# Relatório Técnico e Documentação de Arquitetura de Software

Projeto: RPA-Envio-Emails-STREAMLIT

Organização: DGCA-Electra

Data: 09 de Dezembro de 2025

Autor: Malik Ribeiro Mourad

# 1. Resumo Executivo

## 1.1 Visão Geral do Projeto

Este documento apresenta a documentação técnica e a análise arquitetural do projeto "RPA-Envio-Emails-STREAMLIT". Este projeto é uma solução de Automação Robótica de Processos (RPA) desenvolvida para otimizar o envio de e-mails corporativos. A aplicação utiliza a linguagem Python, o framework Streamlit para criar uma interface web amigável e a biblioteca Pandas para manipular grandes volumes de dados.

A principal função do sistema é transformar o processo manual e repetitivo de envio de e-mails individualizados (baseados em planilhas) em um fluxo de trabalho automatizado, seguro e rastreável.

## 1.2 Objetivos Estratégicos e Escopo

O escopo desta documentação abrange a arquitetura, o fluxo de dados e as decisões de design do software. Os objetivos principais do sistema são:

1. Eficiência Operacional: Reduzir o tempo gasto no envio manual de comunicações padronizadas.
2. Integridade de Dados: Eliminar erros humanos através do processamento automatizado de dados estruturados.
3. Segurança da Informação: Utilizar protocolos de autenticação modernos (OAuth 2.0) e canais de comunicação seguros (Microsoft Graph API).

## 2. Introdução e Fundamentação Teórica

### 2.1 O Paradigma da Automação de Processos (RPA)

O projeto "RPA-Envio-Emails-STREAMLIT" se enquadra na categoria moderna de RPA, onde a automação ocorre na camada de aplicação e dados, e não na interface gráfica (GUI). A escolha do Python garante um código legível e aproveita um vasto ecossistema de bibliotecas. A arquitetura substitui scripts de linha de comando por uma aplicação web interativa, separando a "lógica de negócio" (regras de envio) da "lógica de execução" (código Python).

### 2.2 A Escolha do Framework Streamlit

O Streamlit foi escolhido como frontend (interface do usuário) por permitir a criação de aplicações de dados complexas usando apenas Python. Isso acelera o desenvolvimento, permitindo que o foco permaneça na lógica de automação. O Streamlit utiliza um Modelo de Execução Baseado em Script, onde o código é reexecutado a cada interação do usuário, o que exige um gerenciamento cuidadoso do estado da sessão e do cache de dados.

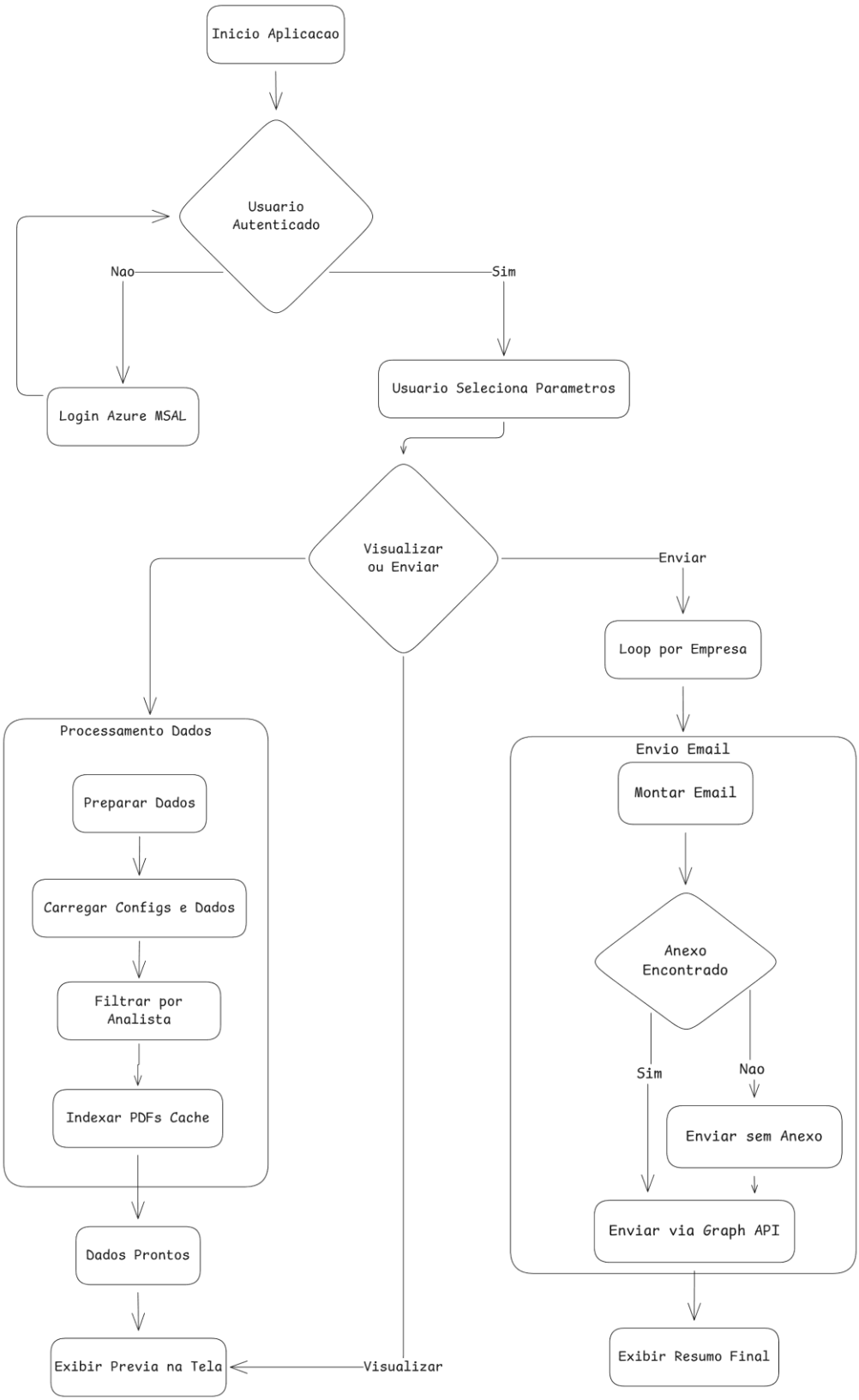
## 3. Arquitetura de Sistemas e Design Tecnológico

### 3.1 Diagrama de Componentes

A arquitetura do sistema é dividida em camadas, garantindo que cada componente tenha uma função isolada e clara.

| Camada | Tecnologia | Responsabilidade Principal |
|---|---|---|
| Apresentação (Frontend) | Streamlit (Python) | Interface do usuário, captura de comandos e visualização de logs. |
| Lógica de Aplicação (Middleware) | Python 3.9+ | Orquestração do fluxo, controle de loops e tratamento de exceções. |
| Processamento de Dados | Pandas / OpenPyXL | Leitura, sanitização, filtragem e normalização de dados. |
| Comunicação (Rede) | Microsoft Graph API (REST) | Envio seguro de e-mails via chamada de API HTTP. |
| Persistência e Configuração | TOML / .env | Gestão de segredos, variáveis de ambiente e configurações. |

### 3.2 Fluxograma do processo



### 3.3 Fluxo de Execução Lógica

O fluxo de trabalho segue as seguintes etapas:

4 Inicialização e Login: O sistema inicia e exige que o usuário se autentique via OAuth 2.0 (Microsoft).

5 Seleção de Parâmetros: O usuário seleciona o Tipo de Relatório, Mês e Ano.

6 Ingestão Automática de Dados: O sistema acessa automaticamente os arquivos de dados (planilhas) nos caminhos de rede mapeados (SharePoint) com base nos parâmetros selecionados.

7 Validação de Critérios: Antes do envio, o sistema filtra e valida os registros (e.g., e-mails malformados, clientes inativos).

8 Loop de Transmissão: O sistema itera sobre cada registro validado, renderiza o e-mail e envia a requisição para a Microsoft Graph API.

9 Logging e Encerramento: O status de cada operação é registrado e consolidado em um relatório final para o usuário.

## 4. Engenharia de Dados e Módulo de Ingestão

### 4.1 Mecanismos de Leitura de Dados

O módulo de ingestão de dados não depende do upload manual de arquivos. Em vez disso, o sistema foi projetado para acessar diretórios mapeados na rede ou pastas sincronizadas do SharePoint de forma automática.

O código (src/config/config_manager.py e src/view/main_page.py) utiliza os parâmetros de Mês e Ano selecionados pelo usuário para construir o caminho exato onde o arquivo de dados (planilha) deve estar localizado. A função resolve_best_paths garante que o sistema tente encontrar o caminho de rede correto para o usuário logado.

Essa abordagem garante a padronização da origem dos dados, reduzindo o erro humano de selecionar o arquivo errado a cada execução. A biblioteca Pandas é então utilizada para ler o arquivo diretamente do caminho de rede.

### 4.2 Sanitização e Validação de Dados

A integridade da automação é mantida por uma camada de sanitização rigorosa:

10 Normalização de Cabeçalhos: Padroniza os nomes das colunas para evitar erros de chave.

11 Tratamento de Valores Nulos: Descarta proativamente registros com campos críticos vazios (como o endereço de e-mail).

12 Limpeza de Strings: Remove espaços em branco invisíveis para garantir que os endereços de e-mail estejam corretos.

## 5. O Motor de Automação e Lógica de Envio

### 5.1 Orquestração do Fluxo de Trabalho

O motor de automação gerencia o estado da conexão, o progresso visual e o tratamento de erros. Para cada registro (linha da planilha), o sistema realiza:

13 Contextualização: Extrai variáveis da linha (Nome, Empresa, etc.).

14 Renderização: Insere essas variáveis no template de e-mail selecionado.

15 Transmissão: Envia o pacote de dados para a Microsoft Graph API.

16 Auditoria: Registra o resultado da operação.

### 5.2 Protocolo de Envio

O sistema não utiliza SMTP. O envio de e-mails é realizado através de uma chamada REST para a Microsoft Graph API (https://graph.microsoft.com/v1.0/me/messages).

| Característica | Detalhe da Implementação (Graph API) |
|---|---|
| Protocolo | HTTP/REST |
| Mecanismo | O sistema envia uma requisição POST para a API. |
| Resultado | A API cria um rascunho de e-mail na caixa de entrada do usuário logado no Exchange Online. |
| Vantagem | Maior segurança e compatibilidade com ambientes Microsoft 365, sem a necessidade de gerenciar portas SMTP ou servidores de correio. |

### 5.3 Formato do E-mail

Para construir a mensagem, o sistema não utiliza objetos MIME (MIMEMultipart, MIMEText). Em vez disso, ele constrói um Payload JSON que é o formato exigido pela Graph API.

17 Corpo do E-mail: O corpo é enviado como uma string HTML dentro do JSON.

18 Anexos: Os arquivos de anexo são lidos, codificados em Base64 (base64.b64encode) e incluídos como um array de objetos no Payload JSON.

## 6. Design de Interface e Experiência do Usuário (UI/UX)

A interface do Streamlit guia o operador de forma linear:

- Barra Lateral (Sidebar): Usada para configurações globais.
- Painel Principal: Fluxo operacional (Seleção de Parâmetros -> Visualização de Dados -> Ação de Envio).

O sistema oferece feedback visual em tempo real, como barras de progresso (st.progress) e tabelas de status final, para que o usuário saiba se o robô está trabalhando ou se a operação foi concluída.

## 7. Protocolos de Comunicação e Segurança de Rede

### 7.1 Protocolo de Envio e Segurança de Transporte

A arquitetura de comunicação do sistema abandona protocolos de e-mail legados (como SMTP na porta 587) em favor de uma abordagem moderna baseada em **API RESTful sobre HTTPS**, garantindo segurança por design em todas as transações de dados.

- **Criptografia em Trânsito (TLS 1.2+):** Todas as interações com a Microsoft Graph API são realizadas exclusivamente através do protocolo **HTTPS** (https://graph.microsoft.com), forçando o uso de **Transport Layer Security (TLS) versão 1.2 ou superior**. Isso assegura criptografia ponta a ponta para o payload da mensagem (corpo do e-mail e anexos em Base64) e protege o Token de Acesso contra interceptação (ataques *Man-in-the-Middle*).
- **Segurança no Cabeçalho HTTP:** Conforme implementado no módulo de serviços (src/services.py), a autenticação na camada de transporte não trafega credenciais de usuário. A autorização é realizada através do cabeçalho padrão HTTP Authorization, utilizando o esquema **Bearer Token**:

Isso garante que, mesmo em nível de log de rede, as credenciais originais do usuário (senha) nunca sejam expostas, trafegando apenas um token temporário e revogável.

- **Validação de Certificados:** A utilização da biblioteca requests do Python para as chamadas de API inclui, nativamente, a validação da cadeia de certificados SSL do servidor da Microsoft.

### 7.2 Autenticação e Gestão de Credenciais

A segurança das credenciais é o ponto mais crítico e é tratada com um protocolo moderno: Autenticação Delegada OAuth 2.0.

| Característica | Detalhe da Implementação (OAuth 2.0) |
|---|---|
| Mecanismo | OAuth 2.0 (Fluxo de Código de Autorização), gerenciado pela biblioteca MSAL (msal.ConfidentialClientApplication). |
| Credenciais | O sistema usa Client ID, Tenant ID e Client Secret (do Azure App Registration) apenas para identificar a aplicação. |
| Login do Usuário | O usuário faz login interativo ("Sign in with Microsoft"), gerando um Access Token. |
| Permissão | A permissão de envio é derivada do Access Token do usuário logado (Autenticação Delegada), usando os escopos User.Read e Mail.ReadWrite. |
| Segurança | Não há senhas de e-mail salvas no código ou em arquivos de segredo, eliminando o risco de senhas de aplicativo ou autenticação básica. |

Este modelo garante que a aplicação só possa agir em nome do usuário enquanto o token for válido, seguindo as melhores práticas de segurança da Microsoft.

## 8. Conclusão

A arquitetura do projeto "RPA-Envio-Emails-STREAMLIT" foi desenvolvido para eliminar o trabalho manual de atualizar planilhas excel, utilizando a linguagem de programação Python, baseada na integração segura com a plataforma Microsoft 365.