QA Dashboard App - Documentação Técnica

Arquitetura do Sistema

Visão Geral

O QA Dashboard App é construído com uma arquitetura modular em Python, utilizando Streamlit para a interface web e múltiplas bibliotecas especializadas para extração e processamento de PDFs.

Componentes Principais

1. Módulo de Extração (`src/pdf_extractor.py`)

Responsabilidade: Extração de dados de arquivos PDF **Tecnologias**: PyPDF2, pdfplumber, tabula-py, Tesseract OCR

```
def extract_data_from_pdf(pdf_path):
    """
    Extrai dados de PDF usando múltiplas técnicas:
    1. PyPDF2 para texto simples
    2. pdfplumber para tabelas estruturadas
    3. tabula-py para tabelas complexas
    4. OCR como fallback para PDFs escaneados
    """
```

Fluxo de Extração: 1. Tentativa com pdfplumber (tabelas estruturadas) 2. Fallback para tabula-py (tabelas complexas) 3. Parsing de texto simples com regex 4. OCR como último recurso

2. Módulo de Processamento (`src/data_processor.py`)

Responsabilidade: Processamento e análise dos dados extraídos **Tecnologias**: pandas, numpy

```
def process_extracted_data(extracted_data):
    """
    Processa dados extraídos:
    1. Normalização de dados
    2. Cálculo de KPIs
    3. Estruturação para visualização
    """
```

KPIs Calculados: - Total de casos de teste - Casos passados/falhados - Percentual de execução - Percentual de sucesso

3. Interface Web (`dashboard.py`)

Responsabilidade: Interface de usuário e visualização **Tecnologias**: Streamlit, Plotly

```
def main():
    """
    Interface principal:
    1. Upload de arquivos
    2. Processamento em tempo real
    3. Visualização interativa
    4. Exportação de dados
    """
```

Componentes da Interface: - Sidebar para upload - Métricas em cards - Gráficos interativos (pizza, barras) - Tabela de dados - Botão de exportação CSV

4. Agendador (`scheduler.py`)

Responsabilidade: Processamento automático agendado **Tecnologias**: schedule, datetime

```
class QAScheduler:
    """
    Agendador para processamento automático:
    1. Monitoramento de pasta
    2. Processamento em lote
    3. Organização de arquivos
    """
```

5. Aplicativo Principal (`app.py`)

Responsabilidade: Orquestração e interface de linha de comando **Tecnologias**: argparse, subprocess

Fluxo de Dados

1. Upload Manual (Dashboard)

```
PDF Upload → Extração → Processamento → Visualização → Exportação
```

2. Processamento Automático (Scheduler)

```
Pasta Monitorada → Extração → Processamento → CSV → Arquivo Movido
```

Estrutura de Classes

PDFExtractor

```
class PDFExtractor:
    def extract_text_from_pdf(self, pdf_path)
    def extract_tables_from_pdf(self, pdf_path)
    def ocr_pdf(self, pdf_path)
    def extract_data_from_pdf(self, pdf_path)
```

DataProcessor

```
class DataProcessor:
    def process_status_data(self, tables)
    def calculate_kpis(self, df_status)
    def process_extracted_data(self, extracted_data)
```

QAScheduler

```
class QAScheduler:
    def __init__(self, input_folder, output_folder)
    def process_pdfs(self)
    def start_scheduler(self, schedule_time)
```

Algoritmos de Extração

1. Extração de Texto Simples

```
def extract_text_from_pdf(pdf_path):
    # Usa PyPDF2 para extrair texto página por página
    # Concatena todo o texto em uma string
    # Retorna texto bruto para processamento posterior
```

2. Extração de Tabelas Estruturadas

```
def extract_tables_from_pdf(pdf_path):
    # 1. Tenta pdfplumber.extract_tables()
    # 2. Fallback para tabula.read_pdf()
    # 3. Parsing manual com regex se necessário
    # 4. Retorna lista de tabelas como arrays 2D
```

3. Processamento OCR

```
def ocr_pdf(pdf_path):
    # 1. Converte PDF para imagens (pdf2image)
    # 2. Aplica OCR em cada imagem (pytesseract)
    # 3. Concatena texto extraído
    # 4. Remove arquivos temporários
```

4. Parsing de Dados QA

```
def parse_qa_data(text_data):
    # 1. Busca padrões de tabela com regex
    # 2. Identifica cabeçalhos (Status | Total)
    # 3. Extrai linhas de dados
    # 4. Estrutura em DataFrame pandas
```

Tratamento de Erros

Estratégia de Fallback

1. **Primeira tentativa**: pdfplumber (mais preciso)
2. **Segunda tentativa**: tabula-py (mais robusto)
3. **Terceira tentativa**: parsing manual de texto

4. **Última tentativa**: OCR (mais lento)

Validação de Dados

```
def validate_extracted_data(data):
    # Verifica se tabelas foram encontradas
    # Valida estrutura de dados esperada
    # Retorna status de validação
```

Log de Erros

```
def log_extraction_error(pdf_path, error):
    # Registra erros com timestamp
    # Inclui informações do arquivo
    # Facilita debugging
```

Otimizações de Performance

1. Cache de Resultados

- Evita reprocessamento de PDFs já analisados
- Usa hash do arquivo como chave

2. Processamento Assíncrono

- Upload não bloqueia interface
- Processamento em background

3. Lazy Loading

- Carrega bibliotecas apenas quando necessário
- Reduz tempo de inicialização

Configurações

Variáveis de Ambiente

```
JAVA_HOME=/path/to/java          # Para tabula-py
TESSDATA_PREFIX=/path/to/tessdata # Para OCR
QA_DASHBOARD_PORT=8501           # Porta do Streamlit
```

Configurações de OCR

```
OCR_CONFIG = {
    'lang': 'por+eng',           # Idiomas suportados
    'psm': 6,                    # Page segmentation mode
    'oem': 3                     # OCR Engine Mode
}
```

Configurações de Agendamento

```
SCHEDULER_CONFIG = {
    'default_time': '09:00',     # Horário padrão
    'check_interval': 60,        # Intervalo de verificação (segundos)
    'max_retries': 3             # Tentativas máximas por arquivo
}
```

Testes

Testes Unitários

```
def test_pdf_extraction():
    # Testa extração com PDF conhecido
    # Verifica estrutura de dados retornada
    # Valida KPIs calculados

def test_data_processing():
    # Testa processamento com dados mock
    # Verifica cálculos de KPIs
    # Valida formatação de saída
```

Testes de Integração

```
def test_full_pipeline():
    # Testa fluxo completo PDF → Dashboard
    # Verifica persistência de dados
    # Valida interface web
```

Testes de Performance

```
def test_large_pdf_processing():
    # Testa com PDFs grandes (>10MB)
    # Mede tempo de processamento
    # Verifica uso de memória
```

Deployment

Empacotamento com PyInstaller

```
pyinstaller --onefile --windowed \
    --add-data "src;src" \
    --hidden-import streamlit \
    app.py
```

Distribuição

1. **Executável standalone**: Para usuários finais
2. **Pacote Python**: Para desenvolvedores
3. **Container Docker**: Para deployment em servidor

Monitoramento

Métricas de Sistema

- Tempo de processamento por PDF
- Taxa de sucesso de extração
- Uso de memória e CPU

Logs de Aplicação

```
import logging

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('qa_dashboard.log'),
        logging.StreamHandler()
    ]
)
```

Extensibilidade

Adicionando Novos Formatos

```
def extract_from_excel(file_path):
    # Implementar extração de Excel
    # Seguir interface padrão de extração
    # Retornar dados no formato esperado
```

Novos Tipos de Visualização

```
def create_timeline_chart(data):
    # Implementar gráfico de timeline
    # Usar Plotly para consistência
    # Integrar com dashboard existente
```

Integração com APIs

```
def sync_with_jira(data):
    # Sincronizar dados com JIRA
    # Usar autenticação OAuth
    # Mapear campos automaticamente
```

Segurança

Validação de Entrada

- Verificação de tipo de arquivo

- Limite de tamanho de upload
- Sanitização de nomes de arquivo

Proteção de Dados

- Não armazenamento de dados sensíveis
- Limpeza automática de arquivos temporários
- Logs sem informações pessoais

Manutenção

Atualizações de Dependências

```
# Verificar vulnerabilidades
pip audit

# Atualizar dependências
pip install -r requirements.txt --upgrade
```

Backup e Recuperação

- Backup automático de dados processados
- Versionamento de configurações
- Procedimentos de recuperação documentados

Conclusão

A arquitetura modular do QA Dashboard App permite fácil manutenção, extensão e debugging. O uso de múltiplas técnicas de extração garante robustez, enquanto a interface web moderna proporciona excelente experiência do usuário.