# Sistema de Storyboard Cinematográfico Automatizado

## Objetivo

Transformar cualquier texto en un video profesional y viral mediante un proceso automatizado que genere storyboard, recursos visuales y audio sincronizado.

## Configuraciones y Parámetros

### Parámetros de Calidad

QUALITY\_THRESHOLDS = {

"clip\_score\_hook": 0.85, # Score mínimo para hook

"clip\_score\_key\_point": 0.80, # Score mínimo para puntos clave

"clip\_score\_normal": 0.70, # Score mínimo para escenas normales

"emotional\_intensity\_high": 0.7, # Umbral para ElevenLabs

"max\_scene\_duration": 6.0, # Duración máxima por escena

"min\_scene\_duration": 2.0 # Duración mínima por escena

}

### Configuración de Búsqueda

SEARCH\_CONFIG = {

"max\_results\_per\_source": 15,

"timeout\_per\_search": 30, # segundos

"fallback\_attempts": 3,

"cache\_duration\_days": 7,

"priority\_sources": {

"generic": ["pexels", "wikimedia", "google"],

"historical": ["wikimedia", "google", "pexels"],

"current": ["google", "pexels", "wikimedia"]

}

}

### Templates de Visual Focus

VISUAL\_FOCUS\_TEMPLATES = {

"banner\_de\_impacto": "Elegant typography with '{text}' in cinematic lighting, minimalist composition, {mood} atmosphere",

"b\_roll\_dinamico": "Dynamic {action} scene, cinematic camera movement, professional lighting, {mood} color grading",

"rafagas\_imagenes": "Quick montage of {concept} imagery, fast-paced editing style, vibrant colors",

"plano\_detalle": "Extreme close-up of {subject}, shallow depth of field, dramatic lighting, emotional intensity"

}

### Lexicón Emocional

EMOTIONAL\_LEXICON = {

"high\_intensity": [

"impresionante", "espectacular", "increíble", "asombroso", "magnifico",

"terrible", "devastador", "brutal", "intenso", "dramático"

],

"medium\_intensity": [

"interesante", "notable", "considerable", "significativo", "importante"

],

"action\_words": [

"correr", "saltar", "explotar", "caer", "volar", "luchar", "perseguir"

]

}

## Flujos de Error y Logging

### Manejo de Errores por Módulo

#### TextAnalyzer

* **Error**: spaCy model no encontrado → Fallback a clasificación básica
* **Error**: Wikipedia API timeout → Continuar sin entidades
* **Log**: Nivel de confianza de clasificaciones

#### VisualResourceManager

* **Error**: API rate limit → Cambiar a siguiente fuente
* **Error**: CLIP model load failed → Usar scoring básico por keywords
* **Error**: Todas las búsquedas fallan → Generar recurso con código
* **Log**: Número de resultados por fuente, scores CLIP

#### AudioVisualSynchronizer

* **Error**: TTS service down → Intentar servicio alternativo
* **Error**: Timestamps malformados → Calcular timing estimado
* **Log**: Duración real vs estimada por escena

### Sistema de Logging

LOGGING\_CONFIG = {

"level": "INFO",

"format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s",

"handlers": {

"file": "storyboard\_generator.log",

"console": True

},

"modules": {

"text\_analyzer": "DEBUG",

"visual\_resource\_manager": "INFO",

"storyboard\_generator": "INFO"

}

}

## Arquitectura del Sistema

### Módulos Principales

#### 1. TextAnalyzer

# Módulo: text\_analyzer.py

class TextAnalyzer:

def analyze\_text(self, texto: str) -> AnalysisResult

def classify\_emotional\_intensity(self, frase: str) -> float

def detect\_narrative\_type(self, frase: str) -> NarrativeType

def extract\_entities(self, frase: str) -> List[Entity]

#### 2. VisualResourceManager

# Módulo: visual\_resource\_manager.py

class VisualResourceManager:

def generate\_visual\_focus(self, frase: str, context: Context) -> str

def search\_resources(self, visual\_focus: str) -> List[Resource]

def filter\_and\_rank(self, resources: List[Resource], criteria: Criteria) -> RankedResources

#### 3. StoryboardGenerator

# Módulo: storyboard\_generator.py

class StoryboardGenerator:

def create\_storyboard(self, analysis: AnalysisResult) -> Storyboard

def optimize\_for\_viral(self, storyboard: Storyboard) -> OptimizedStoryboard

def generate\_specifications(self, scene: Scene) -> SceneSpecs

#### 4. AudioVisualSynchronizer

# Módulo: av\_synchronizer.py

class AudioVisualSynchronizer:

def sync\_audio\_visual(self, audio\_data: AudioData, visual\_data: VisualData) -> SyncResult

def calculate\_timing(self, scene\_duration: float) -> TimingSpecs

#### 5. MotionGraphicsEngine

# Módulo: motion\_engine.py

class MotionGraphicsEngine:

def generate\_banner(self, text: str, style: BannerStyle) -> BannerAsset

def create\_particles(self, config: ParticleConfig) -> ParticleAsset

def apply\_ken\_burns(self, image: Image, duration: float) -> AnimatedAsset

### Estructuras de Datos

#### AnalysisResult

@dataclass

class AnalysisResult:

frases: List[str]

emocional\_intensity: List[float]

narrative\_types: List[NarrativeType]

entities: List[List[Entity]]

viral\_markers: ViralMarkers

#### Entity

@dataclass

class Entity:

texto: str

tipo: str # PERSON, LOCATION, WORK\_OF\_ART, etc.

timestamp\_inicio: float

timestamp\_fin: float

wikipedia\_url: Optional[str]

imagen\_url: Optional[str]

confianza: float

#### Scene

@dataclass

class Scene:

id: int

frase: str

tipo\_recurso: ResourceType

duracion: float

intensidad\_emocional: float

tipo\_narrativo: NarrativeType

audio\_data: AudioData

recursos\_visuales: VisualResources

entidades: List[Entity]

especificaciones: SceneSpecs

metricas\_calidad: QualityMetrics

## Plan de Desarrollo - Paso a Paso

### ETAPA 1: FUNDAMENTOS (Semana 1-2)

#### 1.1 Análisis Textual Básico

* **Tarea**: Integrar clasificación emocional por frase
* **Descripción**: Crear lexicón de palabras emocionales y algoritmo de scoring
* **Output**: Función que recibe frase y retorna intensidad (0-1)

#### 1.2 Reconocimiento de Entidades

* **Tarea**: Implementar NER con spaCy + Wikipedia API
* **Descripción**: Detectar personas, lugares, obras, eventos y extraer imágenes
* **Output**: Lista de entidades con timestamps y URLs de imágenes

#### 1.3 Sistema de Clasificación de Recursos

* **Tarea**: Algoritmo para mapear frase → tipo de recurso visual
* **Descripción**: Reglas heurísticas basadas en tipo narrativo y contenido
* **Output**: Función clasificadora (banner, b-roll, ráfagas, etc.)

### ETAPA 2: GENERACIÓN DE CONTENIDO VISUAL (Semana 3-4)

#### 2.1 Generador de Visual Focus

* **Tarea**: Sistema que convierte frase en descripción cinematográfica
* **Descripción**: Templates por tipo de recurso + análisis de contexto
* **Output**: Visual Focus específico + Negative Prompts por frase

#### 2.2 Buscador Multi-fuente Inteligente

* **Tarea**: Orquestador de búsquedas con priorización automática
* **Descripción**: Decidir qué buscador usar según tipo de contenido
* **Output**: URLs de imágenes ranqueadas por fuente

#### 2.3 Sistema de Fallbacks

* **Tarea**: Manejo de errores en búsqueda con 3 niveles
* **Descripción**: Re-intentos con diferentes estrategias
* **Output**: Siempre retorna un recurso válido (aunque sea generado)

### ETAPA 3: PROCESAMIENTO DE AUDIO (Semana 2-3)

#### 3.1 Orquestador TTS Inteligente

* **Tarea**: Selección automática CoquiTTS vs ElevenLabs por frase
* **Descripción**: Integrar con clasificación emocional
* **Output**: Audio + timestamps sincronizados por motor óptimo

#### 3.2 Sincronización Audio-Visual

* **Tarea**: Alinear recursos visuales con duración de audio
* **Descripción**: Calcular timing de transiciones y movimientos
* **Output**: Timeline completa con sincronización precisa

### ETAPA 4: STORYBOARD INTELIGENTE (Semana 4-5)

#### 4.1 Analizador de Estructura Viral

* **Tarea**: Detectar Hook, puntos clave y cierre en el texto
* **Descripción**: Identificar momentos de máximo impacto emocional
* **Output**: Marcas de timing para elementos especiales

#### 4.2 Generador de Especificaciones Cinematográficas

* **Tarea**: Crear specs técnicas por escena (movimientos, timing, efectos)
* **Descripción**: Reglas basadas en tipo de contenido y posición en narrativa
* **Output**: JSON completo con todas las especificaciones

#### 4.3 Sistema de Métricas de Calidad

* **Tarea**: Scoring automático de potencial viral por escena
* **Descripción**: Evaluar recursos según criterios de Hook/Puntos clave/Cierre
* **Output**: Score de calidad y sugerencias de mejora

### ETAPA 5: GENERACIÓN DE RECURSOS VISUALES (Semana 5-6)

#### 5.1 Motor de Motion Graphics

* **Tarea**: Generador de animaciones CSS/JavaScript
* **Descripción**: Partículas, transiciones, iconografía animada
* **Output**: Assets animados listos para usar

#### 5.2 Sistema de Movimientos Cinematográficos

* **Tarea**: Implementar Ken Burns, Parallax, Ráfagas
* **Descripción**: Código reutilizable para efectos visuales
* **Output**: Librerías de movimientos parameterizables

#### 5.3 Generador de Banners Dinámicos

* **Tarea**: Crear banners de texto con tipografía cinética
* **Descripción**: Templates animados estilo Legend App
* **Output**: Banners personalizados por contenido

### ETAPA 6: OPTIMIZACIÓN Y CACHÉ (Semana 6-7)

#### 6.1 Sistema de Caché Inteligente

* **Tarea**: Almacenar resultados de búsquedas y CLIP scores
* **Descripción**: Hash de Visual Focus para evitar re-procesar
* **Output**: Base de datos local de recursos pre-evaluados

#### 6.2 Presets por Tipo de Contenido

* **Tarea**: Configuraciones optimizadas por dominio
* **Descripción**: Histórico, científico, filosófico con parámetros específicos
* **Output**: Sistema de templates intercambiables

#### 6.3 Adaptación Multi-formato

* **Tarea**: Generador de versiones horizontal y vertical
* **Descripción**: Recorte inteligente manteniendo puntos focales
* **Output**: JSON dual con specs para ambos formatos

### ETAPA 7: INTEGRACIÓN Y TESTING (Semana 7-8)

#### 7.1 Pipeline Completo

* **Tarea**: Conectar todos los módulos en flujo secuencial
* **Descripción**: Orquestador principal que ejecuta todo el proceso
* **Output**: Sistema end-to-end funcional

#### 7.2 Validación de Calidad

* **Tarea**: Testing con diferentes tipos de texto
* **Descripción**: Probar con contenido histórico, científico, narrativo
* **Output**: Métricas de éxito y casos de error identificados

#### 7.3 Optimización de Performance

* **Tarea**: Paralelización y optimización de velocidad
* **Descripción**: Procesar múltiples frases simultáneamente
* **Output**: Sistema optimizado para producción

### ETAPA 8: REFINAMIENTO (Semana 8-9)

#### 8.1 Ajuste de Algoritmos

* **Tarea**: Mejorar precisión basada en testing
* **Descripción**: Refinar clasificadores y umbrales de calidad
* **Output**: Algoritmos calibrados

#### 8.2 Expansión de Recursos

* **Tarea**: Agregar más fuentes de búsqueda si es necesario
* **Descripción**: APIs adicionales gratuitas para mayor variedad
* **Output**: Mayor pool de recursos disponibles

#### 8.3 Documentación Final

* **Tarea**: Documentar configuraciones y uso del sistema
* **Descripción**: Manual de uso y troubleshooting
* **Output**: Sistema listo para producción

## Cronograma Estimado

**Total**: 8-9 semanas de desarrollo **Dependencias críticas**: Etapas 1-2 deben completarse antes de 4-5 **Paralelización posible**: Etapa 3 puede desarrollarse en paralelo con 2 **Hitos clave**:

* Semana 4: Pipeline básico funcional
* Semana 6: Sistema completo con optimizaciones
* Semana 8: Producto terminado y testeado

## Recursos Existentes

### 1. SentenceSplitterService (sentence\_splitter\_service.py)

* **Función**: Divide textos largos en frases individuales en español
* **Método**: Expresiones regulares con manejo de puntuación, mayúsculas y abreviaturas
* **Limitación**: Solo división heurística, no semántica
* **Interface esperada**: split\_text(texto: str) -> List[str]

### 2. TTSService (tts\_orchestrator.py)

* **Función**: Orquesta síntesis de voz con Piper y Coqui
* **Características**: Inicialización, caché, alineación audio-texto
* **Interface esperada**: generate\_audio(texto: str) -> AudioResult

### 3. ElevenLabsTTSService (tts\_service\_elevenlabs.py)

* **Función**: Síntesis con ElevenLabs API
* **Output**:
  + Archivo audio
  + Timestamps de palabras y letras
  + Caché optimizado
* **Interface esperada**: synthesize(texto: str) -> ElevenLabsResult

### 4. filter\_images.py

* **Función**: Filtra imágenes usando CLIP
* **Input**: Descripciones positivas/negativas + URLs de imágenes
* **Output**: JSON con estados (valid, censored, no\_match) y scores
* **Interface esperada**: filter\_images(positive\_prompt: str, negative\_prompt: str, image\_urls: List[str]) -> FilterResult

### 5. Búsqueda de Recursos (Módulos existentes)

* **Fuentes disponibles**: Pexels, Google Images, Wikimedia Commons
* **Tipos**: Fotos y videos
* **Limitaciones**: APIs gratuitas
* **Interface esperada**: search\_images(query: str, source: str) -> List[ImageResult]

## Limitaciones Técnicas

* **Hardware**: Solo CPU, 16GB RAM
* **Presupuesto**: Recursos gratuitos preferentemente
* **Velocidad**: Optimizar para procesamiento rápido
* **Dependencias**: Python 3.8+, spaCy, requests, Pillow
* **Almacenamiento**: Sistema de archivos local para caché e imágenes
* **Concurrencia**: Máximo 3-4 procesos paralelos por limitaciones de RAM

## Flujo del Sistema

### FASE 1: ANÁLISIS TEXTUAL

#### Paso 1.1: Segmentación

Input: Texto completo del artículo

↓

SentenceSplitterService.split\_text()

↓

Output: Array de frases individuales

#### Paso 1.2: Clasificación por Frase

**Intensidad Emocional** (para selección TTS):

* **Método**: Lexicón de palabras emocionales + conteo de:
  + Adjetivos intensos
  + Signos de exclamación
  + Palabras de acción
* **Asignación TTS**:
  + **CoquiTTS**: Frases neutras/informativas
  + **ElevenLabs**: Frases con carga emocional

**Tipo Narrativo** (para selección de recurso):

* **Descriptivo**: Imágenes estáticas con movimiento sutil
* **Acción**: Videos/B-rolls dinámicos
* **Reflexivo**: Banners con tipografía impactante

#### Paso 1.3: Reconocimiento de Entidades (NER)

**Detección de Entidades con Wikipedia**:

* **Herramienta**: spaCy + Wikipedia API (gratuita, local)
* **Entidades a detectar**:
  + Personas famosas
  + Lugares/landmarks
  + Eventos históricos
  + Obras de arte
  + Marcas conocidas
  + Películas, libros, etc.

**Proceso**:

Frase → spaCy NER → Entidades detectadas → Wikipedia API →

Verificar existencia página → Extraer imagen principal →

Sincronizar aparición con timestamp de pronunciación

**Output adicional por frase**:

"entidades": [

{

"texto": "Mona Lisa",

"tipo": "WORK\_OF\_ART",

"timestamp\_inicio": "2.3s",

"timestamp\_fin": "2.8s",

"wikipedia\_url": "https://es.wikipedia.org/wiki/La\_Gioconda",

"imagen\_url": "imagen\_monalisa.jpg",

"confianza": 0.95

}

]

### FASE 2: OPTIMIZACIÓN VIRAL

#### Paso 2.1: Identificar Hook Visual

Determinar tipo de primera imagen:

* **Pregunta visual intrigante**
* **Contraste dramático** (oscuro→claro)
* **Elemento inesperado** que rompa expectativas

#### Paso 2.2: Estructura Narrativa (3 Actos)

* **Acto I (25%)**: Setup + Hook
* **Acto II (50%)**: Desarrollo + Builds emocionales
* **Acto III (25%)**: Clímax + Resolución
* **Micro-clímax**: Cada 8-12 segundos para retención

### FASE 3: STORYBOARD INTELIGENTE

#### Paso 3.1: Mapeo Frase → Recurso Visual

**Banner de Impacto**:

* **Criterio**: Frases quotables, conceptos centrales
* **Características**: Máximo 5-7 palabras, tipografía cinética, animaciones
* **Cuándo**: Mensajes clave, citas importantes

**Ráfagas de Imágenes**:

* **Criterio**: Listas, enumeraciones
* **Implementación**: Una imagen por palabra
* **Ritmo**: 2-3 segundos total

**B-roll Dinámico**:

* **Criterio**: Acciones, movimiento, contexto
* **Tipos**: Videos o animaciones desde imágenes
* **Duración**: 3-5 segundos

**Motion Graphics**:

* **Criterio**: Conceptos abstractos
* **Tipos disponibles**:
  + **Partículas**: CSS animations + JavaScript (puntos flotantes, estrellas)
  + **Transiciones**: Wipes, fades, geometric shapes
  + **Iconografía animada**: SVG + CSS (iconos que se dibujan)
  + **Texto cinético**: Palabras que aparecen letra por letra
* **Generación**: CSS + JavaScript + Canvas básico
* **Duración**: 3-4 segundos

**Plano Detalle**:

* **Criterio**: Emociones intensas, elementos específicos
* **Tipo**: Fotos macro/closeup (manos, ojos, texturas)
* **Diferencia con B-roll**: Estático vs dinámico

#### Paso 3.2: Generación de Prompts

Para cada frase crear:

* **Visual Focus**: Descripción cinematográfica específica completa
* **Negative Prompt**: Elementos a evitar + censura contenido sensible
* **Keywords**: Solo como fallback (3-5 términos)
* **Duración**: Basada en audio + tipo de recurso

**Subdivisión de Frases Largas**:

* **Máximo**: 10-12 palabras por escena
* **Cada sub-escena**: Su propio visual focus, negative prompt, etc.

### FASE 4: GENERACIÓN DE AUDIO

#### Paso 4.1: Síntesis por Lotes

Frases neutras → TTSService (CoquiTTS)

Frases emocionales → ElevenLabsTTSService

**Output por frase**:

* Archivo audio (.wav)
* JSON con timestamps de palabras
* Duración total

### FASE 5: BÚSQUEDA Y FILTRADO DE RECURSOS

#### Paso 5.1: Búsqueda Multi-fuente (Priorizada)

**Estrategia de Priorización**:

1. **Banner de impacto** → No buscar (generar con código)
2. **Contenido genérico** → Pexels primero
3. **Contenido histórico/cultural** → Wikimedia primero
4. **Contenido específico/actual** → Google Images primero
5. **Si falla todo** → Probar otros buscadores

**Proceso**:

Visual Focus → Buscador priorizado → Descargar 10-15 opciones

#### Paso 5.2: Filtrado Inteligente con Fallbacks

**Sistema de Fallbacks**:

Intento 1: Visual Focus completo + CLIP

↓ (Si no encuentra válidas)

Intento 2: Pregunta semántica "¿Esta imagen representa: [frase]?"

↓ (Si falla)

Intento 3: Regenerar Visual Focus más genérico

↓ (Si falla)

Fallback Final: Banner de texto animado + fondo generado

→ Marcar en JSON como "fallback\_usado": true

**Proceso CLIP**:

Input: Visual Focus + Negative Prompt + URLs

↓

filter\_images.py

↓

Output: JSON ranking (valid, no\_match, censored, scores)

### FASE 6: SELECCIÓN FINAL Y OPTIMIZACIÓN

#### Paso 6.1: Selección por Escena

* **Top 3 imágenes válidas** de CLIP
* **Verificar duración** vs duración audio
* **Videos**: Recortar según duración necesaria
* **Fotos**: Aplicar movimiento cinematográfico

#### Paso 6.2: Movimientos Cinematográficos (Detalles Técnicos)

**Ken Burns** (para fotos estáticas):

/\* Implementación básica \*/

.ken-burns {

animation: kenBurns 4s ease-in-out;

transform-origin: center;

}

@keyframes kenBurns {

0% { transform: scale(1.0) translate(0, 0); }

100% { transform: scale(1.1) translate(-2%, -1%); }

}

**Parallax** (para paisajes):

// Separar elementos en capas

foreground.style.transform = `translateX(${offset \* 0.5}px)`;

background.style.transform = `translateX(${offset \* 0.1}px)`;

**Ráfagas de Imágenes**:

.rafaga-item {

animation: fadeInScale 0.3s ease-out;

animation-delay: calc(var(--index) \* 0.1s);

}

@keyframes fadeInScale {

0% { opacity: 0; transform: scale(0.8); }

100% { opacity: 1; transform: scale(1.0); }

}

**Partículas Flotantes**:

// Canvas básico para partículas

function createParticle() {

return {

x: Math.random() \* canvas.width,

y: Math.random() \* canvas.height,

vx: (Math.random() - 0.5) \* 2,

vy: (Math.random() - 0.5) \* 2,

opacity: Math.random()

};

}

#### Paso 6.3: Adaptación de Formato

**Si usuario eligió múltiples formatos**:

* **Detectar orientación** original de recursos
* **Recortar/redimensionar** para 16:9 y 9:16
* **Mantener puntos focales** en ambas versiones

## Output Final: JSON Estructurado

{

"metadata": {

"duracion\_total": "45s",

"formato": ["16:9", "9:16"],

"hook\_type": "pregunta\_intrigante",

"actos": {

"acto\_1": {"inicio": "0s", "fin": "11s"},

"acto\_2": {"inicio": "11s", "fin": "34s"},

"acto\_3": {"inicio": "34s", "fin": "45s"}

}

},

"escenas": [

{

"id": 1,

"frase": "Texto de la frase",

"tipo\_recurso": "banner\_de\_impacto",

"duracion": "4.2s",

"intensidad\_emocional": "alta",

"tipo\_narrativo": "reflexivo",

"audio": {

"archivo": "frase\_001.wav",

"motor": "elevenlabs",

"timestamps": {

"palabras": [...],

"letras": [...]

}

},

"recursos\_visuales": {

"principal": {

"url": "imagen\_seleccionada.jpg",

"score": 0.89,

"tipo": "foto",

"fuente": "pexels"

},

"alternativas": [

{

"url": "imagen\_alt1.jpg",

"score": 0.76,

"tipo": "foto"

}

]

},

"visual\_focus": "Descripción cinematográfica específica",

"negative\_prompt": "Elementos a evitar",

"especificaciones": {

"movimiento\_camara": "ken\_burns",

"duracion\_fade": "0.5s",

"posicion\_texto": "lower\_third",

"velocidad\_zoom": "lento"

},

"fallbacks\_intentados": 0,

"fallback\_usado": false,

"metricas\_calidad": {

"es\_hook": false,

"es\_punto\_clave": true,

"es\_cierre": false,

"score\_viral\_potencial": 0.78

}

}

]

}

## Métricas de Calidad y Viralidad

### Características de Recursos para Hook (0-3 segundos)

**Criterios especiales**:

* **Contraste visual extremo**: Colores saturados vs desaturados
* **Elementos inesperados**: Lo que no esperas ver
* **Composición asimétrica**: Regla de tercios rota intencionalmente
* **Movimiento contra-intuitivo**: Zoom out en vez de zoom in
* **Score mínimo CLIP**: 0.85+ obligatorio

### Características de Recursos para Puntos Clave

**Identificación de puntos clave**:

* Frases con intensidad emocional máxima
* Conceptos centrales del texto
* Revelaciones o momentos "aha"

**Criterios especiales**:

* **Lighting dramático**: Golden hour, contraluz, spot lighting
* **Composición cinematográfica**: Leading lines, framing natural
* **Colorimetría específica**: Teal & orange, monochrome selectivo
* **Score CLIP**: 0.80+ requerido

### Características de Recursos para Cierre (últimos 5 segundos)

**Criterios especiales**:

* **Sensación de completitud**: Círculo que se cierra
* **Mensaje universal**: Que trascienda el tema específico
* **Elemento memorable**: Algo que quede en la mente
* **Call-to-action visual implícito**: Invita a compartir

## Optimizaciones del Sistema

### 1. Caché de Búsquedas

{

"cache\_visual\_focus": {

"hash\_descripcion": "md5\_del\_visual\_focus",

"resultados": [

{

"url": "imagen.jpg",

"score": 0.89,

"timestamp": "2024-08-27",

"fuente": "pexels"

}

]

}

}

### 2. Presets por Tipo de Contenido

**Preset Histórico**:

* **Visual Focus tendencia**: Sepia tones, vintage look, period accurate
* **Fuentes prioritarias**: Wikimedia > Google Images > Pexels
* **Negative prompts**: Modern elements, digital screens, contemporary clothing

**Preset Científico**:

* **Visual Focus tendencia**: Clean, minimal, high contrast, laboratory aesthetic
* **Fuentes prioritarias**: Pexels > Wikimedia > Google Images
* **Motion graphics**: Diagramas animados, iconografía técnica

**Preset Filosófico/Reflexivo**:

* **Visual Focus tendencia**: Abstract, metaphorical, symbolic imagery
* **Banners**: Mayor frecuencia de uso
* **Movimientos**: Ken Burns lentos, transiciones suaves

### 3. Análisis de Tendencias (Explicación)

**No implementar inicialmente**, pero la idea sería:

* Analizar qué Visual Focus generan mejores scores CLIP
* Identificar patterns en recursos que funcionan mejor
* Ajustar keywords automáticamente según éxito histórico
* **Ejemplo**: Si "golden hour portrait" siempre da +0.85 CLIP, priorizarlo