Estrutura Simplificada do Projeto

📁 Estrutura de Arquivos Minimalista

```
conversor-declaracoes/
│
├── 📁 app/
│   ├── __init__.py
│   ├── main.py                    # Aplicação Flask principal
│   ├── converter.py               # Lógica de conversão
│   ├── validators.py              # Validações
│   └── config.py                  # Configurações simples
│
├── 📁 static/
│   ├── 📁 css/
│   │   └── style.css              # Estilos do formulário
│   │
│   ├── 📁 js/
│   │   └── app.js                 # JavaScript para interatividade
│   │
│   └── 📁 downloads/              # Arquivos gerados (gitignored)
│
├── 📁 templates/
│   └── index.html                 # Template único do formulário
│
├── 📁 uploads/                    # Arquivos enviados (gitignored)
```

```
│
├── requirements.txt
├── .env.example
├── .env                          # Gitignored
├── .gitignore
└── README.md
```

🎯 Descrição dos Arquivos

Backend

`app/main.py` (Aplicação Flask)

- Rotas principais:
  - `GET /` - Renderiza o formulário
  - `POST /upload` - Recebe arquivo e inicia conversão
  - `GET /status/<task_id>` - Verifica progresso da conversão
  - `GET /download/<filename>` - Faz download do arquivo gerado
- Gerenciamento de sessão simples
- Limpeza de arquivos temporários

`app/converter.py` (Lógica Principal)

- Classe `Converter`:
  - `process_file()` - Processa CSV/XLSX
  - `generate_txt()` - Gera arquivo TXT
  - `validate_and_transform()` - Valida e transforma dados
- Funções de transformação:
  - Datas → ddmmaaaa
  - Valores monetários → formatação correta
  - CPF/CNPJ → limpeza
  - Booleanos → 0/1

`app/validators.py` (Validações)

- Funções de validação individuais:
  - `validate_cpf_cnpj()`
  - `validate_date()`

    - `validate_cep()`
    - `validate_estado()`
    - `validate_numeric_field()`
- Função principal: `validate_row()` - valida linha completa

`app/config.py` (Configurações)

- Configurações da aplicação
- Limites de tamanho de arquivo
- Extensões permitidas
- Timeouts
- Caminhos de diretórios

Frontend

`templates/index.html` (Template Único)

Estrutura em **3 etapas visíveis no mesmo formulário**:

<!-- ETAPA 1: Upload e Configuração -->

- Área de upload (drag & drop ou botão)

- Campos de configuração:

  * Inscrição Municipal

  * Mês/Ano da Competência

  * Nome/Razão Social do Tomador

  * Código do Serviço

  * Dígito Verificador (checkbox)

  * Separador Decimal (radio)

- Botão "Converter"

<!-- ETAPA 2: Processamento -->

- Barra de progresso animada

- Mensagem de status dinâmica

- Indicador de etapa atual:

  * "Lendo arquivo..."

  * "Validando dados..."

  * "Gerando TXT..."

<!-- ETAPA 3: Resultado -->

- Resumo da conversão:

  * Total de registros processados

  * Registros com sucesso

  * Registros com erro

- Lista de erros (se houver)

- Botão "Baixar Arquivo TXT"

- Botão "Converter Novo Arquivo"

`static/css/style.css` (Estilos)

- Layout responsivo simples
- Estilos para as 3 etapas
- Animação da barra de progresso
- Feedback visual (cores para sucesso/erro/aviso)
- Cards para organizar informações

`static/js/app.js` (Interatividade)

- Controle de etapas (mostrar/ocultar)
- Upload via AJAX
- Polling para verificar status (requisições periódicas)

- Atualização da barra de progresso
- Exibição de resultados
- Download sob demanda
- Validação de formulário no frontend

🔄 Fluxo Simplificado

1. Usuário na Etapa 1 (Upload)

Usuário preenche formulário → Seleciona arquivo → Clica "Converter"

2. JavaScript envia via AJAX

// app.js

FormData → POST /upload → Recebe task_id

3. Backend processa (main.py)

Salva arquivo temporário

Inicia conversão (síncrona ou assíncrona simples)

Retorna task_id

4. Frontend mostra Etapa 2 (Processing)

// Polling a cada 1 segundo

setInterval(() => {

fetch(`/status/${task_id}`)

.then(response => response.json())

.then(data => {

// Atualiza barra de progresso

// Atualiza mensagem de status

```
        if (data.status === 'completed') {
            // Mostra Etapa 3
        }
    });
}, 1000);
```

5. Backend retorna status

```
# /status/<task_id>
{
    "status": "processing",  # ou "completed" ou "error"
    "progress": 75,          # 0-100
    "message": "Validando dados...",
    "current_row": 150,
    "total_rows": 200
}
```

6. Frontend mostra Etapa 3 (Result)

```
// Exibe resumo
// Habilita botão de download
// Mostra erros se houver
```

7. Usuário clica "Baixar Arquivo TXT"

```
// Download sob demanda
window.location.href = `/download/${filename}`;
```

📊 Exemplo de Implementação das Etapas

HTML - Estrutura das Etapas

```
<div class="container">
    <h1>Conversor de Declarações de Serviços</h1>

    <!-- ETAPA 1 -->
    <div id="step-1" class="step show">
        <form id="upload-form">
            <div class="upload-area">
                <input type="file" id="file-input" accept=".csv,.xlsx">
                <label for="file-input">
                    Clique ou arraste o arquivo aqui
                </label>
            </div>

            <div class="form-fields">
                <input type="text" name="inscricao_municipal" placeholder="Inscrição Municipal"
required>
                <input type="number" name="mes" placeholder="Mês (1-12)" min="1" max="12"
required>
                <input type="number" name="ano" placeholder="Ano" required>
                <!-- ... outros campos ... -->
            </div>

            <button type="submit">Converter</button>
        </form>
    </div>
```

```
<!-- ETAPA 2 -->
<div id="step-2" class="step hide">
    <div class="progress-container">
        <div class="progress-bar">
            <div id="progress-fill" style="width: 0%"></div>
        </div>
        <p id="status-message">Iniciando conversão...</p>
        <p id="progress-text">0%</p>
    </div>
</div>

<!-- ETAPA 3 -->
<div id="step-3" class="step hide">
    <div class="result-summary">
        <h2>Conversão Concluída!</h2>
        <div class="stats">
            <p>Total de registros: <span id="total-records">0</span></p>
            <p>Processados com sucesso: <span id="success-records">0</span></p>
            <p>Com erros: <span id="error-records">0</span></p>
        </div>

        <div id="errors-list" class="errors hide">
            <!-- Erros listados aqui -->
        </div>
```

```
            <button id="download-btn" class="btn-primary">
                📥 Baixar Arquivo TXT
            </button>

            <button id="new-conversion-btn" class="btn-secondary">
                🔄 Nova Conversão
            </button>
        </div>
    </div>
</div>
```

🎨 CSS - Gerenciamento de Etapas

```
.step {
    transition: opacity 0.3s ease;
}

.step.show {
    display: block;
    opacity: 1;
}

.step.hide {
    display: none;
    opacity: 0;
}
```

```
.progress-bar {
    width: 100%;
    height: 30px;
    background: #e0e0e0;
    border-radius: 15px;
    overflow: hidden;
}

#progress-fill {
    height: 100%;
    background: linear-gradient(90deg, #4CAF50, #45a049);
    transition: width 0.3s ease;
}
```

⚙️ JavaScript - Controle das Etapas

```
// Controle de navegação entre etapas
function showStep(stepNumber) {
    document.querySelectorAll('.step').forEach(step => {
        step.classList.add('hide');
        step.classList.remove('show');
    });
    document.getElementById(`step-${stepNumber}`).classList.add('show');
    document.getElementById(`step-${stepNumber}`).classList.remove('hide');
}
```

```
// Submit do formulário
document.getElementById('upload-form').addEventListener('submit', async (e) => {
    e.preventDefault();

    const formData = new FormData(e.target);

    // Vai para etapa 2
    showStep(2);

    // Envia arquivo
    const response = await fetch('/upload', {
        method: 'POST',
        body: formData
    });

    const data = await response.json();
    const taskId = data.task_id;

    // Inicia polling
    checkStatus(taskId);
});

// Verifica status da conversão
function checkStatus(taskId) {
    const interval = setInterval(async () => {
```

```
        const response = await fetch(`/status/${taskId}`);
        const data = await response.json();

        // Atualiza barra
        document.getElementById('progress-fill').style.width = data.progress + '%';
        document.getElementById('status-message').textContent = data.message;
        document.getElementById('progress-text').textContent = data.progress + '%';

        if (data.status === 'completed') {
            clearInterval(interval);
            showResults(data);
            showStep(3);
        }

        if (data.status === 'error') {
            clearInterval(interval);
            showError(data.error);
        }
    }, 1000);
}

// Exibe resultados
function showResults(data) {
    document.getElementById('total-records').textContent = data.total;
    document.getElementById('success-records').textContent = data.success;
    document.getElementById('error-records').textContent = data.errors;
```

```
    // Configura download
    document.getElementById('download-btn').onclick = () => {
        window.location.href = `/download/${data.filename}`;
    };

    // Nova conversão
    document.getElementById('new-conversion-btn').onclick = () => {
        showStep(1);
        document.getElementById('upload-form').reset();
    };
}
```

Python - Estrutura Simplificada

app/main.py

```
from flask import Flask, render_template, request, jsonify, send_file
import os
import uuid

app = Flask(__name__)

# Dicionário simples para armazenar status das conversões
conversions = {}

@app.route('/')
```

```
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload():
    # Gera ID único
    task_id = str(uuid.uuid4())

    # Salva arquivo
    file = request.files['file']
    # ... salva temporariamente

    # Inicia conversão (pode ser thread ou celery)
    conversions[task_id] = {'status': 'processing', 'progress': 0}

    # Processa (simplificado - deveria ser assíncrono)
    process_conversion(task_id, file, request.form)

    return jsonify({'task_id': task_id})

@app.route('/status/<task_id>')
def status(task_id):
    return jsonify(conversions.get(task_id, {}))

@app.route('/download/<filename>')
def download(filename):
```

```
    return send_file(
        f'static/downloads/{filename}',
        as_attachment=True,
        download_name=filename
    )
```

📦 Dependências Mínimas

`requirements.txt`

```
Flask==3.0.0
pandas==2.1.3
openpyxl==3.1.2
python-dotenv==1.0.0
```

✅ Vantagens desta Estrutura Simplificada

1. **Poucos arquivos** - Fácil de entender e navegar
2. **Tudo em um lugar** - Formulário único com 3 etapas
3. **Sem reload** - Experiência fluida com AJAX
4. **Download sob demanda** - Usuário controla quando baixar
5. **Feedback visual** - Barra de progresso em tempo real
6. **Fácil manutenção** - Código direto e objetivo
7. **Experimental** - Perfeito para testar conceitos

Esta estrutura é ideal para um projeto experimental e pode crescer organicamente conforme necessário!