



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería  
Informática**

**Sistema de Procesamiento de  
Vídeo  
Documentación Técnica**



Presentado por Álvaro Márquez  
en Universidad de Burgos — 7 de julio de 2025  
Tutor: D. José Miguel Ramírez Sanz y D. José  
Luís Garrido Labrador



---

# Índice general

---

Índice general	i
Índice de figuras	iii
Índice de tablas	v
<b>Apéndice A Plan de Proyecto Software</b>	<b>1</b>
A.1. Introducción . . . . .	1
A.2. Planificación Temporal . . . . .	2
A.3. Estudio de viabilidad . . . . .	6
<b>Apéndice B Especificación de Requisitos</b>	<b>11</b>
B.1. Introducción . . . . .	11
B.2. Objetivos Generales del Proyecto . . . . .	11
B.3. Catálogo de Requisitos . . . . .	11
B.4. Actores del Sistema . . . . .	15
B.5. Catálogo de Requisitos Funcionales . . . . .	16
B.6. Casos de Uso . . . . .	17
B.7. Especificación de requisitos . . . . .	18
<b>Apéndice C Especificación de Diseño</b>	<b>23</b>
C.1. Introducción . . . . .	23
C.2. Diseño de Datos . . . . .	23
C.3. Diseño Arquitectónico . . . . .	24
C.4. Diseño Procedimental . . . . .	26
<b>Apéndice D Documentación Técnica de Programación</b>	<b>31</b>

D.1. Introducción . . . . .	31
D.2. Estructura de Directorios del Repositorio . . . . .	31
D.3. Manual del Programador (Arquitectura del Software) . . . . .	34
D.4. Compilación, Instalación y Ejecución del Proyecto . . . . .	36
D.5. Pruebas del Sistema . . . . .	39
D.6. Fallos y Soluciones . . . . .	41
<b>Apéndice E Documentación de Usuario</b>	<b>47</b>
E.1. Introducción . . . . .	47
E.2. Requisitos Previos del Sistema . . . . .	47
E.3. Instalación del Sistema . . . . .	49
E.4. Manual de Uso . . . . .	51
E.5. Resolución de Problemas (FAQ) . . . . .	57
<b>Apéndice F Anexo de Sostenibilización Curricular</b>	<b>59</b>
F.1. Introducción . . . . .	59
F.2. Competencias de Sostenibilidad Aplicadas al TFG . . . . .	60
<b>Bibliografía</b>	<b>63</b>

---

## Índice de figuras

---

A.1. Ejemplo del tablero Kanban utilizado en Jira para la gestión y seguimiento visual de las tareas del proyecto. . . . .	2
A.2. Ejemplo del épicas generadas durante el proyecto. . . . .	3
B.1. Diagrama de los principales casos de uso del sistema. . . . .	15
B.2. Diagrama de casos de uso. . . . .	19
C.1. Diagrama de la arquitectura general del sistema. . . . .	25
C.2. Arquitectura de red para el despliegue seguro de Jitsi. . . . .	26
C.3. Diagrama de flujo de datos del pipeline de procesamiento. . . . .	27
C.4. Diagrama conceptual del proceso de grabación de Jibri. . . . .	27
C.5. Diagrama de los principales casos de uso del sistema. . . . .	28
C.6. Diagrama de flujo detallado del procesamiento de vídeo en Spark. . . . .	29
D.1. Diagrama de flujo de datos del pipeline de procesamiento implementado en la carpeta /src. . . . .	35
D.2. Salida de la terminal mostrando el despliegue de los servicios del pipeline (Kafka, Spark, etc.) con Docker Compose. . . . .	36
D.3. Configuración del dominio público en el servicio de DNS Dinámico DuckDNS. . . . .	37
D.4. Configuración de la redirección de puertos en el router para permitir el acceso externo a Jitsi. . . . .	38
D.5. Error de conexión encontrado durante la depuración de Jitsi en Debian. . . . .	38
D.6. Captura de servicio <i>worker</i> levantado y corriendo. . . . .	39
D.7. Logs del contenedor <code>python-processor</code> que muestran la ejecución exitosa del pipeline completo. . . . .	40
D.8. Directorio de salida con el fichero de vídeo ya procesado. . . . .	40

D.9. Captura de video ya procesado. . . . .	41
D.10. Capture de error persistente de Conexión. . . . .	43
E.1. Captura de configuración correcta personalizada puertos. . . . .	49
E.2. Captura de servicio Duckdns configurado. . . . .	49
E.3. Pantalla de inicio de Jitsi Meet donde el usuario inicia la conferencia. . . . .	52
E.4. Menú de opciones para iniciar la grabación de la sesión. . . . .	53
E.5. Indicador visual <i>REC</i> que muestra que la sesión se está grabando activamente. . . . .	54
E.6. Ejemplo del directorio de grabaciones de Jibri con el fichero MP4 resultante. . . . .	55
E.7. Monitorización de los logs del procesador Python, confirmando la ejecución del pipeline. . . . .	56
E.8. Directorio de salida mostrando el vídeo final una vez procesado. . . . .	57

---

# Índice de tablas

---

A.1. Tareas principales del Sprint 1. . . . .	3
A.2. Tareas principales del Sprint 2. . . . .	4
A.3. Tareas principales de los Sprints 3 y 4. . . . .	5
A.4. Tareas del Epic 3: Finalización del Proyecto. . . . .	6
A.5. Costes de personal. . . . .	7
A.6. Costes de <i>hardware</i> . . . . .	7
A.7. Costes de servicios. . . . .	7
A.8. Coste total teórico del proyecto. . . . .	8
A.9. Tabla con las licencias de las herramientas utilizadas. . . . .	8
B.1. Caso de uso 1: Desplegar la Infraestructura Completa. . . . .	20
B.2. Caso de uso 2: Ejecutar el Pipeline de Procesamiento de Vídeo. . . . .	21





## Apéndice A

---

# Plan de Proyecto Software

---

### A.1. Introducción

La planificación de un proyecto es una fase fundamental para su correcto desarrollo. En este apartado se comentará, por una parte, la planificación temporal del proyecto mediante los distintos *sprints* que se han llevado a cabo, y por otra parte, se realizará un breve estudio sobre la viabilidad del proyecto.

En este proyecto se ha utilizado una metodología ágil basada en los principios de *Scrum* [4]. Para una correcta planificación, se optó por agrupar el trabajo en Épicas que contienen Sprints con objetivos temáticos, gestionando todas las tareas a través de un tablero en la plataforma **Jira** [1]. Aunque las reuniones con los tutores eran periódicas para comentar dudas y avances, la estructura en sprints permitió un desarrollo iterativo y organizado.

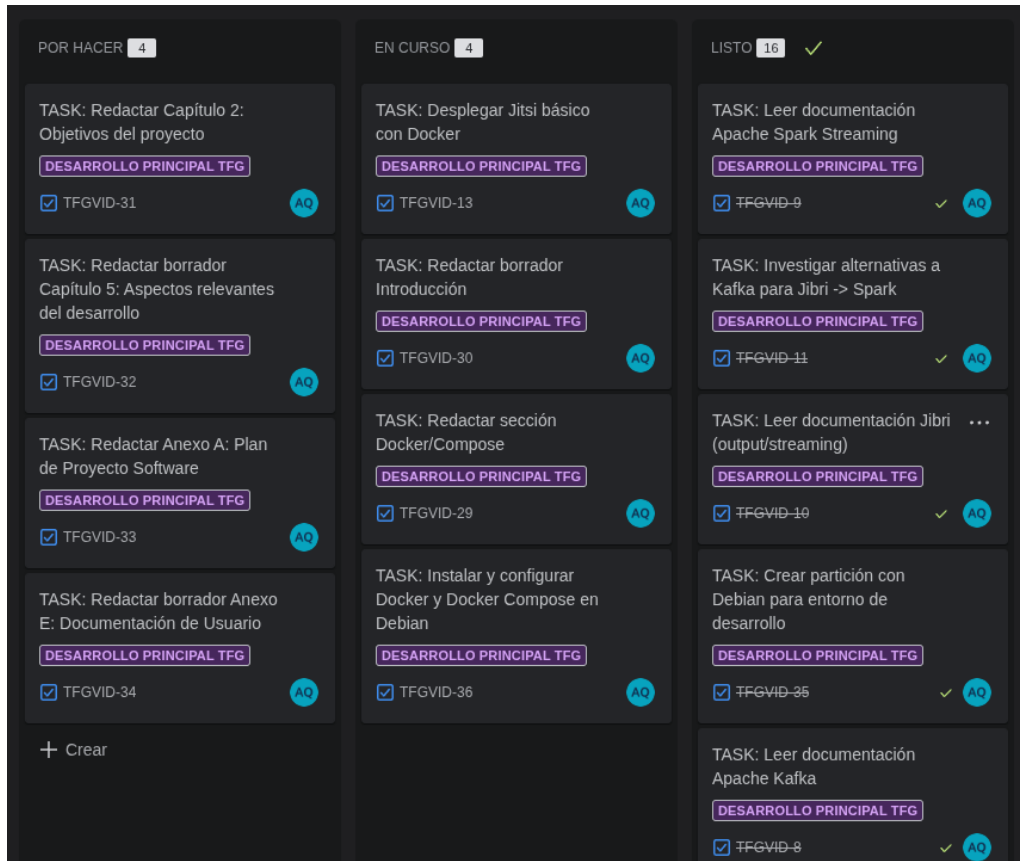


Figura A.1: Ejemplo del tablero Kanban utilizado en Jira para la gestión y seguimiento visual de las tareas del proyecto.

## A.2. Planificación Temporal

El desarrollo del proyecto se ha estructurado en Épicas que agrupan distintos *sprints*, permitiendo un seguimiento claro de los grandes hitos del proyecto. A continuación, se detallan las fases y las tareas más relevantes de cada una, reflejando la evolución real del trabajo.

Actividad	Persona asignada	Informador	Prioridad	Estado	Resolución	Creada
> TFGVID-40 Desarrollo Final TFG	Alvaro Marquez...	Alvaro Marquez...	Medium	FINALIZADA	Listo	14 jun 2025, 20:00
> TFGVID-18 Desarrollo Principal TFG	Alvaro Marquez...	Alvaro Marquez...	Medium	FINALIZADA	Listo	21 abr 2025, 16:47
> TFGVID-1 Configuración Inicial y Herramientas	Alvaro Marquez...	Alvaro Marquez...	Medium	FINALIZADA	Listo	25 mar 2025, 20:30

Figura A.2: Ejemplo del épicas generadas durante el proyecto.

## Épica 1: Investigación, Configuración y Desarrollo Inicial (Febrero 2025 - Abril 2025)

Esta primera épica abarcó desde el inicio del proyecto hasta la pausa de Semana Santa. El objetivo era establecer las bases teóricas y técnicas del TFG, asegurando que se contaba con el conocimiento y las herramientas adecuadas para comenzar el desarrollo.

### Sprint 1: Investigación y Configuración de Herramientas

Este *sprint* inicial fue de carácter exploratorio y de configuración. Se realizó una investigación exhaustiva de las tecnologías clave, se preparó el entorno de control de versiones y se configuraron las herramientas de gestión y documentación.

Tareas (Jira ID)	Est.	Final
TASK: Investigar y seleccionar una plantilla L <sup>A</sup> T <sub>E</sub> X para la UBU (TFGVID-2)	2h	2h
TASK: Configurar repositorio GitHub con estructura inicial (doc/ y src/) (TFGVID-3)	2h	3h
TASK: Configurar Overleaf y vincular con GitHub (si es posible) (TFGVID-4)	2h	2h
TASK: Leer documentación Apache Kafka (TFGVID-8)	16h	20h
TASK: Leer documentación Apache Spark Streaming (TFGVID-9)	20h	18h
TASK: Leer documentación Jibri (output/streaming) (TFGVID-10)	16h	18h
TASK: Investigar alternativas a Kafka para Jibri -> Spark (TFGVID-11)	10h	14h

Tabla A.1: Tareas principales del Sprint 1.

La mayor dificultad durante este *sprint* fue asimilar la gran cantidad de información sobre el ecosistema de tecnologías distribuidas, entendiendo cómo interactúan entre sí Kafka, Spark y Jitsi, así como sus complejas configuraciones.

### Sprint 2: Prototipado del Pipeline Base en Docker

Con las herramientas ya investigadas, el objetivo de este *sprint* fue crear un primer prototipo funcional de la infraestructura. Este *sprint* se centró en la *contenerización* de la aplicación y sus dependencias.

Tareas (Jira ID)	Est.	Final
TASK: Crear Dockerfile básico para código Python (TFGVID-21)	8h	6h
TASK: Construir y probar imagen Docker para script Python (TFGVID-22)	16h	18h
TASK: Añadir Zookeeper a docker-compose.yml (TFGVID-27)	8h	10h
TASK: Añadir Kafka a docker-compose.yml (TFGVID-28)	4h	7h

Tabla A.2: Tareas principales del Sprint 2.

El principal desafío técnico fue la depuración de los primeros ficheros `Dockerfile` y `docker-compose.yml`, resolviendo errores relacionados con el contexto de construcción y la configuración de red entre contenedores.

## Épica 2: Despliegue, Depuración y Pivote de Infraestructura (Abril 2025 - Mayo 2025)

Esta segunda épica se centró en el despliegue del componente más complejo, Jitsi, y la resolución de los problemas de infraestructura que surgieron, lo que llevó a una decisión estratégica clave en el proyecto.

### Sprint 3 y 4: Despliegue de Jitsi, Bloqueo y Pivote Estratégico

El objetivo inicial era desplegar una instancia de `docker-jitsi-meet` en Windows con WSL2. Esta fase se convirtió en un ciclo de depuración intensivo debido a errores de permisos en los volúmenes montados y fallos de autenticación de Jibri. Tras un intento infructuoso de implementar HTTPS,

se tomó la decisión estratégica de migrar el entorno a Debian 12, donde, tras un nuevo ciclo de configuración, se logró un despliegue estable.

Tareas de los Sprints 3 y 4 (Jira ID)	Est.	Final
TASK: Desplegar Jitsi básico con Docker (local) (TFGVID-13)	20h	30h
TASK: (Cancelada) Implementar HTTPS en Jitsi local	6h	6h
TASK: Instalar y configurar Docker y Docker Compose en Debian (TFGVID-26)	20h	24h
TASK: Configurar data-root de Docker para gestión de disco	8h	7h
TASK: Solucionar error de montaje del contenedor web de Jitsi	8h	8h

Tabla A.3: Tareas principales de los Sprints 3 y 4.

Épica 3: Finalización del Proyecto (Mediados de Mayo 2025 en adelante)

Esta épica final engloba todas las tareas restantes para la conclusión del proyecto, divididas en una fase final de implementación técnica y una fase de documentación y entrega.

Fase	Tareas	Est.	Final
	TASK: Preparar entorno limpio para el despliegue de Jitsi	2h	3h
	TASK: Implementar HTTPS en Jitsi con Certificados Let's Encrypt	16h	14h
	TASK: Validar la grabación de vídeo con Jibri	16h	24h
	TASK: Configurar el entorno de procesamiento con Spark	18h	20h
	TASK: Desarrollar el script de procesamiento de vídeo (Prueba de Concepto)	8h	7h
	TASK: Integrar y probar el pipeline completo (Jitsi -> Spark)	20h	16h
	TASK: Definir y documentar el formato de salida de los resultados	4h	6h
	TASK: Documentar el despliegue y la depuración de Jitsi	6h	6h
	TASK: Documentar la implementación del pipeline de Spark	6h	6h
	TASK: Finalizar el diseño de la arquitectura y el diagrama de flujo	8h	6h
	TASK: Redactar la versión final del Manual de Usuario	6h	6h
	TASK: Escribir las conclusiones y realizar la revisión final	6h	6h
	TASK: Preparar el repositorio de GitHub para la entrega	4h	6h
	TASK: Realizar Anexo de Sostenibilidad Curricular	4h	6h

Tabla A.4: Tareas del Epic 3: Finalización del Proyecto.

### A.3. Estudio de viabilidad

En este apartado se va a comentar la viabilidad del proyecto. Por una parte, se redactará la *viabilidad económica*, que hará referencia al coste que supondría el desarrollo del proyecto si se realizara en un entorno empresarial, y por otra parte, se detallará la *viabilidad legal* de las herramientas y librerías utilizadas.

#### Viabilidad económica

En este apartado se presentan unos cálculos económicos teóricos para desarrollar el proyecto. Para realizar el cálculo, se han dividido los gastos en:

1. **Coste de personal:** en la tabla A.5 se encuentra una estimación del coste que supondría contratar a un ingeniero de software durante la duración del TFG (aproximadamente 6 meses).
2. **Coste *hardware*:** la tabla A.6 contiene la inversión en el *hardware* necesario para el desarrollo.
3. **Coste de servicios:** la tabla A.7 detalla los servicios externos utilizados.

Finalmente, en la tabla A.8 se puede observar un cálculo final del coste teórico del proyecto.

Concepto	Coste (€)
Salario mensual bruto (Ingeniero de Software Jr.) [3]	2.100,00
Seguridad Social a cargo de la empresa (aprox. 31 %)	651,00
<b>Coste mensual por empleado</b>	<b>2.751,00</b>
<b>Total 6 meses (1 empleado)</b>	<b>16.506,00</b>

Tabla A.5: Costes de personal.

Concepto	Coste (€)	Coste amortizado (4 años)
Ordenador de desarrollo	1.200,00	150,00
Servidor/Host para despliegue	2.000,00	250,00
<b>Total</b>	<b>3.200,00</b>	<b>400,00</b>

Tabla A.6: Costes de *hardware*.

Concepto	Coste mensual (€)
Conexión a Internet de alta velocidad	50,00
Dominio y servicio DuckDNS	0,00 (Gratuito)
<b>Total (por 6 meses)</b>	<b>300,00</b>

Tabla A.7: Costes de servicios.

Concepto	Coste (€)
Personal (6 meses)	16.506,00
<i>Hardware</i> (amortizado 6 meses)	400,00
Servicios (6 meses)	300,00
<b>Coste total estimado del proyecto</b>	<b>17.206,00</b>

Tabla A.8: Coste total teórico del proyecto.

## Viabilidad legal

En este subapartado se exponen las distintas licencias que tienen las herramientas y librerías utilizadas, así como la licencia final con la que cuenta este proyecto. En la tabla A.9 se muestran las tecnologías utilizadas y su correspondiente licencia.

Librería/Herramienta	Licencia
<i>Debian 12</i>	GPL y otras FOSS
<i>Docker</i>	Apache 2.0
<i>Apache Kafka</i>	Apache 2.0
<i>Apache Spark</i>	Apache 2.0
<i>Jitsi Meet</i>	Apache 2.0
<i>Python</i>	Python Software Foundation License
<i>OpenCV-Python</i>	GNU v3.0 License
<i>NumPy</i>	BSD 3-Clause

Tabla A.9: Tabla con las licencias de las herramientas utilizadas.

La licencia final escogida para el código fuente de este proyecto ha sido **GNU v3.0 License**, ya que es una licencia de software libre muy permisiva que permite su reutilización en prácticamente cualquier escenario, incluyendo proyectos comerciales, sin imponer grandes restricciones.

### *Copyright* de terceros

Este proyecto se ha construido sobre el trabajo de comunidades y organizaciones de código abierto. A continuación, se detallan los principales titulares de derechos de las tecnologías utilizadas:

- **Apache 2.0:**
  - *Apache Kafka* - The Apache Software Foundation



- *Apache Spark* - The Apache Software Foundation
- *Docker* - Docker, Inc.
- *Jitsi Meet* - 8x8, Inc.
- **BSD y GNU v3.0:**
  - *NumPy* - The NumPy community
  - *OpenCV* - OpenCV team
- **PSF:**
  - *Python* - Python Software Foundation



## *Apéndice B*

---

# Especificación de Requisitos

---

### B.1. Introducción

En este anexo se detallan los requisitos que debe cumplir el sistema desarrollado en este Trabajo de Fin de Grado. Se presentarán los objetivos generales del proyecto, un catálogo de requisitos funcionales y no funcionales que definen el comportamiento y las cualidades de la infraestructura, los actores que interactúan con el sistema y los principales casos de uso.

### B.2. Objetivos Generales del Proyecto

Como se detalló en el Capítulo 2, el objetivo principal de este TFG es el **diseño e implementación de una infraestructura de backend robusta, escalable y tolerante a fallos** para un sistema de procesamiento de vídeo. El propósito es crear una base sólida que permita la captura, transporte y análisis de sesiones de telerehabilitación, superando las limitaciones del sistema predecesor.

### B.3. Catálogo de Requisitos

A continuación, se presenta un catálogo detallado de los requisitos que debe satisfacer el sistema.

## Requisitos Técnicos del Entorno

Para la correcta ejecución y despliegue del sistema desarrollado, se deben cumplir una serie de requisitos tanto a nivel de hardware como de software en la máquina anfitriona (*host*).

### Requisitos de Hardware

Los requisitos de *hardware* se refieren a la máquina física o virtual sobre la que se desplegará toda la infraestructura Docker.

- **CPU:** Se recomienda un procesador multi-núcleo (4 o más núcleos) con soporte para tecnologías de virtualización (VT-x/AMD-V) habilitado en la BIOS/UEFI.
- **Memoria RAM:** Debido a la cantidad de servicios que se ejecutan simultáneamente (Jitsi, Kafka, Spark, etc.), se recomienda un mínimo de **16 GB de RAM** para un funcionamiento fluido.
- **Almacenamiento:** Se requiere un mínimo de 50 GB de espacio en disco disponible para el sistema operativo, las imágenes Docker y el almacenamiento de los vídeos generados. Es recomendable utilizar una unidad de estado sólido (SSD) para un mejor rendimiento.
- **Red:** Una conexión a internet estable y un *router* con capacidad para configurar la re-dirección de puertos (*port forwarding*), necesario para el despliegue de Jitsi con acceso externo.

### Requisitos de Software (Sistema Anfitrión)

Estos son los componentes de software que deben estar instalados directamente en el sistema operativo anfitrión.

1. **Sistema Operativo:** Se recomienda una distribución de Linux estable. Todo el desarrollo y las pruebas se han realizado sobre **Debian 12 "Bookworm"**.
2. **Motor de Contenerización:**
  - **Docker Engine:** Versión 20.10 o superior.
  - **Docker Compose:** Versión v2.0 o superior (integrado como plugin de Docker).

### Requisitos de Software (Gestionados por Docker)

Gracias a la contenerización, la mayoría de las dependencias de software están encapsuladas en sus respectivas imágenes Docker, eliminando la necesidad de instalarlas manualmente en el sistema anfitrión. Las imágenes y librerías principales son:

- **Imágenes Docker Base:**
  - *Jitsi Meet Stack*: Imágenes oficiales del proyecto `docker-jitsi-meet` (`jitsi/web`, `jitsi/prosody`, `jitsi/jicofo`, `jitsi/jvb`, `jitsi/jibri`).
  - *Apache Kafka*: Imagen de Confluent Inc. (`confluentinc/cp-kafka`).
  - *Apache Spark*: Imagen de Bitnami (`bitnami/spark`) con un master y un worker.
  - *Python*: Imagen oficial de Python (`python:3.9-slim`) como base para el contenedor de procesamiento.
- **Librerías de Python (instaladas vía `requirements.txt`):**
  - *pyspark*: para la interacción con el clúster de Spark.
  - *kafka-python*: para la comunicación con el broker de Kafka.
  - *numpy==1.26.4*: versión fijada para garantizar la compatibilidad con OpenCV.
  - *opencv-python*: para el procesamiento de los fotogramas de vídeo.

### Requisitos Funcionales (RF)

Los requisitos funcionales describen las capacidades y el comportamiento específico que la infraestructura debe ofrecer.

- **RF-01: Captura de Sesiones de Vídeo.** El sistema debe ser capaz de grabar una sesión de videoconferencia completa, incluyendo audio y vídeo, y almacenarla como un archivo en formato MP4 en un directorio persistente.
- **RF-02: Detección de Nuevos Vídeos.** El sistema de procesamiento debe ser capaz de detectar automáticamente la presencia de nuevos archivos de vídeo en el directorio de entrada.
- **RF-03: Procesamiento de Vídeo.** El sistema debe leer un archivo de vídeo de entrada y aplicar sobre él una transformación de procesamiento de imágenes predefinida (ej. inversión de color) a cada fotograma.

- **RF-04: Almacenamiento de Resultados.** El sistema debe guardar el vídeo resultante del procesamiento como un nuevo archivo MP4 en un directorio de salida específico.
- **RF-05: Archivado de Vídeos Procesados.** Una vez que un vídeo ha sido procesado con éxito, el sistema debe mover el archivo original a un directorio de archivado para evitar su re procesamiento.
- **RF-06: Gestión de Servicios Orquestada.** Todos los componentes de la infraestructura (servicios de captura, bus de mensajería, clúster de procesamiento) deben poder ser desplegados y gestionados de forma conjunta mediante un único comando.

## Requisitos No Funcionales (RNF)

Los requisitos no funcionales definen las cualidades del sistema y las restricciones bajo las cuales debe operar. Son especialmente críticos en un proyecto de infraestructura como este.

- **RNF-01: Portabilidad.** La infraestructura completa debe ser portable entre diferentes máquinas de desarrollo o servidores, garantizando un comportamiento idéntico gracias al uso de contenedores Docker.
- **RNF-02: Escalabilidad.** La arquitectura debe estar diseñada para ser escalable. Esto implica que componentes como los nodos de procesamiento o los brokers de mensajería deben poder replicarse para manejar una mayor carga de trabajo en el futuro.
- **RNF-03: Fiabilidad.** El sistema debe ser fiable. El uso de un bus de mensajería debe garantizar que los datos no se pierdan durante el transporte entre los distintos componentes del pipeline.
- **RNF-04: Mantenibilidad.** El código fuente y la estructura de directorios deben estar organizados de forma clara y modular para facilitar su comprensión y futuras modificaciones.
- **RNF-05: Código Abierto.** Todos los componentes de software utilizados deben tener licencias de código abierto que permitan su uso y modificación en el marco de un proyecto académico.

## B.4. Actores del Sistema

Aunque gran parte de este proyecto se centra en la infraestructura de *backend*, el sistema completo está diseñado para ser utilizado por diferentes tipos de actores, cada uno con un rol específico.

1. *Paciente*: Es el usuario final del sistema de tele-rehabilitación. Su única función es unirse a la sesión de videoconferencia para realizar sus ejercicios.
2. *Terapeuta*: Es el profesional sanitario que dirige la sesión. Además de guiar al paciente, tiene la capacidad de iniciar y detener la grabación de la sesión de ejercicios.
3. *Administrador del Sistema*: Es el usuario técnico responsable de desplegar, gestionar y supervisar la infraestructura. Aunque no participa en la sesión, es quien asegura que el sistema esté operativo y quien podría acceder a los resultados del procesamiento para su validación.

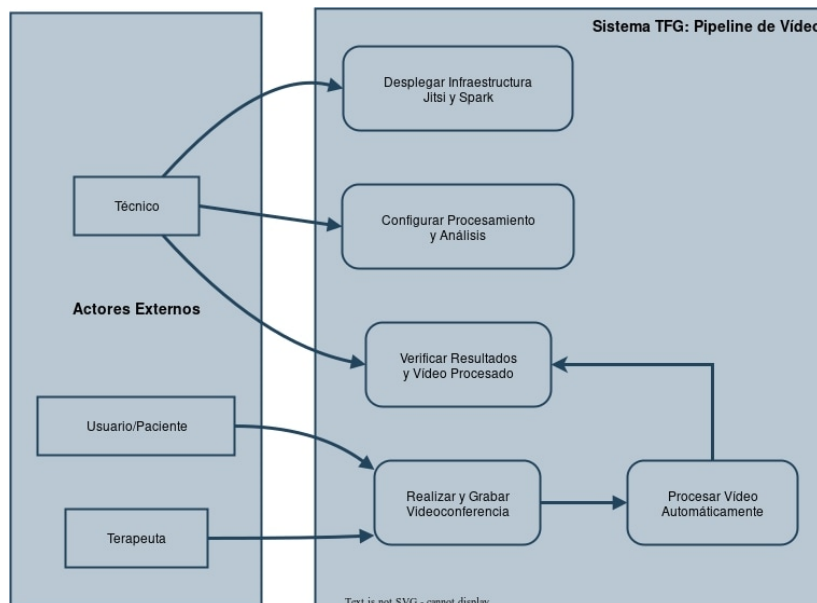


Figura B.1: Diagrama de los principales casos de uso del sistema.

## B.5. Catálogo de Requisitos Funcionales

A continuación, se detallan los requisitos funcionales del sistema, descritos como un flujo de trabajo lógico desde la perspectiva de los actores.

- **RF.1: Realización de una Sesión de Telerehabilitación**
  - **RF.1.1:** El Paciente debe poder unirse a una sala de videoconferencia de Jitsi a través de un enlace web.
  - **RF.1.2:** El Terapeuta debe poder unirse a la misma sala de videoconferencia para dirigir la sesión.
- **RF.2: Gestión de la Grabación de la Sesión**
  - **RF.2.1:** El Terapeuta debe disponer de un botón o función en la interfaz de Jitsi para iniciar la grabación de la sesión.
  - **RF.2.2:** Al iniciar la grabación, el sistema debe activar automáticamente el servicio Jibri, que se unirá a la llamada como un participante silencioso.
  - **RF.2.3:** El Terapeuta debe poder detener la grabación en cualquier momento.
  - **RF.2.4:** Al detener la grabación, el sistema debe guardar la sesión completa como un único archivo de vídeo en formato MP4 en un directorio de entrada predefinido en el servidor.
- **RF.3: Procesamiento Automático del Vídeo Grabado**
  - **RF.3.1:** El sistema debe detectar automáticamente cuándo un nuevo fichero MP4 ha sido depositado en el directorio de entrada.
  - **RF.3.2:** El sistema debe iniciar un proceso que tome el nuevo vídeo y lo envíe a través del pipeline de datos.
  - **RF.3.3:** El pipeline debe aplicar una transformación de vídeo predefinida (la prueba de concepto de inversión de color) a todos los fotogramas del vídeo.
  - **RF.3.4:** El sistema debe guardar el vídeo resultante de la transformación en un directorio de salida.
- **RF.4: Gestión de Ficheros y Resultados**
  - **RF.4.1:** El Administrador del Sistema debe poder acceder al directorio de salida para verificar el vídeo procesado.



- **RF.4.2:** Una vez que un vídeo original ha sido procesado con éxito, el sistema debe moverlo automáticamente a una carpeta de archivado para evitar que sea procesado de nuevo.

## B.6. Casos de Uso

A continuación, se describen los principales casos de uso desde la perspectiva del Administrador del Sistema.

### Caso de Uso 1: Desplegar la Infraestructura Completa

- **Actor:** Administrador del Sistema.
- **Descripción:** El administrador despliega todos los servicios de la arquitectura (Jitsi, Kafka, Spark, etc.) en un entorno limpio.
- **Precondiciones:** El sistema anfitrión (Debian 12) tiene Docker y Docker Compose instalados.
- **Flujo Principal:**
  1. El administrador clona el repositorio del proyecto desde GitHub.
  2. El administrador configura los ficheros `.env` necesarios.
  3. El administrador ejecuta el comando `docker-compose up` principal.
  4. El sistema levanta todos los contenedores, crea las redes y los volúmenes necesarios.
- **Postcondiciones:** Todos los servicios están en ejecución y listos para operar.

### Caso de Uso 2: Ejecutar el Pipeline de Procesamiento de Vídeo

- **Actor:** Administrador del Sistema (o un proceso automatizado).
- **Descripción:** El administrador inicia el script que procesa un vídeo grabado previamente.
- **Precondiciones:** La infraestructura está desplegada y en ejecución. Existe al menos un archivo MP4 en el directorio de entrada.

- **Flujo Principal:**

1. El administrador ejecuta el script de procesamiento de vídeo.
2. El script detecta el vídeo más antiguo en la carpeta de entrada.
3. Los datos del vídeo se envían a través de Kafka y son procesados por Spark.
4. Se genera un nuevo archivo de vídeo procesado en el directorio de salida.
5. El archivo de vídeo original se mueve al directorio de archivado.

- **Postcondiciones:** El vídeo ha sido procesado y archivado. El sistema queda a la espera de nuevos vídeos.

## B.7. Especificación de requisitos

En este apartado se van a mostrar, por una parte el diagrama de caso de uso, como se puede observar en las figuras [B.2](#) y por otra, las tablas de casos de uso.

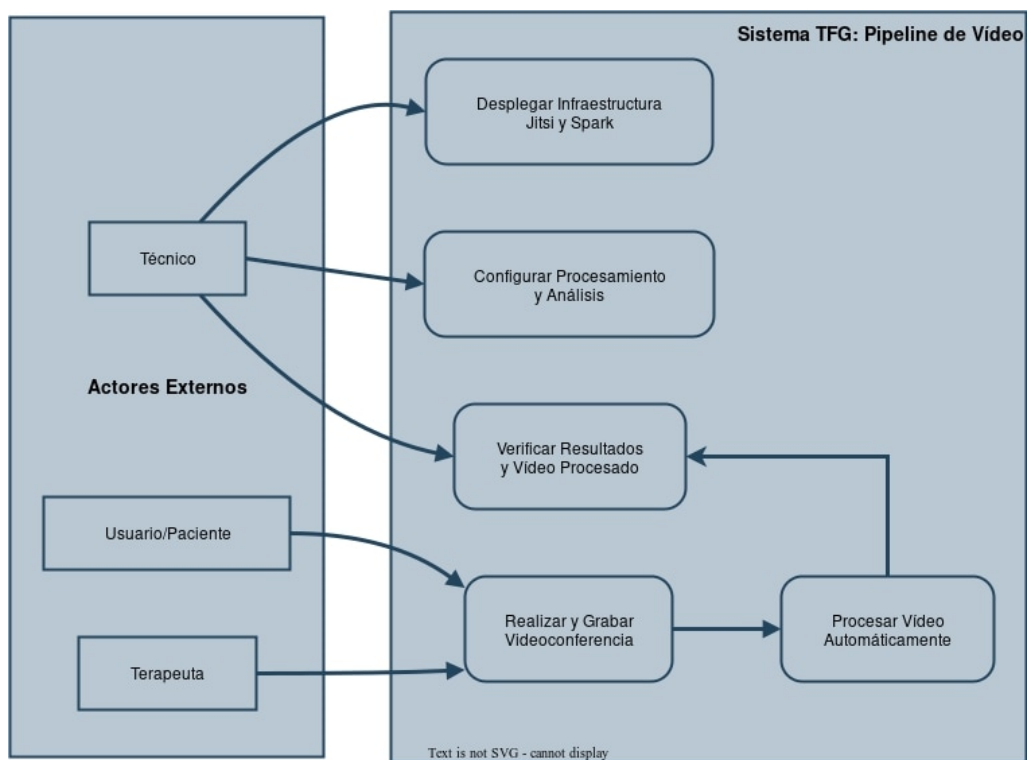


Figura B.2: Diagrama de casos de uso.

Tabla B.1: Caso de uso 1: Desplegar la Infraestructura Completa.

<b>Descripción</b>	Permite al Administrador del Sistema desplegar todos los servicios de la arquitectura de forma orquestada.	
<b>Requisitos</b>	RF.1.1	
	RF.1.2	
	RF.1.3	
<b>Precondiciones</b>	El sistema anfitrión (Debian 12) tiene Docker y Docker Compose instalados y funcionando correctamente.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El Administrador clona el repositorio del proyecto desde GitHub.
	2	El Administrador configura los ficheros de entorno ( <code>.env</code> ) necesarios.
	3	El Administrador ejecuta el comando <code>docker-compose up</code> desde la raíz del proyecto.
	4	El sistema levanta todos los contenedores, crea las redes y los volúmenes necesarios.
<b>Postcondiciones</b>	Todos los servicios de la infraestructura (Jitsi, Kafka, Spark) están en ejecución y listos para operar.	
<b>Excepciones</b>	El host no tiene conexión a internet, lo que impide la descarga de las imágenes Docker desde los registros públicos.	
<b>Importancia</b>	Alta	
<b>Urgencia</b>	Alta	

Tabla B.2: Caso de uso 2: Ejecutar el Pipeline de Procesamiento de Vídeo.

<b>Descripción</b>	Permite procesar un vídeo grabado en una sesión de Jitsi para obtener un resultado analizado.	
<b>Requisitos</b>	RF.2.1, RF.2.2, RF.3.1, RF.3.2, RF.3.3, RF.4.1, RF.4.2, RF.4.3	
<b>Precondiciones</b>	La infraestructura está desplegada y en ejecución. Se ha realizado una sesión de telerehabilitación y el archivo MP4 resultante se encuentra en el directorio de entrada.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El Terapeuta inicia y finaliza la grabación de una sesión en Jitsi. Jibri guarda el fichero MP4.
	2	El Administrador (o un proceso automatizado) ejecuta el script de procesamiento.
	3	El script detecta el vídeo, lo procesa a través del pipeline Kafka/Spark y guarda el resultado.
	4	El script archiva el vídeo original para evitar su reprocesamiento.
<b>Postcondiciones</b>	El vídeo ha sido procesado y el resultado se encuentra en el directorio de salida. El vídeo original ha sido archivado.	
<b>Excepciones</b>	No hay vídeos nuevos en el directorio de entrada; el script finaliza sin realizar ninguna acción.	
<b>Importancia</b>	Alta	
<b>Urgencia</b>	Alta	



## Apéndice C

---

# Especificación de Diseño

---

### C.1. Introducción

La fase de diseño permite planificar un proyecto para su correcta implementación, desarrollo y evolución. En este anexo se exponen los diferentes diseños que se han llevado a cabo para obtener una solución robusta y escalable a los problemas planteados. Se detallará el diseño de los datos que maneja el sistema, el diseño arquitectónico de alto nivel, y el diseño procedimental que describe el flujo de trabajo del pipeline.

### C.2. Diseño de Datos

El diseño de datos se centra en la estructura y el formato de la información que fluye a través del sistema. Aunque el dato principal es el vídeo, su tratamiento y los metadatos asociados son clave.

#### Objeto de Datos Principal: Vídeo de Sesión

El dato principal que se procesa es el archivo de vídeo generado por Jibri. Este es un fichero en formato **MP4**, que contiene un *stream* de vídeo codificado (normalmente con el *códec* H.264) y un *stream* de audio.

#### Formato de Salida del Procesamiento

Una de las tareas de la fase final del proyecto es definir el formato de salida que generará el *script* de Spark. Se ha establecido que el resultado será un nuevo fichero de vídeo también en formato MP4. Para una futura

evolución del proyecto donde se extraigan características específicas, se podría definir un formato de datos estructurado (como JSON o Avro) que contenga metadatos como:

- `id_sesion`: Identificador único de la sesión grabada.
- `timestamp`: Marca de tiempo del procesamiento.
- `algoritmo_aplicado`: Nombre de la transformación realizada (ej. "inversion-color").
- `ruta_video_procesado`: Ruta al fichero de vídeo resultante.
- `métricas`: Un objeto que podría contener métricas de análisis (ej. número de objetos detectados, nivel de movimiento, etc.).

### C.3. Diseño Arquitectónico

El diseño arquitectónico define la estructura de alto nivel del sistema, sus componentes principales y las interacciones entre ellos.

#### Diagrama de Arquitectura General del Sistema

La arquitectura general se ha diseñado como un sistema distribuido basado en micro-servicios, donde cada componente tiene una responsabilidad única. La Figura C.1 ofrece una visión completa de la solución, desde la captura de vídeo hasta el procesamiento final.



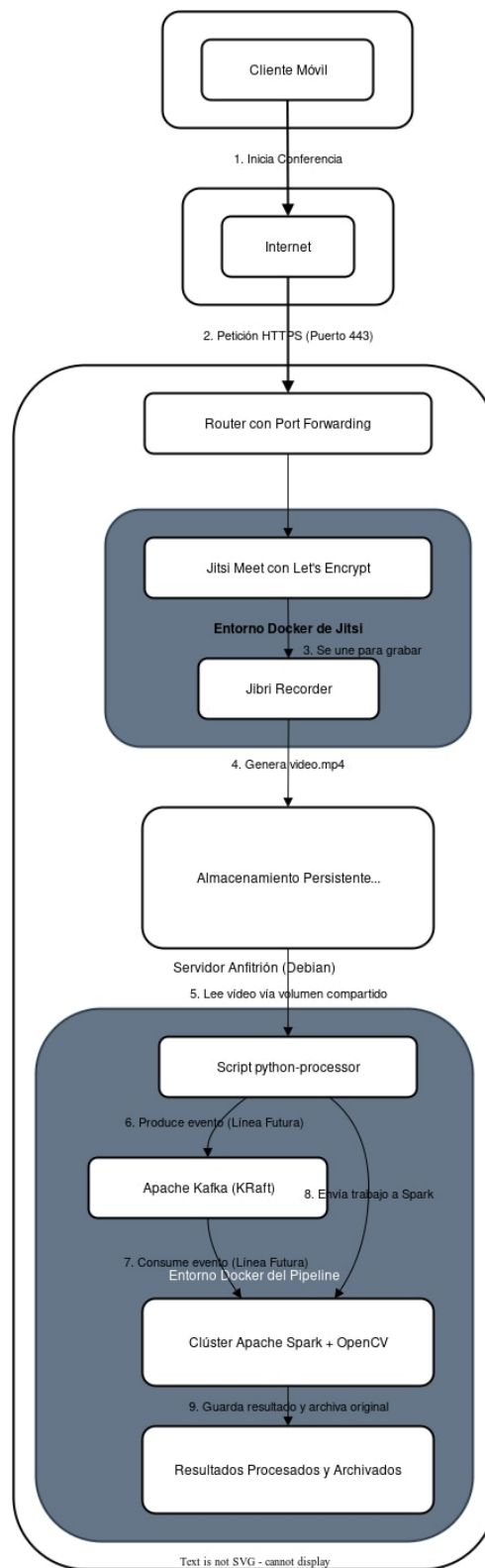


Figura C.1: Diagrama de la arquitectura general del sistema.

Como se puede observar, el sistema se divide en tres grandes bloques: el subsistema de captura de vídeo (basado en Jitsi), el pipeline de datos (orquestrado con Docker Compose y centrado en Kafka y Spark), y el almacenamiento persistente.

## Diagrama de la Arquitectura de Red para Jitsi

El despliegue de Jitsi para que fuera accesible desde el exterior de forma segura fue uno de los principales desafíos técnicos. La solución final, representada en la Figura C.2, se basa en una configuración de red que incluye un servicio de DNS dinámico, redirección de puertos y la generación de certificados SSL/TLS con Let's Encrypt.

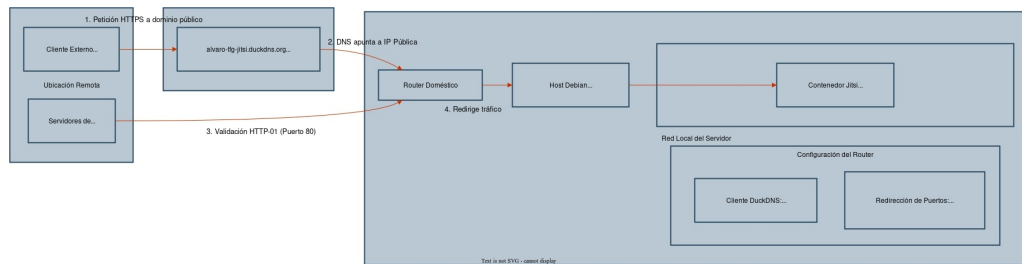


Figura C.2: Arquitectura de red para el despliegue seguro de Jitsi.

## C.4. Diseño Procedimental

El diseño procedimental describe la secuencia de pasos y el flujo de control del sistema.

### Flujo de Datos del Pipeline de Procesamiento

El núcleo de la lógica de negocio reside en el pipeline implementado dentro del directorio `/src`. El flujo de datos (ver Figura C.3) describe la interacción entre el *script* de Python, Kafka y Spark para procesar los vídeos de forma automatizada.

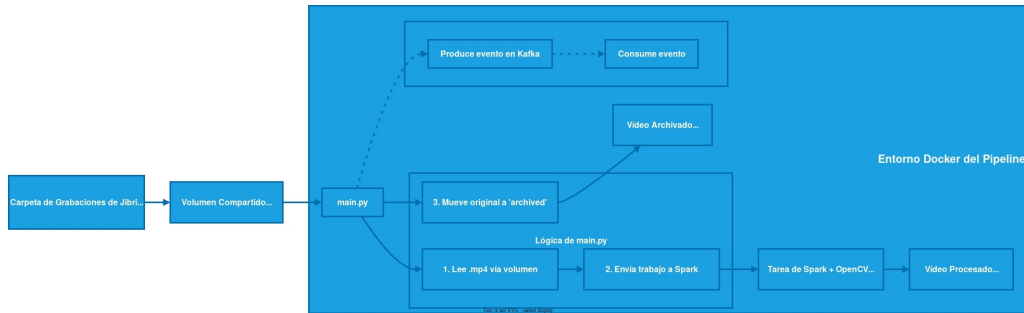


Figura C.3: Diagrama de flujo de datos del pipeline de procesamiento.

El proceso comienza cuando el *script* principal detecta un nuevo archivo MP4. Este archivo es enviado como un mensaje a Kafka. Un consumidor de Spark recibe el mensaje, ejecuta la lógica de procesamiento y guarda el resultado, archivando finalmente el vídeo original.

## Funcionamiento Interno de la Grabación de Jibri

Para comprender cómo se genera la fuente de datos, es útil visualizar el funcionamiento interno de Jibri (Figura C.4). Jibri lanza una instancia de un navegador en modo *headless* que se une a la conferencia, renderiza la vista compuesta y utiliza FFmpeg para capturar esta salida y codificarla en un archivo MP4.

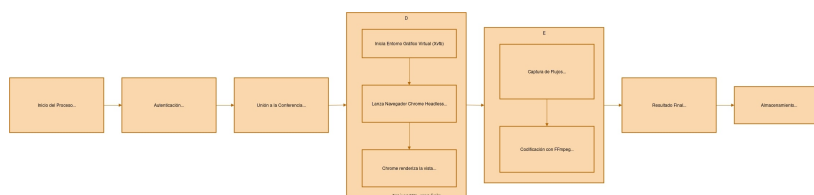


Figura C.4: Diagrama conceptual del proceso de grabación de Jibri.

## Casos de Uso del Sistema

Finalmente, el diagrama de casos de uso (Figura C.5) muestra la interacción de los diferentes actores (Técnico, Usuario, Terapeuta) con las funcionalidades principales del sistema.

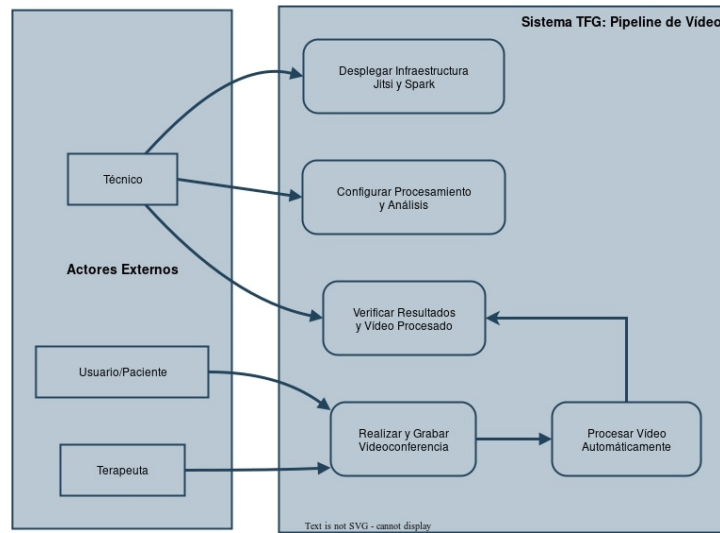


Figura C.5: Diagrama de los principales casos de uso del sistema.

## Flujo Detallado de Transformación en Spark

Para comprender en profundidad cómo se procesan los datos una vez que llegan al consumidor de Spark, la Figura C.6 detalla el flujo de transformación de paquetes que se ejecuta dentro del script de PySpark.

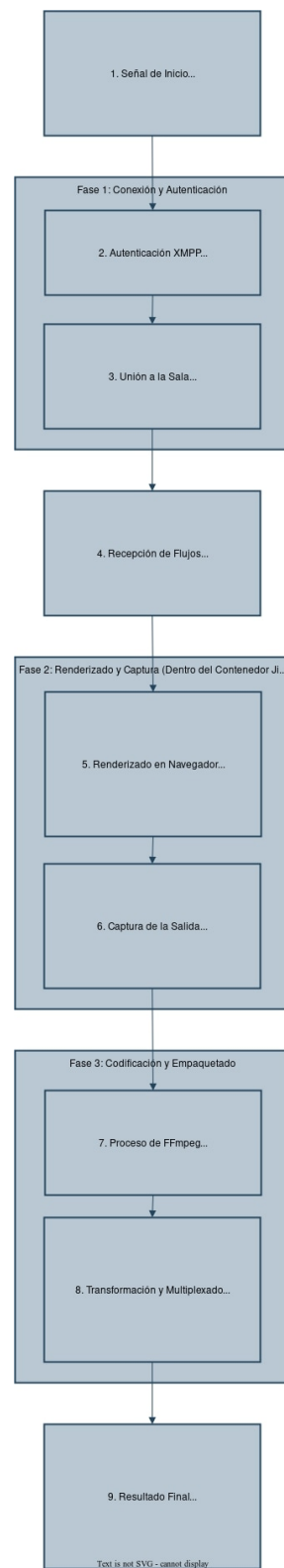


Figura C.6: Diagrama de flujo detallado del procesamiento de vídeo en Spark.

El proceso procedimental es el siguiente:

1. El consumidor Spark recibe un mensaje desde el *topic* de Kafka. Este mensaje contiene los datos o la referencia al vídeo a procesar.
2. Los datos son pasados a una función de procesamiento definida por el usuario.
3. Dentro de esta función, se utiliza la librería OpenCV para abrir el fichero de vídeo y comenzar a leerlo fotograma a fotograma.
4. Se itera sobre cada *frame*: se aplica la transformación de vídeo correspondiente (en la prueba de concepto, una inversión de color) y el *frame* modificado se escribe en un nuevo fichero de vídeo de salida.
5. Una vez procesados todos los *frames*, se cierra el flujo de escritura, finalizando la creación del vídeo procesado.

Este diseño modular permite que la lógica de transformación (paso 4) pueda ser fácilmente modificada o extendida en el futuro para aplicar análisis más complejos.

Para poder ver correctamente los diagramas debido a su tamaño se recomienda descargarlos de la documentación.

## *Apéndice D*

---

# Documentación Técnica de Programación

---

## D.1. Introducción

Este anexo está dirigido a un perfil técnico (un futuro desarrollador o evaluador del proyecto) y tiene como objetivo detallar la estructura del código, la arquitectura de los componentes de software, las decisiones de diseño y el proceso completo de compilación y despliegue del sistema. La finalidad es que cualquier persona con los conocimientos técnicos adecuados pueda comprender, replicar y extender el trabajo realizado.

## D.2. Estructura de Directorios del Repositorio

La organización del repositorio en GitHub ha sido una pieza clave para mantener el proyecto ordenado y comprensible. A continuación, se expone la estructura de directorios que cuelgan desde la raíz del proyecto.

### Directorio de Documentación (‘/docs’)

La carpeta `/docs` contiene toda la documentación del proyecto, principalmente la memoria del TFG y sus recursos asociados. Su estructura es la siguiente:

```
/docs
├── /diagrams
```

```

├── /images
├── /tex
├── memoria.tex
├── anexos.tex
└── bibliografia.bib

```

- `/diagrams`: Almacena los ficheros fuente de los diagramas del proyecto (por ejemplo, los archivos de *Diagrams.net*).
- `/images` (o `/img`): Contiene los ficheros de imagen (PNG, JPG) que se insertan en la memoria, como capturas de pantalla o los diagramas exportados.
- `/tex`: Contiene los ficheros `.tex` individuales para cada capítulo y apéndice de la memoria, permitiendo una organización modular del documento.
- `memoria.tex` y `anexos.tex`: Son los ficheros principales de L<sup>A</sup>T<sub>E</sub>X que estructuran y compilan el documento final.
- `bibliografia.bib`: Fichero de BibTeX que contiene todas las referencias bibliográficas citadas.

## Directorio de Configuración del Despliegue

Esta carpeta contiene los ficheros de configuración validados y necesarios para desplegar los sistemas de terceros, principalmente Jitsi. No contiene una instancia de Jitsi, sino las plantillas para configurarla a partir del repositorio oficial.

```

/configuracion_despliegue
├── /jitsi
│   ├── .gitkeep
│   ├── README.md
│   ├── docker-compose.yml
│   ├── env.example
│   ├── gen-passwords.sh
│   └── jibri.yml

```

Un desarrollador que desee replicar el entorno debe clonar el repositorio oficial de `docker-jitsi-meet` y utilizar los ficheros de esta carpeta como base para su configuración.



## Directorio de Código Fuente (‘/src’)

La carpeta `/src` es el corazón del proyecto. Contiene todo el código fuente de la aplicación de procesamiento y su fichero de orquestación con Docker Compose.

```
/src
├── /data
│   ├── /processed
│   │   ├── .gitkeep
│   │   └── .gitkeep
│   └── .gitkeep
├── /dockers
│   ├── /python_processor
│   │   ├── main.py
│   │   ├── requirements.txt
│   │   └── Dockerfile
└── docker-compose.yml
```

- `docker-compose.yml`: Es el fichero de orquestación que define y levanta los servicios del pipeline de procesamiento de datos: Kafka, Spark (master y worker) y el procesador Python.
- `/dockers/python_processor`: Contiene la lógica de la aplicación principal.
  - `Dockerfile`: Instrucciones para construir la imagen Docker del servicio.
  - `requirements.txt`: Lista de las librerías de Python necesarias para el proyecto.
  - `main.py`: El script principal que contiene la lógica para detectar, procesar y archivar los vídeos.
- `/data`: Actúa como punto de montaje principal para los datos de vídeo. Los vídeos grabados por Jibri se mapean a este directorio para que el pipeline los pueda procesar. Contiene a su vez las subcarpetas `/processed` y `/archived` para almacenar los resultados y los ficheros originales, respectivamente.

## D.3. Manual del Programador (Arquitectura del Software)

### Arquitectura Modular

El sistema se ha diseñado siguiendo un enfoque modular, separando claramente dos responsabilidades principales:

- **Sistema de Captura (Jitsi):** Se encarga exclusivamente de la captura y grabación de las sesiones de vídeo, generando los archivos MP4.
- **Pipeline de Procesamiento (Kafka/Spark):** Es un sistema independiente que se encarga de consumir y procesar dichos archivos.

Esta separación facilita el desarrollo, las pruebas y la escalabilidad, ya que cada componente puede ser modificado o sustituido sin afectar al otro.

### Arquitectura del Pipeline de Procesamiento

La Figura C.3 (incluida en el Apéndice de Diseño) ilustra la interacción entre los servicios definidos en el fichero `docker-compose.yml` de la carpeta `/src`.

- **Kafka:** Actúa como *broker* de mensajería para desacoplar el sistema. Funciona en modo KRaft, sin dependencia de Zookeeper.
- **Spark (Master/Worker):** Conforman el clúster de cómputo donde se ejecutan las tareas de procesamiento de vídeo.
- **python-processor:** Es el servicio principal que contiene la lógica de negocio. Orquesta todo el flujo: detecta los vídeos, los envía a Kafka, y consume los resultados de Spark.

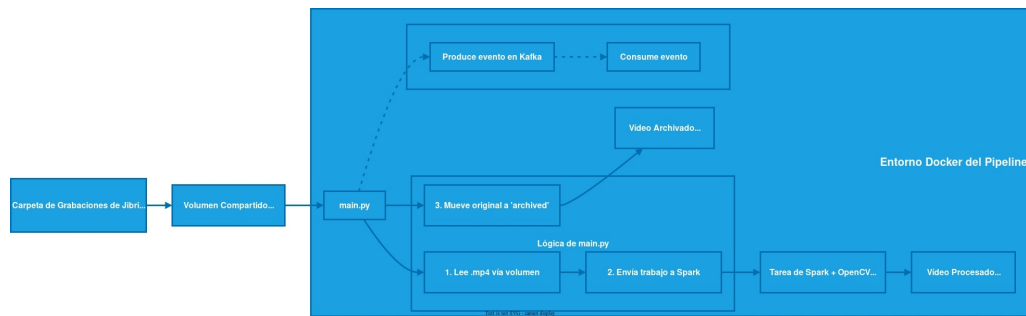


Figura D.1: Diagrama de flujo de datos del pipeline de procesamiento implementado en la carpeta `/src`.

## Descripción del Script `main.py`

El script `main.py` es el cerebro de la aplicación. Su lógica principal es la siguiente:

1. Establece una conexión con la sesión de Spark para poder enviar trabajos de procesamiento.
2. Entra en un bucle que busca el vídeo más antiguo en la carpeta de entrada (`/data`), ignorando explícitamente los subdirectorios `/processed` y `/archived`.
3. Si encuentra un vídeo, lo envía a un *topic* de Kafka y espera el resultado del procesamiento.
4. Una vez procesado, guarda el vídeo resultante y archiva el original.

```

[+] Running 6/6
  ✓ python-processor          Built
  ✓ Network src_processing_net Created
  ✓ Container src-kafka-1      Created
  ✓ Container src-spark-master-1 Created
  ✓ Container src-spark-worker-1 Created
  ✓ Container src-python-processor-1 Created
Attaching to kafka-1, python-processor-1, spark-master-1, spark-worker-1
spark-master-1 | spark 14:48:02.35 INFO ==>
spark-master-1 | spark 14:48:02.35 INFO ==> Welcome to the Bitnami spark container
spark-master-1 | spark 14:48:02.35 INFO ==> Subscribe to project updates by watching https://github.com/bitnami/containers
spark-master-1 | spark 14:48:02.35 INFO ==> Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure software supply chain features, unlimited pulls from Docker, L
TS support, or application customization, see Bitnami Premium or Tanzu Application Catalog. See https://www.arrow.com/globalecs/na/vendors/bitnami/ for more information.
spark-master-1 | spark 14:48:02.35 INFO ==>
kafka-1 | kafka 14:48:02.36 INFO ==>
kafka-1 | kafka 14:48:02.36 INFO ==> Welcome to the Bitnami kafka container
kafka-1 | kafka 14:48:02.36 INFO ==> Subscribe to project updates by watching https://github.com/bitnami/containers
kafka-1 | kafka 14:48:02.37 INFO ==> Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure software supply chain features, unlimited pulls from Docker, L
TS support, or application customization, see Bitnami Premium or Tanzu Application Catalog. See https://www.arrow.com/globalecs/na/vendors/bitnami/ for more information.
kafka-1 | kafka 14:48:02.37 INFO ==>
kafka-1 | kafka 14:48:02.37 INFO ==> ** Starting Kafka setup **
spark-worker-1 | spark 14:48:02.48 INFO ==>
spark-worker-1 | spark 14:48:02.48 INFO ==> Welcome to the Bitnami spark container
spark-worker-1 | spark 14:48:02.48 INFO ==> Subscribe to project updates by watching https://github.com/bitnami/containers
spark-worker-1 | spark 14:48:02.48 INFO ==> Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure software supply chain features, unlimited pulls from Docker, L
TS support, or application customization, see Bitnami Premium or Tanzu Application Catalog. See https://www.arrow.com/globalecs/na/vendors/bitnami/ for more information.
spark-worker-1 | spark 14:48:02.48 INFO ==>

```

Figura D.2: Salida de la terminal mostrando el despliegue de los servicios del pipeline (Kafka, Spark, etc.) con Docker Compose.

## Construcción de la Imagen Docker (Dockerfile)

El Dockerfile del servicio `python-processor` define los pasos para construir su imagen. Las decisiones clave fueron:

- Usar una imagen base oficial de Python (`python:3.9-slim`) por ser ligera y estable.
- Instalar las dependencias de sistema necesarias, como Java, que es un requisito para la comunicación con Spark.
- Instalar las librerías de Python especificadas en el fichero `requirements.txt`, fijando las versiones para garantizar la reproducibilidad y evitar conflictos (como el surgido con NumPy).

## D.4. Compilación, Instalación y Ejecución del Proyecto

### Configuración del Entorno Anfitrión

Los requisitos detallados del sistema anfitrión se encuentran en el Apéndice E. Los puntos clave son el uso de Debian como sistema operativo, la instalación de Docker y Docker Compose, y la instalación del módulo del kernel `v4l2loopback-dkms` para el correcto funcionamiento de Jibri.

## La Crónica del Despliegue de Jitsi

El despliegue de Jitsi fue uno de los mayores desafíos de ingeniería de este proyecto. El proceso de depuración se puede resumir en los siguientes hitos:

1. **Fracaso en Windows/WSL2:** El intento inicial de despliegue en este entorno falló debido a problemas insuperables de permisos de ficheros y autenticación de los servicios.
2. **Pivote a Debian:** Se tomó la decisión estratégica de migrar a un entorno Linux nativo, lo que solucionó los problemas de base.
3. **Depuración de la Conexión:** Se resolvieron sucesivamente problemas de conexión WSS, errores de montaje del fichero `custom.config.js` y fallos de autenticación de usuarios invitados en Prosody.
4. **Implementación de HTTPS:** Se descartó el uso de certificados autofirmados por problemas de confianza y se implementó una solución robusta con **Let's Encrypt**, para lo cual fue necesario configurar un dominio público con DuckDNS y re-dirección de puertos.
5. **Diagnóstico del "Vídeo en Negro":** El problema final de la falta de vídeo en las grabaciones de Jibri se diagnosticó como una necesidad de permisos elevados, solucionándose al añadir la directiva `privileged: true` al servicio.

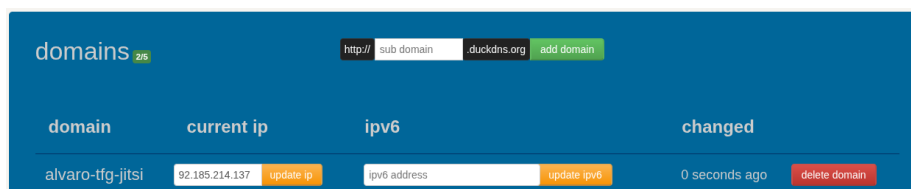


Figura D.3: Configuración del dominio público en el servicio de DNS Dinámico DuckDNS.

DHCP

NAT/PAT

DNS

UPnP

DynDNS

DMZ

NTP

ONT

Configuración de NAT/PAT/CGNAT

Estas normas son necesarias para autorizar una conexión remota desde Internet que llegue a un dispositivo específico de tu red LAN. También puedes definir los puertos(s) que utilizará esta comunicación.

Para crear la regla NAT debes introducir la IPv4 asignada a tu dispositivo en la LAN. Para saber cuál es puedes consultar el listado de IPs asignadas en la pestaña "DHCP" de esta página.

Atención: Asegúrate de que no has filtrado estos puertos en el firewall.

Personalizar reglas

estado	aplicación / servicio	puerto interno	puerto externo	protocolo	IPv4 del dispositivo	
	FTP Server	21	21	TCP	HPFBBF233(192.168.1.1)	<div>añadir</div>
<div></div>	Web Server (HTTP)	80	80	TCP	192.168.1.18	<div>editar</div> <div>borrar</div>
<div></div>	Secure Web Server (HTTPS)	443	443	TCP	192.168.1.18	<div>editar</div> <div>borrar</div>

Figura D.4: Configuración de la redirección de puertos en el router para permitir el acceso externo a Jitsi.

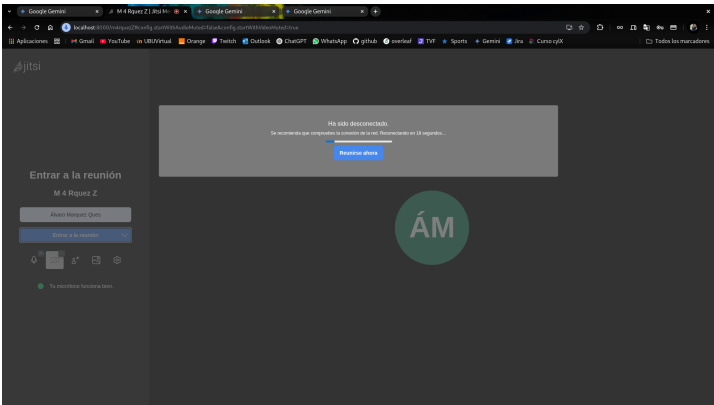
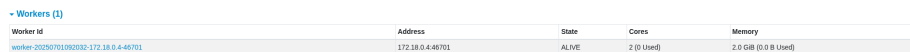


Figura D.5: Error de conexión encontrado durante la depuración de Jitsi en Debian.

## Despliegue del Pipeline de Procesamiento

El despliegue de la pila de Kafka y Spark es un proceso automatizado. Al ejecutar `docker compose up` en el directorio `/src`, todos los servicios se inician en el orden correcto. La conexión con Jitsi se realiza a través del volumen Docker compartido que mapea la carpeta de grabaciones de Jibri al directorio de entrada del pipeline.



Worker Id	Address	State	Cores	Memory
worker-202501011902002-172.18.0.4-46701	172.18.0.4:46701	ALIVE	2 (0 Used)	2.0 GiB (0.0 B Used)

Figura D.6: Captura de servicio *worker* levantado y corriendo.

## D.5. Pruebas del Sistema

### Pruebas de Integración

Se ha probado el flujo completo de extremo a extremo:

1. Se inicia una conferencia en Jitsi y se graba.
2. Se verifica que Jibri genera y guarda correctamente el archivo MP4.
3. Se ejecuta el pipeline de procesamiento.
4. Se comprueba en los *logs* que el *script* detecta el vídeo, lo procesa y lo archiva.
5. Se verifica la existencia y el contenido del vídeo procesado en la carpeta de salida.

Estas pruebas han validado que todos los componentes se comunican correctamente entre sí.

### Pruebas Funcionales (Prueba de Concepto)

Para demostrar la funcionalidad del procesamiento, se ha implementado un algoritmo simple que invierte los colores de cada fotograma. La Figura [D.9](#) muestra una comparación entre un fotograma del vídeo original y el mismo fotograma del vídeo procesado, donde se puede apreciar visualmente el éxito de la transformación aplicada.

```

alvarodebian-tfg-alvaro:~/Documentos/GitHub/TFG-SISTEMA-PROCESAMIENTO-VIDEO/src$ docker compose up -d
[+] Running 5/5
 ✓ Network src_processing_net      Created
 ✓ Container src-spark-master-1    Started
 ✓ Container src-kafka-1          Started
 ✓ Container src-spark-worker-1    Started
 ✓ Container src-python-processor-1 Started
alvarodebian-tfg-alvaro:~/Documentos/GitHub/TFG-SISTEMA-PROCESAMIENTO-VIDEO/src$ docker compose logs -f python-processor
python-processor-1 | WARNING: Using incubator modules: jdk.incubator.vector
python-processor-1 | Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
python-processor-1 | Setting default log level to "WARN".
python-processor-1 | To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
python-processor-1 | 25/07/03 16:32:09 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
python-processor-1 | ¡Conexión con Spark establecida con éxito!
python-processor-1 | =====
python-processor-1 | Procesando el video más antiguo: /app/data/61988609-c033-4431-970c-4ee1e1881e9/testtfg_2025-06-23-13-26-00.mp4
python-processor-1 | Aplicando transformación simple...
python-processor-1 | Procesados 670 frames.
python-processor-1 | Video procesado guardado en: /app/data/processed/procesado_testtfg_2025-06-23-13-26-00.mp4
python-processor-1 | Sesión de Spark detenida.
python-processor-1 | =====
python-processor-1 | python-processor-1 exited with code 0

```

Figura D.7: Logs del contenedor `python-processor` que muestran la ejecución exitosa del pipeline completo.

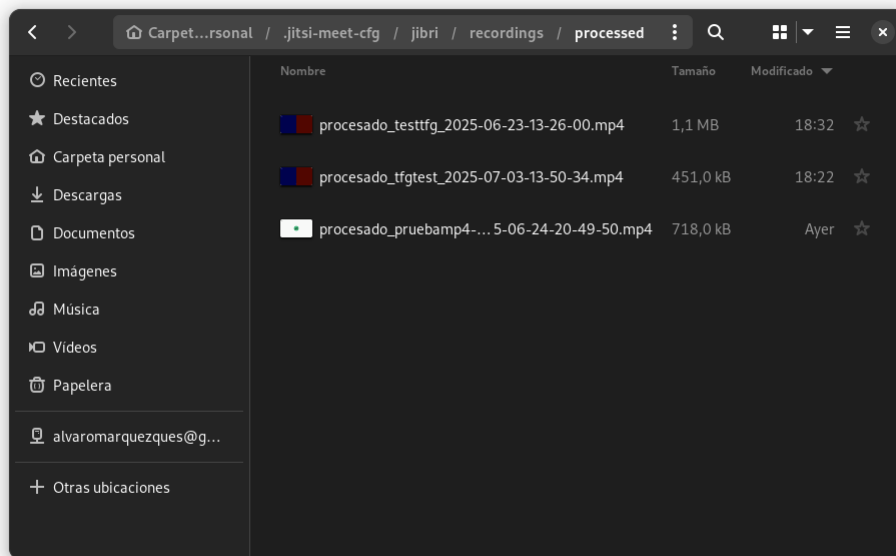


Figura D.8: Directorio de salida con el fichero de vídeo ya procesado.



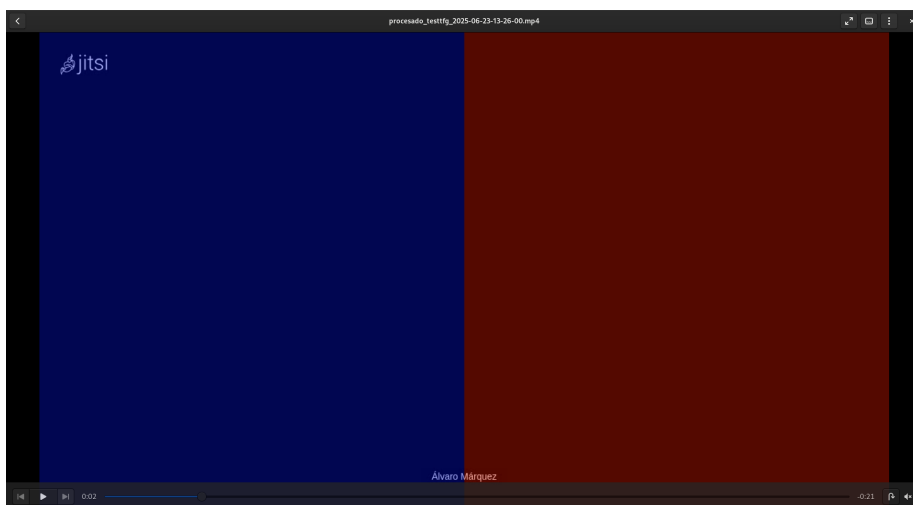


Figura D.9: Captura de video ya procesado.

## D.6. Fallos y Soluciones

En este apartado se exponen algunos de los errores y desafíos técnicos más relevantes por los que se ha visto comprometido el proyecto a lo largo de su desarrollo. Se detalla para cada uno el síntoma observado, el diagnóstico técnico de la causa raíz y la solución final que se implementó.

### Fallo 1: Inestabilidad Crítica del Despliegue en Entorno Windows (WSL2)

- **Síntoma:** Inoperabilidad total de la pila `docker-jitsi-meet`. El *log* del contenedor Prosody (servidor de autenticación) mostraba errores críticos y repetitivos de `Permission denied`. El contenedor Jibri no lograba arrancar o entraba en un bucle de reinicios, mostrando en sus logs errores de autenticación `SASLError`.
- **Diagnóstico Técnico:** Se determinó que existía un conflicto irresoluble en la traducción de permisos entre el sistema de ficheros de Windows (NTFS) y el de Linux (ext4) que usan los contenedores. La capa de virtualización de WSL2 no gestionaba adecuadamente los permisos de propietario/grupo (`chown/chgrp`) que los servicios de Jitsi necesitaban para operar.

- **Solución Aplicada:** Abandono completo del entorno Windows. Se realizó una instalación nativa de Debian 12, proporcionando un entorno de sistema de ficheros y permisos 100 % compatible, lo que se considera la solución estándar para despliegues de servidor robustos.

## Fallo 2: Despliegue de Jitsi en Debian - Errores de Conexión en Cascada

- **Síntoma:** Tras un despliegue exitoso en Debian, la interfaz web no permitía unirse a una sala, mostrando el error "La conexión ha fallado". La depuración reveló múltiples problemas encadenados: el cliente web intentaba una conexión segura (WSS) en un servidor HTTP, un posterior error de montaje `not a directory` al intentar forzar la configuración con `custom.config.js`, y finalmente un error de "Ha sido desconectado" al entrar en una sala.
- **Diagnóstico Técnico:** Se diagnosticaron tres problemas independientes: 1) una configuración por defecto del cliente que forzaba WSS; 2) la creación accidental de un directorio en lugar de un fichero de configuración en el `host`; y 3) la falta de un `VirtualHost` en la configuración de Prosody para la autenticación de usuarios invitados.
- **Solución Aplicada:** Se solucionó cada problema de forma secuencial: se creó correctamente el fichero `custom.config.js` para forzar la conexión HTTP y, crucialmente, se modificó el fichero de configuración `jitsi-meet.cfg.lua` de Prosody para añadir la configuración del `VirtualHost` de invitados.

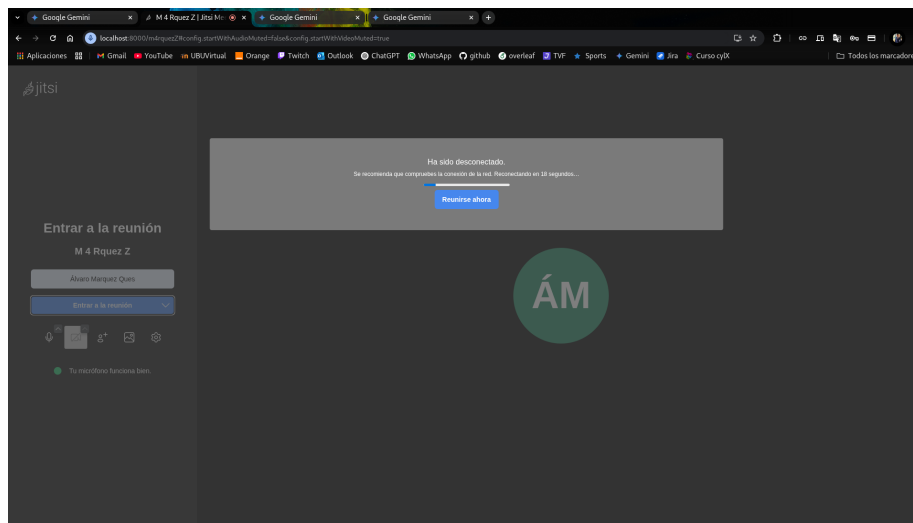


Figura D.10: Capture de error persistente de Conexión.

### Fallo 3: Problemas de Red para la Validación de Certificados SSL

- **Síntoma:** Al intentar implementar HTTPS con Let's Encrypt, el proceso de validación de certificados fallaba.
- **Diagnóstico Técnico:** Se determinó que los servidores de Let's Encrypt no podían alcanzar la máquina anfitriona en los puertos 80 y 443. Esto se debía a la naturaleza dinámica de la IP doméstica y a la falta de configuración en el *router* local.
- **Solución Aplicada:** Se implementó una configuración de red completa: 1) Se configuró un DNS dinámico con **DuckDNS**. 2) Se asignó una IP local estática a la máquina Debian mediante reserva de DHCP en el router. 3) Se configuró la re-dirección de puertos (*Port Forwarding*) en el *router* para dirigir el tráfico de los puertos 80 y 443 a la IP fija de la máquina Debian.

### Fallo 4: Grabaciones de Jibri con Vídeo en Negro

- **Síntoma:** El servicio Jibri generaba un fichero **.mp4** del tamaño correcto, pero este estaba completamente en negro y sin sonido.

- **Diagnóstico Técnico:** El análisis de los *logs* de Jibri ((EE) no screens found(EE)) reveló que el entorno gráfico virtual (Xvfb) no podía iniciarse. La causa raíz fue la falta del módulo del kernel `v4l2loopback` en el sistema anfitrión Debian, necesario para crear un dispositivo de vídeo virtual del cual grabar.
- **Solución Aplicada:** Se realizó la instalación del módulo requerido en el *host* con el comando `sudo apt install v4l2loopback-dkms` y se reinició el sistema.

## Fallo 5: Errores Durante la Construcción de una Imagen Docker Personalizada

- **Síntoma:** Al intentar construir una imagen de Jibri personalizada para depurar, el proceso `docker build` fallaba con errores de `apt-get update`.
- **Diagnóstico Técnico:** Se identificaron dos problemas en el `Dockerfile` base: 1) una clave GPG expirada de un repositorio de Google Chrome y 2) la referencia a un paquete obsoleto (`libindicator3-7`).
- **Solución Aplicada:** Se modificó el `Dockerfile` para eliminar el fichero de repositorio problemático antes de ejecutar `apt-get update` y se reemplazó el paquete obsoleto por su sucesor, `libayatana-appindicator3-1`.

## Fallo 6: Conflictos y Errores en el Pipeline de Spark

- **Síntoma:** Al levantar el pipeline de procesamiento, surgieron múltiples errores: un conflicto de puertos (8080), errores de importación en Python ('ModuleNotFoundError', 'numpy.core.multiarray failed to import'), una excepción 'java.io.InvalidClassException', una condición de carrera.<sup>a</sup>l conectar con Spark y un bucle de procesamiento infinito.
- **Diagnóstico Técnico:** Se diagnosticaron cinco problemas distintos: 1) Conflicto del puerto del Spark Master con el Jitsi Videobridge. 2) Falta de un fichero `requirements.txt` y un conflicto de versiones entre OpenCV y NumPy. 3) Incompatibilidad entre la versión de la librería PySpark y la imagen Docker de Spark. 4) El *script* de Python intentaba conectar antes de que el máster de Spark estuviera listo. 5) El *script* no tenía lógica para gestionar los vídeos ya procesados.

- **Solución Aplicada:** Se aplicaron soluciones específicas para cada problema: 1) Se cambió el puerto del Spark Master al 8081. 2) Se creó un `requirements.txt` fijando las versiones (`pyspark==3.5.0`, `numpy<2.0`). 3) Se fijó la versión de la imagen de Spark a (`bitnami/spark:3.5.0`). 4) Se añadió un retardo de 10 segundos (`time.sleep(10)`) al inicio del *script*. 5) Se implementó la lógica de mover el vídeo original a una carpeta `/archived` tras su procesamiento.



---

## Documentación de Usuario

---

### E.1. Introducción

Este manual proporciona una guía detallada para el despliegue y uso de la infraestructura de procesamiento de vídeo desarrollada en este TFG. El objetivo es ofrecer al usuario técnico (en adelante, el Administrador) todos los pasos necesarios para poner en marcha el sistema completo.

La infraestructura consta de dos sistemas modulares que se despliegan de forma independiente pero que trabajan de manera conjunta: un servidor de captura de vídeo (Jitsi) y un pipeline de procesamiento de datos (Kafka y Spark).

### E.2. Requisitos Previos del Sistema

Antes de comenzar, es fundamental asegurar que la máquina anfitriona (*host*) cumple con los siguientes requisitos.

#### Requisitos de *Hardware*

- **CPU:** Se recomienda un procesador de 4 o más núcleos con soporte para virtualización (VT-x/AMD-V) habilitado en la BIOS/UEFI.
- **Memoria RAM:** Debido a la cantidad de servicios, se recomienda un mínimo de **16 GB de RAM**.
- **Almacenamiento:** Mínimo de 50 GB de espacio libre en disco.

## Requisitos de Software del Sistema Anfitrión (*Host*)

- **Sistema Operativo:** Se requiere una distribución de Linux. Todo el desarrollo y validación se ha realizado sobre **Debian 12 "Bookworm"**.
- **Software Base:** Es necesario tener instalados los siguientes paquetes a través del gestor de paquetes **apt**:
  - **git**: para la clonación de los repositorios.
  - **docker.io**: motor de contenedores Docker.
  - **docker-compose-v2**: herramienta de orquestación (*plugin* de Docker).
  - **curl** y **openssl**: herramientas de diagnóstico de red.
- **Módulo del Kernel para Jibri:** Para que el servicio de grabación Jibri funcione correctamente y no genere vídeos con pantalla en negro, es **crítico** instalar el siguiente módulo del kernel que crea un dispositivo de vídeo virtual:

```
sudo apt update
sudo apt install v4l2loopback-dkms
```

Tras la instalación, se recomienda reiniciar el sistema anfitrión.

## Requisitos de Red (para Jitsi)

Para que el servidor Jitsi sea accesible desde internet y pueda obtener un certificado SSL/TLS válido, se necesita:

- Un **nombre de dominio público**. Para este proyecto se utilizó un dominio gratuito de **DuckDNS**.
- Acceso al **router** para configurar la **re-dirección de puertos** (*Port Forwarding*) de los puertos TCP 80, TCP 443 y UDP 10000 hacia la IP privada de la máquina anfitriona.

(Recomendado) Asignar una dirección IP local estática a la máquina anfitriona en la configuración del *router* (Reserva de DHCP). Esto asegura que la re-dirección de puertos no falle si la IP local del servidor cambia.




DHCP	NAT/PAT	DNS	UPnP	DynDNS	DMZ	NTP	ONT
------	---------	-----	------	--------	-----	-----	-----

**Configuración de NAT/PAT/CGNAT**

Estas normas son necesarias para autorizar una conexión remota desde Internet que llegue a un dispositivo específico de tu red LAN. También puedes definir los puertos(s) que utilizará esta comunicación.

Para crear la regla NAT debes introducir la IPv4 asignada a tu dispositivo en la LAN. Para saber cuál es puedes consultar el listado de IPs asignadas en la pestaña "DHCP" de esta página.

 Atención: Asegúrate de que no has filtrado estos puertos en el firewall.

Personalizar reglas						
estado	aplicación / servicio	puerto interno	puerto externo	protocolo	IPv4 del dispositivo	
	FTP Server	21	21	TCP	HPFBF233(192.168.1.1)	<b>añadir</b>
	Web Server (HTTP)	80	80	TCP	192.168.1.18	<b>editar</b> <b>borrar</b>
	Secure Web Server (HTTPS)	443	443	TCP	192.168.1.18	<b>editar</b> <b>borrar</b>

Figura E.1: Captura de configuración correcta personalizada puertos.

domain	current ip	ipv6	changed
alvaro-tfg-jitsi	92.185.214.137 <b>update ip</b>	ipv6 address <b>update ipv6</b>	0 seconds ago <b>delete domain</b>

Figura E.2: Captura de servicio Duckdns configurado.

## E.3. Instalación del Sistema

La instalación se divide en dos fases: el despliegue del servidor Jitsi y, posteriormente, el despliegue del pipeline de procesamiento.

### Fase 1: Despliegue del Servidor de Captura (Jitsi)

1. Clonar el repositorio oficial de `docker-jitsi-meet`:

```
git clone https://github.com/jitsi/docker-jitsi-meet.git
cd docker-jitsi-meet
```

2. **Copiar y configurar el fichero de entorno:** Se debe copiar el fichero de ejemplo y editarlo para ajustar los parámetros a nuestro entorno.

```
cp env.example .env
nano .env
```

Dentro del fichero `.env`, es crucial configurar las siguientes variables:

- `PUBLIC_URL`: El dominio público que se utilizará (ej. `https://mi-tfg.duckdns.org`).
  - `LETSENCRYPT_DOMAIN`: El mismo nombre de dominio.
  - `LETSENCRYPT_EMAIL`: Un correo electrónico para la gestión de los certificados Let's Encrypt.
  - Habilitar Jibri y la grabación: `ENABLE_RECORDING=1`, `ENABLE_LIVESTREAMING=0`.
3. **Generar contraseñas:** Se ejecuta el *script* proporcionado para generar todas las contraseñas internas de los servicios.

```
./gen-passwords.sh
```

4. **Crear directorios de configuración persistente:**

```
mkdir -p ~/.jitsi-meet-cfg/{web/letsencrypt,transcripts,prosody \
/config,prosody/prosody-plugins-custom,jicofo,jvb,jigasi,jibri}
```

**Nota Importante sobre la Configuración Web:** Es crucial verificar que dentro del directorio `/.jitsi-meet-cfg/web/` no se cree accidentalmente una carpeta/archivo llamada `"custom.config.js"`. Si fuera necesario modificar la configuración del cliente, se deberá crear un fichero de texto con ese nombre.

5. **Levantar los servicios de Jitsi:** Se utiliza el siguiente comando para iniciar todos los contenedores, incluyendo Jibri.

```
docker compose -f docker-compose.yml -f jibri.yml up -d
```

## Fase 2: Despliegue del Pipeline de Procesamiento

1. **Clonar el repositorio del TFG:** En una carpeta diferente, se clona el repositorio principal del proyecto.

```
git clone https://github.com/alvarom Marquez1002/ \
TFG-SISTEMA-PROCESAMIENTO-VIDEO.git
cd TFG-SISTEMA-PROCESAMIENTO-VIDEO/src
```

2. **Configurar el volumen de datos:** Es importante verificar que en el fichero `docker-compose.yml` de esta carpeta, el volumen del servicio `python-processor` esté correctamente mapeado al directorio donde Jibri guarda las grabaciones (por defecto, `~/.jitsi-meet-cfg/jibri/recordings`). (Opcional) Para realizar pruebas de desarrollo sin depender de Jitsi, se puede comentar la línea del volumen de Jibri en el fichero `docker-compose.yml` y activar una línea alternativa como `./data:/app/data` para usar vídeos de prueba locales.
3. **Levantar los servicios del pipeline:** Desde la carpeta `/src`, se ejecuta el siguiente comando. La opción `-build` asegura que la imagen Docker del procesador se construya con el código más reciente.

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

## E.4. Manual de Uso

Una vez desplegados ambos sistemas, el flujo de trabajo para capturar y procesar una sesión es el siguiente.

### Realizar una Grabación con Jitsi

1. **Acceder a la conferencia:** El Terapeuta y el Paciente acceden a la URL del servidor Jitsi (ej. <https://mi-tfg.duckdns.org>) desde sus navegadores web.

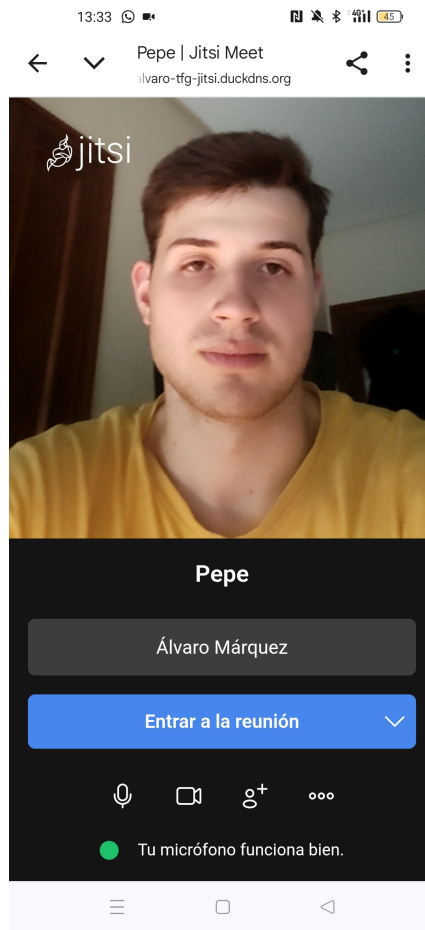


Figura E.3: Pantalla de inicio de Jitsi Meet donde el usuario inicia la conferencia.

2. **Iniciar la grabación:** Una vez en la sala, el Terapeuta hace clic en el botón de "Más acciones"(tres puntos verticales) y selecciona la opción "Iniciar grabación". Se le pedirá confirmación para iniciar la sesión.

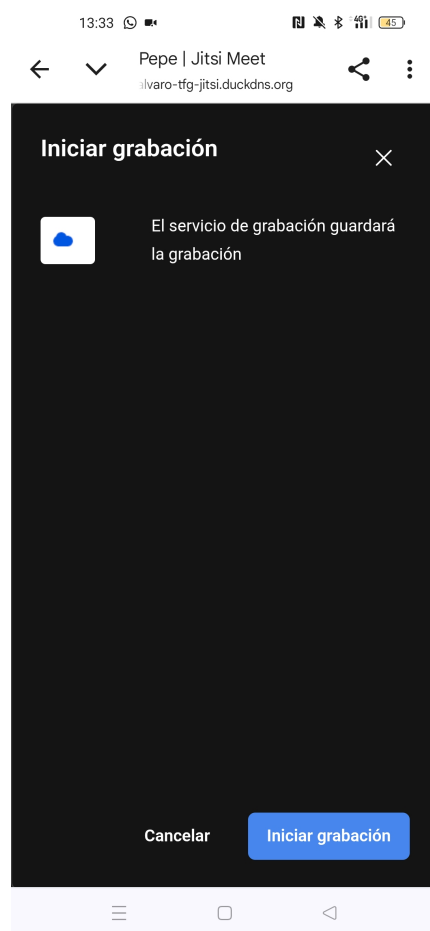


Figura E.4: Menú de opciones para iniciar la grabación de la sesión.

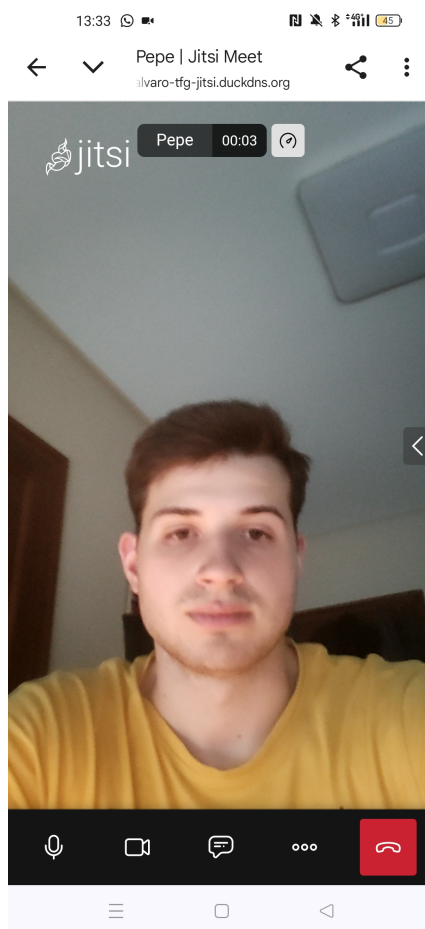


Figura E.5: Indicador visual *REC* que muestra que la sesión se está grabando activamente.

3. **Detener la grabación:** Para finalizar, el Terapeuta repite el proceso y selecciona "Detener grabación".

## Verificar el Vídeo Grabado

Tras detener la grabación, el servicio Jibri tardará unos instantes en procesar y guardar el fichero.

1. **Localizar el fichero:** El archivo MP4 resultante se guardará en el directorio del sistema anfitrión:  
~/.jitsi-meet-cfg/jibri/recordings/.

2. **Verificar contenido:** El Administrador puede reproducir este fichero para confirmar que el audio y el vídeo se han grabado correctamente. Este fichero es la entrada para el pipeline de procesamiento.

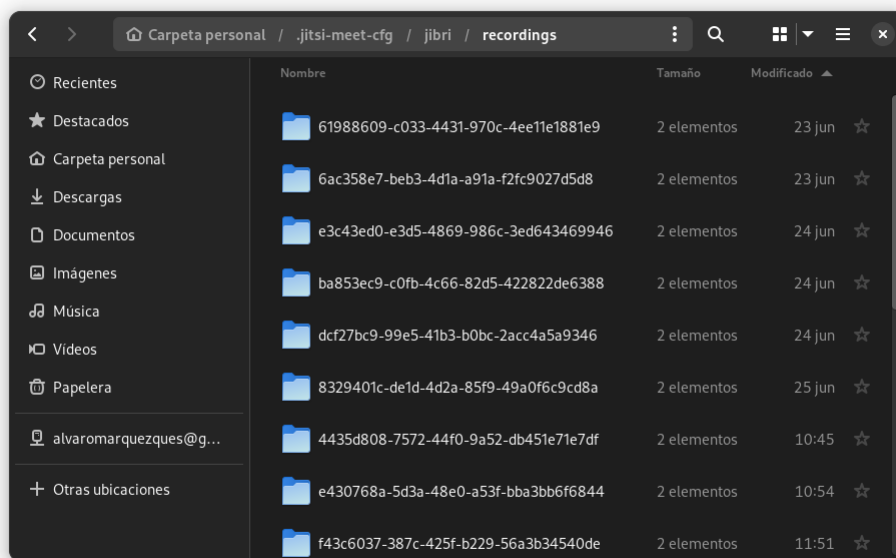


Figura E.6: Ejemplo del directorio de grabaciones de Jibri con el fichero MP4 resultante.

## Ejecutar y Monitorizar el Procesamiento

El pipeline de procesamiento está diseñado para ejecutarse de forma automática y continua.

1. **Ejecución automática:** El *script* `main.py` dentro del contenedor `python-processor` está constantemente buscando nuevos vídeos en el directorio de entrada.
2. **Monitorizar el proceso:** Para ver el progreso en tiempo real, el Administrador puede ejecutar el siguiente comando en la terminal, desde la carpeta `/src` del proyecto:

```
docker compose logs -f python-processor
```

En la salida se podrán observar mensajes indicando qué vídeo se está procesando.

```

alvaro@debian-tfg-alvaro:~/Documentos/GitHub/TFG-SISTEMA-PROCESAMIENTO-VIDEO/src$ docker compose up -d
[+] Running 5/5
 ✓ Network src_processing_net      Created
 ✓ Container src-spark-master-1    Started
 ✓ Container src-kafka-1          Started
 ✓ Container src-spark-worker-1    Started
 ✓ Container src-python-processor-1 Started
alvaro@debian-tfg-alvaro:~/Documentos/GitHub/TFG-SISTEMA-PROCESAMIENTO-VIDEO/src$ docker compose logs -f python-processor
python-processor-1 WARNING: Using incubator modules: jdk.incubator.vector
python-processor-1 Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
python-processor-1 Setting default log level to "WARN".
python-processor-1 To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
python-processor-1 25/07/03 16:32:09 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
python-processor-1 ¡Conexión con Spark establecida con éxito!
python-processor-1 =====
python-processor-1 Procesando el video más antiguo: /app/data/61988609-c033-4431-970c-4ee11e1881e9/testtfg_2025-06-23-13-26-00.mp4
python-processor-1 Aplicando transformación simple...
python-processor-1 Procesados 670 frames.
python-processor-1 Video procesado guardado en: /app/data/processed/procesado_testtfg_2025-06-23-13-26-00.mp4
python-processor-1 Sesión de Spark detenida.
python-processor-1 =====
python-processor-1 python-processor-1 exited with code 0

```

Figura E.7: Monitorización de los logs del procesador Python, confirmando la ejecución del pipeline.

## Acceder al Resultado Final

Una vez que el *log* indica que el procesamiento ha finalizado, los resultados se pueden encontrar en las siguientes rutas (relativas a la carpeta del proyecto). El vídeo procesado aparecerá en la sub-carpeta `/processed` y el vídeo original será movido a la sub-carpeta `/archived`, ambas dentro del directorio donde se guardó originalmente la grabación (por defecto, `/.jitsi-meet-cfg/jibri/recordings/`):

- **Vídeo procesado:** El nuevo vídeo con la transformación aplicada se encontrará en `/src/data/processed/`.
- **Vídeo original archivado:** El vídeo original que ha sido procesado se moverá a `/src/data/archived/` para evitar su procesamiento de nuevo.



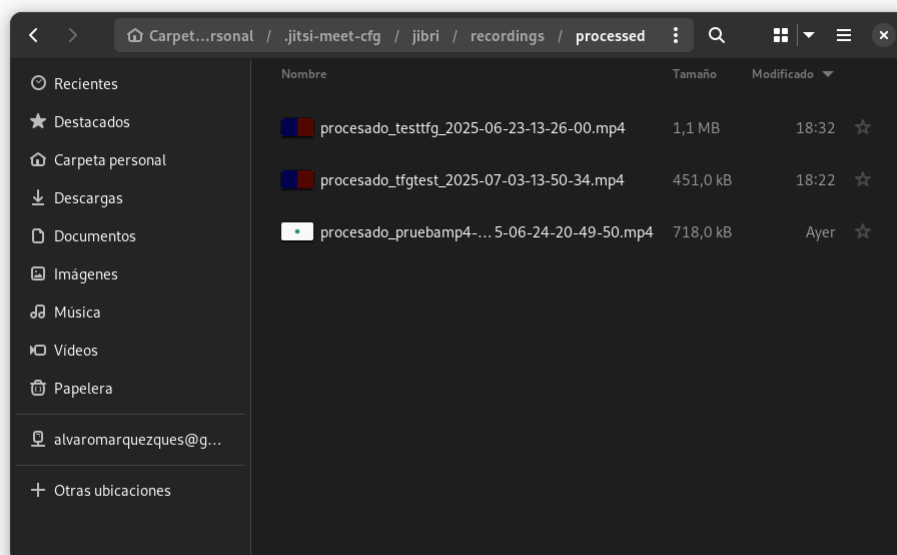


Figura E.8: Directorio de salida mostrando el vídeo final una vez procesado.

## E.5. Resolución de Problemas (FAQ)

- **Problema:** No se puede acceder a la instancia de Jitsi desde internet.
- **Causa y Solución:** Verificar que el servicio de DNS dinámico (DuckDNS) está apuntando a la IP pública correcta y que la re-dirección de puertos (80, 443) está bien configurada en el *router*.
- **Problema:** El contenedor `python-processor` falla al iniciar con un error de conexión a Spark.
- **Causa y Solución:** Esto puede ser una condición de carrera". El *script* intenta conectar con Spark antes de que el clúster esté completamente listo. El *script* incluye un retardo para mitigar esto, pero si el problema persiste, se puede probar a reiniciar únicamente el contenedor del procesador: `docker compose restart python-processor`.
- **Problema:** El *log* del contenedor `python-processor` muestra un error "ModuleNotFoundError." un conflicto de versiones con NumPy.

- **Causa y Solución:** Esto se debe a una incompatibilidad entre las librerías de Python. Asegúrese de que el fichero `requirements.txt` del procesador especifica las versiones correctas y compatibles (`pyspark==3.5.0`, `numpy<2.0`) y reconstruya la imagen con el comando `"docker compose up -build"`.
- **Problema:** Al intentar ejecutar un contenedor de Jitsi (especialmente Jibri) aparece un error `"s6-mkdir: permission denied"` incluso usando `sudo`.
- **Causa y Solución:** Este es un problema avanzado causado por una incompatibilidad entre el sistema de inicio `s6-overlay` de la imagen de Docker y el entorno del *host*. La solución implementada en este TFG fue construir una imagen de Jibri personalizada desde un Dockerfile limpio para evitar el conflicto. Consulte la documentación técnica (Apéndice D) para más detalles sobre esta implementación.

---

## **Anexo de Sostenibilización Curricular**

---

### **F.1. Introducción**

La ingeniería, y en particular la ingeniería informática, tiene un profundo impacto en la sociedad y en el entorno. Por ello, considero que la formación de un ingeniero no puede limitarse únicamente al dominio técnico, sino que debe incorporar una visión crítica y responsable sobre las implicaciones de su trabajo. Este anexo es una reflexión personal sobre cómo los principios del Desarrollo Sostenible, entendido como la capacidad de satisfacer las necesidades actuales sin comprometer las de las generaciones futuras, se han manifestado en mi formación y, de manera específica, en el desarrollo de este Trabajo de Fin de Grado.

A lo largo de mi formación, he adquirido una serie de competencias que me han permitido abordar este proyecto no solo como un desafío técnico, sino también como una oportunidad para aplicar soluciones tecnológicas a problemas reales con una perspectiva social, ambiental y ética. Siguiendo las directrices propuestas por la CRUE [2], a continuación detallo cómo he aplicado estas competencias en mi TFG.

## F.2. Competencias de Sostenibilidad Aplicadas al TFG

### SOS1: Contextualización Crítica del Conocimiento

Esta competencia se refiere a la capacidad de interrelacionar el conocimiento técnico con la problemática social, económica y ambiental. Desde el inicio, este TFG se ha enmarcado en un contexto social muy claro: el desafío que suponen las enfermedades neurodegenerativas, como el Párkinson, para nuestro sistema de salud. Mi trabajo no ha sido un ejercicio técnico aislado, sino que he tenido que comprender las necesidades reales de los pacientes y los terapeutas. Entender que la tecnología que desarrollo puede mejorar la calidad de vida de personas con movilidad reducida, facilitando su acceso a terapias de rehabilitación y reduciendo la carga sobre sus familias, me ha permitido contextualizar mi labor como ingeniero y darle un propósito que va más allá del código. He aprendido que la tecnología solo tiene sentido si responde a una necesidad humana real.

### SOS2: Utilización Sostenible de Recursos y Prevención de Impactos

A primera vista, un proyecto de software puede parecer alejado del concepto de sostenibilidad ambiental, pero la realidad es que el uso de recursos computacionales tiene un impacto directo en el consumo energético. En este sentido, he aplicado esta competencia en varias decisiones clave:

- **Eficiencia de la Arquitectura:** La elección de una arquitectura de microservicios y, en particular, el uso de Jitsi con su tecnología SFU (Selective Forwarding Unit), no es solo una decisión técnica. Una arquitectura SFU es mucho más eficiente en el uso de CPU que las alternativas más antiguas (MCU), lo que se traduce en un menor consumo energético del servidor.
- **Uso de Código Abierto:** Mi decisión de basar todo el proyecto en herramientas de software libre y de código abierto (FOSS) como Linux, Docker, Kafka, Spark y Jitsi, es en sí misma una forma de utilización sostenible de los recursos. En lugar de desarrollar cada componente desde cero, he aprovechado el conocimiento y el trabajo de una comunidad global, contribuyendo a un ecosistema de conocimiento compartido y evitando la duplicación innecesaria de esfuerzos.

- **Impacto Ambiental de la Telerehabilitación:** El propio objetivo del sistema tiene una dimensión de sostenibilidad ambiental. Al facilitar las terapias a distancia, se reduce la necesidad de desplazamientos de pacientes y terapeutas, lo que conlleva una disminución directa de la huella de carbono asociada al transporte.

### SOS3: Participación en Procesos Comunitarios

Esta competencia se refiere a la capacidad de colaborar y participar en la comunidad. He aplicado este principio de varias maneras. En primer lugar, mi TFG se integra en un proyecto colaborativo con una entidad pública como es el HUBU. En segundo lugar, al basar mi trabajo en el TFM y TFG de compañeros de la universidad, he participado en un proceso comunitario de creación de conocimiento dentro de mi propio grupo de investigación. He construido sobre el trabajo de otros y, a su vez, esta memoria y el código que la acompaña servirán como base para futuros estudiantes, creando una cadena de conocimiento.

### SOS4: Aplicación de Principios Éticos

La ética profesional es un pilar fundamental de la ingeniería. En este proyecto, he tenido que tomar decisiones con implicaciones éticas claras:

- **Privacidad y Seguridad de los Datos:** Soy consciente de que el sistema manejará datos de salud extremadamente sensibles. La decisión de pivotar hacia una implementación con HTTPS utilizando certificados de una autoridad de confianza como Let's Encrypt no es solo un requisito técnico, sino una obligación ética para garantizar la confidencialidad e integridad de la información de los pacientes.
- **Licenciamiento y Conocimiento Abierto:** La decisión de publicar mi propio código bajo una licencia permisiva como la MIT, GNU responde al principio ético de compartir el conocimiento y permitir que otros puedan beneficiarse de mi trabajo, mejorarlo y adaptarlo, especialmente al tratarse de un proyecto con una clara vocación social.

En definitiva, este TFG ha sido para mí un ejercicio práctico de cómo la ingeniería informática puede y debe ser una herramienta para contribuir a un desarrollo más sostenible y justo.



---

## Bibliografía

---

- [1] Atlassian. Jira software, 2025. Disponible en: <https://www.atlassian.com/software/jira>.
- [2] Comisión de Calidad Ambiental, Desarrollo Sostenible y Prevención de Riesgos (CADEP-CRUE). Directrices para la introducción de la sostenibilidad en el curriculum. Technical report, Conferencia de Rectores de las Universidades Españolas (CRUE), 2012. Documento revisado y presentado en la Asamblea General de la CRUE de 28 de junio de 2012.
- [3] Ticjob.es. Informe anual sobre el empleo en el sector tic. <https://www.ticjob.es/>, 2024. Consultado para estimaciones de salarios en el sector tecnológico en España.
- [4] Manuel Trigás Gallego. Metodología scrum, 2012.