```
import os
import logging
from flask import Flask, render_template, request, redirect, url_for, flash, jsonify, ses
sion
from werkzeug.utils import secure_filename
import uuid
from document_processor import DocumentProcessor

# Configure logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

# Initialize Flask app
app = Flask(__name__)
app.secret_key = os.environ.get("SESSION_SECRET", "default_secret_key_for_development")
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # 16MB max upload size
app.config['UPLOAD_FOLDER'] = '/tmp/solar_docs'

# Create upload folder if it doesn't exist
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

# Initialize document processor
document_processor = DocumentProcessor()

# Allowed file extensions
ALLOWED_EXTENSIONS = {'pdf', 'png', 'jpg', 'jpeg', 'tiff', 'tif'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'document' not in request.files:
        flash('No file part', 'danger')
        return redirect(url_for('index'))

    file = request.files['document']

    if file.filename == '':
        flash('No file selected', 'danger')
        return redirect(url_for('index'))

    if not allowed_file(file.filename):
        flash('Invalid file type. Please upload PDF, PNG, JPG, JPEG, or TIFF files.', 'da
nger')
        return redirect(url_for('index'))

    try:
        # Create a unique filename
        filename = secure_filename(file.filename)
        unique_filename = f"{uuid.uuid4()}_{filename}"
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], unique_filename)

        # Save the file
        file.save(file_path)
        logger.debug(f"File saved to {file_path}")

        # Process the document
        result = document_processor.process_document(file_path)

        # Store result in session for display
        session['processing_result'] = result

        # Clean up - remove the file after processing
        os.remove(file_path)
        return redirect(url_for('results'))

    except Exception as e:
        logger.error(f"Error processing document: {str(e)}", exc_info=True)
        flash(f'Error processing document: {str(e)}', 'danger')
        return redirect(url_for('index'))

@app.route('/results')
def results():
    result = session.get('processing_result')
    if not result:
        flash('No processing results found', 'warning')
        return redirect(url_for('index'))

    return render_template('results.html', result=result)

@app.route('/api/process', methods=['POST'])
def api_process():
    if 'document' not in request.files:
        return jsonify({'error': 'No file part'}), 400

    file = request.files['document']

    if file.filename == '':
        return jsonify({'error': 'No file selected'}), 400

    if not allowed_file(file.filename):
        return jsonify({'error': 'Invalid file type'}), 400

    try:
        # Create a unique filename
        filename = secure_filename(file.filename)
        unique_filename = f"{uuid.uuid4()}_{filename}"
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], unique_filename)

        # Save the file
        file.save(file_path)

        # Process the document
        result = document_processor.process_document(file_path)

        # Clean up - remove the file after processing
        os.remove(file_path)

        return jsonify(result)

    except Exception as e:
        logger.error(f"API Error: {str(e)}", exc_info=True)
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

```
import os
import re
import logging
import numpy as np
import pytesseract
from PIL import Image
import cv2
from pdf2image import convert_from_path
import pickle
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
import joblib
from models import DOCUMENT_CATEGORIES, CATEGORIES_BY_ID, CATEGORIES_BY_NAME

# Configure logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

class DocumentProcessor:
    def __init__(self):
        self.model_path = os.path.join(os.path.dirname(__file__), 'training_data/document
_classifier.pkl')
        self.vectorizer_path = os.path.join(os.path.dirname(__file__), 'training_data/tfi
df_vectorizer.pkl')

        # Load or initialize the model and vectorizer
        self._load_or_initialize_model()

    def _load_or_initialize_model(self):
        # Check if model and vectorizer exist, otherwise initialize new ones
        try:
            if os.path.exists(self.model_path) and os.path.exists(self.vectorizer_path):
                logger.info("Loading existing model and vectorizer")
                self.classifier = joblib.load(self.model_path)
                self.vectorizer = joblib.load(self.vectorizer_path)
            else:
                logger.info("Initializing new model and vectorizer")
                self.vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
                self.classifier = LogisticRegression(C=1.0, random_state=42, max_iter=100
0)
                # Note: We'll need training data to properly fit these models
        except Exception as e:
            logger.error(f"Error loading model: {str(e)}")
            self.vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
            self.classifier = LogisticRegression(C=1.0, random_state=42, max_iter=1000)

    def process_document(self, file_path):
        """
        Process a document file, classify it, and extract relevant fields
        """
        logger.debug(f"Processing document: {file_path}")

        # Extract text using OCR
        text = self._extract_text(file_path)

        if not text:
            raise ValueError("Could not extract any text from the document")

        # Classify the document
        category, confidence = self._classify_document(text)

        # Extract fields based on category
        extracted_fields = self._extract_fields(text, category, file_path)

        # Return the structured result
        return {
            "category": category,
            "extracted_fields": extracted_fields,
            "raw_text": text,
            "confidence": round(confidence, 2)
        }

    def _extract_text(self, file_path):
        """
        Extract text from document using OCR
        """
        try:
            ext = file_path.split('.')[-1].lower()

            if ext == 'pdf':
                return self._extract_text_from_pdf(file_path)
            elif ext in ['png', 'jpg', 'jpeg', 'tiff', 'tif']:
                return self._extract_text_from_image(file_path)
            else:
                raise ValueError(f"Unsupported file format: {ext}")

        except Exception as e:
            logger.error(f"Error extracting text: {str(e)}")
            raise

    def _extract_text_from_pdf(self, pdf_path):
        """
        Extract text from PDF using pdf2image and pytesseract
        """
        try:
            # Convert PDF to images
            images = convert_from_path(pdf_path)

            # Extract text from each page
            text = ""
            for img in images:
                # Convert PIL Image to OpenCV format
                img_cv = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)

                # Preprocess the image
                img_cv = self._preprocess_image(img_cv)

                # Extract text using pytesseract
                page_text = pytesseract.image_to_string(img_cv)
                text += page_text + "\n\n"

            return text.strip()

        except Exception as e:
            logger.error(f"Error extracting text from PDF: {str(e)}")
            raise

    def _extract_text_from_image(self, image_path):
        """
        Extract text from image using pytesseract
        """
        try:
            # Read the image
            img = cv2.imread(image_path)

            if img is None:
                raise ValueError(f"Could not read image: {image_path}")

            # Preprocess the image
            img = self._preprocess_image(img)

            # Extract text using pytesseract
            text = pytesseract.image_to_string(img)

            return text.strip()

        except Exception as e:
            logger.error(f"Error extracting text from image: {str(e)}")
            raise
    def _preprocess_image(self, img):
        """
        Preprocess image for better OCR results
        """
        # Convert to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Apply threshold to get black and white image
        _, binary = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

        # Apply dilation and erosion to remove noise
        kernel = np.ones((1, 1), np.uint8)
        binary = cv2.dilate(binary, kernel, iterations=1)
        binary = cv2.erode(binary, kernel, iterations=1)

        return binary

    def _classify_document(self, text):
        """
        Classify the document based on its text content
        """
        # Simple rule-based classification as fallback
        categories = []
        confidences = []

        # Keywords for each category
        keywords = {
            "Final Inspection Card": ["inspection", "final inspection", "building permit"
, "approved", "inspector", "code compliance"],
            "Interconnection Agreement": ["interconnection agreement", "grid connection",
 "utility provider", "electric service"],
            "PTO": ["permission to operate", "pto", "authorization to energize", "system
activation"],
            "Warranty Extension": ["warranty", "extended warranty", "product guarantee",
"serial number", "solaredge"],
            "Interconnection / NEM Agreement": ["net energy metering", "nem", "net meteri
ng", "billing arrangement"]
        }

        # Check for keywords in text
        for category, terms in keywords.items():
            matches = sum(1 for term in terms if term.lower() in text.lower())
            score = matches / len(terms) if len(terms) > 0 else 0
            categories.append(category)
            confidences.append(score)

        # Use trained model if available
        try:
            if hasattr(self, 'vectorizer') and hasattr(self, 'classifier'):
                # Transform text to feature vector
                text_features = self.vectorizer.transform([text])

                # Get model prediction and probabilities
                prediction = self.classifier.predict(text_features)[0]
                probabilities = self.classifier.predict_proba(text_features)[0]

                # Get the category and confidence
                predicted_category = CATEGORIES_BY_ID[prediction].name
                confidence = max(probabilities)

                # If model confidence is high, return the model prediction
                if confidence > 0.5:
                    return predicted_category, confidence
        except Exception as e:
            logger.warning(f"Model prediction failed: {str(e)}, falling back to rule-base
d classification")

        # Fall back to rule-based if model prediction fails or has low confidence
        max_confidence_idx = np.argmax(confidences)
        return categories[max_confidence_idx], confidences[max_confidence_idx]
    def _extract_fields(self, text, category, file_path=None):
        """
        Extract relevant fields based on document category
        """
        extracted_fields = {}

        # Convert category string to category object
        if category in CATEGORIES_BY_NAME:
            category_obj = CATEGORIES_BY_NAME[category]
        else:
            logger.warning(f"Unknown category: {category}")
            return extracted_fields

        # Extract fields based on category
        if category == "Final Inspection Card":
            extracted_fields["property_address"] = self._extract_address(text)

            # For FIC image, indicate if it was processed from an image
            if file_path:
                if file_path.lower().endswith(('.jpg', '.jpeg', '.png', '.tiff', '.tif'))
:
                    extracted_fields["fic_image"] = "FIC image was processed from file: "
 + os.path.basename(file_path)
                elif file_path.lower().endswith('.pdf'):
                    extracted_fields["fic_image"] = "FIC was processed from PDF document:
 " + os.path.basename(file_path)
                else:
                    extracted_fields["fic_image"] = "FIC was processed from file: " + os.
path.basename(file_path)
            else:
                extracted_fields["fic_image"] = "No direct image data available"

            extracted_fields["non_fic_proof"] = self._extract_non_fic_proof(text)

        elif category == "Interconnection Agreement":
            # Use more specific address extraction for interconnection agreements
            extracted_fields["home_address"] = self._extract_interconnection_address(text
)
            extracted_fields["homeowner_signature"] = self._extract_signature_presence(te
xt)
            extracted_fields["utility_provider"] = self._extract_utility_provider(text)

        elif category == "PTO":
            extracted_fields["home_address"] = self._extract_address(text)
            extracted_fields["pto_receive_date"] = self._extract_date(text)
            extracted_fields["utility_provider"] = self._extract_utility_provider(text)
            extracted_fields["system_details"] = self._extract_system_details(text)
            extracted_fields["enrollment_program"] = self._extract_enrollment_program(tex
t)

        elif category == "Warranty Extension":
            extracted_fields["warranty_proof"] = self._extract_warranty_proof(text)
            extracted_fields["serial_number"] = self._extract_serial_number(text)

        elif category == "Interconnection / NEM Agreement":
            extracted_fields["document_name"] = self._extract_nem_document_name(text)
            extracted_fields["home_address"] = self._extract_interconnection_address(text
)
            extracted_fields["homeowner_signature"] = self._extract_signature_presence(te
xt)
            extracted_fields["utility_signature"] = self._extract_signature_presence(text
, utility=True)
            extracted_fields["utility_provider"] = self._extract_utility_provider(text)
            extracted_fields["nem_specific_info"] = self._extract_nem_specific_info(text)

        return extracted_fields

    def _extract_address(self, text):
        """
        Extract address from document text
        """
        # Regex patterns for addresses
        address_patterns = [
            # Generic street address patterns
            r'\b\d+\s+[A-Za-z0-9\s,]+(?:Avenue|Lane|Road|Boulevard|Drive|Street|Ave|Dr|Rd
|Blvd|Ln|St|CT|Ct)\.?(?:\s+[A-Za-z]+)?(?:\s+[A-Z]{2}\s+\d{5}(?:-\d{4})?)?\b',
            r'\b\d+\s+[A-Za-z0-9\s,]+\b(?:(?:Avenue|Lane|Road|Boulevard|Drive|Street|Ave|
Dr|Rd|Blvd|Ln|St|CT|Ct)\.?)(?:\s+[A-Za-z]+,\s+[A-Z]{2}\s+\d{5}(?:-\d{4})?)?\b',

            # Address label patterns from real documents
            r'Address[:\s]+([^\n,]+)',
            r'Location of Work[:\s]+([^\n,]+)',
            r'Job Address[:\s]+([^\n,]+)',
            r'Property[:\s]*Address[:\s]*([^\n,]+)',

            # Complex pattern for city, state, zip format
            r'\b(\d+\s+[\w\s]+(?:,\s+)?(?:[\w\s]+)(?:,\s+)?[A-Z]{2}(?:,\s+)?\d{5}(?:-\d{4
})?)\b',

            # Pattern for specific notation in FIC documents
            r'Location of Work:\s*([^P\n]+)',
            r'Property:?\s+([^,\n]+)'
        ]

        for pattern in address_patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            if matches:
                # Some real documents have very long text matches, try to trim to just th
e address
                address = matches[0].strip()
                if isinstance(address, str) and len(address) > 100:
                    # Try to extract just the street address portion
                    street_match = re.search(r'\d+\s+[\w\s]+(?:Avenue|Lane|Road|Boulevard
|Drive|Street|Ave|Dr|Rd|Blvd|Ln|St|CT|Ct)', address, re.IGNORECASE)
                    if street_match:
                        return street_match.group().strip()
                return address

        return "Address not found"

    def _extract_date(self, text):
        """
        Extract date from document text
        """
        # Regex patterns for dates
        date_patterns = [
            r'\b(?:Jan(?:uary)?|Feb(?:ruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:e)?|Jul(?:y
)?|Aug(?:ust)?|Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:ember)?)\s+\d{1,2},?\s+\d{
4}\b',
            r'\b\d{1,2}/\d{1,2}/\d{2,4}\b',
            r'\b\d{1,2}-\d{1,2}-\d{2,4}\b',
            r'\b\d{4}/\d{1,2}/\d{1,2}\b',
            r'\b\d{4}-\d{1,2}-\d{1,2}\b',
            r'\b\d{2}/\d{2}/\d{2,4}\b'
        ]

        # Look for specific PTO date patterns based on real documents
        pto_specific_patterns = [
            # PG&E style
            r'has permission to operate as of (\d{2}/\d{2}/\d{4})',
            r'has permission to operate as of (\d{1,2}/\d{1,2}/\d{4})',
            r'permission to operate as of (\d{1,2}/\d{1,2}/\d{4})',

            # SCE style
            r'Permission to Operate \(PTO\) Granted: (\d{1,2}/\d{1,2}/\d{4})',
            r'Permission to Operate \(PTO\) Granted: (\d{1,2}/\d{1,2}/\d{2})',

            # Oncor style
            r'was approved on (\d{2}/\d{2}/\d{4})',
            # ComEd style
            r'may liven your system.*?([A-Za-z]+ \d{1,2}, \d{4})',

            # General formats
            r'Permission to Operate Granted[:\s]*(\d{1,2}/\d{1,2}/\d{4})',
            r'Permission to[- ]Operate[:\s]*(\d{1,2}/\d{1,2}/\d{4})',
            r'approved for use[:\s]*(\d{1,2}/\d{1,2}/\d{4})',
            r'as of (\d{1,2}/\d{1,2}/\d{4})',
            r'PTO.*?granted[:\s]*(\d{1,2}/\d{1,2}/\d{4})',

            # FPL format
            r'As of (\d{2}/\d{2}/\d{4})',

            # General date after PTO
            r'Permission to Operate.*?(\d{1,2}/\d{1,2}/\d{4})',
            r'Permission to Operate.*?(\d{1,2}-\d{1,2}-\d{4})',
            r'Certificate of Completion.*?Completed on (\d{1,2}/\d{1,2}/\d{4})',

            # Utility signature formats
            r'EDC Signature.*?Date\s+(\d{1,2}/\d{1,2}/\d{4})'
        ]

        # First check for exact PTO-specific dates from real documents
        for pattern in pto_specific_patterns:
            match = re.search(pattern, text, re.IGNORECASE | re.DOTALL)
            if match:
                extracted_date = match.group(1).strip()
                return extracted_date

        # Look for dates near PTO keywords
        pto_keywords = ['permission to operate', 'pto', 'approved', 'approval', 'certific
ate of completion',
                        'interconnection agreement', 'approved for use', 'system approved
']

        # Extract all dates from text
        all_dates = []
        for pattern in date_patterns:
            matches = re.finditer(pattern, text, re.IGNORECASE)
            for match in matches:
                date_text = match.group(0)
                all_dates.append((date_text, match.start()))

        # Sort dates by position in text
        all_dates.sort(key=lambda x: x[1])

        # If we found dates, look for the closest date to PTO keywords
        if all_dates:
            for keyword in pto_keywords:
                keyword_pos = text.lower().find(keyword)
                if keyword_pos != -1:
                    # Find closest date to keyword
                    closest_date = None
                    min_distance = float('inf')

                    for date_text, date_pos in all_dates:
                        distance = abs(date_pos - keyword_pos)
                        if distance < min_distance:
                            min_distance = distance
                            closest_date = date_text

                    if closest_date and min_distance < 200:  # Only use dates within reas
onable proximity
                        return closest_date

            # If no date near keywords, return the most recent date
            # (PTO is typically one of the last dates in the document)
            return all_dates[-1][0]
        return "Date not found"

    def _extract_signature_presence(self, text, utility=False):
        """
        Check for presence of signature in document text
        """
        if utility:
            # Look for utility signature patterns from real documents
            signature_patterns = [
                r'(?:Utility|Company|Provider|EDC|SCE|PG&E)[:\s]+(?:Signature|Signed|Appr
oved)',
                r'(?:Utility|Company|Provider|EDC|SCE|PG&E)[\s\S]+?(?:Signature|Signed|Ap
proved)',
                r'EDC Signature',
                r'Acceptance and Final Approval for Interconnection',
                r'interconnection agreement is approved',
                r'Title[:\s]+(?:Interconnect Specialist|Representative|Supervisor)',
                r'Supervisor, EGI'
            ]
        else:
            # Look for homeowner signature patterns from real documents
            signature_patterns = [
                r'(?:Owner|Customer|Homeowner|Account Holder|Interconnection Customer)[:\
s]+(?:Signature|Signed|Approved)',
                r'(?:Owner|Customer|Homeowner|Account Holder|Interconnection Customer)[\s
\S]+?(?:Signature|Signed|Approved)',
                r'(?:Signature|Signed|Approved)[:\s]+(?:[^\n]+)',
                r'Customer Signature',
                r'Signature Date',
                r'Certificate of Completion',
                r'Jasmin',
                r'DocuSign',
                r'Docusign Envelope ID'
            ]

        for pattern in signature_patterns:
            if re.search(pattern, text, re.IGNORECASE):
                return "Signature detected"

        # Check for name-like patterns following signature indicators
        name_after_signature = re.search(r'Signature[:\s]+([A-Za-z\s\.]+)', text, re.IGNO
RECASE)
        if name_after_signature:
            return "Signature detected: " + name_after_signature.group(1).strip()

        return "No signature detected"

    def _extract_non_fic_proof(self, text):
        """
        Extract proof of non-FIC-required region
        """
        # Look for indicators that FIC is not required
        # Based on real examples from the provided documents
        non_fic_patterns = [
            r'(?:not|no)\s+required',
            r'(?:exempt|exemption|waived|waiver)',
            r'no\s+inspection\s+(?:needed|required)',
            r'no permits no inspections',
            r'permits are not required for the installation of solar panels',
            r'attest that this property is located within a jurisdiction',
            r'moving to interconnection',
            r'private provider inspections'
        ]

        # First check for full attestation statements
        attestation_patterns = [
            r'I,\s+[\w\s]+,\s+attest\s+that\s+this\s+property\s+is\s+located\s+within\s+a
\s+jurisdiction\s+in\s+which\s+permits\s+are\s+not\s+required\s+for\s+the\s+installation\
s+of\s+solar\s+panels'
        ]
        for pattern in attestation_patterns:
            match = re.search(pattern, text, re.IGNORECASE)
            if match:
                return match.group().strip()

        # Then check for other pattern mentions
        for pattern in non_fic_patterns:
            if re.search(pattern, text, re.IGNORECASE):
                context = re.findall(r'.{0,75}' + pattern + r'.{0,75}', text, re.IGNORECA
SE)
                if context:
                    return context[0].strip()

        # Look for specific screenshot or system mentions about no permits
        screenshot_patterns = [
            r'No permits.*moving to interconnection',
            r'PRIVATE PROVIDER.*INSPECTIONS'
        ]

        for pattern in screenshot_patterns:
            match = re.search(pattern, text, re.IGNORECASE | re.DOTALL)
            if match:
                return match.group().strip()

        return "No proof of non-FIC-required region found"

    def _extract_warranty_proof(self, text):
        """
        Extract warranty proof information
        """
        warranty_info = {}

        # Look for extended warranty indicators (SolarEdge pattern)
        solaredge_patterns = [
            r'SolarEdge\s+Extended\s+Warranty',
            r'extended\s+to\s+(\d+)\s+years',
            r'Warranty\s+is\s+valid\s+until[:\s]+(\d{2}/\d{2}/\d{4})',
            r'This warranty certificate is valid'
        ]

        # Check for SolarEdge extended warranty first
        for pattern in solaredge_patterns:
            match = re.search(pattern, text, re.IGNORECASE)
            if match:
                if 'warranty_type' not in warranty_info:
                    warranty_info['warranty_type'] = 'SolarEdge Extended Warranty'

                # If the pattern captured a group (like years or date)
                if match.groups():
                    if 'years' in match.re.pattern:
                        warranty_info['duration'] = f"{match.group(1)} years"
                    elif 'valid until' in match.re.pattern:
                        warranty_info['valid_until'] = match.group(1)

        # Look for general warranty information if no SolarEdge warranty was found
        if not warranty_info:
            general_warranty_patterns = [
                r'(?:warranty|guarantee)[:\s]+([^\n]+)',
                r'(?:warranty|guarantee)[\s\S]+?(?:period|term)[:\s]+([^\n]+)',
                r'(?:extend|extension)[:\s]+([^\n]+)',
                r'warranty period[:\s]+([^\n]+)',
                r'valid until[:\s]+([^\n]+)',
                r'expires?(?:s)?[:\s]+([^\n]+)'
            ]

            for pattern in general_warranty_patterns:
                match = re.search(pattern, text, re.IGNORECASE)
                if match:
                    warranty_info['warranty_details'] = match.group(1).strip()
                    break

        # Format the warranty information as a string
        if warranty_info:
            result = ""
            if 'warranty_type' in warranty_info:
                result += f"Type: {warranty_info['warranty_type']}\n"
            if 'duration' in warranty_info:
                result += f"Duration: {warranty_info['duration']}\n"
            if 'valid_until' in warranty_info:
                result += f"Valid Until: {warranty_info['valid_until']}\n"
            if 'warranty_details' in warranty_info:
                result += f"Details: {warranty_info['warranty_details']}\n"

            return result.strip()

        return "No warranty proof found"

    def _extract_serial_number(self, text):
        """
        Extract serial number from text
        """
        # Check for SolarEdge style serial numbers first
        solaredge_serial_pattern = r'Inverter\s+Serial\s+Number[:\s]*\n*\s*(SB\d+-\w+-\w+
)'
        match = re.search(solaredge_serial_pattern, text, re.IGNORECASE)
        if match:
            return match.group(1).strip()

        # Common serial number patterns
        serial_patterns = [
            r'(?:Serial|S/N|SN)[:\s#]+([A-Za-z0-9-]+)',
            r'(?:Serial\s+Number|S/N|SN)[:\s#]+([A-Za-z0-9-]+)',
            r'Serial\s+Number\s*:?\s*\n*\s*([A-Za-z0-9-]+)',
            r'Device\s+Serial\s+Number[:\s]+([A-Za-z0-9-]+)'
        ]

        for pattern in serial_patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            if matches:
                return matches[0].strip()

        # Look for isolated patterns that resemble serial numbers
        # (especially for SolarEdge format: SBxxxx-xxxxxxxxx-xx)
        isolated_serial_pattern = r'\b(SB\d+-[A-Za-z0-9]+-[A-Za-z0-9]+)\b'
        match = re.search(isolated_serial_pattern, text)
        if match:
            return match.group(1).strip()

        return "No serial number found"

    def _extract_interconnection_address(self, text):
        """
        Extract address specifically from interconnection agreement documents
        """
        # Patterns specific to interconnection agreements based on real examples
        interconnection_address_patterns = [
            # Common patterns in real interconnection agreements
            r'(?:Service|Customer|Installation|Generator)\s+Address[:\s]+([^\n]+)',
            r'(?:Service|Customer|Installation|Generator)\s+Location[:\s]+([^\n]+)',
            r'Address of (?:Facility|Generation|System)[:\s]+([^\n]+)',
            r'Generation Facility location[:\s]+([^\n]+)',

            # Specific patterns from provided documents
            r'Premise Address[:\s]+([^\n]+)',
            r'Electric Service Address[:\s]+([^\n]+)',
            r'Physical Address[:\s]+([^\n]+)',
            r'Site Address[:\s]+([^\n]+)',

            # Patterns for address sections with a house number followed by street
            r'(?:Location|Site|Address)[:\s]*\d+\s+[A-Za-z0-9\s,\.]+(?:Avenue|Lane|Road|B
oulevard|Drive|Street|Ave|Dr|Rd|Blvd|Ln|St|CT|Ct)',

            # City, state, zip format specific to interconnection docs
            r'(?:City|Town)[:\s]+([^\n,]+)(?:\s*,\s*)[A-Z]{2}\s+\d{5}'
        ]

        # First try the interconnection specific patterns
        for pattern in interconnection_address_patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            if matches:
                address = matches[0].strip()
                if isinstance(address, str) and len(address) > 100:
                    # Try to extract just the street address portion
                    street_match = re.search(r'\d+\s+[\w\s]+(?:Avenue|Lane|Road|Boulevard
|Drive|Street|Ave|Dr|Rd|Blvd|Ln|St|CT|Ct)', address, re.IGNORECASE)
                    if street_match:
                        return street_match.group().strip()
                return address

        # If no matches with specific patterns, try the generic address extraction
        return self._extract_address(text)

    def _extract_utility_provider(self, text):
        """
        Extract utility provider information from interconnection agreements
        """
        # Patterns to identify utility provider
        utility_patterns = [
            r'(?:Utility|EDC|Electric Distribution Company)[:\s]+([^\n]+)',
            r'(?:Utility|EDC|Electric Distribution Company)\s+Name[:\s]+([^\n]+)',
            r'(?:Provider|Company)[:\s]+([^\n]+)',

            # Specific utility names
            r'\b(PG&E|Pacific Gas and Electric|Southern California Edison|SCE|SDG&E|San D
iego Gas & Electric)\b',

            # From interconnection agreement document titles
            r'(PG&E|SCE|SDG&E)\s+(?:Interconnection|Agreement)',

            # Utility account number patterns that might identify the provider
            r'(?:Utility|EDC)\s+Account\s+Number[:\s]+([^\n]+)'
        ]

        for pattern in utility_patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            if matches:
                return matches[0].strip()

        # Check for common utility names in the text
        common_utilities = ["PG&E", "Pacific Gas and Electric", "Southern California Edis
on", "SCE",
                           "San Diego Gas & Electric", "SDG&E", "Edison", "ConEdison", "N
ational Grid"]

        for utility in common_utilities:
            if re.search(r'\b' + re.escape(utility) + r'\b', text, re.IGNORECASE):
                return utility

        return "Utility provider not found"

    def _extract_nem_document_name(self, text):
        """
        Extract document name or title specifically for NEM/NBT agreements
        """
        # Look for specific document title patterns in provided examples
        nem_title_patterns = [
            r'AGREEMENT AND CUSTOMER AUTHORIZATION\s+Net\s+Billing\s+Tariff\s+\(NBT\)\s+I
nterconnection',
            r'NET BILLING TARIFF OR NET ENERGY METERING AND RENEWABLE[\s\S]+?INTERCONNECT
```

```
ION AGREEMENT',
            r'NET BILLING TARIFF \(NBT\) OR NET ENERGY METERING \(NEM\)[\s\S]+?INTERCONNE
CTION AGREEMENT',
            r'(?:Net\s+Energy\s+Metering|NEM|Net\s+Metering)\s+(?:Agreement|Application|C
ontract)',
            r'(?:NBT|Net\s+Billing\s+Tariff)\s+(?:Agreement|Application|Contract)',
            r'Form.*?(?:16-344|344|345)',
            r'(NEM\s+Agreement\s+\d+\s+kW)',
            r'(NEM\s+2\.0)',
            r'(California\s+Solar\s+Initiative)',
            r'(Interconnection\s+Agreement\s+For\s+Net\s+Energy\s+Metering)'
        ]

        # First try to find the document name from specific patterns
        for pattern in nem_title_patterns:
            match = re.search(pattern, text, re.IGNORECASE)
            if match:
                return match.group(0).strip().replace('\n', ' ').replace('  ', ' ')

        # Check for docusign envelope ID pattern which often appears in these documents
        docusign_match = re.search(r'Docusign Envelope ID:\s*([A-Z0-9-]+)', text, re.IGNO
RECASE)
        if docusign_match:
            return f"Signed NEM/NBT Agreement (DocuSign ID: {docusign_match.group(1)})"

        # If specific patterns don't match, try general document name extraction
        return self._extract_document_name(text)

    def _extract_nem_specific_info(self, text):
        """
        Extract NEM/NBT-specific information from the document
        """
        nem_info = {}

        # Look for PG&E NBT-specific information
        pge_patterns = {
            "agreement_type": [
                r'Standard Net Billing Tariff \(NBT\) Agreement Type:\s*(?:\u2611|\u2610|
\u2612|â\230\221|â\230\220)\s*(Single Account|Multiple Aggregated Account)',
                r'(?:\u2611|\u2610|\u2612|â\230\221|â\230\220)\s*(Single Account)[\s\S]+?
(?:\u2611|\u2610|\u2612|â\230\221|â\230\220)\s*(Multiple Aggregated Account)'
            ],
            "customer_sector": [
                r'Customer Sector[\s\S]+?(?:\u2611|\u2610|\u2612|â\230\221|â\230\220)\s*(
Residential|Commercial|Industrial|Educational|Military|Non-Profit|Other Government)'
            ],
            "system_size": [
                r'solar lesser of inverter rating or CEC-AC rating\w+\s+(\d+\.?\d*)\s*\(k
W\)',
                r'Total System Size[:\s]+(\d+\.?\d*\s*kW)',
                r'Generating Facility Nameplate Rating \(kW\):\s*(\d+\.?\d*)',
                r'Generating Facility CEC-AC Rating or Equivalent \(kW\):\s*(\d+\.?\d*)'
            ],
            "estimated_annual_production": [
                r'Estimated annual energy production of Generating Facility \(kWh\):\s*(\
d+\.?\d*)',
                r'Total[\s\S]+?Energy[\s\S]+?Production[\s\S]+?=\s*(\d+\.?\d*)\s*\(kWh\)'
            ],
            "account_holder": [
                r'Account Holder Name\*[\s\S]+?([A-Z\s]+)[\s\S]+?Electric Service Agreeme
nt'
            ],
            "service_address": [
                r'Service Address\*\s+([^\n]+)',
                r'Generating Facility Location:\s+([^\n]+)'
            ]
        }

        # SCE-specific patterns
        sce_patterns = {
            "agreement_type": [
                r'NET BILLING TARIFF OR NET ENERGY METERING AND RENEWABLE[\s\S]+?INTERCON
NECTION AGREEMENT'
            ],
            "customer_name": [
                r'is entered into by and between\s+([^\(]+?)\s+\("Customer"\)',
                r'This[\s\S]+?Agreement[\s\S]+?is entered into by and between\s+([^\(]+?)
\s+\("Customer"\)'
            ],
            "customer_account": [
                r'Customer Service Account Number:\s*(\d+)',
                r'Customer Meter Number:\s*(\d+-\d+)'
            ],
            "rate_schedule": [
                r'Applicable Rate Schedule:\s*([A-Z0-9-]+)'
            ],
            "facility_location": [
                r'Generating Facility Location:\s*([^\n]+)'
            ]
        }

        # NBT or NEM classification
        tariff_type_patterns = [
            r'(Net Billing Tariff|NBT) Interconnection',
            r'(Net Energy Metering|NEM) Interconnection',
            r'(NBT) OR (NEM)'
        ]

        # First determine if it's NBT or NEM
        for pattern in tariff_type_patterns:
            match = re.search(pattern, text, re.IGNORECASE)
            if match:
                if match.group(1) and ('NBT' in match.group(1) or 'Billing' in match.grou
p(1)):
                    nem_info['tariff_type'] = 'Net Billing Tariff (NBT)'
                else:
                    nem_info['tariff_type'] = 'Net Energy Metering (NEM)'
                break

        # Determine if it's a PG&E or SCE document
        if 'PG&E' in text or 'Pacific Gas and Electric' in text:
            utility = 'PG&E'
            patterns_to_use = pge_patterns
        elif 'SCE' in text or 'Southern California Edison' in text:
            utility = 'SCE'
            patterns_to_use = sce_patterns
        else:
            utility = 'Unknown Utility'
            # Use both pattern sets
            patterns_to_use = {**pge_patterns, **sce_patterns}

        nem_info['utility'] = utility

        # Extract info based on the determined utility
        for info_type, patterns in patterns_to_use.items():
            for pattern in patterns:
                match = re.search(pattern, text, re.IGNORECASE | re.DOTALL)
                if match:
                    nem_info[info_type] = match.group(1).strip()
                    break

        # Look for system capacity information
        system_capacity_patterns = [
            r'Generating Facility Nameplate Rating \(kW\):\s*(\d+\.?\d*)',
            r'Generating Facility CEC-AC Rating or Equivalent \(kW\):\s*(\d+\.?\d*)',
            r'solar lesser of inverter rating or CEC-AC rating\w+\s+(\d+\.?\d*)\s*\(kW\)'
        ]

        for pattern in system_capacity_patterns:
            match = re.search(pattern, text, re.IGNORECASE)
            if match and 'system_capacity' not in nem_info:
                nem_info['system_capacity'] = f"{match.group(1)} kW"
                break

        # Check for signature evidence
        signature_patterns = [
            r'Docusign Envelope ID:\s*([A-Z0-9-]+)',
            r'DocuSign',
            r'CUSTOMER SIGNATURE',
            r'By checking this box and signing this Agreement'
        ]

        for pattern in signature_patterns:
            if re.search(pattern, text, re.IGNORECASE):
                nem_info['signed'] = 'Yes'
                break

        # Format the results as a string
        if nem_info:
            result = ""
            # Order for better readability
            priority_keys = ['tariff_type', 'utility', 'customer_name', 'account_holder',

                           'service_address', 'facility_location', 'system_capacity',
                           'system_size', 'estimated_annual_production', 'signed']

            # First add priority keys in order
            for key in priority_keys:
                if key in nem_info:
                    nice_key = key.replace('_', ' ').title()
                    result += f"{nice_key}: {nem_info[key]}\n"

            # Then add any remaining keys
            for key, value in nem_info.items():
                if key not in priority_keys:
                    nice_key = key.replace('_', ' ').title()
                    result += f"{nice_key}: {value}\n"

            return result.strip()

        return "No NEM/NBT-specific information found"

    def _extract_system_details(self, text):
        """
        Extract solar system details from PTO documents
        """
        system_details = {}

        # Patterns for system capacity
        capacity_patterns = [
            r'Total Effective Inverter Nameplate Rating: (\d+\.?\d*\s*kW)',
            r'Total Effective Inverter (?:Nameplate|Rating): (\d+\.?\d*\s*kW)',
            r'CEC-AC Nameplate Rating[:\s]*(\d+\.?\d*\s*kW)',
            r'Generating Facility Capacity[:\s]*(\d+\.?\d*\s*kW)',
            r'system size[:\s]*(\d+\.?\d*\s*kW)',
            r'System Size[:\s]*(\d+\.?\d*\s*kW)',
            r'Total System Size[:\s]*(\d+\.?\d*\s*kW)',
            r'(\d+\.?\d*\s*kW)[^\n]*(?:system|capacity)'
        ]

        # Patterns for inverter information
        inverter_patterns = [
            r'Inverter[^\n]*: ([^\n]+)',
            r'Inverter - (?:External|Incorporated)[^\n]*: ([^\n]+)',
            r'(?:External|Incorporated)[^\n]*: ([^\n]+)'
        ]

        # Patterns for panel information
        panel_patterns = [
            r'PV Panels[:\s]*([^\n]+)',
            r'Panel[s]?[:\s]*([^\n]+)',
            r'Module[s]?[:\s]*([^\n]+)'
        ]

        # Patterns for battery information
        battery_patterns = [
            r'Battery[:\s]*([^\n]+)',
            r'Storage[:\s]*([^\n]+)',
            r'Energy Storage Capacity[:\s]*(\d+\.?\d*\s*kW)',
            r'Energy Storage[:\s]*(\d+\.?\d*\s*kW)'
        ]

        # Extract system capacity
        for pattern in capacity_patterns:
            match = re.search(pattern, text, re.IGNORECASE)
            if match:
                system_details["capacity"] = match.group(1).strip()
                break

        # Extract inverter information
        inverter_info = []
        for pattern in inverter_patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            if matches:
                inverter_info.extend(matches)
        if inverter_info:
            system_details["inverters"] = [inv.strip() for inv in inverter_info if len(in
v.strip()) > 5]

        # Extract panel information
        panel_info = []
        for pattern in panel_patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            if matches:
                panel_info.extend(matches)
        if panel_info:
            system_details["panels"] = [panel.strip() for panel in panel_info if len(pane
l.strip()) > 5]

        # Extract battery information
        battery_info = []
        for pattern in battery_patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            if matches:
                battery_info.extend(matches)
        if battery_info:
            system_details["battery"] = [batt.strip() for batt in battery_info if len(bat
t.strip()) > 2]

        # Format the result as string
        if system_details:
            result = ""
            if "capacity" in system_details:
                result += f"System Capacity: {system_details['capacity']}\n"
            if "inverters" in system_details:
                result += f"Inverters: {', '.join(system_details['inverters'])}\n"
            if "panels" in system_details:
                result += f"Panels: {', '.join(system_details['panels'])}\n"
            if "battery" in system_details:
                result += f"Battery: {', '.join(system_details['battery'])}\n"
            return result.strip()

        return "No system details found"

    def _extract_enrollment_program(self, text):
        """
        Extract enrollment program information from PTO documents
        """
        # Look for enrollment program information
        program_patterns = [
            r'enrolled in the following program[:\s]*([A-Za-z0-9\s\.-]+)',
            r'enrolled in[:\s]*([A-Za-z0-9\s\.-]+)',
            r'You are enrolled in[:\s]*([A-Za-z0-9\s\.-]+)',
            r'program[:\s]*([A-Za-z0-9\s\.-]+)',
            r'tariff[:\s]*([A-Za-z0-9\s\.-]+)',
            r'Solar Billing Plan[:\s]*([A-Za-z0-9\s\.-]+)',
            r'Solar Billing Plan / ([A-Za-z0-9\s\.-]+)',
            r'Net Energy Metering[:\s]*([A-Za-z0-9\s\.-]+)',
            r'welcome to ([A-Za-z0-9\s\.-]+) with'
        ]

        # Specific program names/types from real documents
        program_types = [
            r'\b(NEM|Net\s+Energy\s+Metering)\b',
            r'\b(NBT|Net\s+Billing\s+Tariff)\b',
            r'\b(NEM\s*2\.0)\b',
            r'\b(Solar\s+Billing\s+Plan)\b',
            r'\b(VNEM|Virtual\s+Net\s+Energy\s+Metering)\b',
            r'\b(NEMA|Net\s+Energy\s+Metering\s+Aggregation)\b'
        ]

        # First check for explicit enrollment statements
        for pattern in program_patterns:
            match = re.search(pattern, text, re.IGNORECASE)
            if match:
                program = match.group(1).strip()
                # Clean up the program name
                if program.endswith('.'):
                    program = program[:-1]
                return f"Enrollment Program: {program}"

        # Then check for specific program type mentions
        for pattern in program_types:
            match = re.search(pattern, text, re.IGNORECASE)
            if match:
                return f"Enrollment Program: {match.group(0)}"

        return "Enrollment program not specified"

    def _extract_document_name(self, text):
        """
        Extract document name or title
        """
        # Look for document title
        title_patterns = [
            r'^([^\n]+)',  # First line of the document
            r'(?:TITLE|AGREEMENT|CONTRACT)[:\s]+([^\n]+)',
            r'(?:Net\s+Energy\s+Metering|Interconnection)\s+(?:Agreement|Application)'
        ]

        for pattern in title_patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            if matches:
                return matches[0].strip()

        return "Document name not found"
```

```
{% extends 'layout.html' %}

{% block content %}
<div class="row justify-content-center">
    <div class="col-md-8">
        <div class="card shadow-sm">
            <div class="card-header bg-primary text-white">
                <h3 class="card-title mb-0">
                    <i class="fas fa-file-alt me-2"></i>Document Classification & Informa
tion Extraction
                </h3>
            </div>
            <div class="card-body">
                <p class="lead">Upload solar-related documents for automatic classificati
on and information extraction.</p>

                <div class="card mb-4">
                    <div class="card-body">
                        <h5 class="card-title">Supported Document Types</h5>
                        <div class="row">
                            <div class="col-md-6">
                                <ul class="list-group list-group-flush">
                                    <li class="list-group-item bg-transparent">
                                        <i class="fas fa-check-circle text-success me-2">
</i>Final Inspection Card (FIC)
                                    </li>
                                    <li class="list-group-item bg-transparent">
                                        <i class="fas fa-check-circle text-success me-2">
</i>Interconnection Agreement
                                    </li>
                                    <li class="list-group-item bg-transparent">
                                        <i class="fas fa-check-circle text-success me-2">
</i>PTO (Permission-To-Operate)
                                    </li>
                                </ul>
                            </div>
                            <div class="col-md-6">
                                <ul class="list-group list-group-flush">
                                    <li class="list-group-item bg-transparent">
                                        <i class="fas fa-check-circle text-success me-2">
</i>Warranty Extension
                                    </li>
                                    <li class="list-group-item bg-transparent">
                                        <i class="fas fa-check-circle text-success me-2">
</i>Interconnection / NEM Agreement
                                    </li>
                                </ul>
                            </div>
                        </div>
                    </div>
                </div>

                <form action="{{ url_for('upload_file') }}" method="post" enctype="multip
art/form-data" id="uploadForm">
                    <div class="mb-4">
                        <div class="document-upload-area p-4 text-center border rounded">
                            <div class="upload-icon mb-3">
                                <i class="fas fa-cloud-upload-alt fa-4x text-primary"></i
>
                            </div>
                            <h5>Drag & Drop Documents Here</h5>
                            <p class="text-muted">or</p>
                            <div class="input-group">
                                <input type="file" class="form-control" id="document" nam
e="document" accept=".pdf,.png,.jpg,.jpeg,.tiff,.tif">
                                <button class="btn btn-primary" type="submit" id="uploadB
tn">
                                    <i class="fas fa-upload me-2"></i>Upload
                                </button>
                            </div>
                            <small class="form-text text-muted mt-2">
                                Supported formats: PDF, PNG, JPG, JPEG, TIFF (.pdf, .png,
 .jpg, .jpeg, .tiff, .tif)
                            </small>
                        </div>
                    </div>

                    <div id="uploadStatus" class="d-none">
                        <div class="d-flex align-items-center">
                            <div class="spinner-border text-primary me-3" role="status">
                                <span class="visually-hidden">Processing...</span>
                            </div>
                            <div>
                                <h5 class="mb-1">Processing Document</h5>
                                <p class="text-muted mb-0">Please wait while we analyze y
our document...</p>
                            </div>
                        </div>
                        <div class="progress mt-3">
                            <div id="uploadProgress" class="progress-bar progress-bar-str
iped progress-bar-animated" role="progressbar" style="width: 0%"></div>
                        </div>
                    </div>
                </form>
            </div>
        </div>

        <div class="card mt-4 shadow-sm">
            <div class="card-header bg-secondary text-white">
                <h5 class="mb-0">Information We Extract</h5>
            </div>
            <div class="card-body">
                <div class="accordion" id="extractionAccordion">
                    <div class="accordion-item">
                        <h2 class="accordion-header">
                            <button class="accordion-button collapsed" type="button" data
-bs-toggle="collapse" data-bs-target="#ficInfo">
                                Final Inspection Card (FIC)
                            </button>
                        </h2>
                        <div id="ficInfo" class="accordion-collapse collapse" data-bs-par
ent="#extractionAccordion">
                            <div class="accordion-body">
                                <ul>
                                    <li>Full image of the FIC (if available)</li>
                                    <li>Property Address</li>
                                    <li>Proof of non-FIC-required region (if applicable)<
/li>
                                </ul>
                            </div>
                        </div>
                    </div>

                    <div class="accordion-item">
                        <h2 class="accordion-header">
                            <button class="accordion-button collapsed" type="button" data
-bs-toggle="collapse" data-bs-target="#interconnectionInfo">
                                Interconnection Agreement
                            </button>
                        </h2>
                        <div id="interconnectionInfo" class="accordion-collapse collapse"
 data-bs-parent="#extractionAccordion">
                            <div class="accordion-body">
                                <ul>
                                    <li>Home Address</li>
                                    <li>Homeowner’s Signature</li>
                                </ul>
                            </div>
                        </div>
                    </div>
                    <div class="accordion-item">
                        <h2 class="accordion-header">
                            <button class="accordion-button collapsed" type="button" data
-bs-toggle="collapse" data-bs-target="#ptoInfo">
                                PTO (Permission-To-Operate)
                            </button>
                        </h2>
                        <div id="ptoInfo" class="accordion-collapse collapse" data-bs-par
ent="#extractionAccordion">
                            <div class="accordion-body">
                                <ul>
                                    <li>Home Address</li>
                                    <li>PTO Receive Date</li>
                                </ul>
                            </div>
                        </div>
                    </div>

                    <div class="accordion-item">
                        <h2 class="accordion-header">
                            <button class="accordion-button collapsed" type="button" data
-bs-toggle="collapse" data-bs-target="#warrantyInfo">
                                Warranty Extension
                            </button>
                        </h2>
                        <div id="warrantyInfo" class="accordion-collapse collapse" data-b
s-parent="#extractionAccordion">
                            <div class="accordion-body">
                                <ul>
                                    <li>Proof of warranty document for SolarEdge projects
</li>
                                    <li>Serial Number (from warranty document and SolarEd
ge monitoring portal)</li>
                                </ul>
                            </div>
                        </div>
                    </div>

                    <div class="accordion-item">
                        <h2 class="accordion-header">
                            <button class="accordion-button collapsed" type="button" data
-bs-toggle="collapse" data-bs-target="#nemInfo">
                                Interconnection / NEM Agreement
                            </button>
                        </h2>
                        <div id="nemInfo" class="accordion-collapse collapse" data-bs-par
ent="#extractionAccordion">
                            <div class="accordion-body">
                                <ul>
                                    <li>Document Name</li>
                                    <li>Homeowner’s Signature</li>
                                    <li>Utility’s Signature</li>
                                </ul>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

```
<!DOCTYPE html>
<html lang="en" data-bs-theme="dark">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Solar Document Processor</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.replit.com/agent/bootstrap-agent-dark-theme.min.css" rel="sty
lesheet">
    <!-- Font Awesome for icons -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css
" rel="stylesheet">
    <!-- Custom CSS -->
    <link href="{{ url_for('static', filename='css/custom.css') }}" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container">
            <a class="navbar-brand" href="{{ url_for('index') }}">
                <i class="fas fa-solar-panel me-2"></i>
                Solar Document Processor
            </a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-b
s-target="#navbarNav">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ms-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url_for('index') }}">Home</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    <div class="container mt-4 mb-5">
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="alert alert-{{ category }} alert-dismissible fade show" r
ole="alert">
                        {{ message }}
                        <button type="button" class="btn-close" data-bs-dismiss="alert" a
ria-label="Close"></button>
                    </div>
                {% endfor %}
            {% endif %}
        {% endwith %}

        {% block content %}{% endblock %}
    </div>

    <footer class="footer py-3 bg-dark mt-auto">
        <div class="container text-center">
            <span class="text-muted">Solar Document Processor Â© 2023</span>
        </div>
    </footer>

    <!-- Bootstrap JS Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.mi
n.js"></script>
    <!-- Custom JS -->
    <script src="{{ url_for('static', filename='js/main.js') }}"></script>
</body>
</html>
```

```
document.addEventListener('DOMContentLoaded', function() {
    // Handle drag & drop functionality
    const uploadArea = document.querySelector('.document-upload-area');
    const fileInput = document.getElementById('document');
    const uploadForm = document.getElementById('uploadForm');
    const uploadStatus = document.getElementById('uploadStatus');
    const uploadProgress = document.getElementById('uploadProgress');
    const uploadBtn = document.getElementById('uploadBtn');

    if (uploadArea && fileInput) {
        // Prevent default drag behaviors
        ['dragenter', 'dragover', 'dragleave', 'drop'].forEach(eventName => {
            uploadArea.addEventListener(eventName, preventDefaults, false);
        });

        function preventDefaults(e) {
            e.preventDefault();
            e.stopPropagation();
        }

        // Highlight drop area when item is dragged over it
        ['dragenter', 'dragover'].forEach(eventName => {
            uploadArea.addEventListener(eventName, highlight, false);
        });

        ['dragleave', 'drop'].forEach(eventName => {
            uploadArea.addEventListener(eventName, unhighlight, false);
        });

        function highlight() {
            uploadArea.classList.add('bg-light');
        }

        function unhighlight() {
            uploadArea.classList.remove('bg-light');
        }

        // Handle dropped files
        uploadArea.addEventListener('drop', handleDrop, false);

        function handleDrop(e) {
            const dt = e.dataTransfer;
            const files = dt.files;

            if (files.length > 0) {
                fileInput.files = files;

                // Show file name in a meaningful way
                const fileName = files[0].name;
                const fileSize = formatBytes(files[0].size);

                // You could add an element to display the selected file info
                const fileInfoEl = document.createElement('div');
                fileInfoEl.className = 'selected-file-info mt-3 p-2 border rounded';
                fileInfoEl.innerHTML = `
                    <div class="d-flex align-items-center">
                        <i class="fas fa-file-alt text-primary me-2"></i>
                        <div>
                            <strong>${fileName}</strong>
                            <div class="text-muted small">${fileSize}</div>
                        </div>
                        <button type="button" class="btn-close ms-auto" aria-label="Remov
e file"></button>
                    </div>
                `;

                // Replace any existing file info
                const existingFileInfo = uploadArea.querySelector('.selected-file-info');
                if (existingFileInfo) {
                    existingFileInfo.remove();
                }

                uploadArea.appendChild(fileInfoEl);

                // Add event listener to the close button
                const closeBtn = fileInfoEl.querySelector('.btn-close');
                if (closeBtn) {
                    closeBtn.addEventListener('click', function() {
                        fileInput.value = '';
                        fileInfoEl.remove();
                    });
                }
            }
        }
    }

    // Handle form submission and show progress
    if (uploadForm) {
        uploadForm.addEventListener('submit', function(e) {
            if (fileInput.files.length === 0) {
                e.preventDefault();
                showAlert('Please select a file to upload', 'danger');
                return;
            }

            // Show the upload status
            uploadStatus.classList.remove('d-none');
            uploadBtn.disabled = true;

            // Simulate progress (in a real app, this would use XHR or Fetch API)
            simulateProgress();
        });
    }

    function simulateProgress() {
        let progress = 0;
        const interval = setInterval(() => {
            progress += Math.random() * 10;
            if (progress > 100) progress = 100;

            uploadProgress.style.width = `${progress}%`;

            if (progress === 100) {
                clearInterval(interval);
            }
        }, 300);
    }

    function showAlert(message, type) {
        const alertHtml = `
            <div class="alert alert-${type} alert-dismissible fade show" role="alert">
                ${message}
                <button type="button" class="btn-close" data-bs-dismiss="alert" aria-labe
l="Close"></button>
            </div>
        `;

        // Insert alert before the form
        uploadForm.insertAdjacentHTML('beforebegin', alertHtml);
    }

    function formatBytes(bytes, decimals = 2) {
        if (bytes === 0) return '0 Bytes';

        const k = 1024;
        const dm = decimals < 0 ? 0 : decimals;
        const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB'];

        const i = Math.floor(Math.log(bytes) / Math.log(k));
        return parseFloat((bytes / Math.pow(k, i)).toFixed(dm)) + ’ ’ + sizes[i];
    }
});
```

```
from dataclasses import dataclass
from typing import Dict, List, Optional, Any

@dataclass
class DocumentClass:
    id: int
    name: str
    description: str
    fields: List[str]

# Define the document categories
DOCUMENT_CATEGORIES = [
    DocumentClass(
        id=1,
        name="Final Inspection Card",
        description="Certificate showing final building inspection approval",
        fields=["property_address", "fic_image", "non_fic_proof"]
    ),
    DocumentClass(
        id=2,
        name="Interconnection Agreement",
        description="Agreement between homeowner and utility company for grid connection"
,
        fields=["home_address", "homeowner_signature"]
    ),
    DocumentClass(
        id=3,
        name="PTO",
        description="Permission-To-Operate approval from utility",
        fields=["home_address", "pto_receive_date"]
    ),
    DocumentClass(
        id=4,
        name="Warranty Extension",
        description="Document extending warranty on solar equipment",
        fields=["warranty_proof", "serial_number"]
    ),
    DocumentClass(
        id=5,
        name="Interconnection / NEM Agreement",
        description="Agreement for Net Energy Metering and grid connection",
        fields=["document_name", "homeowner_signature", "utility_signature"]
    )
]

# Dictionary for quick lookup by ID
CATEGORIES_BY_ID = {cat.id: cat for cat in DOCUMENT_CATEGORIES}
CATEGORIES_BY_NAME = {cat.name: cat for cat in DOCUMENT_CATEGORIES}

@dataclass
class ProcessingResult:
    category: str
    extracted_fields: Dict[str, str]
    raw_text: str
    confidence: float
```

```
{% extends 'layout.html' %}

{% block content %}
<div class="row justify-content-center">
    <div class="col-md-10">
        <div class="card shadow-sm">
            <div class="card-header bg-success text-white">
                <div class="d-flex justify-content-between align-items-center">
                    <h3 class="mb-0">
                        <i class="fas fa-check-circle me-2"></i>Document Processing Resul
ts
                    </h3>
                    <a href="{{ url_for('index') }}" class="btn btn-outline-light btn-sm"
>
                        <i class="fas fa-upload me-1"></i>Process Another Document
                    </a>
                </div>
            </div>
            <div class="card-body">
                <div class="alert alert-info">
                    <i class="fas fa-info-circle me-2"></i>
                    <strong>Document Classification:</strong> {{ result.category }}
                    <span class="badge bg-primary ms-2">Confidence: {{ result.confidence
}}</span>
                </div>

                <div class="row mt-4">
                    <div class="col-md-6">
                        <div class="card mb-4">
                            <div class="card-header bg-primary text-white">
                                <h5 class="mb-0">
                                    <i class="fas fa-list-alt me-2"></i>Extracted Fields
                                </h5>
                            </div>
                            <div class="card-body">
                                {% if result.extracted_fields %}
                                    <div class="table-responsive">
                                        <table class="table table-striped">
                                            <thead>
                                                <tr>
                                                    <th>Field</th>
                                                    <th>Value</th>
                                                </tr>
                                            </thead>
                                            <tbody>
                                                {% for field, value in result.extracted_f
ields.items() %}
                                                <tr>
                                                    <td><strong>{{ field|replace('_', ' '
)|title }}</strong></td>
                                                    <td>{{ value }}</td>
                                                </tr>
                                                {% endfor %}
                                            </tbody>
                                        </table>
                                    </div>
                                {% else %}
                                    <div class="alert alert-warning">
                                        No fields were extracted from this document.
                                    </div>
                                {% endif %}
                            </div>
                        </div>
                    </div>
                    <div class="col-md-6">
                        <div class="card">
                            <div class="card-header bg-secondary text-white">
                                <h5 class="mb-0">
                                    <i class="fas fa-file-alt me-2"></i>Raw Document Text
                                </h5>
                            </div>
                            <div class="card-body">
                                <div class="raw-text-container p-3 border rounded bg-ligh
t text-dark" style="max-height: 400px; overflow-y: auto;">
                                    <pre>{{ result.raw_text }}</pre>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>

                <div class="card mt-4">
                    <div class="card-header bg-info text-white">
                        <h5 class="mb-0">
                            <i class="fas fa-code me-2"></i>JSON Response
                        </h5>
                    </div>
                    <div class="card-body">
                        <pre class="json-container p-3 border rounded bg-dark text-light"
 style="max-height: 300px; overflow-y: auto;">{{ result|tojson(indent=2) }}</pre>
                    </div>
                </div>

                <div class="text-center mt-4">
                    <a href="{{ url_for('index') }}" class="btn btn-primary">
                        <i class="fas fa-arrow-left me-2"></i>Back to Upload
                    </a>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

```
import os
import logging
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import joblib

# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
logger = logging.getLogger(__name__)

class DocumentClassifierTrainer:
    def __init__(self, data_dir="training_data"):
        self.data_dir = data_dir
        self.model_path = os.path.join(data_dir, "document_classifier.pkl")
        self.vectorizer_path = os.path.join(data_dir, "tfidf_vectorizer.pkl")

        # Create data dir if it doesn't exist
        os.makedirs(data_dir, exist_ok=True)

    def load_training_data(self):
        """
        Load training data from files

        In a real implementation, this would load text samples from
        a structured dataset. For this example, we'll create synthetic examples.
        """
        logger.info("Loading training data...")

        # This is a placeholder for real training data
        # In a real implementation, you'd load actual document text and labels

        texts = []
        labels = []

        # Examples for Final Inspection Card - Using real examples from provided document
s
        fic_examples = [
            "City of Selma BUILDING DIVISION Location of Work: 2252 BERRY ST, SELMA, CA 9
3662 SOLAR INSPECTIONS SOLAR- PANEL HOLD DOWNS SOLAR- UFER GROUND SOLAR - FINAL",
            "CITY OF CREST HILL, ILLINOIS BUILDING DEPARTMENT APPROVED Type of Inspection
 SOLAR FINAL Date of Inspection 2-28-25",
            "NOTICE THIS IS A SERVICE ONLY CERTIFICATE This is to certify that I have thi
s day inspected and approved service CITY ELECTRICAL INSPECTOR CITY OF GALESBURG STATE OF
 ILLINOIS",
            "MCCARTHY SOLAR PV ROOF MOUNTED SOLAR PV SYSTEM Building Final Pass 02/11/202
5 Electrical Final Pass 02/11/2025",
            "CITY OF RIVERSIDE BUILDING & SAFETY DIVISION PERMIT NUMBER BP-2024-18468 INS
TALL PV SOLAR SYSTEM PHOTOVOLTAIC PANELS ON RESIDENTIAL ROOF Date of Final 2/28/25",
            "FINAL INSPECTION CARD Building Department Approval for solar installation PA
SSED POST THIS CARD SO IT IS VISIBLE FROM STREET",
            "City of Yuba City Development Services Department BUILDING PERMIT Type: Resi
dential Permit Sub-Type: SOLAR (SOLAR/R) Category: WITHOUT STORAGE BATTERY",
            "I, Shelby Guenther, attest that this property is located within a jurisdicti
on in which permits are not required for the installation of solar panels.",
            "No permits no inspections moving to interconnection"
        ]
        for text in fic_examples:
            texts.append(text)
            labels.append(1)  # Category 1: Final Inspection Card

        # Examples for Interconnection Agreement - Based on real examples
        ia_examples = [
            "AGREEMENT FOR INTERCONNECTION AND PARALLEL OPERATION OF DISTRIBUTED GENERATI
ON Oncor Electric Delivery Company LLC",
            "Interconnection Agreement is made and entered into this day by Oncor Electri
c Delivery Company LLC and Customer",
            "Certificate of Completion Interconnection Customer Information Name Filomena
 Hernandez Street Address 682 Elsie Ave",
            "SOUTHERN CALIFORNIA EDISON COMPANY NET BILLING TARIFF OR NET ENERGY METERING
 AND RENEWABLE ELECTRICAL GENERATING FACILITY INTERCONNECTION AGREEMENT",
            "This Net Billing Tariff (NBT) or Net Energy Metering (NEM) and Renewable Ele
ctrical Generating Facility Interconnection Agreement",
            "AGREEMENT AND CUSTOMER AUTHORIZATION Net Billing Tariff (NBT) Interconnectio
n for Solar and/or Wind Electric Generating Facilities",
            "Application for Net Metering Services Ameren Illinois",
            "Acceptance and Final Approval for Interconnection The interconnection agreem
ent is approved",
            "Interconnection Agreement for Customer-Owned Renewable Generation Tier 1 - 1
0 kW or Less",
            "The interconnection customer acknowledges that it shall not operate the dist
ributed generation facility until receipt of the final acceptance"
        ]
        for text in ia_examples:
            texts.append(text)
            labels.append(2)  # Category 2: Interconnection Agreement

        # Examples for PTO (Permission-To-Operate)
        pto_examples = [
            "PERMISSION TO OPERATE Notice Date: 06/01/2023 Customer Address: 123 Main St"
,
            "Electric Utility Company PERMISSION TO OPERATE NOTIFICATION Solar Generation
 System",
            "PTO APPROVAL Your solar system at 456 Oak Avenue has been authorized to oper
ate",
            "Permission to Operate (PTO) Date Received: 07/15/2023 Solar Generator Addres
s: 789 Pine Rd",
            "NOTICE OF AUTHORIZATION TO OPERATE RENEWABLE GENERATING FACILITY PTO Date: 0
8/20/2023"
        ]
        for text in pto_examples:
            texts.append(text)
            labels.append(3)  # Category 3: PTO

        # Examples for Warranty Extension
        we_examples = [
            "Solar Panel Warranty Extension Certificate SolarEdge Inverter Serial Number:
 SE12345678",
            "EXTENDED WARRANTY DOCUMENT SolarEdge Technologies Inc. Product Warranty Exte
nsion",
            "SolarEdge Warranty Extension Confirmation Serial Number: SE98765432",
            "Product Warranty Extension Agreement SolarEdge Monitoring Portal Verificatio
n",
            "Solar System Extended Warranty Certificate for SolarEdge Components Serial #
: SE11223344"
        ]
        for text in we_examples:
            texts.append(text)
            labels.append(4)  # Category 4: Warranty Extension

        # Examples for Interconnection / NEM Agreement
        nem_examples = [
            "NET ENERGY METERING (NEM) INTERCONNECTION AGREEMENT Customer Signature: John
 Doe Utility Signature: Jane Smith",
            "SOLAR NEM AGREEMENT Application for Net Energy Metering Solar Facility",
            "Net Energy Metering (NEM) and Interconnection Agreement for Solar Generating
 Facility",
            "UTILITY COMPANY NEM INTERCONNECTION AGREEMENT Renewable Energy Credits (SREC
)",
            "Standard Net Energy Metering Agreement for Customer-Generator Facility"
        ]
        for text in nem_examples:
            texts.append(text)
            labels.append(5)  # Category 5: Interconnection / NEM Agreement

        logger.info(f"Loaded {len(texts)} training examples with {len(set(labels))} categ
ories")
```

```
        return texts, labels

    def train_model(self):
        """
        Train a document classifier model
        """
        try:
            # Load training data
            texts, labels = self.load_training_data()

            # Data augmentation - generate more training samples for each category
            # by creating slight variations of existing examples
            augmented_texts = []
            augmented_labels = []

            # Keep track of samples per class for better balancing
            sample_counts = {}
            for label in labels:
                sample_counts[label] = sample_counts.get(label, 0) + 1

            # Add the original samples
            augmented_texts.extend(texts)
            augmented_labels.extend(labels)

            # For each original text, create variations by removing random words,
            # changing capitalization, etc.
            import random
            for i, (text, label) in enumerate(zip(texts, labels)):
                # Only augment if this class has fewer than average samples
                avg_samples = len(texts) / len(set(labels))
                if sample_counts[label] < avg_samples * 1.2:
                    # Create 1-2 variations for underrepresented classes
                    num_variations = min(2, int(avg_samples - sample_counts[label]) + 1)

                    words = text.split()
                    for _ in range(num_variations):
                        if len(words) > 5:  # Only modify if there are enough words
                            # Remove 1-2 random words (except first and last 2 words)
                            variant_words = words.copy()
                            for _ in range(min(2, len(words) - 4)):
                                idx_to_remove = random.randint(2, len(variant_words) - 3)
                                variant_words.pop(idx_to_remove)

                            # Change capitalization of 1-2 words
                            for _ in range(min(2, len(variant_words))):
                                idx_to_modify = random.randint(0, len(variant_words) - 1)
                                if random.choice([True, False]):
                                    variant_words[idx_to_modify] = variant_words[idx_to_m
odify].upper()
                                else:
                                    variant_words[idx_to_modify] = variant_words[idx_to_m
odify].lower()

                            augmented_text = ' '.join(variant_words)
                            augmented_texts.append(augmented_text)
                            augmented_labels.append(label)
                            sample_counts[label] = sample_counts[label] + 1

            logger.info(f"Original samples: {len(texts)}, Augmented samples: {len(augment
ed_texts)}")

            # Split into train and test sets
            X_train, X_test, y_train, y_test = train_test_split(
                augmented_texts, augmented_labels, test_size=0.2, random_state=42
            )

            # Create and fit the vectorizer with enhanced parameters
            logger.info("Creating TF-IDF features...")
            vectorizer = TfidfVectorizer(
                max_features=5000,
                ngram_range=(1, 3),
                min_df=2,
                use_idf=True,
                sublinear_tf=True
            )
            X_train_tfidf = vectorizer.fit_transform(X_train)
            X_test_tfidf = vectorizer.transform(X_test)

            # Train the classifier with better parameters for small datasets
            logger.info("Training document classifier...")
            # Use higher C value to reduce regularization for small datasets
            classifier = LogisticRegression(
                C=5.0,
                random_state=42,
                max_iter=2000,
                class_weight='balanced',
                solver='liblinear',
                multi_class='ovr'
            )
            classifier.fit(X_train_tfidf, y_train)

            # Evaluate the model
            y_pred = classifier.predict(X_test_tfidf)
            accuracy = accuracy_score(y_test, y_pred)
            logger.info(f"Model accuracy: {accuracy:.4f}")
            logger.info("Classification report:\n" + classification_report(y_test, y_pred
))

            # Save the model and vectorizer
            logger.info(f"Saving model to {self.model_path}")
            joblib.dump(classifier, self.model_path)

            logger.info(f"Saving vectorizer to {self.vectorizer_path}")
            joblib.dump(vectorizer, self.vectorizer_path)

            logger.info("Training completed successfully")
            return True

        except Exception as e:
            logger.error(f"Error training model: {str(e)}", exc_info=True)
            return False

    def test_document_classification(self, test_text):
        """
        Test the trained model on a sample document
        """
        try:
            # Load the model and vectorizer
            classifier = joblib.load(self.model_path)
            vectorizer = joblib.load(self.vectorizer_path)

            # Transform the test text
            test_tfidf = vectorizer.transform([test_text])

            # Predict the category
            category_id = classifier.predict(test_tfidf)[0]
            probabilities = classifier.predict_proba(test_tfidf)[0]
            confidence = np.max(probabilities)

            # Map category ID to name
            category_names = {
                1: "Final Inspection Card",
                2: "Interconnection Agreement",
                3: "PTO",
                4: "Warranty Extension",
                5: "Interconnection / NEM Agreement"
            }

            category_name = category_names.get(category_id, "Unknown")
            return {
                "category": category_name,
                "confidence": confidence,
                "category_id": int(category_id)
            }

        except Exception as e:
            logger.error(f"Error testing document classification: {str(e)}", exc_info=Tru
e)
            return {
                "category": "Error",
                "confidence": 0.0,
                "error": str(e)
            }

if __name__ == "__main__":
    trainer = DocumentClassifierTrainer()
    trainer.train_model()

    # Test with a sample document
    test_sample = "PERMISSION TO OPERATE Notice Date: 10/01/2023 Your solar system at 123
 Example St has been approved"
    result = trainer.test_document_classification(test_sample)
    print(f"Test Sample Classification: {result}")
```