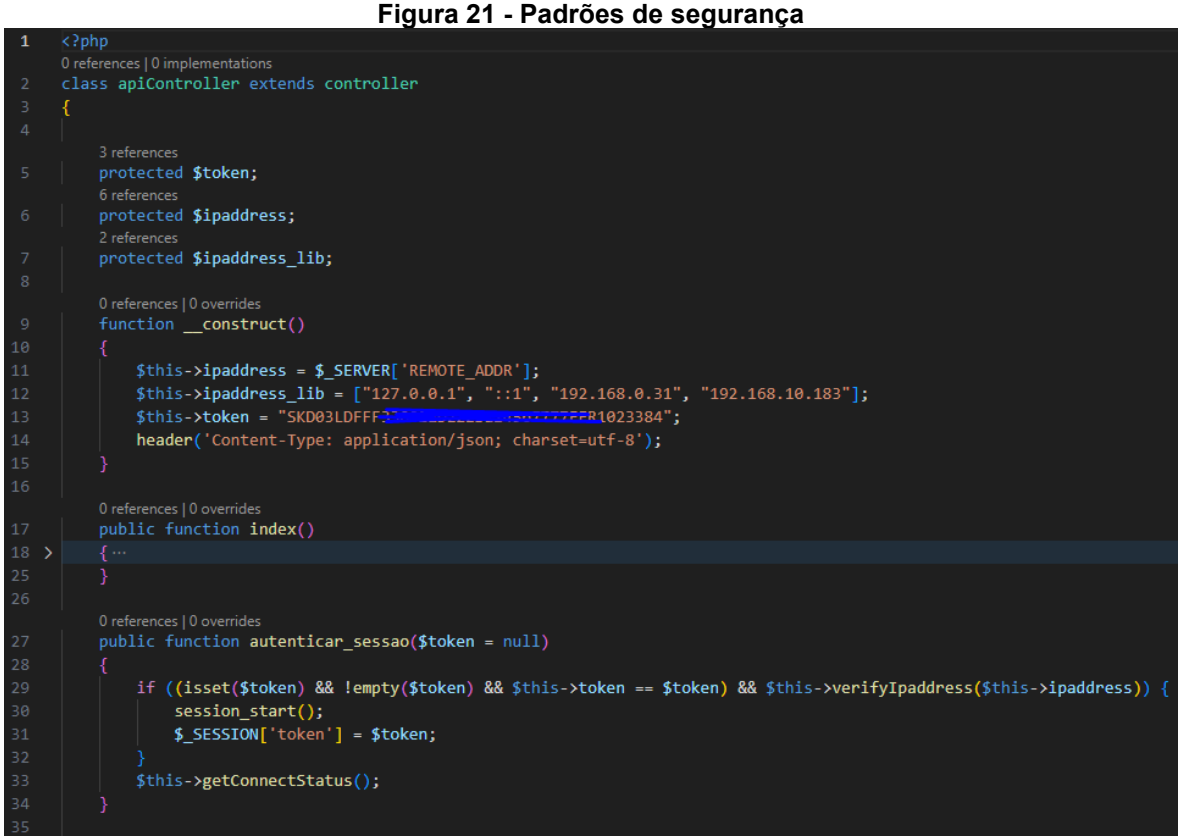
7. removerAduio: A Figura 20 demonstra o *script* PHP chamado "removerAudio.php". Ele inclui arquivos de configuração do FreePBX e utiliza a biblioteca PHPAgi e a classe apiChamado. O script recebe o nome de um arquivo de áudio como argumento e, em seguida, cria uma instância da classe apiChamado. A função removerAudio dessa classe é chamada para remover o arquivo de áudio especificado. O resultado dessa operação é exibido usando a função $AGI->verbose.

Figura 20 - Estrutura do codigo removerAudio.php

```php
<?php

include '/etc/freepbx.conf';
require_once "phpagi.php";
include "serviceChamado.php";
$AGI = new AGI();
$name_audio = $argv[1];
$service = new apiChamado();
$retorno = $service->removerAudio($name_audio);

$AGI->verbose($retorno);



?>
```

Fonte: Autoria própria (2024)

Figura 21 - Padrões de segurança



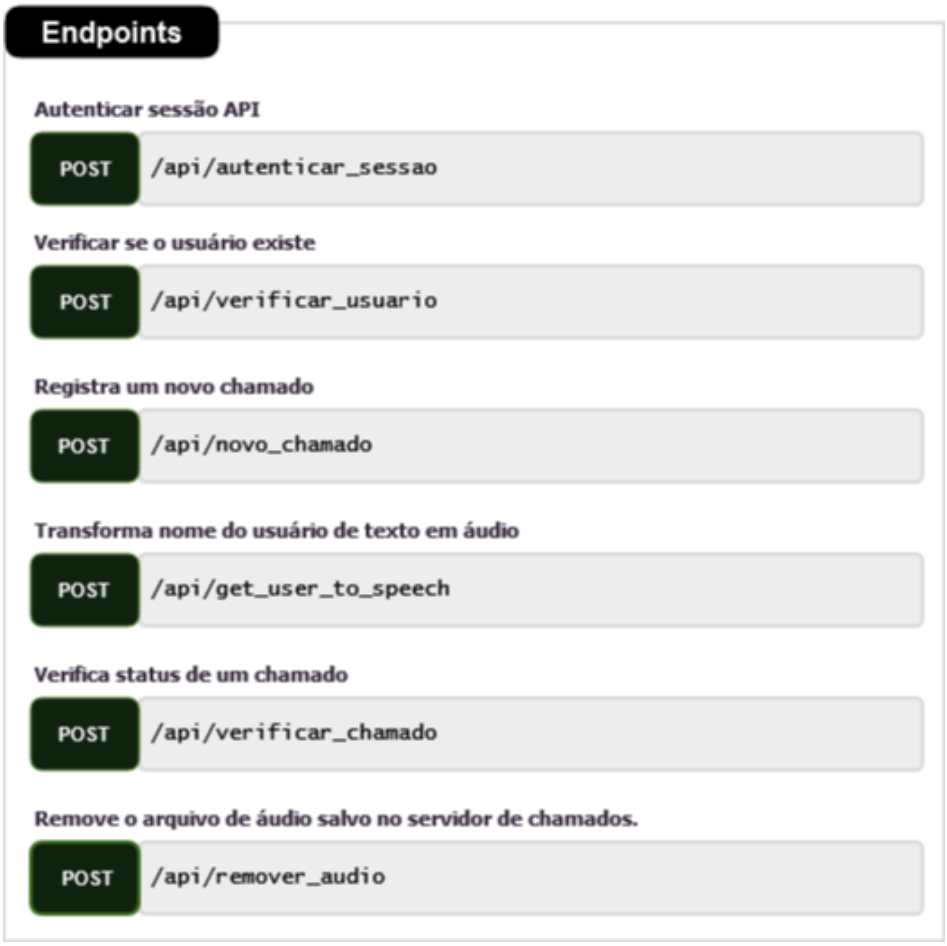
```php
<?php
0 references | 0 implementations
class apiController extends controller
{

    3 references
    protected $token;
    6 references
    protected $ipaddress;
    2 references
    protected $ipaddress_lib;

    0 references | 0 overrides
    function __construct()
    {
        $this->ipaddress = $_SERVER['REMOTE_ADDR'];
        $this->ipaddress_lib = ["127.0.0.1", "::1", "192.168.0.31", "192.168.10.183"];
        $this->token = "SKD03LDFFF[redacted]1023384";
        header('Content-Type: application/json; charset=utf-8');
    }

    0 references | 0 overrides
    public function index()
    {...
    }

    0 references | 0 overrides
    public function autenticar_sessao($token = null)
    {
        if ((isset($token) && !empty($token) && $this->token == $token) && $this->verifyIpaddress($this->ipaddress)) {
            session_start();
            $_SESSION['token'] = $token;
        }
        $this->getConnectStatus();
    }

```

Fonte: Autoria própria (2024)

6.1.2 API Sistema de chamados Web

Como apresentado na Figura 22 apresenta uma lista de *endpoints* da API do sistema de chamados Web. As funcionalidades da API incluem: autenticar a sessão da API, verificar se um usuário existe, registrar um novo chamado, transformar o nome do usuário de texto em áudio, verificar o status de um chamado e remover um arquivo de áudio salvo no servidor de chamados.

Figura 22 - Endpoints da API



Fonte: Autoria própria (2024)

Figura 23 - Resposta de uma requisição da API

Fonte: Autoria própria (2024)

Figura 24 - Endpoints da API Rest de chamados

```php
0 references | 0 overrides
public function verificar_usuario()
{
    $this->protectSession();
    $temp = json_decode(file_get_contents('php://input'));
    $usuario = (isset($temp->usuario) && !empty($temp->usuario)) ? $temp->usuario : "null";
    if (!is_null($usuario)) {
        $_SESSION['tempUser'] = $usuario;
        $result = $this->ApiService->getUsuario($usuario);
        echo json_encode($result['status']);
    }
}

0 references | 0 overrides
public function novo_chamado()
{
    $this->protectSession();
    $temp = json_decode(file_get_contents('php://input'));
    $menuOpcao = $this->ApiService->splitDados($temp->servico);
    $result = $this->ApiService->createChamado($temp->usuario, $menuOpcao[1]);
    echo $result;
}

0 references | 0 overrides
public function verificar_chamado()
{
    $this->protectSession();
    $temp = json_decode(file_get_contents('php://input'));
    $result = $this->ApiService->getChamado($temp->chamado, $temp->usuario);
    echo $result;
}

0 references | 0 overrides
public function verificar_codigo()
{
    $this->protectSession();
    $temp = json_decode(file_get_contents('php://input'));
    $result = $this->ApiService->verificar_codigo($temp->user, $temp->codigo);
    echo json_encode($result['status']);
```

Fonte: Autoria própria (2024)

6.2 Serviços da AWS

Esta seção explora a utilização dos serviços da Amazon Web *Services* (AWS) para garantir a funcionalidade da solução. Especificamente, a configuração da função Lambda será detalhada, demonstrando como ela interage com outros serviços da AWS para fornecer recursos cruciais à aplicação. Serão abordados o método *getQueryParameter*, responsável por capturar parâmetros de entrada, e o método *createAudioData*, que utiliza o serviço *Polly* para converter texto em áudio, um recurso essencial para a interação com o usuário através da URA.

6.2.1 Comunicação com os serviços da AWS

Este método desempenha um papel fundamental na comunicação com os serviços da AWS (*Amazon Web Services*). Para sua execução, foi criado um método protegido que recebe o texto a ser convertido como parâmetro. A biblioteca cURL é utilizada para fazer requisições à AWS, passando o texto no corpo da URL do *API Gateway*. Após receber a resposta da requisição, o áudio é armazenado em um diretório específico no servidor da API de chamados. O método então retorna a URL do diretório para uso posterior, podendo ser removido quando não for mais necessário.

Figura 25 - Chamada para API Gateway (AWS)

```
4 references | 0 overrides
protected function text_to_scpeech($text)
{
    $url = "https://[redacted].execute-api.us-east-1.amazonaws.com/default/polly2?voice=Ricardo&text=" . urlencode($text);
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    $audio = curl_exec($ch);
    curl_close($ch);
    $number = rand(1, 9999);
    $path = "views/audios/";
    $name_audio = "voice_" . $number . '_' . date("Y-m-d_H-i-s") . ".mp3";
    $file = fopen($path . $name_audio, "w+");
    fputs($file, $audio);
    fclose($file);
    //gera link donwload
    $link = "https://" . $_SERVER['HTTP_HOST'] . "/" . $path . $name_audio;

    return json_encode(array("url" => $link, "nome_audio" => $name_audio));
}
```

Fonte: Autoria própria (2024)

6.2.2 Configuração da função Lambda

A Figura 26 apresenta o trecho de código *JavaScript* que define a função chamada *getQueryParameter*. Essa função recebe um evento e uma chave como parâmetros e tenta extrair o valor associado a essa chave dos parâmetros de *string* de consulta do evento. Se o valor não for encontrado, a função lança um erro. Caso contrário, ela retorna o valor encontrado.

Figura 26- Método getQueryParameter

```
const AWS = require("aws-sdk");


function getQueryParameter(event, key) {

    const value = event["queryStringParameters"] && event["queryStringParameters"][key];

    if (!value) {
        throw new Error("Could not get the query parameter \"" + key + "\".");
    }

    return value;

}
```

Fonte: Autoria própria (2024)

Figura 27 - Método createAudioData

```
async function createAudioData(voiceID, text) {
    return new Promise((resolve, reject) => {

        const pollyParams = {
            OutputFormat: "mp3",
            Text: text,
            VoiceId: voiceID,
        };

        let polly = new AWS.Polly();
        polly.synthesizeSpeech(pollyParams, function(error, data) {

            if (error) {
                reject(error);
                return;
            }

            let audioStream = data.AudioStream;

            resolve(audioStream);

        });
    });
}
```

Fonte: Autoria própria (2024)

A Figura 27 mostra a função JavaScript assíncrona chamada *createAudioData*, que recebe um ID de voz (*voiceID*) e um texto (*text*) como entrada então chama o serviço *Polly* da AWS (Amazon Web Services) para sintetizar o texto em áudio no formato MP3.

Primeiramente, a cria uma nova promessa (*Promise*) que será resolvida com o fluxo de áudio gerado ou rejeitada em caso de erro. Em seguida define os

parâmetros para o *Polly*, incluindo o formato de saída (MP3), o texto a ser convertido e o ID da voz.

Uma nova instância do serviço *Polly* é criada e o método *synthesizeSpeech* é chamado, passando os parâmetros e uma função de *callback* para lidar com o resultado. Se ocorrer um erro durante a síntese, a promessa é rejeitada com o erro. Caso contrário, o fluxo de áudio (audioStream) é extraído dos dados retornados pelo *Polly* e a promessa é resolvida com esse fluxo de áudio.

Figura 28 - Manipulador de eventos para conversão de texto em áudio

```
42  exports.handler = async (event) => {
43      try {
44
45          const qpVoiceID = getQueryParameter(event, "voice");
46          const qpText = getQueryParameter(event, "text");
47
48          const audioData = await createAudioData(qpVoiceID, qpText);
49
50          const response = {
51              statusCode: 200,
52              headers: {
53                  "content-type": "audio/mpeg",
54              },
55              body: audioData.toString("base64"),
56              isBase64Encoded: true,
57          };
58          return response;
59      } catch (error) {
60
61          console.error("error", error);
62
63          const response = {
64              statusCode: 500,
65              body: error.toString(),
66          };
67          return response;
68
69      }
70  };
```

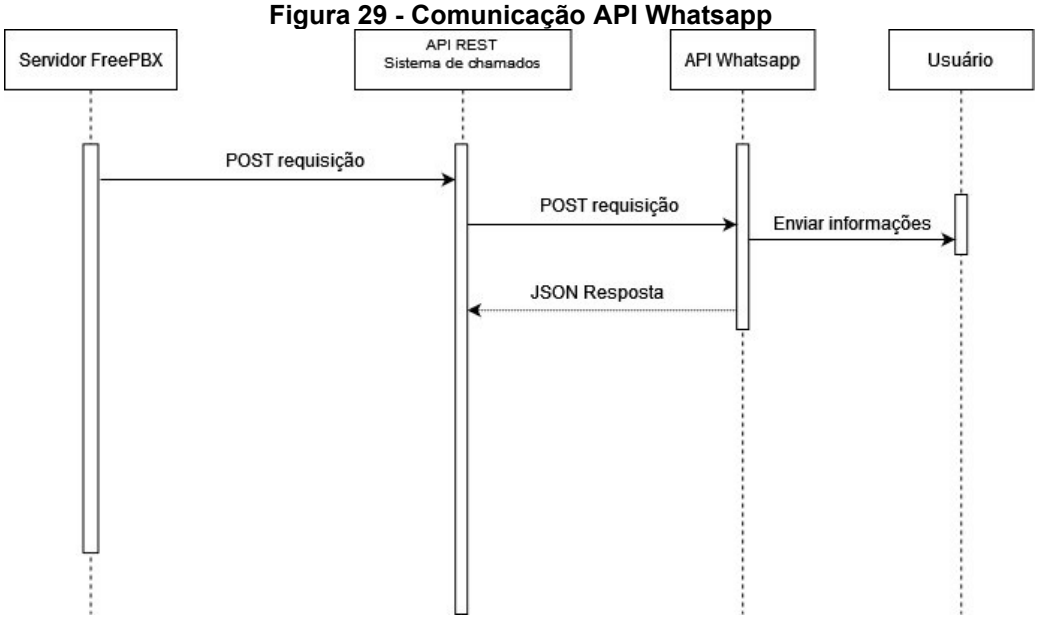
Fonte: Autoria própria (2024)

O trecho de código em JavaScript apresentado na Figura 28, define um manipulador de eventos assíncrono (*exports.handler*). Ele tenta obter parâmetros de consulta chamados "*voice*" e "*text*" de um evento usando a função *getQueryParameter*. Em seguida, ele chama uma função assíncrona *createAudioData* com esses parâmetros para gerar dados de áudio. Se bem-sucedido, constrói uma resposta HTTP com *status* 200, o tipo de conteúdo "*audio*/mpeg" e o áudio codificado em base64 no corpo da resposta. Se houver um erro, registra o erro no console e retorna uma resposta HTTP com status 500 e a mensagem de erro no corpo da resposta.

6.3 API WhatsApp

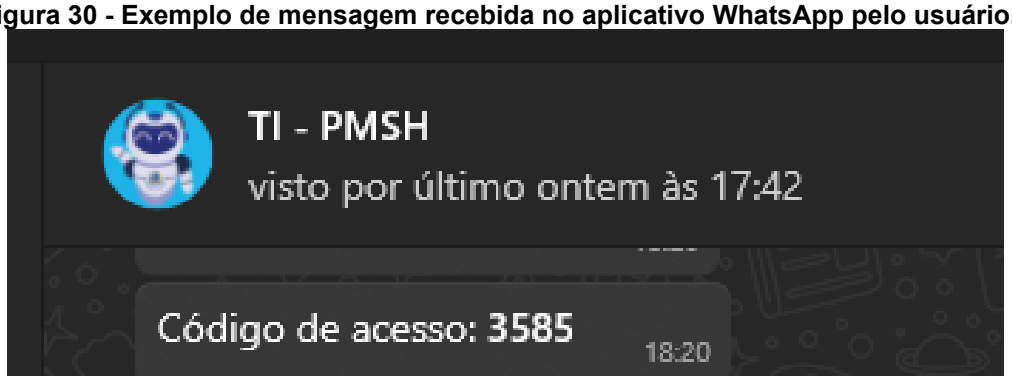
Para desenvolver a API responsável pelo envio de mensagens via WhatsApp aos usuários, optou-se pela linguagem Node.js, devido à disponibilidade de bibliotecas que facilitam essa funcionalidade. A API foi construída utilizando o framework *Express*, que permitiu a definição das rotas da API, e a biblioteca Venom-bot, responsável pelo envio das mensagens para o WhatsApp. As próximas subseções detalharão o desenvolvimento dessas ferramentas.

Figura 29 - Comunicação API Whatsapp



Fonte: Autoria própria (2024)

Figura 30 - Exemplo de mensagem recebida no aplicativo WhatsApp pelo usuário.



Fonte: Autoria própria (2024)

6.3.1 Comunicação com API WhatsApp

Foi criado um método protegido para a comunicação com o serviço de mensagens. Para estabelecer a comunicação, o cabeçalho da requisição é definido com o *token* de acesso. Em seguida, no corpo da requisição, os valores são especificados no formato JSON. Dessa forma, quando o método é chamado e seus atributos são atendidos, a mensagem é enviada para o WhatsApp do usuário.

Figura 31 - Comunicação com API de mensagem

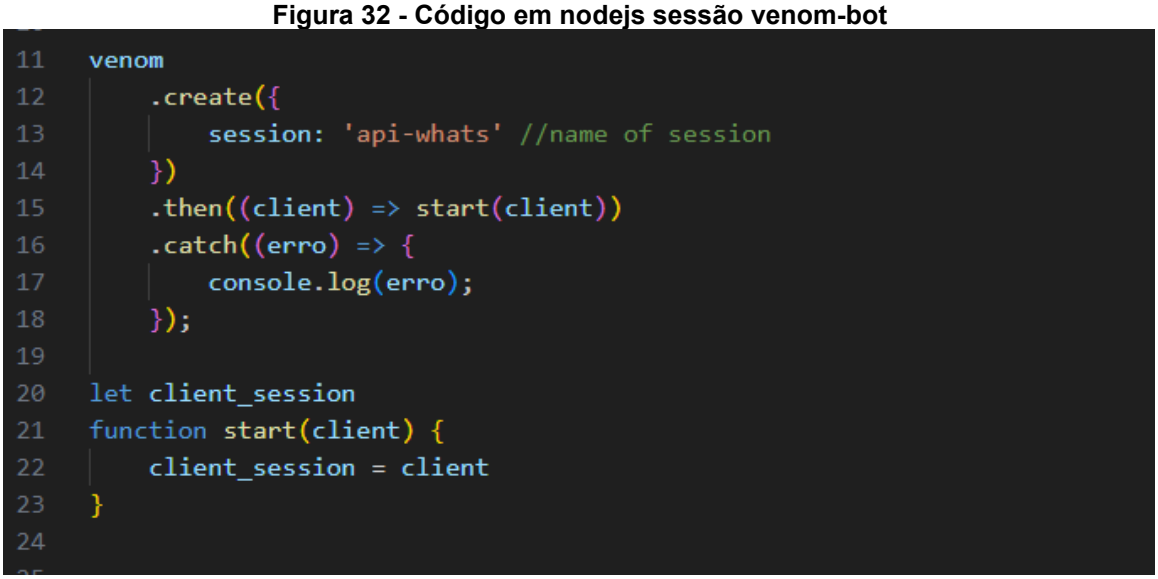
```
1 reference | 0 overrides
protected function send_message_whatsapp($message, $phone)
{
    $Authorization = "[redacted]9832KSKDOF903DDAS";
    $url_request = "http://zap.santahelena.pr.gov.br:3000/api/send-message";
    $headr = array();
    $headr[] = 'Content-type: application/json';
    $headr[] = 'Authorization: ' . $Authorization;
    $ch = curl_init($url_request);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_HTTPHEADER, $headr);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt(
        $ch,
        CURLOPT_POSTFIELDS,
        json_encode([
            "message" => $message,
            "phone" => $phone
        ])
    );
    curl_exec($ch);
    curl_close($ch);
}
```

Fonte: Autoria própria (2024)

6.3.2 Configuração biblioteca venom-bot

A Figura 32 demonstra um trecho de código em Node.js que utiliza a biblioteca Venom para gerenciar sessões de um bot do WhatsApp. O código inicia criando uma sessão com o nome "api-whats" e, em caso de sucesso, chama a função *start*, que armazena o cliente da sessão em uma variável global client_session. Se ocorrer algum erro durante a criação da sessão, ele é registrado no console.

Figura 32 - Código em nodejs sessão venom-bot



```
venom
    .create({
        session: 'api-whats' //name of session
    })
    .then((client) => start(client))
    .catch((erro) => {
        console.log(erro);
    });

let client_session
function start(client) {
    client_session = client
}
```

Fonte: Autoria própria (2024)

6.3.3 Rotas utilizando Express

A Figura 33 demonstra um código em Node.js que define as rotas principais da aplicação usando o framework Express. A primeira rota, acessível via GET em '/api/auth/status', verifica se o usuário está logado em uma sessão do WhatsApp e retorna o *status* da autenticação. A segunda rota, acessível via POST em '/api/send-message', permite o envio de mensagens pelo WhatsApp após verificar a autorização da requisição e se o usuário está logado na sessão. Em caso de sucesso, a mensagem é enviada utilizando a biblioteca 'venom-bot' e uma confirmação é retornada. Caso contrário, mensagens de erro apropriadas são exibidas. Por fim, o código inicia o servidor na porta especificada e exibe uma mensagem no console.

Figura 33 - Código nodejs rotas com express

```
app.get('/api/auth/status', async (req, res) => {
    try {
        const loggedIn = await client_session.isLoggedIn();
        res.json({ loggedIn });
    } catch (error) {
        res.status(500).json({ error: 'Erro ao verificar o status da autenticação.' });
    }
});

// Rota para enviar mensagens via WhatsApp
app.post('/api/send-message', async (req, res) => {
    try {
        const { phone, message } = req.body;
        const authorization = req.headers.authorization;

        if (authorization !== config.authorization) {
            throw new Error('Unauthorized');
        }
        if (!client_session || !(await client_session.isLoggedIn())) {
            throw new Error('Você precisa estar autenticado para enviar mensagens.');
        }

        await client_session.sendText(phoneNumberFormatter(phone), message);

        res.status(200).json({ message: 'Mensagem enviada com sucesso.' });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

app.listen(port, () => {
    console.log(`Servidor rodando na porta ${port}`);
});
```

Fonte: Autoria própria (2024)