📘 Documentación - Media Enhancer

Aplicación de Análisis y Mejora de Medios con Inteligencia Artificial

Powered by Google Gemini 2.0 + Imagen 3

📋 Índice

🎯 ¿Qué es Media Enhancer?

Media Enhancer es una aplicación web prototipo que utiliza inteligencia artificial (Google Gemini 2.0) para analizar, mejorar y generar contenido multimedia.

Propósito

Proporcionar un conjunto de herramientas de IA para:

- Analizar imágenes, vídeos y audio
- Mejorar la calidad de archivos multimedia
- Generar imágenes creativas desde descripciones
- Crear storyboards automáticos de vídeos

URLs del Proyecto

- **Aplicación Desplegada**: https://media-enhancer.vercel.app/
- **Repositorio GitHub**: https://github.com/VCNPRO/Media-Enhancer
- **Panel Vercel**: https://vercel.com/solammedia-9886s-projects/media-enhancer
- **AI Studio**: https://ai.studio/apps/drive/1ESq_gN8TKwn7XJE3oWIG9V_zR1d18PFs

---

⚙️ Funcionalidades Principales

Media Enhancer está organizado en **4 módulos principales**:

1️⃣ Análisis de Medios (Analysis)

Componente: `AnalysisPanel.tsx`

Formatos Soportados:

- 📷 Imágenes (JPG, PNG, GIF, WebP)
- 🎥 Vídeos (MP4, WebM, MOV, AVI)
- 🎵 Audio (MP3, WAV, M4A, OGG)

Capacidades:

Análisis Conversacional con IA

- Haz preguntas sobre cualquier archivo multimedia
- Gemini 2.0 analiza el contenido y responde
- Soporte para análisis visual, de audio y de vídeo

Ejemplos de preguntas:

```
"Describe detalladamente qué hay en esta imagen"
"Transcribe el audio de este vídeo"
"¿Qué objetos aparecen en la escena?"
"¿De qué trata esta conversación?"
"Identifica las personas y lugares mencionados"
```

Generación de Storyboard (Solo Vídeos)

- Extrae fotogramas clave del vídeo
- Genera descripciones para cada fotograma
- Ideal para:
  - Previsualización rápida de contenido
  - Documentación de vídeos
  - Creación de índices visuales

Resultado:

```
{
  description: "Descripción general del contenido",
  tags: ["tag1", "tag2", "tag3"],
  transcript?: "Transcripción completa del audio (si aplica)"
}

// Storyboard:
[
  {
    imageUrl: "...",
    description: "Descripción del fotograma 1"
  },
  {
    imageUrl: "...",
    description: "Descripción del fotograma 2"
  }
]
```

2️⃣ Mejora de Medios (Enhancement)

Componente: `EnhancementPanel.tsx`

Formatos Soportados:

- 📷 Imágenes
- 🎵 Audio

Mejora de Imágenes

Funcionalidades:

- ✅ Mejora automática de calidad
- ✅ Mejora de nitidez y claridad
- ✅ Optimización de colores
- ✅ **Colorización** (convertir blanco y negro a color)

Opciones Disponibles:

```
{
  colorize: boolean  // Activar colorización de imágenes B&N
}
```

Herramientas Incluidas:

- **Comparador Antes/Después**: Slider interactivo para comparar original vs mejorado
- Componente: `ImageComparator.tsx`

Mejora de Audio

Funcionalidades:

- ✅ Reducción de ruido
- ✅ Mejora de claridad
- ✅ Normalización de volumen
- ✅ Salida en formato WebM

Resultado:

```
{
  originalUrl: "blob:...",
  enhancedUrl: "data:audio/webm;base64,...",
  type: "audio"
}
```

3️⃣ Generación Creativa (Creative Tools)

Componente: `CreativeToolsPanel.tsx`

Qué hace: Genera imágenes desde cero usando **Google Imagen 3** (modelo de generación de imágenes de última generación).

Proceso:

1. Usuario escribe un prompt descriptivo
2. Imagen 3 genera la imagen
3. Resultado en formato base64

Ejemplos de Prompts:

```
"Un astronauta montando a caballo en Marte, estilo fotorrealista"

"Un bosque mágico con árboles brillantes y criaturas fantásticas, estilo Stud

"Retrato de una mujer victoriana en jardín, óleo sobre lienzo"

"Ciudad cyberpunk futurista con neones, lluvia, estilo blade runner"
```

Consejos para Buenos Prompts:

- Sé específico con los detalles
- Menciona el estilo artístico deseado
- Describe iluminación, colores, ambiente
- Incluye referencias a estilos conocidos

Resultado:

```
{
  prompt: "Descripción utilizada",
  imageUrl: "..."
}
```

4️⃣ Historial (History)

Componente: `HistorySidebar.tsx`

Funcionalidades:

- ✅ Almacena todos los análisis realizados
- ✅ Guarda todas las mejoras aplicadas
- ✅ Registra todas las imágenes generadas
- ✅ Permite recuperar resultados anteriores
- ✅ Mantiene referencia al archivo original

Tipos de Elementos en Historial:

```
type HistoryItemType =
  | 'analysis'      // Análisis de medios
  | 'enhancement'   // Mejora de imagen/audio
  | 'creative'      // Generación de imagen
  | 'storyboard'    // Storyboard de vídeo
```

Información Guardada:

- ID único
- Timestamp
- Tipo de operación
- Tipo de medio (imagen/vídeo/audio)
- Resultado completo
- Archivo original (si aplica)

💼 Casos de Uso

Caso 1: Análisis de Contenido de Vídeo

Situación: Tienes un vídeo largo y necesitas saber qué contiene sin verlo completo.

Proceso:

1. Sube el vídeo a Media Enhancer
2. Usa "Análisis" con prompt: "Resume el contenido de este vídeo"
3. Genera Storyboard para ver fotogramas clave
4. Obtén transcripción si hay audio

Resultado: Resumen completo + storyboard visual en minutos

Caso 2: Restauración de Fotos Antiguas

Situación: Fotos antiguas en blanco y negro o de baja calidad.

Proceso:

1. Sube la foto antigua
2. Ve a "Mejora"
3. Activa "Colorización" si es B&N
4. Compara antes/después con el slider
5. Descarga la imagen mejorada

Resultado: Foto restaurada y colorizada con IA

Caso 3: Generación de Imágenes para Proyectos

Situación: Necesitas imágenes conceptuales para presentación o diseño.

Proceso:

1. Ve a "Herramientas Creativas"
2. Escribe descripción detallada
3. Genera la imagen
4. Ajusta prompt si es necesario
5. Descarga resultado

Resultado: Imágenes únicas generadas por IA

Caso 4: Transcripción de Audio

Situación: Tienes archivo de audio de reunión o entrevista.

Proceso:

1. Sube el archivo de audio
2. Usa "Análisis" con prompt: "Transcribe este audio completo"
3. Obtén texto completo

Resultado: Transcripción precisa del audio

Caso 5: Mejora de Calidad de Audio

Situación: Audio con mucho ruido de fondo.

Proceso:

1. Sube el audio
2. Ve a "Mejora"
3. Aplica mejora automática
4. Escucha el resultado
5. Descarga audio mejorado

Resultado: Audio limpio y claro

🛠️ Stack Técnico

Frontend

Framework y Bibliotecas:

```
{
  "react": "^18.3.1",
  "typescript": "~5.6.2",
  "vite": "^5.4.2",
  "tailwindcss": "^3.4.13"
}
```

Componentes:

- `App.tsx` - Componente principal y orquestador
- `Header.tsx` - Cabecera de la aplicación
- `FileUpload.tsx` - Carga de archivos (drag & drop)
- `Tabs.tsx` - Sistema de pestañas
- `AnalysisPanel.tsx` - Panel de análisis
- `EnhancementPanel.tsx` - Panel de mejora
- `CreativeToolsPanel.tsx` - Panel de generación
- `HistorySidebar.tsx` - Barra lateral de historial
- `CustomVideoPlayer.tsx` - Reproductor de vídeo
- `ImageComparator.tsx` - Comparador de imágenes
- `Loader.tsx` - Indicador de carga
- `ErrorBoundary.tsx` - Manejo de errores

Backend / Servicios

IA y APIs:

- **Google Gemini 2.0**: Análisis de medios, transcripción, mejora
- **Google Imagen 3**: Generación de imágenes
- **API Key**: Configurada en `.env.local`

Servicios (`services/geminiService.ts`):

```typescript
// Análisis de medios (imagen/vídeo/audio)
analyzeMedia(file: File, prompt: string): Promise<AnalysisResult>

// Generación de storyboard (vídeo)
generateStoryboard(file: File): Promise<StoryboardFrame[]>

// Mejora de imagen
enhanceImage(file: File, options: ImageEnhancementOptions): Promise<strin

// Mejora de audio
enhanceAudio(file: File): Promise<string>

// Generación de imagen
generateImage(prompt: string): Promise<string>
```

Tipos TypeScript

Definiciones (`types.ts`):

```ts
export type MediaType = 'image' | 'video' | 'audio';

export interface MediaFile {
    file: File;
    name: string;
    url: string;
    type: MediaType;
}

export interface AnalysisResult {
    description: string;
    tags: string[];
    transcript?: string;
}

export interface ImageEnhancementOptions {
    colorize: boolean;
}

export interface EnhancementResult {
    originalUrl: string;
    enhancedUrl: string;
    type: MediaType;
}

export interface StoryboardFrame {
    imageUrl: string;
    description: string;
}

export interface CreativeResult {
    prompt: string;
    imageUrl: string;
}

export type HistoryItemPayload =
    | AnalysisResult
    | EnhancementResult
    | CreativeResult
    | StoryboardFrame[];

export interface HistoryItem {
    id: string;
    timestamp: Date;
```

```
    type: 'analysis' | 'enhancement' | 'creative' | 'storyboard';
    mediaType: MediaType | 'none';
    payload: HistoryItemPayload;
    mediaFile?: MediaFile;
}
```

🚀 Instalación y Configuración

Prerrequisitos

- **Node.js** (versión 16 o superior)
- **npm** o **yarn**
- **Gemini API Key** (obtener en https://aistudio.google.com/apikey)

Pasos de Instalación

1. **Clonar el repositorio**:

```
git clone https://github.com/VCNPRO/Media-Enhancer.git
cd Media-Enhancer
```

2. **Instalar dependencias**:

```
npm install
```

3. **Configurar API Key**:

Crear archivo `.env.local` en la raíz del proyecto:

```
VITE_GEMINI_API_KEY=tu_api_key_aquí
```

4. **Ejecutar en desarrollo**:

```
npm run dev
```

La aplicación estará disponible en: `http://localhost:5173`

5. **Build para producción**:

```
npm run build
```

Deployment en Vercel

1. Conecta tu repositorio de GitHub a Vercel
2. Configura la variable de entorno:
   - `VITE_GEMINI_API_KEY` = tu API key
3. Deploy automático

📖 Guía de Uso

Flujo de Trabajo General

```
1. SUBIR ARCHIVO
   └─ Arrastra archivo o haz clic en zona de carga
   └─ Soporta: imágenes, vídeos, audio

2. SELECCIONAR PESTAÑA
   ├─ 🎨 Herramientas Creativas (generar desde cero)
   ├─ 🔍 Análisis (analizar contenido)
   └─ ✨ Mejora (mejorar calidad)

3. CONFIGURAR OPERACIÓN
   └─ Escribe prompts o selecciona opciones

4. EJECUTAR
   └─ Click en botón correspondiente

5. VER RESULTADOS
   └─ Resultados mostrados en pantalla
   └─ Guardados automáticamente en Historial

6. DESCARGAR
   └─ Click derecho > Guardar imagen como...
   └─ O usar botones de descarga
```

Uso de Análisis

Para Imágenes:

```
Prompts útiles:
- "Describe esta imagen en detalle"
- "¿Qué objetos hay en la escena?"
- "Identifica el estilo artístico"
- "¿Qué emociones transmite?"
```

Para Vídeos:

```
Prompts útiles:
- "Resume el contenido de este vídeo"
- "Transcribe el audio"
- "¿Qué acciones ocurren?"
- "Identifica las escenas principales"

Storyboard:
- Click en "Generar Storyboard"
- Obtén fotogramas clave automáticamente
```

Para Audio:

```
Prompts útiles:
- "Transcribe este audio"
- "¿De qué trata la conversación?"
- "Resume los puntos principales"
- "Identifica los hablantes"
```

Uso de Mejora

Para Imágenes:

1. Sube la imagen
2. Ve a pestaña "Mejora"
3. (Opcional) Activa "Colorizar" si es B&N
4. Click en "Mejorar Imagen"
5. Usa el slider para comparar
6. Descarga resultado

Para Audio:

1. Sube el audio
2. Ve a pestaña "Mejora"
3. Click en "Mejorar Audio"
4. Escucha el resultado
5. Descarga audio mejorado

Uso de Generación Creativa

1. Ve a pestaña "Herramientas Creativas"
2. Escribe descripción detallada:

```
Ejemplo: "Un paisaje de montañas nevadas al atardecer,
cielo naranja y púrpura, estilo fotorrealista, 8K"
```

3. Click en "Generar Imagen"
4. Espera resultado (~10-30 segundos)
5. Descarga imagen generada

Consejos para Prompts Creativos:

- ✅ Sé específico con detalles visuales
- ✅ Menciona estilo artístico (realista, anime, pintura, etc.)
- ✅ Describe iluminación y atmósfera
- ✅ Incluye colores y texturas
- ✅ Referencia artistas o estilos conocidos

🔧 API y Servicios

Servicio de Gemini (`geminiService.ts`)

`analyzeMedia(file: File, prompt: string)`

Descripción: Analiza cualquier archivo multimedia con un prompt personalizado.

Parámetros:

- `file` : Archivo a analizar (imagen/vídeo/audio)
- `prompt` : Pregunta o instrucción para el análisis

Retorna:

```
Promise<{
  description: string;
  tags: string[];
  transcript?: string;
}>
```

Uso:

```
const result = await analyzeMedia(
  videoFile,
  "Transcribe el audio y describe las escenas principales"
);
```

`generateStoryboard(file: File)`

Descripción: Genera storyboard automático de un vídeo.

Parámetros:

- `file`: Archivo de vídeo

Retorna:

```
Promise<StoryboardFrame[]>
// Array de { imageUrl: string, description: string }
```

Uso:

```
const storyboard = await generateStoryboard(videoFile);
```

`enhanceImage(file: File, options: ImageEnhancementOptions)`

Descripción: Mejora la calidad de una imagen.

Parámetros:

- `file`: Archivo de imagen
- `options`: `{ colorize: boolean }`

Retorna:

```
Promise<string>  // Base64 de imagen mejorada
```

Uso:

```
const enhancedBase64 = await enhanceImage(
  imageFile,
  { colorize: true }
);
```

`enhanceAudio(file: File)`

Descripción: Mejora la calidad de un archivo de audio.

Parámetros:

- `file`: Archivo de audio

Retorna:

```
Promise<string>  // Base64 de audio mejorado (WebM)
```

Uso:

```
const enhancedAudioBase64 = await enhanceAudio(audioFile);
```

`generateImage(prompt: string)`

Descripción: Genera una imagen desde una descripción textual.

Parámetros:

- `prompt`: Descripción de la imagen a generar

Retorna:

```
Promise<string>  // Base64 de imagen generada
```

Uso:

```
const imageBase64 = await generateImage(
  "Un gato astronauta en el espacio, estilo cartoon"
);
```

⚠️ Limitaciones y Consideraciones

Limitaciones Técnicas

Tamaño de Archivos:

- Imágenes: Recomendado <10MB
- Vídeos: Recomendado <50MB
- Audio: Recomendado <20MB

Formatos:

- Algunos formatos pueden no ser soportados completamente
- Para mejor compatibilidad usar: JPG, PNG, MP4, MP3

Procesamiento:

- Tiempos de procesamiento varían según tamaño de archivo
- Análisis de vídeo puede tardar varios minutos
- Generación de imágenes: ~10-30 segundos

Limitaciones de la API

Gemini API:

- Límites de rate (consultas por minuto)
- Límites de cuota mensual
- Dependiente de disponibilidad del servicio

Imagen 3:

- Políticas de contenido (no genera contenido inapropiado)
- Límites de generación
- Calidad puede variar según prompt

Consideraciones de Privacidad

Datos del Usuario:

- Archivos se procesan en los servidores de Google
- No se almacenan permanentemente en el servidor
- Historial solo se guarda localmente en el navegador

Recomendaciones:

- No subir contenido sensible o privado
- Revisar términos de servicio de Google Gemini
- Usar en cumplimiento con GDPR si aplica

🎨 Diseño y UX

Paleta de Colores

Tema Oscuro (por defecto):

- Fondo principal: `bg-gray-900`
- Fondo secundario: `bg-gray-800`
- Bordes: `border-gray-600` / `border-gray-700`
- Texto principal: `text-white`
- Texto secundario: `text-gray-300` / `text-gray-400`
- Acento principal: `bg-red-600` / `text-red-500`

Componentes UI

Botones:

- Primario: `bg-red-600 hover:bg-red-700`
- Deshabilitado: `bg-gray-600`
- Estilo: `rounded-md` , `font-bold`

Inputs:

- Fondo: `bg-gray-800`
- Borde: `border-gray-600`
- Focus: `ring-red-500`

Tarjetas:

- Fondo: `bg-gray-800/50`
- Borde: `border-gray-700`
- Padding: `p-4` / `p-6`

📊 Estructura del Proyecto

```
Media-Enhancer/
├── components/
│   ├── AnalysisPanel.tsx          # Panel de análisis
│   ├── CreativeToolsPanel.tsx     # Panel de generación
│   ├── CustomVideoPlayer.tsx      # Reproductor de vídeo
│   ├── EnhancementPanel.tsx       # Panel de mejora
│   ├── FileUpload.tsx             # Carga de archivos
│   ├── Header.tsx                 # Cabecera
│   ├── HistorySidebar.tsx         # Historial
│   ├── ImageComparator.tsx        # Comparador de imágenes
│   ├── Loader.tsx                 # Loader animado
│   └── Tabs.tsx                   # Sistema de pestañas
├── services/
│   └── geminiService.ts           # Servicios de API Gemini
├── App.tsx                        # Componente principal
├── ErrorBoundary.tsx              # Manejo de errores
├── index.tsx                      # Punto de entrada
├── types.ts                       # Definiciones TypeScript
├── index.css                      # Estilos globales
├── tailwind.config.js             # Configuración Tailwind
├── vite.config.ts                 # Configuración Vite
├── tsconfig.json                  # Configuración TypeScript
├── package.json                   # Dependencias
└── README.md                      # Documentación básica
```

🔮 Futuras Mejoras

Funcionalidades Pendientes

- [ ] Edición de vídeo básica
- [ ] Extracción de clips de vídeo
- [ ] Generación de vídeo desde imágenes
- [ ] Mejora de vídeo (resolución, FPS)
- [ ] Traducción de audio/vídeo
- [ ] Exportación de resultados (PDF, JSON)
- [ ] Compartir resultados (enlaces)
- [ ] Modo colaborativo
- [ ] Procesamiento por lotes
- [ ] Integración con almacenamiento cloud

Mejoras Técnicas

- [ ] Sistema de caché para resultados
- [ ] Optimización de rendimiento
- [ ] Progressive Web App (PWA)
- [ ] Modo offline limitado
- [ ] Tests unitarios y e2e
- [ ] Documentación API completa
- [ ] Internacionalización (i18n)
- [ ] Tema claro/oscuro
- [ ] Modo de alto contraste (accesibilidad)

📞 Soporte y Recursos

Enlaces Útiles

- **Aplicación**: https://media-enhancer.vercel.app/
- **GitHub**: https://github.com/VCNPRO/Media-Enhancer
- **Vercel**: https://vercel.com/solammedia-9886s-projects/media-enhancer
- **AI Studio**: https://ai.studio/apps/drive/1ESq_gN8TKwn7XJE3oWIG9V_zR1d18PFs

Documentación Externa

- **Google Gemini API**: https://ai.google.dev/docs
- **Imagen 3**: https://cloud.google.com/vertex-ai/docs/generative-ai/image/overview
- **React**: https://react.dev
- **Vite**: https://vitejs.dev
- **Tailwind CSS**: https://tailwindcss.com

📜 Licencia y Créditos

Desarrollado por: VCNPRO

Tecnologías:

- Google Gemini 2.0
- Google Imagen 3
- React + TypeScript
- Vite
- Tailwind CSS

Prototipo creado con Google AI Studio

Documentación generada el 1 de noviembre de 2025