Critical Fixes Applied - October 8, 2025

This document describes the three critical fixes applied to resolve PM2 log errors.

Issue 1: Q&A Generation Timeouts

Problem

Multiple documentation files were timing out after 300 seconds during Q&A generation:
- AI_BACKEND_IMPLEMENTATION_COMPLETE.md
- AI_BACKEND_IMPLEMENTATION_COMPLETE.pdf
- AI_BACKEND_SETUP.md
- AI_ASSISTANT_IMPLEMENTATION.md
- AI_ASSISTANT_QUICK_START.md
- AI_ASSISTANT_QUICK_START.pdf

Root Cause

1. Large files (>2MB) were being processed in a single request
2. 300-second timeout was insufficient for complex documents
3. No chunking strategy for large files
4. No retry mechanism for failed requests

Solution Applied

File: `src/lib/services/qa-generator.ts`

1. **Increased Timeout**: Extended from 300s to 600s (10 minutes)
   ```
   typescript
   const QA_GENERATION_TIMEOUT = 600000; // 10 minutes
   ```

2. **Implemented Content Chunking**: Added `chunkContent()` function to split large files into 3000-character chunks
   ```
   typescript
   const CHUNK_SIZE = 3000;
   function chunkContent(content: string, maxChunkSize: number = CHUNK_SIZE): string[]
   ```

3. **Reduced File Size Limit**: Lowered from 5MB to 2MB to prevent timeouts
   ```
   typescript
   const MAX_FILE_SIZE_MB = 2;
   ```

4. **Added Retry Logic**: Implemented up to 2 retries with 2-second delays
   ```
   typescript
   const MAX_RETRIES = 2;
   while (retries <= MAX_RETRIES) { ... }
   ```

5. **Reduced Concurrency**: Lowered from 3 to 2 concurrent files to prevent Ollama overload
   ```
   typescript
   const MAX_CONCURRENT_FILES = 2;
   ```

Expected Outcome

- Large files are now processed in manageable chunks
- Timeouts are less likely with 10-minute limit
- Failed requests are automatically retried
- System load is reduced with lower concurrency

Issue 2: Invalid JSON Responses

Problem

AI was not returning valid JSON for some files:
- “No JSON found in response for AI_DEVICE_ENHANCEMENTS.md”
- “No JSON found in response for AI_ASSISTANT_IMPLEMENTATION.pdf”
- “No JSON found in response for AI_BACKEND_SETUP.md”
- “No JSON found in response for AI_ASSISTANT_IMPLEMENTATION.md”

Root Cause

1. Prompt did not explicitly enforce JSON-only output
2. Model was including markdown code blocks (`json`)
3. Temperature was too high (0.7), causing inconsistent outputs
4. No format enforcement in API request
5. Weak JSON parsing that couldn’t handle variations

Solution Applied

File: `src/lib/services/qa-generator.ts`

1. **Enhanced Prompt with Explicit JSON Instructions**:
   ``typescript
   const prompt = You are a helpful assistant that generates question-answer pairs from documentation.

CRITICAL INSTRUCTIONS:
1. You MUST respond with ONLY valid JSON - no other text before or after
2. Do NOT include markdown code blocks or any formatting
3. Start your response with { and end with }
...
Remember: Output ONLY the JSON object, nothing else.`;
```

1. **Added Format Enforcement in API Call**:
   ```typescript
   body: JSON.stringify({
     model: options.model || DEFAULT_MODEL,
     prompt,
     stream: false,
     format: 'json', // NEW: Enforce JSON format
     options: {
       temperature: 0.3, // OPTIMIZED: Lower for consistency
       top_p: 0.9,

```
      num_predict: 1000,
    },
  })
```

2. **Improved JSON Parsing with Markdown Removal**:
   ```
   typescript
   function parseGeneratedQAs(response: string, sourceFile: string): GeneratedQA[] {
     // Clean the response - remove markdown code blocks if present
     let cleanedResponse = response.trim();
     cleanedResponse = cleanedResponse.replace(/ json\s/g, ‘’).replace(/```\s/g, ‘’);

   // Try to find JSON object in the response
   const jsonMatch = cleanedResponse.match(/{[\s\S]*}/);
   …
   }
   ```

3. **Enhanced Error Logging**:
   ```
   typescript
   console.warn(`Response preview: ${response.substring(0, 200)}...`);
   console.error(`Attempted to parse: ${jsonMatch[0].substring(0, 200)}...`);
   ```

4. **Lowered Temperature**: Reduced from 0.7 to 0.3 for more deterministic outputs

Expected Outcome

- Model will consistently output valid JSON
- Markdown code blocks are automatically stripped
- Better error messages for debugging
- More reliable Q&A generation

Issue 3: Invalid Image Errors

Problem

Multiple layout images showing as invalid:
- “The requested resource isn’t a valid image for /uploads/layouts/136fc844-932c-4303-9691-5c190a1213bb.png received null”
- Similar errors for other layout UUIDs

Root Cause

1. No validation that uploaded files are actually valid images
2. PDF conversion could produce corrupted images
3. No MIME type verification after processing
4. Files could be saved but not be valid image formats
5. Next.js Image component was trying to optimize invalid images

Solution Applied

File: `src/app/api/bartender/upload-layout/route.ts`

1. **Added Image Validation Function**:
```
typescript
  async function validateImage(filepath: string): Promise<boolean> {
    try {
      const metadata = await sharp(filepath).metadata();
      // Check if we got valid image metadata
      return !!(metadata.width && metadata.height && metadata.format);
    } catch (error) {
      console.error('Image validation failed:', error);
      return false;
    }
  }
```

2. **Validate Direct Image Uploads**:
```
typescript
  // For direct image uploads, validate the image
  const isValidImage = await validateImage(filepath);
  if (!isValidImage) {
    console.error('Uploaded image validation failed');
    await fs.unlink(filepath).catch(err => console.log('Cleanup error:', err));
    return NextResponse.json(
      { error: 'Invalid image file. The file may be corrupted or not a valid image
format.' },
      { status: 400 }
    )
  }
```

3. **Validate PDF Conversions**:
```
typescript
  // Validate the converted image
  const isValidImage = await validateImage(optimizedPath);
  if (!isValidImage) {
    console.error('Converted image validation failed');
    await fs.unlink(optimizedPath).catch(err => console.log('Cleanup error:', err));
    throw new Error('PDF conversion produced invalid image');
  }
```

4. **Enhanced Error Messages**:
```
typescript
  return NextResponse.json(
    { error: error instanceof Error ? error.message : 'Failed to upload file' },
    { status: 500 }
  )
```

Expected Outcome

- Only valid images are saved to the filesystem
- Corrupted uploads are rejected with clear error messages

- PDF conversions are validated before being served
- Invalid files are automatically cleaned up
- No more “invalid image” errors in Next.js

Testing Recommendations

For Q&A Generation:

1. Test with large documentation files (>1MB)
2. Monitor PM2 logs for timeout errors
3. Verify Q&As are generated in chunks
4. Check retry logic works for transient failures

For JSON Parsing:

1. Generate Q&As from various file types
2. Verify all responses are valid JSON
3. Check error logs for parsing issues
4. Confirm temperature setting produces consistent outputs

For Image Validation:

1. Upload various image formats (PNG, JPEG, GIF, WebP)
2. Upload PDFs and verify conversion
3. Try uploading corrupted/invalid files
4. Verify error messages are clear and helpful
5. Check that invalid files are cleaned up

Configuration Changes Summary

Setting	Old Value	New Value	Reason
QA_GENERATION_TIMEOUT	300000ms (5min)	600000ms (10min)	Prevent timeouts on large files
MAX_FILE_SIZE_MB	5MB	2MB	Reduce processing time
MAX_CONCURRENT_FILES	3	2	Reduce Ollama load
Temperature	0.7	0.3	More consistent JSON output
CHUNK_SIZE	N/A	3000 chars	Enable chunking
MAX_RETRIES	N/A	2	Handle transient failures

Monitoring

After deployment, monitor:
1. PM2 logs for timeout errors
2. Q&A generation success rate
3. JSON parsing error rate
4. Image upload success rate
5. System resource usage (CPU, memory)

If issues persist:
- Increase timeout further
- Reduce chunk size
- Lower concurrency more
- Adjust temperature
- Check Ollama service health