Claude Code Debug Prompt: HEIC Upload Still Broken

Context - What Was Already Fixed

Based on your previous analysis, we made these changes:
1. ✅ Removed `Content-Disposition: "attachment"` from S3 presigned URL generation
2. ✅ Removed Content-Disposition header logic from client upload code
3. ✅ Changed `/api/file-url` to use presigned URLs (`isPublic: false`)
4. ✅ Added URL resolution in admin dashboard to properly display S3 images
5. ✅ Modified `convertHeicToJpeg` to return `{ blob, preview, success }`
6. ✅ Modified `handleFiles` to upload converted JPEG instead of raw HEIC
7. ✅ `heic2any` is installed in package.json: `"heic2any": "^0.0.4"`

What's Working

- JPEG, PNG, WebP uploads work perfectly
- These formats show previews during upload
- These formats display correctly in admin dashboard after submission

What's Still Broken

- **HEIC files show broken preview** during upload (empty/broken image)
- **HEIC files show broken in admin dashboard** after submission
- This suggests the HEIC conversion is **failing silently** and either:
- The original HEIC is being uploaded (which browsers can't display)
- Nothing is being uploaded at all

The Problem

iPhones use HEIC as the native photo format. If users take photos on iPhone and submit them, they won't work. This is a critical issue.

Current Code

photo-uploader.tsx (handles HEIC conversion)

```
"use client";

import { useState, useCallback, useRef } from "react";
import { Upload, X, AlertTriangle, FileImage, Loader2 } from "lucide-react";

interface PhotoFile {
  id: string;
  file: File;
  preview: string;
  caption: string;
  subCategory?: string;
  orientation: 'landscape' | 'portrait';
  isUploading?: boolean;
  cloud_storage_path?: string;
  fileName?: string;
  isConverting?: boolean;
}

interface PhotoUploaderProps {
  roomCategory: string;
  subcategories?: string[];
  photos: PhotoFile[];
  onPhotosChange: (photos: PhotoFile[]) => void;
  maxPhotos?: number;
}

export function PhotoUploader({
  roomCategory,
  subcategories,
  photos,
  onPhotosChange,
  maxPhotos = 60
}: PhotoUploaderProps) {
  const [dragActive, setDragActive] = useState(false);
  const inputRef = useRef<HTMLInputElement>(null);
  const [previewErrors, setPreviewErrors] = useState<Record<string, boolean>>({});
  const [convertingCount, setConvertingCount] = useState(0);

  // Convert HEIC to JPEG using heic2any library
  const convertHeicToJpeg = async (file: File): Promise<{ blob: Blob; preview:
string; success: boolean }> => {
    try {
      const heic2any = (await import('heic2any')).default;
      const convertedBlob = await heic2any({
        blob: file,
        toType: 'image/jpeg',
        quality: 0.9
      });

      // heic2any can return array or single blob
      const blob = Array.isArray(convertedBlob) ? convertedBlob[0] : convertedBlob;
      const preview = URL.createObjectURL(blob);
      return { blob, preview, success: true };
    } catch (error) {
      console.error('HEIC conversion failed:', error);
      // DON'T return the original HEIC file - browsers can't display it
      // Return empty preview so the fallback UI (filename icon) shows
      return { blob: file, preview: '', success: false };
    }
  };

  const checkOrientation = (blobOrFile: Blob | File): Promise<'landscape' |
```

```
'portrait'> => {
    return new Promise((resolve) => {
      const img = document.createElement('img');
      const url = URL.createObjectURL(blobOrFile);
      img.onload = () => {
        resolve(img.width >= img.height ? 'landscape' : 'portrait');
        URL.revokeObjectURL(url);
      };
      img.onerror = () => {
        URL.revokeObjectURL(url);
        resolve('landscape');
      };
      img.src = url;
    });
  };

  const isHeicFile = (file: File): boolean => {
    const fileExt = file.name?.split('.').pop()?.toLowerCase();
    const mimeType = file?.type?.toLowerCase() || '';
    return mimeType.includes('heic') || mimeType.includes('heif') ||
           fileExt === 'heic' || fileExt === 'heif';
  };

  const handleFiles = useCallback(async (files: FileList) => {
    const validTypes = ['image/jpeg', 'image/png', 'image/webp', 'image/heic', 'image/
heif'];
    const filesToProcess: File[] = [];

    for (let i = 0; i < files.length; i++) {
      const file = files[i];
      const fileExt = file.name?.split('.').pop()?.toLowerCase();
      const isValidType = validTypes.includes(file?.type?.toLowerCase() || '') ||
                          fileExt === 'heic' || fileExt === 'heif';
      if (!isValidType) continue;
      if (photos.length + filesToProcess.length >= maxPhotos) break;
      filesToProcess.push(file);
    }

    // Count HEIC files for loading indicator
    const heicCount = filesToProcess.filter(isHeicFile).length;
    if (heicCount > 0) {
      setConvertingCount(heicCount);
    }

    const newPhotos: PhotoFile[] = [];

    for (let i = 0; i < filesToProcess.length; i++) {
      const file = filesToProcess[i];

      let preview: string;
      let orientation: 'landscape' | 'portrait';
      let fileToUpload: File = file;

      if (isHeicFile(file)) {
        // Convert HEIC to JPEG for preview AND upload
        const converted = await convertHeicToJpeg(file);
        preview = converted.preview;
        setConvertingCount(prev => Math.max(0, prev - 1));

        if (converted.success) {
          // Create a new File from the converted blob for upload
          const convertedFile = new File(
            [converted.blob],
```

```
            file.name.replace(/\.heic$/i, '.jpg').replace(/\.heif$/i, '.jpg'),
            { type: 'image/jpeg' }
          );
          fileToUpload = convertedFile;
          orientation = await checkOrientation(converted.blob);
        } else {
          // Conversion failed - still add the photo but with empty preview
          // The original HEIC will be uploaded (admin can request re-upload if
needed)
          orientation = 'landscape'; // Default since we can't check
        }
      } else {
        preview = URL.createObjectURL(file);
        orientation = await checkOrientation(file);
      }

      newPhotos.push({
        id: `${Date.now()}-${i}`,
        file: fileToUpload, // Upload converted JPEG for HEIC, original for others
        preview,
        caption: '',
        subCategory: subcategories?.[0] || undefined,
        orientation,
        fileName: file.name // Keep original filename for display
      });
    }

    setConvertingCount(0);
    onPhotosChange([...photos, ...newPhotos]);
  }, [photos, onPhotosChange, maxPhotos, subcategories]);

  // ... rest of component
}
```

submission-form.tsx (handles upload to S3)

```
const uploadPhotos = async (): Promise<{ roomCategory: string; subCategory?: string;
caption: string; originalUrl: string; orientation: string }[]> => {
    const uploadedPhotos: { roomCategory: string; subCategory?: string; caption: strin
g; originalUrl: string; orientation: string }[] = [];

    for (const [room, photos] of Object.entries(photosByRoom ?? {})) {
      for (const photo of (photos ?? [])) {
        if (!photo?.file) continue;

        const contentType = getContentType(photo.file);

        // Get presigned URL
        const presignedRes = await fetch('/api/upload/presigned', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({
            fileName: photo.file.name,
            contentType: contentType,
            isPublic: true
          })
        });

        if (!presignedRes.ok) {
          console.error('Failed to get presigned URL:', await presignedRes.text());
          throw new Error('Failed to get upload URL');
        }

        const { uploadUrl, cloud_storage_path } = await presignedRes.json();

        if (!uploadUrl || !cloud_storage_path) {
          console.error('Invalid presigned URL response');
          throw new Error('Invalid upload URL');
        }

        // Upload to S3 - just need Content-Type header
        const uploadRes = await fetch(uploadUrl, {
          method: 'PUT',
          body: photo.file,
          headers: {
            'Content-Type': contentType
          }
        });

        if (!uploadRes.ok) {
          console.error('Failed to upload to S3:', uploadRes.status, await up-
loadRes.text());
          throw new Error('Failed to upload photo');
        }

        uploadedPhotos.push({
          roomCategory: room,
          subCategory: photo.subCategory,
          caption: photo.caption,
          originalUrl: cloud_storage_path,
          orientation: photo.orientation
        });
      }
    }
    return uploadedPhotos;
  };

// Helper to get content type
```

```
const getContentType = (file: File): string => {
  if (file.type) return file.type;
  const ext = file.name?.split('.').pop()?.toLowerCase();
  const mimeTypes: Record<string, string> = {
    'jpg': 'image/jpeg',
    'jpeg': 'image/jpeg',
    'png': 'image/png',
    'webp': 'image/webp',
    'heic': 'image/heic',
    'heif': 'image/heif'
  };
  return mimeTypes[ext || ''] || 'application/octet-stream';
};
```

Questions to Investigate

1. **Is heic2any actually loading?** The dynamic import `await import('heic2any')` might be failing silently in production or in certain browsers.
2. **Is the conversion actually happening?** Even if heic2any loads, the conversion might be failing. Check if there's a browser compatibility issue.
3. **Is the File object being properly created?** After conversion, we create a new File from the blob:
   `js`
   `const convertedFile = new File(`
   `[converted.blob],`
   `file.name.replace(/\.heic$/i, '.jpg').replace(/\.heif$/i, '.jpg'),`
   `{ type: 'image/jpeg' }`
   `);`
   Is this working correctly?
4. **Is photo.file getting the converted file?** When `fileToUpload` is assigned to `newPhotos`, is it actually being used during upload in submission-form.tsx?
5. **Could there be a timing/async issue?** Is the state update with the converted file happening before the upload begins?

What I Need From You

1. **Root cause analysis**: Why is heic2any conversion failing (or not happening at all)?
2. **Specific code fixes**: Exact changes needed to make HEIC uploads work
3. **Debugging suggestions**: What logs should we add to pinpoint the exact failure point?
4. **Alternative approaches**: If heic2any is unreliable, what are other options for HEIC conversion?

Additional Context

- This is a Next.js 14 app with App Router
- Running on Node.js in production
- heic2any version: 0.0.4
- The conversion should happen client-side in the browser before upload
- Target users include iPhone users who will have HEIC photos by default