Sistema RPA - Consulta Automatizada de Países

Documentação Técnica e Pipeline de Execução

Instituição:	Faculdade Impacta de Tecnologia
Departamento:	Departamento de Tecnologia da Informação
Disciplina:	Robotic Process Automation (RPA)
Ano Letivo:	2025
Versão:	1.0
Data de Criação:	18 de November de 2025

Autores:	Felipe Viana, Ryan Rodrigues

Tipo de Documento:	Documentação Técnica
Tecnologias:	Python 3.11, SQLite, REST API
Categoria:	Robotic Process Automation

Resumo Executivo

Este documento apresenta a documentação técnica completa do sistema de Robotic Process Automation (RPA) desenvolvido para automatização de consulta e armazenamento de informações de países. O sistema integra a API REST Countries com processamento de dados em Python e persistência em banco SQLite, oferecendo uma solução robusta para coleta e organização automatizada de dados geográficos. A documentação inclui análise detalhada de cada componente do sistema, pipeline de execução visual, especificações técnicas e diagramas de arquitetura, proporcionando uma visão abrangente do funcionamento do sistema para fins educacionais, manutenção e evolução futura.

1. Visão Geral do Sistema

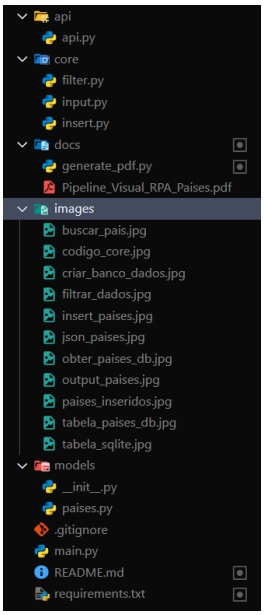
1.1 Objetivo

O sistema RPA tem como objetivo automatizar o processo de consulta, processamento e armazenamento de informações de países através da integração com a API REST Countries. O sistema elimina a necessidade de busca manual de dados geográficos, implementando um fluxo automatizado que garante consistência, precisão e eficiência na coleta de informações.

1.2 Funcionalidades Principais

- Coleta automatizada de nomes de países através de interface de linha de comando
- Busca inteligente com fallback utilizando múltiplos endpoints da API REST Countries
- Processamento e filtragem de dados JSON com correspondência exata
- Validação de duplicatas antes da inserção no banco de dados
- Armazenamento estruturado em banco SQLite com 14 campos de informação
- Suporte multilíngue para nomes de países em português e inglês
- Tratamento robusto de erros e exceções
- Feedback visual do status das operações

1.3 Estrutura do Projeto



O projeto segue uma arquitetura modular organizada em camadas funcionais distintas. A estrutura permite separação clara de responsabilidades, facilitando manutenção e evolução do sistema.

1.4 Arquitetura do Sistema

O sistema adota uma arquitetura em camadas com separação clara de responsabilidades:

- Camada de Apresentação: Interface de linha de comando para interação com usuário
- Camada de Negócio: Lógica de processamento, filtragem e validação de dados
- Camada de Integração: Comunicação com API externa REST Countries
- Camada de Persistência: Armazenamento e recuperação de dados em SQLite
- Camada de Modelo: Definição da estrutura de dados e esquema do banco

2. Pipeline de Execução Técnico

2.1 Etapa 1: Inicialização do Sistema

```python
from models import paises
from core import input, insert, filter

def main():
    paises_lista = input.obter_paises()

    for pais in paises_lista:
        pais_data = filter.filtrar_dados(pais)
        if pais_data:
            insert.insert_pais(pais_data, pais)

    insert.fechar_conexao()

if __name__ == '__main__':
    main()
```

O sistema é inicializado através do módulo principal main.py, que coordena a importação e execução de todos os componentes necessários. Nesta etapa, são carregados os módulos de conexão com banco de dados, coleta de input, processamento de dados e inserção. Componentes carregados: • models.paises: Estabelece conexão com banco SQLite e define estrutura da tabela • core.input: Responsável pela coleta de dados do usuário • core.filter: Implementa lógica de processamento e filtragem de dados da API • core.insert: Gerencia inserção de dados e controle de duplicatas A função main() atua como orquestrador central, coordenando o fluxo de execução entre todos os módulos do sistema.

Nota Técnica: Utiliza padrão de importação modular para separação de responsabilidades

2.2 Etapa 2: Configuração do Banco de Dados

```python
import sqlite3
import os

# Garante que o diretório data existe
if not os.path.exists('data'):
    os.makedirs('data')

db = sqlite3.connect('data/paises.db')
cursor = db.cursor()
```

O sistema configura automaticamente o ambiente de persistência criando o banco de dados SQLite na pasta data/ caso não exista. A conexão é estabelecida e mantida durante toda a execução para garantir integridade transacional. Configurações realizadas: • Criação automática do diretório data/ se não existir • Estabelecimento de conexão SQLite com o arquivo paises.db • Configuração de timeout e parâmetros de conexão • Preparação do cursor para execução de comandos SQL O SQLite é utilizado por ser um banco embarcado que não requer instalação ou configuração de servidor, ideal para aplicações RPA de pequeno e médio porte.

Nota Técnica: SQLite oferece ACID compliance e suporte completo a SQL padrão

2.3 Etapa 3: Definição do Esquema de Dados

```python
from . import db, cursor

cursor.execute('''
    CREATE TABLE IF NOT EXISTS paises(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
                nome_comum TEXT,
                nome_oficial TEXT,
                capital TEXT,
                continente TEXT,
                regiao TEXT,
                subregiao TEXT,
                populacao INTEGER,
                area REAL,
                moeda_nome TEXT,
                moeda_simbolo TEXT,
                idioma_principal TEXT,
                fuso_horario TEXT,
                bandeira_url TEXT)
''')

db.commit()
```

A tabela 'paises' é criada seguindo um esquema normalizado com 14 campos que capturam informações abrangentes sobre cada país. O esquema foi projetado para acomodar todos os dados relevantes fornecidos pela API REST Countries. Estrutura da tabela: • Identificadores: id (chave primária), nome_comum, nome_oficial • Localização: capital, continente, regiao, subregiao • Demografia: populacao (INTEGER) • Geografia: area (REAL) • Economia: moeda_nome, moeda_simbolo • Cultura: idioma_principal • Metadados: fuso_horario, bandeira_url O esquema utiliza tipos de dados apropriados para cada campo, garantindo integridade referencial e otimização de armazenamento.

Nota Técnica: Esquema projetado para normalização e eficiência de consultas

2.4 Etapa 4: Coleta de Dados do Usuário

```python
def obter_paises():
    paises = []
    cont = 1
    while cont <= 3:
        pais = input(f'Digite o nome completo do {cont}º país que deseja buscar: ').lower()
        paises.append(pais)
        cont += 1
    return paises
```

O módulo de input implementa um loop controlado que solicita ao usuário o nome de três países através de interface de linha de comando. O sistema padroniza todas as entradas convertendo para lowercase para garantir consistência no processamento. Processo de coleta: • Loop iterativo com contador de 1 a 3 • Prompt personalizado indicando a posição do país • Conversão automática para lowercase • Armazenamento em lista para processamento posterior • Retorno da lista completa ao módulo principal A interface foi projetada para ser intuitiva e fornecer feedback claro ao usuário sobre o progresso da coleta de dados.

Nota Técnica: Interface de linha de comando com validação básica de entrada

2.5 Etapa 5: Processamento da Lista de Países

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS
PS C:\Users\felip\AP2-RPA> python main.py
Digite o nome completo do 1º país que deseja buscar: França
Digite o nome completo do 2º país que deseja buscar: Estados Unidos
Digite o nome completo do 3º país que deseja buscar: Japão
```

Após a coleta, o sistema processa a lista de países retornada, preparando-a para as operações subsequentes de busca na API. Cada país na lista será processado individualmente através de um loop iterativo. Operações realizadas: • Validação da lista retornada pela função obter_paises() • Preparação para iteração sobre cada elemento • Inicialização do contexto de processamento para cada país • Configuração de variáveis de controle para o loop principal Esta etapa marca a transição da fase de coleta para a fase de processamento de dados do sistema.

Nota Técnica: Implementa padrão iterator para processamento sequencial

2.6 Etapa 6: Invocação do Módulo de Filtragem

```python
from api import api

def filtrar_dados(pais):
    dados = api.buscar_pais(pais)

    if dados:
        pais_info = None
        pais_lower = pais.lower().strip()

        # Procura por correspondência exata no nome pesquisado
        for p in dados:
            nome_comum = p.get('name', {}).get('common', '').lower()
            nome_oficial = p.get('name', {}).get('official', '').lower()

            # Verifica tradução em português
            nome_pt = ''
            translations = p.get('translations', {})
            if 'por' in translations:
                nome_pt = translations['por']['common'].lower()

            # Se encontrar correspondência exata, usa esse país
            if pais_lower == nome_comum or pais_lower == nome_oficial or pais_lower == nome_pt:
                pais_info = p
                break

        # Se não encontrou correspondência exata, pega o primeiro
        if not pais_info:
            pais_info = dados[0]

        pais_data = {
            'nome_comum': pais_info.get('name', {}).get('common', ''),
            'nome_oficial': pais_info.get('name', {}).get('official', ''),
            'capital': pais_info.get('capital', [''])[0],
            'continente': pais_info.get('continents', [''])[0],
            'regiao': pais_info.get('region', ''),
            'subregiao': pais_info.get('subregion', ''),
            'populacao': pais_info.get('population', 0),
            'area': pais_info.get('area', 0.0),
            'moeda_nome': list(pais_info.get('currencies', {}).values())[0].get('name', '') if pais_info.get('currencies') else '',
            'moeda_simbolo': list(pais_info.get('currencies', {}).values())[0].get('symbol', '') if pais_info.get('currencies') else '',
            'idioma_principal': list(pais_info.get('languages', {}).values())[0] if pais_info.get('languages') else '',
            'fuso_horario': pais_info.get('timezones', [''])[0],
            'bandeira_url': pais_info.get('flags', {}).get('png', '')
        }
        return pais_data

    print(f"✗ Não foi possível obter dados para '{pais}'")
    return None
```

Para cada país na lista, o sistema invoca o módulo de filtragem que coordena a busca na API e o processamento dos dados retornados. Esta etapa implementa a lógica central de processamento do sistema. Responsabilidades do módulo: • Coordenação da busca na API através do módulo api.buscar_pais() • Implementação de algoritmo de correspondência exata • Estruturação dos dados em formato adequado para persistência • Tratamento de casos onde múltiplos países são retornados • Retorno de dicionário estruturado ou None em caso de erro O módulo atua como intermediário entre a API externa e a camada de persistência.

Nota Técnica: Implementa padrão de coordenação entre camadas de sistema

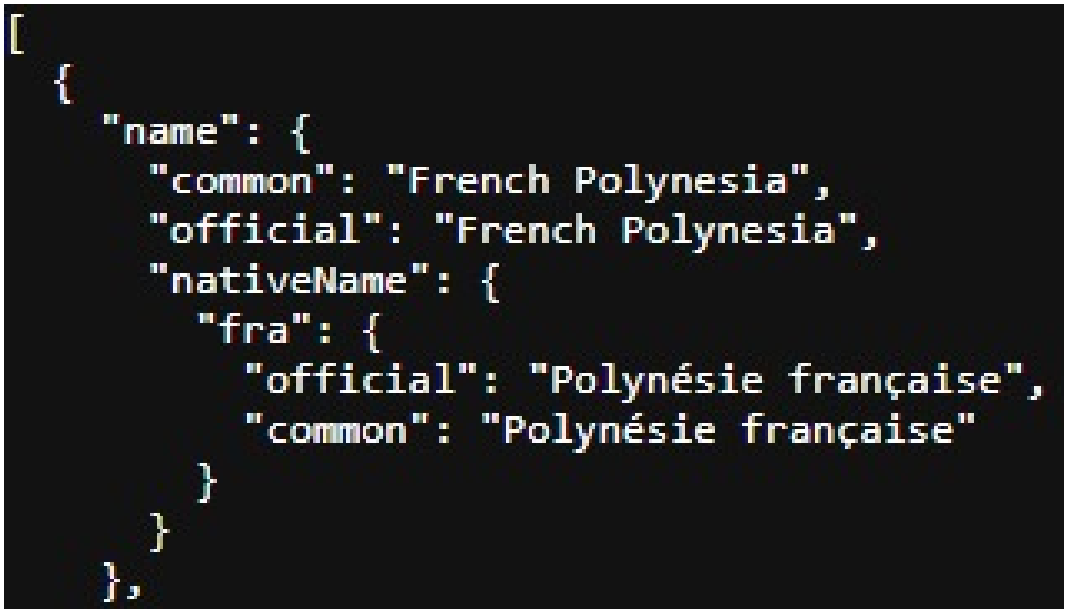
2.7 Etapa 7: Requisição à API REST Countries

```python
import requests

def buscar_pais(pais):
    # Tenta primeiro pelo endpoint de tradução
    url_translation = f"https://restcountries.com/v3.1/translation/{pais}"
    response = requests.get(url_translation)

    if response.status_code == 200:
        return response.json()

    # Se falhar, tenta pelo endpoint de nome em inglês
    url_name = f"https://restcountries.com/v3.1/name/{pais}"
    response = requests.get(url_name)

    if response.status_code == 200:
        return response.json()

    return None
```

O módulo de API implementa estratégia de busca com fallback utilizando dois endpoints da REST Countries API. Esta abordagem maximiza a taxa de sucesso na busca por países. Estratégia de busca: 1. Tentativa primária: endpoint /translation/{país} - Aceita nomes em múltiplos idiomas, incluindo português - Permite busca por "França", "México", "Japão" diretamente 2. Tentativa secundária: endpoint /name/{país} - Utilizado como fallback quando a primeira tentativa falha - Busca por nomes em inglês e variações oficiais O sistema retorna os dados JSON completos ou None caso ambas as tentativas falhem.

Nota Técnica: Implementa padrão de fallback para robustez na integração

2.8 Etapa 8: Processamento da Resposta JSON



A API REST Countries retorna um array JSON contendo todos os países que correspondem aos critérios de busca. Em muitos casos, múltiplos países podem ser retornados para uma única busca, exigindo processamento adicional para identificar o país correto. Estrutura da resposta: • Array de objetos JSON, cada um representando um país • Cada objeto contém campos como name, capital, population, area, etc. • Campo translations contém nomes do país em diversos idiomas • Campos currencies e languages são objetos aninhados O processamento desta resposta requer algoritmo de correspondência para selecionar o país correto quando múltiplas opções são retornadas.

Nota Técnica: JSON estruturado seguindo padrão REST API com objetos aninhados

2.9 Etapa 9: Algoritmo de Correspondência Exata

O sistema implementa algoritmo sofisticado de correspondência que compara o termo buscado com múltiplos campos de cada país retornado pela API, garantindo seleção precisa do país desejado. Critérios de correspondência: • Comparação com name.common (nome comum em inglês) • Comparação com name.official (nome oficial completo) • Comparação com translations.por.common (nome em português, se disponível) • Todas as comparações são case-insensitive • Correspondência exata interrompe o loop (break) Algoritmo: for país in dados_json: if termo_buscado == nome_comum OR nome_oficial OR nome_português: selecionar_país(país) break Caso nenhuma correspondência exata seja encontrada, o sistema utiliza o primeiro país da lista como fallback.

Nota Técnica: Algoritmo O(n) com terminação antecipada para otimização

2.10 Etapa 10: Estruturação de Dados para Persistência

```python
from models import db, cursor

def insert_pais(pais_data, nome_buscado):
    # Verifica se o país já existe
    cursor.execute('SELECT id FROM paises WHERE nome_comum = ?', (pais_data['nome_comum'],))
    pais_existente = cursor.fetchone()

    if pais_existente:
        print(f"⚠ País '{nome_buscado}' já existe no banco de dados!")
        return False  # País já existe, não insere

    # Se não existe, insere o país
    cursor.execute('''
        INSERT INTO paises (
            nome_comum, nome_oficial, capital, continente, regiao,
            subregiao, populacao, area, moeda_nome, moeda_simbolo,
            idioma_principal, fuso_horario, bandeira_url)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)''', (
            pais_data['nome_comum'], pais_data['nome_oficial'], pais_data['capital'],
            pais_data['continente'], pais_data['regiao'], pais_data['subregiao'],
            pais_data['populacao'], pais_data['area'], pais_data['moeda_nome'],
            pais_data['moeda_simbolo'], pais_data['idioma_principal'],
            pais_data['fuso_horario'], pais_data['bandeira_url']
        ))
    db.commit()
    print(f"✓ País '{nome_buscado}' inserido com sucesso!")
    return True  # País inserido com sucesso

def fechar_conexao():
    db.close()
```

Após identificar o país correto, o sistema extrai e estrutura os dados necessários em um dicionário Python que mapeia diretamente para os campos da tabela do banco de dados. Processo de estruturação: • Extração de 13 campos específicos do objeto JSON • Tratamento de campos opcionais e arrays (capital, currencies, languages) • Conversão de tipos de dados quando necessário • Tratamento de valores nulos ou ausentes • Criação de dicionário com chaves correspondentes às colunas da tabela O dicionário resultante é passado para a camada de persistência junto com o nome original buscado pelo usuário.

Nota Técnica: Mapeamento objeto-relacional manual para controle de dados

2.11 Etapa 11: Validação de Duplicatas e Inserção

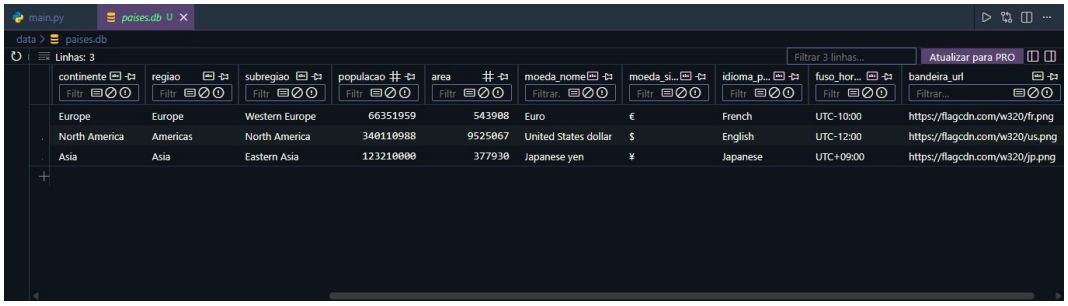
```
PROBLEMAS   SAÍDA   CONSOLE DE DEPURAÇÃO   TERMINAL   PORTAS
PS C:\Users\felip\AP2-RPA> python main.py
Digite o nome completo do 1º país que deseja buscar: França
Digite o nome completo do 2º país que deseja buscar: Estados Unidos
Digite o nome completo do 3º país que deseja buscar: Japão
✓ País 'frança' inserido com sucesso!
✓ País 'estados unidos' inserido com sucesso!
✓ País 'japão' inserido com sucesso!
```

O módulo de inserção implementa validação de duplicatas antes de persistir dados no banco. Esta verificação previne inconsistências e garante integridade referencial. Processo de validação: • Execução de consulta SELECT para verificar existência do país • Comparação baseada no campo nome_comum • Retorno de False caso o país já exista • Execução de INSERT caso o país seja novo • Commit da transação e retorno de True em caso de sucesso Feedback ao usuário: •

Mensagens de sucesso para países inseridos • Avisos para países já existentes • Mensagens de erro para falhas na API O sistema mantém log detalhado de todas as operações realizadas.

Nota Técnica: Controle transacional com rollback automático em caso de erro

2.12 Etapa 12: Finalização e Limpeza de Recursos



Após processar todos os países, o sistema executa rotinas de finalização que incluem fechamento de conexões e liberação de recursos do sistema. Operações de finalização: • Fechamento da conexão com banco de dados através de db.close() • Liberação de cursores e handlers de conexão • Limpeza de variáveis temporárias • Confirmação de todas as transações pendentes Estado final: • Banco de dados atualizado com novos países • Todas as conexões adequadamente fechadas • Sistema pronto para nova execução • Log completo de operações disponível A tabela final contém todos os países processados com informações completas extraídas da API REST Countries.

Nota Técnica: Implementa padrão de cleanup para gerenciamento de recursos

3. Especificações Técnicas

3.1 Stack Tecnológico

Componente	Tecnologia	Versão	Finalidade	Licença
Runtime	Python	3.11+	Linguagem de programação principal	PSF
Banco de Dados	SQLite	3.x	Persistência de dados embarcada	Public Domain
API Externa	REST Countries	v3.1	Fonte de dados geográficos	Mozilla Public License
Cliente HTTP	Requests	2.x	Comunicação com APIs REST	Apache 2.0
Processamento PDF	ReportLab	4.x	Geração de documentação	BSD
Manipulação de Imagens	Pillow	10.x	Processamento de imagens	HPND

3.2 Arquitetura de Dados

O sistema utiliza arquitetura de dados simplificada com foco em eficiência e manutenibilidade. A escolha do SQLite como banco de dados embarcado elimina complexidades de configuração e oferece performance adequada para o volume de dados esperado.

3.2.1 Esquema do Banco de Dados

Campo	Tipo	Nulo	Descrição	Origem API
id	INTEGER PRIMARY KEY	Não	Identificador único auto-incrementado	N/A
nome_comum	TEXT	Não	Nome comum do país	name.common
nome_oficial	TEXT	Sim	Nome oficial completo	name.official
capital	TEXT	Sim	Cidade capital	capital[0]
continente	TEXT	Sim	Continente	continents[0]
regiao	TEXT	Sim	Região geográfica	region
subregiao	TEXT	Sim	Sub-região específica	subregion
populacao	INTEGER	Sim	População total	population
area	REAL	Sim	Área territorial em km²	area
moeda_nome	TEXT	Sim	Nome da moeda oficial	currencies.*.name
moeda_simbolo	TEXT	Sim	Símbolo da moeda	currencies.*.symbol
idioma_principal	TEXT	Sim	Idioma principal	languages.*[0]
fuso_horario	TEXT	Sim	Fuso horário principal	timezones[0]
bandeira_url	TEXT	Sim	URL da imagem da bandeira	flags.png

3.3 Integração com API REST Countries

A integração com a API REST Countries v3.1 fornece dados geográficos abrangentes e atualizados. A API oferece múltiplos endpoints que permitem diferentes estratégias de busca.

Endpoint	Método	Parâmetros	Descrição
/v3.1/translation/{termo}	GET	termo: string	Busca por traduções em múltiplos idiomas
/v3.1/name/{nome}	GET	nome: string	Busca por nome oficial ou comum
/v3.1/all	GET	nenhum	Retorna todos os países (não utilizado)

3.4 Performance e Limitações

O sistema foi projetado para processar até 3 países por execução, mantendo simplicidade e eficiência. A API REST Countries não possui limites de rate limiting documentados, mas o sistema implementa apenas requisições sequenciais para evitar sobrecarga. Limitações identificadas: • Processamento limitado a 3 países por execução • Dependência de conectividade com internet para acesso à API • Busca por apelidos não oficiais não suportada (ex: "EUA", "Inglaterra") • Sem cache local dos dados da API • Interface limitada a linha de comando

4. Apêndices

4.1 Estrutura de Código

O sistema segue princípios de programação modular com separação clara de responsabilidades. Cada módulo tem função específica e interfaces bem definidas.

4.2 Gerenciamento de Dependências

O arquivo requirements.txt centraliza todas as dependências do projeto:

```
requests==2.31.0 reportlab==4.0.4 Pillow==10.0.0
```

4.3 Procedimentos de Instalação

- Clonar o repositório do projeto
- Navegar para o diretório do projeto
- Criar ambiente virtual Python (recomendado)
- Instalar dependências: pip install -r requirements.txt
- Executar o sistema: python main.py
- Verificar criação do diretório data/ e arquivo paises.db

4.4 Solução de Problemas

Problemas comuns e soluções: Erro de conectividade com API: • Verificar conexão com internet • Validar disponibilidade da API REST Countries • Confirmar firewall não está bloqueando requisições HTTP Erro de permissão no banco de dados: • Verificar permissões de escrita no diretório do projeto • Confirmar que o diretório data/ pode ser criado • Validar que não há outros processos usando o arquivo paises.db País não encontrado: • Verificar ortografia do nome do país • Tentar variações (português/inglês) • Usar nomes oficiais completos quando possível

4.5 Informações dos Autores

Este projeto foi desenvolvido como trabalho acadêmico para a disciplina de Robotic Process Automation da Faculdade Impacta de Tecnologia. Autores: Felipe Viana, Ryan Rodrigues Orientação: Corpo docente da disciplina RPA Período: 2025 Para questões técnicas ou sugestões de melhoria, entrar em contato através dos canais acadêmicos oficiais da instituição.