
# UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
# FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES


**Proyecto Final: Editor de GIFs:**
**Manual Tecnico**

**Docente Responsable:**
GRIFFITH, EDUARDO

**Nombre de la Asignatura:**
Desarrollo de Software V

**Fecha de entrega:**
Martes 9 de julio  del 2024

**Estudiantes:**

Anthuan Austin8-997-326
Lineth Cherigo 8-896-96
Gabriel Garcia 8-883-2158
Castulo Castillo 8-988-785
Estefanie Alzamora 8-984-380

**Grupo:**
1LS231

# Indice

## Introducción

Este manual técnico proporciona una guía detallada sobre la estructura y los componentes utilizados en el desarrollo del proyecto, así como la interacción entre ellos y la documentación del código fuente. Está diseñado para ofrecer una comprensión completa y clara del funcionamiento interno del sistema, facilitando su mantenimiento y evolución.

En primer lugar, se presenta información completa sobre los componentes y librerías utilizados. Esto incluye una descripción de los frameworks empleados, detallando sus versiones específicas, como por ejemplo Node.js v14.17.0. Además, se enumeran todas las librerías adicionales que fueron utilizadas en el proyecto, junto con sus versiones, como Express.js v4.17.1. También se proporciona información sobre los servidores utilizados tanto en el entorno de desarrollo como en producción. Por ejemplo, se menciona el servidor Apache v2.4.46 utilizado durante el desarrollo y el servidor Nginx v1.18.0 en producción. Asimismo, se detalla el entorno de desarrollo integrado (IDE) empleado, como Visual Studio Code v1.56.2, y las herramientas de construcción y gestión de dependencias utilizadas, tales como npm v6.14.13 o Yarn v1.22.10. Otros componentes importantes mencionados incluyen la base de datos MongoDB v4.4.6 y las herramientas de prueba como Jest v26.6.3.

Además, se documenta la interacción entre los diferentes componentes del sistema. Se explica cómo se realiza la comunicación entre el cliente y el servidor para diversas operaciones, como la solicitud de recursos o la actualización de datos. Se detallan todas las instancias en las que un componente interactúa con otro, describiendo el método de comunicación utilizado, ya sea HTTP o WebSocket, los datos intercambiados y el flujo de trabajo correspondiente. Esta sección proporciona una comprensión clara de cómo los componentes del sistema trabajan juntos para cumplir con las funcionalidades requeridas.

Por último, se ofrece una documentación exhaustiva del código del proyecto. Se proporciona un resumen de la estructura general del código, incluyendo una vista general de las clases y funciones principales. Cada clase es descrita en términos de su propósito y cómo se relaciona con otros componentes. Se documenta cada función desarrollada, detallando los datos de entrada y su tipo, como por ejemplo functionName(param1: string, param2: number), así como los datos de salida y su tipo. También se indica dónde se utiliza o se referencia cada clase o función dentro del proyecto.

Este manual está diseñado para ser una referencia integral para desarrolladores y mantenedores del sistema, proporcionando toda la información necesaria para comprender y trabajar con el proyecto de manera eficiente.

**Información completa sobre los componentes / librerías utilizados para construir el proyecto.**

- **Frameworks y Librerias**
  Node,js. Version: **20.12.2**
  Express.js. Version:**4.19.2**
  Fluent-ffmpeg. Version: **2.1.3**
  Gifshot. Version:**4.5**
  IziToast:**1.4.0**

- **Servidores**
  Express.js (Servidor HTTP) : **4.19.2**
- **Entornos de Desarrollo**
  Visual Studio Code

# Documentación de la interacción entre los componentes:

El aplicativo se compone de un servidor backend y un frontend que interactúan entre sí para ofrecer una plataforma robusta y funcional para la gestión de archivos multimedia. El servidor backend, desarrollado en Node.js con Express.js, actúa como el núcleo de la aplicación, proporcionando servicios de almacenamiento, procesamiento y recuperación de datos multimedia. Utiliza librerías como Multer para la carga de archivos y ffmpeg para manipulación de videos, asegurando una gestión eficiente de los recursos multimedia.

Los componentes clave del servidor incluyen:

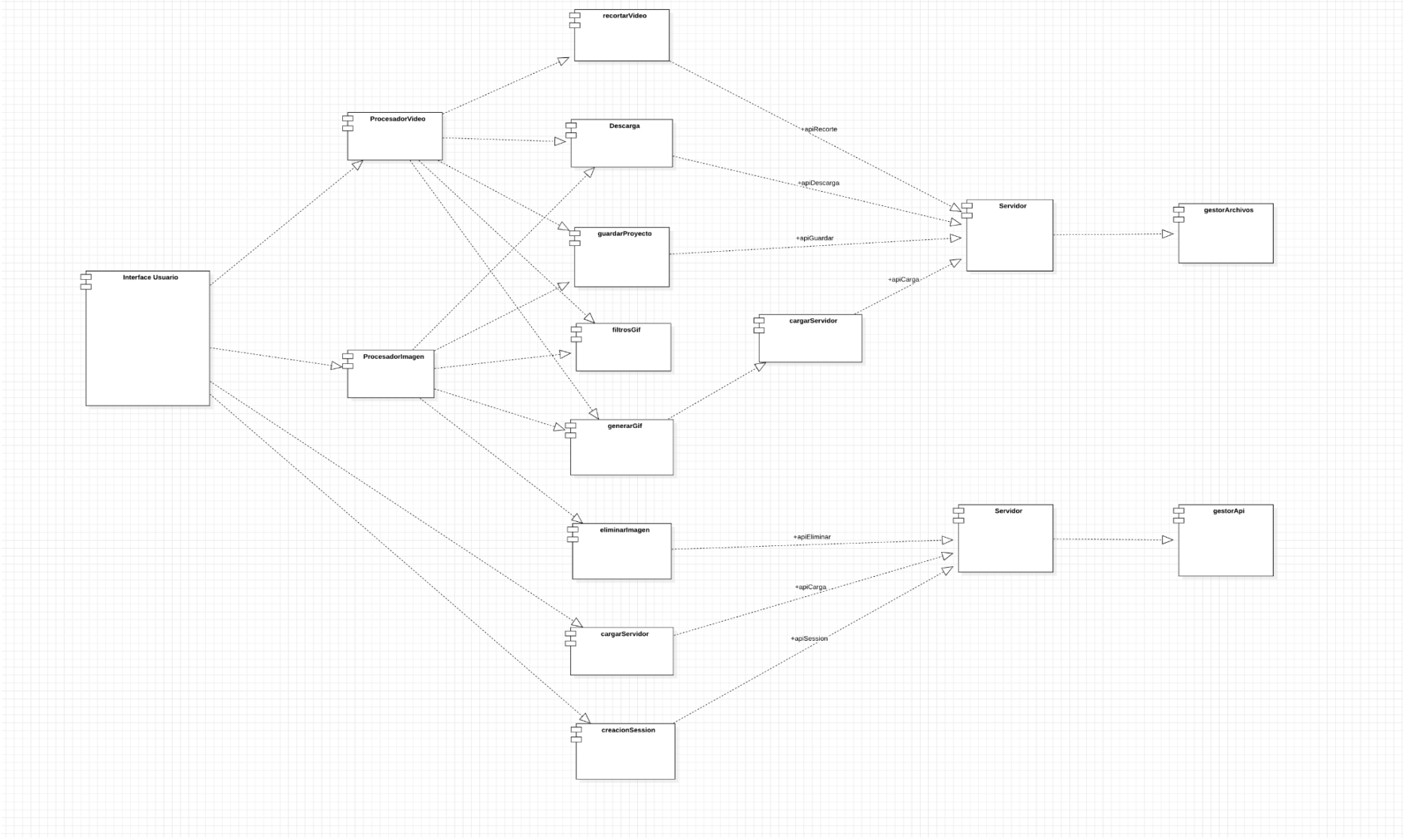
1. **Gestión de Archivos**: Utiliza Multer para recibir y almacenar imágenes y videos en directorios específicos basados en el usuario, asegurando un almacenamiento organizado y accesible.
2. **Manipulación de Multimedia**: Emplea ffmpeg para tareas como la obtención de la duración de los videos y el recorte de segmentos específicos, proporcionando funcionalidades avanzadas para la manipulación y edición de archivos multimedia.
3. **Operaciones de Proyecto**: Facilita la creación, copia y gestión de proyectos de usuario, permitiendo la recuperación y apertura de proyectos previamente guardados con todos sus archivos asociados.
4. **Integración de Filtros**: Permite aplicar filtros personalizados a archivos GIF utilizando la biblioteca `canvas`, proporcionando una forma interactiva para que los usuarios ajusten y visualicen los efectos aplicados en tiempo real.

El frontend, por otro lado, se encarga de la interfaz de usuario (UI) y la experiencia del usuario (UX):

1. **Interfaz de Usuario Intuitiva**: Proporciona una interfaz amigable para que los usuarios interactúen con las funcionalidades del aplicativo, como la carga de archivos, la visualización de proyectos, la aplicación de filtros y la descarga de archivos esto aplica para la sección de video e imágenes.
2. **Integración con el Backend**: Utiliza APIs proporcionadas por el servidor para obtener datos como listas de archivos multimedia, detalles de proyectos y aplicaciones de filtros, asegurando una experiencia coherente y fluida.
3. **Feedback Visual y Operacional**: Ofrece retroalimentación visual inmediata sobre el estado de las operaciones, como la carga de archivos y la aplicación de filtros, mejorando la usabilidad y la satisfacción del usuario.

En conjunto, el servidor backend y el frontend forman un sistema integral que permite a los usuarios gestionar eficientemente sus archivos multimedia, aplicar modificaciones avanzadas y mantener proyectos organizados y accesibles. La interacción entre estos componentes asegura una experiencia de usuario completa y satisfactoria, desde la carga inicial de archivos hasta la gestión avanzada de proyectos y aplicaciones de filtros personalizados.

# Diagrama de componentes y su relación funcional en el aplicativo:



**Documentación del código del proyecto**

Documentar todas las clases y funciones desarrolladas.

## Index.js

`get_server_files(url)`

```
//index.js
Parámetros: {string} url - La URL del servidor desde donde se obtendrán los
archivos.
Retorna: {Promise<Array>} - Retorna una promesa que resuelve en un array de
archivos obtenidos del servidor.
Referenciada: Esta función es referenciada en la función cargar_proyectos().
const get_server_files = async (url) => {
    let response = await fetch(url);
    return response.json();
}
```

**Cargar_Proyectos()**

```
//index.js
Parametros: No tiene
Referenciada: Esta función es invocada en el contexto de la interfaz
de usuario para cargar dinámicamente los proyectos disponibles.
```

```
const cargar_proyectos = async () => {

    let proyect_list = "";
    proyect_list = await get_server_files('/proyects');

    let array = []
    array = proyect_list.file;


    let x = 0;

    proyectos_container.innerHTML = "";

    array.sort();

    array.forEach(element => {

        proyectos_container.innerHTML += `
            <div id='gifGen_${x}' class="GIFGenerated"
style=background-image:url("./proyects/${element}/gif/gif.gif"
onmouseover="onHover_index(this.id, 1)"
onmouseout="onHover_index(this.id, 2)"
onclick="cargar_proyecto(this.id)")>
            </div>
        `;

        x++;
```

```
    });

    for ( let index = 0; index <  8 - array.length; index++) {

        proyectos_container.innerHTML += `
            <div id='gifGen' class="GIFGenerated"
style='background-image:url("./assets/images/preview.gif")' ></div>
        `;
    }

}
```

**onHover_index(id, condition)**

```
//index.js

Maneja el evento hover sobre un elemento.
Parametros: {string} id - El ID del elemento sobre el cual
se está aplicando el evento hover. y {number} condition -
La condición del evento hover (1 para mouseover, 2 para
mouseout).
Referencia: Esta función es invocada en los atributos
onmouseover y onmouseout de los elementos de proyecto para
cambiar la apariencia y comportamiento del elemento en
respuesta al hover.
```

```
const onHover_index = (id, condition) => {
    let modElement = document.getElementById(id);

    if(condition == 1 ){
        lastImage = modElement.style.backgroundImage;
        //CL"background image= " + lastImage);
        modElement.removeAttribute('style');
        modElement.style.backgroundColor = "white";
        modElement.innerHTML += "ABRIR";
    }

    if(condition == 2 ){
        //CLlastImage);
        modElement.style.backgroundImage = lastImage;
        modElement.style.backgroundColor = "white";
        modElement.innerHTML = "";
    }
}

const request = (url, data) => {

    fetch(url, data);

}
```

**request(url,data)**

```
Realiza una solicitud fetch a la URL especificada con los datos proporcionados.
Parametros: {string} url - La URL a la cual se realizará la solicitud fetch. y
{Object} data - Objeto que contiene las opciones para la solicitud fetch (método,
headers, body, etc.).
Referenciada:  Esta función puede ser utilizada para realizar solicitudes HTTP
genéricas al servidor.

const request = (url, data) => {

    fetch(url, data);

}
```

**cargar_proyecto(id)**

```
Carga un proyecto específico basado en su ID.
Parametros:{string} id - El ID del proyecto a cargar.
Referenciada:  Esta función es invocada al hacer clic en un
proyecto para abrirlo en el editor correspondiente.
const cargar_proyecto = (id) => {
    let filter = "";

    filter = lastImage.replace('url("./proyects/',"");
    filter = filter.replace('/gif/gif.gif")',"");

    request(`abrir_proyecto/${filter}/${userInformation}`);
```

```
    if(filter.includes('IMAGE')) {
        window.location.href = `editor.html?type=image`;
    }

    if(filter.includes("VIDEO")) {
        window.location.href = `editor.html?type=video`;
    }

}
```

create_session()

```
Crea una nueva sesión de usuario y la almacena en el
localStorage.
Referenciada: Esta función es invocada durante la carga inicial
de la página para asegurar que haya una sesión de usuario
activa.
const create_session = () => {
    let user = "USER-" + Math.round( Math.random() * 10000);
    localStorage.setItem("user", user);
    request("/session/" + localStorage.getItem('user') );
}
```

mostrarNotificacionError()

```
Muestra una notificación de error utilizando iziToast.
Referenciada: Esta función es invocada cuando ocurre un error al
subir un archivo no admitido.
function mostrarNotificacionError() {
    iziToast.error({
        title: 'Archivo No Admitido',
        message: '¡Intntente Denuevo! Solo Imagenes y Videos',
        position: 'topCenter'
    });
}
```

file_selector.addEventListener('change', async (event))

```
Maneja el evento de cambio en el selector de archivos, sube el archivo seleccionado al servidor y
redirige al editor correspondiente.
Parametros: {Event} event - El evento de cambio del selector de archivos.
Referenciada:Esta función es invocada cuando el usuario selecciona un archivo para subirlo al servidor.
```

```
file_selector.addEventListener('change', async (event) => {

    const file = event.target.files[0];

    if (!file) {
        return;
    }

    const formData = new FormData();
    formData.append('file', file);

    try {

        const response = await fetch(`/upload/${userInformation}`, {
            method: 'POST',
            body: formData
        });

        const data = await response.json();

        if (data.fileType.includes("video") ||
data.fileType.includes("image")) {
            window.location.href = `editor.html?type=${data.fileType}`;
        }else{
           mostrarNotificacionError();
        }
```

```
    } catch (error) {
        console.error('Error al subir archivo:', error);
    }


});
```

document.addEventListener('DOMContentLoaded', async () => {...})

```
Maneja la carga inicial del documento, asegurando que haya una sesión de usuario activa y cargando los
proyectos disponibles.
Referenciada: Esta función es invocada cuando el documento ha terminado de cargarse.
```

```
document.addEventListener('DOMContentLoaded', async () => {

    if( !localStorage.getItem('user') ){
        create_session();
    }else {
        userInformation = localStorage.getItem('user');
        request("/session/" +  userInformation);
    }

    cargar_proyectos();
```

```
});
```

## Editor.JS

### filterOptions.forEach(option => {})

```
/**
 * Agrega un evento de clic a cada opción de filtro, actualizando el valor del slider y el nombre del
filtro seleccionado.
 * Parámetros: {NodeList} filterOptions - Lista de opciones de filtro.
 * Retorna: {void}
 * Referenciada: Esta función se ejecuta para cada opción de filtro al inicializar el editor.
 */
filterOptions.forEach(option => {
    option.addEventListener("click", () => {
        document.querySelector(".active").classList.remove("active");
        option.classList.add("active");
        filterName.innerText = option.innerText;

        if (option.id === "brightness") {
            filterSlider.max = "200";
            filterSlider.value = brightness;
            filterValue.innerText = `${brightness}%`;
```

```
        } else if (option.id === "saturation") {
            filterSlider.max = "200";
            filterSlider.value = saturation;
            filterValue.innerText = `${saturation}%`;
        } else if (option.id === "inversion") {
            filterSlider.max = "100";
            filterSlider.value = inversion;
            filterValue.innerText = `${inversion}%`;
        } else {
            filterSlider.max = "100";
            filterSlider.value = grayscale;
            filterValue.innerText = `${grayscale}%`;
        }
    });
});
```

**rotateOptions.forEach(option => {})**

```
/**
 * Agrega un evento de clic a cada opción de rotación, actualizando los valores de rotación y flip.
 * Parámetros: {NodeList} rotateOptions - Lista de opciones de rotación.
 * Retorna: {void}
 * Referenciada: Esta función se ejecuta para cada opción de rotación al inicializar el editor.
 */
rotateOptions.forEach(option => {
    option.addEventListener("click", () => {
```

```
        if (option.id === "left") {
            rotate -= 90;
        } else if (option.id === "right") {
            rotate += 90;
        } else if (option.id === "horizontal") {
            flipHorizontal = flipHorizontal === 1 ? -1 : 1;
        } else {
            flipVertical = flipVertical === 1 ? -1 : 1;
        }
        applyFilter();
    });
});
```

**applyFilter()**

```
/**
 * Aplica los filtros seleccionados y envía los datos al servidor.
 * Parámetros: Ninguno.
 * Retorna: {void}
 * Referenciada: Esta función es invocada cuando se cambian los valores de los filtros o las opciones de rotación.
 */
const applyFilter = () => {
    const filterData = {
        brightness,
        saturation,
        inversion,
```

```
        grayscale,
        rotate,
        flipHorizontal,
        flipVertical,
        userInformation
    };

    if (window.location.href.includes("type=image")) {
        initial_state_imageEditor_app();
    } else if (window.location.href.includes("type=video")) {
        recibirFiltros(brightness, saturation, inversion, grayscale, rotate, flipHorizontal,
flipVertical);
    }

    guardarFiltrosData(filterData);
}
```

**applyFiltersAndOverwriteGIF(filters)**

```
/**
 * Aplica los filtros y sobrescribe el GIF en el servidor.
 * Parámetros: {Object} filters - Objeto que contiene los datos de los filtros.
 * Retorna: {Promise<void>}
 * Referenciada: Esta función es invocada dentro de applyFilter() para aplicar filtros a un GIF.
 */
async function applyFiltersAndOverwriteGIF(filters) {
```

```
    try {
        const response = await axios.post('http://localhost:3000/apply-filters', filters);
        if (response.data.success) {
            console.log('Filtros aplicados y GIF sobrescrito correctamente');
        }
    } catch (error) {
        console.error('Error al aplicar filtros y sobrescribir el GIF:', error);
    }
};
```

**guardarFiltrosData (filterData)**

```
/**
 * Envía los datos de los filtros al servidor para guardarlos.
 * Parámetros: {Object} filterData - Objeto que contiene los datos de los filtros.
 * Retorna: {void}
 * Referenciada: Esta función es invocada dentro de applyFilter() para guardar los datos de los filtros
en el servidor.
 */
const guardarFiltrosData = (filterData) => {
    fetch(`/save-filters/${userInformation}`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
```

```
        },
        body: JSON.stringify(filterData)
    })
    .then(response => response.json())
    .then(data => {
        console.log('Success:', data);
    })
    .catch((error) => {
        console.error('Error:', error);
    });
}
```

```
/**
 * Restablece los valores de los filtros a sus valores predeterminados y aplica los filtros.
 * Parámetros: Ninguno.
 * Retorna: {void}
 * Referenciada: Esta función puede ser invocada para restablecer los filtros a sus valores iniciales.
 */
const resetFilter = () => {
    brightness = "100"; saturation = "100"; inversion = "0"; grayscale = "0";
    rotate = 0; flipHorizontal = 1; flipVertical = 1;
    filterOptions[0].click();
    applyFilter();
}
```

**fetchGif()**

```
/**
 * Obtiene el GIF del servidor.
 * Parámetros: Ninguno.
 * Retorna: {Promise<string>} - Retorna una promesa que resuelve en la URL del GIF.
 * Referenciada: Esta función puede ser utilizada para obtener la URL del GIF del servidor.
 */
const fetchGif = async () => {
    const response = await fetch(`/get-gif/${userInformation}`);
    const blob = await response.blob();
    const url = URL.createObjectURL(blob);
    return url;
};
```

**saveImage()**

```
/**
 * Guarda la imagen en el formato especificado.
 * Parámetros: {string} formato - El formato en el cual se guardará la imagen (e.g., "png", "jpeg").
 * Retorna: {void}
 * Referenciada: Esta función puede ser utilizada para guardar la imagen en el formato seleccionado.
 */
const saveImage = async(formato) => {
    const downloadUrl = `/download?format=${formato}&userInformation=${userInformation}`;
```

```
    window.location.href = downloadUrl;
};
```

**updateFilter()**

```
/**
 * Actualiza el valor del filtro seleccionado y aplica los filtros.
 * Parámetros: Ninguno.
 * Retorna: {void}
 * Referenciada: Esta función es invocada cuando el slider de filtro cambia su valor.
 */
const updateFilter = () => {
    filterValue.innerText = `${filterSlider.value}%`;
    const selectedFilter = document.querySelector(".filter .active");

    if (selectedFilter.id === "brightness") {
        brightness = filterSlider.value;
    } else if (selectedFilter.id === "saturation") {
        saturation = filterSlider.value;
    } else if (selectedFilter.id === "inversion") {
        inversion = filterSlider.value;
    } else {
        grayscale = filterSlider.value;
    }
    applyFilter();
}
```

**calculateTrimLength()**

```
/**
 * Calcula y muestra la longitud de recorte del video.
 * Parámetros: Ninguno.
 * Retorna: {void}
 * Referenciada: Esta función es invocada para calcular la longitud de recorte del video en el editor.
 */
function calculateTrimLength() {
    var trimLengthValueField = document.getElementById('trim-length-value');
    var trimLengthField = document.getElementById('trim-length');
    var startFieldValue = document.getElementById('start').value;
    var endFieldValue = document.getElementById('end').value;
    var trimLength = endFieldValue - startFieldValue;

    trimLengthValueField.textContent = trimLength >= 2 ? `${trimLength.toFixed(1)} sec` :
trimLength.toFixed(1);
    trimLengthField.style.width = `${(trimLength * 100) / maxVideoDuration}%`;
    trimLengthField.style.marginLeft = `${(startFieldValue * 100) / maxVideoDuration}%`;
}
```

**getSelectedVideoFile()**

```
/**
 * Obtiene el archivo de video seleccionado del servidor.
 * Parámetros: Ninguno.
 * Retorna: {Promise<string>} - Retorna una promesa que resuelve en el nombre del archivo de video
seleccionado.
 * Referenciada: Esta función es utilizada para obtener el nombre del archivo de video del servidor.
 */
function getSelectedVideoFile() {
    return new Promise((resolve, reject) => {
        $.get(`/videoNombre/${userInformation}`)
            .done(function (data, status) {
                if (status === 'success' && data.video) {
                    resolve(data.video);
                } else {
                    reject(new Error('No se pudo obtener el video'));
                }
            })
            .fail(function () {
                reject(new Error('Error al obtener el video'));
            });
    });
}
```

**display(error)**

```
/**
 * Muestra un mensaje de error en la interfaz de usuario.
 * Parámetros: {Object|string} error - El error a mostrar.
 * Retorna: {void}
 * Referenciada: Esta función es utilizada para mostrar mensajes de error en la interfaz de usuario.
 */
function displayError(error) {
    if (typeof error === 'string') {
        $('#errors').text(error).removeClass('d-hide
```

**imprimir_img_cargadas_servidor(images)**

```
/**
 * Función asincrónica para imprimir imágenes cargadas desde el servidor.
 * @param {Array} images - Arreglo de URLs de las imágenes.
 * Referenciada: Llamada en `initial_state_imageEditor_app` para mostrar imágenes cargadas desde el
servidor.
 */
let imprimir_img_cargadas_servidor = async (images) => {
    let x = 0;
    console.log(images);

    let imagesContainer = document.getElementById('images-container');
```

```
    imagesContainer.innerHTML = "";

    images.forEach(image => {
        let imgElement = `
            <div class="imgElement" id="imgElement_${x}" onmouseover="onHover_imgLoaded(this.id, 1)"
onmouseout="onHover_imgLoaded(this.id, 2)" onclick="remove_imgLoaded(this.id)"
style="background-image:url('${image}')">
            </div>
        `;
        imagesContainer.innerHTML += imgElement;
        x++;
    });
}
```

**onHover_imgLoaded(id,condition)**

```
/**
 * Función para manejar eventos de hover en imágenes cargadas.
 * @param {string} id - ID del elemento de imagen.
 * @param {number} condition - Condición del evento (1 para hover, 2 para salir del hover).
 * Referenciada: Utilizada en `imprimir_img_cargadas_servidor` para agregar efectos de hover a las
imágenes.
 */
let onHover_imgLoaded = (id, condition) => {
    let modElement = document.getElementById(id);

    if (condition == 1) {
```

```
        lastImage = modElement.style.backgroundImage;
        modElement.removeAttribute('style');
        modElement.style.backgroundColor = "white";
        modElement.innerHTML += "ELIMINAR";
    }

    if (condition == 2) {
        modElement.style.backgroundImage = lastImage;
        modElement.style.backgroundColor = "white";
        modElement.innerHTML = "";
    }
}
```

**get_server_files(url)**

```
/**
 * Función asincrónica para obtener archivos del servidor.
 * @param {string} url - URL para la solicitud de archivos.
 * @returns {Promise<Array>} - Retorna una promesa que resuelve en un arreglo de URLs de archivos.
 * Referenciada: Llamada en `initial_state_imageEditor_app` para obtener imágenes del servidor.
 */
const get_server_files = async (url) => {
    let response = await fetch(url);
    return response.json();
}
```

**remove_imgLoaded (id)**

```
/**
 * Función para eliminar una imagen cargada.
 * @param {string} id - ID del elemento de imagen a eliminar.
 * Referenciada: Utilizada en `imprimir_img_cargadas_servidor` para eliminar imágenes cuando se hace
clic en ellas.
 */
let remove_imgLoaded = (id) => {
    let delImage = lastImage;
    delImage = delImage.replace('url("/files/images/',"");
    delImage = delImage.replace('")',"");

    fetch(`./delete/${delImage}`).then(async () => {
        initial_state_imageEditor_app();
    });
}
```

**createGif(images,interval)**

```
/**
 * Función para crear un GIF a partir de imágenes.
 * @param {Array} images - Arreglo de URLs de imágenes.
 * @param {number} interval - Intervalo de tiempo entre los fotogramas del GIF.
 * Referenciada: Llamada en `initial_state_imageEditor_app` y `intervalRange` para crear y actualizar el
GIF.
 */
const createGif = (images, interval) => {
```

```
    intervalRage = interval;
    console.log("gifshot");
    gifshot.createGIF({
        images: images,
        filter: `brightness(${brightness}%) saturate(${saturation}%) invert(${inversion}%)
grayscale(${grayscale}%)`,
        interval: interval,
        numFrames: 10,
        frameDuration: 1,
        gifWidth: 400,
        gifHeight: 400,
    }, function (obj) {
        if (!obj.error) {
            previewGif.innerHTML = "";
            previewGif.innerHTML = `<img src=${obj.image} id='preview' alt="preview-gifImage">`;
            imageElement_to_server('preview');
        }
    });
}
```

```
/**
 * Función asincrónica para enviar un elemento de imagen al servidor.
 * @param {string} imageID - ID del elemento de imagen.
 * Referenciada: Utilizada en `createGif` para subir el GIF generado al servidor.
 */
const imageElement_to_server = async (imageID) => {
    let imagenElement = document.getElementById(imageID);
    let imagenSrc = imagenElement.src;

    fetch(imagenSrc).then(res => res.blob()).then(blob => {
        let file = new File([blob], 'imagen.gif', { type: 'image/gif' });
        let formData = new FormData();
        formData.append('file', file);

        fetch(`/uploadBLOB/${userInformation}`, {
            method: 'POST',
            body: formData
        }).then(response => {
            if (!response.ok) {
                throw new Error('Error al enviar la imagen');
            }
        }).catch(error => {
            console.error('Error:', error);
        });
    });
}
```

```
/**
 * Función para hacer solicitudes HTTP.
 * @param {string} url - URL de la solicitud.
 * @param {Object} data - Datos a enviar en la solicitud.
 * Referenciada: Utilizada en múltiples partes del código para hacer peticiones al servidor.
 */
const request = (url, data) => {
    fetch(url, data);
}
```

```
/**
 * Función asincrónica para inicializar el estado de la aplicación de edición de imágenes.
 * Referenciada: Llamada al cargar la página y al realizar cambios en el intervalo del GIF.
 */
let initial_state_imageEditor_app = async () => {
    console.log("initial");
    let url = `./images/${userInformation}`;
    let images = get_server_files(url);

    imprimir_img_cargadas_servidor(await images);

    setTimeout(async () => {
        createGif(await images, (intervalRange.value / 100) * 1);
    }, 1000);
}
```

```
// Eventos para el control de intervalos
intervalRange.addEventListener('change', async () => {
    try {
        initial_state_imageEditor_app();
        intervalRange.addEventListener('input', createGif);
        await saveInterval();
    } catch (error) {
        console.error('Error al obtener las imágenes:', error);
    }
});
```

```
/**
 * Función asincrónica para guardar el intervalo del GIF en el servidor.
 * Referenciada: Llamada en el evento de cambio del intervalo.
 */
const saveInterval = async () => {
    const interval = (intervalRange.value / 100) * 1;
    const filters = { userInformation: userInformation, interval: interval };

    try {
        const response = await axios.post('http://localhost:3000/save-interval', filters);
        if (response.data.success) {
            console.log(response.data.message);
        }
    } catch (error) {
        console.error('Error al guardar intervalo.json:', error);
```

```
    }
}
```

```
// Eventos para abrir y cerrar la vista previa del video
open_video_preview.addEventListener('click', async () => {
    document.getElementById('video-preview-page').style.display = 'flex';
    let response = await fetch(`/video-url/${userInformation}`).then(DATA => {
        return DATA.json();
    }).then(data => {
        document.getElementById('video-reproductor').innerHTML = `
            <video class='video-preview-20' controls muted>
                <source src="files/${userInformation}/videos/${data}" type="video/mp4">
                <source src="movie.ogg" type="video/ogg">
                Your browser does not support the video tag.
            </video>
        `;
    });
});
```

```
close_video_preview.addEventListener('click', async () => {
    document.getElementById('video-preview-page').style.display = 'none';
});
```

```
// Evento para descargar el archivo
```

```
DESCARGAR_ARCHIVO.addEventListener('click', () => {
    iziToast.question({
        rtl: false,
        layout: 1,
        drag: false,
        timeout: 200000,
        close: true,
        closeOnEscape: true,
        overlay: true,
        displayMode: 1,
        id: 'question',
        progressBar: true,
        title: 'Descargar GIF',
        message: 'Selecciona el formato en que deseas descargar el GIF:',
        position: 'center',
        inputs: [['<select id="formatSelect"><option value="gif">GIF</option><option
value="mp4">MP4</option></select>', 'change', function (instance, toast, select, e) {
            formatoSelec = select.options[select.selectedIndex].value;
        }]],
        buttons: [['<button>Descargar</button>', function (instance, toast) {
            saveImage(formatoSelec);
            instance.hide({
                transitionOut: 'fadeOutUp',
                onClosing: function(instance, toast, closedBy){
                    console.info('closedBy: ' + closedBy);
                }
            }, toast, 'buttonName');
```

```
        }]]
    });
});
```

```
// Evento para guardar el proyecto y mostrar notificación
guardar_proyecto.addEventListener('click', () => {
    request(`/safe-proyect/${userInformation}`);
    mostrarNotificacion();
});
```

```
function mostrarNotificacion() {
    iziToast.success({
        title: 'Éxito',
        message: '¡Proyecto guardado exitosamente!',
        position: 'topRight'
    });
}
```

```
// Control de la interfaz de filtros y tiempo
controles_filtros.forEach(input => {
    input.addEventListener('input', async () => {
```

```
        initial_state_imageEditor_app();
    });
});
```

## Video.js

```
/**
 * Función para recibir y actualizar los filtros.
 * Parámetros:
 *   - brillo (String): Valor del brillo.
 *   - saturacion (String): Valor de la saturación.
 *   - inversion (String): Valor de la inversión.
 *   - escalaDeGrises (String): Valor de la escala de grises.
 *   - rotacion (Number): Valor de la rotación.
 *   - espejoHorizontal (Number): Valor del espejo horizontal.
 *   - espejoVertical (Number): Valor del espejo vertical.
 * Retorna: void - No retorna ningún valor.
 */
const recibirFiltros = async (brillo, saturacion, inversion, escalaDeGrises, rotacion, espejoHorizontal,
espejoVertical) => {
    this.brightnesss = brillo;
    this.saturation = saturacion;
```

```
    this.invert = inversion;
    this.grayscale = escalaDeGrises;
    this.rotate = rotacion;
    this.flipHorizontal = espejoHorizontal;
    this.flipVertical = espejoVertical;

    createGifFromVideo(urlvideo);
};
```

```
/**
 * Función para crear un GIF a partir de un video.
 * Parámetros:
 *   - fetchvideo (String): URL del video a procesar.
 * Retorna: void - No retorna ningún valor.
 */
const createGifFromVideo = async (fetchvideo) => {
    urlvideo = fetchvideo;
    try {
        // Obtener el video mediante fetch
        let response = await fetch(fetchvideo);
        let blob = await response.blob();

        let videoBlobUrl = URL.createObjectURL(blob);
        let previewGif = document.getElementById("preview-gif");

        // Convertir el blob del video a una URL
```

```
        let videoUrl = URL.createObjectURL(blob);

        let frameCount = await getFrameCountFromVideoBlobUrl(videoBlobUrl);

        gifshot.createGIF({
            video: videoUrl,
            numFrames: 80, // Número de fotogramas del GIF
            filter: `brightness(${brightnesss}%) saturate(${saturation}%) invert(${invert}%)
grayscale(${grayscale}%)`,
            frameDuration: 1, // Duración de cada fotograma en segundos
            gifWidth: 400, // Ancho del GIF en píxeles
            gifHeight: 400 // Alto del GIF en píxeles
        }, function (obj) {
            if (!obj.error) {
                previewGif.innerHTML = "";
                previewGif.innerHTML = `<img src=${obj.image} id='preview'>`;
                imageElement_to_server('preview');
            }
        });
    } catch (error) {
        console.error('Error al obtener el video:', error);
    }
};
```

```
/**
```

```
 * Función para obtener el número de fotogramas de un video.
 * Parámetros:
 *   - videoBlobUrl (String): URL del blob del video.
 * Retorna: Promise<Number> - Retorna una promesa que resuelve con el número de fotogramas del video.
 */
const getFrameCountFromVideoBlobUrl = async (videoBlobUrl) => {
    // Crear un elemento de video en memoria
    let video = document.createElement('video');
    video.src = videoBlobUrl;

    // Esperar a que el video se cargue
    await new Promise((resolve, reject) => {
        video.addEventListener('loadedmetadata', () => {
            resolve();
        });

        video.addEventListener('error', (error) => {
            reject(error);
        });
    });

    // Obtener el número de fotogramas del video
    let frameCount = video.duration * video.framerate;

    // Liberar recursos
    URL.revokeObjectURL(videoBlobUrl);
```

```
    return
frameCount;
};
```

```
/**
 * Función para hacer polling del estado del video.
 * Parámetros:
 *   - id (String): ID del video.
 * Retorna: void - No retorna ningún valor.
 */
function pollVideoStatus(id) {
    $.get(apiEndpoint + '/' + id, function (response) {
        updateStatus(response.data.status);

        if (!(response.data.status === 'done' || response.data.status === 'failed')) {
            setTimeout(function () {
                pollVideoStatus(id);
            }, pollIntervalSeconds * 1000);
        } else if (response.data.status === 'failed') {
            updateStatus(response.data.status);
        } else {
            saveCutVideo(response.data.url, 'public/user-media/video-cortado/video_cortado.mp4');
        }
    });
}
```

```
/**
 * Evento para prevenir la entrada manual en el campo de inicio.
 * Parámetros:
 *   - event (Event): Evento del DOM.
 * Retorna: void - No retorna ningún valor.
 */
startInput.addEventListener('keydown', (event) => {
    event.preventDefault();
});
```

```
/**
 * Evento para prevenir la entrada manual en el campo de fin.
 * Parámetros:
 *   - event (Event): Evento del DOM.
 * Retorna: void - No retorna ningún valor.
 */
endInput.addEventListener('keydown', (event) => {
    event.preventDefault();
});
```

```
/**
 * Función para mostrar una notificación de éxito.
 * Parámetros: Ninguno.
 * Retorna: void - No retorna ningún valor.
 */
```

```
function mostrarNotificacionVideo() {
    iziToast.success({
        title: 'Éxito',
        message: '¡Video Cortado Exitosamente!',
        position: 'topRight'
    });
}
```

```
/**
 * Función para enviar la edición del video.
 * Parámetros: Ninguno.
 * Retorna: void - No retorna ningún valor.
 */
async function submitVideoEdit() {
    try {
        const start = document.getElementById('start').value;
        const end = document.getElementById('end').value;

        const videoFile = await getSelectedVideoFile();

        const response = await fetch(`/submit/${userInformation}`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({ start, end, video: videoFile }),
```

```
        });

        const data = await response.json();

        if (data.status === 'success') {
            console.log('Video recortado:', data.video);
            mostrarNotificacionVideo();
            createGifFromVideo(`/video_recortado/${userInformation}`);
        } else {
            console.error('Error al recortar el video:', data.message);
        }
    } catch (error) {
        console.error('Error al enviar los datos:', error);
    }
}
```

```
/**
 * Función para actualizar el estado del video.
 * Parámetros:
 *   - status (String): Estado del video.
 * Retorna: void - No retorna ningún valor.
 */
function updateStatus(status) {
    $('#status').removeClass('d-none');
    $('#instructions').addClass('d-none');
    if (progress <= 90)
```

```
        progress += progressIncrement;
    if (progress <= 90) {
        progress += progressIncrement;
    }
    if (status === 'submitted') {
        $('#status .fas').attr('class', 'fas fa-spinner fa-spin fa-2x');
        $('#status p').text('SUBMITTED');
    } else if (status === 'queued') {
        $('#status .fas').attr('class', 'fas fa-history fa-2x');
        $('#status p').text('QUEUED');
    } else if (status === 'fetching') {
        $('#status .fas').attr('class', 'fas fa-cloud-download-alt fa-2x');
        $('#status p').text('DOWNLOADING ASSETS');
    } else if (status === 'rendering') {
        $('#status .fas').attr('class', 'fas fa-server fa-2x');
        $('#status p').text('RENDERING VIDEO');
    } else if (status === 'saving') {
        $('#status .fas').attr('class', 'fas fa-save fa-2x');
        $('#status p').text('SAVING VIDEO');
    } else if (status === 'done') {
        $('#status .fas').attr('class', 'fas fa-check-circle fa-2x');
        $('#status p').text('READY');
        progress = 100;
    } else {
        $('#status .fas').attr('class', 'fas fa-exclamation-triangle fa-2x');
        $('#status p').text('SOMETHING WENT WRONG');
        $('#submit-video').prop('disabled', false);
```

```
        progress = 0;
    }
    $('.progress-bar').css('width', progress + '%').attr('aria-valuenow', progress);
}
```

```
/**
 * Función para establecer la duración del video desde el archivo.
 * Parámetros:
 *   - url (String): URL del archivo de video.
 * Retorna: void - No retorna ningún valor.
 */
const setVideoDurationFromFile = (url) => {
    var $sourceLengthValueField = $('#source-length-value');
    var $startField = $('#start');
    var $endField = $('#end');

    $.get(url, function (data, status) {
        var duration = data.duration;
        // duration ya está en segundos
        var maxVideoDuration = Math.round(duration * 10) / 10;
        $sourceLengthValueField.text(maxVideoDuration);
        $startField.val(0);
        $endField.val(maxVideoDuration);
    });
};
```

## Conclusión

En conclusión, este manual técnico proporciona una visión integral y detallada del proyecto, abordando todos los aspectos clave desde los componentes utilizados hasta la interacción entre ellos y la documentación específica del código. Al detallar las versiones de frameworks, servidores y entornos de desarrollo, así como las herramientas y librerías empleadas, se asegura que cualquier desarrollador o mantenedor tenga un conocimiento claro de la infraestructura del sistema. La documentación exhaustiva de la interacción entre componentes y la comunicación entre el cliente y el servidor facilita la comprensión de los flujos de trabajo y las dependencias internas del sistema, lo cual es crucial para la resolución de problemas y la implementación de mejoras.

Además, la descripción detallada de las clases y funciones, con un enfoque en los datos de entrada, salida y referencias, permite a los desarrolladores entender rápidamente el propósito y la funcionalidad de cada parte del código. Esto no solo optimiza el proceso de desarrollo y mantenimiento, sino que también asegura que el proyecto pueda ser escalado y adaptado a futuras necesidades con mayor facilidad.

Este manual técnico es una herramienta esencial para garantizar la continuidad y el éxito del proyecto, proporcionando una base sólida de conocimiento que facilita la colaboración y la eficiencia en todas las etapas del ciclo de vida del software. Con esta documentación, el equipo de desarrollo está bien equipado para abordar cualquier desafío y continuar evolucionando el sistema de manera efectiva.