INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Fake Document Detection System</title>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.0/css/all.min.css">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    }

    body {
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      min-height: 100vh;
      padding: 20px;
    }

    .container {
      max-width: 1200px;
      margin: 0 auto;
      background: white;
      border-radius: 20px;
      box-shadow: 0 20px 60px rgba(0,0,0,0.3);
      overflow: hidden;
    }

    .header {
      background: linear-gradient(135deg, #1e3c72 0%, #2a5298 100%);
      color: white;
      padding: 30px;
      text-align: center;
    }

    .header h1 {
      font-size: 2.5em;
      margin-bottom: 10px;
    }

    .header p {
      opacity: 0.9;
```

```
    font-size: 1.1em;
}

.main-content {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 30px;
    padding: 30px;
}

@media (max-width: 900px) {
    .main-content {
        grid-template-columns: 1fr;
    }
}

.upload-section {
    background: #f8f9fa;
    border-radius: 15px;
    padding: 30px;
    border: 2px dashed #667eea;
    transition: all 0.3s;
}

.upload-section:hover {
    border-color: #764ba2;
    transform: translateY(-2px);
}

.upload-box {
    text-align: center;
    padding: 40px 20px;
}

.upload-icon {
    font-size: 4em;
    color: #667eea;
    margin-bottom: 20px;
}

.file-input {
    display: none;
}

.file-label {
    display: inline-block;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
```

```
    padding: 15px 30px;
    border-radius: 50px;
    cursor: pointer;
    font-size: 1.1em;
    font-weight: 600;
    transition: all 0.3s;
    margin-bottom: 20px;
}

.file-label:hover {
    transform: scale(1.05);
    box-shadow: 0 10px 20px rgba(102, 126, 234, 0.4);
}

.file-info {
    margin-top: 20px;
    padding: 15px;
    background: white;
    border-radius: 10px;
    border: 1px solid #dee2e6;
    text-align: left;
}

.analyze-btn {
    background: linear-gradient(135deg, #00b09b 0%, #96c93d 100%);
    color: white;
    border: none;
    padding: 15px 40px;
    border-radius: 50px;
    font-size: 1.2em;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.3s;
    margin-top: 20px;
    width: 100%;
}

.analyze-btn:hover:not(:disabled) {
    transform: scale(1.05);
    box-shadow: 0 10px 20px rgba(0, 176, 155, 0.4);
}

.analyze-btn:disabled {
    opacity: 0.5;
    cursor: not-allowed;
}

.results-section {
```

```
    background: #f8f9fa;
    border-radius: 15px;
    padding: 30px;
}

.results-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 30px;
    padding-bottom: 15px;
    border-bottom: 2px solid #dee2e6;
}

.results-header h2 {
    color: #1e3c72;
}

.risk-badge {
    padding: 8px 20px;
    border-radius: 50px;
    font-weight: 600;
    font-size: 1.1em;
}

.risk-low { background: #d4edda; color: #155724; }
.risk-medium { background: #fff3cd; color: #856404; }
.risk-high { background: #f8d7da; color: #721c24; }

.score-display {
    display: flex;
    justify-content: space-around;
    margin-bottom: 30px;
}

.score-box {
    text-align: center;
    padding: 20px;
    border-radius: 15px;
    background: white;
    box-shadow: 0 5px 15px rgba(0,0,0,0.08);
    flex: 1;
    margin: 0 10px;
}

.score-value {
    font-size: 2.5em;
    font-weight: 700;
```

```
    margin: 10px 0;
}

.authenticity-score .score-value { color: #00b09b; }
.risk-score .score-value { color: #dc3545; }

.checks-list {
    max-height: 400px;
    overflow-y: auto;
    margin-bottom: 30px;
}

.check-item {
    background: white;
    padding: 15px;
    margin-bottom: 10px;
    border-radius: 10px;
    border-left: 5px solid #dee2e6;
}

.check-item.passed {
    border-left-color: #00b09b;
}

.check-item.failed {
    border-left-color: #dc3545;
}

.check-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 10px;
}

.check-status {
    font-weight: 600;
    padding: 3px 10px;
    border-radius: 20px;
    font-size: 0.9em;
}

.status-pass { background: #d4edda; color: #155724; }
.status-fail { background: #f8d7da; color: #721c24; }

.check-details {
    font-size: 0.9em;
    color: #6c757d;
```

```
    margin-top: 5px;
}

.recommendations {
    background: white;
    padding: 20px;
    border-radius: 15px;
    margin-top: 30px;
}

.recommendations h3 {
    color: #1e3c72;
    margin-bottom: 15px;
    padding-bottom: 10px;
    border-bottom: 2px solid #dee2e6;
}

.recommendation-item {
    padding: 10px 0;
    border-bottom: 1px solid #f8f9fa;
}

.recommendation-item:last-child {
    border-bottom: none;
}

.loading {
    display: none;
    text-align: center;
    padding: 40px;
}

.spinner {
    width: 50px;
    height: 50px;
    border: 5px solid #f3f3f3;
    border-top: 5px solid #667eea;
    border-radius: 50%;
    animation: spin 1s linear infinite;
    margin: 0 auto 20px;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

.error-message {
```

```
        background: #f8d7da;
        color: #721c24;
        padding: 15px;
        border-radius: 10px;
        margin: 20px 0;
        display: none;
      }

      .chart-container {
        height: 200px;
        margin: 20px 0;
      }
    </style>
</head>
<body>
    <div class="container">
      <div class="header">
        <h1><i class="fas fa-passport"></i> Fake Document Detection System</h1>
        <p>Upload PDFs, passports, IDs, or driver licenses for authenticity verification</p>
      </div>

      <div class="main-content">
        <div class="upload-section">
          <div class="upload-box">
            <div class="upload-icon">
              <i class="fas fa-file-upload"></i>
            </div>
            <h2>Upload Document</h2>
            <p>Supported formats: PDF, PNG, JPG, JPEG</p>

            <input type="file" id="fileInput" class="file-input" accept=".pdf,.png,.jpg,.jpeg">
            <label for="fileInput" class="file-label">
              <i class="fas fa-cloud-upload-alt"></i> Choose File
            </label>

            <div class="file-info" id="fileInfo" style="display: none;">
              <p><strong>Selected File:</strong> <span id="fileName"></span></p>
              <p><strong>File Size:</strong> <span id="fileSize"></span></p>
              <p><strong>Type:</strong> <span id="fileType"></span></p>
            </div>

            <button id="analyzeBtn" class="analyze-btn" disabled>
              <i class="fas fa-search"></i> Analyze Document
            </button>
          </div>

          <div class="loading" id="loading">
            <div class="spinner"></div>
```

```
      <h3>Analyzing Document...</h3>
      <p>Checking for inconsistencies and security features</p>
    </div>

    <div class="error-message" id="errorMessage"></div>
  </div>

  <div class="results-section">
    <div class="results-header">
      <h2><i class="fas fa-clipboard-check"></i> Analysis Results</h2>
      <div id="riskBadge" class="risk-badge" style="display: none;">
        Risk: <span id="riskLevel">Low</span>
      </div>
    </div>

    <div id="noResults" style="text-align: center; padding: 60px 20px; color: #6c757d;">
      <i class="fas fa-file-alt" style="font-size: 4em; margin-bottom: 20px; opacity: 0.5;"></i>
      <h3>No Document Analyzed Yet</h3>
      <p>Upload a document to begin analysis</p>
    </div>

    <div id="resultsContent" style="display: none;">
      <div class="score-display">
        <div class="score-box authenticity-score">
          <h4>Authenticity Score</h4>
          <div class="score-value" id="authenticityScore">0%</div>
          <p>Document Trustworthiness</p>
        </div>
        <div class="score-box risk-score">
          <h4>Risk Score</h4>
          <div class="score-value" id="riskScore">0%</div>
          <p>Forgery Probability</p>
        </div>
      </div>

      <div class="chart-container">
        <canvas id="resultsChart"></canvas>
      </div>

      <h3><i class="fas fa-tasks"></i> Security Checks</h3>
      <div class="checks-list" id="checksList"></div>

      <div class="recommendations">
        <h3><i class="fas fa-lightbulb"></i> Recommendations</h3>
        <div id="recommendationsList"></div>
      </div>
    </div>
  </div>
```

```
    </div>
  </div>

  <script>
    let selectedFile = null;
    let chartInstance = null;

    // File input handler
    document.getElementById('fileInput').addEventListener('change', function(e) {
      if (this.files.length > 0) {
        selectedFile = this.files[0];

        // Update file info display
        document.getElementById('fileName').textContent = selectedFile.name;
        document.getElementById('fileSize').textContent = formatFileSize(selectedFile.size);
        document.getElementById('fileType').textContent = selectedFile.type ||
selectedFile.name.split('.').pop().toUpperCase();
        document.getElementById('fileInfo').style.display = 'block';

        // Enable analyze button
        document.getElementById('analyzeBtn').disabled = false;
      }
    });

    // Analyze button handler
    document.getElementById('analyzeBtn').addEventListener('click', function() {
      if (!selectedFile) return;

      // Show loading, hide results and errors
      document.getElementById('loading').style.display = 'block';
      document.getElementById('errorMessage').style.display = 'none';
      document.getElementById('analyzeBtn').disabled = true;

      // Create form data
      const formData = new FormData();
      formData.append('file', selectedFile);

      // Send to backend
      fetch('/analyze', {
        method: 'POST',
        body: formData
      })
      .then(response => response.json())
      .then(data => {
        document.getElementById('loading').style.display = 'none';

        if (data.error) {
          showError(data.error);
```

```
            return;
        }

        displayResults(data);
    })
    .catch(error => {
        document.getElementById('loading').style.display = 'none';
        document.getElementById('analyzeBtn').disabled = false;
        showError('Network error. Please try again.');
        console.error('Error:', error);
    });
});

function displayResults(data) {
    // Show results section
    document.getElementById('noResults').style.display = 'none';
    document.getElementById('resultsContent').style.display = 'block';

    // Update scores
    document.getElementById('authenticityScore').textContent =
        data.authenticity_score.toFixed(1) + '%';
    document.getElementById('riskScore').textContent =
        data.overall_risk_score.toFixed(1) + '%';

    // Update risk badge
    const riskBadge = document.getElementById('riskBadge');
    const riskLevel = document.getElementById('riskLevel');

    if (data.overall_risk_score < 30) {
        riskBadge.className = 'risk-badge risk-low';
        riskLevel.textContent = 'Low';
    } else if (data.overall_risk_score < 70) {
        riskBadge.className = 'risk-badge risk-medium';
        riskLevel.textContent = 'Medium';
    } else {
        riskBadge.className = 'risk-badge risk-high';
        riskLevel.textContent = 'High';
    }

    riskBadge.style.display = 'flex';

    // Display checks
    const checksList = document.getElementById('checksList');
    checksList.innerHTML = '';

    data.detailed_results.forEach(check => {
        const checkItem = document.createElement('div');
        checkItem.className = `check-item ${check.passed ? 'passed' : 'failed'}`;
```

```
        checkItem.innerHTML = `
            <div class="check-header">
                <strong>${check.check_name.replace(/_/g, ' ')}</strong>
                <span class="check-status ${check.passed ? 'status-pass' : 'status-fail'}">
                    ${check.passed ? 'PASS' : 'FAIL'}
                </span>
            </div>
            <div class="check-details">
                ${check.details} | Risk: ${check.risk_score.toFixed(1)}%
            </div>
        `;

        checksList.appendChild(checkItem);
    });

    // Display recommendations
    const recommendationsList = document.getElementById('recommendationsList');
    recommendationsList.innerHTML = '';

    if (data.recommendations && data.recommendations.length > 0) {
        data.recommendations.forEach(rec => {
            const recItem = document.createElement('div');
            recItem.className = 'recommendation-item';
            recItem.innerHTML = `<i class="fas fa-chevron-right"></i> ${rec}`;
            recommendationsList.appendChild(recItem);
        });
    } else {
        recommendationsList.innerHTML = '<p>No specific recommendations.</p>';
    }

    // Create chart
    createChart(data);

    // Enable analyze button again
    document.getElementById('analyzeBtn').disabled = false;
}

function createChart(data) {
    const ctx = document.getElementById('resultsChart').getContext('2d');

    // Destroy previous chart if exists
    if (chartInstance) {
        chartInstance.destroy();
    }

    // Prepare data for chart
    const passedChecks = data.passed_checks || 0;
```

```
const failedChecks = (data.total_checks || 0) - passedChecks;

chartInstance = new Chart(ctx, {
  type: 'doughnut',
  data: {
    labels: ['Passed Checks', 'Failed Checks', 'Risk Level'],
    datasets: [{
      data: [passedChecks, failedChecks, data.overall_risk_score / 10],
      backgroundColor: [
        '#00b09b',
        '#dc3545',
        data.overall_risk_score < 30 ? '#28a745' :
        data.overall_risk_score < 70 ? '#ffc107' : '#dc3545'
      ],
      borderWidth: 2,
      borderColor: '#fff'
    }]
  },
  options: {
    responsive: true,
    maintainAspectRatio: false,
    plugins: {
      legend: {
        position: 'bottom',
        labels: {
          padding: 20,
          font: {
            size: 12
          }
        }
      },
      tooltip: {
        callbacks: {
          label: function(context) {
            let label = context.label || '';
            if (label) {
              label += ': ';
            }
            if (context.label === 'Risk Level') {
              label += data.overall_risk_score.toFixed(1) + '%';
            } else {
              label += context.raw;
            }
            return label;
          }
        }
      }
    }
```

```
        }
      });
    }

    function showError(message) {
      const errorDiv = document.getElementById('errorMessage');
      errorDiv.textContent = message;
      errorDiv.style.display = 'block';

      // Re-enable analyze button
      document.getElementById('analyzeBtn').disabled = false;
    }

    function formatFileSize(bytes) {
      if (bytes === 0) return '0 Bytes';

      const k = 1024;
      const sizes = ['Bytes', 'KB', 'MB', 'GB'];
      const i = Math.floor(Math.log(bytes) / Math.log(k));

      return parseFloat((bytes / Math.pow(k, i)).toFixed(2)) + ' ' + sizes[i];
    }

    // Health check on load
    fetch('/api/health')
      .then(response => response.json())
      .then(data => {
        console.log('Backend status:', data.status);
      })
      .catch(error => {
        console.warn('Backend not responding:', error);
      });
  </script>
</body>
</html>
```

PYTHON

```
import os
import json
import tempfile
import traceback
from datetime import datetime
from flask import Flask, request, jsonify, render_template, send_from_directory
from flask_cors import CORS
import pdf2image
import pytesseract
from PIL import Image
import cv2
import numpy as np
import re
from werkzeug.utils import secure_filename
import PyPDF2
import io

# Import our document detector
from document_detector import FakeDocumentDetector

app = Flask(__name__)
CORS(app)
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # 16MB max upload
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['ALLOWED_EXTENSIONS'] = {'pdf', 'png', 'jpg', 'jpeg'}

# Initialize detector
detector = FakeDocumentDetector()

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

def extract_text_from_pdf(pdf_path):
    """Extract text from PDF using OCR and direct text extraction"""
    text_content = ""

    try:
        # First try direct text extraction
        with open(pdf_path, 'rb') as file:
            pdf_reader = PyPDF2.PdfReader(file)
            for page_num in range(len(pdf_reader.pages)):
                page = pdf_reader.pages[page_num]
                text_content += page.extract_text() + "\n"
```

```
        # If no text found or minimal text, use OCR
        if len(text_content.strip()) < 50:
            text_content = ""
            # Convert PDF to images for OCR
            images = pdf2image.convert_from_path(pdf_path, dpi=300)

            for i, image in enumerate(images):
                # Convert PIL image to OpenCV format
                open_cv_image = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)

                # Preprocess for better OCR
                gray = cv2.cvtColor(open_cv_image, cv2.COLOR_BGR2GRAY)
                denoised = cv2.medianBlur(gray, 3)

                # OCR with Tesseract
                page_text = pytesseract.image_to_string(denoised)
                text_content += f"\n--- Page {i+1} ---\n" + page_text + "\n"

    except Exception as e:
        print(f"PDF extraction error: {str(e)}")
        # Fallback to basic OCR
        try:
            images = pdf2image.convert_from_path(pdf_path, dpi=200)
            for image in images:
                text_content += pytesseract.image_to_string(image) + "\n"
        except:
            text_content = "Error extracting text from document"

    return text_content

def extract_document_data_from_text(text):
    """Parse document data from extracted text"""
    data = {}

    # Convert to uppercase for easier matching
    text_upper = text.upper()

    # Extract document number/ID
    id_patterns = [
        r'NO[:\s]*([A-Z0-9]{6,15})',
        r'NUMBER[:\s]*([A-Z0-9]{6,15})',
        r'ID[:\s]*([A-Z0-9]{6,15})',
        r'PASSPORT[:\s]*([A-Z0-9]{6,15})',
        r'LICENSE[:\s]*([A-Z0-9]{6,15})'
    ]

    for pattern in id_patterns:
        match = re.search(pattern, text_upper)
```

```
    if match:
        data['id_number'] = match.group(1)
        break

# Extract dates
date_patterns = [
    r'(\d{1,2}[/\-]\d{1,2}[/\-]\d{2,4})',
    r'(\d{4}[/\-]\d{1,2}[/\-]\d{1,2})',
    r'(?:DOB|BIRTH)[:\s]*(\d{1,2}[/\-]\d{1,2}[/\-]\d{2,4})',
    r'(?:ISSUE|ISSUED)[:\s]*(\d{1,2}[/\-]\d{1,2}[/\-]\d{2,4})',
    r'(?:EXPIR|EXPIRY|VALID)[:\s]*(\d{1,2}[/\-]\d{1,2}[/\-]\d{2,4})'
]

dates_found = []
for pattern in date_patterns:
    matches = re.findall(pattern, text)
    dates_found.extend(matches)

# Try to categorize dates
if len(dates_found) >= 1:
    data['date_of_birth'] = dates_found[0] if len(dates_found) > 0 else None
    data['date_of_issue'] = dates_found[1] if len(dates_found) > 1 else None
    data['date_of_expiry'] = dates_found[2] if len(dates_found) > 2 else None

# Extract names
name_patterns = [
    r'NAME[:\s]*([A-Z\s]{3,50})',
    r'SURNAME[:\s]*([A-Z\s]{3,50})',
    r'GIVEN[:\s]*([A-Z\s]{3,50})',
    r'FULL[:\s]*NAME[:\s]*([A-Z\s]{3,50})'
]

for pattern in name_patterns:
    match = re.search(pattern, text_upper)
    if match:
        data['full_name'] = match.group(1).strip()
        break

# Extract nationality
nat_match = re.search(r'NATIONALITY[:\s]*([A-Z\s]{3,30})', text_upper)
if nat_match:
    data['nationality'] = nat_match.group(1).strip()

# Extract authority
auth_match = re.search(r'AUTHORITY[:\s]*([A-Z\s]{3,50})', text_upper)
if auth_match:
    data['authority'] = auth_match.group(1).strip()
```

```
    return data

def simulate_ocr_metadata(text):
    """Simulate OCR metadata for quality assessment"""
    # In real system, you would get this from the OCR engine
    words = text.split()
    char_count = len(text)
    word_count = len(words)

    # Simulate confidence scores (in real system, get from pytesseract)
    avg_confidence = min(0.85 + (len([w for w in words if len(w) > 3]) / max(word_count, 1)) * 0.15,
0.98)

    # Simulate font detection based on text characteristics
    fonts_detected = []
    if sum(1 for c in text if c.isupper()) / max(char_count, 1) > 0.7:
        fonts_detected.append("OCR-A")
    if any(word in text.upper() for word in ["MRZ", "PASSPORT", "OFFICIAL"]):
        fonts_detected.append("DocumentFont")

    return {
        "confidence_scores": [avg_confidence] * min(word_count, 10),
        "fonts_detected": fonts_detected or ["UnknownFont"],
        "text_angles": [0.0, 0.1, -0.2, 0.3, 0.0],  # Simulated
        "word_count": word_count,
        "char_count": char_count,
        "avg_word_length": sum(len(w) for w in words) / max(word_count, 1) if word_count > 0 else 0
    }

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

@app.route('/analyze', methods=['POST'])
def analyze_document():
    if 'file' not in request.files:
        return jsonify({'error': 'No file uploaded'}), 400

    file = request.files['file']

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

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

    try:
        # Save uploaded file
```

```
    filename = secure_filename(file.filename)
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    unique_filename = f"{timestamp}_{filename}"
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], unique_filename)
    file.save(filepath)

    # Extract text from document
    if filename.lower().endswith('.pdf'):
      extracted_text = extract_text_from_pdf(filepath)
    else:
      # Handle image files
      image = Image.open(filepath)
      extracted_text = pytesseract.image_to_string(image)

    # Parse document data from text
    document_data = extract_document_data_from_text(extracted_text)

    # Generate OCR metadata
    ocr_metadata = simulate_ocr_metadata(extracted_text)

    # Add extracted text to metadata
    ocr_metadata['extracted_text_preview'] = extracted_text[:500] + "..." if len(extracted_text) > 500
else extracted_text
    ocr_metadata['total_text_length'] = len(extracted_text)

    # Run document analysis
    analysis_result = detector.analyze_document(
      extracted_text,
      document_data,
      ocr_metadata
    )

    # Add file information to result
    analysis_result['file_info'] = {
      'filename': filename,
      'upload_time': timestamp,
      'file_size': os.path.getsize(filepath),
      'text_extracted': len(extracted_text.strip()) > 0
    }

    # Add extracted data preview
    analysis_result['extracted_data'] = document_data
    analysis_result['text_preview'] = extracted_text[:200] + "..." if len(extracted_text) > 200 else
extracted_text

    # Clean up uploaded file
    os.remove(filepath)
```

```
        return jsonify(analysis_result)

    except Exception as e:
        print(f"Analysis error: {str(e)}")
        print(traceback.format_exc())
        return jsonify({
            'error': f'Analysis failed: {str(e)}',
            'traceback': traceback.format_exc() if app.debug else None
        }), 500

@app.route('/api/health')
def health_check():
    return jsonify({'status': 'healthy', 'timestamp': datetime.now().isoformat()})

if __name__ == '__main__':
    # Create uploads directory if it doesn't exist
    os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

    app.run(debug=True, port=5000)
```