IRCC PII Detection System - Implementation Specification

Executive Summary

This document provides complete technical specifications for implementing an enterprise-grade PII detection system for Immigration, Refugees and Citizenship Canada (IRCC). The system must handle sensitive immigration documents with complete data sovereignty, multilingual support, and government-level security compliance.

Table of Contents

1. System Overview
2. Technical Requirements
3. Architecture Specification
4. Implementation Roadmap
5. Core Components
6. Security Implementation
7. Testing Strategy
8. Deployment Guide
9. Performance Benchmarks
10. Compliance Framework

System Overview

Purpose

Develop a comprehensive PII detection and anonymization system specifically for IRCC that:

- Processes immigration documents (applications, permits, correspondence)
- Detects Canadian and international PII with 95%+ accuracy
- Maintains complete data sovereignty within Canada
- Supports English, French, and major immigrant languages
- Provides real-time processing for officer workflows
- Generates comprehensive audit trails for compliance

Key Stakeholders

- **Primary Users**: IRCC Immigration Officers, Case Processing Agents

- **Secondary Users**: Privacy Officers, Compliance Teams, System Administrators
- **Regulatory Bodies**: Privacy Commissioner of Canada, Treasury Board Secretariat

Success Criteria

- **Accuracy**: 95%+ PII detection accuracy across document types
- **Performance**: <500ms processing time per standard document
- **Compliance**: 100% audit trail coverage, zero compliance violations
- **Security**: Meets Protected A/B classification requirements
- **Availability**: 99.9% uptime during business hours
- **Scalability**: Handle 10,000+ documents per day

Technical Requirements

Functional Requirements

Core PII Detection

1. **Immigration-Specific PII Types**
   - Unique Client Identifier (UCI): `\b\d{4}[-\s]?\d{4}\b`
   - Immigration File Numbers: `\b[A-Z]{1,2}\d{6,10}\b`
   - Work Permit Numbers: `\bWP\d{8,10}\b`
   - Study Permit Numbers: `\bSP\d{8,10}\b`
   - Temporary Resident Visa: `\bTRV\d{8,10}\b`
   - Permanent Resident Card: `\bPR\d{8}\b`
2. **Canadian Government PII**
   - Social Insurance Number with validation
   - Canadian Passport Numbers: `\b[A-Z]{2}\d{6,8}\b`
   - Health Card Numbers (provincial variations)
   - Canadian Postal Codes with validation
   - Provincial Driver's License patterns
3. **International PII Support**
   - Foreign passport numbers (various formats)
   - International phone numbers
   - Foreign addresses and postal codes
   - Names in various scripts (Latin, Arabic, Chinese, etc.)
4. **Document Type Classification**
   - Immigration applications (PR, citizenship, work permits)
   - Supporting documents (passports, birth certificates)
   - Correspondence (letters, emails, case notes)
   - Decision letters and certificates

Language Support

1. **Official Languages**: English and French (full support)
2. **Immigrant Languages**: Spanish, Arabic, Mandarin, Hindi, Tagalog, Punjabi
3. **Auto-detection**: Automatic language identification
4. **Mixed-language**: Handle documents with multiple languages

Processing Capabilities

1. **Document Formats**: PDF, DOC/DOCX, TXT, images (OCR), emails
2. **Batch Processing**: Handle large document sets efficiently
3. **Real-time Processing**: Interactive officer workflows
4. **Version Control**: Track document processing history

Non-Functional Requirements

Security Requirements

1. **Data Sovereignty**: All processing within Canadian borders
2. **Encryption**: AES-256 for data at rest, TLS 1.3 for transit
3. **Access Control**: Role-based access with MFA
4. **Audit Logging**: Complete activity logs for compliance
5. **Network Isolation**: Air-gapped deployment capability
6. **Classification Support**: Unclassified, Protected A, Protected B

Performance Requirements

1. **Processing Speed**: <500ms for standard documents (<10KB)
2. **Throughput**: 100+ documents per minute sustained
3. **Concurrent Users**: Support 200+ simultaneous officers
4. **Response Time**: <2s for web interface interactions
5. **Batch Processing**: Handle 10,000 documents overnight

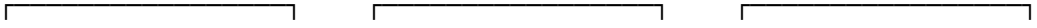
Reliability Requirements

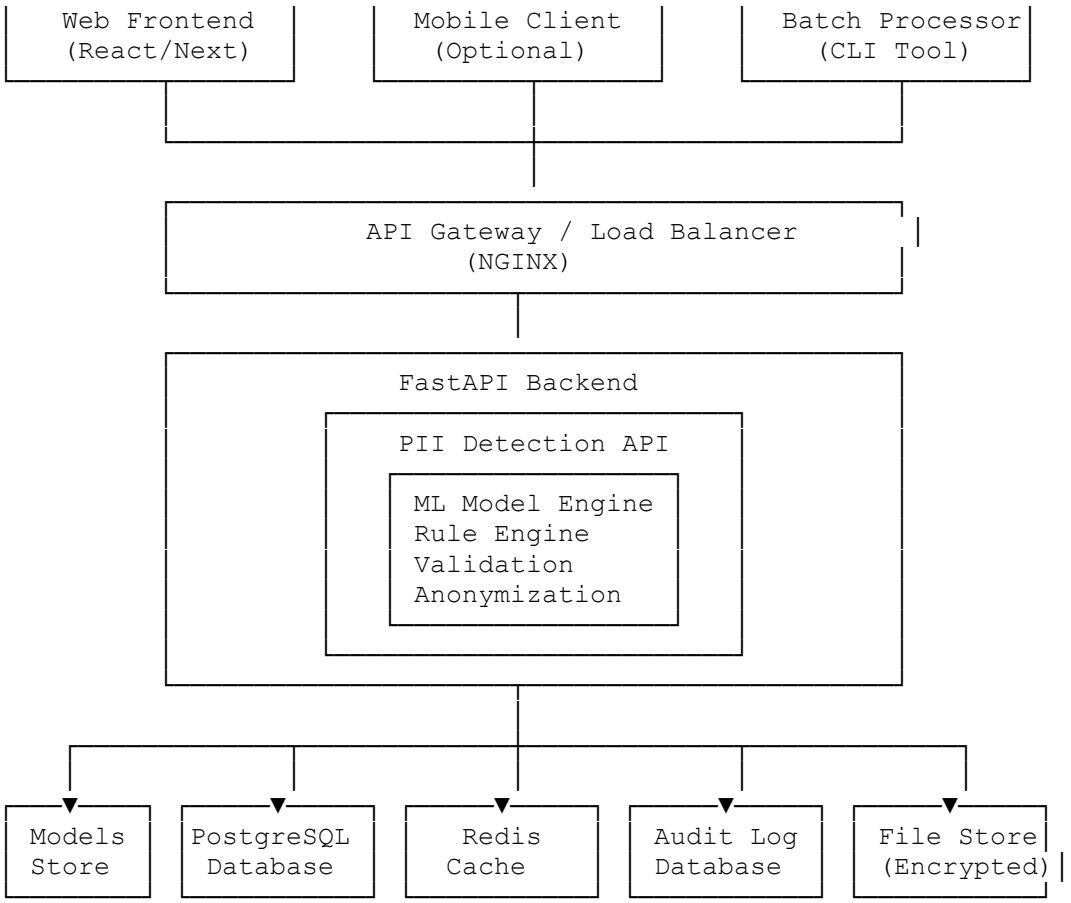
1. **Availability**: 99.9% uptime during business hours (7 AM - 7 PM ET)
2. **Recovery Time**: <4 hours for system restoration
3. **Data Backup**: Real-time replication, daily backups
4. **Error Handling**: Graceful degradation, no data loss
5. **Monitoring**: Real-time system health and performance metrics

---

Architecture Specification

High-Level Architecture





Component Architecture

1. FastAPI Backend Structure

```
app/
├── api/
│   ├── v1/
│   │   ├── endpoints/
│   │   │   ├── pii_detection.py      # Core PII detection endpoints
│   │   │   ├── document_processing.py # Document upload/processing
│   │   │   ├── compliance.py         # Compliance reporting
│   │   │   ├── audit.py              # Audit trail management
│   │   │   └── health.py             # System health checks
│   │   └── router.py
│   └── dependencies.py
├── core/
│   ├── config.py                     # Environment configuration
│   ├── security.py                   # Authentication/authorization
│   ├── logging.py                    # Structured logging
│   └── middleware.py                 # Request/response middleware
├── services/
│   ├── pii/
│   │   ├── ircc_detector.py          # Main PII detection service
│   │   ├── pattern_engine.py         # Rule-based pattern matching
```

```
│   │   ├── ml_engine.py              # Machine learning models
│   │   ├── language_detector.py      # Language identification
│   │   ├── validator.py              # PII validation logic
│   │   └── anonymizer.py             # Anonymization strategies
│   ├── document/
│   │   ├── processor.py              # Document parsing
│   │   ├── classifier.py             # Document type classification
│   │   └── ocr_service.py            # OCR for images
│   ├── compliance/
│   │   ├── audit_service.py          # Audit logging
│   │   ├── privacy_engine.py         # Privacy compliance
│   │   └── retention_manager.py      # Data retention policies
│   └── integration/
│       ├── gcms_connector.py         # GCMS integration
│       └── file_manager.py           # Secure file handling
├── models/
│   ├── database/
│   │   ├── audit.py                  # Audit log models
│   │   ├── document.py               # Document metadata
│   │   └── user.py                   # User management
│   └── schemas/
│       ├── pii_detection.py          # PII detection schemas
│       ├── document.py               # Document processing schemas
│       └── compliance.py             # Compliance reporting schemas
├── utils/
│   ├── encryption.py                 # Encryption utilities
│   ├── validation.py                 # Input validation
│   └── performance.py                # Performance monitoring
└── tests/
    ├── unit/
    ├── integration/
    └── performance/
```

2. Database Schema Design

```sql
-- Audit and compliance tables
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    user_id VARCHAR(50) NOT NULL,
    officer_badge VARCHAR(20),
    action_type VARCHAR(50) NOT NULL,
    document_id UUID,
    case_id VARCHAR(50),
    pii_entities_detected INTEGER DEFAULT 0,
    processing_time_ms INTEGER,
    classification_level VARCHAR(20) DEFAULT 'unclassified',
    ip_address INET,
    user_agent TEXT,
    request_id UUID,
    success BOOLEAN NOT NULL,
    error_message TEXT,
    metadata JSONB
);

-- Document processing records
```

```
CREATE TABLE document_processing (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_hash VARCHAR(64) UNIQUE NOT NULL,
    original_filename VARCHAR(255),
    document_type VARCHAR(50),
    file_size INTEGER,
    language_detected VARCHAR(10),
    processing_status VARCHAR(20) DEFAULT 'pending',
    pii_detection_results JSONB,
    anonymized_content_hash VARCHAR(64),
    retention_date DATE,
    classification_level VARCHAR(20),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- PII entity catalog
CREATE TABLE pii_entities (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_id UUID REFERENCES document_processing(id),
    entity_type VARCHAR(50) NOT NULL,
    entity_value_hash VARCHAR(64), -- Hashed for privacy
    confidence_score DECIMAL(3,2),
    detection_method VARCHAR(20), -- 'pattern', 'ml', 'hybrid'
    anonymization_strategy VARCHAR(50),
    position_start INTEGER,
    position_end INTEGER,
    context_snippet_hash VARCHAR(64),
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- System configuration
CREATE TABLE system_config (
    key VARCHAR(100) PRIMARY KEY,
    value JSONB NOT NULL,
    description TEXT,
    updated_by VARCHAR(50),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Performance metrics
CREATE TABLE performance_metrics (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    metric_name VARCHAR(50) NOT NULL,
    metric_value DECIMAL(10,4),
    document_type VARCHAR(50),
    processing_mode VARCHAR(20), -- 'realtime', 'batch'
    server_instance VARCHAR(50)
);

-- Compliance violations
CREATE TABLE compliance_violations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    violation_type VARCHAR(50) NOT NULL,
    severity VARCHAR(20) NOT NULL, -- 'low', 'medium', 'high', 'critical'
    description TEXT NOT NULL,
```

```
    document_id UUID REFERENCES document_processing(id),
    user_id VARCHAR(50),
    detected_at TIMESTAMPTZ DEFAULT NOW(),
    resolved_at TIMESTAMPTZ,
    resolution_notes TEXT,
    status VARCHAR(20) DEFAULT 'open' -- 'open', 'investigating', 'resolved'
);

-- Create indexes for performance
CREATE INDEX idx_audit_logs_timestamp ON audit_logs(timestamp);
CREATE INDEX idx_audit_logs_user_id ON audit_logs(user_id);
CREATE INDEX idx_document_processing_hash ON
document_processing(document_hash);
CREATE INDEX idx_pii_entities_document_id ON pii_entities(document_id);
CREATE INDEX idx_performance_metrics_timestamp ON
performance_metrics(timestamp);
```

Implementation Roadmap

Phase 1: Foundation (Weeks 1-4)

Objective: Establish core infrastructure and basic PII detection

Week 1: Project Setup

- [ ] Initialize FastAPI project structure
- [ ] Set up development environment with Docker
- [ ] Configure PostgreSQL database with encryption
- [ ] Implement basic authentication and authorization
- [ ] Set up logging and monitoring infrastructure

Week 2: Core PII Detection

- [ ] Implement rule-based pattern detection for Canadian PII
- [ ] Set up Hugging Face Transformers pipeline
- [ ] Create basic document processing pipeline
- [ ] Implement input validation and sanitization
- [ ] Add basic error handling and logging

Week 3: ML Model Integration

- [ ] Integrate Stanford deidentifier model
- [ ] Add multilingual NER model support
- [ ] Implement confidence scoring and validation
- [ ] Create model caching and optimization
- [ ] Add performance monitoring

Week 4: API Development

- [ ] Create core PII detection endpoints
- [ ] Implement document upload and processing
- [ ] Add basic audit logging
- [ ] Create health check endpoints
- [ ] Implement rate limiting and security middleware

Phase 2: Enhancement (Weeks 5-8)

Objective: Add IRCC-specific features and compliance

Week 5: IRCC-Specific PII

- [ ] Implement UCI and immigration file number detection
- [ ] Add work/study permit number validation
- [ ] Create passport number detection with country codes
- [ ] Implement Canadian postal code validation
- [ ] Add provincial health card detection

Week 6: Document Classification

- [ ] Implement document type classification
- [ ] Add immigration form recognition
- [ ] Create document metadata extraction
- [ ] Implement OCR for scanned documents
- [ ] Add file format conversion utilities

Week 7: Multilingual Support

- [ ] Integrate French language processing
- [ ] Add support for major immigrant languages
- [ ] Implement automatic language detection
- [ ] Create multilingual PII pattern recognition
- [ ] Add Unicode and script handling

Week 8: Anonymization Engine

- [ ] Implement context-aware anonymization
- [ ] Create anonymization strategy selection
- [ ] Add format-preserving anonymization
- [ ] Implement reversible anonymization (for authorized access)
- [ ] Add anonymization quality validation

Phase 3: Integration & Compliance (Weeks 9-12)

Objective: GCMS integration and full compliance framework

Week 9: GCMS Integration

- [ ] Implement GCMS API connectors
- [ ] Create case management integration
- [ ] Add officer workflow support
- [ ] Implement document retrieval from GCMS
- [ ] Add status synchronization

Week 10: Compliance Framework

- [ ] Implement Privacy Act compliance checks
- [ ] Add PIPEDA compliance validation
- [ ] Create audit trail generation
- [ ] Implement data retention policies
- [ ] Add compliance reporting dashboard

Week 11: Security Hardening

- [ ] Implement encryption at rest and in transit
- [ ] Add role-based access control
- [ ] Create security monitoring and alerting
- [ ] Implement audit log protection
- [ ] Add intrusion detection

Week 12: Performance Optimization

- [ ] Optimize ML model inference
- [ ] Implement caching strategies
- [ ] Add load balancing and scaling
- [ ] Optimize database queries
- [ ] Add performance monitoring dashboard

Phase 4: Deployment & Operations (Weeks 13-16)

Objective: Production deployment and operational readiness

Week 13: Deployment Infrastructure

- [ ] Create production Docker containers
- [ ] Set up Kubernetes deployment
- [ ] Implement CI/CD pipelines
- [ ] Add infrastructure monitoring
- [ ] Create deployment documentation

Week 14: Testing & Validation

- [ ] Comprehensive integration testing
- [ ] Performance and load testing
- [ ] Security penetration testing
- [ ] User acceptance testing with IRCC
- [ ] Compliance validation testing

Week 15: Training & Documentation

- [ ] Create operator training materials
- [ ] Write technical documentation
- [ ] Develop troubleshooting guides
- [ ] Create compliance documentation
- [ ] Prepare go-live procedures

Week 16: Go-Live & Support

- [ ] Production deployment
- [ ] User training sessions
- [ ] Monitor system performance
- [ ] Address initial issues
- [ ] Knowledge transfer to operations team

Core Components

1. IRCC PII Detection Engine

```
# app/services/pii/ircc_detector.py

from typing import Dict, List, Optional, Tuple
from dataclasses import dataclass
import asyncio
import hashlib
import logging
from datetime import datetime
import spacy
from transformers import pipeline, AutoTokenizer
import re

logger = logging.getLogger(__name__)

@dataclass
class IRCCPIIEntity:
    """Represents a detected PII entity with IRCC-specific metadata"""
    text: str
    entity_type: str
    start_position: int
    end_position: int
    confidence_score: float
```

```
    detection_method: str  # 'pattern', 'ml', 'hybrid'
    anonymization_strategy: str
    sensitivity_level: str  # 'low', 'medium', 'high', 'critical'
    retention_category: str
    validation_status: str  # 'valid', 'invalid', 'uncertain'

@dataclass
class IRCCDocument:
    """Represents a document with IRCC-specific metadata"""
    content: str
    document_type: str
    language: str
    case_id: Optional[str]
    classification_level: str
    source_system: str

@dataclass
class IRCCPIIResult:
    """Complete PII detection result for IRCC"""
    document_id: str
    entities: List[IRCCPIIEntity]
    anonymized_content: str
    processing_metadata: Dict
    compliance_status: Dict
    audit_record: Dict

class IRCCPIIDetector:
    """
    Main PII detection engine for IRCC
    Combines rule-based patterns, ML models, and validation
    """

    def __init__(self):
        self.ircc_patterns = self._initialize_ircc_patterns()
        self.ml_models = self._initialize_ml_models()
        self.validators = self._initialize_validators()
        self.anonymizers = self._initialize_anonymizers()
        self.performance_tracker = IRCCPerformanceTracker()

    def _initialize_ircc_patterns(self) -> Dict:
        """Initialize IRCC-specific PII patterns"""
        return {
            # Immigration identifiers
            'uci_number': {
                'pattern': r'\b\d{4}[-\s]?\d{4}\b',
                'validator': self._validate_uci,
                'sensitivity': 'critical',
                'description': 'Unique Client Identifier'
            },
            'immigration_file_number': {
                'pattern': r'\b[A-Z]{1,2}\d{6,10}\b',
                'validator': self._validate_file_number,
                'sensitivity': 'critical',
                'description': 'Immigration File Number'
            },
            'work_permit_number': {
                'pattern': r'\bWP\d{8,10}\b',
```

```
            'validator': self._validate_work_permit,
            'sensitivity': 'high',
            'description': 'Work Permit Number'
        },
        'study_permit_number': {
            'pattern': r'\bSP\d{8,10}\b',
            'validator': self._validate_study_permit,
            'sensitivity': 'high',
            'description': 'Study Permit Number'
        },
        'pr_card_number': {
            'pattern': r'\bPR\d{8}\b',
            'validator': self._validate_pr_card,
            'sensitivity': 'critical',
            'description': 'Permanent Resident Card Number'
        },
        'temporary_resident_visa': {
            'pattern': r'\bTRV\d{8,10}\b',
            'validator': self._validate_trv,
            'sensitivity': 'high',
            'description': 'Temporary Resident Visa'
        },

        # Canadian government identifiers
        'social_insurance_number': {
            'pattern': r'\b\d{3}[-\s]?\d{3}[-\s]?\d{3}\b',
            'validator': self._validate_sin,
            'sensitivity': 'critical',
            'description': 'Social Insurance Number'
        },
        'canadian_passport': {
            'pattern': r'\b[A-Z]{2}\d{6,8}\b',
            'validator': self._validate_canadian_passport,
            'sensitivity': 'critical',
            'description': 'Canadian Passport Number'
        },
        'canadian_postal_code': {
            'pattern': r'\b[A-Za-z]\d[A-Za-z][-\s]?\d[A-Za-z]\d\b',
            'validator': self._validate_postal_code,
            'sensitivity': 'medium',
            'description': 'Canadian Postal Code'
        },

        # Provincial health cards
        'ontario_health_card': {
            'pattern': r'\b\d{4}[-\s]?\d{3}[-\s]?\d{3}[-\s]?\d{2}\b',
            'validator': self._validate_ontario_health,
            'sensitivity': 'high',
            'description': 'Ontario Health Card'
        },
        'quebec_health_card': {
            'pattern': r'\b[A-Z]{4}\d{8}\b',
            'validator': self._validate_quebec_health,
            'sensitivity': 'high',
            'description': 'Quebec Health Card'
        },
```

```
            # International identifiers
            'foreign_passport': {
                'pattern': r'\b[A-Z0-9]{6,12}\b',
                'validator': self._validate_foreign_passport,
                'sensitivity': 'high',
                'description': 'Foreign Passport Number'
            },
            'international_phone': {
                'pattern': r'\+\d{1,3}[-\s]?\d{1,4}[-\s]?\d{1,4}[-
\s]?\d{1,9}',
                'validator': self._validate_international_phone,
                'sensitivity': 'medium',
                'description': 'International Phone Number'
            }
        }

    def _initialize_ml_models(self) -> Dict:
        """Initialize machine learning models for PII detection"""
        return {
            'primary_ner': pipeline(
                "ner",
                model="StanfordAIMI/stanford-deidentifier-base",
                aggregation_strategy="simple",
                device=-1,  # CPU for security
                return_all_scores=True
            ),
            'multilingual_ner': pipeline(
                "ner",
                model="Davlan/bert-base-multilingual-cased-ner-hrl",
                aggregation_strategy="simple",
                device=-1
            ),
            'french_ner': pipeline(
                "ner",
                model="Jean-Baptiste/roberta-large-ner-french",
                aggregation_strategy="simple",
                device=-1
            )
        }

    def _initialize_validators(self) -> Dict:
        """Initialize PII validation functions"""
        return {
            'luhn_algorithm': self._luhn_check,
            'checksum_validation': self._checksum_validate,
            'format_validation': self._format_validate,
            'context_validation': self._context_validate
        }

    def _initialize_anonymizers(self) -> Dict:
        """Initialize anonymization strategies"""
        return {
            'redaction': lambda text, entity: '[REDACTED]',
            'masking': lambda text, entity: self._mask_entity(text, entity),
            'pseudonymization': lambda text, entity:
self._pseudonymize_entity(text, entity),
```

```
            'generalization': lambda text, entity:
self._generalize_entity(text, entity),
            'date_shifting': lambda text, entity: self._shift_date(text,
entity)
        }

    async def detect_pii(
        self,
        document: IRCCDocument,
        user_id: str,
        processing_options: Optional[Dict] = None
    ) -> IRCCPIIResult:
        """
        Main PII detection method for IRCC documents

        Args:
            document: IRCCDocument with content and metadata
            user_id: ID of the user requesting detection
            processing_options: Optional processing configuration

        Returns:
            IRCCPIIResult with detected entities and anonymized content
        """

        start_time = datetime.utcnow()
        document_id = self._generate_document_id(document)

        try:
            # Step 1: Document preprocessing and validation
            preprocessed_content = await self._preprocess_document(document)

            # Step 2: Language detection and model selection
            detected_language = await
self._detect_language(preprocessed_content)
            selected_models =
self._select_models_for_language(detected_language)

            # Step 3: Multi-layer PII detection
            detection_tasks = [
                self._pattern_based_detection(preprocessed_content,
document.document_type),
                self._ml_based_detection(preprocessed_content,
selected_models),
                self._context_based_detection(preprocessed_content, document)
            ]

            pattern_entities, ml_entities, context_entities = await
asyncio.gather(*detection_tasks)

            # Step 4: Entity consolidation and validation
            all_entities = pattern_entities + ml_entities + context_entities
            validated_entities = await
self._validate_and_consolidate_entities(
                all_entities, preprocessed_content, document
            )

            # Step 5: Anonymization
```

```
            anonymized_content = await self._anonymize_content(
                preprocessed_content, validated_entities,
document.classification_level
            )

            # Step 6: Compliance checking
            compliance_status = await self._check_compliance(
                validated_entities, document, processing_options
            )

            # Step 7: Generate audit record
            processing_time = (datetime.utcnow() -
start_time).total_seconds() * 1000
            audit_record = await self._generate_audit_record(
                document_id, user_id, validated_entities, processing_time,
document
            )

            # Step 8: Performance tracking
            await self.performance_tracker.record_processing(
                document_type=document.document_type,
                processing_time=processing_time,
                entity_count=len(validated_entities),
                success=True
            )

            return IRCCPIIResult(
                document_id=document_id,
                entities=validated_entities,
                anonymized_content=anonymized_content,
                processing_metadata={
                    'processing_time_ms': processing_time,
                    'language_detected': detected_language,
                    'models_used': list(selected_models.keys()),
                    'total_entities': len(validated_entities),
                    'entity_breakdown':
self._get_entity_breakdown(validated_entities)
                },
                compliance_status=compliance_status,
                audit_record=audit_record
            )

        except Exception as e:
            logger.error(f"PII detection failed for document {document_id}:
{str(e)}")

            # Record failure for monitoring
            await self.performance_tracker.record_processing(
                document_type=document.document_type,
                processing_time=(datetime.utcnow() -
start_time).total_seconds() * 1000,
                entity_count=0,
                success=False,
                error=str(e)
            )

            raise IRCCPIIDetectionError(f"PII detection failed: {str(e)}")
```

```
    async def _pattern_based_detection(
        self,
        content: str,
        document_type: str
    ) -> List[IRCCPIIEntity]:
        """Detect PII using rule-based patterns"""

        entities = []

        for pattern_name, pattern_config in self.ircc_patterns.items():
            pattern = pattern_config['pattern']
            matches = re.finditer(pattern, content, re.IGNORECASE)

            for match in matches:
                # Validate the match using pattern-specific validator
                if pattern_config.get('validator'):
                    validation_result =
pattern_config['validator'](match.group())
                    if not validation_result['valid']:
                        continue

                entity = IRCCPIIEntity(
                    text=match.group(),
                    entity_type=pattern_name,
                    start_position=match.start(),
                    end_position=match.end(),
                    confidence_score=0.95,  # High confidence for validated
patterns
                    detection_method='pattern',

anonymization_strategy=self._get_anonymization_strategy(pattern_name),
                    sensitivity_level=pattern_config['sensitivity'],

retention_category=self._get_retention_category(pattern_name),
                    validation_status='valid'
                )

                entities.append(entity)

        return entities

    async def _ml_based_detection(
        self,
        content: str,
        models: Dict
    ) -> List[IRCCPIIEntity]:
        """Detect PII using machine learning models"""

        entities = []

        for model_name, model in models.items():
            try:
                # Run NER model
                ner_results = model(content)

                for result in ner_results:
```

```
                    # Map NER labels to IRCC entity types
                    ircc_entity_type =
self._map_ner_label_to_ircc(result['entity_group'])

                    if ircc_entity_type:
                        entity = IRCCPIIEntity(
                            text=result['word'],
                            entity_type=ircc_entity_type,
                            start_position=result['start'],
                            end_position=result['end'],
                            confidence_score=result['score'],
                            detection_method='ml',

anonymization_strategy=self._get_anonymization_strategy(ircc_entity_type),

sensitivity_level=self._get_sensitivity_level(ircc_entity_type),

retention_category=self._get_retention_category(ircc_entity_type),
                            validation_status='uncertain'  # Requires
validation
                        )

                        entities.append(entity)

            except Exception as e:
                logger.warning(f"ML model {model_name} failed: {str(e)}")
                continue

        return entities

    async def _validate_and_consolidate_entities(
        self,
        entities: List[IRCCPIIEntity],
        content: str,
        document: IRCCDocument
    ) -> List[IRCCPIIEntity]:
        """Validate and consolidate detected entities"""

        # Remove duplicates based on position overlap
        deduplicated = self._remove_overlapping_entities(entities)

        # Validate entities using multiple methods
        validated_entities = []

        for entity in deduplicated:
            validation_result = await self._comprehensive_validation(entity,
content, document)

            if validation_result['valid']:
                entity.validation_status = 'valid'
                entity.confidence_score = min(
                    entity.confidence_score *
validation_result['confidence_multiplier'],
                    1.0
                )
                validated_entities.append(entity)
            elif validation_result['uncertain']:
```

```
                entity.validation_status = 'uncertain'
                validated_entities.append(entity)
            # Invalid entities are discarded

        return validated_entities

    # Validation methods for specific PII types
    def _validate_uci(self, uci: str) -> Dict:
        """Validate UCI number format and checksum"""
        clean_uci = re.sub(r'[-\s]', '', uci)

        if len(clean_uci) != 8 or not clean_uci.isdigit():
            return {'valid': False, 'reason': 'Invalid format'}

        # UCI uses a simple checksum algorithm
        checksum = sum(int(digit) * (i + 1) for i, digit in
enumerate(clean_uci[:7])) % 10
        expected_check = int(clean_uci[7])

        return {
            'valid': checksum == expected_check,
            'reason': 'Checksum validation' if checksum == expected_check
else 'Invalid checksum'
        }

    def _validate_sin(self, sin: str) -> Dict:
        """Validate Social Insurance Number using Luhn algorithm"""
        clean_sin = re.sub(r'[-\s]', '', sin)

        if len(clean_sin) != 9 or not clean_sin.isdigit():
            return {'valid': False, 'reason': 'Invalid format'}

        # SIN validation using modified Luhn algorithm
        def luhn_checksum(number):
            total = 0
            reverse_digits = number[::-1]

            for i, digit in enumerate(reverse_digits):
                n = int(digit)
                if i % 2 == 1:  # Every second digit
                    n *= 2
                    if n > 9:
                        n = (n // 10) + (n % 10)
                total += n

            return total % 10 == 0

        return {
            'valid': luhn_checksum(clean_sin),
            'reason': 'Luhn validation'
        }

    def _validate_postal_code(self, postal_code: str) -> Dict:
        """Validate Canadian postal code format"""
        clean_postal = re.sub(r'[-\s]', '', postal_code.upper())

        if len(clean_postal) != 6:
```

```
            return {'valid': False, 'reason': 'Invalid length'}

        # Canadian postal code pattern: Letter-Digit-Letter Digit-Letter-
Digit
        pattern = r'^[A-Z]\d[A-Z]\d[A-Z]\d$'

        return {
            'valid': bool(re.match(pattern, clean_postal)),
            'reason': 'Format validation'
        }

class IRCCPerformanceTracker:
    """Track performance metrics for IRCC PII detection"""

    def __init__(self):
        self.metrics = {
            'processing_times': [],
            'entity_counts': [],
            'success_rate': 0.0,
            'error_counts': {}
        }

    async def record_processing(
        self,
        document_type: str,
        processing_time: float,
        entity_count: int,
        success: bool,
        error: Optional[str] = None
    ):
        """Record processing metrics"""

        # Store metrics in database for analysis
        # Implementation depends on chosen database/monitoring solution
        pass

class IRCCPIIDetectionError(Exception):
    """Custom exception for IRCC PII detection errors"""
    pass
```

2. Document Processing Pipeline

```
# app/services/document/ircc_processor.py

from typing import Dict, List, Optional, BinaryIO
from dataclasses import dataclass
import asyncio
import aiofiles
import magic
from PIL import Image
import pytesseract
from pdf2image import convert_from_bytes
import docx
from email import message_from_string
import logging
```

```
logger = logging.getLogger(__name__)

@dataclass
class IRCCDocumentMetadata:
    """Metadata for IRCC documents"""
    filename: str
    file_size: int
    mime_type: str
    document_type: str
    language: str
    classification_level: str
    case_id: Optional[str]
    source_system: str
    extraction_method: str
    page_count: Optional[int]
    ocr_confidence: Optional[float]

class IRCCDocumentProcessor:
    """Process various document types for IRCC PII detection"""

    def __init__(self):
        self.supported_formats = {
            'application/pdf': self._process_pdf,
            'application/msword': self._process_doc,
            'application/vnd.openxmlformats-
officedocument.wordprocessingml.document': self._process_docx,
            'text/plain': self._process_text,
            'message/rfc822': self._process_email,
            'image/jpeg': self._process_image,
            'image/png': self._process_image,
            'image/tiff': self._process_image
        }

        self.document_classifiers = {
            'application_forms': self._classify_application_form,
            'supporting_documents': self._classify_supporting_document,
            'correspondence': self._classify_correspondence,
            'decision_letters': self._classify_decision_letter
        }

    async def process_document(
        self,
        file_data: bytes,
        filename: str,
        case_id: Optional[str] = None,
        classification_level: str = 'unclassified'
    ) -> Tuple[str, IRCCDocumentMetadata]:
        """
        Process a document and extract text content

        Args:
            file_data: Raw file bytes
            filename: Original filename
            case_id: Associated case ID
            classification_level: Security classification

        Returns:
```

```
            Tuple of (extracted_text, metadata)
        """

        try:
            # Detect file type
            mime_type = magic.from_buffer(file_data, mime=True)

            if mime_type not in self.supported_formats:
                raise UnsupportedDocumentFormat(f"Unsupported format:
{mime_type}")

            # Extract text using appropriate processor
            processor = self.supported_formats[mime_type]
            extraction_result = await processor(file_data, filename)

            # Detect language
            detected_language = await
self._detect_language(extraction_result['text'])

            # Classify document type
            document_type = await self._classify_document_type(
                extraction_result['text'],
                filename
            )

            # Create metadata
            metadata = IRCCDocumentMetadata(
                filename=filename,
                file_size=len(file_data),
                mime_type=mime_type,
                document_type=document_type,
                language=detected_language,
                classification_level=classification_level,
                case_id=case_id,
                source_system='ircc_pii_system',
                extraction_method=extraction_result['method'],
                page_count=extraction_result.get('page_count'),
                ocr_confidence=extraction_result.get('ocr_confidence')
            )

            return extraction_result['text'], metadata

        except Exception as e:
            logger.error(f"Document processing failed for {filename}:
{str(e)}")
            raise DocumentProcessingError(f"Failed to process document:
{str(e)}")

    async def _process_pdf(self, file_data: bytes, filename: str) -> Dict:
        """Process PDF documents"""
        try:
            # First try text extraction (for text-based PDFs)
            import PyPDF2
            from io import BytesIO

            pdf_buffer = BytesIO(file_data)
            pdf_reader = PyPDF2.PdfReader(pdf_buffer)
```

```
            text_content = ""
            for page in pdf_reader.pages:
                text_content += page.extract_text() + "\n"

            # If text extraction successful and content found
            if text_content.strip():
                return {
                    'text': text_content,
                    'method': 'text_extraction',
                    'page_count': len(pdf_reader.pages)
                }

            # Fall back to OCR for image-based PDFs
            images = convert_from_bytes(file_data)
            ocr_text = ""
            confidence_scores = []

            for image in images:
                # Use pytesseract for OCR
                ocr_result = pytesseract.image_to_data(
                    image,
                    output_type=pytesseract.Output.DICT,
                    config='--oem 3 --psm 6'  # Optimized for documents
                )

                page_text = " ".join([
                    word for i, word in enumerate(ocr_result['text'])
                    if int(ocr_result['conf'][i]) > 30  # Filter low
confidence words
                ])

                ocr_text += page_text + "\n"

                # Calculate average confidence for this page
                valid_confidences = [
                    int(conf) for conf in ocr_result['conf']
                    if int(conf) > 0
                ]
                if valid_confidences:
                    confidence_scores.append(sum(valid_confidences) /
len(valid_confidences))

            avg_confidence = sum(confidence_scores) / len(confidence_scores)
if confidence_scores else 0

            return {
                'text': ocr_text,
                'method': 'ocr',
                'page_count': len(images),
                'ocr_confidence': avg_confidence
            }

        except Exception as e:
            raise DocumentProcessingError(f"PDF processing failed: {str(e)}")

    async def _process_docx(self, file_data: bytes, filename: str) -> Dict:
```

```
        """Process Word documents (.docx)"""
        try:
            from io import BytesIO

            doc_buffer = BytesIO(file_data)
            doc = docx.Document(doc_buffer)

            text_content = ""
            for paragraph in doc.paragraphs:
                text_content += paragraph.text + "\n"

            # Extract text from tables
            for table in doc.tables:
                for row in table.rows:
                    for cell in row.cells:
                        text_content += cell.text + " "
                text_content += "\n"

            return {
                'text': text_content,
                'method': 'structured_extraction',
                'page_count': len(doc.element.xpath('//w:sectPr'))
            }

        except Exception as e:
            raise DocumentProcessingError(f"DOCX processing failed:
{str(e)}")

    async def _process_image(self, file_data: bytes, filename: str) -> Dict:
        """Process image files using OCR"""
        try:
            from io import BytesIO

            image_buffer = BytesIO(file_data)
            image = Image.open(image_buffer)

            # Preprocess image for better OCR
            # Convert to grayscale and enhance contrast
            if image.mode != 'L':
                image = image.convert('L')

            # Use pytesseract for OCR with configuration optimized for
documents
            ocr_result = pytesseract.image_to_data(
                image,
                output_type=pytesseract.Output.DICT,
                config='--oem 3 --psm 6 -l eng+fra'  # English and French
            )

            # Filter out low confidence words
            text_content = " ".join([
                word for i, word in enumerate(ocr_result['text'])
                if int(ocr_result['conf'][i]) > 40
            ])

            # Calculate average confidence
            valid_confidences = [
```

```
                int(conf) for conf in ocr_result['conf']
                if int(conf) > 0
            ]
            avg_confidence = sum(valid_confidences) / len(valid_confidences)
if valid_confidences else 0

            return {
                'text': text_content,
                'method': 'ocr',
                'page_count': 1,
                'ocr_confidence': avg_confidence
            }

        except Exception as e:
            raise DocumentProcessingError(f"Image processing failed:
{str(e)}")

    async def _classify_document_type(self, text: str, filename: str) -> str:
        """Classify the type of IRCC document"""

        # Check filename patterns first
        filename_lower = filename.lower()

        if any(pattern in filename_lower for pattern in ['imm1294',
'work_permit', 'wp_']):
            return 'work_permit_application'
        elif any(pattern in filename_lower for pattern in ['imm5709',
'study_permit', 'sp_']):
            return 'study_permit_application'
        elif any(pattern in filename_lower for pattern in ['imm5444',
'pr_application']):
            return 'permanent_residence_application'
        elif any(pattern in filename_lower for pattern in ['imm0008',
'generic_application']):
            return 'generic_application_form'
        elif 'passport' in filename_lower:
            return 'passport_copy'
        elif 'birth_certificate' in filename_lower:
            return 'birth_certificate'
        elif any(pattern in filename_lower for pattern in ['letter',
'correspondence']):
            return 'correspondence'
        elif 'decision' in filename_lower or 'approval' in filename_lower:
            return 'decision_letter'

        # Analyze content for document type indicators
        text_lower = text.lower()

        # Look for form-specific keywords
        if 'application for work permit' in text_lower:
            return 'work_permit_application'
        elif 'application for study permit' in text_lower:
            return 'study_permit_application'
        elif 'application for permanent residence' in text_lower:
            return 'permanent_residence_application'
        elif 'citizenship application' in text_lower:
            return 'citizenship_application'
```

```
        elif 'temporary resident visa' in text_lower:
            return 'temporary_resident_visa_application'
        elif 'passport' in text_lower and 'republic' in text_lower:
            return 'passport_copy'
        elif 'birth certificate' in text_lower or 'certificate of birth' in
text_lower:
            return 'birth_certificate'
        elif 'marriage certificate' in text_lower:
            return 'marriage_certificate'
        elif 'diploma' in text_lower or 'degree' in text_lower or
'transcript' in text_lower:
            return 'educational_document'
        elif 'employment letter' in text_lower or 'job offer' in text_lower:
            return 'employment_document'
        elif 'medical exam' in text_lower or 'health assessment' in
text_lower:
            return 'medical_document'
        elif 'police certificate' in text_lower or 'criminal record' in
text_lower:
            return 'police_certificate'

        # Default classification
        return 'supporting_document'

class UnsupportedDocumentFormat(Exception):
    """Exception for unsupported document formats"""
    pass

class DocumentProcessingError(Exception):
    """Exception for document processing errors"""
    pass
```

3. Security and Compliance Framework

```
# app/services/compliance/ircc_compliance.py

from typing import Dict, List, Optional
from dataclasses import dataclass
from datetime import datetime, timedelta
import hashlib
import hmac
import logging
from enum import Enum

logger = logging.getLogger(__name__)

class ClassificationLevel(Enum):
    UNCLASSIFIED = "unclassified"
    PROTECTED_A = "protected_a"
    PROTECTED_B = "protected_b"

class RetentionCategory(Enum):
    TEMPORARY = "temporary"  # 2 years
    OPERATIONAL = "operational"  # 7 years
    HISTORICAL = "historical"  # 25 years
    PERMANENT = "permanent"  # Indefinite
```

```
@dataclass
class ComplianceViolation:
    violation_type: str
    severity: str  # 'low', 'medium', 'high', 'critical'
    description: str
    recommendation: str
    regulatory_reference: str

@dataclass
class AuditRecord:
    audit_id: str
    timestamp: datetime
    user_id: str
    action_type: str
    document_id: str
    case_id: Optional[str]
    pii_entities_detected: int
    classification_level: str
    processing_time_ms: float
    success: bool
    violations: List[ComplianceViolation]
    metadata: Dict

class IRCCComplianceEngine:
    """Compliance engine for IRCC PII processing"""

    def __init__(self):
        self.privacy_act_rules = self._initialize_privacy_act_rules()
        self.pipeda_rules = self._initialize_pipeda_rules()
        self.retention_policies = self._initialize_retention_policies()
        self.classification_rules = self._initialize_classification_rules()

    def _initialize_privacy_act_rules(self) -> Dict:
        """Initialize Privacy Act compliance rules"""
        return {
            'purpose_limitation': {
                'description': 'Personal information used only for stated
purposes',
                'validator': self._validate_purpose_limitation,
                'reference': 'Privacy Act s.7'
            },
            'minimal_collection': {
                'description': 'Collect only necessary personal information',
                'validator': self._validate_minimal_collection,
                'reference': 'Privacy Act s.4'
            },
            'accuracy_requirement': {
                'description': 'Personal information must be accurate and
current',
                'validator': self._validate_accuracy_requirement,
                'reference': 'Privacy Act s.6'
            },
            'security_safeguards': {
                'description': 'Appropriate security measures for personal
information',
                'validator': self._validate_security_safeguards,
```

```
                'reference': 'Privacy Act s.8'
            },
            'retention_limits': {
                'description': 'Personal information retained only as long as
necessary',
                'validator': self._validate_retention_limits,
                'reference': 'Privacy Act Schedule'
            }
        }

    def _initialize_pipeda_rules(self) -> Dict:
        """Initialize PIPEDA compliance rules"""
        return {
            'consent_requirements': {
                'description': 'Valid consent for collection, use, and
disclosure',
                'validator': self._validate_consent_requirements,
                'reference': 'PIPEDA Schedule 1, Principle 3'
            },
            'limiting_collection': {
                'description': 'Limit collection to stated purposes',
                'validator': self._validate_limiting_collection,
                'reference': 'PIPEDA Schedule 1, Principle 4'
            },
            'safeguards_principle': {
                'description': 'Security safeguards appropriate to
sensitivity',
                'validator': self._validate_safeguards_principle,
                'reference': 'PIPEDA Schedule 1, Principle 7'
            },
            'openness_principle': {
                'description': 'Policies and practices about personal
information',
                'validator': self._validate_openness_principle,
                'reference': 'PIPEDA Schedule 1, Principle 8'
            }
        }

    def _initialize_retention_policies(self) -> Dict:
        """Initialize data retention policies"""
        return {
            'immigration_applications': {
                'retention_period': timedelta(days=2555),  # 7 years
                'category': RetentionCategory.OPERATIONAL,
                'disposition': 'destroy_after_retention'
            },
            'work_permits': {
                'retention_period': timedelta(days=2555),  # 7 years
                'category': RetentionCategory.OPERATIONAL,
                'disposition': 'destroy_after_retention'
            },
            'study_permits': {
                'retention_period': timedelta(days=2555),  # 7 years
                'category': RetentionCategory.OPERATIONAL,
                'disposition': 'destroy_after_retention'
            },
            'citizenship_records': {
```

```
                'retention_period': timedelta(days=9125),  # 25 years
                'category': RetentionCategory.HISTORICAL,
                'disposition': 'transfer_to_archives'
            },
            'permanent_residence': {
                'retention_period': timedelta(days=9125),  # 25 years
                'category': RetentionCategory.HISTORICAL,
                'disposition': 'transfer_to_archives'
            },
            'temporary_records': {
                'retention_period': timedelta(days=730),   # 2 years
                'category': RetentionCategory.TEMPORARY,
                'disposition': 'destroy_after_retention'
            },
            'audit_logs': {
                'retention_period': timedelta(days=3650),  # 10 years
                'category': RetentionCategory.OPERATIONAL,
                'disposition': 'destroy_after_retention'
            }
        }

    async def assess_compliance(
        self,
        pii_entities: List,
        document_metadata: Dict,
        processing_context: Dict
    ) -> Dict:
        """Comprehensive compliance assessment"""

        violations = []
        compliance_score = 100.0

        # Privacy Act compliance checks
        privacy_violations = await self._check_privacy_act_compliance(
            pii_entities, document_metadata, processing_context
        )
        violations.extend(privacy_violations)

        # PIPEDA compliance checks (if applicable)
        if self._is_pipeda_applicable(document_metadata):
            pipeda_violations = await self._check_pipeda_compliance(
                pii_entities, document_metadata, processing_context
            )
            violations.extend(pipeda_violations)

        # Classification level compliance
        classification_violations = await
self._check_classification_compliance(
            pii_entities, document_metadata
        )
        violations.extend(classification_violations)

        # Data retention compliance
        retention_violations = await self._check_retention_compliance(
            document_metadata, processing_context
        )
        violations.extend(retention_violations)
```

```
        # Calculate compliance score
        for violation in violations:
            if violation.severity == 'critical':
                compliance_score -= 25
            elif violation.severity == 'high':
                compliance_score -= 15
            elif violation.severity == 'medium':
                compliance_score -= 10
            elif violation.severity == 'low':
                compliance_score -= 5

        compliance_score = max(compliance_score, 0.0)

        return {
            'compliance_score': compliance_score,
            'violations': violations,
            'privacy_act_compliant': len([v for v in violations if 'Privacy
Act' in v.regulatory_reference]) == 0,
            'pipeda_compliant': len([v for v in violations if 'PIPEDA' in
v.regulatory_reference]) == 0,
            'classification_compliant': len([v for v in violations if
'classification' in v.violation_type.lower()]) == 0,
            'retention_compliant': len([v for v in violations if 'retention'
in v.violation_type.lower()]) == 0,
            'recommendations': [v.recommendation for v in violations if
v.severity in ['high', 'critical']]
        }

    async def generate_audit_record(
        self,
        user_id: str,
        action_type: str,
        document_id: str,
        pii_entities: List,
        processing_metadata: Dict,
        case_id: Optional[str] = None
    ) -> AuditRecord:
        """Generate comprehensive audit record"""

        audit_id = self._generate_audit_id(user_id, document_id)

        # Assess compliance
        compliance_result = await self.assess_compliance(
            pii_entities, processing_metadata, {'user_id': user_id,
'case_id': case_id}
        )

        audit_record = AuditRecord(
            audit_id=audit_id,
            timestamp=datetime.utcnow(),
            user_id=user_id,
            action_type=action_type,
            document_id=document_id,
            case_id=case_id,
            pii_entities_detected=len(pii_entities),
```

```
classification_level=processing_metadata.get('classification_level',
'unclassified'),
            processing_time_ms=processing_metadata.get('processing_time_ms',
0),
            success=processing_metadata.get('success', True),
            violations=compliance_result['violations'],
            metadata={
                'compliance_score': compliance_result['compliance_score'],
                'document_type': processing_metadata.get('document_type'),
                'language_detected':
processing_metadata.get('language_detected'),
                'models_used': processing_metadata.get('models_used', []),
                'entity_breakdown':
processing_metadata.get('entity_breakdown', {}),
                'ip_address': processing_metadata.get('ip_address'),
                'user_agent': processing_metadata.get('user_agent')
            }
        )

        # Store audit record in secure database
        await self._store_audit_record(audit_record)

        return audit_record

    def _generate_audit_id(self, user_id: str, document_id: str) -> str:
        """Generate unique audit ID"""
        timestamp = datetime.utcnow().isoformat()
        content = f"{user_id}:{document_id}:{timestamp}"
        return hashlib.sha256(content.encode()).hexdigest()[:16]

    async def _check_privacy_act_compliance(
        self,
        pii_entities: List,
        document_metadata: Dict,
        processing_context: Dict
    ) -> List[ComplianceViolation]:
        """Check Privacy Act compliance"""

        violations = []

        # Check purpose limitation
        if not self._validate_purpose_limitation(processing_context):
            violations.append(ComplianceViolation(
                violation_type='purpose_limitation',
                severity='high',
                description='Personal information processed beyond stated
purposes',
                recommendation='Ensure processing aligns with stated
collection purposes',
                regulatory_reference='Privacy Act s.7'
            ))

        # Check minimal collection principle
        if not self._validate_minimal_collection(pii_entities,
document_metadata):
            violations.append(ComplianceViolation(
```

```
                violation_type='excessive_collection',
                severity='medium',
                description='More personal information collected than
necessary',
                recommendation='Review collection practices to minimize
personal information',
                regulatory_reference='Privacy Act s.4'
            ))

        # Check retention limits
        retention_violation = await
self._check_retention_compliance_detailed(document_metadata)
        if retention_violation:
            violations.append(retention_violation)

        return violations

    def _validate_purpose_limitation(self, processing_context: Dict) -> bool:
        """Validate that processing aligns with stated purposes"""

        # Define legitimate purposes for IRCC
        legitimate_purposes = [
            'immigration_assessment',
            'eligibility_determination',
            'identity_verification',
            'case_processing',
            'compliance_monitoring',
            'quality_assurance'
        ]

        stated_purpose = processing_context.get('purpose',
'immigration_assessment')
        return stated_purpose in legitimate_purposes

    def _validate_minimal_collection(self, pii_entities: List,
document_metadata: Dict) -> bool:
        """Validate minimal collection principle"""

        document_type = document_metadata.get('document_type', '')

        # Define necessary PII for different document types
        necessary_pii_by_type = {
            'work_permit_application': [
                'name', 'date_of_birth', 'passport_number', 'uci_number',
'address'
            ],
            'study_permit_application': [
                'name', 'date_of_birth', 'passport_number', 'uci_number',
'address', 'educational_institution'
            ],
            'permanent_residence_application': [
                'name', 'date_of_birth', 'passport_number', 'uci_number',
'address', 'country_of_birth'
            ],
            'supporting_document': [
                'name', 'date_of_birth', 'passport_number'
            ]
```

```
        }

        necessary_pii = necessary_pii_by_type.get(document_type,
necessary_pii_by_type['supporting_document'])
        detected_pii_types = [entity.entity_type for entity in pii_entities]

        # Check if we're collecting more than necessary
        unnecessary_pii = [pii_type for pii_type in detected_pii_types if
pii_type not in necessary_pii]

        return len(unnecessary_pii) <= 2  # Allow for some variation

    async def determine_retention_schedule(self, document_metadata: Dict) ->
Dict:
        """Determine appropriate retention schedule"""

        document_type = document_metadata.get('document_type',
'supporting_document')
        classification_level = document_metadata.get('classification_level',
'unclassified')

        # Get base retention policy
        retention_policy = self.retention_policies.get(document_type,
self.retention_policies['temporary_records'])

        # Adjust based on classification level
        if classification_level == ClassificationLevel.PROTECTED_B.value:
            # Extended retention for high-value records
            retention_policy['retention_period'] =
retention_policy['retention_period'] * 1.5

        # Calculate destruction date
        creation_date = document_metadata.get('created_at',
datetime.utcnow())
        destruction_date = creation_date +
retention_policy['retention_period']

        return {
            'retention_category': retention_policy['category'].value,
            'retention_period_days':
retention_policy['retention_period'].days,
            'destruction_date': destruction_date.isoformat(),
            'disposition_method': retention_policy['disposition'],
            'review_required': classification_level in
[ClassificationLevel.PROTECTED_A.value,
ClassificationLevel.PROTECTED_B.value]
        }
```

Security Implementation

Access Control and Authentication

```
# app/core/security.py
```

```
from typing import Optional, List
from datetime import datetime, timedelta
import jwt
from passlib.context import CryptContext
from fastapi import HTTPException, Depends, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
import logging

logger = logging.getLogger(__name__)

class IRCCRoleManager:
    """Role-based access control for IRCC users"""

    ROLES = {
        'immigration_officer': {
            'permissions': [
                'read:documents', 'process:applications', 'view:pii',
                'create:audit_logs', 'read:case_files'
            ],
            'classification_access': ['unclassified', 'protected_a']
        },
        'senior_officer': {
            'permissions': [
                'read:documents', 'process:applications', 'view:pii',
                'create:audit_logs', 'read:case_files',
'approve:applications',
                'access:compliance_reports'
            ],
            'classification_access': ['unclassified', 'protected_a',
'protected_b']
        },
        'privacy_officer': {
            'permissions': [
                'read:audit_logs', 'access:compliance_reports',
'view:pii_summary',
                'manage:retention_policies', 'investigate:violations'
            ],
            'classification_access': ['unclassified', 'protected_a',
'protected_b']
        },
        'system_admin': {
            'permissions': [
                'manage:system', 'view:performance_metrics',
'access:all_logs',
                'configure:settings'
            ],
            'classification_access': ['unclassified']
        }
    }

    @classmethod
    def has_permission(cls, user_role: str, required_permission: str) ->
bool:
        """Check if user role has required permission"""
        role_config = cls.ROLES.get(user_role, {})
        permissions = role_config.get('permissions', [])
        return required_permission in permissions
```

```
    @classmethod
    def can_access_classification(cls, user_role: str, classification_level:
str) -> bool:
        """Check if user can access given classification level"""
        role_config = cls.ROLES.get(user_role, {})
        classification_access = role_config.get('classification_access', [])
        return classification_level in classification_access

class IRCCSecurityManager:
    """Security manager for IRCC PII system"""

    def __init__(self, secret_key: str):
        self.secret_key = secret_key
        self.algorithm = "HS256"
        self.pwd_context = CryptContext(schemes=["bcrypt"],
deprecated="auto")
        self.security = HTTPBearer()

    def create_access_token(self, data: dict, expires_delta:
Optional[timedelta] = None):
        """Create JWT access token"""
        to_encode = data.copy()
        if expires_delta:
            expire = datetime.utcnow() + expires_delta
        else:
            expire = datetime.utcnow() + timedelta(hours=8)  # 8-hour work
day

        to_encode.update({"exp": expire})
        encoded_jwt = jwt.encode(to_encode, self.secret_key,
algorithm=self.algorithm)
        return encoded_jwt

    def verify_token(self, token: str) -> dict:
        """Verify and decode JWT token"""
        try:
            payload = jwt.decode(token, self.secret_key,
algorithms=[self.algorithm])
            return payload
        except jwt.ExpiredSignatureError:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail="Token has expired"
            )
        except jwt.JWTError:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail="Could not validate credentials"
            )

    async def get_current_user(self, credentials:
HTTPAuthorizationCredentials = Depends(HTTPBearer())):
        """Get current authenticated user"""
        token = credentials.credentials
        payload = self.verify_token(token)
```

```
        user_id = payload.get("sub")
        user_role = payload.get("role")

        if user_id is None:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail="Could not validate credentials"
            )

        return {
            "user_id": user_id,
            "role": user_role,
            "badge_number": payload.get("badge_number"),
            "classification_level": payload.get("classification_level",
"unclassified")
        }

def require_permission(required_permission: str):
    """Decorator to require specific permission"""
    def decorator(func):
        async def wrapper(*args, **kwargs):
            # Get current user from dependencies
            current_user = kwargs.get('current_user')
            if not current_user:
                raise HTTPException(
                    status_code=status.HTTP_401_UNAUTHORIZED,
                    detail="Authentication required"
                )

            user_role = current_user.get('role')
            if not IRCCRoleManager.has_permission(user_role,
required_permission):
                raise HTTPException(
                    status_code=status.HTTP_403_FORBIDDEN,
                    detail=f"Insufficient permissions. Required:
{required_permission}"
                )

            return await func(*args, **kwargs)
        return wrapper
    return decorator

def require_classification_access(classification_level: str):
    """Decorator to require access to specific classification level"""
    def decorator(func):
        async def wrapper(*args, **kwargs):
            current_user = kwargs.get('current_user')
            if not current_user:
                raise HTTPException(
                    status_code=status.HTTP_401_UNAUTHORIZED,
                    detail="Authentication required"
                )

            user_role = current_user.get('role')
            if not IRCCRoleManager.can_access_classification(user_role,
classification_level):
                raise HTTPException(
```

```
                    status_code=status.HTTP_403_FORBIDDEN,
                    detail=f"Insufficient classification access. Required:
{classification_level}"
                )

            return await func(*args, **kwargs)
        return wrapper
    return decorator
```

Encryption and Data Protection

```
# app/utils/encryption.py

from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
import base64
import os
import hashlib
import hmac
from typing import Union, bytes

class IRCCEncryptionManager:
    """Encryption manager for IRCC sensitive data"""

    def __init__(self, master_key: str):
        self.master_key = master_key.encode()
        self._fernet = self._create_fernet_instance()

    def _create_fernet_instance(self) -> Fernet:
        """Create Fernet instance with derived key"""
        salt = b'ircc_pii_salt_2024'  # Should be stored securely in
production
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=32,
            salt=salt,
            iterations=100000,
        )
        key = base64.urlsafe_b64encode(kdf.derive(self.master_key))
        return Fernet(key)

    def encrypt_pii(self, data: Union[str, bytes]) -> str:
        """Encrypt PII data"""
        if isinstance(data, str):
            data = data.encode()

        encrypted_data = self._fernet.encrypt(data)
        return base64.urlsafe_b64encode(encrypted_data).decode()

    def decrypt_pii(self, encrypted_data: str) -> str:
        """Decrypt PII data"""
        encrypted_bytes = base64.urlsafe_b64decode(encrypted_data.encode())
        decrypted_data = self._fernet.decrypt(encrypted_bytes)
        return decrypted_data.decode()
```

```
    def hash_pii(self, data: str, salt: Optional[str] = None) -> str:
        """Create irreversible hash of PII for storage/comparison"""
        if salt is None:
            salt = os.urandom(32)
        elif isinstance(salt, str):
            salt = salt.encode()

        return hashlib.pbkdf2_hmac('sha256', data.encode(), salt,
100000).hex()

    def create_secure_hash(self, data: str) -> str:
        """Create HMAC hash for data integrity"""
        return hmac.new(
            self.master_key,
            data.encode(),
            hashlib.sha256
        ).hexdigest()

    def verify_hash(self, data: str, expected_hash: str) -> bool:
        """Verify HMAC hash"""
        computed_hash = self.create_secure_hash(data)
        return hmac.compare_digest(computed_hash, expected_hash)
```

Testing Strategy

Unit Tests

```
# tests/unit/test_ircc_pii_detector.py

import pytest
import asyncio
from app.services.pii.ircc_detector import IRCCPIIDetector, IRCCDocument
from app.models.schemas.pii_detection import IRCCPIIEntity

class TestIRCCPIIDetector:

    @pytest.fixture
    def detector(self):
        return IRCCPIIDetector()

    @pytest.fixture
    def sample_document(self):
        return IRCCDocument(
            content="John Doe, UCI: 1234-5678, Work Permit: WP12345678",
            document_type="work_permit_application",
            language="english",
            case_id="CASE123",
            classification_level="protected_a",
            source_system="gcms"
        )

    @pytest.mark.asyncio
    async def test_uci_detection(self, detector):
        """Test UCI number detection"""
```

```
        content = "Client UCI number is 1234-5678"
        document = IRCCDocument(
            content=content,
            document_type="correspondence",
            language="english",
            case_id=None,
            classification_level="unclassified",
            source_system="test"
        )

        result = await detector.detect_pii(document, "test_user")

        # Should detect UCI number
        uci_entities = [e for e in result.entities if e.entity_type ==
'uci_number']
        assert len(uci_entities) == 1
        assert uci_entities[0].text == "1234-5678"
        assert uci_entities[0].confidence_score > 0.9

    @pytest.mark.asyncio
    async def test_sin_validation(self, detector):
        """Test SIN validation logic"""
        # Valid SIN
        valid_sin = "046-454-286"
        validation_result = detector._validate_sin(valid_sin)
        assert validation_result['valid'] == True

        # Invalid SIN
        invalid_sin = "123-456-789"
        validation_result = detector._validate_sin(invalid_sin)
        assert validation_result['valid'] == False

    @pytest.mark.asyncio
    async def test_multilingual_detection(self, detector):
        """Test PII detection in French content"""
        french_content = "Nom: Jean Dupont, Numéro de passeport: AB123456"
        document = IRCCDocument(
            content=french_content,
            document_type="supporting_document",
            language="french",
            case_id=None,
            classification_level="unclassified",
            source_system="test"
        )

        result = await detector.detect_pii(document, "test_user")

        # Should detect name and passport number
        assert len(result.entities) >= 2

    @pytest.mark.asyncio
    async def test_anonymization_strategies(self, detector, sample_document):
        """Test different anonymization strategies"""
        result = await detector.detect_pii(sample_document, "test_user")

        # Verify anonymized content doesn't contain original PII
        assert "1234-5678" not in result.anonymized_content
```

```
        assert "WP12345678" not in result.anonymized_content
        assert "[UCI]" in result.anonymized_content or "[REDACTED]" in
result.anonymized_content

    def test_performance_requirements(self, detector):
        """Test that detection meets performance requirements"""
        import time

        content = "Large document with multiple PII entities..." * 100
        document = IRCCDocument(
            content=content,
            document_type="application",
            language="english",
            case_id="PERF_TEST",
            classification_level="unclassified",
            source_system="test"
        )

        start_time = time.time()

        # Run detection
        loop = asyncio.get_event_loop()
        result = loop.run_until_complete(detector.detect_pii(document,
"test_user"))

        processing_time = (time.time() - start_time) * 1000  # Convert to ms

        # Should complete within 500ms for standard documents
        assert processing_time < 500
```

Integration Tests

```
# tests/integration/test_full_pipeline.py

import pytest
import asyncio
from fastapi.testclient import TestClient
from app.main import app
from app.services.pii.ircc_detector import IRCCPIIDetector
from app.services.document.ircc_processor import IRCCDocumentProcessor

class TestFullPipeline:

    @pytest.fixture
    def client(self):
        return TestClient(app)

    @pytest.fixture
    def auth_headers(self):
        # Mock authentication for testing
        return {"Authorization": "Bearer test_token"}

    def test_document_upload_and_processing(self, client, auth_headers):
        """Test complete document processing pipeline"""

        # Prepare test document
```

```
        test_content = b"John Doe, UCI: 1234-5678, SIN: 046-454-286"

        response = client.post(
            "/api/v1/documents/process",
            files={"file": ("test.txt", test_content, "text/plain")},
            data={
                "case_id": "TEST123",
                "classification_level": "protected_a"
            },
            headers=auth_headers
        )

        assert response.status_code == 200
        result = response.json()

        # Verify response structure
        assert "document_id" in result
        assert "entities" in result
        assert "anonymized_content" in result
        assert "compliance_status" in result

        # Verify PII detection
        entities = result["entities"]
        assert len(entities) >= 2  # Should detect UCI and SIN

        # Verify anonymization
        assert "1234-5678" not in result["anonymized_content"]
        assert "046-454-286" not in result["anonymized_content"]

    def test_batch_processing(self, client, auth_headers):
        """Test batch document processing"""

        documents = [
            {"filename": "doc1.txt", "content": "UCI: 1111-2222"},
            {"filename": "doc2.txt", "content": "SIN: 046-454-286"},
            {"filename": "doc3.txt", "content": "Passport: AB123456"}
        ]

        response = client.post(
            "/api/v1/documents/batch-process",
            json={"documents": documents, "case_id": "BATCH123"},
            headers=auth_headers
        )

        assert response.status_code == 200
        results = response.json()

        assert len(results["results"]) == 3
        for result in results["results"]:
            assert "entities" in result
            assert len(result["entities"]) >= 1

    def test_compliance_reporting(self, client, auth_headers):
        """Test compliance reporting functionality"""

        response = client.get(
            "/api/v1/compliance/report",
```

```
            params={"start_date": "2024-01-01", "end_date": "2024-12-31"},
            headers=auth_headers
        )

        assert response.status_code == 200
        report = response.json()

        # Verify report structure
        assert "total_documents_processed" in report
        assert "compliance_score" in report
        assert "violations" in report
        assert "recommendations" in report
```

Performance Tests

```
# tests/performance/test_load_performance.py

import pytest
import asyncio
import time
from concurrent.futures import ThreadPoolExecutor
from app.services.pii.ircc_detector import IRCCPIIDetector, IRCCDocument

class TestPerformance:

    @pytest.fixture
    def detector(self):
        return IRCCPIIDetector()

    @pytest.mark.asyncio
    async def test_concurrent_processing(self, detector):
        """Test system under concurrent load"""

        # Create test documents
        documents = []
        for i in range(100):
            documents.append(IRCCDocument(
                content=f"Document {i}: UCI 1234-567{i}, SIN 046-454-28{i %
10}",
                document_type="application",
                language="english",
                case_id=f"LOAD_TEST_{i}",
                classification_level="unclassified",
                source_system="load_test"
            ))

        start_time = time.time()

        # Process documents concurrently
        tasks = [
            detector.detect_pii(doc, f"test_user_{i}")
            for i, doc in enumerate(documents)
        ]

        results = await asyncio.gather(*tasks)
```

```
        total_time = time.time() - start_time

        # Verify all documents processed successfully
        assert len(results) == 100
        for result in results:
            assert len(result.entities) >= 2  # UCI and SIN

        # Performance requirements
        avg_time_per_doc = total_time / 100
        assert avg_time_per_doc < 0.5  # Less than 500ms per document

        print(f"Processed 100 documents in {total_time:.2f}s")
        print(f"Average time per document: {avg_time_per_doc*1000:.2f}ms")

    def test_memory_usage(self, detector):
        """Test memory usage under load"""
        import psutil
        import os

        process = psutil.Process(os.getpid())
        initial_memory = process.memory_info().rss

        # Process large document
        large_content = "Test content with PII: UCI 1234-5678. " * 10000
        document = IRCCDocument(
            content=large_content,
            document_type="large_document",
            language="english",
            case_id="MEMORY_TEST",
            classification_level="unclassified",
            source_system="memory_test"
        )

        # Process multiple times
        loop = asyncio.get_event_loop()
        for i in range(10):
            result = loop.run_until_complete(detector.detect_pii(document,
f"memory_test_{i}"))

        final_memory = process.memory_info().rss
        memory_increase = final_memory - initial_memory

        # Memory increase should be reasonable (less than 100MB)
        assert memory_increase < 100 * 1024 * 1024

        print(f"Memory increase: {memory_increase / 1024 / 1024:.2f}MB")
```

Deployment Guide

Docker Configuration

```
# Dockerfile for IRCC PII Detection System

FROM python:3.11-slim
```

```
# Set environment variables
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
ENV DEBIAN_FRONTEND=noninteractive

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    libpq-dev \
    libmagic1 \
    tesseract-ocr \
    tesseract-ocr-eng \
    tesseract-ocr-fra \
    poppler-utils \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Create app user
RUN useradd --create-home --shell /bin/bash app

# Set work directory
WORKDIR /app

# Copy requirements and install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Download spaCy models
RUN python -m spacy download en_core_web_sm
RUN python -m spacy download fr_core_news_sm

# Create necessary directories
RUN mkdir -p /app/data/models /app/data/uploads /app/logs /app/audit

# Copy application code
COPY . .

# Set ownership
RUN chown -R app:app /app

# Switch to app user
USER app

# Expose port
EXPOSE 8000

# Health check
HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

# Start command
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--
workers", "4"]
```

Kubernetes Deployment

```
# k8s/ircc-pii-deployment.yaml

apiVersion: v1
kind: Namespace
metadata:
  name: ircc-pii-system
  labels:
    name: ircc-pii-system
    classification: protected-a

---
apiVersion: v1
kind: Secret
metadata:
  name: ircc-pii-secrets
  namespace: ircc-pii-system
type: Opaque
data:
  database-url: <base64-encoded-database-url>
  encryption-key: <base64-encoded-encryption-key>
  jwt-secret: <base64-encoded-jwt-secret>

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: ircc-pii-config
  namespace: ircc-pii-system
data:
  ENVIRONMENT: "production"
  LOG_LEVEL: "INFO"
  CLASSIFICATION_DEFAULT: "protected_a"
  AUDIT_RETENTION_DAYS: "3650"
  MAX_DOCUMENT_SIZE: "50MB"
  PROCESSING_TIMEOUT: "300"

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ircc-pii-backend
  namespace: ircc-pii-system
  labels:
    app: ircc-pii-backend
    classification: protected-a
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  selector:
    matchLabels:
```

```
    app: ircc-pii-backend
template:
  metadata:
    labels:
      app: ircc-pii-backend
      classification: protected-a
  spec:
    serviceAccountName: ircc-pii-service-account
    securityContext:
      runAsNonRoot: true
      runAsUser: 1000
      fsGroup: 1000
    containers:
    - name: ircc-pii-backend
      image: ircc-pii-backend:latest
      imagePullPolicy: Always
      ports:
      - containerPort: 8000
        name: http
      env:
      - name: DATABASE_URL
        valueFrom:
          secretKeyRef:
            name: ircc-pii-secrets
            key: database-url
      - name: ENCRYPTION_KEY
        valueFrom:
          secretKeyRef:
            name: ircc-pii-secrets
            key: encryption-key
      - name: JWT_SECRET
        valueFrom:
          secretKeyRef:
            name: ircc-pii-secrets
            key: jwt-secret
      envFrom:
      - configMapRef:
          name: ircc-pii-config
      resources:
        requests:
          memory: "2Gi"
          cpu: "1000m"
        limits:
          memory: "4Gi"
          cpu: "2000m"
      livenessProbe:
        httpGet:
          path: /health
          port: 8000
        initialDelaySeconds: 30
        periodSeconds: 10
        timeoutSeconds: 5
        failureThreshold: 3
      readinessProbe:
        httpGet:
          path: /health/ready
          port: 8000
```

```
          initialDelaySeconds: 5
          periodSeconds: 5
          timeoutSeconds: 3
          failureThreshold: 3
        volumeMounts:
        - name: model-storage
          mountPath: /app/data/models
          readOnly: true
        - name: upload-storage
          mountPath: /app/data/uploads
        - name: audit-logs
          mountPath: /app/audit
        securityContext:
          allowPrivilegeEscalation: false
          readOnlyRootFilesystem: true
          capabilities:
            drop:
            - ALL
      volumes:
      - name: model-storage
        persistentVolumeClaim:
          claimName: ircc-model-storage
      - name: upload-storage
        persistentVolumeClaim:
          claimName: ircc-upload-storage
      - name: audit-logs
        persistentVolumeClaim:
          claimName: ircc-audit-logs
      nodeSelector:
        security-level: high
      tolerations:
      - key: "security-level"
        operator: "Equal"
        value: "high"
        effect: "NoSchedule"

---
apiVersion: v1
kind: Service
metadata:
  name: ircc-pii-backend-service
  namespace: ircc-pii-system
spec:
  selector:
    app: ircc-pii-backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8000
  type: ClusterIP

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ircc-pii-service-account
  namespace: ircc-pii-system
```

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ircc-pii-network-policy
  namespace: ircc-pii-system
spec:
  podSelector:
    matchLabels:
      app: ircc-pii-backend
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ircc-frontend
    ports:
    - protocol: TCP
      port: 8000
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: ircc-database
    ports:
    - protocol: TCP
      port: 5432
```

Performance Benchmarks

Target Performance Metrics

```
# Performance benchmarks and monitoring

PERFORMANCE_TARGETS = {
    'processing_time': {
        'small_document': '<100ms',    # < 1KB text
        'medium_document': '<300ms',   # 1-10KB text
        'large_document': '<500ms',    # 10-50KB text
        'batch_processing': '<5s',     # 100 documents
    },
    'throughput': {
        'concurrent_users': 200,       # Simultaneous users
        'documents_per_minute': 120,   # Sustained processing
        'peak_load': 300,              # Maximum burst capacity
    },
    'accuracy': {
        'pii_detection': '95%',        # Overall detection accuracy
        'false_positive_rate': '<5%',  # Incorrect PII identification
        'false_negative_rate': '<3%',  # Missed PII entities
    },
```

```
    'availability': {
        'uptime': '99.9%',              # System availability
        'recovery_time': '<4h',         # Maximum downtime
        'failover_time': '<30s',        # Automatic failover
    },
    'resource_usage': {
        'memory_per_request': '<100MB', # Memory usage per request
        'cpu_per_request': '<500ms',    # CPU time per request
        'storage_growth': '<1GB/day',   # Audit log storage growth
    },
    'security': {
        'encryption_overhead': '<10%',  # Performance impact of encryption
        'audit_logging_overhead': '<5%', # Performance impact of auditing
        'authentication_time': '<100ms', # User authentication time
    }
}

# Monitoring and alerting configuration
MONITORING_CONFIG = {
    'metrics': {
        'pii_detection_accuracy': {
            'threshold': 0.95,
            'alert_below': 0.90,
            'measurement_window': '1h'
        },
        'processing_latency': {
            'threshold_p95': 500,  # 95th percentile in ms
            'threshold_p99': 1000, # 99th percentile in ms
            'alert_above': 750
        },
        'error_rate': {
            'threshold': 0.01,  # 1% error rate
            'alert_above': 0.05, # 5% error rate
            'measurement_window': '5m'
        },
        'memory_usage': {
            'threshold': 0.80,  # 80% of allocated memory
            'alert_above': 0.90, # 90% of allocated memory
        },
        'compliance_violations': {
            'threshold': 0,     # Zero tolerance for critical violations
            'alert_above': 1,   # Any critical violation triggers alert
        }
    },
    'dashboards': [
        'real_time_processing_metrics',
        'compliance_monitoring',
        'security_events',
        'system_performance',
        'audit_trail_analysis'
    ]
}
```

Load Testing Configuration

```
# tests/performance/load_test_config.py
```

```
import asyncio
import aiohttp
import time
from typing import List, Dict
from dataclasses import dataclass

@dataclass
class LoadTestResult:
    total_requests: int
    successful_requests: int
    failed_requests: int
    average_response_time: float
    p95_response_time: float
    p99_response_time: float
    requests_per_second: float
    error_rate: float

class IRCCLoadTester:
    """Load testing framework for IRCC PII system"""

    def __init__(self, base_url: str, auth_token: str):
        self.base_url = base_url
        self.auth_token = auth_token
        self.session = None

    async def setup(self):
        """Initialize load testing session"""
        connector = aiohttp.TCPConnector(limit=300, limit_per_host=100)
        timeout = aiohttp.ClientTimeout(total=30)
        self.session = aiohttp.ClientSession(
            connector=connector,
            timeout=timeout,
            headers={'Authorization': f'Bearer {self.auth_token}'}
        )

    async def teardown(self):
        """Clean up load testing session"""
        if self.session:
            await self.session.close()

    async def test_pii_detection_endpoint(
        self,
        concurrent_users: int = 100,
        requests_per_user: int = 10,
        test_duration: int = 60
    ) -> LoadTestResult:
        """Load test the PII detection endpoint"""

        test_documents = [
            "John Doe, UCI: 1234-5678, SIN: 046-454-286",
            "Application for Work Permit WP12345678",
            "Passport Number: AB123456, Country: Canada",
            "Study Permit SP87654321 for University of Toronto",
            "Permanent Resident Card PR11223344"
        ]
```

```
response_times = []
successful_requests = 0
failed_requests = 0

async def make_request(session, document_content):
    """Make a single PII detection request"""
    start_time = time.time()
    try:
        async with session.post(
            f"{self.base_url}/api/v1/pii/detect",
            json={
                "content": document_content,
                "document_type": "application",
                "classification_level": "protected_a"
            }
        ) as response:
            response_time = (time.time() - start_time) * 1000
            response_times.append(response_time)

            if response.status == 200:
                successful_requests += 1
            else:
                failed_requests += 1

    except Exception as e:
        response_time = (time.time() - start_time) * 1000
        response_times.append(response_time)
        failed_requests += 1

# Create load test tasks
tasks = []
for user_id in range(concurrent_users):
    for request_id in range(requests_per_user):
        document = test_documents[request_id % len(test_documents)]
        task = make_request(self.session, document)
        tasks.append(task)

# Execute load test
start_time = time.time()
await asyncio.gather(*tasks)
total_time = time.time() - start_time

# Calculate metrics
total_requests = len(tasks)
average_response_time = sum(response_times) / len(response_times)

response_times.sort()
p95_index = int(len(response_times) * 0.95)
p99_index = int(len(response_times) * 0.99)

return LoadTestResult(
    total_requests=total_requests,
    successful_requests=successful_requests,
    failed_requests=failed_requests,
    average_response_time=average_response_time,
    p95_response_time=response_times[p95_index],
    p99_response_time=response_times[p99_index],
```

```
            requests_per_second=total_requests / total_time,
            error_rate=failed_requests / total_requests
        )

# Performance test scenarios
LOAD_TEST_SCENARIOS = {
    'baseline_load': {
        'concurrent_users': 50,
        'requests_per_user': 20,
        'test_duration': 300,  # 5 minutes
        'expected_rps': 10
    },
    'peak_load': {
        'concurrent_users': 200,
        'requests_per_user': 10,
        'test_duration': 600,  # 10 minutes
        'expected_rps': 30
    },
    'stress_test': {
        'concurrent_users': 500,
        'requests_per_user': 5,
        'test_duration': 300,  # 5 minutes
        'expected_rps': 50
    },
    'endurance_test': {
        'concurrent_users': 100,
        'requests_per_user': 100,
        'test_duration': 3600,  # 1 hour
        'expected_rps': 15
    }
}
```

Compliance Framework

Regulatory Compliance Matrix

```
# app/services/compliance/regulatory_matrix.py

from typing import Dict, List, Optional
from dataclasses import dataclass
from enum import Enum

class RegulatoryFramework(Enum):
    PRIVACY_ACT = "privacy_act"
    PIPEDA = "pipeda"
    ACCESS_TO_INFORMATION = "access_to_information"
    GOVERNMENT_SECURITY_POLICY = "government_security_policy"
    TB_SECRETARIAT = "treasury_board_secretariat"

@dataclass
class ComplianceRequirement:
    requirement_id: str
    framework: RegulatoryFramework
    title: str
```

```
    description: str
    mandatory: bool
    implementation_guidance: str
    validation_method: str
    evidence_required: List[str]
    responsible_role: str

class IRCCRegulatoryMatrix:
    """Comprehensive regulatory compliance matrix for IRCC PII system"""

    def __init__(self):
        self.requirements = self._initialize_requirements()

    def _initialize_requirements(self) -> Dict[str, ComplianceRequirement]:
        """Initialize all regulatory requirements"""

        requirements = {}

        # Privacy Act Requirements
        privacy_act_reqs = [
            ComplianceRequirement(
                requirement_id="PA-001",
                framework=RegulatoryFramework.PRIVACY_ACT,
                title="Purpose Limitation",
                description="Personal information must be collected and used
only for purposes authorized by law",
                mandatory=True,
                implementation_guidance="Implement purpose validation in PII
processing pipeline",
                validation_method="Automated validation against predefined
purpose list",
                evidence_required=["purpose_validation_logs",
"processing_audit_trail"],
                responsible_role="privacy_officer"
            ),
            ComplianceRequirement(
                requirement_id="PA-002",
                framework=RegulatoryFramework.PRIVACY_ACT,
                title="Accuracy and Currency",
                description="Personal information must be accurate, complete,
and current",
                mandatory=True,
                implementation_guidance="Implement data quality checks and
validation rules",
                validation_method="Data quality metrics and validation
reports",
                evidence_required=["data_quality_reports",
"validation_error_logs"],
                responsible_role="data_steward"
            ),
            ComplianceRequirement(
                requirement_id="PA-003",
                framework=RegulatoryFramework.PRIVACY_ACT,
                title="Security Safeguards",
                description="Appropriate security measures to protect
personal information",
                mandatory=True,
```

```
                implementation_guidance="Implement encryption, access
controls, and audit logging",
                validation_method="Security assessment and penetration
testing",
                evidence_required=["security_assessment_report",
"encryption_verification"],
                responsible_role="security_officer"
            ),
            ComplianceRequirement(
                requirement_id="PA-004",
                framework=RegulatoryFramework.PRIVACY_ACT,
                title="Retention and Disposal",
                description="Personal information retained only as long as
necessary",
                mandatory=True,
                implementation_guidance="Implement automated retention policy
enforcement",
                validation_method="Retention policy compliance audits",
                evidence_required=["retention_policy_documents",
"disposal_certificates"],
                responsible_role="records_manager"
            )
        ]

        # PIPEDA Requirements (when applicable)
        pipeda_reqs = [
            ComplianceRequirement(
                requirement_id="PIPEDA-001",
                framework=RegulatoryFramework.PIPEDA,
                title="Consent Requirements",
                description="Meaningful consent for collection, use, and
disclosure of personal information",
                mandatory=True,
                implementation_guidance="Implement granular consent
management system",
                validation_method="Consent audit trail and validation",
                evidence_required=["consent_records",
"consent_withdrawal_logs"],
                responsible_role="privacy_officer"
            ),
            ComplianceRequirement(
                requirement_id="PIPEDA-002",
                framework=RegulatoryFramework.PIPEDA,
                title="Individual Access",
                description="Individuals can access their personal
information",
                mandatory=True,
                implementation_guidance="Implement individual access request
processing",
                validation_method="Access request fulfillment metrics",
                evidence_required=["access_request_logs",
"response_time_metrics"],
                responsible_role="privacy_officer"
            )
        ]

        # Government Security Policy Requirements
```

```
        security_reqs = [
            ComplianceRequirement(
                requirement_id="GSP-001",
                framework=RegulatoryFramework.GOVERNMENT_SECURITY_POLICY,
                title="Information Classification",
                description="Proper classification and handling of government
information",
                mandatory=True,
                implementation_guidance="Implement automated classification
and handling controls",
                validation_method="Classification accuracy assessment",
                evidence_required=["classification_audit",
"handling_compliance_report"],
                responsible_role="security_officer"
            ),
            ComplianceRequirement(
                requirement_id="GSP-002",
                framework=RegulatoryFramework.GOVERNMENT_SECURITY_POLICY,
                title="Personnel Security",
                description="Appropriate security screening for personnel
accessing classified information",
                mandatory=True,
                implementation_guidance="Implement role-based access control
with security clearance validation",
                validation_method="Access control audit and clearance
verification",
                evidence_required=["clearance_verification_logs",
"access_control_audit"],
                responsible_role="security_officer"
            )
        ]

        # Combine all requirements
        all_reqs = privacy_act_reqs + pipeda_reqs + security_reqs

        for req in all_reqs:
            requirements[req.requirement_id] = req

        return requirements

    def get_requirements_by_framework(self, framework: RegulatoryFramework) -
> List[ComplianceRequirement]:
        """Get all requirements for a specific framework"""
        return [req for req in self.requirements.values() if req.framework ==
framework]

    def get_mandatory_requirements(self) -> List[ComplianceRequirement]:
        """Get all mandatory requirements"""
        return [req for req in self.requirements.values() if req.mandatory]

    def generate_compliance_checklist(self) -> Dict:
        """Generate compliance checklist for implementation"""
        checklist = {}

        for framework in RegulatoryFramework:
            framework_reqs = self.get_requirements_by_framework(framework)
            checklist[framework.value] = {
```

```
                'total_requirements': len(framework_reqs),
                'mandatory_requirements': len([req for req in framework_reqs
if req.mandatory]),
                'requirements': [
                    {
                        'id': req.requirement_id,
                        'title': req.title,
                        'mandatory': req.mandatory,
                        'implementation_guidance':
req.implementation_guidance,
                        'evidence_required': req.evidence_required
                    }
                    for req in framework_reqs
                ]
            }

        return checklist
```

Compliance Monitoring and Reporting

```
# app/services/compliance/monitoring.py

from typing import Dict, List, Optional
from datetime import datetime, timedelta
import asyncio
from dataclasses import dataclass

@dataclass
class ComplianceMetric:
    metric_name: str
    current_value: float
    target_value: float
    threshold_warning: float
    threshold_critical: float
    measurement_unit: str
    last_updated: datetime

@dataclass
class ComplianceReport:
    report_id: str
    report_period: Dict
    overall_score: float
    framework_scores: Dict[str, float]
    violations: List[Dict]
    recommendations: List[str]
    trends: Dict[str, List[float]]
    generated_at: datetime

class IRCCComplianceMonitor:
    """Real-time compliance monitoring for IRCC PII system"""

    def __init__(self):
        self.metrics = self._initialize_metrics()
        self.violation_thresholds = self._initialize_thresholds()
        self.monitoring_active = False
```

```
def _initialize_metrics(self) -> Dict[str, ComplianceMetric]:
    """Initialize compliance metrics"""
    return {
        'pii_detection_accuracy': ComplianceMetric(
            metric_name='PII Detection Accuracy',
            current_value=0.0,
            target_value=95.0,
            threshold_warning=90.0,
            threshold_critical=85.0,
            measurement_unit='percentage',
            last_updated=datetime.utcnow()
        ),
        'audit_trail_completeness': ComplianceMetric(
            metric_name='Audit Trail Completeness',
            current_value=0.0,
            target_value=100.0,
            threshold_warning=99.0,
            threshold_critical=95.0,
            measurement_unit='percentage',
            last_updated=datetime.utcnow()
        ),
        'data_retention_compliance': ComplianceMetric(
            metric_name='Data Retention Compliance',
            current_value=0.0,
            target_value=100.0,
            threshold_warning=98.0,
            threshold_critical=95.0,
            measurement_unit='percentage',
            last_updated=datetime.utcnow()
        ),
        'encryption_coverage': ComplianceMetric(
            metric_name='Encryption Coverage',
            current_value=0.0,
            target_value=100.0,
            threshold_warning=99.0,
            threshold_critical=95.0,
            measurement_unit='percentage',
            last_updated=datetime.utcnow()
        ),
        'access_control_violations': ComplianceMetric(
            metric_name='Access Control Violations',
            current_value=0.0,
            target_value=0.0,
            threshold_warning=1.0,
            threshold_critical=5.0,
            measurement_unit='count_per_day',
            last_updated=datetime.utcnow()
        ),
        'privacy_breach_incidents': ComplianceMetric(
            metric_name='Privacy Breach Incidents',
            current_value=0.0,
            target_value=0.0,
            threshold_warning=0.0,
            threshold_critical=1.0,
            measurement_unit='count_per_month',
            last_updated=datetime.utcnow()
        )
```

```
        }

    async def start_monitoring(self):
        """Start continuous compliance monitoring"""
        self.monitoring_active = True

        # Start monitoring tasks
        monitoring_tasks = [
            self._monitor_pii_detection_accuracy(),
            self._monitor_audit_trail_completeness(),
            self._monitor_data_retention_compliance(),
            self._monitor_access_control_violations(),
            self._monitor_encryption_coverage(),
            self._generate_periodic_reports()
        ]

        await asyncio.gather(*monitoring_tasks)

    async def _monitor_pii_detection_accuracy(self):
        """Monitor PII detection accuracy in real-time"""
        while self.monitoring_active:
            try:
                # Query recent PII detection results
                accuracy = await self._calculate_pii_accuracy()

                metric = self.metrics['pii_detection_accuracy']
                metric.current_value = accuracy
                metric.last_updated = datetime.utcnow()

                # Check thresholds and generate alerts
                if accuracy < metric.threshold_critical:
                    await self._trigger_alert('critical', f'PII detection
accuracy below critical threshold: {accuracy}%')
                elif accuracy < metric.threshold_warning:
                    await self._trigger_alert('warning', f'PII detection
accuracy below warning threshold: {accuracy}%')

                # Wait before next check
                await asyncio.sleep(300)  # Check every 5 minutes

            except Exception as e:
                logger.error(f"Error monitoring PII detection accuracy: {e}")
                await asyncio.sleep(60)

    async def _monitor_audit_trail_completeness(self):
        """Monitor audit trail completeness"""
        while self.monitoring_active:
            try:
                # Check audit log completeness
                completeness = await self._calculate_audit_completeness()

                metric = self.metrics['audit_trail_completeness']
                metric.current_value = completeness
                metric.last_updated = datetime.utcnow()

                if completeness < metric.threshold_critical:
```

```
                    await self._trigger_alert('critical', f'Audit trail
completeness below threshold: {completeness}%')

                await asyncio.sleep(600)  # Check every 10 minutes

            except Exception as e:
                logger.error(f"Error monitoring audit trail: {e}")
                await asyncio.sleep(60)

    async def generate_compliance_report(
        self,
        start_date: datetime,
        end_date: datetime,
        report_type: str = 'comprehensive'
    ) -> ComplianceReport:
        """Generate comprehensive compliance report"""

        report_id = f"IRCC-COMP-{datetime.utcnow().strftime('%Y%m%d%H%M%S')}"

        # Calculate overall compliance score
        overall_score = await
self._calculate_overall_compliance_score(start_date, end_date)

        # Calculate framework-specific scores
        framework_scores = await self._calculate_framework_scores(start_date,
end_date)

        # Gather violations
        violations = await self._gather_violations(start_date, end_date)

        # Generate recommendations
        recommendations = await self._generate_recommendations(violations,
framework_scores)

        # Calculate trends
        trends = await self._calculate_compliance_trends(start_date,
end_date)

        return ComplianceReport(
            report_id=report_id,
            report_period={
                'start_date': start_date.isoformat(),
                'end_date': end_date.isoformat(),
                'period_days': (end_date - start_date).days
            },
            overall_score=overall_score,
            framework_scores=framework_scores,
            violations=violations,
            recommendations=recommendations,
            trends=trends,
            generated_at=datetime.utcnow()
        )

    async def _calculate_overall_compliance_score(self, start_date: datetime,
end_date: datetime) -> float:
        """Calculate overall compliance score"""
```

```
        # Weight different compliance areas
        weights = {
            'privacy_act': 0.35,
            'security_policy': 0.25,
            'data_protection': 0.20,
            'audit_compliance': 0.20
        }

        # Calculate weighted score
        privacy_score = await
self._calculate_privacy_act_compliance(start_date, end_date)
        security_score = await
self._calculate_security_compliance(start_date, end_date)
        data_protection_score = await
self._calculate_data_protection_compliance(start_date, end_date)
        audit_score = await self._calculate_audit_compliance(start_date,
end_date)

        overall_score = (
            privacy_score * weights['privacy_act'] +
            security_score * weights['security_policy'] +
            data_protection_score * weights['data_protection'] +
            audit_score * weights['audit_compliance']
        )

        return round(overall_score, 2)

    async def _generate_recommendations(self, violations: List[Dict],
framework_scores: Dict[str, float]) -> List[str]:
        """Generate compliance improvement recommendations"""

        recommendations = []

        # Analyze violations for patterns
        violation_types = {}
        for violation in violations:
            v_type = violation.get('violation_type', 'unknown')
            violation_types[v_type] = violation_types.get(v_type, 0) + 1

        # Generate recommendations based on violations
        if violation_types.get('purpose_limitation', 0) > 0:
            recommendations.append("Strengthen purpose validation controls to
ensure PII processing aligns with stated purposes")

        if violation_types.get('retention_policy', 0) > 0:
            recommendations.append("Implement automated data retention policy
enforcement and monitoring")

        if violation_types.get('access_control', 0) > 0:
            recommendations.append("Review and strengthen role-based access
controls and user authentication")

        # Analyze framework scores
        if framework_scores.get('privacy_act', 100) < 90:
            recommendations.append("Conduct comprehensive Privacy Act
compliance review and remediation")
```

```
        if framework_scores.get('security_policy', 100) < 95:
            recommendations.append("Enhance security controls and conduct
security assessment")

        # Add proactive recommendations
        recommendations.extend([
            "Conduct regular privacy impact assessments for new features",
            "Implement continuous compliance monitoring and alerting",
            "Provide regular privacy and security training for all users",
            "Establish incident response procedures for privacy breaches"
        ])

        return recommendations[:10]  # Limit to top 10 recommendations
```

Implementation Checklist

Development Phase Checklist

```
## Phase 1: Foundation (Weeks 1-4)

### Week 1: Project Setup
- [ ] Initialize FastAPI project with proper structure
- [ ] Set up development environment with Docker
- [ ] Configure PostgreSQL with encryption at rest
- [ ] Implement JWT-based authentication system
- [ ] Set up structured logging with audit capabilities
- [ ] Create database schema for audit logs
- [ ] Implement basic error handling and validation
- [ ] Set up CI/CD pipeline with security scanning

### Week 2: Core PII Detection
- [ ] Implement rule-based pattern detection for all IRCC PII types
- [ ] Create pattern validation functions (UCI, SIN, passport numbers)
- [ ] Set up Hugging Face Transformers pipeline
- [ ] Implement basic document processing (text, PDF)
- [ ] Create PII entity data structures and schemas
- [ ] Add input validation and sanitization
- [ ] Implement basic anonymization strategies
- [ ] Create unit tests for pattern detection

### Week 3: ML Model Integration
- [ ] Download and configure Stanford deidentifier model
- [ ] Integrate multilingual NER models (French, other languages)
- [ ] Implement confidence scoring and validation
- [ ] Create model caching and optimization
- [ ] Add performance monitoring for model inference
- [ ] Implement fallback mechanisms for model failures
- [ ] Create integration tests for ML components
- [ ] Document model selection and configuration

### Week 4: API Development
- [ ] Create core PII detection endpoints
- [ ] Implement document upload and processing endpoints
- [ ] Add comprehensive audit logging
```

- [ ] Create health check and monitoring endpoints
- [ ] Implement rate limiting and security middleware
- [ ] Add request/response validation
- [ ] Create API documentation with examples
- [ ] Implement error handling and status codes

Phase 2: Enhancement (Weeks 5-8)

Week 5: IRCC-Specific Features
- [ ] Implement UCI validation with checksum algorithm
- [ ] Add immigration file number pattern recognition
- [ ] Create work/study permit number validation
- [ ] Implement passport number detection with country validation
- [ ] Add Canadian postal code validation
- [ ] Implement provincial health card detection
- [ ] Create document type classification
- [ ] Add Canadian government specific PII patterns

Week 6: Document Processing
- [ ] Implement advanced PDF processing (text + OCR)
- [ ] Add Microsoft Word document support
- [ ] Integrate OCR for scanned documents
- [ ] Create email message processing
- [ ] Implement image file processing
- [ ] Add document metadata extraction
- [ ] Create file format validation
- [ ] Implement large file handling

Week 7: Multilingual Support
- [ ] Integrate French language NER models
- [ ] Add automatic language detection
- [ ] Implement Unicode and script handling
- [ ] Create multilingual PII pattern recognition
- [ ] Add support for immigrant languages (Spanish, Arabic, Chinese)
- [ ] Implement context-aware language switching
- [ ] Create multilingual test datasets
- [ ] Document language support capabilities

Week 8: Anonymization Engine
- [ ] Implement context-aware anonymization strategies
- [ ] Create format-preserving anonymization
- [ ] Add reversible anonymization for authorized access
- [ ] Implement anonymization quality validation
- [ ] Create anonymization policy configuration
- [ ] Add anonymization effectiveness metrics
- [ ] Implement custom anonymization rules
- [ ] Create anonymization audit trail

Phase 3: Integration & Compliance (Weeks 9-12)

Week 9: GCMS Integration
- [ ] Create GCMS API connection framework
- [ ] Implement case management integration
- [ ] Add officer workflow support
- [ ] Create document retrieval from GCMS
- [ ] Implement status synchronization
- [ ] Add error handling for GCMS failures

- [ ] Create GCMS integration testing
- [ ] Document integration procedures

Week 10: Compliance Framework
- [ ] Implement Privacy Act compliance validation
- [ ] Add PIPEDA compliance checks
- [ ] Create comprehensive audit trail generation
- [ ] Implement data retention policy enforcement
- [ ] Add compliance violation detection
- [ ] Create compliance reporting dashboard
- [ ] Implement compliance metrics collection
- [ ] Add compliance alerting system

Week 11: Security Implementation
- [ ] Implement AES-256 encryption for data at rest
- [ ] Add TLS 1.3 for data in transit
- [ ] Create role-based access control system
- [ ] Implement multi-factor authentication
- [ ] Add security monitoring and alerting
- [ ] Create audit log protection and integrity
- [ ] Implement intrusion detection
- [ ] Add security incident response procedures

Week 12: Performance Optimization
- [ ] Optimize ML model inference performance
- [ ] Implement Redis caching strategies
- [ ] Add database query optimization
- [ ] Create load balancing configuration
- [ ] Implement connection pooling
- [ ] Add performance monitoring dashboard
- [ ] Create performance alerting
- [ ] Document performance tuning procedures

Phase 4: Deployment & Operations (Weeks 13-16)

Week 13: Production Infrastructure
- [ ] Create production Docker containers
- [ ] Set up Kubernetes deployment manifests
- [ ] Implement secrets management
- [ ] Create monitoring and logging infrastructure
- [ ] Set up backup and disaster recovery
- [ ] Create network security configuration
- [ ] Implement auto-scaling policies
- [ ] Document infrastructure architecture

Week 14: Testing & Validation
- [ ] Complete comprehensive integration testing
- [ ] Perform load and stress testing
- [ ] Conduct security penetration testing
- [ ] Execute user acceptance testing with IRCC
- [ ] Validate compliance requirements
- [ ] Test disaster recovery procedures
- [ ] Complete performance validation
- [ ] Document test results and sign-offs

Week 15: Training & Documentation
- [ ] Create operator training materials

- [ ] Write technical operations manual
- [ ] Develop user guides and procedures
- [ ] Create troubleshooting documentation
- [ ] Prepare compliance documentation
- [ ] Create security procedures manual
- [ ] Develop incident response playbooks
- [ ] Conduct training sessions

Week 16: Production Deployment
- [ ] Execute production deployment
- [ ] Validate system functionality
- [ ] Monitor system performance
- [ ] Address initial production issues
- [ ] Complete knowledge transfer
- [ ] Establish ongoing support procedures
- [ ] Create maintenance schedules
- [ ] Document lessons learned

Quality Assurance Checklist

Security Validation

- [ ] All PII data encrypted at rest using AES-256
- [ ] All data transmission uses TLS 1.3
- [ ] Authentication uses JWT with proper expiration
- [ ] Authorization implements role-based access control
- [ ] All user actions logged in tamper-evident audit trail
- [ ] Security headers implemented (HSTS, CSP, etc.)
- [ ] Input validation prevents injection attacks
- [ ] Rate limiting protects against DoS attacks
- [ ] Secrets management properly implemented
- [ ] Security scanning integrated in CI/CD pipeline

Privacy Compliance

- [ ] Purpose limitation enforced in all processing
- [ ] Data minimization principles implemented
- [ ] Consent management for applicable scenarios
- [ ] Individual access rights supported
- [ ] Data retention policies automated
- [ ] Secure data disposal procedures
- [ ] Privacy impact assessment completed
- [ ] Breach notification procedures established
- [ ] Cross-border data transfer restrictions enforced
- [ ] Privacy by design principles followed

Performance Validation

- [ ] PII detection completes within 500ms for standard documents
- [ ] System supports 200 concurrent users
- [ ] Batch processing handles 10,000 documents/day
- [ ] API response times under 2 seconds
- [ ] Memory usage under 4GB per instance
- [ ] CPU utilization under 80% at peak load
- [ ] Database queries optimized with proper indexing
- [ ] Caching reduces redundant processing

```
- [ ] Auto-scaling responds to load changes
- [ ] Recovery time under 4 hours for major incidents

## Functional Validation

- [ ] UCI number detection with checksum validation
- [ ] SIN validation using Luhn algorithm
- [ ] Passport number recognition for multiple countries
- [ ] Work/study permit number validation
- [ ] Canadian postal code format validation
- [ ] Provincial health card detection
- [ ] Multilingual PII detection (English/French minimum)
- [ ] Document type classification accuracy >90%
- [ ] OCR processing for scanned documents
- [ ] Email and attachment processing

## Integration Validation

- [ ] GCMS connectivity and data exchange
- [ ] Database integration with proper transactions
- [ ] External API error handling and retries
- [ ] Message queue processing for async tasks
- [ ] File storage integration with encryption
- [ ] Monitoring system integration
- [ ] Alerting system configuration
- [ ] Backup and restore procedures
- [ ] Log aggregation and analysis
- [ ] Configuration management system
```

Deployment Instructions for Cline

Prerequisites

Before starting implementation, ensure the following are available:

1. **Development Environment**
   - Python 3.11+ with virtual environment
   - PostgreSQL 15+ with encryption support
   - Redis 7+ for caching
   - Docker and Docker Compose
   - Git for version control
2. **API Keys and Secrets**
   - Hugging Face API token (optional, for model downloads)
   - Database encryption keys
   - JWT secret keys
   - GCMS integration credentials (when available)
3. **Infrastructure Access**
   - Kubernetes cluster (for production deployment)
   - Network access to IRCC systems
   - Secure file storage for model artifacts

- Monitoring and logging infrastructure

Implementation Priority Order

1. **Start with Core PII Detection** (Highest Priority)
   - Focus on `app/services/pii/ircc_detector.py`
   - Implement Canadian-specific patterns first
   - Add ML model integration second
2. **Build Document Processing Pipeline**
   - Implement `app/services/document/ircc_processor.py`
   - Support PDF and text documents initially
   - Add OCR support for scanned documents
3. **Create API Endpoints**
   - Build `app/api/v1/endpoints/pii_detection.py`
   - Implement basic authentication
   - Add comprehensive error handling
4. **Add Compliance Framework**
   - Implement `app/services/compliance/ircc_compliance.py`
   - Create audit logging system
   - Add regulatory compliance validation
5. **Deploy and Test**
   - Use Docker Compose for development
   - Implement Kubernetes for production
   - Add monitoring and alerting

Critical Implementation Notes

1. **Security First**: Every component must include security controls from the beginning
2. **Audit Everything**: All PII processing must be logged for compliance
3. **Performance Monitoring**: Include performance metrics in all components
4. **Error Handling**: Implement graceful degradation for all failure scenarios
5. **Testing**: Write tests for each component as it's developed

Success Criteria

The implementation will be considered successful when:

- [ ] PII detection accuracy exceeds 95% on IRCC test documents
- [ ] Processing time remains under 500ms for standard documents
- [ ] All regulatory compliance requirements are met
- [ ] Security assessment passes with no critical findings
- [ ] User acceptance testing completed successfully
- [ ] System handles peak load without degradation
- [ ] Full audit trail captured for all operations
- [ ] Integration with GCMS systems functional
- [ ] Comprehensive documentation completed

- [ ] Operations team trained and ready

This specification provides the complete technical foundation needed to build an enterprise-grade PII detection system for IRCC that meets all security, compliance, and performance requirements.