Fix: Imagen de Perfil Cliente - Almacenamiento Local

Fecha: 13 de noviembre de 2025
Tipo: Fix de funcionalidad
Prioridad: Alta

🔍 Problema Reportado

Al actualizar la imagen de perfil de un cliente desde el panel de administración:
1. Se abría el selector de archivos correctamente
2. Al seleccionar una imagen, la pantalla cambiaba inesperadamente
3. Se producía un error al intentar guardar
4. La imagen no se guardaba correctamente

Causa raíz: El sistema estaba intentando usar el almacenamiento unificado (que puede intentar usar S3/Google Drive), causando errores de configuración o permisos.

✅ Solución Implementada

Cambios Realizados

1. API Endpoint de Imagen de Perfil

Archivo: `app/api/clients/[id]/profile-image/route.ts`

Cambios principales:
- ✅ Eliminada dependencia de `unified-storage`
- ✅ Usa directamente `saveFileLocally` y `deleteFileLocally`
- ✅ Crea directorio dedicado para imágenes de perfil: `/app/uploads/profile-images/`
- ✅ Simplificado el flujo de guardado
- ✅ Mejor manejo de errores

Antes:

```
import { uploadFile, deleteFile, getStorageInfo } from '@/lib/unified-storage';
// ...
const uploadResult = await uploadFile(buffer, fileName, file.type, {...});
```

Después:

```
import { saveFileLocally, deleteFileLocally } from '@/lib/local-storage';
// ...
const PROFILE_IMAGES_DIR = process.env.LOCAL_STORAGE_PATH || '/app/uploads';
const PROFILE_FOLDER = 'profile-images';

function ensureProfileDirectory(): string {
  const profilePath = path.join(PROFILE_IMAGES_DIR, PROFILE_FOLDER);
  if (!existsSync(profilePath)) {
    mkdirSync(profilePath, { recursive: true });
  }
  return profilePath;
}

const profileDir = ensureProfileDirectory();
const relativePath = await saveFileLocally(buffer, fileName, profileDir);
```

2. Componente de Imagen de Perfil

Archivo: `app/components/clients/client-profile-image.tsx`

Mejoras implementadas:
- ✅ `preventDefault()` y `stopPropagation()` en eventos
- ✅ Atributo `type="button"` en todos los botones
- ✅ Logs de consola para debugging
- ✅ Mejor manejo de eventos en el input file

Antes:

```
<Button onClick={() => document.getElementById(...)?.click()}>
```

Después:

```
<Button
  type="button"
  onClick={(e) => {
    e.preventDefault();
    e.stopPropagation();
    document.getElementById(...)?.click();
  }}
>
```

3. Fix de Contraste en Botón de Login

Archivos:
- `app/components/auth/login-form.tsx`
- `app/app/page.tsx`

Problema detectado: Botones con `text-white` sobre fondo blanco (ratio de contraste: 1:1)

Solución:

```
// login-form.tsx
className="w-full bg-blue-600 text-white py-3.5 px-4 rounded-lg ..."

// page.tsx
className="px-8 border-white text-white bg-transparent hover:bg-white hover:text-
blue-600"
```

🎯 Funcionalidad Actual

Flujo de Carga de Imagen

1. **Usuario hace clic en “Cambiar” o “Subir”**
   - Se previene el comportamiento por defecto
   - Se abre el selector de archivos nativo

2. **Usuario selecciona una imagen**
   - Validación de tipo (JPEG, PNG, WebP)
   - Validación de tamaño (máximo 5MB)
   - Se muestra spinner de carga

3. **Subida al servidor**
   - Se crea FormData con el archivo
   - POST a `/api/clients/[id]/profile-image`
   - El archivo se guarda en `/app/uploads/profile-images/`
   - Se genera nombre único: `profile-{clientId}-{timestamp}.{ext}`

4. **Actualización en base de datos**
   - Se guarda la ruta relativa en `client.profileImage`
   - Se elimina la imagen anterior si existía

5. **Actualización en UI**
   - Se actualiza el estado local
   - Se notifica al componente padre
   - Se muestra toast de éxito

Rutas de Almacenamiento

```
/app/uploads/
└── profile-images/
    ├── profile-cuid123-1699900000000.jpg
    ├── profile-cuid456-1699900001000.png
    └── profile-cuid789-1699900002000.webp
```

Permisos

- **Cliente**: Solo puede subir imagen al registrarse (si no tiene imagen)
- **Admin**: Puede cambiar la imagen en cualquier momento
- **Eliminación**: Solo admin puede eliminar imágenes

🔧 Endpoints API Modificados

POST `/api/clients/[id]/profile-image`

Sube o actualiza la imagen de perfil

Request:

```
FormData {
  file: File
}
```

Response:

```
{
  "success": true,
  "message": "Imagen de perfil actualizada correctamente",
  "client": {
    "id": "...",
    "firstName": "...",
    "lastName": "...",
    "profileImage": "profile-images/profile-cuid-timestamp.jpg"
  },
  "storage": "local"
}
```

DELETE `/api/clients/[id]/profile-image`

Elimina la imagen de perfil (solo admin)

Response:

```
{
  "success": true,
  "message": "Imagen de perfil eliminada correctamente"
}
```

GET `/api/clients/[id]/profile-image`

Obtiene la ruta de la imagen

Response:

```
{
  "success": true,
  "profileImage": "profile-images/profile-cuid-timestamp.jpg",
  "storage": "local"
}
```

🧪 Pruebas Realizadas

✅ Casos de Prueba Exitosos

1. **Subida de imagen nueva**
   - Cliente sin imagen puede subir
   - Admin puede subir imagen para cualquier cliente
   - Se crea el directorio si no existe

2. **Actualización de imagen existente**
   - Admin puede cambiar la imagen
   - La imagen anterior se elimina correctamente
   - No hay imágenes huérfanas

3. **Validaciones**
   - ✅ Rechaza tipos de archivo no permitidos (PDF, DOC, etc)
   - ✅ Rechaza archivos mayores a 5MB
   - ✅ Valida permisos correctamente

4. **Interfaz de usuario**
   - ✅ No hay cambio de pantalla al seleccionar archivo
   - ✅ Spinner muestra durante la carga
   - ✅ Toast de confirmación se muestra
   - ✅ Imagen se actualiza inmediatamente

5. **Build y despliegue**
   - ✅ Build exitoso sin errores
   - ✅ TypeScript compila correctamente
   - ✅ No hay problemas de contraste (WCAG AA)

📊 Impacto

Beneficios

- ✅ **Funcionalidad restaurada**: Admin puede actualizar imágenes sin problemas
- ✅ **Experiencia de usuario mejorada**: No hay cambios inesperados de pantalla
- ✅ **Simplicidad**: No depende de configuraciones externas (S3, Google Drive)
- ✅ **Confiabilidad**: El almacenamiento local es más predecible
- ✅ **Accesibilidad**: Mejor contraste en botones (cumple WCAG AA)

Archivos Modificados

- ✅ `app/api/clients/[id]/profile-image/route.ts`
- ✅ `app/components/clients/client-profile-image.tsx`
- ✅ `app/components/auth/login-form.tsx`
- ✅ `app/app/page.tsx`

Sin Impacto en

- ✅ Otros tipos de archivos (documentos, contratos, etc.)
- ✅ Sistema de almacenamiento unificado para otros usos
- ✅ Permisos y validaciones existentes

- ✅ Funcionalidad del resto de la aplicación

🚀 Despliegue

Variables de Entorno Necesarias

```
LOCAL_STORAGE_PATH=/app/uploads  # (opcional, valor por defecto)
```

Pasos de Despliegue

1. **Pull del repositorio**
   `bash`
   `git pull origin main`

2. **Limpiar caché**
   - En EasyPanel, limpiar el build cache

3. **Rebuild**
   - Hacer rebuild completo de la aplicación

4. **Verificar**
   - Probar subida de imagen desde admin
   - Verificar que la imagen se muestra correctamente
   - Verificar que no hay errores en logs

Verificación Post-Despliegue

```
# Verificar que existe el directorio
ls -la /app/uploads/profile-images/

# Ver imágenes subidas
ls -lh /app/uploads/profile-images/

# Ver logs de la aplicación
docker logs [container-id] | grep "ClientProfileImage"
```

📝 Notas Técnicas

Estructura de Directorios

```
/app/uploads/
├── escalafin/              # Otros archivos del sistema
│   ├── sistema/
│   └── clientes/
└── profile-images/         # Imágenes de perfil (nuevo)
    └── profile-*.{jpg,png,webp}
```

Formato de Nombres de Archivo

```
profile-{clientId}-{timestamp}.{extension}

Ejemplo:
profile-cm2abc123xyz-1699900000000.jpg
```

Validaciones Implementadas

- Tipos permitidos: `image/jpeg`, `image/jpg`, `image/png`, `image/webp`
- Tamaño máximo: 5MB (5 * 1024 * 1024 bytes)
- Permisos: Cliente (solo al registrarse), Admin (siempre)

⚠️ Consideraciones

Espacio en Disco

- Las imágenes de perfil ocupan espacio en el servidor
- Considerar implementar limpieza automática de imágenes antiguas
- Estimar ~500KB promedio por imagen
- Para 1000 clientes: ~500MB de espacio

Backups

- Las imágenes de perfil deben incluirse en los backups
- Directorio: `/app/uploads/profile-images/`
- Considerar backup incremental

Escalabilidad

- Para grandes volúmenes, considerar migrar a S3/CDN en el futuro
- El sistema actual es adecuado para < 10,000 usuarios
- Para más usuarios, implementar CDN para mejor rendimiento

🔗 Referencias

- Prisma Schema: `app/prisma/schema.prisma` (campo `profileImage`)
- Local Storage: `app/lib/local-storage.ts`
- Image API: `app/api/images/[...path]/route.ts`
- Client API: `app/api/clients/[id]/route.ts`

✅ Checklist de Verificación

- [x] API endpoint modificado y probado
- [x] Componente actualizado con mejores eventos
- [x] Validaciones implementadas
- [x] Pruebas de subida exitosas

- [x] Pruebas de actualización exitosas
- [x] Pruebas de eliminación exitosas
- [x] Build exitoso
- [x] No hay errores de TypeScript
- [x] No hay problemas de contraste
- [x] Documentación completada
- [x] Checkpoint guardado

Estado: ✅ Completado y listo para despliegue
Última actualización: 13 de noviembre de 2025