🔧 Fix: Mejora en Upload de Imagen de Perfil del Cliente

Fecha: 15 de Noviembre 2025
Tipo: Fix de manejo de errores y debugging
Impacto: Resuelve error “JSON.parse: unexpected character”

📋 Problema Identificado

Error Reportado por Usuario

```
JSON.parse: unexpected character at line 1 column 1 of the JSON data
```

Causa Raíz

1. **Falta de validación de Content-Type**: El frontend intentaba parsear respuestas como JSON sin verificar el tipo de contenido
2. **Errores no capturados**: Errores en el servidor no estaban siendo manejados correctamente
3. **Falta de logging**: Era difícil diagnosticar dónde fallaba el proceso de upload

---

✅ Solución Implementada

1. Mejoras en el Frontend (client-profile-image.tsx)

Validación de Content-Type

```
// NUEVO: Verificar content-type antes de parsear
const contentType = response.headers.get('content-type');
if (!contentType || !contentType.includes('application/json')) {
  const textResponse = await response.text();
  console.error('[ClientProfileImage] Respuesta no JSON:', textResponse);
  throw new Error('El servidor no devolvió una respuesta JSON válida...');
}

const data = await response.json();
```

Beneficios:
- ✅ Detecta respuestas no JSON (HTML de error, texto plano)
- ✅ Muestra el contenido real de la respuesta en consola
- ✅ Proporciona un mensaje de error más descriptivo al usuario

Logging Mejorado

```
console.log('[ClientProfileImage] Status de respuesta:', response.status);
console.log('[ClientProfileImage] Content-Type:', response.headers.get('content-type')
);
```

2. Mejoras en el Backend (profile-image/route.ts)

Configuración de Runtime

```
// NUEVO: Configuración para manejar archivos
export const runtime = 'nodejs';
export const dynamic = 'force-dynamic';
```

Beneficios:
- ✅ Usa runtime de Node.js para mejor manejo de archivos
- ✅ Fuerza renderizado dinámico (necesario para FormData)

Manejo de Errores Granular

```
// 1. Error al parsear FormData
try {
  formData = await request.formData();
} catch (parseError: any) {
  return NextResponse.json(
    { error: 'Error al procesar el archivo...', details: parseError.message },
    { status: 400 }
  );
}

// 2. Error al crear Buffer
try {
  const arrayBuffer = await file.arrayBuffer();
  buffer = Buffer.from(arrayBuffer);
} catch (bufferError: any) {
  return NextResponse.json(
    { error: 'Error al procesar el archivo', details: bufferError.message },
    { status: 500 }
  );
}

// 3. Error al crear directorio
try {
  profileDir = ensureProfileDirectory();
} catch (dirError: any) {
  return NextResponse.json(
    { error: 'Error al crear directorio...', details: dirError.message },
    { status: 500 }
  );
}

// 4. Error al guardar archivo
try {
  relativePath = await saveFileLocally(buffer, fileName, profileDir);
} catch (saveError: any) {
  return NextResponse.json(
    { error: 'Error al guardar el archivo', details: saveError.message },
    { status: 500 }
  );
}
```

Beneficios:
- ✅ Captura errores en cada paso del proceso
- ✅ Siempre devuelve respuestas JSON válidas
- ✅ Proporciona detalles específicos del error

Logging Detallado

```
console.log('[profile-image POST] Inicio de request para clientId:', params.id);
console.log('[profile-image POST] Content-Type de request:', request.headers.get('con-
tent-type'));
console.log('[profile-image POST] Usuario autenticado:', session.user.email);
console.log('[profile-image] Convirtiendo archivo a buffer...');
console.log('[profile-image] Buffer creado correctamente:', buffer.length, 'bytes');
console.log('[profile-image] Nombre de archivo generado:', fileName);
console.log('[profile-image] Directorio de perfil:', profileDir);
console.log('[profile-image] Guardando archivo...');
console.log('[profile-image] Archivo guardado en:', relativePath);
```

Beneficios:
- ✅ Trazabilidad completa del proceso de upload
- ✅ Facilita debugging en producción
- ✅ Permite identificar exactamente dónde falla el proceso

📦 Archivos Modificados

```
app/api/clients/[id]/profile-image/route.ts          # Backend API
app/components/clients/client-profile-image.tsx     # Frontend Component
```

🔍 Debugging en Producción

Cómo Diagnosticar Problemas

1. Logs del Frontend (Browser Console)

```
[ClientProfileImage] Iniciando upload de imagen...
[ClientProfileImage] Status de respuesta: 200
[ClientProfileImage] Content-Type: application/json
[ClientProfileImage] Respuesta del servidor: {...}
```

2. Logs del Backend (Server Console)

```
[profile-image POST] Inicio de request para clientId: abc123
[profile-image POST] Content-Type de request: multipart/form-data
[profile-image POST] Usuario autenticado: admin@escalafin.com Role: ADMIN
[profile-image] Convirtiendo archivo a buffer...
[profile-image] Buffer creado correctamente: 245678 bytes
[profile-image] Nombre de archivo generado: profile-abc123-1700000000.jpg
[profile-image] Directorio de perfil: /app/uploads/profile-images
[profile-image] Guardando archivo...
[profile-image] Archivo guardado en: profile-images/profile-abc123-1700000000.jpg
```

3. Errores Comunes y Soluciones

Error	Causa	Solución
`unexpected character at line 1`	Respuesta HTML en lugar de JSON	Verificar logs del servidor
`Error al parsear FormData`	Archivo corrupto o muy grande	Verificar tamaño y formato
`Error al crear directorio`	Permisos insuficientes	Verificar permisos del directorio
`Error al guardar el archivo`	Disco lleno o permisos	Verificar espacio y permisos

🧪 Cómo Probar

Test 1: Upload Normal

```
1. Ir a Admin → Clientes → [Cliente] → Editar
2. Click en "Cambiar" o "Subir" foto
3. Seleccionar una imagen válida (JPG/PNG, < 5MB)
4. Verificar que se muestra "Imagen actualizada correctamente"
5. Verificar en consola los logs de éxito
```

Test 2: Archivo No Válido

```
1. Intentar subir un archivo PDF o TXT
2. Verificar mensaje: "Solo se permiten imágenes (JPEG, PNG, WebP)"
```

Test 3: Archivo Muy Grande

```
1. Intentar subir una imagen > 5MB
2. Verificar mensaje: "La imagen es demasiado grande. Máximo 5MB"
```

Test 4: Sin Permisos

```
1. Como cliente (no admin), intentar cambiar foto existente
2. Verificar mensaje: "Solo puede subir la foto al registrarse..."
```

🎯 Resultado Esperado

Casos de Éxito

```
✅ Frontend recibe respuesta JSON válida
✅ Logs detallados en ambos lados
✅ Mensajes de error descriptivos
✅ Archivo se guarda correctamente
✅ Base de datos se actualiza
```

Casos de Error

```
✅ Errores se capturan correctamente
✅ Siempre se devuelve JSON (nunca HTML)
✅ Logs muestran dónde falló
✅ Usuario recibe mensaje claro
```

🔄 Próximos Pasos

Inmediato

1. ✅ Commit y push de cambios
2. ⏳ Deploy en EasyPanel
3. ⏳ Probar upload de imagen en producción
4. ⏳ Verificar logs en producción

Futuro

- Considerar agregar preview de imagen antes de upload
- Implementar crop/resize de imágenes en el frontend
- Agregar indicador de progreso para archivos grandes
- Implementar retry automático en caso de fallo temporal

📊 Comparación Antes/Después

Aspecto	Antes	Después
Validación Content-Type	❌ No existe	✅ Valida antes de parsear
Manejo de errores	⚠️ Básico	✅ Granular por etapa
Logging	⚠️ Mínimo	✅ Detallado y trazable
Mensajes de error	⚠️ Genéricos	✅ Específicos y descriptivos
Debugging	❌ Difícil	✅ Fácil con logs completos
Respuestas	⚠️ Inconsistentes	✅ Siempre JSON válido

📝 Notas Técnicas

¿Por qué `runtime = 'nodejs'`?

- Next.js por defecto usa Edge Runtime para API routes
- Edge Runtime tiene limitaciones con archivos grandes y Buffer
- Node.js runtime es necesario para `fs.writeFile` y operaciones de archivo

¿Por qué `dynamic = 'force-dynamic'`?

- FormData requiere renderizado dinámico (no puede ser estático)
- Evita errores de “Dynamic server usage” durante build
- Garantiza que la ruta siempre se ejecuta en el servidor

Seguridad

- ✅ Validación de tipo de archivo (JPEG, PNG, WebP)
- ✅ Validación de tamaño (máx 5MB)
- ✅ Validación de permisos (solo admin puede cambiar)
- ✅ Sanitización de nombres de archivo
- ✅ Path traversal protection (uso de path.join)

🔗 Referencias

- Next.js API Routes (https://nextjs.org/docs/app/building-your-application/routing/route-handlers)
- FormData API (https://developer.mozilla.org/en-US/docs/Web/API/FormData)
- Node.js Buffer (https://nodejs.org/api/buffer.html)
- Next.js Runtime Configuration (https://nextjs.org/docs/app/api-reference/file-conventions/route-segment-config#runtime)

Estado: ✅ Implementado y testeado
Build: ✅ Exitoso
Commit: 7815072
Pendiente: Deploy en EasyPanel