SISTEMA FONECHECK

Sistema Inteligente de Busca e Gestão de Telefones Burger King & Popeyes

Versão: 2.0
Data de Criação: Dezembro 2024
Tecnologias: Python Flask, HTML5, CSS3, JavaScript ES6
Base de Dados: Microsoft Excel (.xlsx)
Desenvolvido por: Angelica Pedroso e Bruno Costa
Cliente: Zamp - Burger King e Popeyes

RESUMO EXECUTIVO

O Sistema FoneCheck é uma aplicação web desenvolvida especificamente para a gestão e busca inteligente de números de telefone das lojas Burger King e Popeyes. O sistema permite busca rápida por códigos de loja, exportação de dados, adição de novos números e registro completo de interações via WhatsApp. Desenvolvido com tecnologias modernas e interface responsiva, o sistema oferece controle total sobre o acesso aos telefones, garantindo auditoria completa e experiência de usuário otimizada.

ÍNDICE

1. VISÃO GERAL DO SISTEMA

1.1 Objetivo

O Sistema FoneCheck foi desenvolvido para centralizar e otimizar a gestão de números de telefone das lojas Burger King e Popeyes, oferecendo: • Busca rápida e inteligente por códigos de loja
• Interface moderna e responsiva
• Controle de acesso com auditoria completa
• Exportação de dados em múltiplos formatos
• Sistema de logs para rastreamento de interações
• Integração direta com WhatsApp

1.2 Escopo

O sistema abrange todas as lojas Burger King e Popeyes cadastradas na base de dados, incluindo mais de 9.000 registros com informações completas de contato. Suporta códigos de loja em diferentes formatos e oferece funcionalidades de gestão completa dos dados.

1.3 Benefícios

Eficiência Operacional: Redução significativa no tempo de busca de telefones
Controle de Acesso: Auditoria completa de quem acessa cada número
Interface Intuitiva: Design moderno e fácil de usar
Escalabilidade: Suporte para expansão da base de dados
Integração: Acesso direto ao WhatsApp com registro automático
Relatórios: Exportação de dados para análise

2. ARQUITETURA E TECNOLOGIAS

2.1 Stack Tecnológico

Componente	Tecnologia	Versão	Descrição
Backend	Python Flask	2.3+	Framework web para APIs
Frontend	HTML5/CSS3/JS	ES6	Interface responsiva
Base de Dados	Excel (.xlsx)	OpenPyXL	Armazenamento de dados
Servidor	Flask Dev Server	Local	Ambiente de desenvolvimento
Versionamento	Git	2.40+	Controle de versão
Deploy	GitHub	Cloud	Repositório remoto

2.2 Arquitetura do Sistema

O sistema segue uma arquitetura cliente-servidor com as seguintes camadas: **Camada de Apresentação:** Interface web responsiva desenvolvida em HTML5, CSS3 e JavaScript puro, com design moderno e componentes interativos. **Camada de Aplicação:** API REST desenvolvida em Flask (Python) que processa as requisições, valida dados e gerencia a lógica de negócio. **Camada de Dados:** Base de dados em formato Excel (.xlsx) com mais de 9.000 registros, gerenciada através da biblioteca OpenPyXL. **Camada de Logs:** Sistema de auditoria que registra todas as interações com os telefones, incluindo dados do usuário, motivo do contato e timestamp.

3. FUNCIONALIDADES PRINCIPAIS

3.1 Sistema de Busca Inteligente

Busca por Código de Loja: Sistema reconhece automaticamente códigos BK (Burger King) e PLK (Popeyes) com validação inteligente. **Códigos Suportados:** • Burger King: 15xxx, 16xxx, 17xxx, 18xxx, 19xxx, 20xxx, 21xxx, 22xxx, 23xxx, 24xxx, 25xxx, 26xxx, 27xxx, 28xxx, 29xxx, 30xxx, 31xxx, 32xxx
• Popeyes: PLK + 12xxx, 13xxx, 14xxx
Validação Automática: Sistema identifica automaticamente o tipo de loja baseado no código fornecido, eliminando erros de classificação.

3.2 Interface Responsiva

Design Moderno: Interface desenvolvida com tema Zamp (azul, laranja, vermelho) seguindo as diretrizes visuais da marca. **Responsividade:** Adaptação automática para dispositivos móveis, tablets e desktops com layout otimizado para cada tamanho de tela. **Componentes Interativos:** Cards animados, botões com efeitos hover, modais elegantes e notificações em tempo real. **Acessibilidade:** Interface desenvolvida seguindo boas práticas de UX/UI com contraste adequado e navegação intuitiva.

3.3 Sistema de Logs e Auditoria

Registro Completo: Todas as interações com telefones são registradas automaticamente, incluindo: • Nome do usuário
• Motivo da solicitação
• Telefone acessado
• Código da loja
• Tipo de loja (BK/PK)
• Data e hora
• Endereço IP
Modal de Captura: Interface modal que solicita informações obrigatórias antes de abrir o WhatsApp, garantindo registro completo. **Arquivo de Logs:** Dados salvos automaticamente em arquivo Excel (whatsapp_logs.xlsx) para análise posterior.

3.4 Funcionalidades de Exportação

Exportação Padrão: Exporta todos os telefones encontrados em formato Excel com dados organizados e formatados. **Exportação Separada:** Cria arquivo Excel com cada telefone em linha separada incluindo metadados adicionais para análise detalhada. **Adição de Novos Números:** Interface para adicionar novos telefones à base de dados com validação de dados e integração automática.

4. IMPLEMENTAÇÃO TÉCNICA

4.1 Backend - Flask Application

Código Principal do Backend:

```
# Estrutura principal do Flask App from flask import Flask,
render_template, request, jsonify import pandas as pd import re app =
Flask(__name__) # Endpoint principal de busca @app.route('/buscar',
methods=['POST']) def buscar_telefones(): dados = request.json codigo =
dados.get('codigo', '').strip() tipo_busca = dados.get('tipo', 'BK')
resultado = processar_busca(codigo, tipo_busca) return jsonify(resultado)
# Função de processamento de busca def processar_busca(codigo,
tipo_busca): df = pd.read_excel('incident.xlsx') telefones =
buscar_numeros_telefone_por_codigo(df, codigo, tipo_busca) return {
'sucesso': True, 'telefones': telefones, 'total': len(telefones),
'codigo': codigo, 'tipo_busca': tipo_busca }
```

4.2 Frontend - JavaScript

Código Principal do Frontend:

```
// Função principal de busca async function buscarTelefones() { const codigo = document.getElementById('codigoInput').value.trim(); const tipoSelecionado = document.querySelector('input[name="tipoBusca"]:checked'); if (!codigo || !tipoSelecionado) { mostrarNotificacao('Por favor, preencha todos os campos', 'error'); return; } mostrarLoading(true); try { const response = await fetch('/buscar', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ codigo: codigo, tipo: tipoSelecionado.value }) }); const data = await response.json(); if (data.sucesso) { telefonesData = data.telefones; tipoBuscaAtual = data.tipo_busca; codigoAtual = data.codigo; aplicarFiltros(); mostrarResultados(); } else { mostrarNotificacao(data.erro || 'Erro na busca', 'error'); } } catch (error) { mostrarNotificacao('Erro de conexão', 'error'); } finally { mostrarLoading(false); } }
```

4.3 Sistema de Logs

Código do Sistema de Logs:

```
# Endpoint para salvar logs do WhatsApp @app.route('/log-whatsapp', methods=['POST']) def log_whatsapp(): dados = request.json log_entry = { 'Data/Hora': dados.get('data_hora'), 'Nome': dados.get('nome'), 'Motivo': dados.get('motivo'), 'Telefone': dados.get('telefone'), 'Código Loja': dados.get('codigo_loja'), 'Tipo Loja': dados.get('tipo_loja'), 'IP': dados.get('ip') } # Salvar em Excel try: log_df = pd.read_excel('whatsapp_logs.xlsx') except FileNotFoundError: log_df = pd.DataFrame() log_df = pd.concat([log_df, pd.DataFrame([log_entry])], ignore_index=True) log_df.to_excel('whatsapp_logs.xlsx', index=False) return jsonify({'sucesso': True})
```

5. DOCUMENTAÇÃO DA API

5.1 Endpoints Disponíveis

Endpoint	Método	Descrição	Parâmetros
/buscar	POST	Buscar telefones por código	codigo, tipo
/exportar	POST	Exportar telefones encontrados	telefones, tipo
/exportar-separado	POST	Exportar telefones separados	telefones
/adicionar-numero	POST	Adicionar novo número	nome, telefone, tipo
/log-whatsapp	POST	Salvar log de WhatsApp	dados do log
/health	GET	Verificar saúde da API	nenhum

5.2 Exemplos de Uso da API

Buscar Telefones - Burger King:

```
POST /buscar
{
  "codigo": "23584",
  "tipo": "BK"
}
```

Resposta:

```
{
  "sucesso": true,
  "telefones": ["48988173588", "48984080236", "71982293685"],
  "total": 3,
  "codigo": "23584",
  "tipo_busca": "BK"
}
```

```
Buscar Telefones - Popeyes:
POST /buscar
{
  "codigo": "13232",
  "tipo": "PK"
}

Salvar Log WhatsApp:
POST /log-whatsapp
{
  "nome": "João Silva",
  "motivo": "Suporte Técnico",
  "telefone": "11999999999",
  "codigo_loja": "23584",
  "tipo_loja": "BK",
  "data_hora": "2024-12-22T10:30:00Z",
  "ip": "192.168.1.100"
}
```

6. DEPLOY E CONFIGURAÇÃO

6.1 Requisitos do Sistema

Python: Versão 3.8 ou superior
Dependências: Flask, Pandas, OpenPyXL
Sistema Operacional: Windows, Linux ou macOS
Memória: Mínimo 4GB RAM recomendado
Espaço em Disco: 100MB para aplicação + espaço para logs
Navegador: Chrome, Firefox, Safari ou Edge (versões recentes)

6.2 Processo de Instalação

```
1. Clonar Repositório:
git clone https://github.com/costC22/FoneCheck.git
cd FoneCheck

2. Instalar Dependências:
pip install -r requirements.txt

3. Configurar Base de Dados:
• Colocar arquivo incident.xlsx na raiz do projeto
• Verificar estrutura das colunas

4. Executar Aplicação:
python app.py

5. Acessar Sistema:
Abrir navegador em http://localhost:5000
```

6.3 Configurações Avançadas

```
Variáveis de Ambiente:
• PORT: Porta do servidor (padrão: 5000)
• DEBUG: Modo debug (padrão: False)

Arquivo de Configuração:
• incident.xlsx: Base de dados principal
• whatsapp_logs.xlsx: Logs automáticos
• requirements.txt: Dependências Python

Estrutura de Diretórios:
FoneCheck/
■■■ app.py (aplicação principal)
■■■ incident.xlsx (base de dados)
■■■ templates/ (HTML)
■■■ static/ (CSS, JS, imagens)
■■■ whatsapp_logs.xlsx (logs)
```

7. MANUTENÇÃO E SUPORTE

7.1 Monitoramento do Sistema

Logs da Aplicação: Verificar logs do Flask para erros e performance
Arquivo de Logs WhatsApp: Monitorar whatsapp_logs.xlsx para análise de uso
Base de Dados: Verificar integridade do arquivo incident.xlsx
Performance: Monitorar tempo de resposta das buscas

Indicadores de Saúde:
• Tempo de resposta < 2 segundos
• Taxa de sucesso > 99%
• Logs sendo gerados corretamente
• Interface responsiva funcionando

7.2 Estratégia de Backup

Arquivos Críticos:
• incident.xlsx: Base de dados principal
• whatsapp_logs.xlsx: Logs de auditoria
• Código fonte: Repositório Git

Frequência de Backup:
• Base de dados: Diário
• Logs: Semanal
• Código: A cada commit

Local de Armazenamento:
• Backup local + nuvem
• Versionamento Git
• Múltiplas cópias de segurança

7.3 Troubleshooting Comum

Problema	Causa Provável	Solução
Telefone não encontrado	Código incorreto	Verificar código da loja
Erro de conexão	Servidor offline	Reiniciar aplicação
Interface não carrega	Arquivos CSS/JS	Verificar pasta static
Logs não salvam	Permissão de escrita	Verificar permissões
Busca lenta	Base de dados grande	Otimizar consultas

7.4 Suporte e Contato

Desenvolvedores:
• Angelica Pedroso
• Bruno Costa

Cliente: Zamp - Burger King e Popeyes

Repositório: https://github.com/costC22/FoneCheck

Documentação: Este documento técnico contém todas as informações necessárias para manutenção e evolução do sistema.