



UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES



Departamento de Desarrollo de Software

Desarrollo de Software 5

Integrantes:

Alberto Delgado

8-887-2206

Erick Asprilla

Grupo:

1LS-232

Proyecto Final

Editor GIFs

Profesor:

Eduardo Griffith

INDICE

Table of Contents

INDICE.....	2
Introducción.....	3
Interfaz Gráfica.....	4
Carpetas contenidas	4
Codigo en JAVA.....	5
Codigo en html en CSS.....	9
Codigo en HTML	16
Codigo De PY	20
Conclusión.....	27

Introducción

Esta aplicación web permite a los usuarios subir archivos de video e imágenes, aplicar efectos visuales, y convertirlos en GIFs u otros formatos de video como MP4 o WebM. La aplicación está construida con Flask, una microframework de Python para desarrollo web, y utiliza varias bibliotecas para el procesamiento de imágenes y videos como OpenCV, MoviePy, y Numpy.

Funcionalidades Principales

Subida de Archivos: Los usuarios pueden subir archivos de video (en formatos como MP4, AVI, MOV) e imágenes (en formatos como JPG, JPEG, PNG, BMP). La interfaz de usuario permite la selección múltiple de archivos para su procesamiento.

Aplicación de Efectos: Una variedad de efectos visuales pueden ser aplicados a los videos o imágenes subidos, incluyendo: Escala de grises Sepia Inversión de colores Desenfoque Aumento de contraste Reducción de ruido Detección de bordes Mosaico Distorsión de onda Efecto de espejo (horizontal y vertical) Rotación Efecto de reflejo de agua Recorte de Videos:

Los usuarios pueden seleccionar el inicio y el fin del segmento de video que desean convertir y editar.

Conversión y Exportación: Los archivos procesados pueden ser exportados en diferentes formatos como GIF, MP4, y WebM, permitiendo una alta flexibilidad en el uso del contenido generado.

Interfaz Gráfica





Carpetas contenidas

Contenido del proyecto

Name	Date modified	Type	Size
static	7/10/2024 7:53 AM	File folder	
templates	5/25/2024 1:50 PM	File folder	
upload	7/9/2024 9:58 AM	File folder	
app.py	7/10/2024 7:52 AM	Python Source File	14 KB


Carpeta Static

Tenemos el diseño del proyecto

Name	Date modified	Type	Size
 script.js	7/9/2024 9:24 AM	JavaScript Source ...	5 KB
 styles.css	7/1/2024 9:32 PM	CSS Source File	6 KB



Carpeta Templates

Donde tenemos toda la estructura del código

Name	Date modified	Type	Size
 index.html	7/10/2024 7:42 AM	Chrome HTML Do...	6 KB

Carpeta Upload

Donde manejaremos cada cuenta

Name	Date modified	Type	Size
 usuario221f	7/9/2024 9:58 AM	File folder	
 usuarioe27f	7/9/2024 9:51 AM	File folder	

Código en JAVA

Este código JavaScript agrega funcionalidad a una página web para permitir la selección, vista previa y procesamiento de archivos multimedia, así como la descarga y eliminación de archivos del servidor.

```
document.addEventListener('DOMContentLoaded', function() {
  var fileInput = document.getElementById('mediaInput');
  var gifForm = document.getElementById('gifForm');
  var videoPreview = document.getElementById('videoPreview');
  var selectedImages = document.getElementById('selectedImages');
```

Este bloque de código maneja el evento de cambio cuando el usuario selecciona un archivo.

Dependiendo del tipo de archivo (imagen o video), se actualiza la vista previa:

- **Imagen:** Se crea un elemento de imagen () y se añade al contenedor selectedImages. Además, se oculta el elemento de video.
- **Video:** Se actualiza la src del elemento videoPreview y se muestra. Además, se vacía el contenedor selectedImages.

```

fileInput.addEventListener('change', function(event) {
  var files = event.target.files;
  if (files.length > 0) {
    var file = files[0];
    var fileURL = URL.createObjectURL(file);
    if (file.type.startsWith('image/')) {
      // Mostrar imagen seleccionada
      var img = document.createElement('img');
      img.src = fileURL;
      img.classList.add('selected-image');
      selectedImages.appendChild(img);
      // Ocultar el video
      videoPreview.classList.add('hidden');
    } else if (file.type.startsWith('video/')) {
      // Mostrar video seleccionado
      videoPreview.src = fileURL;
      videoPreview.classList.remove('hidden');
      // Ocultar las imágenes
      selectedImages.innerHTML = "";
    }
  } else {
    // Si no se selecciona ningún archivo, oculta el elemento de videoPreview
    videoPreview.classList.add('hidden');
    selectedImages.innerHTML = "";
  }
});

```

Este bloque de código maneja el evento de envío del formulario.

- ☐ Se recogen varios valores de entrada del formulario.
- ☐ Se crea un objeto FormData y se agregan todos los datos del formulario.
- ☐ Se envía una solicitud fetch al servidor para crear un GIF, utilizando el método POST con el FormData como cuerpo de la solicitud.
- ☐ Los resultados de la solicitud se manejan para mostrar un mensaje de alerta con el resultado.

```

gifForm.addEventListener('submit', function(event) {
  event.preventDefault();
  var files = fileInput.files;
  var duration = document.getElementById('duration').value;
  var start = document.getElementById('start').value;
  var end = document.getElementById('end').value;
  var fpsInput = document.getElementById('fps');
  var fps = fpsInput.value;
  var selectedEffect = document.getElementById('effect').value;

```

```

var percentage = document.getElementById("porcentaje").value;
var exportFormat = document.getElementById('exportFormat').value;

var formData = new FormData();
formData.append('duration', duration);
formData.append('start', start);
formData.append('end', end);
formData.append('fps', fps);
formData.append('effect', selectedEffect);
formData.append("percent", percentage);
formData.append("exportFormat", exportFormat);

for (var i = 0; i < files.length; i++) {
  formData.append('fileInput', files[i]);
}

fetch('/create_gif', {
  method: 'POST',
  body: formData
})
.then(response => response.text())
.then(result => {
  showAlert(result);
})
.catch(error => {
  console.error('Error:', error);
});
});

```

Esta función actualiza el texto del elemento con id='percentage-value' para mostrar el porcentaje de un valor.

```

function showAlert(message) {
  alert(message);
  location.reload();
}

function updateValue(value) {
  document.getElementById('percentage-value').textContent = value + '%';
}

```

este bloque de código maneja el evento de clic del botón de descarga.

- `event.preventDefault()` previene la acción predeterminada.
- Se envía una solicitud fetch al servidor para obtener la lista de archivos disponibles para descargar.

- Si la solicitud es exitosa y hay archivos disponibles, se inicia la descarga de los archivos llamando a `downloadFiles(files, 0)`.

```
document.getElementById('download').addEventListener('click', function(event) {
    event.preventDefault(); // Prevenir cualquier acción predeterminada

    fetch('/get_files')
    .then(response => response.json())
    .then(data => {
        if (data.status === 'success') {
            if (data.files.length > 0) {
                downloadFiles(data.files, 0);
            } else {
                alert('No hay archivos para descargar.');
```

Esta función descarga los archivos uno por uno.

- Se crea un enlace (`<a>`) dinámicamente, se establece su href al archivo a descargar, y se dispara el evento de clic para iniciar la descarga.
- Después de un breve intervalo (1 segundo), se llama recursivamente a sí misma para descargar el siguiente archivo.
- Cuando todos los archivos han sido descargados, se llama a `deleteFiles()` para eliminarlos del servidor.

```
function downloadFiles(files, index) {
    if (index < files.length) {
        var link = document.createElement('a');
        link.href = `/download_file/${files[index]}`;
        link.download = files[index];
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);

        // Espera un pequeño intervalo antes de continuar con la siguiente descarga
        setTimeout(() => {
            downloadFiles(files, index + 1);
        }, 1000);
    }
}
```



```

    } else {
      deleteFiles();
    }
  }
}

```

Esta función envía una solicitud fetch al servidor para eliminar los archivos descargados.

- La respuesta de la solicitud muestra un mensaje de alerta y recarga la página.

Este código es un buen ejemplo de manejo de archivos en el lado del cliente utilizando JavaScript, interacción con un servidor mediante fetch y manipulación del DOM para actualizar la interfaz del usuario.

```

function deleteFiles() {
  fetch('/delete_files', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    }
  })
  .then(response => response.json())
  .then(data => {
    alert(data.message);
    location.reload();
  })
  .catch(error => alert('Error al eliminar archivos: ' + error));
}

```

Código en HTML en CSS

Este código CSS define el estilo de una página web, incluyendo el diseño y la apariencia de varios elementos como el fondo, el texto, formularios, áreas de visualización y más.

```

body {
  background-color: rgb(38, 51, 93);
  color: white;
}

#barra-herramientas {
  background-color: #333;
}

```

```
color: white;
padding: 20px;
text-align: center;
font-family: Georgia, 'Times New Roman', Times, serif;
}
```

```
#leyenda {
color: rgb(249, 116, 75);
font-weight: bold;
text-align: center;
padding: 20px;
}
```

```
#area-visualizacion {
display: flex;
flex-direction: column;
align-items: center;
padding: 20px;
}
```

```
#gifForm {
display: flex;
flex-wrap: wrap;
}
```

```
#carga {
background-color: #f9f9f9;
border: 1px solid #ddd;
border-radius: 10px;
padding: 20px;
text-align: center;
box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
transition: box-shadow 0.3s ease-in-out;
position: relative;
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
}
```

```
#carga input[type="file"] {
position: absolute;
top: 0;
left: 0;
width: 100%;
```

```
height: 100%;
opacity: 0;
cursor: pointer;
z-index: 2;
}

#carga:hover {
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
}

#carga h3 {
  margin-bottom: 10px;
  color: #333;
  font-size: 1.2rem;
}

#edicion {
  background: #f9f9f9;
  padding: 20px;
  color: #333;
  border-radius: 10px;
  margin: 20px;
  width: 80%;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

#edicion label {
  font-size: 1rem;
  margin-bottom: 5px;
}

#edicion input[type="number"],
#edicion select {
  width: 100px;
  font-size: 1rem;
  margin-bottom: 10px;
  padding: 5px;
  border-radius: 5px;
  border: 1px solid #ddd;
}
```

```
#edicion button {
width: 100px;
font-size: 1rem;
padding: 5px;
border-radius: 5px;
border: none;
background-color: rgb(249, 116, 75);
color: white;
cursor: pointer;
transition: background-color 0.3s ease;
}

#edicion button:hover {
background-color: rgb(236, 92, 52);
}

#videoPreview {
width: 40%;
height: 70%;
}

.hidden {
display: none;
}

#selectedImages {
width: 40%;
height: 70%;
}

.selected-image {
width: 100%;
height: auto;
}

.selected-video {
width: 100%;
height: auto;
}

@media only screen and (max-width: 600px) {
  #barra-herramientas h1 {
    font-size: 1.5rem;
  }
}
```

```
#leyenda {
  font-size: 0.8rem;
}

#area-visualizacion {
  font-size: 0.8rem;
}

#gifContainer {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
}

#gifContainer img, #gifContainer video {
  width: 20%;
  height: auto;
  margin: 1%;
  box-sizing: border-box;
  transition: transform 0.2s;
}

#gifContainer img:hover {
  transform: scale(1.1);
}

@media (max-width: 1024px) {
  #gifContainer img {
    width: 30%;
    margin: 1%;
  }
}

@media (max-width: 768px) {
  #gifContainer img {
    width: 45%;
    margin: 2.5%;
  }
}

@media (max-width: 480px) {
  #gifContainer img {
    width: 90%;
    margin: 5%;
  }
}
```

```
}  
}  
  
#botones{  
  display: flex;  
  flex-wrap: wrap;  
}  
  
#saveButton{  
  margin-right: 20px;  
}  
  
/* Estilos generales para el input range */  
input[type="range"] {  
  width: 100%; /* Establece el ancho completo */  
  height: 10px; /* Ajusta la altura */  
  background: #ddd; /* Fondo del rango */  
  outline: none; /* Elimina el contorno */  
  opacity: 0.7; /* Ajusta la opacidad */  
  transition: opacity 0.2s; /* Transición suave para la opacidad */  
}  
  
input[type="range"]:hover {  
  opacity: 1; /* Opacidad completa al pasar el ratón */  
}  
  
/* Estilos personalizados para el pulgar/control deslizante */  
input[type="range"]::-webkit-slider-thumb {  
  -webkit-appearance: none; /* Elimina el estilo predeterminado en Chrome/Safari */  
  appearance: none;  
  width: 5%; /* Ancho del pulgar en porcentaje */  
  height: 25px; /* Altura del pulgar */  
  background: #4CAF50; /* Color del pulgar */  
  cursor: pointer; /* Cambia el cursor a puntero */  
  border-radius: 50%; /* Hace el pulgar circular */  
  box-shadow: 0 0 2px 0 rgba(0, 0, 0, 0.2); /* Sombra para el pulgar */  
}  
  
input[type="range"]::-moz-range-thumb {  
  width: 5%; /* Ancho del pulgar en Firefox */  
  height: 25px; /* Altura del pulgar en Firefox */  
  background: #4CAF50; /* Color del pulgar en Firefox */  
  cursor: pointer; /* Cambia el cursor a puntero */  
  border-radius: 50%; /* Hace el pulgar circular en Firefox */  
  box-shadow: 0 0 2px 0 rgba(0, 0, 0, 0.2); /* Sombra para el pulgar en Firefox */  
}
```

```
}

/* Estilos personalizados para la pista del rango en Firefox */
input[type="range"]::-moz-range-track {
  width: 100%;
  height: 10px;
  background: #ddd;
}

input[type="range"]::-ms-thumb {
  width: 5%;
  height: 25px;
  background: #4CAF50;
  cursor: pointer;
  border-radius: 50%;
  box-shadow: 0 0 2px 0 rgba(0, 0, 0, 0.2);
}

.slider-container {
  display: flex;
  align-items: center;
  width: 80%;
  margin: 0 auto;
}

#percentage-value {
  margin-left: 10px;
  font-size: 1.2em;
}

@media only screen and (min-width: 601px) {
  #barra-herramientas h1 {
    font-size: 2rem;
  }

  #leyenda {
    font-size: 1rem;
  }

  #area-visualizacion {
    font-size: 1rem;
  }
}
```

Código en HTML

Este código HTML describe una página web para un editor de GIFs. La página permite a los usuarios subir videos o imágenes, aplicar varios efectos y configurar ajustes para crear y descargar GIFs u otros formatos de video.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>GIF Creator</title>
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='styles.css') }}"
    />
  </head>
  <body>
```

- **Barra de Herramientas:** Contiene el título del editor de GIFs.
- **Leyenda:** Describe brevemente la funcionalidad del editor.
- **Área de Visualización:** Contiene el formulario para subir archivos y configurar ajustes de edición.

```
<div id="barra-herramientas">
  <h1>Editor de GIF's</h1>
</div>
<div>
  <h2 id="leyenda">
    Con este editor de gif podrás convertir tus mejores videos o imagenes en
    gif de una manera super sencilla.
  </h2>
</div>
<div id="area-visualizacion">
  <form
    id="gifForm"
    action="/create_gif"
```



```

method="post"
enctype="multipart/form-data"
>
<div id="carga">
  <h3>Subir Archivo de Video o Imagenes</h3>
  
  <input
    type="file"
    id="mediaInput"
    name="fileInput"
    accept="image/*, video/*, .gif,.webp,.mp4"
    multiple
  />
</div>

```

- ❑ **Inputs:** Permiten configurar la duración, velocidad (fotogramas por segundo), inicio y fin del recorte.
- ❑ **Efectos:** Selección de varios efectos que se pueden aplicar al GIF.
- ❑ **Slider:** Ajusta un valor porcentual que podría ser utilizado para algunos efectos.
- ❑ **Formato de Exportación:** Permite elegir entre GIF, MP4 y WebM.
- ❑ **Botones:** Botón para enviar el formulario y crear el GIF, y botón para descargar el archivo generado.
- ❑ **Vista Previa de Video e Imágenes Seleccionadas:** Elementos para mostrar una vista previa del contenido cargado.

```

<div id="edicion">
  <label for="duration">Duración (segundos):</label>
  <input
    type="number"
    id="duration"
    name="duration"
    value="3"
    min="1"
    step="1"
    max="15"
  /><br />
  <label for="fps">Velocidad (Fotogramas por segundo):</label>
  <input
    type="number"
    id="fps"
    name="fps"
    value="24"
    min="1"
    step="1"
  /><br />

```

```
<label for="start">Inicio del recorte (segundos):</label>
<input
  type="number"
  id="start"
  name="start"
  value="0"
  min="0"
  step="1"
/><br />
<label for="end">Fin del recorte (segundos):</label>
<input
  type="number"
  id="end"
  name="end"
  value="0"
  min="0"
  step="1"
/><br />
<label for="effect">Efecto:</label>
<select id="effect" name="effect">
  <option value="none">Ninguno</option>
  <option value="grayscale">Escala de Grises</option>
  <option value="invert_colors">Colores Invertidos</option>
  <option value="sepia">Sepia</option>
  <option value="blur">Desenfoque</option>
  <option value="contrast">Aumento de Contraste</option>
  <option value="noise_reduction">Reducción de Ruido</option>
  <option value="edge_detection">Bordes Detectados</option>
  <option value="mosaic">Mosaico</option>
  <option value="wave_distortion">Distorsión de Onda</option>
  <option value="mirror_horizontal">
    Efecto de Espejo (Horizontal)
  </option>
  <option value="mirror_vertical">Efecto de Espejo (Vertical)</option>
  <option value="rotate">Rotación</option>
  <option value="water_reflection">Efecto de Reflejo de Agua</option>
</select>
<div class="slider-container">
  <input
    type="range"
    id="porcentaje"
    value="0"
    min="0"
    max="100"
    oninput="updateValue(this.value)"
```

```

    />
    <span id="percentage-value">0%</span>
  </div>
  <div>
    <label for="exportFormat">Formato de Exportación:</label>
    <select id="exportFormat" name="exportFormat">
      <option value="gif">GIF</option>
      <option value="mp4">MP4</option>
      <option value="webm">WebM</option>
    </select>
  </div>
  <div id="botones">
    <button type="submit" id="saveButton">Realizar</button>
    <button id="download">Descargar</button>
  </div>
</div>
</form>
<video id="videoPreview" class="hidden" controls></video>
<div id="selectedImages"></div>
</div>

```

Archivos GIF Disponibles: Muestra los archivos GIF, MP4 y WebM disponibles. Utiliza Jinja2 para iterar sobre los archivos y mostrarlos según su tipo.

```

<h1>Archivos GIF Disponibles</h1>
<div id="gifContainer">
  {% for file in gif_files %} {% if file.endswith('.gif') %}
  
  {% elif file.endswith('.mp4') %}
  <video controls>
    <source src="{{ url_for('static', filename=file) }}" type="video/mp4" />
    Your browser does not support the video tag.
  </video>
  {% elif file.endswith('.webm') %}
  <video controls>
    <source
      src="{{ url_for('static', filename=file) }}"
      type="video/webm"
    />
    Your browser does not support the video tag.
  </video>
  {% endif %} {% endfor %}
</div>

<script src="https://cdn.socket.io/4.0.1/socket.io.min.js"></script>
<script src="{{ url_for('static', filename='script.js') }}"></script>
</body>

```

```
</html>
```

- ❑ **Socket.io:** Biblioteca para la comunicación en tiempo real entre el cliente y el servidor.
- ❑ **Script.js:** Archivo JavaScript que maneja la lógica de la página, como la carga de archivos, la vista previa, y posiblemente la generación de GIFs.

Código De PY

El código proporcionado es una aplicación web desarrollada con Flask y Flask-SocketIO que permite a los usuarios cargar archivos de video o imágenes, aplicar varios efectos a esos archivos y luego descargarlos en formatos específicos como GIF, MP4 o WebM.

```
import os
import uuid
import cv2
import numpy as np
from flask import Flask, render_template, request, send_file, jsonify, send_from_directory
from flask_socketio import SocketIO
from moviepy.editor import VideoFileClip, ImageSequenceClip
import imageio
import shutil
```

- ❑ Crea una instancia de Flask y Flask-SocketIO.
- ❑ Define una carpeta de subida para los archivos cargados.
- ❑ Crea la carpeta de subida si no existe.

```
app = Flask(__name__, static_folder='static')
socketio = SocketIO(app)

UPLOAD_FOLDER = 'upload'

if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

Cada función aplica un efecto específico a una imagen utilizando OpenCV y devuelve la imagen procesada.

```
def convert_to_grayscale(img, percentage):
    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    gray_img = cv2.cvtColor(gray_img, cv2.COLOR_GRAY2RGB)
    blended_img = cv2.addWeighted(img, 1 - percentage, gray_img, percentage, 0)
    return blended_img
```

```

def apply_sepia(img, percentage):
    sepia_filter = np.array([[0.393 + 0.607 * (1 - percentage), 0.769 - 0.769 * (1 - percentage), 0.189 -
0.189 * (1 - percentage)],
                             [0.349 - 0.349 * (1 - percentage), 0.686 + 0.314 * (1 - percentage), 0.168 - 0.168 * (1 -
percentage)],
                             [0.272 - 0.272 * (1 - percentage), 0.534 - 0.534 * (1 - percentage), 0.131 + 0.869 * (1 -
percentage)]])
    sepia_img = cv2.transform(img, sepia_filter)
    sepia_img = np.clip(sepia_img, 0, 255).astype(np.uint8)
    return sepia_img

def invert_colors(img, percentage):
    inverted_img = cv2.bitwise_not(img)
    return cv2.addWeighted(img, 1 - percentage, inverted_img, percentage, 0)

def apply_blur(img, percentage):
    ksize = int(15 * percentage)
    if ksize % 2 == 0:
        ksize += 1
    blurred_img = cv2.GaussianBlur(img, (ksize, ksize), 0)
    return blurred_img

def increase_contrast(img, percentage):
    lab_img = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab_img)
    clahe = cv2.createCLAHE(clipLimit=3.0 * percentage, tileGridSize=(8, 8))
    l = clahe.apply(l)
    lab_img = cv2.merge((l, a, b))
    contrast_img = cv2.cvtColor(lab_img, cv2.COLOR_LAB2BGR)
    return contrast_img

def reduce_noise(img, percentage):
    h = 10 * percentage
    denoised_img = cv2.fastNlMeansDenoisingColored(img, None, h, h, 7, 21)
    return denoised_img

def detect_edges(img, percentage):
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    if percentage == 0:
        return cv2.cvtColor(gray_img, cv2.COLOR_GRAY2BGR)

    low_threshold = int(100 * percentage)
    high_threshold = int(200 * percentage)

```

```

edges_img = cv2.Canny(gray_img, low_threshold, high_threshold)
return cv2.cvtColor(edges_img, cv2.COLOR_GRAY2BGR)

def apply_mosaic(img, percentage):
    size = max(1, int(10 * (1 - percentage)))
    small_img = cv2.resize(img, None, fx=1.0/size, fy=1.0/size, interpolation=cv2.INTER_AREA)
    return cv2.resize(small_img, img.shape[:2][::-1], interpolation=cv2.INTER_NEAREST)

def apply_wave_distortion(img, percentage):
    rows, cols = img.shape[:2]
    map_x = np.zeros((rows, cols), dtype=np.float32)
    map_y = np.zeros((rows, cols), dtype=np.float32)
    for i in range(rows):
        for j in range(cols):
            map_x[i, j] = int(j + 20 * percentage * np.sin(2 * np.pi * i / 180))
            map_y[i, j] = int(i + 20 * percentage * np.sin(2 * np.pi * j / 180))
    wave_distorted_img = cv2.remap(img, map_x, map_y, interpolation=cv2.INTER_LINEAR)
    return wave_distorted_img

def apply_mirror_effect(img, horizontal=True, percentage=None):
    if horizontal:
        return cv2.flip(img, 1)
    else:
        return cv2.flip(img, 0)

def apply_rotation(img, angle, percentage=None):
    rows, cols = img.shape[:2]
    M = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
    rotated_img = cv2.warpAffine(img, M, (cols, rows))
    return rotated_img

def apply_water_reflection(img, percentage):
    reflection_img = cv2.flip(img, 0)
    alpha = percentage
    beta = (1.0 - alpha)
    reflected_water_img = cv2.addWeighted(img, alpha, reflection_img, beta, 0.0)
    return reflected_water_img

```

Esta ruta maneja la página principal de la aplicación, buscando archivos GIF, MP4 y WebM en la carpeta estática y los pasa al archivo HTML para renderizar.

```

@app.route("/")
def index():
    gif_files = []
    static_folder = app.static_folder

    for root, dirs, files in os.walk(static_folder):

```

```

for file in files:
    if file.endswith(('gif', '.mp4', '.webm')):
        file_path = os.path.relpath(os.path.join(root, file), static_folder)
        file_path = file_path.replace('\\', '/') # Asegura que las barras estén correctas para URL
        gif_files.append(file_path)

return render_template('index.html', gif_files=gif_files)

@app.route('/create_gif', methods=['POST'])
def create_gif():
    duration = float(request.form['duration'])
    start = float(request.form['start'])
    end = float(request.form['end'])
    fps = int(request.form['fps'])
    selected_effect = request.form['effect']
    percentage = float(request.form['percent']) / 100.0
    export_format = request.form['exportFormat']

    files = request.files.getlist('fileInput')
    alert_message = ""

    if files:
        random_code = str(uuid.uuid4())[:4] # Nuevo: Generar código aleatorio para subcarpetas de usuario
        user_folder = os.path.join(app.config['UPLOAD_FOLDER'], f'usuario{random_code}') # Nuevo:
        Crear subcarpeta en 'upload'
        static_user_folder = os.path.join(app.root_path, 'static', f'usuario{random_code}') # Nuevo: Crear
        subcarpeta en 'static'
        os.makedirs(user_folder, exist_ok=True)
        os.makedirs(static_user_folder, exist_ok=True)

        for file in files:
            if file.filename.endswith(('mp4', '.avi', '.mov', '.webp', '.gif')):
                random_filename = str(uuid.uuid4()) + os.path.splitext(file.filename)[1]
                video_path = os.path.join(user_folder, random_filename) # Nuevo: Guardar en subcarpeta de
                usuario
                file.save(video_path)

            if start > 0 or end > 0:
                video = VideoFileClip(video_path).subclip(start, end)
                segment_duration = end - start
            else:
                video = VideoFileClip(video_path)
                segment_duration = video.duration

            if selected_effect != 'none':

```

```

    if selected_effect == 'grayscale':
        video = video.fl_image(lambda img: convert_to_grayscale(img, percentage))
    elif selected_effect == 'sepia':
        video = video.fl_image(lambda img: apply_sepia(img, percentage))
    elif selected_effect == 'invert_colors':
        video = video.fl_image(lambda img: invert_colors(img, percentage))
    elif selected_effect == 'blur':
        video = video.fl_image(lambda img: apply_blur(img, percentage))
    elif selected_effect == 'contrast':
        video = video.fl_image(lambda img: increase_contrast(img, percentage))
    elif selected_effect == 'noise_reduction':
        video = video.fl_image(lambda img: reduce_noise(img, percentage))
    elif selected_effect == 'edge_detection':
        video = video.fl_image(lambda img: detect_edges(img, percentage))
    elif selected_effect == 'mosaic':
        video = video.fl_image(lambda img: apply_mosaic(img, percentage))
    elif selected_effect == 'wave_distortion':
        video = video.fl_image(lambda img: apply_wave_distortion(img, percentage))
    elif selected_effect == 'mirror_horizontal':
        video = video.fl_image(lambda img: apply_mirror_effect(img, horizontal=True,
percentage=percentage))
    elif selected_effect == 'mirror_vertical':
        video = video.fl_image(lambda img: apply_mirror_effect(img, horizontal=False,
percentage=percentage))
    elif selected_effect == 'rotate':
        video = video.fl_image(lambda img: apply_rotation(img, angle=90,
percentage=percentage))
    elif selected_effect == 'water_reflection':
        video = video.fl_image(lambda img: apply_water_reflection(img, percentage))

gif_fps = fps / segment_duration if segment_duration > 0 else 1

output_filename = str(uuid.uuid4()) + '.' + export_format
output_path = os.path.join(static_user_folder, output_filename) # Nuevo: Guardar en
subcarpeta de usuario

if export_format == 'gif':
    video.write_gif(output_path, fps=gif_fps)
elif export_format == 'mp4':
    video.write_videofile(output_path, fps=fps)
elif export_format == 'webm':
    video.write_videofile(output_path, codec='libvpx', fps=fps)

video.close()
os.remove(video_path)

```



```

        alert_message = '¡El archivo de video se ha cargado y procesado correctamente!'
        return alert_message

    elif file.filename.endswith((''.jpg', '.jpeg', '.png', '.bmp')):
        frames = []
        img = cv2.imdecode(np.frombuffer(file.read(), np.uint8), -1)

        if selected_effect != 'none':
            if selected_effect == 'grayscale':
                img = convert_to_grayscale(img, percentage)
            elif selected_effect == 'sepia':
                img = apply_sepia(img, percentage)
            elif selected_effect == 'invert_colors':
                img = invert_colors(img, percentage)
            elif selected_effect == 'blur':
                img = apply_blur(img, percentage)
            elif selected_effect == 'contrast':
                img = increase_contrast(img, percentage)
            elif selected_effect == 'noise_reduction':
                img = reduce_noise(img, percentage)
            elif selected_effect == 'edge_detection':
                img = detect_edges(img, percentage)
            elif selected_effect == 'mosaic':
                img = apply_mosaic(img, percentage)
            elif selected_effect == 'wave_distortion':
                img = apply_wave_distortion(img, percentage)
            elif selected_effect == 'mirror_horizontal':
                img = apply_mirror_effect(img, horizontal=True, percentage=percentage)
            elif selected_effect == 'mirror_vertical':
                img = apply_mirror_effect(img, horizontal=False, percentage=percentage)
            elif selected_effect == 'rotate':
                img = apply_rotation(img, angle=90, percentage=percentage)
            elif selected_effect == 'water_reflection':
                img = apply_water_reflection(img, percentage)

        frames.append(img)

    if frames:
        output_filename = str(uuid.uuid4()) + '.' + export_format
        output_path = os.path.join(static_user_folder, output_filename) # Nuevo: Guardar en
subcarpeta de usuario

        if export_format == 'gif':
            imageio.mimsave(output_path, frames, format='GIF', fps=fps)
        elif export_format == 'mp4':

```

```

        clip = ImageSequenceClip(frames, fps=fps)
        clip.write_videofile(output_path, codec='libx264')
    elif export_format == 'webm':
        clip = ImageSequenceClip(frames, fps=fps)
        clip.write_videofile(output_path, codec='libvpx')

    alert_message = '¡El archivo de imagen se ha cargado y procesado correctamente!'
    return send_file(output_path, as_attachment=True, download_name=output_filename)

else:
    alert_message = '¡No se encontraron archivos válidos para procesar!'
else:
    alert_message = '¡No brindaste ningún archivo!'

return alert_message

@app.route('/download_file/<path:filename>', methods=['GET'])
def download_file(filename):
    return send_from_directory('static', filename)

@app.route('/get_files', methods=['GET'])
def get_files():
    static_folder = 'static'
    files = []
    for root, dirs, file_list in os.walk(static_folder):
        for file in file_list:
            if file.endswith(('gif', 'mp4', 'webp')):
                files.append(os.path.relpath(os.path.join(root, file), static_folder))

    if not files:
        return jsonify({'status': 'error', 'message': '¡No se encontraron archivos válidos para procesar!'})
    return jsonify({'status': 'success', 'files': files})

@app.route('/delete_files', methods=['POST'])
def delete_files():
    static_folder = 'static'
    try:
        for folder_name in os.listdir(static_folder):
            folder_path = os.path.join(static_folder, folder_name)
            if os.path.isdir(folder_path):
                for file_name in os.listdir(folder_path):
                    if file_name.endswith(('gif', 'mp4', 'webp')):
                        os.remove(os.path.join(folder_path, file_name))
                os.rmdir(folder_path) # Elimina la subcarpeta de usuario

```

```
        return jsonify({'status': 'success', 'message': 'Todos los archivos y subcarpetas se han eliminado con éxito.'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': f'Error al eliminar archivos: {str(e)}'})

if __name__ == '__main__':
    socketio.run(app, debug=True)
```

Conclusión

En resumen, esta aplicación web desarrollada con Flask y potenciada por bibliotecas como OpenCV, MoviePy y Numpy ofrece a los usuarios una plataforma versátil para la manipulación y transformación de archivos multimedia. Desde la subida de videos e imágenes hasta la aplicación de una amplia gama de efectos visuales y la exportación en formatos como GIF, MP4 y WebM, la aplicación proporciona herramientas robustas tanto para usuarios individuales como para profesionales creativos. La combinación de un backend sólido en Python con un frontend dinámico en HTML, CSS y JavaScript asegura una experiencia de usuario intuitiva y eficiente. Con su capacidad para editar, mejorar y convertir contenido multimedia de manera flexible y accesible, esta aplicación representa una solución poderosa para las necesidades de edición y creación digital en la web actual.