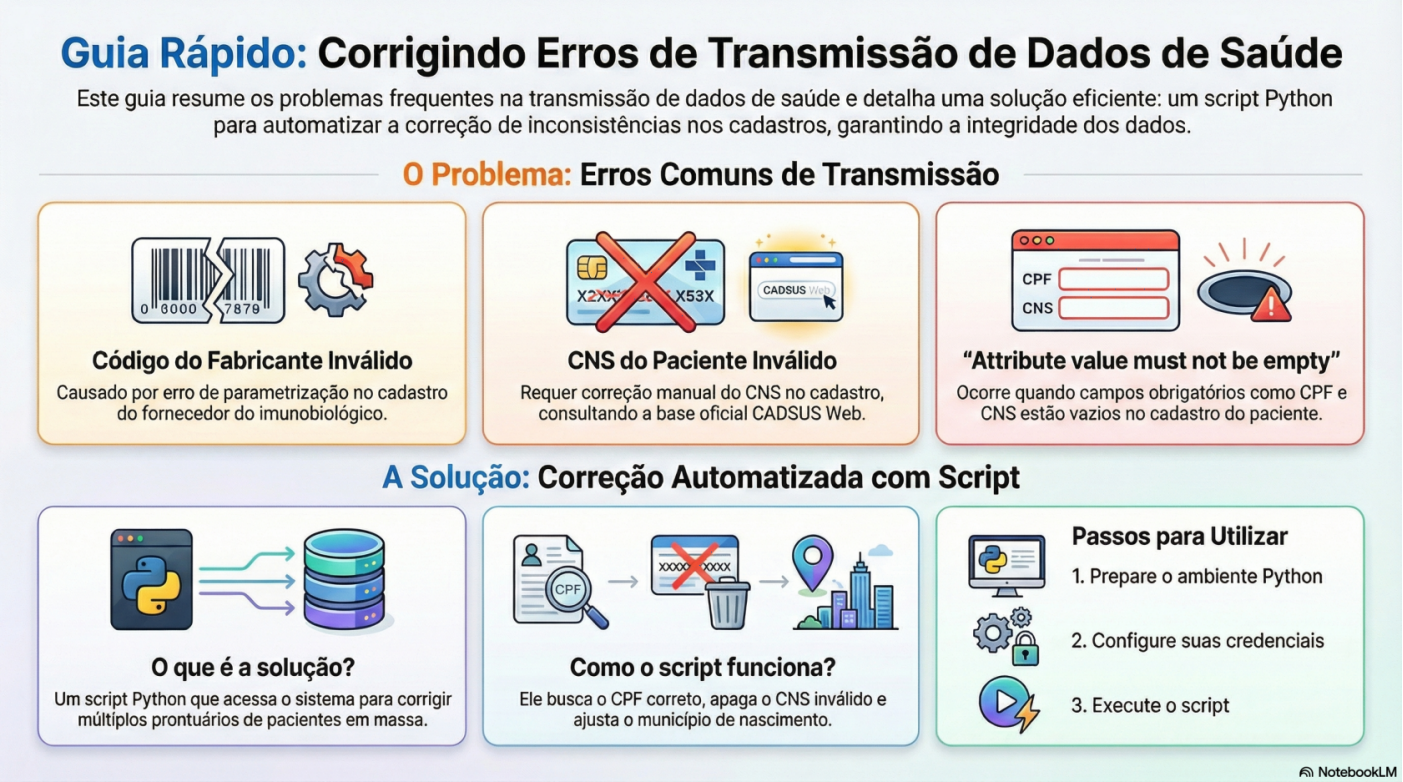
Problemas e Correções



É indicado que as transmissões sejam realizadas com intervalo de dois dias anteriores a data da transmissão, para que haja tempo hábil da transmissão por parte de todas as unidades de saúde.

Sempre que transmitido, o transmissor deve verificar os logs de erro para realizar as correções e realizar nova transmissão. Há disponibilização de um documento específico para Mensagens de Erro no Portal de Serviços - RIA.

Também foi criado, baseado nesse documento, um GPT específico que pode orientar quanto aos erros demonstrados em log (clique aqui para acessar o GPT "Assistente de Erros RNDS - Por DIEGO BISPO FERNANDES").

Dos erros mais comuns encontrados nas transmissões, até o presente momento (28/05/2025), as soluções são dadas a partir de:

- **Erro de código do Fabricante (0) -** Erro de parametrização do Fornecedor do Imunobiológico, para isso, é indicado a visualização da aplicação no paciente em Imunização > Registro no cartão, dessa maneira você será capaz de identificar qual é o fornecedor do imunobiológico e, assim, deve ser localizado o Fornecedor em Almoxarifado > Cadastro de Fabricante e corrigido o código RNDS, de acordo com a tabela Simplifier do RNDS.
- **Erros de registro**

- **Erros de dados Cadastrais**
  - **Erro de CNS do Paciente -** Para corrigir, basta corrigir o CNS no cadastro do paciente, de acordo com o CNS encontrado CADSUS Web oficial. A Integração não corrige na totalidade os cadastros por algum problema do webservice, portanto, indica-se sempre atualizar de acordo com o oficial (ATENÇÃO: Cadastros novos são validados em 01 dia, portanto, a retransmissão de cadastros novos deve ocorrer 01 dia após a correção).
  - **Erro "Attribute value must not be empty" -** Esse erro representa falha no cadastro do paciente, onde um ou mais campos responsáveis pela integração estão vazios. Via de regra, CPF e CNS constam vazios e impossibilitam a integração.

Para os erros de dados Cadastrais, encontramos uma solução que apoia a retransmissão. Trata-se de um script Python que realiza login no sistema com as credenciais de quem estiver atarefado a realizar as correções, verifica a lista de prontuários indicada pelo corretor e que, ao executar, entra prontuário a prontuário consultando os dados na integração ao CADSUS para capturar o CPF do Paciente, Apagar o CNS e corrigir a informação de Município de Nascimento do Paciente.

Antes de fazer download do Script, é importante garantir que o seu computador tenha os recursos necessários para a execução do mesmo, para isso, preparamos os materiais abaixo:

Preparando ambiente Python para Windows

Execute no PowerShell o script abaixo

```
# Instalar Python 3.12
winget install --id Python.Python.3.12 -e --accept-package-agreements --accept-source-agreements

# Recarregar variáveis de ambiente
$env:Path = [System.Environment]::GetEnvironmentVariable("Path","Machine") + ";" +
            [System.Environment]::GetEnvironmentVariable("Path","User")

# Criar ambiente virtual
python -m venv venv

# Ativar ambiente virtual
.\venv\Scripts\Activate

# Atualizar pip
python -m pip install --upgrade pip

# Instalar dependências necessárias
pip install requests beautifulsoup4
```

```
# Teste de imports
python - << EOF
import requests
from bs4 import BeautifulSoup
from datetime import datetime
import time
import re
print("Ambiente Python pronto para execução do script.")
EOF
```

Preparando ambiente Python para Zorin (Linux)

Execute no Terminal o script abaixo:

```
#!/bin/bash

# Atualizar repositórios
sudo apt update -y

# Instalar Python 3, pip, venv e dependências de compilação
sudo apt install -y python3 python3-pip python3-venv build-essential

# Criar ambiente virtual
python3 -m venv venv

# Ativar ambiente virtual
source venv/bin/activate

# Atualizar pip
pip install --upgrade pip

# Instalar dependências necessárias
pip install requests beautifulsoup4

# Teste de imports
```

```
python - << 'EOF'
import requests
from bs4 import BeautifulSoup
from datetime import datetime
import time
import re
print("Ambiente Python pronto para execução do script.")
EOF
```

Agora, com o ambiente preparado, você pode baixar o script no link a seguir: wsCorrigeCadastroSIS.py

```
import requests
from bs4 import BeautifulSoup
from datetime import datetime
import time
import re

#                 □□ CONFIGURAÇÃO DE USUÁRIO E SENHA - EDITE AQUI □□
# ═══════════════════════════════════════════════════════════════════════════
USUARIO_SIS = "[INFORME SEU SMS]"  # ⬅ EDITE AQUI: Informe seu SMS
SENHA_SIS = "[INFORME SUA SENHA]"  # ⬅ EDITE AQUI: Coloque a senha do sistema
# ═══════════════════════════════════════════════════════════════════════════
#                 □□ LISTA DE PRONTUÁRIOS - COLE AQUI □□
# ═══════════════════════════════════════════════════════════════════════════
# IMPORTANTE: Cole a lista de prontuários abaixo, um por linha
#
# REGRAS DE FORMATAÇÃO:
# - Cada prontuário deve estar em uma linha separada
# - Separe os prontuários por VÍRGULA
# - O ÚLTIMO prontuário NÃO deve ter vírgula no final
# ═══════════════════════════════════════════════════════════════════════════
PRONTUARIOS = [
1035375,
459159,
760375,
995608,
```

```
1136393
]
# ═══════════════════════════════════════════════════════════════════════════
#                    FIM DA LISTA DE PRONTUÁRIOS
# ═══════════════════════════════════════════════════════════════════════════

# === CONFIGURAÇÕES GERAIS ===
# O sistema adiciona automaticamente o domínio ao usuário
USUARIO_PADRAO = f"{USUARIO_SIS}@sisweb.sorocaba.sp.gov.br"
SENHA_PADRAO = SENHA_SIS
URL_BASE = "https://sisweb.sorocaba.sp.gov.br"

# Valores padrão usados quando campos obrigatórios estão vazios
COD_MUNICIPIO_PADRAO = "3552205"
UF_PADRAO = "SP"
COD_PAIS_NASC_PADRAO = "10"  # Brasil (igual exemplo do formulário)

# Lista para registrar prontuários pulados por falta de CPF
PRONTUARIOS_SEM_CPF = []

# Lista para registrar prontuários processados com sucesso (melhor visibilidade em integrações)
PRONTUARIOS_CORRIGIDOS = []

# Lista para armazenar registros detalhados das atualizações (para CSV)
REGISTROS_ATUALIZACAO = []



def fazer_login(usuario=None, senha=None):
    """
    Realiza login no SISWEB e retorna (session, csrf_token).
    Se usuário/senha não forem informados, usa os padrões.
    Adiciona automaticamente @sisweb.sorocaba.sp.gov.br se o usuário não tiver domínio.
    """
    if not usuario:
        usuario = USUARIO_PADRAO
    if not senha:
        senha = SENHA_PADRAO

    # Adiciona o domínio automaticamente se o usuário não tiver @
    if "@" not in usuario:
```

```
        usuario = f"{usuario}@sisweb.sorocaba.sp.gov.br"

    print(f"□□ Fazendo login no SISWEB com o usuário {usuario}...")

    session = requests.Session()
    session.headers.update({
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/142.0.0.0 Safari/537.36",
        "Accept-Language": "pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7,es;q=0.6",
        "Connection": "keep-alive",
        "X-Requested-With": "XMLHttpRequest",
    })

    # Página de login para obter o CSRF
    res_login = session.get(f"{URL_BASE}/login")
    res_login.raise_for_status()

    try:
        csrf_token = res_login.text.split('meta name="csrf-token" content="')[1].split('"')[0]
    except Exception:
        raise RuntimeError("Não foi possível localizar o CSRF token na tela de login.")

    # A "conta" é a parte antes do @
    conta = usuario.split("@")[0] if "@" in usuario else usuario

    payload = {
        "utf8": "✓",
        "page": "",
        "page_query": "",
        "conta": conta,
        "password": senha,
        "commit": "Entrar",
    }

    resp = session.post(
        f"{URL_BASE}/login/create",
        data=payload,
        headers={
            "Referer": f"{URL_BASE}/login",
            "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
```

```
            "X-CSRF-Token": csrf_token,
        },
    )
    resp.raise_for_status()
    print("✅ Login realizado.\n")
    return session, csrf_token


def buscar_paciente_id_por_prontuario(session, csrf_token, numprontuario):
    """
    Usa o endpoint amb/paciente.json para encontrar o ID interno do paciente (DT_RowId)
    a partir do número de prontuário.
    """
    params = {
        "draw": "1",
        "order[0][column]": "1",
        "order[0][dir]": "asc",
        "start": "0",
        "length": "1",
        "search[value]": "",
        "search[regex]": "false",
        "search_operator": "",
        "amb_paciente[numprontuario]": str(numprontuario),
        # datas podem ser vazias; o filtro principal é o prontuário
        "amb_paciente[datcadastro]": "",
        "amb_paciente[datalteracao]": "",
        "workMode": "wmSearchResult",
        "oldWorkMode": "wmSearch",
        "_": str(int(time.time() * 1000)),  # timestamp em ms, igual ao navegador
    }

    resp = session.get(
        f"{URL_BASE}/amb/paciente.json",
        params=params,
        headers={
            "Accept": "application/json, text/javascript, */*; q=0.01",
            "X-CSRF-Token": csrf_token,
            "X-Requested-With": "XMLHttpRequest",
            "Referer": f"{URL_BASE}/amb/paciente",
        },
```

```
    )
    resp.raise_for_status()

    try:
        data = resp.json()
    except Exception as e:
        print(f"  Erro ao decodificar JSON de amb/paciente.json (prontuário {numprontuario}): {e}")
        print(f"  Resposta bruta: {resp.text[:500]}")
        return None

    registros = data.get("data", [])
    if not registros:
        print(f"  Nenhum registro encontrado em amb/paciente.json para o prontuário {numprontuario}.")
        return None

    primeiro = registros[0]
    paciente_id = primeiro.get("DT_RowId")
    if not paciente_id:
        print(f"  Campo DT_RowId não encontrado no retorno de amb/paciente.json para o prontuário
{numprontuario}.")
        return None

    print(f"  Paciente encontrado em amb/paciente.json: ID interno {paciente_id}.")
    return paciente_id



def carregar_form_edicao(session, paciente_id):
    """
    Carrega a tela de edição do paciente e devolve (soup, authenticity_token_form).
    """
    url_edit = f"{URL_BASE}/amb/paciente/{paciente_id}/edit"
    params = {
        "workMode": "wmEdit",
        "oldWorkMode": "wmBrowse",
    }

    resp = session.get(
        url_edit,
        params=params,
        headers={
```

```
            "Accept": "text/html, */*; q=0.01",
            "Referer": f"{URL_BASE}/amb/paciente",
        },
    )
    resp.raise_for_status()

    soup = BeautifulSoup(resp.text, "html.parser")

    # authenticity_token do formulário (hidden field)
    token_input = soup.find("input", {"name": "authenticity_token"})
    if not token_input or not token_input.get("value"):
        raise RuntimeError("Não foi possível localizar authenticity_token no formulário de edição.")

    authenticity_token = token_input["value"]
    return soup, authenticity_token


def extrair_payload_form(soup):
    """
    Lê todos os campos do formulário principal de paciente e devolve um dict
    pronto para ser usado no POST (patch).
    """
    form = soup.find("form")
    if not form:
        raise RuntimeError("Formulário de paciente não encontrado na página de edição.")

    payload = {}

    # Inputs
    for inp in form.find_all("input"):
        name = inp.get("name")
        if not name:
            continue

        input_type = (inp.get("type") or "").lower()

        if input_type in ["checkbox", "radio"]:
            if inp.has_attr("checked"):
                value = inp.get("value", "1")
            else:
```

```
                # Não envia checkbox/radio não marcados
                continue
            else:
                value = inp.get("value", "")

            payload[name] = value

    # Textareas
    for ta in form.find_all("textarea"):
        name = ta.get("name")
        if not name:
            continue
        payload[name] = ta.text or ""

    # Selects
    for sel in form.find_all("select"):
        name = sel.get("name")
        if not name:
            continue
        option_sel = sel.find("option", selected=True)
        if option_sel is not None:
            value = option_sel.get("value", "")
        else:
            # fallback: primeira opção
            first_opt = sel.find("option")
            value = first_opt.get("value", "") if first_opt else ""
        payload[name] = value

    return payload


def buscar_cpf_cadsus(session, csrf_token, payload, numprontuario):
    """
    Tenta buscar o CPF na integração CADSUS usando nome do paciente e nome da mãe.
    Retorna o CPF (string) ou None se não encontrar.
    """
    nome = payload.get("amb_paciente[nompaciente]", "").strip()
    mae = payload.get("amb_paciente[nommae]", "").strip()

    if not nome or not mae:
```

```
        print("  ⚠ Não há nome e/ou nome da mãe suficientes para pesquisar no CADSUS.")
        return None

    params = {
        "amb_paciente[nome_completo_cadsus]": nome,
        "amb_paciente[nome_mae_cadsus]": mae,
        "amb_paciente[nome_pai_cadsus]": "",
        "amb_paciente[data_nascimento_cadsus]": "",
        "amb_paciente[cpf_cadsus]": "",
        "amb_paciente[cns_cadsus]": "",
        "amb_paciente[tela]": "paciente",
        "_": str(int(time.time() * 1000)),
    }

    try:
        resp = session.get(
            f"{URL_BASE}/int/data_sus/cad_sus/pesquisar",
            params=params,
            headers={
                "Accept": "text/javascript, application/javascript, application/ecmascript, application/x-ecmascript, */*;
q=0.01",
                "X-CSRF-Token": csrf_token,
                "X-Requested-With": "XMLHttpRequest",
                "Referer": f"{URL_BASE}/amb/paciente",
            },
            timeout=30,
        )
        resp.raise_for_status()
    except Exception as e:
        print(f"  □  Erro ao consultar CADSUS para prontuário {numprontuario}: {e}")
        return None

    # A resposta é um JavaScript que injeta uma tabela HTML. Procuramos os atributos data-cpf no botão.
    # O HTML vem escapado dentro de string JS, então primeiro "desescapamos" aspas.
    texto = resp.text.replace('\\"', '"')
    # Primeiro data-cpf encontrado será usado
    m = re.search(r'data-cpf="([^"]+)"', texto)
    if not m:
        print(f"  ⚠ Nenhum CPF encontrado no CADSUS para prontuário {numprontuario}.")
        return None
```

```
    cpf = m.group(1).strip()
    print(f"  CPF obtido via CADSUS para prontuário {numprontuario}: {cpf}")
    return cpf


def adicionar_mensagem_observacao(payload, mensagem, usuario=None):
    """
    Adiciona uma mensagem no campo de observação (infoadicional), preservando o conteúdo existente.
    Se usuario não for informado, usa o padrão.
    """
    if not usuario:
        usuario = USUARIO_PADRAO

    # Extrai o operador (parte antes do @)
    operador = usuario.split("@")[0] if "@" in usuario else usuario

    # Data e hora atual
    agora = datetime.now()
    data_hora = agora.strftime("%d/%m/%Y %H:%M")

    # Monta a mensagem completa
    mensagem_completa = f"{data_hora} - Atualizado com Script pelo(a) Operador(a) {operador}, resultado:
{mensagem}"

    # Pega o conteúdo atual do campo de observação
    campo_obs = "amb_paciente[infoadicional]"
    conteudo_atual = payload.get(campo_obs, "").strip()

    # Se já tem conteúdo, adiciona nova linha antes da nova mensagem
    if conteudo_atual:
        novo_conteudo = f"{conteudo_atual}\n{mensagem_completa}"
    else:
        novo_conteudo = mensagem_completa

    payload[campo_obs] = novo_conteudo
    return mensagem_completa


def ajustar_campos_obrigatorios(payload, numprontuario):
```

```
    """
    Ajusta/valida campos obrigatórios:
    - CPF não pode ficar vazio (retorna False se faltar)
    - Município de nascimento não pode ficar vazio (preenche com padrão se necessário)
    - Endereço estruturado não pode ficar vazio (CEP + logradouro + bairro)

    Retorna:
    - True: todos os campos obrigatórios estão OK
    - False: faltam dados obrigatórios (CPF ou endereço)
    """
    global PRONTUARIOS_SEM_CPF

    # Município de nascimento - sempre preenche se vazio
    codmun_nasc = payload.get("amb_paciente[codmunicipionasc]", "").strip()
    if not codmun_nasc:
        print(f"  ℹ Município de nascimento vazio, preenchendo com padrão
{COD_MUNICIPIO_PADRAO}/{UF_PADRAO}.")
        payload["amb_paciente[codmunicipionasc]"] = COD_MUNICIPIO_PADRAO
        if "amb_paciente[sglufnasc]" in payload and not payload.get("amb_paciente[sglufnasc]", "").strip():
            payload["amb_paciente[sglufnasc]"] = UF_PADRAO

    # País de nascimento (mantém se já tiver; se não tiver, preenche Brasil padrão)
    if "amb_paciente[codpaisnasc]" in payload and not payload.get("amb_paciente[codpaisnasc]", "").strip():
        payload["amb_paciente[codpaisnasc]"] = COD_PAIS_NASC_PADRAO

    cpf = payload.get("amb_paciente[numcpf]", "").strip()
    if not cpf:
        print(f"  ⚠ CPF vazio no prontuário {numprontuario}.")
        PRONTUARIOS_SEM_CPF.append(numprontuario)
        return False

    # Endereço estruturado: município de residência, CEP, logradouro e bairro
    campos_endereco = {
        "amb_paciente[codmunicipiores]": "município de residência",
        "amb_paciente[codcep]": "CEP",
        "amb_paciente[codlogradouro]": "código do logradouro",
        "amb_paciente[codbairro]": "código do bairro",
    }

    for campo, desc in campos_endereco.items():
```

```
        if campo not in payload or not str(payload.get(campo, "")).strip():
            print(
                f"  ⚠ Campo de endereço '{desc}' ({campo}) está vazio no prontuário {numprontuario}."
            )
            return False

    return True



def tem_cns(payload):
    """
    Verifica se o payload tem algum campo de CNS preenchido.
    Retorna True se encontrar algum campo de CNS/cartão SUS com valor não vazio.
    """
    for chave in payload.keys():
        nome_lower = chave.lower()
        # pega qualquer campo de CNS e qualquer variação de numcartao/numccartao
        if "cns" in nome_lower or "numcartao" in nome_lower:
            valor = str(payload.get(chave, "")).strip()
            if valor:  # Se tem algum valor, retorna True
                return True
    return False



def obter_cns_antes(payload):
    """
    Retorna o primeiro valor de CNS encontrado no payload (antes de limpar).
    Usado para registrar no CSV.
    """
    for chave in payload.keys():
        nome_lower = chave.lower()
        if "cns" in nome_lower or "numcartao" in nome_lower:
            valor = str(payload.get(chave, "")).strip()
            if valor:
                return valor
    return ""



def limpar_cns(payload):
    """
```

```
    Zera todos os campos relacionados a CNS:
    - Qualquer campo cujo nome contenha 'cns'
    - Campos de cartão SUS usados como CNS (ex: numcartao / numccartao)
    """
    chaves_apagadas = []

    for chave in list(payload.keys()):
        nome_lower = chave.lower()
        # pega qualquer campo de CNS e qualquer variação de numcartao/numccartao
        if "cns" in nome_lower or "numcartao" in nome_lower:
            payload[chave] = ""
            chaves_apagadas.append(chave)

    return chaves_apagadas



def salvar_paciente(session, csrf_token, paciente_id, payload):
    """
    Envia o PATCH para salvar o paciente com CNS apagado.
    """
    url_save = f"{URL_BASE}/amb/paciente/{paciente_id}"

    # Garante parâmetros de modo de trabalho e método PATCH
    payload.setdefault("utf8", "✓")
    payload["_method"] = "patch"
    payload.setdefault("workMode", "wmEdit")
    payload.setdefault("oldWorkMode", "wmBrowse")

    resp = session.post(
        url_save,
        data=payload,
        headers={
            "Accept": "text/html, */*; q=0.01",
            "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
            "X-CSRF-Token": csrf_token,
            "X-Requested-With": "XMLHttpRequest",
            "Referer": f"{URL_BASE}/amb/paciente",
            "Origin": URL_BASE,
        },
    )
```

```
    if resp.status_code not in (200, 302):
        print(f"  Erro ao salvar (HTTP {resp.status_code}).")
        return False

    return True


def processar_prontuario(session, csrf_token, numprontuario, indice=None, total=None, usuario=None):
    global REGISTROS_ATUALIZACAO

    # Mostra progresso se indice e total foram fornecidos
    if indice is not None and total is not None:
        print(f"\n Prontuário {indice}/{total} - Processando prontuário {numprontuario}...")
    else:
        print(f"\n Processando prontuário {numprontuario}...")

    # Inicializa registro para CSV
    registro = {
        "prontuario": numprontuario,
        "cpf_antes": "",
        "cns_antes": "",
        "municipio_antes": "",
        "estado_antes": "",
        "cpf_depois": "",
        "cns_apagado": "Não",
        "sucesso": "Não",
        "mensagem": "",
        "data_hora": datetime.now().strftime("%d/%m/%Y %H:%M:%S"),
        "operador": ""
    }

    try:
        paciente_id = buscar_paciente_id_por_prontuario(session, csrf_token, numprontuario)
        if not paciente_id:
            registro["mensagem"] = "Paciente não encontrado"
            registro["operador"] = usuario.split("@")[0] if usuario and "@" in usuario else
(USUARIO_PADRAO.split("@")[0] if "@" in USUARIO_PADRAO else USUARIO_PADRAO)
            REGISTROS_ATUALIZACAO.append(registro)
            return
```

```
        soup, authenticity_token = carregar_form_edicao(session, paciente_id)
        payload = extrair_payload_form(soup)

        # Garante que o authenticity_token correto vá no payload
        payload["authenticity_token"] = authenticity_token

        # === CAPTURA DADOS ANTES DAS ALTERAÇÕES ===
        cpf_antes = payload.get("amb_paciente[numcpf]", "").strip()
        cns_antes = obter_cns_antes(payload)
        municipio_antes = payload.get("amb_paciente[codmunicipionasc]", "").strip()
        estado_antes = payload.get("amb_paciente[sglufnasc]", "").strip()

        registro["cpf_antes"] = cpf_antes
        registro["cns_antes"] = cns_antes
        registro["municipio_antes"] = municipio_antes
        registro["estado_antes"] = estado_antes

        # Verifica CPF atual (antes de buscar no CADSUS)
        cpf_atual = cpf_antes
        cpf_capturado_cadsus = None  # Rastreia se o CPF foi capturado do CADSUS

        # Se já tem CPF e não tem CNS, pode pular para acelerar o processo
        if cpf_atual:
            if not tem_cns(payload):
                print(f"  Prontuário já tem CPF e não tem CNS. Pulando para acelerar o processo.")
                registro["mensagem"] = "Prontuário já estava correto (tem CPF e não tem CNS)"
                registro["operador"] = usuario.split("@")[0] if usuario and "@" in usuario else
(USUARIO_PADRAO.split("@")[0] if "@" in USUARIO_PADRAO else USUARIO_PADRAO)
                REGISTROS_ATUALIZACAO.append(registro)
                return

        # Se CPF estiver vazio, tenta buscar via integração CADSUS
        if not cpf_atual:
            print("  CPF vazio no cadastro, tentando buscar via integração CADSUS...")
            cpf_cadsus = buscar_cpf_cadsus(session, csrf_token, payload, numprontuario)
            if cpf_cadsus:
                payload["amb_paciente[numcpf]"] = cpf_cadsus
                cpf_capturado_cadsus = cpf_cadsus  # Marca que foi capturado do CADSUS
```

```
# === LOGA DADOS IMPORTANTES DO CADASTRO ANTES DE QUALQUER ALTERAÇÃO ===
def pega(nome):
    return payload.get(nome, "")

print("  Dados atuais no cadastro:")
print(f"    - Prontuário.....: {pega('amb_paciente[numprontuario]')}")
print(f"    - Nome paciente.....: {pega('amb_paciente[nompaciente]')}")
print(f"    - Data nascimento.....: {pega('amb_paciente[datnascimento]')}")
print(f"    - Nome mãe.....: {pega('amb_paciente[nommae]')}")
print(f"    - CPF.....: {pega('amb_paciente[numcpf]')}")
print(f"    - Cartão SUS (numcartao)...: {pega('amb_paciente[numcartao]')}")
print(f"    - Cod logradouro.....: {pega('amb_paciente[codlogradouro]')}")
print(f"    - Cod bairro.....: {pega('amb_paciente[codbairro]')}")
print(f"    - CEP.....: {pega('amb_paciente[codcep]')}")
print(f"    - Cod munic. nascimento....: {pega('amb_paciente[codmunicipionasc]')}")
print(f"    - UF nascimento.....: {pega('amb_paciente[sglufnasc]')}")

# Ajusta campos obrigatórios (sempre preenche município de nascimento se vazio)
campos_ok = ajustar_campos_obrigatorios(payload, numprontuario)

# Se faltar CPF ou endereço, ainda tenta salvar pelo menos o município de nascimento
if not campos_ok:
    cpf = payload.get("amb_paciente[numcpf]", "").strip()
    tem_endereco_completo = all([
        payload.get("amb_paciente[codmunicipiores]", "").strip(),
        payload.get("amb_paciente[codcep]", "").strip(),
        payload.get("amb_paciente[codlogradouro]", "").strip(),
        payload.get("amb_paciente[codbairro]", "").strip(),
    ])

    if not cpf:
        # Falta CPF - adiciona mensagem e tenta salvar pelo menos município de nascimento
        usuario_para_msg = usuario if usuario else USUARIO_PADRAO
        mensagem = adicionar_mensagem_observacao(
            payload,
            "sem sucesso por dados insuficientes para coleta de CPF via integracao",
            usuario=usuario_para_msg
        )
        print(f"  Mensagem adicionada no campo de observacao: {mensagem}")
        print("  ⚠ Tentando salvar pelo menos municipio de nascimento...")
```

```
            # Tenta salvar mesmo sem CPF completo (só município de nascimento)
            ok = salvar_paciente(session, csrf_token, paciente_id, payload)
            registro["cpf_depois"] = ""
            registro["cns_apagado"] = "Sim" if cns_antes else "Não"
            registro["sucesso"] = "Parcial" if ok else "Não"
            registro["mensagem"] = "sem sucesso por dados insuficientes para coleta de CPF via integracao"
            registro["operador"] = usuario_para_msg.split("@")[0] if "@" in usuario_para_msg else
usuario_para_msg
            REGISTROS_ATUALIZACAO.append(registro)
            if ok:
                print(f"  Municipio de nascimento salvo com sucesso para o prontuario {numprontuario}.")
            else:
                print(f"  Falha ao salvar prontuario {numprontuario}.")
            return

        elif not tem_endereco_completo:
            # Falta endereço - não salva para não quebrar o cadastro
            print("  Prontuario pulado por falta de informacao de endereco obrigatoria.")
            registro["mensagem"] = "Pulado por falta de informação de endereço obrigatória"
            registro["operador"] = usuario.split("@")[0] if usuario and "@" in usuario else
(USUARIO_PADRAO.split("@")[0] if "@" in USUARIO_PADRAO else USUARIO_PADRAO)
            REGISTROS_ATUALIZACAO.append(registro)
            return

    # Limpa CNS (inclui numcartao)
    chaves_cns = limpar_cns(payload)
    cns_foi_apagado = len(chaves_cns) > 0
    if not chaves_cns:
        print("  ℹ Nenhum campo de CNS/cartao SUS encontrado para limpar (talvez ja esteja vazio).")
    else:
        print(f"  ✂ Campos CNS/cartao SUS apagados: {', '.join(chaves_cns)}")

    # Salva
    ok = salvar_paciente(session, csrf_token, paciente_id, payload)
    if ok:
        # Adiciona mensagem de sucesso no campo de observação
        usuario_para_msg = usuario if usuario else USUARIO_PADRAO
        cpf_final = payload.get("amb_paciente[numcpf]", "").strip()
```

```
        # Monta a mensagem de sucesso
        if cpf_capturado_cadsus:
            # CPF foi capturado do CADSUS
            mensagem_texto = f"atualizacao realizada com sucesso, CNS apagado e inserido CPF {cpf_capturado_cadsus} capturado pela integração CADSUS"
        else:
            # CPF já existia no cadastro
            mensagem_texto = f"atualizacao realizada com sucesso, CNS apagado. CPF {cpf_final} já estava no prontuário"

        mensagem = adicionar_mensagem_observacao(
            payload,
            mensagem_texto,
            usuario=usuario_para_msg
        )
        print(f"  Mensagem de sucesso adicionada no campo de observacao: {mensagem}")

        # Salva novamente com a mensagem de sucesso
        salvar_paciente(session, csrf_token, paciente_id, payload)

        print(f"  Salvamento reportado como sucesso pelo servidor para o prontuario {numprontuario}.")
        # Marca como corrigido (para consumo por integracoes / API)
        global PRONTUARIOS_CORRIGIDOS
        PRONTUARIOS_CORRIGIDOS.append(numprontuario)

        # Atualiza registro para CSV
        registro["cpf_depois"] = cpf_final
        registro["cns_apagado"] = "Sim" if cns_foi_apagado else "Não"
        registro["sucesso"] = "Sim"
        registro["mensagem"] = mensagem_texto
        registro["operador"] = usuario_para_msg.split("@")[0] if "@" in usuario_para_msg else usuario_para_msg
    else:
        print(f"  Falha ao salvar prontuario {numprontuario} (HTTP diferente de 200/302).")
        # Atualiza registro para CSV
        registro["cpf_depois"] = payload.get("amb_paciente[numcpf]", "").strip()
        registro["cns_apagado"] = "Sim" if cns_foi_apagado else "Não"
        registro["sucesso"] = "Não"
        registro["mensagem"] = "Falha ao salvar (erro HTTP)"
        registro["operador"] = usuario.split("@")[0] if usuario and "@" in usuario else (USUARIO_PADRAO.split("@")[0] if "@" in USUARIO_PADRAO else USUARIO_PADRAO)
```

```
        # Adiciona registro ao CSV
        REGISTROS_ATUALIZACAO.append(registro)

        # Pequena pausa entre pacientes
        time.sleep(1)

    except Exception as e:
        print(f"  Erro ao processar prontuário {numprontuario}: {e}")
        registro["mensagem"] = f"Erro ao processar: {str(e)}"
        registro["operador"] = usuario.split("@")[0] if usuario and "@" in usuario else
(USUARIO_PADRAO.split("@")[0] if "@" in USUARIO_PADRAO else USUARIO_PADRAO)
        REGISTROS_ATUALIZACAO.append(registro)


def gerar_txt_relatorio(nome_arquivo=None, operador=None):
    """
    Gera um arquivo TXT simples com o relatório de atualizações.
    Formato do nome: YYYY-MM-DD HHMM ws Corrige CPF SIS [operador].txt
    Conteúdo: Prontuário e status de sucesso (similar ao terminal)
    """
    if not REGISTROS_ATUALIZACAO:
        print("\n⚠ Nenhum registro para gerar relatório.")
        return

    if not nome_arquivo:
        agora = datetime.now()
        # Formato: YYYY-MM-DD HHMM (data ao contrário + hora sem dois pontos)
        data_hora = agora.strftime("%Y-%m-%d %H%M")

        # Extrai o operador (parte antes do @) se não foi informado
        if not operador:
            # Tenta pegar do primeiro registro ou usa padrão
            if REGISTROS_ATUALIZACAO and REGISTROS_ATUALIZACAO[0].get('operador'):
                operador = REGISTROS_ATUALIZACAO[0]['operador']
            else:
                operador = USUARIO_SIS

        nome_arquivo = f"{data_hora} ws Corrige CPF SIS {operador}.txt"
```

```
try:
    with open(nome_arquivo, 'w', encoding='utf-8') as txtfile:
        # Escreve cabeçalho
        txtfile.write("=" * 60 + "\n")
        txtfile.write("RELATÓRIO DE ATUALIZAÇÕES\n")
        txtfile.write(f"Operador: {operador}\n")
        txtfile.write(f"Data/Hora: {datetime.now().strftime('%d/%m/%Y %H:%M:%S')}\n")
        txtfile.write("=" * 60 + "\n\n")

        # Agrupa por status
        sucesso = []
        falha = []
        outros = []

        for registro in REGISTROS_ATUALIZACAO:
            prontuario = registro['prontuario']
            status = registro['sucesso']

            if status == "Sim":
                sucesso.append(prontuario)
            elif status == "Não":
                falha.append(prontuario)
            else:
                outros.append((prontuario, status))

        # Escreve prontuários com sucesso
        if sucesso:
            txtfile.write("□  PRONTUÁRIOS ATUALIZADOS COM SUCESSO:\n")
            for p in sucesso:
                txtfile.write(f"  - {p}\n")
            txtfile.write(f"\nTotal: {len(sucesso)} prontuário(s)\n\n")

        # Escreve prontuários com falha
        if falha:
            txtfile.write("□  PRONTUÁRIOS NÃO ATUALIZADOS:\n")
            for p in falha:
                txtfile.write(f"  - {p}\n")
            txtfile.write(f"\nTotal: {len(falha)} prontuário(s)\n\n")

        # Escreve outros status (parcial, pulado, etc)
```

```
            if outros:
                txtfile.write("⚠ OUTROS STATUS:\n")
                for p, status in outros:
                    txtfile.write(f"  - {p} ({status})\n")
                txtfile.write(f"\nTotal: {len(outros)} prontuário(s)\n\n")

            # Resumo final
            txtfile.write("=" * 60 + "\n")
            txtfile.write(f"RESUMO:\n")
            txtfile.write(f"  Total processado: {len(REGISTROS_ATUALIZACAO)}\n")
            txtfile.write(f"  Sucesso: {len(sucesso)}\n")
            txtfile.write(f"  Falha: {len(falha)}\n")
            if outros:
                txtfile.write(f"  Outros: {len(outros)}\n")
            txtfile.write("=" * 60 + "\n")

        print(f"\n Relatório TXT gerado com sucesso: {nome_arquivo}")
        print(f"   Total de registros: {len(REGISTROS_ATUALIZACAO)}")
        return nome_arquivo
    except Exception as e:
        print(f"\n  Erro ao gerar relatório TXT: {e}")
        return None


def main():
    global REGISTROS_ATUALIZACAO
    REGISTROS_ATUALIZACAO = []  # Limpa registros anteriores

    inicio = datetime.now()
    print(f" Início da execução: {inicio.strftime('%d/%m/%Y %H:%M:%S')}")

    session, csrf_token = fazer_login()

    # Acessa uma vez a tela de cadastro (como no uso manual)
    try:
        session.get(
            f"{URL_BASE}/amb/paciente",
            headers={
                "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
```

```
n/signed-exchange;v=b3;q=0.7",
                "Referer": f"{URL_BASE}/desktop",
            },
        )
    except Exception:
        # Não é crítico, segue o fluxo mesmo assim
        pass

    total_prontuarios = len(PRONTUARIOS)
    for indice, num in enumerate(PRONTUARIOS, start=1):
        processar_prontuario(session, csrf_token, num, indice=indice, total=total_prontuarios,
usuario=USUARIO_PADRAO)

    # Resumo de prontuários não alterados por falta de CPF
    if PRONTUARIOS_SEM_CPF:
        print("\n⚠ Prontuários NÃO alterados por estarem sem CPF:")
        for p in PRONTUARIOS_SEM_CPF:
            print(f"  - {p}")
    else:
        print("\n Nenhum prontuário foi pulado por falta de CPF.")

    # Gera relatório TXT
    operador_txt = USUARIO_SIS  # Usa o operador configurado
    gerar_txt_relatorio(operador=operador_txt)

    fim = datetime.now()
    duracao = (fim - inicio).total_seconds()
    print(f"\n Fim da execução em {duracao:.1f} segundos.")



def run_batch(prontuarios, usuario=None, senha=None):
    """
    Executa a correção para uma lista de prontuários e retorna um resumo em dict.
    Pensado para uso via API / interface web.
    """
    global PRONTUARIOS_SEM_CPF, PRONTUARIOS_CORRIGIDOS, REGISTROS_ATUALIZACAO
    PRONTUARIOS_SEM_CPF = []
    PRONTUARIOS_CORRIGIDOS = []
    REGISTROS_ATUALIZACAO = []
```

```
    inicio = datetime.now()
    session, csrf_token = fazer_login(usuario=usuario, senha=senha)

    # Acessa uma vez a tela de cadastro (como no uso manual)
    try:
        session.get(
            f"{URL_BASE}/amb/paciente",
            headers={
                "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7",
                "Referer": f"{URL_BASE}/desktop",
            },
        )
    except Exception:
        pass

    total_prontuarios = len(prontuarios)
    usuario_para_processar = usuario if usuario else USUARIO_PADRAO
    for indice, num in enumerate(prontuarios, start=1):
        processar_prontuario(session, csrf_token, num, indice=indice, total=total_prontuarios,
usuario=usuario_para_processar)

    fim = datetime.now()
    duracao = (fim - inicio).total_seconds()

    return {
        "total": len(prontuarios),
        "corrigidos": PRONTUARIOS_CORRIGIDOS,
        "nao_alterados_sem_cpf": PRONTUARIOS_SEM_CPF,
        "duracao": duracao,
    }



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

Revisão #3
Criado 5 janeiro 2026 11:42:52 por Diego Bispo Fernandes
Atualizado: 7 janeiro 2026 19:21:51 por João Carlos Ferreira de Araújo