Statement Analyzer — Multi-Format File Handler

The Problem

The statement analyzer needs to handle THREE different input types:

File Type	Examples	Processing Method
PDF	.pdf statements	Text extraction OR Claude native
Images	.jpg, .png, .jpeg, .webp (photos of statements)	Vision API (requires base64)
Spreadsheets	.xlsx, .xls, .csv	Direct parsing (no AI needed)

The current error (`Invalid image URL`) happens because PDFs are being sent to a vision API incorrectly. But even with that fixed, images and Excel files need different handling.

Solution: Smart File Router

```
const fs = require('fs');
const path = require('path');
const pdfParse = require('pdf-parse');
const XLSX = require('xlsx');
const Anthropic = require('@anthropic-ai/sdk');
const OpenAI = require('openai');

const anthropic = new Anthropic();
const openai = new OpenAI();

/**
 * Main entry point - routes to appropriate handler based on file type
 */
async function analyzeStatement(file) {
  const filename = file.originalname || file.name;
  const ext = path.extname(filename).toLowerCase();
  const buffer = file.buffer || fs.readFileSync(file.path);
```

```
  console.log(`Processing file: ${filename} (${ext}, ${buffer.length} bytes)`);

  // Route to appropriate handler
  switch (ext) {
    case '.pdf':
      return await analyzePDF(buffer, filename);

    case '.jpg':
    case '.jpeg':
    case '.png':
    case '.webp':
      return await analyzeImage(buffer, ext);

    case '.xlsx':
    case '.xls':
      return await analyzeExcel(buffer, filename);

    case '.csv':
      return await analyzeCSV(buffer);

    default:
      throw new Error(`Unsupported file type: ${ext}. Please upload PDF, image (J
  }
}

module.exports = { analyzeStatement };
```

Handler 1: PDF Files (Text Extraction)

PDFs contain searchable text — extract it and send to a regular LLM (no vision needed).

```
/**
 * Handle PDF statements - extract text and analyze
 */
async function analyzePDF(buffer, filename) {
  console.log('Processing PDF...');

  try {
    // Step 1: Extract text from PDF
    const pdfData = await pdfParse(buffer);
    const text = pdfData.text;

    console.log(`Extracted ${text.length} characters from PDF`);
```

```
    if (text.length < 100) {
      // PDF might be scanned/image-based - fall back to vision
      console.log('PDF appears to be image-based, using vision...');
      return await analyzePDFWithVision(buffer);
    }

    // Step 2: Send to Claude for analysis
    const response = await anthropic.messages.create({
      model: "claude-sonnet-4-20250514",
      max_tokens: 4096,
      messages: [
        {
          role: "user",
          content: `Analyze this merchant processing statement and extract all ke

STATEMENT TEXT:
${text}

Return ONLY valid JSON with this structure:
{
  "merchant": {
    "name": "",
    "address": "",
    "merchant_number": ""
  },
  "statement_period": {
    "start": "",
    "end": ""
  },
  "processor": "",
  "summary": {
    "total_volume": 0,
    "total_transactions": 0,
    "total_fees": 0,
    "effective_rate": 0,
    "amount_funded": 0
  },
  "card_breakdown": [
    {
      "card_type": "",
      "volume": 0,
      "transactions": 0,
      "average_ticket": 0
    }
  ],
  "fee_breakdown": [
```

```
    {
      "category": "",
      "description": "",
      "amount": 0
    }
  ],
  "monthly_fees": {
    "interchange": 0,
    "markup": 0,
    "dues_assessments": 0,
    "per_item_fees": 0,
    "monthly_fees": 0,
    "other_fees": 0
  }
}`
        }
      ]
    });

    // Parse response
    const content = response.content[0].text;
    const jsonMatch = content.match(/\{[\s\S]*\}/);

    if (jsonMatch) {
      return {
        success: true,
        method: 'pdf_text_extraction',
        data: JSON.parse(jsonMatch[0])
      };
    }

    throw new Error('Could not parse response');

  } catch (error) {
    console.error('PDF analysis error:', error);
    throw error;
  }
}

/**
 * Fallback for scanned/image-based PDFs - use Claude's native PDF support
 */
async function analyzePDFWithVision(buffer) {
  console.log('Using Claude vision for image-based PDF...');

  const base64 = buffer.toString('base64');
```

```
  const response = await anthropic.messages.create({
    model: "claude-sonnet-4-20250514",
    max_tokens: 4096,
    messages: [
      {
        role: "user",
        content: [
          {
            type: "document",
            source: {
              type: "base64",
              media_type: "application/pdf",
              data: base64
            }
          },
          {
            type: "text",
            text: `This is a merchant processing statement. Analyze it and extrac

Return ONLY valid JSON with:
- Merchant name and address
- Statement period
- Total volume, transactions, fees
- Effective rate (fees / volume * 100)
- Breakdown by card type
- Detailed fee breakdown`
          }
        ]
      }
    ]
  });

  const content = response.content[0].text;
  const jsonMatch = content.match(/\{[\s\S]*\}/);

  return {
    success: true,
    method: 'pdf_vision',
    data: jsonMatch ? JSON.parse(jsonMatch[0]) : null
  };
}
```

Handler 2: Images (Vision API Required)

Photos of statements MUST use vision — this is where base64 encoding is critical.

```
/**
 * Handle image files - REQUIRES vision API with proper base64 encoding
 */
async function analyzeImage(buffer, ext) {
  console.log('Processing image with vision API...');

  // Determine MIME type
  const mimeTypes = {
    '.jpg': 'image/jpeg',
    '.jpeg': 'image/jpeg',
    '.png': 'image/png',
    '.webp': 'image/webp'
  };
  const mimeType = mimeTypes[ext] || 'image/jpeg';

  // Convert to base64 data URL - THIS IS THE CRITICAL PART
  const base64 = buffer.toString('base64');
  const dataUrl = `data:${mimeType};base64,${base64}`;

  console.log(`Image converted to base64 (${base64.length} chars)`);

  // Option A: Use Claude Vision
  try {
    const response = await anthropic.messages.create({
      model: "claude-sonnet-4-20250514",
      max_tokens: 4096,
      messages: [
        {
          role: "user",
          content: [
            {
              type: "image",
              source: {
                type: "base64",
                media_type: mimeType,
                data: base64  // Note: Claude wants raw base64, not data URL
              }
            },
            {
              type: "text",
              text: `This is a photo of a merchant processing statement. Analyze

Return ONLY valid JSON with:
{
```

```
  "merchant": { "name": "", "address": "" },
  "statement_period": "",
  "summary": {
    "total_volume": 0,
    "total_transactions": 0,
    "total_fees": 0,
    "effective_rate": 0
  },
  "card_breakdown": [...],
  "fee_breakdown": [...],
  "notes": "any text that was hard to read or unclear"
}`
            }
          ]
        }
      ]
    });

    const content = response.content[0].text;
    const jsonMatch = content.match(/\{[\s\S]*\}/);

    return {
      success: true,
      method: 'image_vision_claude',
      data: jsonMatch ? JSON.parse(jsonMatch[0]) : null
    };

  } catch (claudeError) {
    console.log('Claude vision failed, trying OpenAI...', claudeError.message);

    // Option B: Fall back to OpenAI Vision
    const response = await openai.chat.completions.create({
      model: "gpt-4o",
      messages: [
        {
          role: "user",
          content: [
            {
              type: "text",
              text: `Analyze this merchant processing statement image and extract
            },
            {
              type: "image_url",
              image_url: {
                url: dataUrl  // OpenAI wants the full data URL
              }
            }
```

```
          ]
        }
      ],
      max_tokens: 4096
    });

    const content = response.choices[0].message.content;
    const jsonMatch = content.match(/\{[\s\S]*\}/);

    return {
      success: true,
      method: 'image_vision_openai',
      data: jsonMatch ? JSON.parse(jsonMatch[0]) : null
    };
  }
}
```

Handler 3: Excel Files (Direct Parsing — No AI Needed)

Excel files have structured data — parse directly without AI.

```
/**
 * Handle Excel files - parse directly, no AI needed for structure
 */
async function analyzeExcel(buffer, filename) {
  console.log('Processing Excel file...');

  // Parse Excel
  const workbook = XLSX.read(buffer, { type: 'buffer' });

  // Get all sheet names
  const sheetNames = workbook.SheetNames;
  console.log('Sheets found:', sheetNames);

  // Convert all sheets to JSON
  const sheets = {};
  for (const name of sheetNames) {
    sheets[name] = XLSX.utils.sheet_to_json(workbook.Sheets[name], { header: 1 })
  }

  // Try to auto-detect statement structure
  const data = detectStatementStructure(sheets);

  if (data) {
```

```
    return {
      success: true,
      method: 'excel_direct_parse',
      data: data
    };
  }

  // If structure not recognized, use AI to interpret
  console.log('Excel structure not recognized, using AI...');

  const sheetText = Object.entries(sheets)
    .map(([name, rows]) => `Sheet: ${name}\n${rows.map(r => r.join('\t')).join('\
    .join('\n\n');

  const response = await anthropic.messages.create({
    model: "claude-sonnet-4-20250514",
    max_tokens: 4096,
    messages: [
      {
        role: "user",
        content: `This is data from a merchant processing statement Excel file. E

${sheetText}

Return JSON with: merchant info, total volume, transactions, fees, effective rate
      }
    ]
  });

  const content = response.content[0].text;
  const jsonMatch = content.match(/\{[\s\S]*\}/);

  return {
    success: true,
    method: 'excel_ai_interpreted',
    data: jsonMatch ? JSON.parse(jsonMatch[0]) : null
  };
}

/**
 * Try to detect common statement Excel structures
 */
function detectStatementStructure(sheets) {
  // Look for common column headers
  const firstSheet = Object.values(sheets)[0];

  if (!firstSheet || firstSheet.length < 2) return null;
```

```
  // Check for typical statement headers
  const headerRow = firstSheet[0] || [];
  const headerStr = headerRow.join(' ').toLowerCase();

  // PCBancard pricing comparison format
  if (headerStr.includes('volume') && headerStr.includes('rate') && headerStr.inc
    return parsePCBancardFormat(firstSheet);
  }

  // Generic statement format with card types
  if (headerStr.includes('visa') || headerStr.includes('mastercard') || headerStr
    return parseGenericStatementFormat(firstSheet);
  }

  return null;
}

/**
 * Parse PCBancard pricing comparison Excel format
 */
function parsePCBancardFormat(rows) {
  const data = {
    current: {},
    proposed: {},
    savings: {}
  };

  for (const row of rows) {
    const label = String(row[0] || '').toLowerCase();

    if (label.includes('total processing fees')) {
      data.current.total_fees = parseFloat(row[1]) || 0;
      data.proposed.total_fees = parseFloat(row[2]) || 0;
    }

    if (label.includes('monthly processing savings')) {
      data.savings.monthly = parseFloat(row[1]) || 0;
    }

    if (label.includes('yearly processing savings')) {
      data.savings.annual = parseFloat(row[1]) || 0;
    }

    // Parse card volumes
    if (label.includes('visa') && label.includes('volume')) {
      data.current.visa_volume = parseFloat(row[1]) || 0;
```

```
    }
    // ... add more parsing rules
  }

  return data;
}

/**
 * Parse generic statement format
 */
function parseGenericStatementFormat(rows) {
  // Implementation depends on specific format
  // Add detection logic for common formats
  return null;
}
```

Handler 4: CSV Files

```
/**
 * Handle CSV files
 */
async function analyzeCSV(buffer) {
  console.log('Processing CSV file...');

  const text = buffer.toString('utf-8');

  // Parse CSV
  const rows = text.split('\n').map(row => row.split(',').map(cell => cell.trim()

  // Try to detect structure
  const data = detectStatementStructure({ 'CSV': rows });

  if (data) {
    return {
      success: true,
      method: 'csv_direct_parse',
      data: data
    };
  }

  // Fall back to AI interpretation
  const response = await anthropic.messages.create({
    model: "claude-sonnet-4-20250514",
    max_tokens: 4096,
```

```
    messages: [
      {
        role: "user",
        content: `This is CSV data from a merchant processing statement. Extract

${text.substring(0, 10000)}

Return JSON with merchant info, volumes, fees, and breakdowns.`
      }
    ]
  });

  const content = response.content[0].text;
  const jsonMatch = content.match(/\{[\s\S]*\}/);

  return {
    success: true,
    method: 'csv_ai_interpreted',
    data: jsonMatch ? JSON.parse(jsonMatch[0]) : null
  };
}
```

Installation

```
npm install pdf-parse xlsx @anthropic-ai/sdk openai
```

Summary: When to Use What

File Type	Processing Method	AI Model	Base64 Needed?
PDF (text-based)	Extract text → LLM	Claude/GPT-4 (text)	No
PDF (scanned)	Claude native PDF	Claude Vision	Yes
JPG/PNG/WEBP	Vision API	Claude or GPT-4 Vision	**Yes (critical!)**
XLSX/XLS	Direct parse with xlsx	Only if structure unknown	No

CSV	Direct parse	Only if structure unknown	No

The Critical Base64 Fix

For **images only**, this is the fix for the original error:

```
// ❌ WRONG - These will ALL fail:
image_url: { url: file.path }                    // Local path
image_url: { url: `/uploads/${filename}` }       // Relative path
image_url: { url: URL.createObjectURL(file) }    // Blob URL (browser only)

// ✅ CORRECT - Base64 data URL:
const buffer = fs.readFileSync(file.path);
const base64 = buffer.toString('base64');

// For OpenAI:
image_url: { url: `${base64}` }

// For Claude:
source: { type: "base64", media_type: "image/png", data: base64 }
```

Error Handling

```
async function analyzeStatement(file) {
  try {
    // ... routing logic ...
  } catch (error) {
    console.error('Statement analysis failed:', error);

    // Return helpful error message
    if (error.message.includes('Invalid image URL')) {
      return {
        success: false,
        error: 'Image processing error. The file could not be converted for analy
        suggestion: 'Try uploading a PDF version of the statement instead.'
      };
    }

    if (error.message.includes('Unsupported file type')) {
      return {
```

```
          success: false,
          error: error.message,
          supported: ['PDF', 'JPG', 'PNG', 'XLSX', 'CSV']
        };
      }

      return {
        success: false,
        error: 'Analysis failed. Please try again or contact support.',
        details: error.message
      };
    }
  }
```

Testing

After implementing, test with:

1. **PDF statement** (like Brickworks Dental) → Should use text extraction
2. **Photo of statement** (JPG/PNG) → Should use vision with base64
3. **Excel export** → Should parse directly
4. **CSV file** → Should parse directly
5. **Scanned PDF** (image-based) → Should detect and use vision

Each should return structured JSON with the extracted data.