import json

import re

import hashlib

import logging

from datetime import datetime, timedelta

from typing import Dict, List, Optional, Tuple, Any

from decimal import Decimal

import pandas as pd

import numpy as np

from dataclasses import dataclass

# Configuração de logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

@dataclass

class ResultadoAnalise:

    """Classe para estruturar resultados de análise"""

    tipo: str

    score\_confianca: float

    alertas: List[Dict]

    recomendacoes: List[Dict]

    inconsistencias: List[Dict]

    oportunidades: List[Dict]

    detalhes: Dict

    tempo\_processamento: float

class AgenteIAService:

    """Serviço principal do Agente de IA Tributário"""

    def \_\_init\_\_(self):

        self.versao\_algoritmo = "2.0.0"

        self.modulo\_pln = ModuloPLN()

        self.modulo\_raciocinio = ModuloRaciocinio()

        self.modulo\_base\_conhecimento = ModuloBaseConhecimento()

        self.modulo\_insights = ModuloInsights()

        self.modulo\_validacao = ModuloValidacao()

    def processar\_documento\_fiscal(self, dados\_documento: Dict) -> Dict:

        """Processa um documento fiscal completo"""

        inicio = datetime.now()

        try:

            # 1. Validação inicial

            validacao = self.modulo\_validacao.validar\_documento(dados\_documento)

            if not validacao['valido']:

                return {

                    'sucesso': False,

                    'erro': 'Documento inválido',

                    'detalhes': validacao['erros']

                }

            # 2. Extração e estruturação de dados

            dados\_estruturados = self.\_extrair\_dados\_documento(dados\_documento)

            # 3. Análises principais

            resultados\_analises = {}

            # Análise de conformidade

            resultados\_analises['conformidade'] = self.analisar\_conformidade(dados\_estruturados)

            # Análise de otimização

            resultados\_analises['otimizacao'] = self.analisar\_otimizacao(dados\_estruturados)

            # Análise de auditoria

            resultados\_analises['auditoria'] = self.analisar\_auditoria(dados\_estruturados)

            # Análise de riscos

            resultados\_analises['riscos'] = self.analisar\_riscos(dados\_estruturados)

            # 4. Geração de insights consolidados

            insights = self.modulo\_insights.gerar\_insights\_consolidados(resultados\_analises)

            # 5. Cálculo de métricas

            tempo\_total = (datetime.now() - inicio).total\_seconds()

            return {

                'sucesso': True,

                'documento\_id': dados\_estruturados.get('id'),

                'hash\_documento': self.\_calcular\_hash\_documento(dados\_documento),

                'analises': resultados\_analises,

                'insights': insights,

                'tempo\_processamento': tempo\_total,

                'versao\_algoritmo': self.versao\_algoritmo,

                'data\_processamento': datetime.now().isoformat()

            }

        except Exception as e:

            logger.error(f"Erro no processamento do documento: {str(e)}")

            return {

                'sucesso': False,

                'erro': str(e),

                'tempo\_processamento': (datetime.now() - inicio).total\_seconds()

            }

    def analisar\_conformidade(self, dados\_documento: Dict) -> ResultadoAnalise:

        """Realiza análise de conformidade fiscal"""

        inicio = datetime.now()

        alertas = []

        inconsistencias = []

        recomendacoes = []

        score\_confianca = 0.0

        try:

            # Verificação de campos obrigatórios

            campos\_obrigatorios = self.\_verificar\_campos\_obrigatorios(dados\_documento)

            if campos\_obrigatorios['problemas']:

                alertas.extend(campos\_obrigatorios['problemas'])

            # Validação de alíquotas

            validacao\_aliquotas = self.\_validar\_aliquotas(dados\_documento)

            if validacao\_aliquotas['inconsistencias']:

                inconsistencias.extend(validacao\_aliquotas['inconsistencias'])

            # Verificação de CFOP

            validacao\_cfop = self.\_validar\_cfop(dados\_documento)

            if validacao\_cfop['problemas']:

                alertas.extend(validacao\_cfop['problemas'])

            # Validação de NCM

            validacao\_ncm = self.\_validar\_ncm(dados\_documento)

            if validacao\_ncm['problemas']:

                alertas.extend(validacao\_ncm['problemas'])

            # Verificação de cálculos tributários

            validacao\_calculos = self.\_validar\_calculos\_tributarios(dados\_documento)

            if validacao\_calculos['inconsistencias']:

                inconsistencias.extend(validacao\_calculos['inconsistencias'])

            # Cálculo do score de conformidade

            total\_verificacoes = 5

            verificacoes\_ok = sum([

                1 if not campos\_obrigatorios['problemas'] else 0,

                1 if not validacao\_aliquotas['inconsistencias'] else 0,

                1 if not validacao\_cfop['problemas'] else 0,

                1 if not validacao\_ncm['problemas'] else 0,

                1 if not validacao\_calculos['inconsistencias'] else 0

            ])

            score\_confianca = verificacoes\_ok / total\_verificacoes

            # Geração de recomendações

            if alertas or inconsistencias:

                recomendacoes = self.\_gerar\_recomendacoes\_conformidade(alertas, inconsistencias)

            tempo\_processamento = (datetime.now() - inicio).total\_seconds()

            return ResultadoAnalise(

                tipo="conformidade",

                score\_confianca=score\_confianca,

                alertas=alertas,

                recomendacoes=recomendacoes,

                inconsistencias=inconsistencias,

                oportunidades=[],

                detalhes={

                    'total\_verificacoes': total\_verificacoes,

                    'verificacoes\_ok': verificacoes\_ok,

                    'status\_geral': 'CONFORME' if score\_confianca >= 0.8 else 'NAO\_CONFORME'

                },

                tempo\_processamento=tempo\_processamento

            )

        except Exception as e:

            logger.error(f"Erro na análise de conformidade: {str(e)}")

            return ResultadoAnalise(

                tipo="conformidade",

                score\_confianca=0.0,

                alertas=[{'tipo': 'erro', 'mensagem': f'Erro na análise: {str(e)}'}],

                recomendacoes=[],

                inconsistencias=[],

                oportunidades=[],

                detalhes={'erro': str(e)},

                tempo\_processamento=(datetime.now() - inicio).total\_seconds()

            )

    def analisar\_otimizacao(self, dados\_documento: Dict) -> ResultadoAnalise:

        """Realiza análise de oportunidades de otimização tributária"""

        inicio = datetime.now()

        oportunidades = []

        recomendacoes = []

        score\_confianca = 0.0

        try:

            # Análise de regime tributário

            analise\_regime = self.\_analisar\_regime\_tributario(dados\_documento)

            if analise\_regime['oportunidades']:

                oportunidades.extend(analise\_regime['oportunidades'])

            # Análise de benefícios fiscais

            analise\_beneficios = self.\_analisar\_beneficios\_fiscais(dados\_documento)

            if analise\_beneficios['oportunidades']:

                oportunidades.extend(analise\_beneficios['oportunidades'])

            # Análise de planejamento tributário

            analise\_planejamento = self.\_analisar\_planejamento\_tributario(dados\_documento)

            if analise\_planejamento['oportunidades']:

                oportunidades.extend(analise\_planejamento['oportunidades'])

            # Cálculo de economia potencial

            economia\_total = sum([op.get('economia\_estimada', 0) for op in oportunidades])

            # Score baseado no potencial de economia

            valor\_total = float(dados\_documento.get('valor\_total', 0))

            if valor\_total > 0:

                score\_confianca = min(economia\_total / valor\_total, 1.0)

            # Geração de recomendações

            if oportunidades:

                recomendacoes = self.\_gerar\_recomendacoes\_otimizacao(oportunidades)

            tempo\_processamento = (datetime.now() - inicio).total\_seconds()

            return ResultadoAnalise(

                tipo="otimizacao",

                score\_confianca=score\_confianca,

                alertas=[],

                recomendacoes=recomendacoes,

                inconsistencias=[],

                oportunidades=oportunidades,

                detalhes={

                    'economia\_total\_estimada': economia\_total,

                    'percentual\_economia': (economia\_total / valor\_total \* 100) if valor\_total > 0 else 0,

                    'total\_oportunidades': len(oportunidades)

                },

                tempo\_processamento=tempo\_processamento

            )

        except Exception as e:

            logger.error(f"Erro na análise de otimização: {str(e)}")

            return ResultadoAnalise(

                tipo="otimizacao",

                score\_confianca=0.0,

                alertas=[{'tipo': 'erro', 'mensagem': f'Erro na análise: {str(e)}'}],

                recomendacoes=[],

                inconsistencias=[],

                oportunidades=[],

                detalhes={'erro': str(e)},

                tempo\_processamento=(datetime.now() - inicio).total\_seconds()

            )

    def analisar\_auditoria(self, dados\_documento: Dict) -> ResultadoAnalise:

        """Realiza análise de auditoria fiscal"""

        inicio = datetime.now()

        alertas = []

        inconsistencias = []

        recomendacoes = []

        score\_confianca = 0.0

        try:

            # Detecção de padrões suspeitos

            padroes\_suspeitos = self.\_detectar\_padroes\_suspeitos(dados\_documento)

            if padroes\_suspeitos['alertas']:

                alertas.extend(padroes\_suspeitos['alertas'])

            # Análise de consistência temporal

            consistencia\_temporal = self.\_analisar\_consistencia\_temporal(dados\_documento)

            if consistencia\_temporal['inconsistencias']:

                inconsistencias.extend(consistencia\_temporal['inconsistencias'])

            # Verificação de relacionamentos entre documentos

            relacionamentos = self.\_verificar\_relacionamentos\_documentos(dados\_documento)

            if relacionamentos['alertas']:

                alertas.extend(relacionamentos['alertas'])

            # Análise de risco fiscal

            risco\_fiscal = self.\_calcular\_risco\_fiscal(dados\_documento)

            # Score baseado no nível de risco

            score\_confianca = 1.0 - risco\_fiscal['nivel\_risco']

            # Geração de recomendações

            if alertas or inconsistencias:

                recomendacoes = self.\_gerar\_recomendacoes\_auditoria(alertas, inconsistencias)

            tempo\_processamento = (datetime.now() - inicio).total\_seconds()

            return ResultadoAnalise(

                tipo="auditoria",

                score\_confianca=score\_confianca,

                alertas=alertas,

                recomendacoes=recomendacoes,

                inconsistencias=inconsistencias,

                oportunidades=[],

                detalhes={

                    'nivel\_risco': risco\_fiscal['nivel\_risco'],

                    'fatores\_risco': risco\_fiscal['fatores'],

                    'total\_alertas': len(alertas),

                    'total\_inconsistencias': len(inconsistencias)

                },

                tempo\_processamento=tempo\_processamento

            )

        except Exception as e:

            logger.error(f"Erro na análise de auditoria: {str(e)}")

            return ResultadoAnalise(

                tipo="auditoria",

                score\_confianca=0.0,

                alertas=[{'tipo': 'erro', 'mensagem': f'Erro na análise: {str(e)}'}],

                recomendacoes=[],

                inconsistencias=[],

                oportunidades=[],

                detalhes={'erro': str(e)},

                tempo\_processamento=(datetime.now() - inicio).total\_seconds()

            )

    def analisar\_riscos(self, dados\_documento: Dict) -> ResultadoAnalise:

        """Realiza análise de riscos fiscais"""

        inicio = datetime.now()

        alertas = []

        recomendacoes = []

        score\_confianca = 0.0

        try:

            # Análise de risco de autuação

            risco\_autuacao = self.\_calcular\_risco\_autuacao(dados\_documento)

            if risco\_autuacao['nivel'] > 0.5:

                alertas.append({

                    'tipo': 'risco\_autuacao',

                    'severidade': 'ALTA' if risco\_autuacao['nivel'] > 0.8 else 'MEDIA',

                    'mensagem': f"Risco de autuação: {risco\_autuacao['nivel']:.2%}",

                    'fatores': risco\_autuacao['fatores']

                })

            # Análise de risco de multa

            risco\_multa = self.\_calcular\_risco\_multa(dados\_documento)

            if risco\_multa['probabilidade'] > 0.3:

                alertas.append({

                    'tipo': 'risco\_multa',

                    'severidade': 'ALTA' if risco\_multa['probabilidade'] > 0.7 else 'MEDIA',

                    'mensagem': f"Risco de multa: {risco\_multa['probabilidade']:.2%}",

                    'valor\_estimado': risco\_multa['valor\_estimado']

                })

            # Análise de risco de perda de benefícios

            risco\_beneficios = self.\_calcular\_risco\_perda\_beneficios(dados\_documento)

            if risco\_beneficios['nivel'] > 0.4:

                alertas.append({

                    'tipo': 'risco\_beneficios',

                    'severidade': 'MEDIA',

                    'mensagem': f"Risco de perda de benefícios: {risco\_beneficios['nivel']:.2%}",

                    'beneficios\_em\_risco': risco\_beneficios['beneficios']

                })

            # Score geral de risco (invertido - quanto menor o risco, maior o score)

            risco\_total = max(risco\_autuacao['nivel'], risco\_multa['probabilidade'], risco\_beneficios['nivel'])

            score\_confianca = 1.0 - risco\_total

            # Geração de recomendações

            if alertas:

                recomendacoes = self.\_gerar\_recomendacoes\_risco(alertas)

            tempo\_processamento = (datetime.now() - inicio).total\_seconds()

            return ResultadoAnalise(

                tipo="riscos",

                score\_confianca=score\_confianca,

                alertas=alertas,

                recomendacoes=recomendacoes,

                inconsistencias=[],

                oportunidades=[],

                detalhes={

                    'risco\_total': risco\_total,

                    'risco\_autuacao': risco\_autuacao['nivel'],

                    'risco\_multa': risco\_multa['probabilidade'],

                    'risco\_beneficios': risco\_beneficios['nivel'],

                    'classificacao\_risco': self.\_classificar\_nivel\_risco(risco\_total)

                },

                tempo\_processamento=tempo\_processamento

            )

        except Exception as e:

            logger.error(f"Erro na análise de riscos: {str(e)}")

            return ResultadoAnalise(

                tipo="riscos",

                score\_confianca=0.0,

                alertas=[{'tipo': 'erro', 'mensagem': f'Erro na análise: {str(e)}'}

(Content truncated due to size limit. Use line ranges to read in chunks)