



API REST - Endoscope Application + Gemini AI

 FastAPI 0.109.0

 Python 3.8+

 Google Gemini AI

License MIT

Sistema completo de gestión de archivos con análisis avanzado de IA usando Google Gemini API



Tabla de Contenidos

- [Descripción](#)
- [Características](#)
- [Tecnologías](#)
- [Instalación](#)
- [Configuración](#)
- [Estructura del Proyecto](#)
- [Uso](#)
- [API Endpoints](#)
- [Ejemplos](#)
- [Documentación](#)
- [Contribución](#)
- [Licencia](#)



Descripción

API REST desarrollada con FastAPI que combina capacidades de procesamiento de archivos con análisis avanzado de inteligencia artificial usando Google Gemini. Diseñada específicamente para aplicaciones de segmentación endoscópica, pero extensible a múltiples casos de uso.

✦ Características Principales



Gestión de Archivos (10 tipos)

- **Imágenes** (JPG, PNG, GIF) - Análisis de dimensiones y formato
- **Videos** (MP4, AVI, MOV) - Validación y metadatos
- **Archivos ZIP** - Lista de contenido y ratios de compresión
- **Documentos PDF** - Validación de estructura
- **Archivos Excel** (XLSX, XLS) - Procesamiento de hojas de cálculo
- **Archivos CSV** - Análisis de columnas con preview
- **Documentos Word** (DOCX, DOC) - Validación de documentos
- **Audio** (MP3, WAV, OGG) - Procesamiento de archivos de audio
- **JSON** - Validación y análisis de estructura
- **XML** - Parseo y validación

Análisis con IA (Google Gemini)

Análisis de Texto:

- Análisis de texto con contexto
- Chat conversacional con historial
- Análisis de sentimientos (básico y detallado)
- Traducción multiidioma
- Resúmenes con diferentes longitudes
- Corrección gramatical y ortográfica

Análisis de Imágenes:

- Análisis de imágenes con Gemini Vision
- Comparación de múltiples imágenes
- Detección de objetos y descripción
- Análisis combinado (archivo + IA)

Procesamiento de Documentos:

- Extracción de información estructurada
- Análisis de documentos (resumen, puntos clave, entidades)
- Análisis de datos tabulares (CSV/Excel)
- Generación de embeddings para búsqueda semántica

Generación de Contenido:

- Contenido creativo (historias, poemas, artículos)
- Generación de código
- Respuestas con temperatura ajustable

Tecnologías

- **FastAPI** - Framework web moderno y rápido
- **Google Gemini API** - IA generativa de Google
- **Pillow** - Procesamiento de imágenes
- **Uvicorn** - Servidor ASGI de alto rendimiento
- **Python 3.8+** - Lenguaje de programación

Instalación

Prerrequisitos

- Python 3.8 o superior
- pip (gestor de paquetes de Python)
- Google API Key ([Obtener aquí](#))

Pasos de Instalación

1. Clonar el repositorio

```
git clone https://github.com/tu-usuario/tu-repositorio.git
cd tu-repositorio
```

2. Crear entorno virtual

```
# Windows
python -m venv venv
venv\Scripts\activate

# Linux/Mac
python3 -m venv venv
source venv/bin/activate
```

3. Instalar dependencias

```
pip install -r requirements.txt
```

 requirements.txt

```
fastapi==0.109.0
uvicorn[standard]==0.27.0
python-multipart==0.0.6
google-generativeai==0.3.2
pillow==10.2.0
python-dotenv==1.0.0
```

Configuración

1. Variables de Entorno

Crea un archivo `.env` en la raíz del proyecto:

```
# .env
GOOGLE_API_KEY=tu_api_key_de_google_aquí
```

2. Obtener API Key de Google

1. Visita [Google AI Studio](#)
2. Inicia sesión con tu cuenta de Google
3. Crea un nuevo proyecto (si es necesario)

4. Genera una nueva API key
5. Copia la clave en tu archivo `.env`

3. Configuración de CORS (Opcional)

En `main.py`, modifica los orígenes permitidos:

```
origins = [  
    "http://localhost:3000",    # Frontend en desarrollo  
    "https://tu-dominio.com",  # Producción  
    "*",                       # Permitir todos (solo desarrollo)  
]
```

Estructura del Proyecto




```
tu-proyecto/  
├── main.py                # Punto de entrada principal  
├── requirements.txt       # Dependencias del proyecto  
├── .env                  # Variables de entorno (no subir a git)  
├── .gitignore            # Archivos ignorados por git  
├── README.md             # Este archivo  
└── app/  
    ├── __init__.py  
    └── routes/  
        ├── __init__.py  
        ├── router.py      # Router de manejo de archivos  
        ├── gemini_api.py  # Cliente de Google Gemini  
        └── ai_routes.py   # Router de endpoints de IA
```

Uso

Iniciar el servidor

```
# Opción 1: Usando Python directamente  
python main.py  
  
# Opción 2: Usando Uvicorn  
uvicorn main:app --reload --host 0.0.0.0 --port 8000  
  
# Opción 3: Para producción (sin reload)  
uvicorn main:app --host 0.0.0.0 --port 8000 --workers 4
```

El servidor estará disponible en:

-  API: <http://localhost:8000>
-  Documentación Swagger: <http://localhost:8000/docs>
-  Documentación ReDoc: <http://localhost:8000/redoc>



API Endpoints



Información General

Método	Endpoint	Descripción
GET	/	Información general de la API
GET	/health	Estado de salud de los servicios
GET	/version	Versión actual de la API



Endpoints de Archivos

Método	Endpoint	Descripción
POST	/files/upload/image	Subir y analizar imagen
POST	/files/upload/video	Subir video
POST	/files/upload/zip	Subir y listar contenido ZIP
POST	/files/upload/pdf	Subir documento PDF
POST	/files/upload/excel	Subir archivo Excel
POST	/files/upload/csv	Subir y analizar CSV
POST	/files/upload/word	Subir documento Word
POST	/files/upload/audio	Subir archivo de audio
POST	/files/upload/json	Subir y validar JSON
POST	/files/upload/xml	Subir y validar XML
POST	/files/upload/multiple	Subir múltiples archivos (máx 10)
GET	/files/info	Información de tipos soportados



Endpoints de IA - Análisis de Texto

Método	Endpoint	Descripción
POST	/ai/analyze-text	Analizar texto con Gemini
POST	/ai/chat	Chat conversacional
GET	/ai/chat/history	Obtener historial de chat
DELETE	/ai/chat/history	Limpiar historial de chat

Método	Endpoint	Descripción
POST	/ai/sentiment	Análisis de sentimientos
POST	/ai/translate	Traducir texto
POST	/ai/summarize	Resumir texto
POST	/ai/grammar-check	Revisar gramática

Endpoints de IA - Análisis de Imágenes

Método	Endpoint	Descripción
POST	/ai/analyze-image	Analizar imagen con Vision
POST	/ai/compare-images	Comparar dos imágenes
POST	/ai/combined/image-analysis	Subir y analizar imagen

Endpoints de IA - Análisis de Documentos

Método	Endpoint	Descripción
POST	/ai/analyze-document	Analizar documento
POST	/ai/extract-structured-data	Extraer datos estructurados
POST	/ai/analyze-csv	Analizar datos CSV con IA
POST	/ai/combined/document-analysis	Subir y analizar documento

Endpoints de IA - Generación de Contenido

Método	Endpoint	Descripción
POST	/ai/generate-content	Generar contenido creativo
POST	/ai/embeddings	Generar embeddings
GET	/ai/models	Listar modelos disponibles
GET	/ai/status	Estado del servicio Gemini

Ejemplos

Ejemplo 1: Subir y analizar una imagen

Con cURL:

```
curl -X POST "http://localhost:8000/files/upload/image" \
  -F "file=@imagen.jpg"
```

Con Python:

```
import requests

url = "http://localhost:8000/files/upload/image"
files = {"file": open("imagen.jpg", "rb")}
response = requests.post(url, files=files)
print(response.json())
```

Respuesta:

```
{
  "success": true,
  "message": "Imagen procesada exitosamente",
  "data": {
    "filename": "imagen.jpg",
    "format": "JPEG",
    "width": 1920,
    "height": 1080,
    "size_mb": 2.45
  },
  "timestamp": "2024-01-15T10:30:00.000Z"
}
```

Ejemplo 2: Analizar imagen con IA

Con cURL:

```
curl -X POST "http://localhost:8000/ai/analyze-image" \
-F "file=@imagen.jpg" \
-F "prompt=Describe esta imagen médica en detalle"
```

Con Python:

```
import requests

url = "http://localhost:8000/ai/analyze-image"
files = {"file": open("endoscopia.jpg", "rb")}
data = {"prompt": "Identifica anomalías en esta imagen endoscópica"}
response = requests.post(url, files=files, data=data)
print(response.json())
```

Ejemplo 3: Chat conversacional

Con cURL:

```
# Primera pregunta
curl -X POST "http://localhost:8000/ai/chat" \
  -F "message=Hola, ¿puedes ayudarme con análisis de imágenes médicas?"

# Segunda pregunta (mantiene contexto)
curl -X POST "http://localhost:8000/ai/chat" \
  -F "message=¿Qué tipo de anomalías puedes detectar?"
```

Con Python:

```
import requests

url = "http://localhost:8000/ai/chat"

# Primera pregunta
response1 = requests.post(url, data={
    "message": "¿Qué es la segmentación de imágenes médicas?"
})
print(response1.json()["response"])

# Segunda pregunta (mantiene contexto)
response2 = requests.post(url, data={
    "message": "¿Puedes darme un ejemplo práctico?"
})
print(response2.json()["response"])
```

Ejemplo 4: Análisis de documento CSV

Con cURL:

```
curl -X POST "http://localhost:8000/ai/analyze-csv" \
  -F "file=@datos.csv" \
  -F "question=¿Cuáles son las tendencias principales en estos datos?"
```

Con Python:

```
import requests

url = "http://localhost:8000/ai/analyze-csv"
files = {"file": open("resultados.csv", "rb")}
data = {"question": "Genera un resumen estadístico de estos datos"}
response = requests.post(url, files=files, data=data)
print(response.json())
```


Ejemplo 5: Extraer información estructurada

Con Python:

```
import requests
import json

url = "http://localhost:8000/ai/extract-structured-data"

# Definir schema
schema = {
    "paciente": "Nombre del paciente",
    "edad": "Edad en años",
    "diagnostico": "Diagnóstico principal",
    "tratamiento": "Tratamiento recomendado"
}

files = {"file": open("informe_medico.txt", "rb")}
data = {"schema": json.dumps(schema)}
response = requests.post(url, files=files, data=data)
print(response.json())
```

Ejemplo 6: Traducir texto

Con cURL:

```
curl -X POST "http://localhost:8000/ai/translate" \
-F "text=The medical imaging shows normal results" \
-F "target_language=español"
```

Ejemplo 7: Generar resumen

Con cURL:

```
curl -X POST "http://localhost:8000/ai/summarize" \
-F "text=Tu texto largo aquí..." \
-F "summary_length=short" \
-F "bullet_points=true"
```

Ejemplo 8: Subir múltiples archivos

Con Python:

```
import requests

url = "http://localhost:8000/files/upload/multiple"
files = [
    ("files", open("imagen1.jpg", "rb")),
    ("files", open("imagen2.jpg", "rb")),
    ("files", open("documento.pdf", "rb"))
]
response = requests.post(url, files=files)
print(response.json())
```

Documentación

Documentación Interactiva

Una vez que el servidor esté en ejecución, puedes acceder a la documentación interactiva:

- **Swagger UI:** <http://localhost:8000/docs>
 - Interfaz interactiva para probar todos los endpoints
 - Documentación generada automáticamente
 - Posibilidad de ejecutar requests directamente
- **ReDoc:** <http://localhost:8000/redoc>
 - Documentación alternativa más detallada
 - Mejor para lectura y referencia

Modelos de Gemini Disponibles

La API utiliza dos modelos de Google Gemini:

- **gemini-1.5-flash-latest** (por defecto)
 - Rápido y eficiente
 - Ideal para la mayoría de tareas
 - Menor latencia
- **gemini-1.5-pro-latest**
 - Mayor capacidad de razonamiento
 - Mejor para tareas complejas
 - Soporta análisis multimodal

Configuración de Temperatura

La temperatura controla la creatividad de las respuestas:

- **0.0 - 0.3:** Respuestas más deterministas y consistentes
- **0.4 - 0.7:** Balance entre creatividad y coherencia (recomendado)

- 0.8 - 1.0: Respuestas más creativas y variadas

Seguridad

Mejores Prácticas

1. Nunca subas tu archivo `.env` a repositorios públicos

```
# Agrega a .gitignore
.env
venv/
__pycache__/
*.pyc
```

2. Usa variables de entorno en producción

```
export GOOGLE_API_KEY="tu_api_key"
```

3. Configura CORS adecuadamente

- En desarrollo: Permite todos los orígenes (*)
- En producción: Especifica dominios exactos

4. Limita el tamaño de archivos

- Los límites están configurados en `router.py`
- Ajústalos según tus necesidades

5. Implementa rate limiting (recomendado para producción)

```
pip install slowapi
```

Solución de Problemas

Error: "No module named 'google.generativeai'"

Solución:

```
pip install google-generativeai
```

Error: "Servicio de IA no disponible"

Causas posibles:

1. API key no configurada en `.env`
2. API key inválida
3. Problemas de conectividad

Solución:

```
# Verifica que el archivo .env existe
cat .env

# Verifica que la variable está cargada
python -c "import os; from dotenv import load_dotenv; load_dotenv();
print(os.getenv('GOOGLE_API_KEY'))"
```

Error al subir archivos grandes

Solución:

Ajusta los límites en `router.py`:

```
MAX_FILE_SIZE = {
    "image": 20 * 1024 * 1024, # Aumentar a 20 MB
    # ...
}
```

Puerto 8000 ya en uso

Solución:

```
# Usar otro puerto
uvicorn main:app --port 8001

# O matar el proceso en el puerto 8000
# Windows
netstat -ano | findstr :8000
taskkill /PID <PID> /F

# Linux/Mac
lsof -ti:8000 | xargs kill -9
```

Despliegue

Docker (Recomendado)

Dockerfile:

```
FROM python:3.10.0

RUN mkdir /app

WORKDIR /app

RUN apt-get update && \
    apt-get install -y --no-install-recommends \
    libgl1-mesa-glx && \
    rm -rf /var/lib/apt/lists/*

COPY requirements_deploy.txt .
RUN pip install --no-cache-dir -r requirements_deploy.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80", "--reload"]
```

docker-compose.yml:

```
version: '3.8'

services:
  api:
    build: .
    ports:
      - "8000:8000"
    environment:
      - GOOGLE_API_KEY=${GOOGLE_API_KEY}
    volumes:
      - ./app:/app/app
    restart: unless-stopped
```

Ejecutar:

```
docker-compose up -d
```

Heroku

```
# Instalar Heroku CLI
# Crear Procfile
echo "web: uvicorn main:app --host 0.0.0.0 --port \$PORT" > Procfile

# Deploy
heroku create tu-app
```

```
heroku config:set GOOGLE_API_KEY=tu_api_key
git push heroku main
```

Railway / Render

1. Conecta tu repositorio de GitHub
2. Configura la variable de entorno `GOOGLE_API_KEY`
3. El servicio detectará automáticamente FastAPI

Contribución

¡Las contribuciones son bienvenidas! Por favor sigue estos pasos:

1. Fork el proyecto
2. Crea una rama para tu feature (`git checkout -b feature/AmazingFeature`)
3. Commit tus cambios (`git commit -m 'Add some AmazingFeature'`)
4. Push a la rama (`git push origin feature/AmazingFeature`)
5. Abre un Pull Request

Estándares de Código

- Sigue PEP 8 para Python
- Documenta todas las funciones y clases
- Agrega pruebas para nuevas funcionalidades
- Actualiza el README si es necesario

Licencia

Este proyecto está bajo la Licencia MIT. Ver el archivo `LICENSE` para más detalles.

Autores

- **Tu Nombre** - *Desarrollo Inicial* - [TuGitHub](#)

Agradecimientos

- [FastAPI](#) por el increíble framework
- [Google](#) por la API de Gemini
- La comunidad open source

Contacto

- Email: rfloresa1900@alumno.ipn.mx
- GitHub: [@RodrigoFA216](#)
- LinkedIn: [Rodrigo Flores](#)

Roadmap

- ☐ Implementar autenticación JWT
- ☐ Agregar rate limiting
- ☐ Sistema de caché para respuestas frecuentes
- ☐ Soporte para más tipos de archivos
- ☐ Procesamiento asíncrono de archivos grandes
- ☐ Dashboard de monitoreo
- ☐ Tests unitarios y de integración
- ☐ CI/CD con GitHub Actions

★ Si este proyecto te fue útil, considera darle una estrella en GitHub!

Desarrollado con ❤️ usando FastAPI y Google Gemini