**Projeto Final - Avaliação A3**

**VERIFICADOR DE BOLETOS**

Relatório técnico apresentado na UC Sistemas Distribuídos e Mobile orientado pelo Prof. MSc Flávio Henrique da Silva.

**Curitiba**

**2025**

**INTEGRANTES DO GRUPO**

Ana Julia Bernardo Lazaro – 172211672

Carlos Daniel Steindorf Pereira - 172220112

João Gabriel Breve - 172317201

**SUMÁRIO**

[**1 INTRODUÇÃO 4**](#_gi27gizbrylp)

[**2 DESENVOLVIMENTO 5**](#_6wxveu53vdjh)

[**2.1 Divisão das Tarefas 5**](#_ixkxve60l395)

[**2.2 Estrutura do Projeto 5**](#_63gh3cl9hbdf)

[**2.3 Explicação da Aplicação/Software 5**](#_xkuqbicun17u)

[**2.4 Orientações de execução da Aplicação/Software 5**](#_jookgbrqzcr2)

[**2.5 Repositório 5**](#_xnms1s2npl41)

[**3 CONCLUSÃO 6**](#_bb9sa9mxt6n2)

[**REFERÊNCIAS 7**](#_dz5kaqeu3jpt)

# 1 INTRODUÇÃO

O presente projeto foi desenvolvido com o objetivo de aplicar, de forma prática, os conhecimentos adquiridos na Unidade Curricular de Sistemas Distribuídos e Mobile, por meio da criação de uma ferramenta voltada à verificação de boletos bancários fraudulentos. Diante do aumento significativo de golpes envolvendo esse tipo de pagamento, torna-se cada vez mais necessário dispor de soluções tecnológicas que garantam a segurança e a confiabilidade das transações financeiras. De acordo com a revista Veja, golpistas vêm recorrendo a estratégias como a alteração de dados bancários, a criação de boletos falsos com logotipos de instituições reconhecidas e o envio desses documentos por canais não oficiais, como e-mails e mensagens de texto, com o intuito de enganar usuários desavisados.

Considerando esse cenário, o projeto foi idealizado para oferecer ao usuário final uma experiência segura, eficiente e objetiva. A ferramenta desenvolvida prioriza a proteção dos dados pessoais e bancários, ao mesmo tempo em que contribui para a educação digital, capacitando os usuários a reconhecer sinais de fraude e a tomar decisões conscientes antes de realizar pagamentos. Dessa forma, além de consolidar a aplicação de conhecimentos técnicos, este projeto propõe uma solução concreta e relevante para um problema atual, impactando positivamente a rotina de consumidores e empresas.

# 2 DESENVOLVIMENTO

Este projeto teve como objetivo principal o desenvolvimento de uma ferramenta capaz de verificar a autenticidade de boletos bancários, contribuindo para a prevenção de fraudes financeiras. A solução proposta visa oferecer uma forma simples e eficaz de análise, permitindo ao usuário identificar possíveis irregularidades antes de efetuar pagamentos. A motivação partiu da crescente ocorrência de golpes relacionados a boletos falsificados, o que destaca a importância de soluções tecnológicas que reforcem a segurança digital.

Ao longo do documento, serão apresentadas as principais etapas do desenvolvimento do projeto, iniciando pela divisão de tarefas entre os membros da equipe, evidenciando a colaboração e a organização do grupo durante o processo. Em seguida, será detalhada a estrutura do código e as tecnologias utilizadas, e por fim, será feita uma explicação das funcionalidades da ferramenta, demonstrando como ela opera na prática para auxiliar o usuário na verificação de boletos suspeitos.

2.1 Divisão das Tarefas

Durante o desenvolvimento do projeto as atividades foram distribuídas entre os integrantes do grupo de forma equilibrada, visando um fluxo de trabalho colaborativo e eficiente.

Ana Julia Bernardo Lazaro ficou responsável pela definição dos objetivos e funcionalidades do projeto, conduzindo a concepção inicial do sistema e estabelecendo as diretrizes para seu funcionamento. Além disso, também foi incumbida da documentação do sistema, garantindo o registro das decisões técnicas e funcionais ao longo do desenvolvimento.

Carlos Daniel Steindorf Pereira teve como foco a parte técnica inicial do projeto, sendo o responsável pelo desenvolvimento do código-fonte da ferramenta; realizando testes frequentes e aprimoramentos contínuos no ambiente de desenvolvimento, assegurando que o sistema evoluísse de forma estável e eficiente. Também foi o responsável pela criação do banco de dados, estruturando as tabelas necessárias para armazenar informações de instituições, boletos, tentativas suspeitas e a blacklist.

João Gabriel Breve atuou principalmente na camada de interface e integração, sendo o responsável pela criação do escopo HTML para testes, que permitiu simular ações como o cadastro de instituições, e simulação de pagamentos. Ele também implementou as rotas da API para registro e verificação de boletos, garantindo que todas as funcionalidades estivessem acessíveis e integradas ao sistema de forma segura e funcional.

Por fim, todos os integrantes participaram ativamente da distribuição das tarefas, garantindo que cada etapa do projeto fosse conduzida com comprometimento, cooperação e alinhamento com os objetivos propostos.

**2.2 Estrutura do Projeto**

**2.2.1 Bibliotecas e Frameworks Utilizados**

No desenvolvimento do sistema foram utilizadas diversas bibliotecas e frameworks que desempenham papéis essenciais tanto na estrutura da aplicação quanto no tratamento e análise dos dados. O principal framework utilizado foi o *Flask*, que permite a criação rápida de APIs RESTful e rotas de comunicação entre o front-end e o back-end. Para o armazenamento de dados, foi utilizado o *sqlite3*, um banco de dados relacional leve que já vem integrado ao Python.

Diversas bibliotecas da biblioteca padrão do Python também foram fundamentais para funcionalidades auxiliares. A biblioteca *secrets* foi utilizada para a geração segura de tokens e chaves aleatórias, contribuindo para a autenticação e segurança do sistema. A biblioteca *uuid* permite a criação de identificadores únicos universais para registros ou requisições, enquanto *re* (expressões regulares) é utilizada na validação e extração de padrões de texto, como o código de barras dos boletos. Já *random* e *string* auxiliam na geração de dados simulados, como nomes ou códigos aleatórios, para testes e simulações, e a biblioteca *datetime*, é essencial para manipulação de datas, como vencimento de boletos ou registros de verificação.

Uma funcionalidade importante do sistema é a comparação de nomes ou beneficiários, com o objetivo de detectar pequenas variações ou tentativas de disfarce. Para isso, é utilizada a classe *SequenceMatcher* do módulo *difflib*, que calcula a similaridade entre duas sequências de texto, possibilitando identificar nomes falsificados que são visualmente semelhantes ao original, um recurso comum em boletos fraudulentos.

Juntas, essas bibliotecas oferecem uma base sólida para o desenvolvimento de um sistema confiável de detecção de fraudes em boletos, combinando segurança, performance e flexibilidade no processamento das informações.

**2.2.2 Estrutura dos Arquivos**

O sistema foi organizado de forma modular, seguindo boas práticas de desenvolvimento web utilizando *Python* e *Flask*. Sua estrutura é formada basicamente por:

- app.py: É o ponto de entrada da aplicação. Onde é feita a inicialização do *Flask*, o registro dos blueprints (módulos de rotas) e a configuração geral da API.

- database.py: Responsável por gerenciar a conexão com o banco de dados *SQLite*.

- utils.py: Arquivo dedicado a funções auxiliares reutilizáveis, como o registro de logs e manipulação de dados, centralizando funcionalidades comuns da aplicação.

- schema.sql: Script SQL que define a estrutura do banco de dados.

- requirements.txt: Lista todas as dependências e bibliotecas utilizadas no projeto.

- routes/: Diretório que agrupa os arquivos responsáveis pelas rotas (endpoints) da API.

- templates/: Contém páginas HTML utilizadas para a interface web.

**2.2.3 Principais Objetos, Classes e Funções**

Diversas bibliotecas da própria linguagem Python foram utilizadas para garantir funcionalidades específicas. Entre elas, estão *secrets*, *uuid*, *random*, *string* e *datetime*, empregadas na geração de tokens, manipulação de dados e controle de datas. A biblioteca *re* foi usada para validações com expressões regulares, como no caso do CNPJ. Para verificação de semelhança entre nomes, a função “SequenceMatcher” da biblioteca *difflib* foi empregada, permitindo detectar pequenas variações que indicam tentativas de fraude. Por exemplo:

from flask import Blueprint, request, jsonify

from database import get\_connection

import secrets

import re

from utils import registrar\_log

A estrutura do projeto foi organizada de forma clara e modular. O arquivo principal, *app.py*, é o ponto de entrada da aplicação, onde a instância do *Flask* é criada e os módulos são registrados.

from flask import Flask, render\_template

from database import init\_db

from routes.instituicoes import bp as instituicoes\_bp

from routes.boletos import bp as boletos\_bp

from routes.verificacao import bp as verificacao\_bp

from routes.tentativa import bp as tentativas\_bp

app = Flask(\_\_name\_\_)

# Inicializa o banco de dados, se necessário

init\_db()

# Registra os blueprints (rotas organizadas por função)

app.register\_blueprint(instituicoes\_bp)

app.register\_blueprint(boletos\_bp)

app.register\_blueprint(verificacao\_bp)

app.register\_blueprint(tentativas\_bp)

# Rotas para as páginas web de demonstração

@app.route("/gerar-boleto")

def gerar\_boleto():

return render\_template("gerar\_boleto.html")

@app.route("/pagar-boleto")

def pagar\_boleto():

return render\_template("pagar\_boleto.html")

@app.route("/cadastrar-instituicao")

def cadastrar\_instituicao():

return render\_template("cadastrar\_instituicao.html")

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

O arquivo *database.py* é responsável pela conexão com o banco de dados, oferecendo a função “get\_connection()”, que retorna a conexão com resultados em formato de dicionário, e “init\_db()”, que inicializa as tabelas com base no arquivo *schema.sql*, garantindo que a estrutura mínima do sistema seja criada se não existir.

import sqlite3

import os

DB\_PATH = "boleto.db"

def get\_connection():

conn = sqlite3.connect(DB\_PATH)

conn.row\_factory = sqlite3.Row # Retorna dicionários ao invés de tuplas

return conn

def init\_db():

if not os.path.exists(DB\_PATH):

with open("schema.sql", "r", encoding="utf-8") as f:

script = f.read()

conn = get\_connection()

conn.executescript(script)

conn.commit()

conn.close()

As funções auxiliares, como o registro de logs de auditoria, estão em *utils.py*, com destaque para a função “registrar\_log()”, que grava eventos no banco de dados para monitoramento.

from database import get\_connection

def registrar\_log(acao, origem, dados):

"""

Registra uma ação no log do sistema.

:param acao: Nome da ação executada (ex: "registro\_boleto")

:param origem: Identificador da origem (ex: CNPJ, token, "usuário")

:param dados: Descrição ou conteúdo relevante do evento

"""

try:

conn = get\_connection()

cur = conn.cursor()

cur.execute("""

INSERT INTO log\_eventos (acao, origem, dados)

VALUES (?, ?, ?)

""", (acao, origem, dados))

conn.commit()

except Exception as e:

print(f"[ERRO AO REGISTRAR LOG] {e}")

finally:

conn.close()

As rotas da aplicação estão organizadas em módulos separados no diretório *routes/*, promovendo separação de responsabilidades. O módulo *instituicoes.py* é responsável pelo cadastro de instituições parceiras por meio do endpoint */registrar-instituicao*. Esse cadastro realiza validações de CNPJ e gera automaticamente um token de acesso para a instituição. Exemplo de parte do código:

bp = Blueprint("instituicoes", \_\_name\_\_)

@bp.route("/registrar-instituicao", methods=["POST"])

def registrar\_instituicao():

data = request.json

nome = data.get("nome\_fantasia")

cnpj = data.get("cnpj")

O módulo *boletos.py* trata do registro de boletos com o endpoint */registrar-boleto*, que exige o token da instituição para garantir a segurança. Além do registro, essa rota permite recuperar boletos associados a tentativas de fraude. Exemplo de parte do código que verifica se o token pertence à alguma instituição:

cursor.execute("SELECT id FROM instituicoes WHERE token\_acesso = ?", (token,))

instituicao = cursor.fetchone()

if not instituicao:

conn.close()

return jsonify({"erro": "Token inválido. Acesso negado."}), 403

Já o módulo *verificacao.py* é central no combate às fraudes. Ele oferece o endpoint */verificar-boleto*, que verifica se o boleto existe, se está na blacklist ou já foi pago. O endpoint */pagar-boleto* simula o pagamento de boletos, analisando divergências entre os dados do pagador e do beneficiário, e registrando tentativas suspeitas, se necessário. A função “nomes\_parecidos()” utilizada nesse módulo permite comparar a similaridade entre nomes, o que é crucial na detecção de alterações sutis que possam caracterizar fraude.

def nomes\_parecidos(nome1, nome2):

return SequenceMatcher(None, nome1.lower(), nome2.lower()).ratio() > 0.6

Para monitoramento e gestão de segurança, o módulo *tentativa.py* fornece rotas como */tentativas-suspeitas*, que exibe todas as ocorrências registradas com inconsistências, além de endpoints para gerenciar a blacklist, como */bloquear*, */blacklist* e */desbloquear*. Esses recursos permitem que administradores analisem e intervenham em casos considerados fraudulentos. Um exemplo prático é a parte do código que verifica se já existe o mesmo código de barras na blackllist:

cursor.execute("SELECT \* FROM blacklist WHERE codigo\_barras = ?", (tentativa["codigo\_barras"],))

existe = cursor.fetchone()

if existe:

conn.close()

return redirect("/blacklist") # Ou poderia mostrar uma mensagem

O banco de dados, definido no arquivo *schema.sql*, contém tabelas fundamentais para o funcionamento da aplicação. A tabela *instituicoes* armazena os dados das empresas cadastradas, como nome, CNPJ e token. A tabela boletos mantém os dados completos dos boletos gerados, incluindo código de barras, status, pagador e beneficiário. Já a tabela *tentativas\_suspeitas* registra todos os pagamentos que apresentaram divergências, enquanto a blacklist armazena CPFs ou CNPJs bloqueados por atividades suspeitas. Por fim, a tabela *log\_eventos* é usada para manter um histórico detalhado das ações executadas no sistema, oferecendo maior segurança e transparência. Exemplo de uma das tabelas mencionadas:

CREATE TABLE IF NOT EXISTS instituicoes (

id INTEGER PRIMARY KEY AUTOINCREMENT,

nome\_fantasia TEXT NOT NULL,

cnpj TEXT NOT NULL UNIQUE,

token\_acesso TEXT NOT NULL UNIQUE,

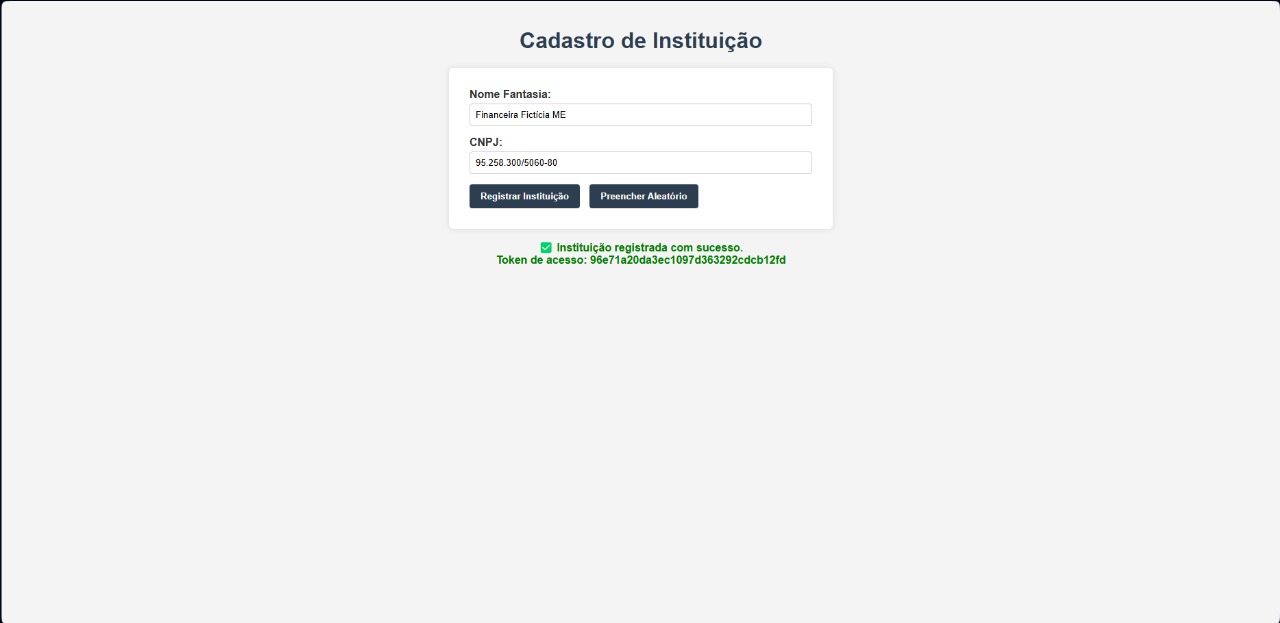
data\_cadastro DATETIME DEFAULT CURRENT\_TIMESTAMP

);

Além da API, a aplicação conta com uma interface web desenvolvida com templates HTML. A página *cadastrar\_instituicao.html* permite o cadastro de instituições via formulário. A *gerar\_boleto.html* oferece uma interface simples para geração de boletos, enquanto a *pagar\_boleto.html* simula a verificação e o pagamento de boletos. Também há páginas administrativas, como *ver\_tentativas\_suspeitas.html* e *ver\_blacklist.html*, que possibilitam a visualização e gerenciamento das tentativas de fraude e das entidades bloqueadas, respectivamente.

2.3 Explicação da Aplicação/Software

Primeiramente será necessário que seja realizado o cadastro de uma instituição que possa gerar boletos e receber o pagamento dos mesmos. Selecionando a opção “Preencher Aleatório” serão gerados dados para que uma instituição válida seja criada. Após a geração dos dados, será necessário selecionar a opção “Registrar Instituição”, onde após essa ação, irá ser gerado token.

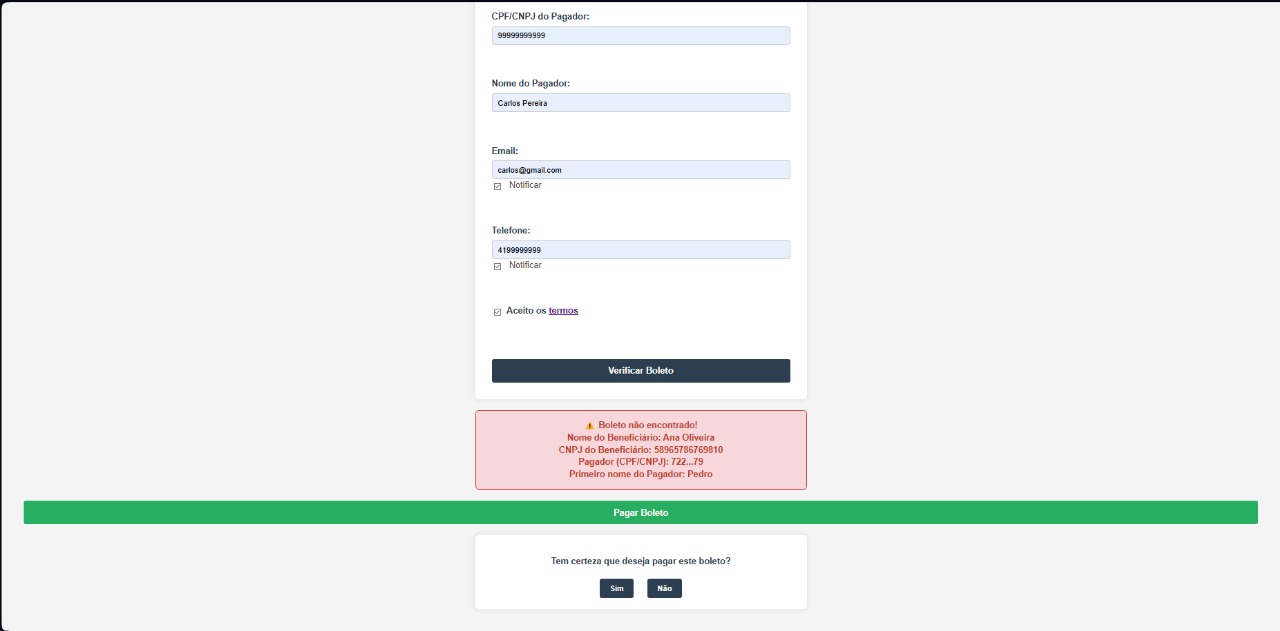


O token deverá ser utilizado na página “Simulador de Geração de Boleto (Instituição)” no campo “Token da Instituição”. Após a sua inserção, o usuário selecionará a opção “Preencher Aleatório” para que sejam criados os dados do boleto, depois “Registrar Boleto” (dessa forma será criado um boleto válido).

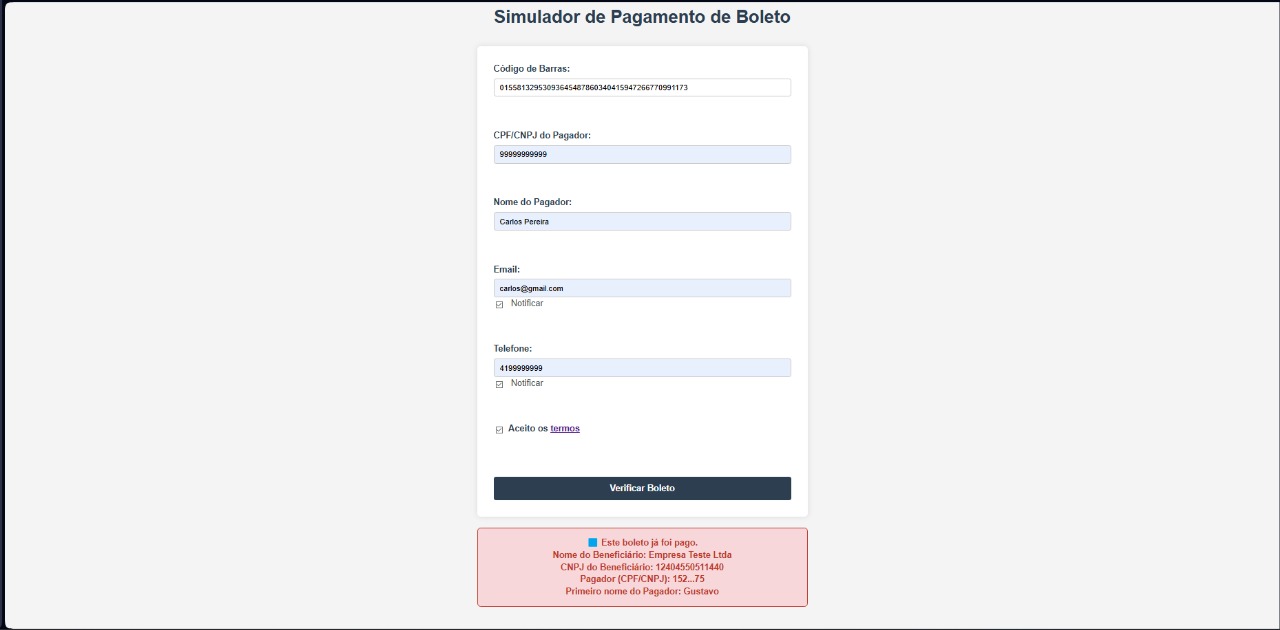


Na página “Simulador de Pagamento de Boleto” deverá ser informado o código do boleto, CNPJ ou CPF e nome do pagador, e-mail e telefone (sendo opcional receber notificações ou não). Após a inserção das informações, o usuário deve aceitar os termos de uso, e selecionar a opção “Verificar Boleto”. Após isso, podem ser retornadas ao usuário cinco tipos de mensagens:

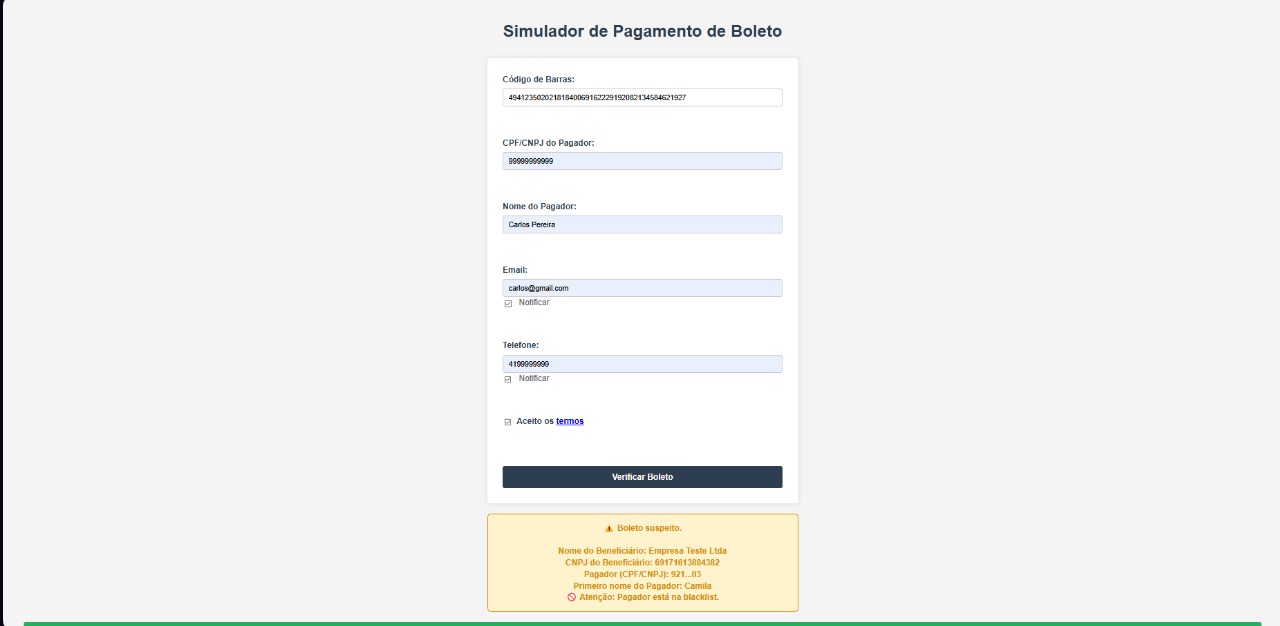
“Boleto não encontrado”, sendo um boleto que não existe:



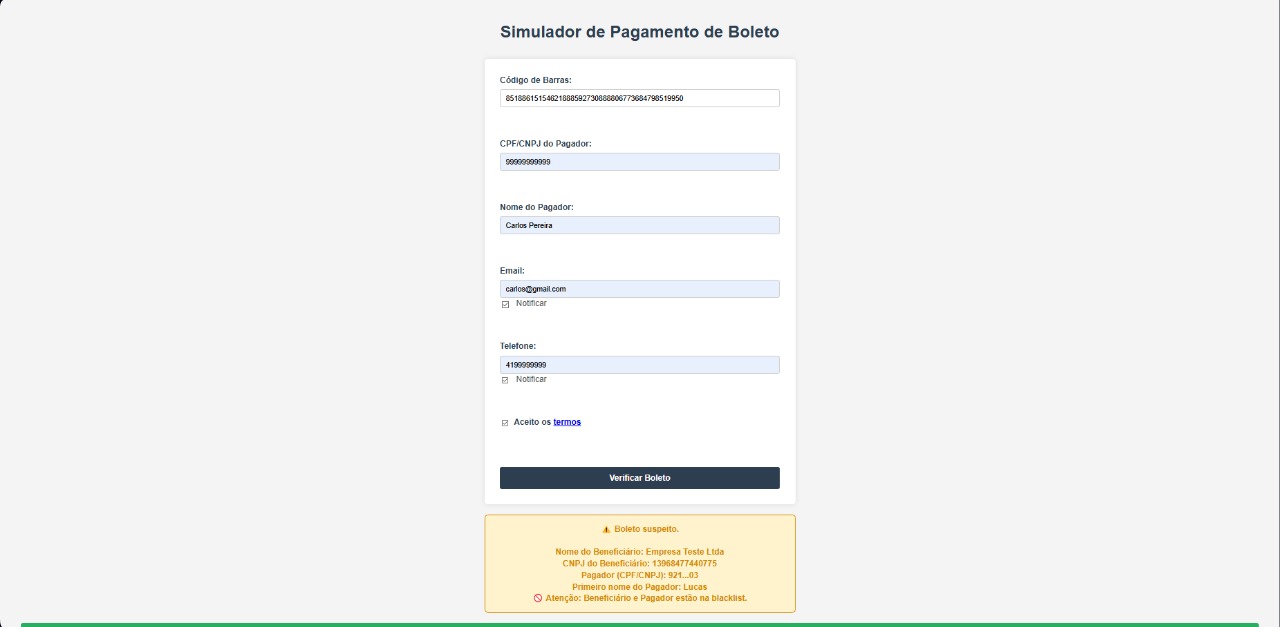
“Este boleto já foi pago”, para boletos que já foram pagos:



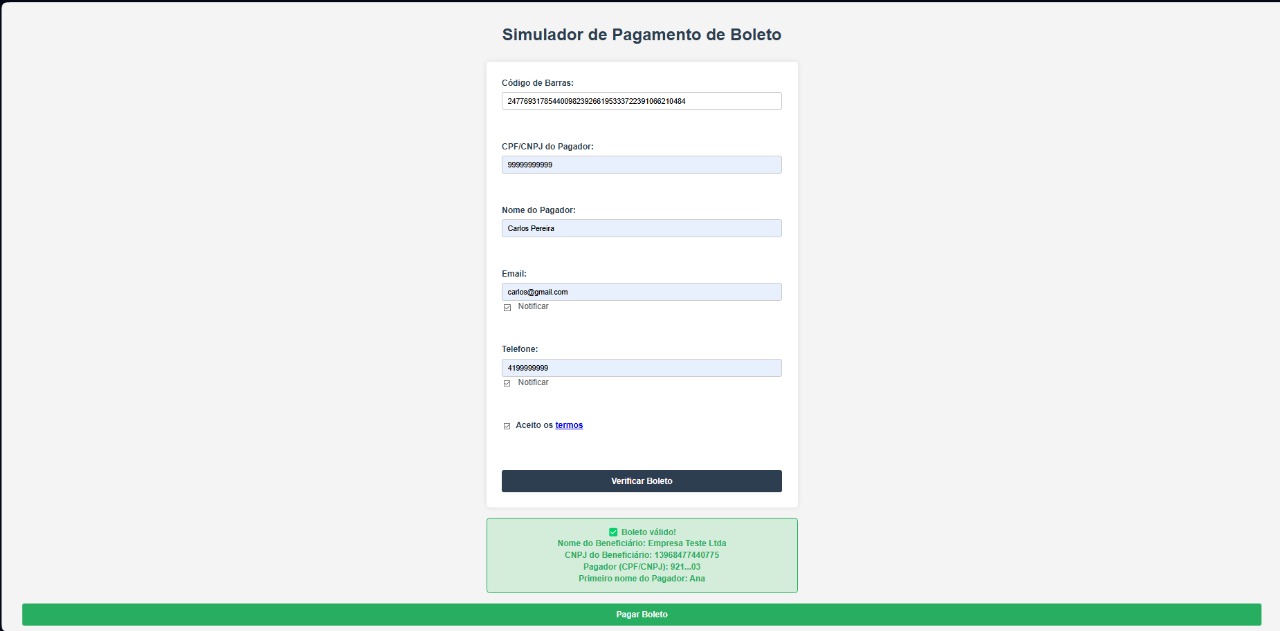
“Boleto suspeito”, se o beneficiário estiver na blacklist:



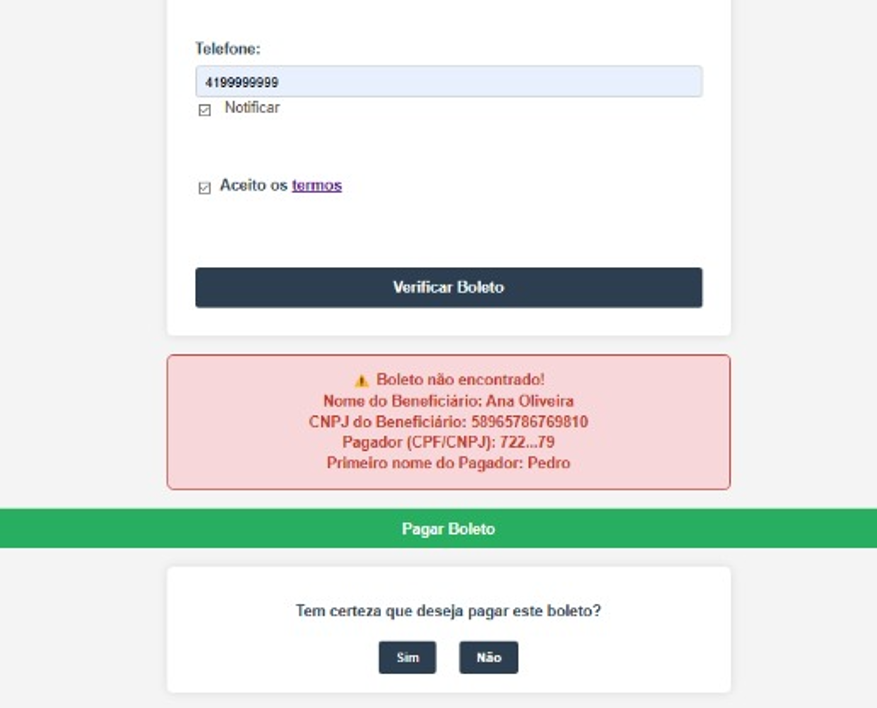
“Boleto suspeito”, se o pagador e o beneficiário estiverem na blacklist:



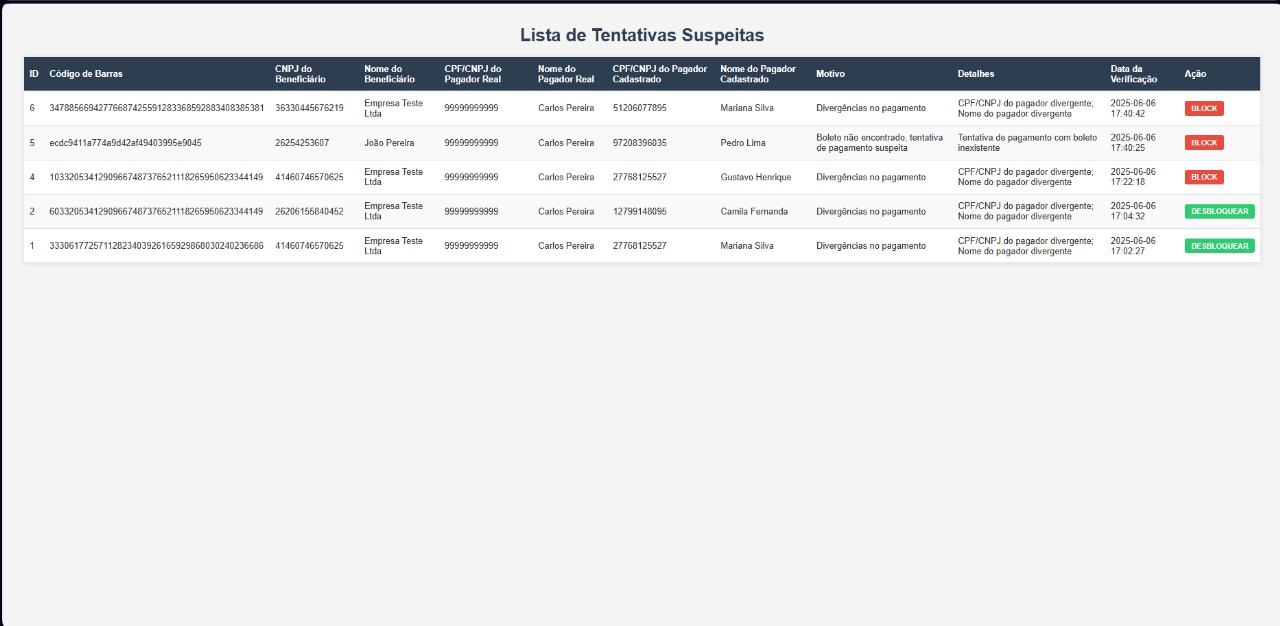
Por fim, se o boleto estiver válido irá retornar a mensagem “Boleto válido!”:



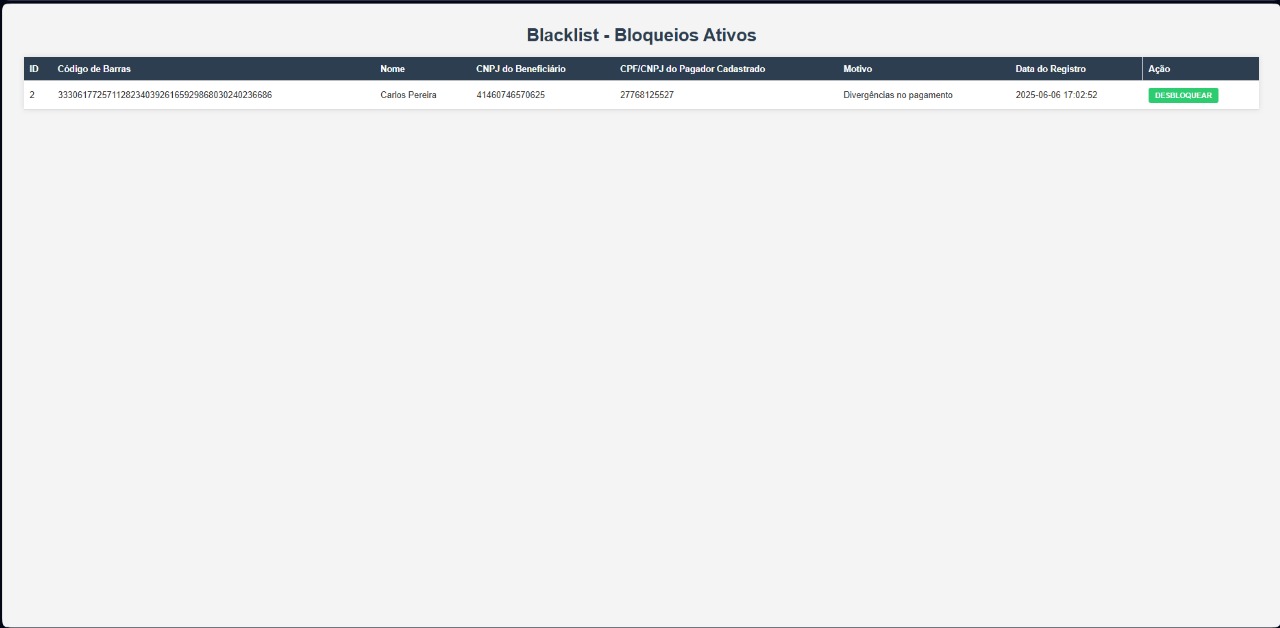
Independente da mensagem de retorno que o usuário receber, ele poderá realizar o pagamento do boleto, selecionando a opção “Pagar Boleto”:



Na tela “Lista de Tentativas Suspeitas” irão aparecer todos os boletos pagos que apresentam algum risco de acordo com os parâmetros estabelecidos, sendo possível inserir o bolero na blacklist, ou desbloqueá-lo. Um boleto é registrado como suspeito quando o pagador real é diferente do pagador esperado ou quando ocorre de o boleto não ser encontrado na base.



Na tela “Blacklist - Bloqueios Ativos” estarão listados todos os boletos que se encontram “bloqueados”, podendo ser realizada a sua liberação. As instituições também têm a opção de inserir CPFs ou CNPJs na blacklist manualmente, com base em denúncias.



**2.3.1 Como funciona o fluxo de verificação:**

Quando o cliente tenta pagar um boleto, o sistema segue as etapas abaixo:

1. Verifica se o código de barras existe na base de boletos:

Se sim, ocorre a autorização imediata do pagamento.

Se não, segue-se a validação.

1. Se o código de barras não for encontrado, verifica se o CPF/CNPJ do beneficiário já aparece em boletos válidos da base:

Se sim, apresenta uma mensagem de erro, podendo ser para boletos inválidos, vencidos, ou já registrados como pagos.

Se não, realiza a verificação adicional:

1. Verifica se há correspondência entre CPF/CNPJ e nome social:

Se não houver correspondência, o beneficiário não foi cadastrado.

Se houver correspondência o boleto ainda não é considerado suspeito, mas será autorizado com alerta: pagamento não recomendado.

Observação: Caso o pagamento ocorra, os dados são armazenados na tabela de tentativas suspeitas A partir daí, em novas transações, se o CPF/CNPJ estiver na blacklist, o sistema emitirá alerta explícito de risco, mas não bloqueará o pagamento.

2.4 Orientações de execução da Aplicação/Software

Para executar a aplicação, é necessário que o usuário tenha o Python 3.7 ou superior instalado, juntamente com as bibliotecas *flask, database, uuid, utils, secrets, re, difflib, datetime, random,* e *string.*

2.5 Repositório

https://github.com/joaogbreve/BoletoAPI

# 3 CONCLUSÃO

A construção do sistema verificador de boletos fraudulentos demonstrou, ao longo deste projeto, a relevância de soluções tecnológicas no enfrentamento de fraudes financeiras. Utilizando o framework *Flask* para a criação da API e o banco de dados *SQLite*, foi possível desenvolver um ambiente funcional para o registro, verificação e controle de boletos bancários. As bibliotecas da linguagem Python permitiram realizar validações rigorosas, gerar tokens de autenticação e identificar divergências em dados informados pelas instituições ou usuários.

A estrutura modular do sistema, com divisão clara entre os arquivos principais, utilitários, rotas e páginas HTML, favoreceu a organização e a manutenibilidade do código. As verificações aplicadas nos endpoints, como a validação de tokens, a análise de semelhança entre nomes e o controle de blacklist, mostraram-se eficazes na identificação de tentativas suspeitas e na prevenção de fraudes.

Conclui-se, portanto, que o sistema proposto alcançou seus objetivos ao oferecer uma solução prática e segura para a verificação de boletos bancários.

# REFERÊNCIAS

https://veja.abril.com.br/brasil/boletos-falsos-como-identificar-e-evitar-golpes

https://www.bcb.gov.br/meubc/faqs/p/vitima-pagou-um-boleto-e-caiu-em-um-golpe