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 — 6 de julio de 2025
Tutor: D. José Miguel Ramírez Sanz y D. José Luís Garrido Labrador

Índice general

Índice de figuras

Índice de tablas

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* [**?**]. 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** [**?**]. Aunque las reuniones con los tutores eran periódicas para comentar dudas y avances, la estructura en sprints permitio un desarrollo iterativo y organizado.

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.

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 LaTeX 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.) [2]	2.100,00
Seguridad Social a cargo de la empresa (aprox. 31 %)	651,00
Coste mensual por empleado	**2.751,00**
Total 6 meses (1 empleado)	**16.506,00**

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
Total	**3.200,00**	**400,00**

Tabla A.6: Costes de *hardware*.

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

Tabla A.7: Costes de servicios.

Concepto	Coste (€)
Personal (6 meses)	16.506,00
Hardware (amortizado 6 meses)	400,00
Servicios (6 meses)	300,00
Coste total estimado del proyecto	**17.206,00**

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
Debian 12	GPL y otras FOSS
Docker	Apache 2.0
Apache Kafka	Apache 2.0
Apache Spark	Apache 2.0
Jitsi Meet	Apache 2.0
Python	Python Software Foundation License
OpenCV-Python	GNU v3.0 License
NumPy	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 **??**, 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.

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.1 y por otra, las tablas de casos de uso.

Figura B.1: Diagrama de casos de uso.

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

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

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

Descripción	Permite procesar un vídeo grabado en una sesión de Jitsi para obtener un resultado analizado.	
Requisitos	RF.2.1, RF.2.2, RF.3.1, RF.3.2, RF.3.3, RF.4.1, RF.4.2, RF.4.3	
Precondiciones	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.	
Secuencia Normal	**Paso**	**Acción**
	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.
Postcondiciones	El vídeo ha sido procesado y el resultado se encuentra en el directorio de salida. El vídeo original ha sido archivado.	
Excepciones	No hay vídeos nuevos en el directorio de entrada; el script finaliza sin realizar ninguna acción.	
Importancia	Alta	
Urgencia	Alta	

Apéndice C

Especificación de diseño

C.1. Introducción

C.2. Diseño de datos

C.3. Diseño arquitectónico

C.4. Diseño procedimental

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

Estructura de directorios

La gestión del código fuente y la documentación de mi TFG se realiza a través del sistema de control de versiones Git, alojado en el repositorio público de GitHub: https://github.com/alvaromarquez1002/TFG-SISTEMA-PROCESAMIENTO-VIDEO

La estructura principal de directorios en la raíz del repositorio es la siguiente:

`src/` Contiene el código fuente del proyecto.

`doc/` Contiene los ficheros fuente LaTeX para la Memoria y Anexos (incluyendo imágenes en `img/` y secciones en `tex/`).

`.gitignore` Especifica los archivos y directorios ignorados por Git.

`README.md` Descripción general del proyecto.

D.3. Manual del programador

D.4. Compilación, instalación y ejecución del proyecto

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Apéndice F

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 [1], 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] 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.

[2] 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.