URGENT FIX: Statement Analyzer — “Invalid image URL” Error

The Error

```
500: {"error":"AI analysis failed: 400 Invalid image URL. The URL must be a valid
```

What’s Wrong

The code is trying to send a PDF to an AI vision API (OpenAI GPT-4 Vision or similar) but:

1. It’s passing a file path or blob URL instead of a valid image URL
2. PDFs need to be converted to images first (vision APIs don’t accept PDFs directly)
3. The image data needs to be base64-encoded as a data URL

The Fix

Option A: Convert PDF to Images, Then Base64 (Recommended for Vision API)

```
const pdf = require('pdf-poppler');  // or pdf2pic
const fs = require('fs');
const path = require('path');

async function analyzeStatementWithVision(pdfBuffer, filename) {
  // Step 1: Save PDF temporarily
  const tempPdfPath = `/tmp/${filename}`;
  fs.writeFileSync(tempPdfPath, pdfBuffer);

  // Step 2: Convert PDF pages to images
  const outputDir = `/tmp/${filename}_images`;
  fs.mkdirSync(outputDir, { recursive: true });

  const opts = {
    format: 'png',
    out_dir: outputDir,
  out_prefix: 'page',
  page: null  // Convert all pages
};

await pdf.convert(tempPdfPath, opts);

// Step 3: Read images and convert to base64
const imageFiles = fs.readdirSync(outputDir)
  .filter(f => f.endsWith('.png'))
  .sort();

const base64Images = imageFiles.map(file => {
  const imagePath = path.join(outputDir, file);
  const imageBuffer = fs.readFileSync(imagePath);
  const base64 = imageBuffer.toString('base64');
  return `data:image/png;base64,${base64}`;
});

// Step 4: Send to OpenAI Vision API
const response = await openai.chat.completions.create({
  model: "gpt-4-vision-preview",
  messages: [
    {
      role: "user",
      content: [
        {
          type: "text",
          text: `Analyze this merchant processing statement and extract:
            - Business name
            - Statement period
            - Total volume processed
            - Total transactions
            - Total fees charged
            - Effective rate (fees / volume)
            - Breakdown by card type (Visa, MC, Amex, Discover)
            - List of individual fees
            Return as JSON.`
        },
        // Add each page as an image
        ...base64Images.map(img => ({
          type: "image_url",
          image_url: { url: img }
        }))
      ]
    }
  ],
  max_tokens: 4096
  });

  // Step 5: Cleanup temp files
  fs.rmSync(outputDir, { recursive: true });
  fs.unlinkSync(tempPdfPath);

  return JSON.parse(response.choices[0].message.content);
}
```

Option B: Extract PDF Text First, Then Analyze (Simpler, No Vision Needed)

This approach doesn't need vision at all — just extract the text and send to a regular LLM:

```
const pdfParse = require('pdf-parse');

async function analyzeStatementWithText(pdfBuffer) {
  // Step 1: Extract text from PDF
  const pdfData = await pdfParse(pdfBuffer);
  const text = pdfData.text;

  console.log('Extracted PDF text length:', text.length);
  console.log('First 500 chars:', text.substring(0, 500));

  // Step 2: Send text to OpenAI (no vision needed)
  const response = await openai.chat.completions.create({
    model: "gpt-4-turbo",  // or gpt-4o
    messages: [
      {
        role: "system",
        content: `You are a payment processing statement analyzer. Extract key da
      },
      {
        role: "user",
        content: `Analyze this merchant processing statement and extract the foll

STATEMENT TEXT:
${text}

EXTRACT THIS DATA:
{
  "merchant_name": "",
  "merchant_address": "",
  "statement_period": "",
  "processor": "",
  "total_volume": 0,
  "total_transactions": 0,
  "total_fees": 0,
  "effective_rate": 0,
  "amount_funded": 0,
  "card_breakdown": {
    "visa": { "volume": 0, "transactions": 0 },
    "visa_debit": { "volume": 0, "transactions": 0 },
    "mastercard": { "volume": 0, "transactions": 0 },
    "mastercard_debit": { "volume": 0, "transactions": 0 },
    "amex": { "volume": 0, "transactions": 0 },
    "discover": { "volume": 0, "transactions": 0 }
  },
  "fee_breakdown": {
    "interchange": 0,
    "dues_assessments": 0,
    "processor_fees": 0,
    "other_fees": 0
  },
  "monthly_fees": [
    { "description": "", "amount": 0 }
  ]
}`
      }
    ],
    response_format: { type: "json_object" }
  });

  return JSON.parse(response.choices[0].message.content);
}
```

Option C: Use Claude Instead of OpenAI (Handles PDFs Natively)

Claude can accept PDFs directly as base64:

```
const Anthropic = require('@anthropic-ai/sdk');
const anthropic = new Anthropic();

async function analyzeStatementWithClaude(pdfBuffer) {
  // Convert PDF buffer to base64
  const base64Pdf = pdfBuffer.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: base64Pdf
            }
          },
          {
            type: "text",
            text: `Analyze this merchant processing statement and extract:

1. Business name and address
2. Statement period
3. Processor name
4. Total volume processed
5. Total transactions
6. Total fees charged
7. Effective rate (fees ÷ volume × 100)
8. Amount funded to merchant
9. Breakdown by card type (volume and transactions for each)
10. Detailed fee breakdown

Return as structured JSON only, no other text.`
          }
        ]
      }
    ]
  });

  // Parse the response
  const content = response.content[0].text;

  // Extract JSON from response (Claude might wrap it in markdown)
  const jsonMatch = content.match(/\{[\s\S]*\}/);
  if (jsonMatch) {
    return JSON.parse(jsonMatch[0]);
  }

  throw new Error('Could not parse JSON from Claude response');
}
```

Quick Diagnosis

Add this logging to find exactly where it's failing:

```
async function analyzeStatement(file) {
  console.log('=== STATEMENT ANALYSIS DEBUG ===');
  console.log('File received:', {
    name: file.name || file.originalname,
    size: file.size,
    type: file.type || file.mimetype
  });

  // Get the buffer
  let buffer;
  if (file.buffer) {
    buffer = file.buffer;
  } else if (file.path) {
    buffer = fs.readFileSync(file.path);
  } else {
    console.error('Cannot get file buffer');
    throw new Error('Invalid file input');
  }

  console.log('Buffer size:', buffer.length);

  // Check what's being sent to the AI
  // THIS IS LIKELY WHERE THE BUG IS
  // Look for code that creates the image_url and log it:

  const imageUrl = /* whatever is being generated */;
  console.log('Image URL being sent:', imageUrl?.substring(0, 100));

  // It should start with either:
  // - "https://" (public URL)
  // - "data:image/png;base64," (base64 data URL)
  // - "data:application/pdf;base64," (for Claude)

  // If it starts with "file://", "blob:", or is just a path like "/tmp/...",
  // THAT'S THE BUG
}
```

The Actual Bug (Most Likely)

Find code that looks like this:

```
// WRONG - This won't work
```

```
const imageUrl = `/uploads/${filename}`;  // Local path, not a URL
// or
const imageUrl = file.path;  // File system path
// or
const imageUrl = URL.createObjectURL(file);  // Blob URL (browser only)
```

Replace with:

```
// CORRECT - Base64 data URL
const buffer = fs.readFileSync(file.path);
const base64 = buffer.toString('base64');
const imageUrl = `data:image/png;base64,${base64}`;
```

For Your Specific Statement (Brickworks Dental)

This is what the AI should extract from the uploaded statement:

```
{
  "merchant_name": "Brickworks Dental",
  "merchant_address": "5429 Harding Hwy Ste 101, Mays Landing NJ 08330-2263",
  "statement_period": "12/01/25 - 12/31/25",
  "processor": "CardConnect Direct",
  "merchant_number": "5180897322021480",

  "total_volume": 99682.53,
  "total_transactions": 402,
  "total_fees": 3307.19,
  "effective_rate": 3.32,
  "amount_funded": 96375.34,

  "card_breakdown": {
    "visa": { "volume": 35772.42, "transactions": 110, "avg_ticket": 325.20 },
    "visa_debit": { "volume": 30362.85, "transactions": 155, "avg_ticket": 195.89
    "mastercard": { "volume": 16650.62, "transactions": 68, "avg_ticket": 244.86
    "mastercard_debit": { "volume": 6026.34, "transactions": 31, "avg_ticket": 19
    "amex": { "volume": 6207.76, "transactions": 17, "avg_ticket": 365.16 },
    "discover": { "volume": 4662.54, "transactions": 21, "avg_ticket": 222.03 }
  },

  "fee_breakdown": {
    "total_card_fees": 3132.59,
    "total_misc_fees": 174.60,
    "interchange": 1509.27,
    "discount_fees": 1028.23,
    "item_fees": 261.81,
    "dues_assessments": 128.13,
    "other_card_fees": 205.15,
    "batch_header": 3.75,
    "pci_non_compliance": 99.00,
    "regulatory_product": 14.95,
    "cardpointe_fee": 49.95,
    "data_breach": 6.95
  },

  "ytd_volume": 1351271.37
}
```

Installation

If you need pdf-parse:

```
npm install pdf-parse
```

If you need pdf-poppler (for PDF to image):

```
npm install pdf-poppler
```

Note: pdf-poppler requires system dependencies. In Replit, you may need to use pdf2pic or sharp instead.

Recommended Approach

Use Option B (text extraction) or Option C (Claude with native PDF support).

Vision APIs are overkill for processing statements — the text extraction approach is:

- Faster
- Cheaper (no vision tokens)
- More reliable
- Works with any PDF

If you're already using Claude elsewhere in the app, Option C is cleanest since Claude handles PDFs natively.