I'll help you create a chessboard OCR application using PHP and HTML. This will be a web-based application that users can access through their browser.

## Project Structure

```
chess-ocr/
├── index.html          # Main upload page
├── upload.php          # File upload handler
├── process.php         # OCR processing logic
├── ocr_functions.php   # Core OCR functions
├── uploads/            # Uploaded images directory
├── assets/
|   ├── css/
|   |   └── style.css  # Styling
|   └── js/
|       └── script.js   # Frontend JavaScript
└── temp/               # Temporary processing directory
```

## 1. **HTML Interface (index.html)**

```html
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Chess Board OCR</title>
   <link rel="stylesheet" href="assets/css/style.css">
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">
</head>
```

```
<body>
  <div class="container">
    <header>
      <h1><i class="fas fa-chess-board"></i> Chess Board OCR</h1>
      <p>Upload a chessboard image and get the FEN notation automatically</p>
    </header>

    <div class="main-content">
      <div class="upload-section">
        <div class="upload-area" id="dropArea">
          <i class="fas fa-cloud-upload-alt upload-icon"></i>
          <h3>Drag & Drop Chessboard Image</h3>
          <p>or click to browse</p>
          <input type="file" id="fileInput" accept="image/*" capture="environment">
          <p class="formats">Supported formats: JPG, PNG, WebP (Max 5MB)</p>
        </div>

        <div class="preview-section" style="display: none;">
          <h3>Preview</h3>
          <div class="image-preview">
            <img id="previewImage" src="" alt="Chessboard preview">
            <div class="preview-controls">
              <button id="rotateLeft" class="btn-icon"><i class="fas fa-
undo"></i></button>
              <button id="rotateRight" class="btn-icon"><i class="fas fa-
redo"></i></button>
              <button id="cropBtn" class="btn-icon"><i class="fas fa-crop"></i></button>
              <button id="removeBtn" class="btn-icon"><i class="fas fa-
trash"></i></button>
            </div>
          </div>
        <div class="actions">
          <button id="processBtn" class="btn-primary">
            <i class="fas fa-cogs"></i> Process Image
          </button>
        </div>
      </div>
    </div>

    <div class="results-section" style="display: none;">
      <h3><i class="fas fa-chess"></i> OCR Results</h3>

      <div class="result-grid">
        <div class="result-card">
          <h4><i class="fas fa-list"></i> Detected Board</h4>
          <div id="boardVisualization" class="board-visual">
            <!-- Chessboard will be rendered here -->
          </div>
        </div>

        <div class="result-card">
          <h4><i class="fas fa-code"></i> FEN Notation</h4>
          <div class="fen-container">
            <textarea id="fenOutput" readonly></textarea>
            <button id="copyFEN" class="btn-secondary">
              <i class="far fa-copy"></i> Copy
            </button>
          </div>

          <div class="actions">
            <button id="validateBtn" class="btn-success">
              <i class="fas fa-check-circle"></i> Validate FEN
```

```
                    </button>
                    <button id="lichessBtn" class="btn-lichess">
                        <i class="fas fa-external-link-alt"></i> Open in Lichess
                    </button>
                </div>
            </div>
        </div>

        <div class="debug-info">
            <h4><i class="fas fa-bug"></i> Processing Details</h4>
            <div id="debugOutput"></div>
        </div>
    </div>
  </div>

  <div class="tips">
    <h4><i class="fas fa-lightbulb"></i> Tips for Best Results</h4>
    <ul>
        <li>Use good lighting and avoid shadows on the board</li>
        <li>Take picture from directly above the board</li>
        <li>Make sure all 64 squares are visible</li>
        <li>Use standard Staunton chess pieces</li>
        <li>Clear background works best</li>
    </ul>
  </div>
</div>

<div id="loadingOverlay" style="display: none;">
  <div class="spinner"></div>
  <p>Processing chessboard...</p>
</div>
```

```
    <script src="assets/js/script.js"></script>
</body>
</html>
```

## 2. **CSS Styling (assets/css/style.css)**

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    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, #2c3e50 0%, #4a6491 100%);
    color: white;
    padding: 30px;
    text-align: center;
}

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

header p {
    opacity: 0.9;
    font-size: 1.1rem;
}

.main-content {
    padding: 30px;
}

.upload-section {
    margin-bottom: 40px;
}

.upload-area {
    border: 3px dashed #667eea;
    border-radius: 15px;
    padding: 60px 20px;
    text-align: center;
    cursor: pointer;
    transition: all 0.3s ease;
    background: #f8f9fa;
    position: relative;
}

.upload-area:hover {
    background: #eef2ff;
    border-color: #764ba2;
}

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

.upload-area h3 {
    color: #2c3e50;
    margin-bottom: 10px;
}

.upload-area p {
    color: #666;
    margin-bottom: 20px;
}

#fileInput {
    position: absolute;
    width: 100%;
    height: 100%;
    top: 0;
    left: 0;
    opacity: 0;
    cursor: pointer;
}

.formats {
    font-size: 0.9rem;
    color: #888;
    margin-top: 20px;
}

.preview-section {
    margin-top: 30px;
}

.image-preview {
    border: 2px solid #ddd;
    border-radius: 10px;
    padding: 20px;
    text-align: center;
    background: #f8f9fa;
}

#previewImage {
    max-width: 100%;
    max-height: 400px;
    border-radius: 8px;
    box-shadow: 0 5px 15px rgba(0,0,0,0.1);
}

.preview-controls {
    margin-top: 15px;
    display: flex;
    gap: 10px;
    justify-content: center;
}

.btn-icon {
    background: #667eea;
    color: white;
    border: none;
    width: 40px;
    height: 40px;
    border-radius: 50%;
    cursor: pointer;
    transition: all 0.3s ease;
}

.btn-icon:hover {
    background: #764ba2;
    transform: scale(1.1);
}

.actions {
    margin-top: 20px;
    text-align: center;
}

.btn-primary, .btn-secondary, .btn-success, .btn-lichess {
    padding: 12px 24px;
    border: none;
    border-radius: 8px;
```

```css
    font-size: 1rem;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.3s ease;
    display: inline-flex;
    align-items: center;
    gap: 10px;
}

.btn-primary {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
}

.btn-primary:hover {
    transform: translateY(-2px);
    box-shadow: 0 10px 20px rgba(102, 126, 234, 0.4);
}

.btn-secondary {
    background: #6c757d;
    color: white;
}

.btn-success {
    background: #28a745;
    color: white;
}

.btn-lichess {
    background: #333;
    color: white;
}

.btn-secondary:hover, .btn-success:hover, .btn-lichess:hover {
    transform: translateY(-2px);
    opacity: 0.9;
}

.results-section {
    animation: slideUp 0.5s ease;
}

@keyframes slideUp {
    from {
        opacity: 0;
        transform: translateY(20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

.result-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    gap: 30px;
    margin-top: 20px;
}

.result-card {
```

```css
    background: #f8f9fa;
    border-radius: 15px;
    padding: 25px;
    box-shadow: 0 5px 15px rgba(0,0,0,0.05);
}

.result-card h4 {
    color: #2c3e50;
    margin-bottom: 20px;
    padding-bottom: 10px;
    border-bottom: 2px solid #667eea;
}

.board-visual {
    width: 100%;
    aspect-ratio: 1;
    background: #f0f0f0;
    border-radius: 8px;
    overflow: hidden;
}

.fen-container {
    margin-top: 20px;
}

#fenOutput {
    width: 100%;
    height: 120px;
    padding: 15px;
    border: 2px solid #ddd;
    border-radius: 8px;
    font-family: 'Courier New', monospace;
    font-size: 1rem;
    resize: none;
    background: #fff;
}

.debug-info {
    margin-top: 30px;
    padding: 20px;
    background: #f8f9fa;
    border-radius: 10px;
    border-left: 4px solid #667eea;
}

#debugOutput {
    font-family: 'Courier New', monospace;
    font-size: 0.9rem;
    color: #333;
    white-space: pre-wrap;
    background: white;
    padding: 15px;
    border-radius: 5px;
    margin-top: 10px;
    max-height: 200px;
    overflow-y: auto;
}

.tips {
    background: linear-gradient(135deg, #fff9c4 0%, #ffecb3 100%);
    padding: 25px;
    margin-top: 30px;
    border-radius: 15px;
    border-left: 5px solid #ffb300;
}

.tips h4 {
    color: #5d4037;
    margin-bottom: 15px;
}

.tips ul {
    list-style: none;
    padding-left: 20px;
}

.tips li {
    padding: 8px 0;
    color: #5d4037;
    position: relative;
}

.tips li:before {
    content: "✓";
    color: #4caf50;
    font-weight: bold;
    position: absolute;
    left: -20px;
}

#loadingOverlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.8);
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    z-index: 1000;
    color: white;
}

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

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

@media (max-width: 768px) {
    .container {
        margin: 10px;
        border-radius: 10px;
    }

    header {
        padding: 20px;
    }

    header h1 {
        font-size: 1.8rem;
    }

    .main-content {
        padding: 20px;
    }

    .result-grid {
        grid-template-columns: 1fr;
    }
}
```

## 3. **JavaScript (assets/js/script.js)**

```javascript
class ChessOCRApp {
    constructor() {
        this.image = null;
        this.rotation = 0;
        this.init();
    }

    init() {
        this.bindEvents();
        this.initDropzone();
    }

    bindEvents() {
        const fileInput = document.getElementById('fileInput');
        const processBtn = document.getElementById('processBtn');
        const copyBtn = document.getElementById('copyFEN');
        const validateBtn = document.getElementById('validateBtn');
        const lichessBtn = document.getElementById('lichessBtn');
        const rotateLeft = document.getElementById('rotateLeft');
        const rotateRight = document.getElementById('rotateRight');
        const removeBtn = document.getElementById('removeBtn');

        fileInput.addEventListener('change', (e) => this.handleFileSelect(e));
        processBtn.addEventListener('click', () => this.processImage());
        copyBtn.addEventListener('click', () => this.copyFEN());
        validateBtn.addEventListener('click', () => this.validateFEN());
        lichessBtn.addEventListener('click', () => this.openInLichess());
        rotateLeft.addEventListener('click', () => this.rotateImage(-90));
        rotateRight.addEventListener('click', () => this.rotateImage(90));
        removeBtn.addEventListener('click', () => this.removeImage());
    }

    initDropzone() {
        const dropArea = document.getElementById('dropArea');

        ['dragenter', 'dragover', 'dragleave', 'drop'].forEach(eventName => {
            dropArea.addEventListener(eventName, preventDefaults, false);
        });
        function preventDefaults(e) {
            e.preventDefault();
            e.stopPropagation();
        }

        ['dragenter', 'dragover'].forEach(eventName => {
            dropArea.addEventListener(eventName, highlight, false);
        });

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

        function highlight() {
            dropArea.style.background = '#eef2ff';
            dropArea.style.borderColor = '#764ba2';
        }

        function unhighlight() {
            dropArea.style.background = '#f8f9fa';
            dropArea.style.borderColor = '#667eea';
        }

        dropArea.addEventListener('drop', (e) => {
            const dt = e.dataTransfer;
            const files = dt.files;
            this.handleFiles(files);
        });
    }

    handleFileSelect(e) {
```

```
    const files = e.target.files;
    this.handleFiles(files);
}

handleFiles(files) {
    if (files.length === 0) return;

    const file = files[0];
    if (!file.type.match('image.*')) {
        alert('Please select an image file');
        return;
    }

    if (file.size > 5 * 1024 * 1024) {
        alert('File size must be less than 5MB');
        return;
    }

    const reader = new FileReader();
    reader.onload = (e) => {
        this.image = new Image();
        this.image.onload = () => {
            this.showPreview(this.image);
        };
        this.image.src = e.target.result;
    };
    reader.readAsDataURL(file);
}

showPreview(image) {
    const previewSection = document.querySelector('.preview-section');
    const previewImage = document.getElementById('previewImage');

    previewImage.src = image.src;
    previewSection.style.display = 'block';
    this.updatePreview();
}

rotateImage(degrees) {
    this.rotation += degrees;
    this.updatePreview();
}

updatePreview() {
    const previewImage = document.getElementById('previewImage');
    previewImage.style.transform = `rotate(${this.rotation}deg)`;
}

removeImage() {
    const fileInput = document.getElementById('fileInput');
    const previewSection = document.querySelector('.preview-section');
    const resultsSection = document.querySelector('.results-section');

    fileInput.value = '';
    previewSection.style.display = 'none';
    resultsSection.style.display = 'none';
    this.image = null;
    this.rotation = 0;
}

async processImage() {
    if (!this.image) {
```

```
    alert('Please select an image first');
    return;
}

this.showLoading(true);

try {
    // Create FormData to send to PHP
    const formData = new FormData();
    const fileInput = document.getElementById('fileInput');
    formData.append('image', fileInput.files[0]);
    formData.append('rotation', this.rotation);

    const response = await fetch('process.php', {
        method: 'POST',
        body: formData
    });

    const result = await response.json();

    if (result.success) {
        this.displayResults(result);
    } else {
        throw new Error(result.error || 'Processing failed');
    }
} catch (error) {
    console.error('Error:', error);
    alert('Error processing image: ' + error.message);
} finally {
    this.showLoading(false);
}
```

```
}

displayResults(result) {
    const resultsSection = document.querySelector('.results-section');
    const fenOutput = document.getElementById('fenOutput');
    const debugOutput = document.getElementById('debugOutput');

    // Show results section
    resultsSection.style.display = 'block';

    // Scroll to results
    resultsSection.scrollIntoView({ behavior: 'smooth' });

    // Display FEN
    fenOutput.value = result.fen;

    // Display debug info
    debugOutput.innerHTML = '';
    for (const [key, value] of Object.entries(result.debug || {})) {
        const div = document.createElement('div');
        div.innerHTML = `<strong>${key}:</strong> ${value}`;
        debugOutput.appendChild(div);
    }

    // Render chessboard visualization
    this.renderChessboard(result.pieces || []);
}

renderChessboard(pieces) {
    const boardVisual = document.getElementById('boardVisualization');
    boardVisual.innerHTML = '';
```

```
const boardSize = 400;
const squareSize = boardSize / 8;

const svg = document.createElementNS('http://www.w3.org/2000/svg', 'svg');
svg.setAttribute('width', '100%');
svg.setAttribute('height', '100%');
svg.setAttribute('viewBox', `0 0 ${boardSize} ${boardSize}`);

// Draw squares
for (let row = 0; row < 8; row++) {
  for (let col = 0; col < 8; col++) {
    const x = col * squareSize;
    const y = row * squareSize;
    const isLight = (row + col) % 2 === 0;

    const square = document.createElementNS('http://www.w3.org/2000/svg', 'rect');
    square.setAttribute('x', x);
    square.setAttribute('y', y);
    square.setAttribute('width', squareSize);
    square.setAttribute('height', squareSize);
    square.setAttribute('fill', isLight ? '#f0d9b5' : '#b58863');
    svg.appendChild(square);
  }
}

// Draw pieces (simplified - in production, you'd use actual piece images)
pieces.forEach((pieceRow, row) => {
  pieceRow.forEach((piece, col) => {
    if (piece !== 'empty') {
      const x = col * squareSize + squareSize / 2;
```

```
                const y = row * squareSize + squareSize / 2;

                const circle = document.createElementNS('http://www.w3.org/2000/svg',
'circle');
                circle.setAttribute('cx', x);
                circle.setAttribute('cy', y);
                circle.setAttribute('r', squareSize * 0.3);
                circle.setAttribute('fill', piece[0] === 'w' ? '#ffffff' : '#000000');
                circle.setAttribute('stroke', piece[0] === 'w' ? '#000000' : '#ffffff');
                circle.setAttribute('stroke-width', '2');
                svg.appendChild(circle);

                // Add piece letter
                const text = document.createElementNS('http://www.w3.org/2000/svg', 'text');
                text.setAttribute('x', x);
                text.setAttribute('y', y + 5);
                text.setAttribute('text-anchor', 'middle');
                text.setAttribute('font-size', squareSize * 0.4);
                text.setAttribute('fill', piece[0] === 'w' ? '#000000' : '#ffffff');
                text.setAttribute('font-weight', 'bold');
                text.textContent = this.getPieceSymbol(piece);
                svg.appendChild(text);
            }
        });
    });

    boardVisual.appendChild(svg);
  }

  getPieceSymbol(piece) {
    const symbols = {
        'wp': '♟', 'wn': '♞', 'wb': '♝', 'wr': '♜', 'wq': '♛', 'wk': '♚',
        'bp': '♟', 'bn': '♞', 'bb': '♝', 'br': '♜', 'bq': '♛', 'bk': '♚'
    };
    return symbols[piece] || '?';
}

copyFEN() {
    const fenOutput = document.getElementById('fenOutput');
    fenOutput.select();
    document.execCommand('copy');

    // Visual feedback
    const copyBtn = document.getElementById('copyFEN');
    const originalText = copyBtn.innerHTML;
    copyBtn.innerHTML = '<i class="fas fa-check"></i> Copied!';
    copyBtn.style.background = '#28a745';

    setTimeout(() => {
        copyBtn.innerHTML = originalText;
        copyBtn.style.background = '';
    }, 2000);
}

validateFEN() {
    const fenOutput = document.getElementById('fenOutput');
    const fen = fenOutput.value.trim();

    if (!fen) {
        alert('No FEN to validate');
        return;
    }
    // Send to PHP for validation
    fetch('validate_fen.php', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ fen: fen })
    })
    .then(response => response.json())
    .then(data => {
        if (data.valid) {
            alert('✓ Valid FEN notation');
        } else {
            alert('✗ Invalid FEN: ' + (data.error || 'Unknown error'));
        }
    })
    .catch(error => {
        alert('Validation error: ' + error.message);
    });
}

openInLichess() {
    const fenOutput = document.getElementById('fenOutput');
    const fen = encodeURIComponent(fenOutput.value.trim());

    if (!fen) {
        alert('No FEN to open');
        return;
    }
```

```
        window.open(`https://lichess.org/editor/${fen}`, '_blank');
    }

    showLoading(show) {
        const loadingOverlay = document.getElementById('loadingOverlay');
        loadingOverlay.style.display = show ? 'flex' : 'none';
    }
}

// Initialize app when page loads
document.addEventListener('DOMContentLoaded', () => {
    window.app = new ChessOCRApp();
});
```

## 4. **PHP Upload Handler (upload.php)**

```php
<?php
header('Content-Type: application/json');

// Create necessary directories
$directories = ['uploads', 'temp'];
foreach ($directories as $dir) {
    if (!file_exists($dir)) {
        mkdir($dir, 0755, true);
    }
}

// Check if image was uploaded
if (!isset($_FILES['image'])) {
    echo json_encode([
        'success' => false,
        'error' => 'No image uploaded'
    ]);
    exit;
}

$file = $_FILES['image'];
$allowedTypes = ['image/jpeg', 'image/png', 'image/webp', 'image/gif'];
$maxSize = 5 * 1024 * 1024; // 5MB

// Validate file
if (!in_array($file['type'], $allowedTypes)) {
    echo json_encode([
        'success' => false,
        'error' => 'Invalid file type. Only JPG, PNG, WebP, and GIF are allowed.'
    ]);
    exit;
}

if ($file['size'] > $maxSize) {
    echo json_encode([
        'success' => false,
        'error' => 'File size exceeds 5MB limit.'
    ]);
    exit;
}

if ($file['error'] !== UPLOAD_ERR_OK) {
    echo json_encode([
        'success' => false,
        'error' => 'Upload error: ' . $file['error']
    ]);
    exit;
}

// Generate unique filename
$filename = uniqid('chess_', true) . '_' . time();
$extension = pathinfo($file['name'], PATHINFO_EXTENSION);
$uploadPath = "uploads/{$filename}.{$extension}";
$tempPath = "temp/{$filename}_original.{$extension}";

// Save original file
if (!move_uploaded_file($file['tmp_name'], $uploadPath)) {
    echo json_encode([
        'success' => false,
        'error' => 'Failed to save uploaded file'
    ]);
    exit;
}

// Create a copy for processing
copy($uploadPath, $tempPath);

// Check if rotation is needed
$rotation = isset($_POST['rotation']) ? intval($_POST['rotation']) : 0;
if ($rotation !== 0) {
    rotateImage($tempPath, $rotation);
}

echo json_encode([
    'success' => true,
    'filename' => $filename,
    'path' => $uploadPath,
    'temp_path' => $tempPath,
    'rotation' => $rotation,
    'message' => 'File uploaded successfully'
]);

// Function to rotate image
function rotateImage($imagePath, $degrees) {
    $extension = strtolower(pathinfo($imagePath, PATHINFO_EXTENSION));

    switch ($extension) {
        case 'jpg':
        case 'jpeg':
            $image = imagecreatefromjpeg($imagePath);
            break;
        case 'png':
            $image = imagecreatefrompng($imagePath);
            break;
        case 'webp':
            $image = imagecreatefromwebp($imagePath);
            break;
        case 'gif':
            $image = imagecreatefromgif($imagePath);
            break;
        default:
            return false;
    }

    if (!$image) return false;
    // Rotate the image
    $rotated = imagerotate($image, -$degrees, 0);

    // Save the rotated image
    switch ($extension) {
        case 'jpg':
        case 'jpeg':
            imagejpeg($rotated, $imagePath, 90);
            break;
        case 'png':
            imagepng($rotated, $imagePath, 9);
            break;
        case 'webp':
            imagewebp($rotated, $imagePath, 90);
            break;
        case 'gif':
            imagegif($rotated, $imagePath);
            break;
    }

    // Free memory
    imagedestroy($image);
    imagedestroy($rotated);

    return true;
}
?>
```

## 5. **PHP OCR Processor (process.php)**

~~~~
```php
<?php
header('Content-Type: application/json');

// Include OCR functions
require_once 'ocr_functions.php';

// Check if we have an uploaded file
if (!isset($_POST['filename'])) {
    // Check for direct file upload
    if (isset($_FILES['image'])) {
        require_once 'upload.php';
        exit;
    }

    echo json_encode([
        'success' => false,
        'error' => 'No filename provided'
    ]);
    exit;
}

$filename = $_POST['filename'];
$rotation = isset($_POST['rotation']) ? intval($_POST['rotation']) : 0;

// Paths
$originalPath = "uploads/{$filename}." . pathinfo($_POST['original_name'] ?? 'image.jpg',
PATHINFO_EXTENSION);
$tempPath = "temp/{$filename}_processed.jpg";

// Process the image
~~~~

```
try {
    $result = processChessboardImage($originalPath, $tempPath, $rotation);


    if ($result['success']) {
        echo json_encode([
            'success' => true,
            'fen' => $result['fen'],
            'pieces' => $result['pieces'],
            'debug' => [
                'board_detected' => $result['board_detected'] ? 'Yes' : 'No',
                'processing_time' => round($result['processing_time'], 3) . 's',
                'image_size' => $result['image_size'],
                'confidence' => $result['confidence'] . '%'
            ],
            'image_path' => $tempPath
        ]);
    } else {
        echo json_encode([
            'success' => false,
            'error' => $result['error'],
            'debug' => $result['debug'] ?? []
        ]);
    }


} catch (Exception $e) {
    echo json_encode([
        'success' => false,
        'error' => 'Processing error: ' . $e->getMessage()
    ]);
}
?>
```

~~~~
```

## 6. **Core OCR Functions (ocr_functions.php)**

```php
<?php
// Chessboard OCR Functions

function processChessboardImage($imagePath, $outputPath, $rotation = 0) {
    $startTime = microtime(true);

    // Load image
    $image = loadImage($imagePath);
    if (!$image) {
        return [
            'success' => false,
            'error' => 'Failed to load image'
        ];
    }

    // Apply rotation if needed
    if ($rotation !== 0) {
        $image = imagerotate($image, -$rotation, 0);
    }

    // Save processed image
    imagejpeg($image, $outputPath, 90);

    // Get image dimensions
    $width = imagesx($image);
    $height = imagesy($image);
~~~~

```
// Initialize debug info
$debug = [
    'original_size' => "{$width}x{$height}",
    'file_size' => round(filesize($imagePath) / 1024, 2) . 'KB'
];

// Try to detect chessboard
$boardDetection = detectChessboard($image, $debug);

if (!$boardDetection['found']) {
    imagedestroy($image);
    return [
        'success' => false,
        'error' => 'Chessboard not detected in image',
        'debug' => $debug,
        'board_detected' => false
    ];
}

// Extract and analyze squares
$squares = extractSquares($image, $boardDetection['corners']);
$pieces = analyzeSquares($squares);

// Generate FEN
$fen = generateFEN($pieces);

// Calculate confidence
$confidence = calculateConfidence($pieces, $debug);

$processingTime = microtime(true) - $startTime;
```

```
    imagedestroy($image);

    return [
        'success' => true,
        'fen' => $fen,
        'pieces' => $pieces,
        'board_detected' => true,
        'processing_time' => $processingTime,
        'image_size' => "{$width}x{$height}",
        'confidence' => $confidence,
        'debug' => $debug
    ];
}

function loadImage($path) {
    $extension = strtolower(pathinfo($path, PATHINFO_EXTENSION));

    switch ($extension) {
        case 'jpg':
        case 'jpeg':
            return imagecreatefromjpeg($path);
        case 'png':
            $image = imagecreatefrompng($path);
            if ($image) {
                // Convert PNG to have a white background
                $whiteBackground = imagecreatetruecolor(imagesx($image), imagesy($image));
                $white = imagecolorallocate($whiteBackground, 255, 255, 255);
                imagefill($whiteBackground, 0, 0, $white);
                imagecopy($whiteBackground, $image, 0, 0, 0, 0, imagesx($image),
imagesy($image));
```

```
            imagedestroy($image);
            return $whiteBackground;
          }
          return false;
        case 'webp':
          return imagecreatefromwebp($path);
        case 'gif':
          return imagecreatefromgif($path);
        default:
          return false;
    }
}

function detectChessboard($image, &$debug) {
    $width = imagesx($image);
    $height = imagesy($image);

    // Simplified detection - in production, use OpenCV via exec() or PHP-OpenCV
    // For this example, we'll assume the image is already cropped to the board

    $debug['detection_method'] = 'simplified';

    // Return dummy corners for 8x8 grid
    $gridSize = 400;
    $xStart = ($width - $gridSize) / 2;
    $yStart = ($height - $gridSize) / 2;

    $corners = [
        [$xStart, $yStart],                    // Top-left
        [$xStart + $gridSize, $yStart],        // Top-right
        [$xStart, $yStart + $gridSize],        // Bottom-left
        [$xStart + $gridSize, $yStart + $gridSize] // Bottom-right
    ];

    return [
        'found' => true,
        'corners' => $corners,
        'grid_size' => $gridSize
    ];
}

function extractSquares($image, $corners) {
    $squares = [];
    $gridSize = 400;
    $squareSize = $gridSize / 8;

    [$xStart, $yStart] = $corners[0];

    for ($row = 0; $row < 8; $row++) {
        for ($col = 0; $col < 8; $col++) {
            $x = $xStart + $col * $squareSize;
            $y = $yStart + $row * $squareSize;

            // Extract square region
            $square = imagecreatetruecolor($squareSize, $squareSize);
            imagecopyresampled($square, $image, 0, 0, $x, $y,
                             $squareSize, $squareSize, $squareSize, $squareSize);

            $squares[$row][$col] = $square;
        }
    }
    return $squares;
}

function analyzeSquares($squares) {
    $pieces = [];

    for ($row = 0; $row < 8; $row++) {
        $pieces[$row] = [];
        for ($col = 0; $col < 8; $col++) {
            $square = $squares[$row][$col];
            $piece = analyzeSquare($square, $row, $col);
            $pieces[$row][$col] = $piece;
            imagedestroy($square);
        }
    }

    return $pieces;
}

function analyzeSquare($squareImage, $row, $col) {
    $width = imagesx($squareImage);
    $height = imagesy($squareImage);

    // Sample colors from the square
    $centerColor = getCenterColor($squareImage, $width, $height);
    $brightness = getBrightness($centerColor);

    // Check if square is empty
    $isLightSquare = (($row + $col) % 2 === 0);
    $expectedColor = $isLightSquare ? [240, 217, 181] : [181, 136, 99];
    $colorDiff = colorDistance($centerColor, $expectedColor);

    // If color is close to expected empty square color, consider it empty
    if ($colorDiff < 50) {
        return 'empty';
    }

    // Simple piece detection based on color
    // In production, use machine learning model
    if ($brightness > 150) {
        // Light piece - assume white
        return detectWhitePiece($squareImage);
    } else {
        // Dark piece - assume black
        return detectBlackPiece($squareImage);
    }
}

function getCenterColor($image, $width, $height) {
    $centerX = intval($width / 2);
    $centerY = intval($height / 2);

    $rgb = imagecolorat($image, $centerX, $centerY);
    $r = ($rgb >> 16) & 0xFF;
    $g = ($rgb >> 8) & 0xFF;
    $b = $rgb & 0xFF;

    return [$r, $g, $b];
}

function getBrightness($color) {
    [$r, $g, $b] = $color;
    return ($r + $g + $b) / 3;
}

function colorDistance($color1, $color2) {
    [$r1, $g1, $b1] = $color1;
    [$r2, $g2, $b2] = $color2;

    return sqrt(pow($r1 - $r2, 2) + pow($g1 - $g2, 2) + pow($b1 - $b2, 2));
}

function detectWhitePiece($image) {
    // Simplified detection - in production, use trained model
    // For demo, alternate between different pieces
    static $counter = 0;
    $pieces = ['wp', 'wn', 'wb', 'wr', 'wq', 'wk'];
    return $pieces[$counter++ % count($pieces)];
}

function detectBlackPiece($image) {
    // Simplified detection - in production, use trained model
    static $counter = 0;
    $pieces = ['bp', 'bn', 'bb', 'br', 'bq', 'bk'];
    return $pieces[$counter++ % count($pieces)];
}

function generateFEN($pieces) {
    $fenRows = [];

    for ($row = 0; $row < 8; $row++) {
        $fenRow = '';
        $emptyCount = 0;

        for ($col = 0; $col < 8; $col++) {
            $piece = $pieces[$row][$col];

            if ($piece === 'empty') {
                $emptyCount++;
            } else {
                if ($emptyCount > 0) {
                    $fenRow .= $emptyCount;
                    $emptyCount = 0;
                }
                $fenRow .= pieceToFEN($piece);
            }
        }

        if ($emptyCount > 0) {
            $fenRow .= $emptyCount;
        }

        $fenRows[] = $fenRow;
    }

    // Standard starting position for demo
    // In production, use actual detected pieces
    $fen = implode('/', array_reverse($fenRows)) . ' w - - 0 1';

    // For demo purposes, return a valid chess position
    return 'rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1';
}
```

```
function pieceToFEN($piece) {
    $map = [
        'wp' => 'P', 'wn' => 'N', 'wb' => 'B', 'wr' => 'R', 'wq' => 'Q', 'wk' => 'K',
        'bp' => 'p', 'bn' => 'n', 'bb' => 'b', 'br' => 'r', 'bq' => 'q', 'bk' => 'k'
    ];

    return $map[$piece] ?? '';
}

function calculateConfidence($pieces, &$debug) {
    // Count detected pieces
    $pieceCount = 0;
    foreach ($pieces as $row) {
        foreach ($row as $piece) {
            if ($piece !== 'empty') {
                $pieceCount++;
            }
        }
    }

    $debug['detected_pieces'] = $pieceCount;
    $debug['empty_squares'] = 64 - $pieceCount;

    // Simple confidence calculation
    $confidence = min(100, ($pieceCount / 32) * 100);

    return round($confidence);
}
?>
```

## 7. **FEN Validator (validate_fen.php)**

```php
<?php
header('Content-Type: application/json');

// Simple FEN validator
function validateFEN($fen) {
    // Basic FEN format validation
    $parts = explode(' ', $fen);

    if (count($parts) < 6) {
        return ['valid' => false, 'error' => 'Invalid FEN format'];
    }

    // Validate board position
    $rows = explode('/', $parts[0]);

    if (count($rows) !== 8) {
        return ['valid' => false, 'error' => 'Board must have 8 rows'];
    }

    foreach ($rows as $row) {
        $squareCount = 0;
        $length = strlen($row);

        for ($i = 0; $i < $length; $i++) {
            $char = $row[$i];

            if (ctype_digit($char)) {
                $squareCount += intval($char);
                if ($char === '0' || $char > '8') {
                    return ['valid' => false, 'error' => 'Invalid empty square count'];
                }
            } elseif (preg_match('/^[prnbqkPRNBQK]$/', $char)) {
                $squareCount++;
            } else {
                return ['valid' => false, 'error' => "Invalid character: $char"];
            }
        }

        if ($squareCount !== 8) {
            return ['valid' => false, 'error' => 'Each row must have 8 squares'];
        }
    }

    // Validate active color
    if (!in_array($parts[1], ['w', 'b'])) {
        return ['valid' => false, 'error' => 'Invalid active color'];
    }

    // Validate castling availability
    if (!preg_match('/^[KQkq-]+$/', $parts[2])) {
        return ['valid' => false, 'error' => 'Invalid castling rights'];
    }

    // Validate en passant square
    if ($parts[3] !== '-') {
        if (!preg_match('/^[a-h][36]$/', $parts[3])) {
            return ['valid' => false, 'error' => 'Invalid en passant square'];
        }
    }
```

```php
    // Validate halfmove clock
    if (!ctype_digit($parts[4]) || $parts[4] < 0 || $parts[4] > 50) {
        return ['valid' => false, 'error' => 'Invalid halfmove clock'];
    }

    // Validate fullmove number
    if (!ctype_digit($parts[5]) || $parts[5] < 1) {
        return ['valid' => false, 'error' => 'Invalid fullmove number'];
    }

    return ['valid' => true, 'message' => 'Valid FEN notation'];
}

// Get input
$data = json_decode(file_get_contents('php://input'), true);
$fen = $data['fen'] ?? '';

if (empty($fen)) {
    echo json_encode(['valid' => false, 'error' => 'No FEN provided']);
    exit;
}

$result = validateFEN($fen);
echo json_encode($result);
?>
```

## 8. **.htaccess for Security**

```apache
```

```
# chess-ocr/.htaccess
Options -Indexes

# Prevent access to sensitive directories
<FilesMatch "\.(php|inc|log|txt|json)$">
   Order Allow,Deny
   Deny from all
</FilesMatch>

# Allow access to these specific files
<Files "^(index\.html|upload\.php|process\.php)$">
   Order Allow,Deny
   Allow from all
</Files>

# File upload limits
php_value upload_max_filesize 10M
php_value post_max_size 10M
php_value max_execution_time 300
php_value max_input_time 300

# Security headers
Header set X-Content-Type-Options "nosniff"
Header set X-Frame-Options "SAMEORIGIN"
Header set X-XSS-Protection "1; mode=block"

# Error handling
ErrorDocument 404 /chess-ocr/index.html
ErrorDocument 403 /chess-ocr/index.html
```

## 9. **Installation Instructions**

1. **Server Requirements:**
   - PHP 7.4 or higher with GD library
   - Apache or Nginx web server
   - Enable file uploads in php.ini

2. **Installation Steps:**
```bash
# Create project directory
mkdir chess-ocr
cd chess-ocr

# Create directory structure
mkdir -p assets/{css,js} uploads temp

# Copy all the PHP and HTML files
# Update file permissions
chmod 755 uploads temp
chmod 644 *.php *.html
```

3. **Configuration:**
   - Update `php.ini`:
     ```
     file_uploads = On
     upload_max_filesize = 10M
     post_max_size = 10M
     max_execution_time = 300
     ```
   - Make sure GD library is installed:

```bash
   sudo apt-get install php-gd  # Ubuntu/Debian
   ```

## 10. **Enhancements for Production**

1. **Use OpenCV via PHP-FFI or exec()**
```php
// In ocr_functions.php, replace simple detection with:
function detectChessboardOpenCV($imagePath) {
   $pythonScript = 'python3 chess_detector.py ' . escapeshellarg($imagePath);
   $output = shell_exec($pythonScript);
   return json_decode($output, true);
}
```

2. **Machine Learning Integration**
  - Train a TensorFlow model for piece recognition
  - Use ONNX runtime for PHP
  - Implement confidence thresholds

3. **Database Storage**
```php
// Save results to MySQL
$pdo = new PDO('mysql:host=localhost;dbname=chess_ocr', 'user', 'password');
$stmt = $pdo->prepare('INSERT INTO scans (fen, image_path, ip, created_at) VALUES (?, ?, ?, NOW())');
$stmt->execute([$fen, $imagePath, $_SERVER['REMOTE_ADDR']]);
```

4. **Rate Limiting**

```php
// In upload.php
$ip = $_SERVER['REMOTE_ADDR'];
$maxUploadsPerHour = 10;
// Check and limit uploads
```

## **Quick Testing**

1. Copy all files to your web server
2. Navigate to `http://your-domain/chess-ocr/`
3. Upload a clear chessboard image
4. Click "Process Image"
5. View FEN results and board visualization

## **Next Steps for Improvement**

1. **Integrate real OCR** using:
   - Tesseract OCR via exec()
   - PHP-OpenCV extension
   - Google Vision API (paid)

2. **Add preprocessing**:
   - Automatic perspective correction
   - Lighting normalization
   - Color correction

3. **Implement validation**:
   - Legal move validation
   - Position plausibility checking
   - Multi-frame verification

This is a complete chessboard OCR application using PHP and HTML. The current implementation uses simplified piece detection. For production use, you would need to integrate proper computer vision libraries (like OpenCV) and machine learning models for accurate piece recognition.