



Universidad Politécnica
de Madrid



**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Interfaz Gráfica para Control de
Sesiones de Dispositivo Médico con
Raspberry Pi**

Autor: Jesús Quirante Domínguez

Tutor(a): Nazario Félix González

Madrid, junio de 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Interfaz Gráfica para Control de Sesiones de Dispositivo Médico con Raspberry Pi

Junio 2025

Autor: Jesús Quirante Domínguez

Tutor:

Nazario Félix González

Arquitectura y Tecnología de Sistemas Informáticos

ETSI Informáticos

Universidad Politécnica de Madrid

Resumen

Este Trabajo Fin de Grado surge de la necesidad de mejorar la interfaz de usuario del dispositivo médico MINESTIM APCM-01, utilizado en el tratamiento de la fibromialgia mediante estimulación magnética transcraneal de baja intensidad. El sistema original, basado únicamente en indicadores LED y un botón físico, presentaba importantes limitaciones en cuanto a usabilidad y retroalimentación al usuario, dificultando su operación en entornos clínicos.

Para abordar este problema, se ha desarrollado una solución integral que combina tecnologías modernas en un sistema para Raspberry Pi. La arquitectura propuesta consta de tres componentes principales: un frontend desarrollado con Electron que proporciona una interfaz gráfica intuitiva, un backend en Python que gestiona toda la lógica de control, y un servidor WebSocket basado en Socket.IO que permite la comunicación en tiempo real entre ambos módulos. Esta estructura modular facilita tanto el desarrollo como el mantenimiento futuro del sistema, no obstante, cada uno de estos componentes presentaron un desafío como, por ejemplo, la comunicación serial y la gestión de pines GPIO.

La interfaz gráfica implementada ofrece numerosas ventajas sobre el sistema original. Permite visualizar claramente el estado del dispositivo, el tiempo restante de sesión y el número de sesiones disponibles, dando lugar a una mejor visión integral de la información. Además, incorpora funcionalidades avanzadas como la gestión de parámetros mediante comunicación serial, la detección de errores a través de los pines GPIO, y la actualización persistente de configuraciones. Todo ello con un diseño centrado en la usabilidad para el personal médico.

Los resultados obtenidos demuestran que la solución desarrollada supera ampliamente las limitaciones del sistema original. Las pruebas realizadas con un simulador basado en ESP8266 han confirmado el cumplimiento de todos los requisitos funcionales y no funcionales establecidos, incluyendo usabilidad, fiabilidad y rendimiento. Asimismo, la interfaz gráfica ha demostrado ser particularmente efectiva para reducir errores operativos y mejorar la experiencia del usuario final.

Abstract

This Final Degree Project arises from the need to improve the user interface of the MINESTIM APCM-01 medical device, which is used to treat fibromyalgia through low-intensity transcranial magnetic stimulation. Based solely on LED indicators and one physical button, the original system presented significant limitations regarding usability and user feedback, making its use difficult in clinical environments.

To address this issue, a comprehensive solution has been developed that integrates modern technologies into a system for Raspberry Pi. The proposed architecture consists of three main components: an Electron-developed frontend that provides an intuitive graphical interface, a Python backend that manages all control logic, and a WebSocket server based on Socket.IO that enables real-time communication between both modules. This modular structure facilitates both development and future maintenance of the system. Each of these components, however, posed a challenge, such as serial communication and GPIO pin management.

The implemented graphical interface offers numerous advantages over the original system. It displays the device's status clearly, as well as remaining session time and the number of available sessions, providing a more comprehensive view of the information. Additionally, it incorporates advanced features such as parameter management via serial communication, error detection through GPIO pins, and persistent configuration updates—all with a design focused on usability for medical personnel.

The results achieved demonstrate that the developed solution significantly overcomes the limitations of the original system. Tests conducted using an ESP8266-based simulator confirmed compliance with all defined functional and non-functional requirements, including usability, reliability, and performance. Furthermore, the graphical interface has proven particularly effective in reducing operational errors and improving the end-user experience.

Tabla de contenidos

1	Introducción	1
2	Desarrollo	4
2.1	Sistema Actual	4
2.1.1	Descripción del Producto	4
2.1.2	Modo de Empleo Actual y Estados	4
2.1.3	Limitaciones del Sistema Actual	5
2.1.3.1	Limitaciones en la retroalimentación al usuario	6
2.1.3.2	Dificultad en el uso intuitivo	6
2.2	Sistema Objetivo	6
2.2.1	Requisitos Funcionales	6
2.2.2	Requisitos No Funcionales	7
2.3	Arquitectura de la Aplicación	7
2.3.1	Diagrama de Arquitectura General	8
2.3.1.1	Componentes del sistema	8
2.3.2	Frontend (Electron)	9
2.3.2.1	Diseño de la Interfaz	9
2.3.2.2	Árbol de directorios	12
2.3.2.3	Comunicación con Backend	13
2.3.2.4	Gestión de Estados	14
2.3.3	Backend (Python)	19
2.3.3.1	Árbol de directorios	20
2.3.3.2	Servidor Socket.IO	21
2.3.3.3	Comunicación Serial	21
2.3.3.4	Control de GPIO	23
2.3.3.5	Cliente Backend	25
2.3.4	Hardware	27
2.3.4.1	Configuración Raspberry Pi	28
2.3.4.2	Proceso de Compilación y Ejecución Automática	28
3	Resultados y conclusiones	30
4	Análisis de Impacto	32
5	Bibliografía	34
6	Anexos	36
	TFG\package.json	36
	TFG\requirements.txt	38
	TFG\src\renderer\styles.css	39
	TFG\src\renderer\renderer.js	52
	TFG\src\renderer\index.html	59
	TFG\src\main\preload.js	61

TFG\src\main\main.js	62
TFG\src\main\utils\clientSocketIO.js	70
TFG\resources\data.json.....	73
TFG\python\Simulator.py.....	73
TFG\python\serial_escucha.py	76
TFG\python\server\serverSIO.py.....	77
TFG\python\client\started_ClientSIO.py.....	81
TFG\python\client\serialEvent.py	84
TFG\python\client\GPIO_Event.py	90
TFG\python\client\clientSocketIO.py	95
TFG\python\client\commandHandlers\socketIOCommandHandler.py	97
TFG\python\client\commandHandlers\serialCommandHandler.py.....	102
TFG\python\client\commandHandlers\jsonCommandHandler.py	106
TFG\ESP8266\Emisor_receptor.ino	108
TFG\Help-files\autostart.desktop	110
TFG\Help-files\compiler	111
TFG\Help-files\CreateVenv	112
TFG\Help-files\start	112

1 Introducción

La fibromialgia es un trastorno crónico caracterizado por dolor musculoesquelético generalizado, fatiga persistente, alteraciones del sueño y déficits cognitivos, lo que afecta significativamente la calidad de vida de los pacientes. Aunque su etiología sigue sin estar completamente definida, se asocia a una sensibilización del sistema nervioso central que amplifica la percepción del dolor. Actualmente, su tratamiento combina enfoques farmacológicos (como analgésicos y antidepresivos) y no farmacológicos (ejercicio moderado, terapia psicológica), así como dispositivos médicos especializados, como el MINESTIM APCM-01, que utiliza estimulación magnética transcraneal de baja intensidad (Li-rTMS) para aliviar los síntomas asociados a esta condición.

El MINESTIM es un dispositivo médico compuesto por un generador de señales, un aplicador (que se coloca en la cabeza del paciente), una base de carga y una fuente de alimentación. Su funcionamiento actual depende de un sistema basado en Arduino, que controla las sesiones terapéuticas mediante señales enviadas a través de los pines GPIO, con una interfaz de usuario limitada a indicadores luminosos (LEDs) y un botón físico para iniciar/detener el tratamiento. Esta configuración, aunque funcional, presenta importantes limitaciones:

- Ausencia de una interfaz gráfica intuitiva, lo que dificulta su operación y la interpretación de los estados del dispositivo.
- Falta de registro digital de sesiones, imposibilitando el seguimiento clínico y la integración con sistemas hospitalarios.

Ante estas restricciones, este Trabajo de Fin de Grado (TFG) propone el desarrollo de una aplicación de escritorio para Raspberry Pi que modernice el sistema, incorporando:

- Una interfaz gráfica amigable diseñada para personal sanitario, con información clara sobre el estado del dispositivo, sesiones restantes y duración del tratamiento.
- Comunicación serial con el hardware (reemplazando el Arduino actual) para la carga de las sesiones terapéuticas.

El objetivo principal es mejorar la usabilidad y eficiencia del MINESTIM, facilitando su adopción en entornos clínicos sin requerir una curva de aprendizaje compleja. Para ello, se emplearán tecnologías como Electron (para

la interfaz), Python (backend de control) y WebSockets (comunicación en tiempo real entre frontend usando Electron y backend).

Electron es un framework de código abierto desarrollado por GitHub que permite la creación de aplicaciones de escritorio multiplataforma utilizando tecnologías web como HTML, CSS y JavaScript. Este ofrece múltiples ventajas, entre ellas la posibilidad de desarrollar una única base de código que funcione en Windows, macOS y Linux, facilitando así la escalabilidad a futuro si se quisiera y reduciendo costos de desarrollo. Además, su integración con Chromium proporciona herramientas avanzadas de desarrollo y una experiencia de usuario fluida. Aunque Electron puede presentar desafíos como un mayor consumo de memoria y tamaño de los ejecutables, su flexibilidad y facilidad de desarrollo lo convierten en una opción ideal para aplicaciones médicas que requieren una interfaz intuitiva y accesible. Se destaca que se ha consolidado como una tecnología clave en el desarrollo de aplicaciones de escritorio modernas, permitiendo que plataformas ampliamente utilizadas como Microsoft Teams, Visual Studio Code, Discord, WhatsApp Desktop y Slack funcionen correctamente en múltiples sistemas operativos. Su uso extendido en productos tecnológicos de alto impacto demuestra su versatilidad y fiabilidad para la creación de software intuitivo y accesible.

Uno de los aspectos más llamativos es su naturaleza de código abierto, lo que permite a la comunidad de desarrolladores contribuir activamente a su evolución. Esto garantiza un ciclo de vida prolongado, con actualizaciones constantes que mejoran el rendimiento y la seguridad, asegurando que las aplicaciones creadas con este framework se mantengan alineadas con las necesidades del usuario y los avances tecnológicos.

El backend de control se desarrollará en Python, un lenguaje ampliamente utilizado en el ámbito de la programación y la automatización, lo que lo convierte en una opción ideal para su implementación en Raspberry Pi, ya que está en constante actualización y desarrollo. Es especialmente adecuado para este tipo de dispositivos debido a su compatibilidad con Raspberry Pi OS, que incluye un entorno de desarrollo preconfigurado. Otro aspecto clave es la extensa comunidad de desarrolladores que han ido estableciendo una gran cantidad de bibliotecas específicas para Raspberry Pi, facilitando la interacción con los pines GPIO, como RPi.GPIO. De este modo, se permite que Python sea utilizado para el control de hardware de manera eficiente, sin necesidad de recurrir a lenguajes de bajo nivel como C o ensamblador. Además de su compatibilidad con frameworks de comunicación como WebSockets, dando lugar a que sea una opción robusta para la gestión del dispositivo médico.

Para la comunicación en tiempo real entre el frontend y el backend, tal y como se mencionó anteriormente se empleará WebSockets, un protocolo que permite la transmisión bidireccional de datos con baja latencia y menor sobrecarga en comparación con las solicitudes HTTP tradicionales. WebSockets es ideal para

aplicaciones médicas que requieren actualizaciones instantáneas, como el monitoreo de sesiones terapéuticas y la interacción en tiempo real con el dispositivo. Su capacidad para mantener una conexión abierta entre el cliente y el servidor garantiza una comunicación eficiente y escalable, mejorando la experiencia del usuario y la fiabilidad del sistema.

Este proyecto no solo busca modernizar un dispositivo médico existente, sino también contribuir al tratamiento de la fibromialgia mediante una solución tecnológica accesible, segura y adaptable a las necesidades reales de los profesionales de la salud. Los resultados esperados incluyen una reducción de errores operativos y una experiencia optimizada tanto para médicos como para pacientes.

2 Desarrollo

Este capítulo presenta el desarrollo de una interfaz gráfica para el dispositivo médico MINESTIM APCM-01, destinado al tratamiento de la fibromialgia. Se detallan los requisitos, la arquitectura del sistema y la implementación técnica, incluyendo el frontend (Electron), el backend (Python) y la comunicación en tiempo real mediante WebSockets.

2.1 Sistema Actual

El dispositivo actual presenta un diseño funcional que satisface los requisitos básicos de operación. Sin embargo, tras un análisis exhaustivo, en esta sección se hace hincapié en tres aspectos fundamentales: las características técnicas del producto, su protocolo de uso y los diferentes estados operativos. Adicionalmente, se identificarán y evaluarán las limitaciones inherentes al sistema actual, lo que permitirá sentar las bases para posibles mejoras en su interfaz.

Este enfoque metodológico permitirá comprender tanto las capacidades como las restricciones del dispositivo en su configuración presente, proporcionando una visión completa de su operatividad y áreas de oportunidad para su desarrollo futuro.

2.1.1 Descripción del Producto

El MINESTIM APCM-01 es un dispositivo médico de estimulación magnética transcraneal de baja intensidad (Li-rTMS), diseñado específicamente para el tratamiento del dolor asociado a la fibromialgia. Desarrollado por el Centro de Tecnología Biomédica (CTB - UPM), el equipo genera campos magnéticos pulsados de muy baja intensidad (menos de 10 μ T a 8 Hz), evitando superar los límites de seguridad establecidos por la Organización Mundial de la Salud (OMS) por un amplio margen. El sistema incluye un dispositivo generador (con batería interna recargable), un dispositivo aplicador, una base de carga y una fuente de alimentación. Los parámetros de estimulación están preconfigurados en el fabricante, con una señal de pulsos cuadrados, una intensidad de corriente de 0-800 μ A, una frecuencia fija de 8 Hz y una duración de sesión de 20 minutos. Su uso está restringido a profesionales sanitarios y requiere la prescripción de un médico especialista.

2.1.2 Modo de Empleo Actual y Estados

El dispositivo MINESTIM APCM-01 presenta una interfaz de usuario minimalista basada en indicadores luminosos y acústicos para su operación.

Cuenta con dos LEDs (verde y rojo) y una señal sonora como sistema de feedback, junto con un único botón físico multipropósito que gestiona tanto el inicio como la interrupción de las sesiones. En su funcionamiento, el dispositivo transita por diferentes estados operativos claramente diferenciados: en modo de espera (LED verde parpadeando cada 2 segundos con zumbido inicial), durante la carga de sesiones (ambos LEDs encendidos), en tratamiento activo (LED verde parpadeando rápidamente a 2Hz), en estado de error (LED rojo intermitente acompañado de 5 tonos de alarma cuando detecta fallos de conexión), y en situación de desconexión durante sesión (LED rojo parpadeante con 6 zumbidos de alerta). El apagado completo se indica mediante la ausencia de cualquier indicación luminosa. Este diseño de estados proporciona una comunicación clara del estado del dispositivo al operador, aunque la simplicidad de la interfaz limita las posibilidades de interacción avanzada o personalización durante el tratamiento.

Estado	Indicador LED	Sonido	Descripción
Encendido (Modo espera)	LED verde parpadea 1 vez/2 seg	Zumbido inicial	El dispositivo está listo para iniciar una sesión.
Carga de sesiones	LED verde y rojo	Ninguno	Modo carga de sesiones
Sesión activa	LED verde parpadea 2 veces/seg	Ninguno (solo al inicio/fin)	El tratamiento está en curso (20 min). El LED del aplicador también parpadea.
Error (fallo conexión)	LED rojo parpadea 1 vez/2 seg (10 seg)	5 tonos de alarma (sirena)	El aplicador no está conectado o hay un fallo.
Desconexión durante sesión	LED rojo parpadea + 6 zumbidos	Alarma intermitente	Si el aplicador se desconecta y no se reconecta en 60 seg, el dispositivo pasa a modo de espera con error.
Apagado	Todos los LEDs apagados	Ninguno	El dispositivo se apaga manualmente tras finalizar su uso.

2.1.3 Limitaciones del Sistema Actual

El sistema actual presenta varias limitaciones significativas en comparación con un sistema equipado con una interfaz gráfica (GUI). Estas restricciones afectan tanto a la usabilidad como a la flexibilidad del dispositivo:

2.1.3.1 Limitaciones en la retroalimentación al usuario

Los indicadores luminosos y sonoros solo pueden comunicar estados operativos básicos (encendido, sesión activa o error), sin ofrecer datos esenciales como el tiempo restante de la sesión en curso. Esta falta de información detallada se puede ver perjudicada ante situaciones de error, ya que un simple LED rojo parpadeante puede corresponder a diversos estados, obligando al operador a consultar el manual físico para realizar un diagnóstico adecuado. Adicionalmente, el sistema carece por completo de un indicador que muestre el número de sesiones restantes disponibles, lo que dificulta la planificación de los tratamientos y la carga de sesiones. Con ello se reduce significativamente la usabilidad del dispositivo y aumentan la dependencia de documentación externa para su correcta operación, ralentizando todo el proceso.

2.1.3.2 Dificultad en el uso intuitivo

El sistema de interfaz basado exclusivamente en LEDs y señales acústicas presenta una curva de aprendizaje más pronunciada para los profesionales, ya que deben memorizar y reconocer los diferentes patrones de parpadeo y secuencias de tonos asociados a cada estado operativo, en lugar de contar con unas instrucciones claras y explícitas que podrían proporcionar una pantalla gráfica. La ausencia de confirmaciones visuales ante acciones críticas, como la interrupción de sesiones, incrementa el margen de error operativo y agrava la limitación, al facilitar la ejecución accidental de comandos no deseados. De este modo, el personal médico requiere un mayor entrenamiento previo para poder hacer uso del instrumento.

2.2 Sistema Objetivo

El sistema objetivo busca superar las limitaciones del dispositivo actual mediante la implementación de una interfaz gráfica de usuario (GUI) que mejore la usabilidad, la retroalimentación en tiempo real y la personalización del tratamiento. A continuación, se detallan los requisitos funcionales y no funcionales que guiarán el rediseño del sistema.

2.2.1 Requisitos Funcionales

Deberá garantizar una gestión integral de las sesiones de tratamiento, proporcionando información clara y accesible para el usuario. En primer lugar, la interfaz requerirá incluir un temporizador visible que indique el tiempo restante de la sesión en curso, mostrando una cuenta regresiva desde los 20 minutos predefinidos. También, será necesario visualizar el número de sesiones disponibles en el dispositivo, permitiendo así al profesional sanitario planificar su uso de manera eficiente. De igual forma, el sistema debe ofrecer la posibilidad

de iniciar, pausar o reanudar una sesión. Por último, se incorporará una función de apagado solicitando confirmación visual antes de ejecutar dicha acción, reduciendo de este modo la probabilidad de apagados accidentales.

En lo que respecta a la retroalimentación en tiempo real, la interfaz gráfica deberá mostrar de manera intuitiva el estado operativo del dispositivo, diferenciando entre los modos de inicio, espera, activo, error y carga de sesiones. Para facilitar la identificación rápida de problemas, se mostrará un mensaje textual en la pantalla que indique un error como la desconexión o la falta de sesiones cargadas.

Finalmente, la configuración del dispositivo se gestionará mediante comunicación serial, permitiendo la actualización de parámetros esenciales, el número de serie y la cantidad de sesiones disponibles.

2.2.2 Requisitos No Funcionales

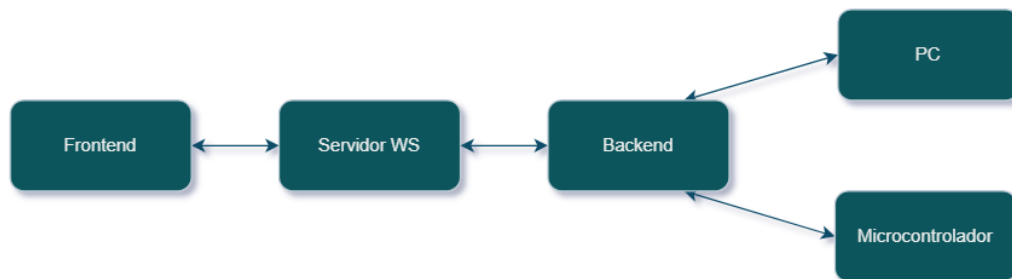
Los requisitos no funcionales del sistema establecen criterios esenciales de calidad, rendimiento y seguridad que garantizarán su eficacia en entornos clínicos. Referente a usabilidad, la interfaz debe ser intuitiva, con una jerarquía visual clara que priorice información crítica como el tiempo restante de sesión, y fácil empleo para el personal sanitario, incluyendo una pantalla legible bajo diversas condiciones de iluminación. Respecto a la fiabilidad, el sistema debe responder en menos de un segundo a las acciones del usuario y contar con tolerancia a fallos, permitiendo la recuperación automática ante errores de conexión del hardware. Finalmente, en relación con el rendimiento, el sistema debe garantizar compatibilidad con actualizaciones de estados y una comunicación eficiente con el microcontrolador, asegurando un funcionamiento estable y sincronizado en todo momento.

2.3 Arquitectura de la Aplicación

En este apartado se presenta un análisis detallado de la arquitectura técnica del sistema, describiendo cada uno de sus componentes fundamentales, las tecnologías seleccionadas para su implementación y los criterios que justifican dicha elección. El estudio abarca tanto las características técnicas de cada elemento como los protocolos de comunicación establecidos entre ellos, proporcionando una visión integral de la estructura del sistema.

2.3.1 Diagrama de Arquitectura General

A continuación, se mostrará un diagrama general de la comunicación existente entre los diferentes componentes del sistema:



2.3.1.1 Componentes del sistema

El sistema propuesto sigue una arquitectura modular compuesta por tres componentes principales que trabajan de forma coordinada para ofrecer una solución completa al problema planteado. Esta estructura permite una clara separación de responsabilidades y facilita tanto el desarrollo como el mantenimiento de la aplicación.

El componente frontend, desarrollado con Electron, constituye la capa de presentación del sistema. Esta solución tecnológica permite implementar una interfaz que cumple con múltiples funciones esenciales: renderiza todos los elementos visuales, gestiona eficientemente las interacciones del usuario y muestra en tiempo real los estados del dispositivo. La elección de Electron.js resulta particularmente adecuada para este proyecto, ya que facilita la creación de una interfaz optimizada para la interacción táctil. El diseño de esta capa se ha realizado considerando especialmente los requisitos de usabilidad establecidos para la aplicación.

El backend constituye el núcleo de procesamiento del sistema, desarrollado en Python por su eficiencia en aplicaciones de control y su amplio ecosistema de bibliotecas especializadas. Este componente implementa toda la lógica de control de la aplicación, aprovechando herramientas como PySerial para la comunicación serial con el dispositivo médico, RPi.GPIO para la gestión de los pines de la Raspberry Pi, y Socket.IO para establecer la conexión con el servidor Web Socket. La comunicación serial será necesaria para establecer enlace con un ordenador a través del puerto UART con el fin de cargar sesiones, cambiar parámetros y obtener o modificar el número de serie del dispositivo. Por otro lado, la interacción con el microcontrolador se realizará con los pines GPIO de entrada y salida.

Como elemento integrador del sistema, el servidor WebSocket actúa como pasarela de comunicación entre el frontend y el backend. Esta solución tecnológica establece un canal de comunicación bidireccional y en tiempo real, fundamental para garantizar la sincronización instantánea entre la interfaz de usuario y la lógica de control.

2.3.2 Frontend (Electron)

Electron, al estar basado en Node.js y tecnologías web como HTML, CSS y JavaScript, permite un desarrollo ágil y familiar para los programadores frontend. Su versatilidad no solo brinda un alto rendimiento en aplicaciones de escritorio, sino que también facilita una futura migración o adaptación a una versión web, reduciendo significativamente el esfuerzo de desarrollo gracias a la reutilización de código. Esto lo convierte en una opción estratégica para proyectos que buscan escalabilidad y flexibilidad a largo plazo.

Una vez definido el motivo de la elección de Electron para el desarrollo del frontend, se explicará el diseño de la interfaz, la comunicación con el backend y la gestión de estados.

2.3.2.1 Diseño de la Interfaz

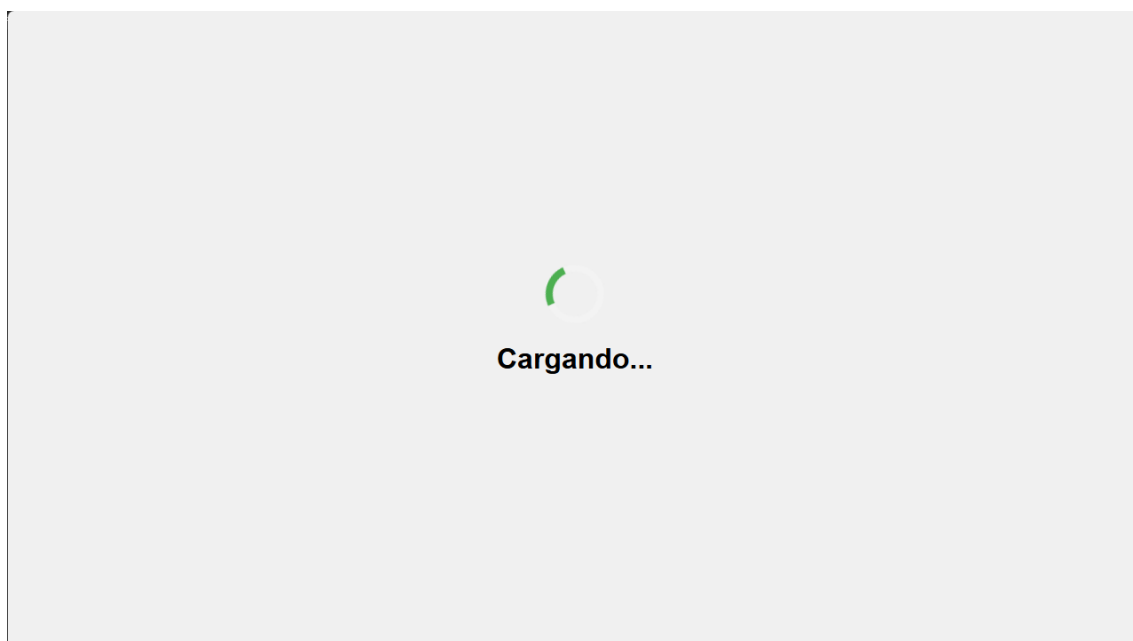
Desde un principio se estableció que la interfaz estuviera constituida por una única pantalla que fuera reactiva con un botón que sirve para iniciar, parar y continuar una sesión. En esta pantalla se debe incluir otro botón adicional para apagar el dispositivo y se deben mostrar los parámetros de la sesión, el número de sesiones y el número de serie del dispositivo.

El diseño final de la pantalla se muestra en la siguiente imagen:



En las esquinas se mostrarán los parámetros e información para la interpretación de la interfaz del siguiente modo: En la sección superior izquierda se muestra el número de sesiones disponibles, en la parte superior derecha los diferentes códigos de colores para controlar el estado a modo de leyenda, en la esquina inferior izquierda el número de serie y en la inferior derecha los parámetros de la sesión. Más adelante, en el punto 2.3.2.4, se indicará más en profundidad cómo visualizar el estado actual.

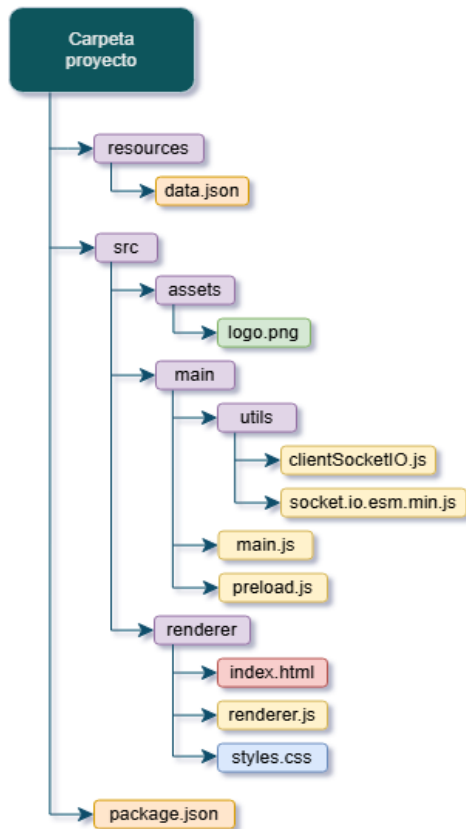
Por último, cabe destacar que existen dos pantallas auxiliares que se utilizan en puntos concretos del uso de la interfaz. En primer lugar, una ventana de carga que se mostrara hasta que la aplicación esté disponible para su funcionamiento pleno.



Y como último medio auxiliar, una ventana emergente al pulsar el botón de apagar, evitando apagados involuntarios en el uso de la aplicación.



2.3.2.2 Árbol de directorios



En la imagen adjunta se puede observar la estructura de directorios que conforma el módulo frontend de la aplicación. El proyecto sigue una organización jerárquica clara, centrada en el directorio principal *src/*, que contiene todos los componentes esenciales para el funcionamiento de la interfaz.

El directorio *assets/* está diseñado para albergar los recursos estáticos necesarios para la aplicación, incluyendo imágenes. Actualmente, únicamente se guarda el logo, no obstante, en un futuro podría utilizarse para agregar imágenes, fuentes tipográficas y otros elementos multimedia que complementen la experiencia de usuario.

Dentro de la arquitectura principal, la carpeta *main/* contiene los archivos fundamentales para la ejecución de

Electron:

- *main.js*: Constituye el punto de entrada de la aplicación, responsable de inicializar el proceso principal de Electron y gestionar el ciclo de vida de la ventana.
- *preload.js*: Actúa como puente de comunicación entre el proceso principal (*main*) y el proceso de renderizado (*renderer*), permitiendo la carga segura de módulos específicos.

El subdirectorio *utils/* incluye dos componentes críticos para la comunicación en tiempo real:

- *clientSocketIO.js*: Gestiona la conexión con el servidor Socket.IO y facilita el envío y recepción de mensajes.

- `socket.io.esm.min.js`: Representa la versión minificada de la librería Socket.IO, que se carga directamente en el `index.html` a través de la sección `<head>` para habilitar la comunicación WebSocket.

La capa de presentación se encuentra en el directorio `renderer/`, compuesto por:

- `index.html`: Archivo base que define la estructura HTML de la aplicación.
- `renderer.js`: Contiene la lógica principal del proceso de renderizado.
- `styles.css`: Alberga todos los estilos CSS que definen la apariencia visual de la interfaz.

Por último, el archivo `package.json` completa la estructura, conteniendo la configuración del proyecto y las dependencias necesarias para la correcta instalación y ejecución de Electron. Este archivo especifica los paquetes requeridos, scripts de ejecución y metadatos del proyecto, sirviendo como base para el sistema de construcción.

2.3.2.3 Comunicación con Backend

Como ya se mencionó anteriormente, para garantizar una sincronización en tiempo real entre el frontend y el backend, se utiliza un servidor WebSocket como intermediario, empleando la librería Socket.IO. Este flujo permite que las acciones del usuario desencadenen actualizaciones de estado, las cuales son procesadas por el backend y reflejadas dinámicamente en la interfaz. A continuación, se detalla el proceso paso a paso:

Frontend → Servidor WebSocket:

- El usuario hace clic en un botón de acción en el frontend.
- El frontend envía el estado actual de la aplicación al servidor WebSocket usando Socket.IO (`socket.emit`).

Servidor WebSocket → Backend:

- El servidor WebSocket recibe el mensaje y lo reenvía al cliente backend (que también está conectado vía Socket.IO).

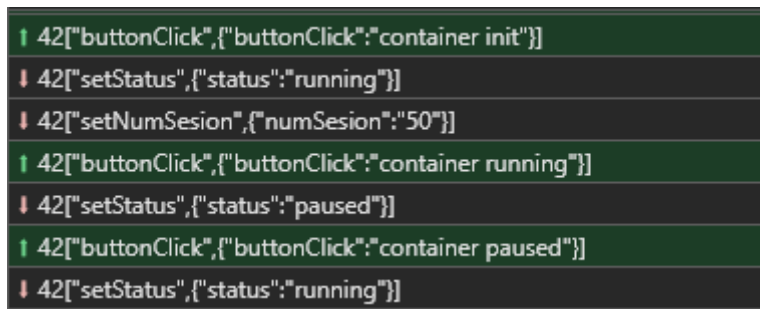
Backend → Servidor WebSocket:

- El backend procesa el estado recibido, genera un nuevo estado y lo envía de vuelta al servidor WebSocket.

Servidor WebSocket → Frontend:

- El servidor WebSocket reenvía el nuevo estado al frontend.
- El frontend actualiza su interfaz según el nuevo estado.

En la figura de debajo se puede observar la comunicación que se realiza en la parte del frontend donde las líneas de color verde representan los mensajes enviados al servidor para el backend y las otras representan los mensajes recibidos desde el servidor enviados por el backend.



El diagrama muestra una secuencia de mensajes entre un frontend (representado por líneas verdes) y un backend (representado por líneas grises). Los mensajes son:

- Frontend → Backend: `42["buttonClick",{"buttonClick":"container init"}]`
- Backend → Frontend: `42["setStatus",{"status":"running"}]`
- Backend → Frontend: `42["setNumSesion",{"numSesion":"50"}]`
- Frontend → Backend: `42["buttonClick",{"buttonClick":"container running"}]`
- Backend → Frontend: `42["setStatus",{"status":"paused"}]`
- Frontend → Backend: `42["buttonClick",{"buttonClick":"container paused"}]`
- Backend → Frontend: `42["setStatus",{"status":"running"}]`

2.3.2.4 Gestión de Estados

Como se expuso con anterioridad, la aplicación opera bajo cinco estados posibles: inicial, en ejecución, pausa, error y carga de sesiones. Para identificar de manera intuitiva el estado actual, se ha implementado un sistema visual basado en colores, representado por un borde que rodea toda la interfaz y cambia según el estado activo. Una leyenda ubicada en la esquina superior derecha define el código de colores asociado a cada estado, permitiendo al usuario reconocer de forma inmediata en qué fase se encuentra la aplicación. Por ejemplo, un borde rojo indicaría un estado de error, mientras que uno verde correspondería a en ejecución. Este diseño garantiza una retroalimentación clara y accesible en todo momento.

Estado inicial

Al iniciar cada sesión, la aplicación siempre arrancará en el estado inicial, representado visualmente por un color azul. Este tono no solo identifica el estado actual, sino que también se refleja en el botón de acción, el cual adoptará el mismo color azul para indicar que el sistema está listo y disponible para comenzar. Una vez pulsado el botón de empezar, el número de sesiones disponibles disminuirá en uno.



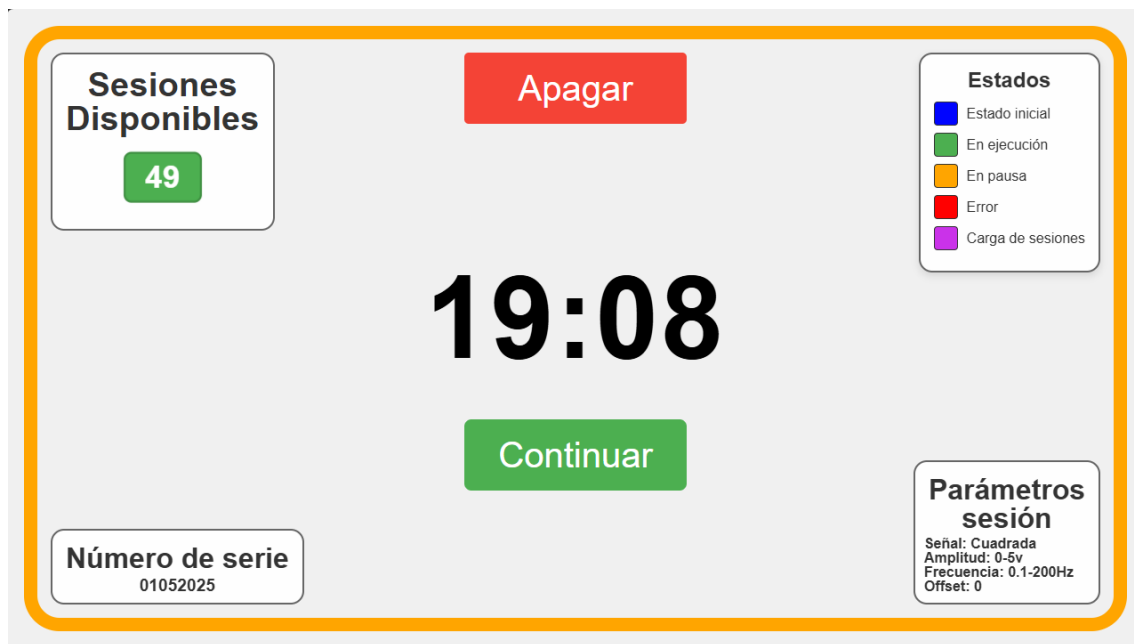
Estado en ejecución

El estado en ejecución se identifica visualmente mediante un color verde, señalando que la sesión está activa y operativa, por lo que la cuenta atrás situada en la parte central estará en funcionamiento. Para llegar a este estado, el usuario puede pulsar el botón de empezar (desde el estado inicial) o el botón de continuar (desde el estado de pausa). Durante este modo, el botón de acción mostrará el texto "Pausar" en color amarillo, indicando que, al pulsarlo, la aplicación pasará al estado de pausa. Este diseño asegura una navegación intuitiva y una retroalimentación visual clara en todo momento.



Estado en pausa

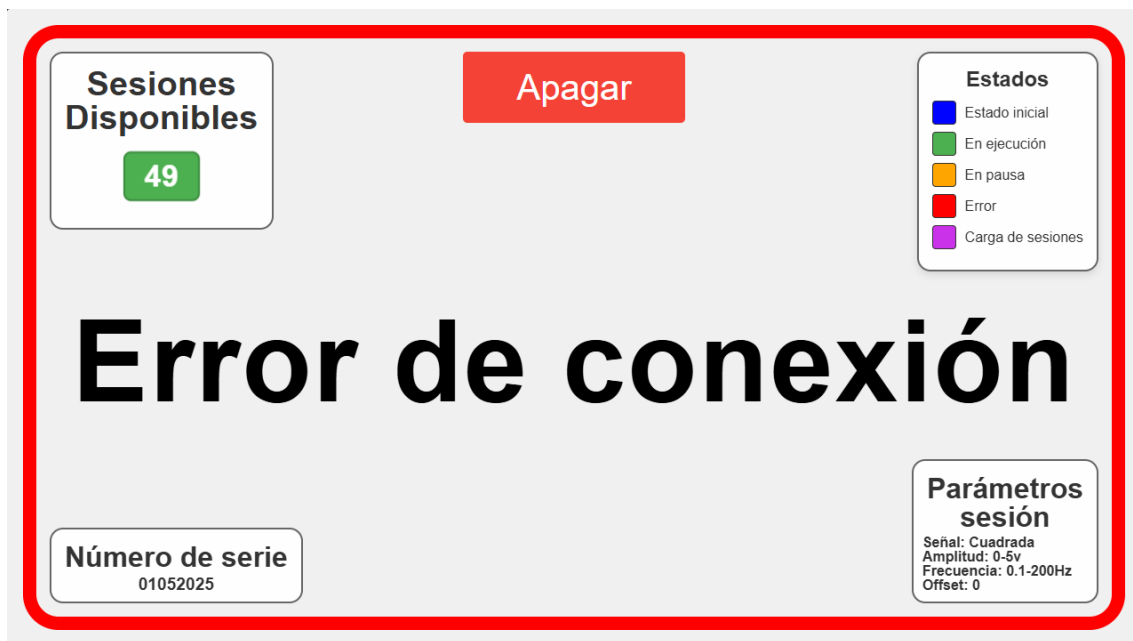
El estado de pausa se visualiza mediante un color amarillo, indicando que el contador o proceso principal ha sido temporalmente detenido. Este estado solo puede activarse desde el estado en ejecución (verde) al pulsar el botón de acción "Parar". Durante la pausa, la aplicación mantiene todos los datos y configuraciones actuales, permitiendo al usuario reanudar la sesión cuando lo necesite mediante el botón de "Continuar", que devolverá el sistema al estado de ejecución. El botón de continuar indicará la acción que se realizará y mediante el color del botón el estado al que se viaja.



Estado de error

El estado de error se identifica mediante un contorno rojo en la interfaz, acompañado de la desactivación del botón de acción principal (que desaparece) y la permanencia exclusiva del botón de apagado. Este estado no puede activarse manualmente mediante interacción del usuario, sino que se produce automáticamente en dos escenarios críticos:

- Agotamiento del número de sesiones permitidas, lo que requerirá una recarga manual de las mismas para reiniciar el sistema.
- Desconexión del dispositivo aplicador (fallo en la comunicación con los pines GPIO de la Raspberry Pi), que exigirá reconectar físicamente el hardware. Una vez reconectado el hardware la aplicación recuperará el último estado de la aplicación automáticamente.



Estado de carga de sesiones

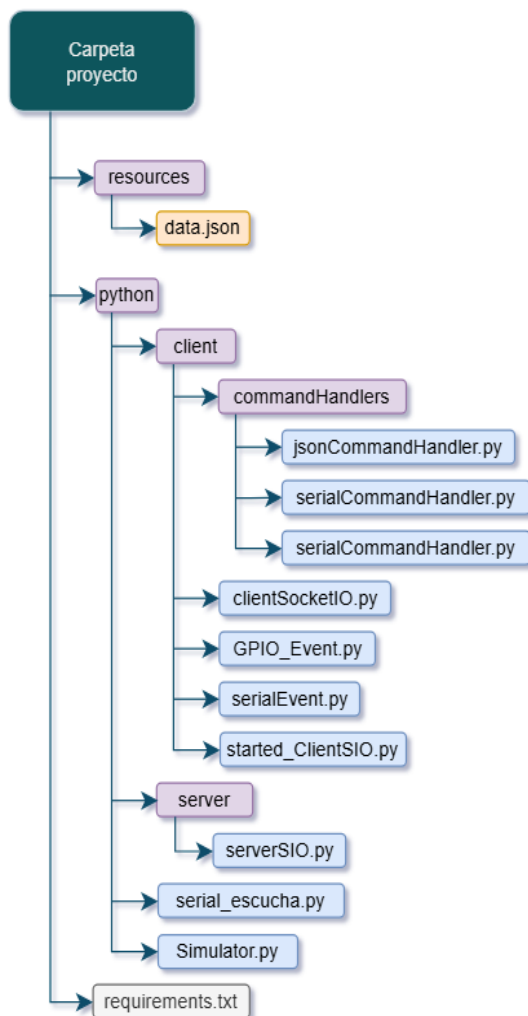
El estado de carga de sesiones viene indicado por el color morado. Este únicamente se muestra cuando se cargan las sesiones vía serial. En este estado se ocultan todos los botones para evitar apagados durante el proceso o cambios de estados.



2.3.3 Backend (Python)

En este apartado se presentará una descripción detallada del árbol de directorios que incluye el cliente backend y el servidor WebSockets. Se explicará la estructura de archivos y la función específica de cada uno, detallando el funcionamiento del servidor WebSockets. Se mostrarán los comandos utilizados en la interfaz serial para la gestión del dispositivo médico. Asimismo, se describirá el control implementado sobre los pines GPIO de la Raspberry Pi, fundamental para la interacción con el hardware, y finalmente se analizará el cliente Socket.IO, encargado de establecer la comunicación bidireccional en tiempo real entre el frontend y el backend. Este análisis integral permitirá comprender el flujo de información y la coordinación entre todos los módulos del sistema.

2.3.3.1 Árbol de directorios



En el árbol de directorios del backend, se encuentran dos carpetas principales. Una de ellas es la carpeta *resources*, donde se ubica el archivo *data.json*. Este archivo cumple la función de almacenar los datos esenciales de la aplicación, como el número de sesiones disponibles, los parámetros configurados y el número de serie del dispositivo. Su contenido es leído por el frontend al iniciar la aplicación, lo que permite cargar la información necesaria para su funcionamiento.

Dentro del directorio de Python, hay dos subcarpetas y dos archivos. Estos archivos no se compilan y están destinados a realizar pruebas. Uno de ellos es *Simulator.py*, que actúa como un simulador de cliente, permitiendo enviar distintos estados al frontend para probar su comportamiento. El otro archivo, *serial_escucha.py*, se encarga de gestionar la comunicación serial para pruebas, tanto para enviar como para recibir información.

En la carpeta *server*, se encuentra *serverSIO.py*, cuyo rol es funcionar como servidor, facilitando la comunicación entre los diferentes componentes del sistema. Por otro lado, dentro de la carpeta *client*, están los *commandHandlers*, que son módulos diseñados para gestionar eventos específicos. Entre ellos destacan *clientSocketIO.py* (encargado de manejar eventos de tipo Socket.IO), *GPIO_Event.py* (que procesa eventos de tipo GPIO) y *serialEvent.py* (dedicado a eventos de comunicación serial).

Finalmente, el componente que integra y coordina todos estos procesos es *started_ClientSIO.py*. Este archivo actúa como el hilo principal del cliente, iniciando y gestionando cada uno de los módulos mencionados para garantizar el correcto funcionamiento del sistema.

2.3.3.2 Servidor Socket.IO

El servidor Socket.IO es el componente encargado de establecer una comunicación bidireccional en tiempo real entre el frontend y el backend mediante WebSockets. Su principal función es garantizar que los datos se intercambien de manera fluida y sin interrupciones, actuando como un puente entre la interfaz de usuario y la lógica del servidor. Para mantener la conexión activa, el servidor emplea un mecanismo conocido como keepalive, el cual consiste en el envío periódico de pequeños paquetes de datos que verifican si la conexión sigue estable.

Además de gestionar la conexión, el servidor monitorea el estado de las conexiones activas. En este proyecto, si una conexión se interrumpe, la aplicación puede bloquearse automáticamente como medida de seguridad. Esto previene acciones inseguras o inconsistentes, garantizando que el sistema solo funcione cuando todos los componentes estén correctamente enlazados.

Dado que la aplicación opera en un entorno local sin dependencia de internet, el servidor se ejecuta en localhost, lo que restringe el acceso únicamente a la máquina donde está alojado. Para reforzar la seguridad, se implementa una contraseña compartida entre el backend y el frontend, la cual autentica las conexiones y evita accesos no autorizados. De esta manera, el servidor Socket.IO no solo facilita la comunicación en tiempo real, sino que también asegura que el intercambio de datos sea eficiente, estable y protegido.

2.3.3.3 Comunicación Serial

La Raspberry Pi debe poder conectarse por USB a un ordenador para hacer la carga de parámetros, establecer un número de serie del dispositivo y para cargar sesiones disponibles. Para ello se utiliza una conexión serial a 9600 baudios y se necesita un cable especial que se conecta a puerto UART de la Raspberry PI. Se han establecido 4 comandos para la comunicación. Estos comandos pueden escribirse en mayúsculas y minúsculas, pero los parámetros se registran en el sistema tal y como se escriban exceptuado el carácter “_”, que será remplazado por un espacio. A continuación, se muestran los comandos con sus ejemplos:

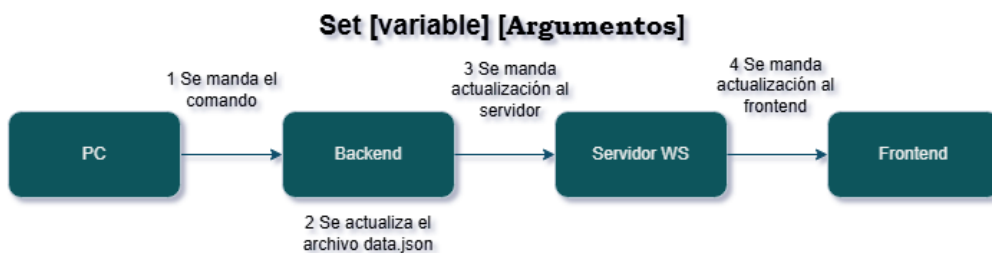
- Get SerialNumber



Sirve para obtener el número de serie del dispositivo, primero se manda el comando a la Raspberry Pi desde el PC y esta devuelve la respuesta indicando el número de serie.

Los argumentos de tipo Set siguen el mismo flujo que se describe a continuación:

Inicialmente se manda el comando al cliente backend que se encarga de gestionar el comando. Este controla si se encuentra bien estructurado y de ser así se actualiza el archivo data.json y se envía un mensaje al servidor para que se lo reenvíe al frontend. Una vez el frontend obtenga el mensaje, este lo interpreta y provoca la actualización correspondiente en la interfaz.



- Set SerialNumber [Número de series]

Modifica el número de serie, este es una cadena de texto, ejemplo de uso:
Set SerialNumber 01052025

- Set NumberSession [Número de sesiones]

Actualiza el número de sesiones disponibles, este comando únicamente acepta valores enteros. Ejemplo de uso:
Set NumberSession 50

- Set Countdown [Duración de la sesión en minutos]

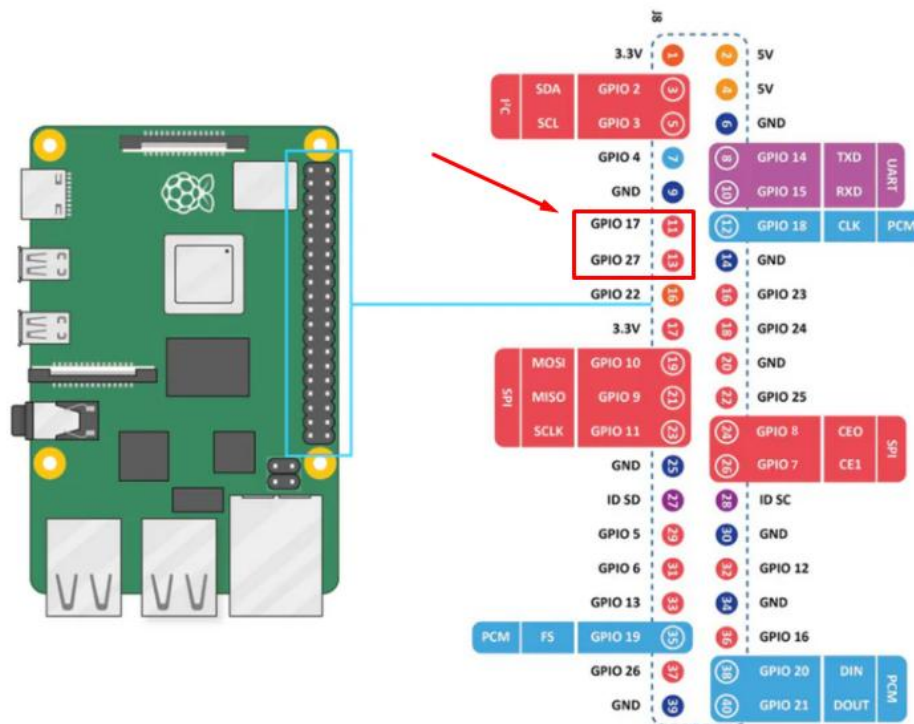
Reemplaza la duración actual de la sesión por la indicada en el comando, se aceptan variables de tipo float e indica los minutos de la sesión. Ejemplo de uso:
Set Countdown 20

- Set Param [Tipo de señal] [Amplitud] [Frecuencia] [Offset]

Establece en la interfaz los parámetros utilizados en la sesión, los argumentos son cadenas de caracteres. Ejemplo de uso:
Set Param Cuadrada 0-5_V 0.1-200_Hz 0

2.3.3.4 Control de GPIO

GPIO (del inglés General-Purpose Input/Output) se refiere a pines de propósito general en dispositivos electrónicos, como microcontroladores (Arduino, Raspberry Pi, ESP8266). En este proyecto se utilizarán 2 pines, uno de entrada del microcontrolador a la Raspberry Pi y otro de salida de la Raspberry Pi al microcontrolador. Los pines GPIO elegidos son el 17 y el 27 para entrada y salida respectivamente.

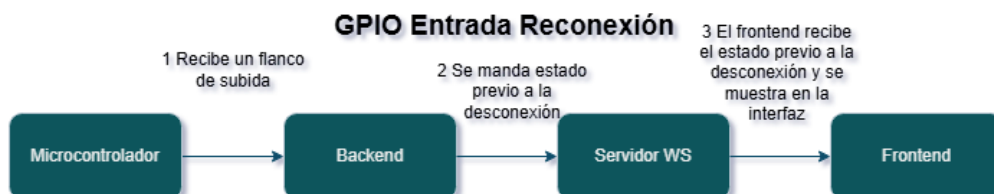


Respecto al flujo de salida, cada vez que se aprieta el botón de acción en el frontend, este envía un mensaje al servidor WebSocket y lo retransmite al cliente backend. El cliente backend se encarga de gestionar el mensaje y envía un pulso de 0,3 segundos a través de GPIO de salida.



Respecto al flujo de entrada, este es más complejo ya que, el cliente backend debe detectar si hay un error de conexión. Una vez reestablecida la conexión se debe regresar al estado anterior a la interrupción. Por este motivo, la aplicación debe tener una comunicación bidireccional en tiempo real, no siendo útil un API REST genérico. Para detectar si hay una desconexión hay que mencionar la señal que se recibe por dicho GPIO, se trata de una señal cuadrada con un flanco de subida constante. Si se produce un flanco de bajada superior a medio

segundo se produce un error de desconexión y se debe informar en la pantalla de la interfaz.



Para realizar pruebas se ha trabajado con un ESP8266 el cual se ha programado con el comportamiento que tendría el microcontrolador final de la aplicación.

2.3.3.5 Cliente Backend

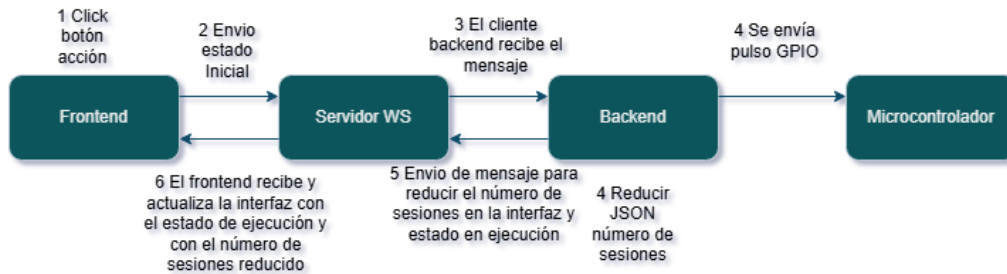
El cliente es quien tiene la mayor importancia en la parte lógica del sistema, es el encargado de gestionar eventos mediante comunicación serial, WebSocket y GPIO. Este ordena al frontend a qué estado debe cambiar y quién se encarga de actualizar *data.json* para las futuras ejecuciones. A continuación, se explicará el flujo que se realiza en cada cambio de estado.

Cuando un usuario presiona el botón de acción en la interfaz del frontend, se inicia un flujo de comunicación en tiempo real entre los diferentes componentes del sistema. Este proceso comienza cuando el frontend envía un mensaje al servidor Socket.IO, conteniendo el estado actual de la aplicación en ese momento preciso.

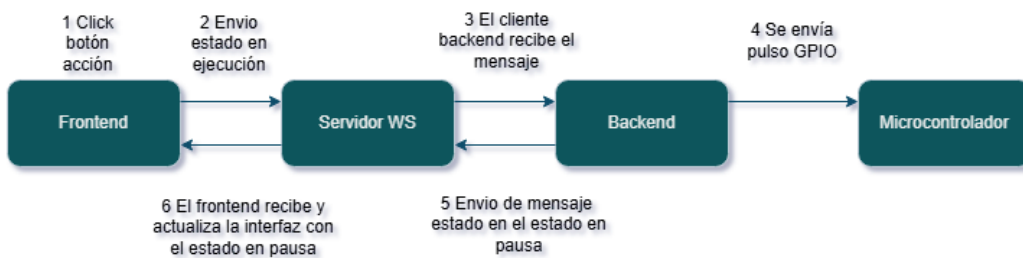
El servidor Socket.IO, actuando como intermediario, recibe inmediatamente este mensaje y lo retransmite al cliente backend. Este cliente backend contiene la lógica principal para determinar las transiciones de estado del sistema. Al recibir el estado actual, analiza la situación y decide cuál debe ser el próximo estado del sistema.

En el caso particular donde el estado actual coincida con el estado inicial del sistema (el punto de partida del flujo), se ejecuta una acción especial: el cliente backend envía una petición específica de vuelta al frontend. Esta petición tiene un efecto concreto, reducir en una unidad el contador de sesiones disponibles. Este decremento queda reflejado tanto en la interfaz de usuario como en el archivo *data.json*, manteniendo así la consistencia de los datos en toda la aplicación.

Flujo para pasar del estado inicial a en ejecución



Flujo para pasar del estado en ejecución a en pausa



Flujo para pasar del estado en pausa a en ejecución



El sistema cuenta con dos estados adicionales importantes: el estado de carga de sesiones y el estado de error. Mientras que el estado de error está directamente relacionado con el control GPIO (como se explicó anteriormente), el estado de carga de sesiones tiene un comportamiento particular.

El proceso de carga de sesiones se activa mediante una conexión serial cuando se recibe el comando específico “*Set NumberSession*” seguido del número deseado de sesiones. Este comando no se ejecuta a través de ningún botón en la interfaz, sino que requiere una comunicación directa vía puerto serial. Al recibir este comando, el sistema genera un evento serial que desencadena una serie de acciones:

Inicio del proceso:

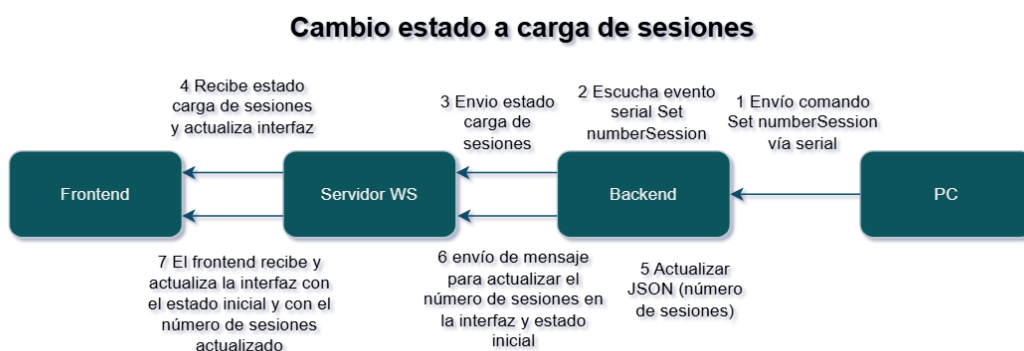
Al establecerse la conexión serial y detectarse el comando, el cliente backend envía inmediatamente un mensaje de carga de sesiones al servidor Socket.IO. Este servidor actúa como intermediario, reenviando el mensaje al frontend.

Actualización de la interfaz:

El frontend, al recibir este mensaje, actualiza la interfaz gráfica para reflejar el estado de carga de sesiones. Durante este proceso, la interfaz muestra un mensaje informativo, proporcionando feedback claro al usuario sobre la operación en curso.

Finalización del proceso:

Una vez completada la carga de sesiones, el cliente backend envía dos mensajes al frontend a través del servidor. Con el fin de establecer el estado inicial y actualizar el contador con el nuevo número de sesiones especificado.



2.3.4 Hardware

El desarrollo de la aplicación ha requerido el uso de varios componentes hardware seleccionados para garantizar un funcionamiento óptimo y una perfecta integración entre ellos. El dispositivo principal del sistema es una Raspberry Pi, que actúa como el núcleo de procesamiento, ejecutando tanto el

backend como el frontend de la aplicación. Esta placa fue elegida por su potencia, versatilidad y amplia compatibilidad con periféricos, lo que la convierte en una solución ideal para este tipo de proyectos.

Para la interfaz de usuario, se ha utilizado una pantalla táctil Raspberry Pi Touch Display de 7 pulgadas, que ofrece una resolución de 720×1280 píxeles. Esta pantalla utiliza tecnología IPS, lo que proporciona ángulos de visión amplios y colores vibrantes, para diferenciar claramente los diferentes estados.

En cuanto a la simulación del microcontrolador, se ha empleado un ESP8266, que opera a 3.3V en sus pines GPIO, al igual que la Raspberry Pi y el microcontrolador destino final. Este componente se utilizó durante la fase de desarrollo para emular el comportamiento del hardware objetivo, permitiendo pruebas tempranas de la lógica de control. Su compatibilidad en voltaje con los demás elementos del sistema eliminó la necesidad de conversores de nivel, simplificando el diseño del circuito y reduciendo posibles fuentes de error.

2.3.4.1 Configuración Raspberry Pi

Para el correcto funcionamiento del sistema en la Raspberry Pi, es necesario instalar varios componentes esenciales. Entre ellos se encuentran; Node.js, que proporciona el entorno de ejecución para JavaScript; Electron, utilizado para construir la interfaz de usuario del frontend y Python, que sirve como base para el backend. Las dependencias y librerías adicionales requeridas por Python se especifican en el archivo `requirements.txt`, mientras que las dependencias de Electron se detallan en el archivo `package.json`.

El proyecto cuenta con dos modos de ejecución principales para Electron. El primero, iniciado con `"npm run dev"`, está diseñado específicamente para el entorno de desarrollo. Este modo ejecuta los archivos Python directamente y carga variables de configuración adaptadas al desarrollo, incluyendo herramientas de depuración de Electron. El segundo modo, iniciado con `"npm run start"`, está pensado para entornos de producción y utiliza archivos binarios de Python, lo que permite la ejecución del sistema en cualquier máquina sin necesidad de tener Python instalado previamente.

2.3.4.2 Proceso de Compilación y Ejecución Automática

Para facilitar la compilación del servidor, el cliente frontend y el backend, se ha creado una carpeta especial llamada `Help-files`. Esta carpeta contiene varios scripts de utilidad que automatizan diferentes procesos: uno para compilar el proyecto, otro para crear un entorno virtual de Python, un tercero para iniciar

el programa final y, finalmente, el archivo autostart.desktop. Este último archivo es fundamental para garantizar que la aplicación se ejecute automáticamente al iniciar sesión en la Raspberry Pi. Para que funcione correctamente, debe colocarse en la ruta específica /home/[nombre_usuario]/.config/autostart, donde [nombre_usuario] debe ser reemplazado por el nombre de usuario correspondiente de la Raspberry Pi, normalmente pi en configuraciones estándar. Esta configuración asegura que el sistema esté listo para funcionar inmediatamente después del arranque, sin necesidad de intervención manual.

3 Resultados y conclusiones

El desarrollo de este Trabajo Fin de Grado ha permitido crear un sistema integral para el dispositivo MINESTIM APCM-01, superando las limitaciones de su interfaz original mediante una solución gráfica intuitiva y funcional. La implementación de una GUI desarrollada con Electron.js ha proporcionado información clara y en tiempo real sobre el estado del dispositivo, incluyendo el tiempo restante de sesión y el número de sesiones disponibles. Esta mejora ha eliminado la ambigüedad del sistema basado únicamente en indicadores LED, facilitando su uso por parte del personal médico.

La comunicación en tiempo real entre el frontend y el backend, gestionada mediante Socket.IO, ha demostrado ser altamente eficiente, con latencias inferiores a un segundo en todas las pruebas realizadas. Además, la integración con los pines GPIO de la Raspberry Pi y la comunicación serial ha permitido un control preciso del dispositivo, asegurando una operación estable incluso en condiciones reales de uso. La automatización del inicio de la aplicación mediante el archivo autostart.desktop ha garantizado que el sistema esté listo para funcionar inmediatamente después del arranque, optimizando su implementación en entornos clínicos.

Las pruebas realizadas con el prototipo, utilizando un ESP8266 como simulador del microcontrolador final, validaron el cumplimiento de todos los requisitos funcionales y no funcionales definidos inicialmente. El sistema mostró un comportamiento robusto en las transiciones entre estados, la detección de errores y la actualización persistente de los parámetros en el archivo data.json. Estos resultados confirman que la solución desarrollada no solo satisface las necesidades técnicas del proyecto, sino que también mejora significativamente la experiencia del usuario.

En conclusión, este trabajo ha demostrado cómo la modernización de interfaces en dispositivos médicos puede incrementar su eficiencia y reducir errores operativos. La elección de tecnologías como Electron, Python y Socket.IO ha sido fundamental para lograr un sistema modular, escalable y fácil de mantener. Entre las lecciones más valiosas obtenidas destacan la importancia del feedback visual para evitar ambigüedades y los retos asociados a la integración hardware-software, que requirieron un diseño cuidadoso de los mecanismos de control y recuperación ante fallos.

A nivel personal, este proyecto ha resultado sumamente enriquecedor, no solo por el aprendizaje técnico adquirido, sino también por su aplicación directa en el desarrollo profesional. La implementación de tecnologías como WebSockets, junto con la integración de un frontend y backend robustos, han permitido profundizar en conceptos clave que son fundamentales en diferentes puestos de trabajo, donde se utilizan sistemas a gran escala en entornos productivos. Esta

experiencia no solo refuerza la comprensión de arquitecturas de comunicación en tiempo real, sino que también proporciona herramientas prácticas para optimizar procesos similares en el ámbito laboral, consolidando así tanto crecimiento técnico como la capacidad para abordar desafíos complejos en entornos de alta demanda.

4 Análisis de Impacto

El desarrollo de este Trabajo Fin de Grado no solo ha generado resultados técnicos significativos, sino que también tiene el potencial de producir un impacto relevante en múltiples ámbitos. A continuación, se analizan las implicaciones en diferentes contextos, considerando tanto los beneficios esperados como los posibles desafíos.

Impacto Personal

A nivel individual, este proyecto ha representado una oportunidad valiosa para profundizar en tecnologías clave como WebSockets, Electron y Python, fortaleciendo las competencias técnicas y la capacidad para diseñar sistemas integrados. La experiencia adquirida tiene aplicación directa en el entorno laboral, donde se trabaja a menudo con arquitecturas similares en sistemas de producción a gran escala. Además, el enfoque en usabilidad y diseño centrado en el usuario amplían la perspectiva sobre la importancia de crear soluciones accesibles y eficientes.

Impacto Empresarial

Para el sector empresarial, especialmente en el ámbito de dispositivos médicos, la solución desarrollada ofrece ventajas competitivas. La interfaz intuitiva y la comunicación en tiempo real pueden reducir errores operativos, mejorar la eficiencia clínica y disminuir los costes asociados a formación de personal. Sin embargo, su implementación requeriría una inversión inicial en hardware (Raspberry Pi, pantallas táctiles) y adaptación de protocolos existentes.

Impacto Social

El proyecto contribuye positivamente a la sociedad al mejorar la experiencia de uso de un dispositivo médico destinado a pacientes con fibromialgia, una condición que afecta significativamente la calidad de vida. Una interfaz clara y fiable puede aumentar la adherencia al tratamiento y reducir la ansiedad tanto en pacientes como en profesionales sanitarios. Este avance alinea con el ODS 3 (Salud y Bienestar), promoviendo tecnologías accesibles para mejorar la atención médica.

Impacto Económico

Económicamente, el sistema propuesto es rentable al utilizar componentes de bajo coste (como Raspberry Pi) frente a alternativas comerciales. No obstante, su escalabilidad requeriría validaciones clínicas adicionales, lo que podría incrementar costes iniciales. A largo plazo, la reducción de errores y la optimización de sesiones podrían traducirse en ahorros para instituciones de salud.

Decisiones Basadas en Impacto

A lo largo del desarrollo, se tomaron decisiones clave considerando estos aspectos:

- **Accesibilidad:** La GUI se diseñó con colores intuitivos y retroalimentación clara para usuarios con distintos niveles de formación técnica.
- **Sostenibilidad:** Se optó por hardware reutilizable y software modular para facilitar actualizaciones futuras sin obsolescencia programada.

En conclusión, este proyecto tiene potencial para generar impactos positivos multisectoriales, especialmente en salud y tecnología médica, mientras aborda desafíos como la escalabilidad y sostenibilidad.

5 Bibliografía

- [1] *Aplicaciones construidas con Electron.js*, OpenJS Foundation, 2025. [En línea]. Disponible: <https://www.electronjs.org/apps>. [Accedido: 20-mar-2025].
- [2] *Documentación de GPIO Zero (versión latest)*, Raspberry Pi Foundation, 2025. [En línea]. Disponible: <https://gpiozero.readthedocs.io/en/latest/>. [Accedido: 10-abr-2025].
- [3] *Documentación de asyncio en Python (3.x)*, Python Software Foundation, 2025. [En línea]. Disponible: <https://docs.python.org/es/3/library/asyncio.html>. [Accedido: 25-abr-2025].
- [4] *Documentación de la API de Node.js (versión latest)*, OpenJS Foundation, 2025. [En línea]. Disponible: <https://nodejs.org/docs/latest/api/>. [Accedido: 05-may-2025].
- [5] *Documentación del servidor Python-SocketIO*, Miguel Grinberg, 2025. [En línea]. Disponible: <https://python-socketio.readthedocs.io/en/latest/server.html>. [Accedido: 15-may-2025].
- [6] *Documentación del sistema operativo Raspberry Pi*, Raspberry Pi Ltd, 2025. [En línea]. Disponible: <https://www.raspberrypi.com/documentation/computers/os.html>. [Accedido: 12-may-2025].
- [8] *Documentación de PySerial (versión latest)*, Python Serial Port Extension Project, 2025. [En línea]. Disponible: <https://pyserial.readthedocs.io/en/latest/>. [Accedido: 08-may-2025].
- [9] *draw.io: Herramienta de diagramación en línea*, JGraph Ltd, 2025. [En línea]. Disponible: <https://www.drawio.com/>. [Accedido: 22-mar-2025].
- [10] *Fork: Cliente gráfico de Git*, 2025. [En línea]. Disponible: <https://git-fork.com/>. [Accedido: 18-abr-2025].
- [11] *GitHub: Plataforma de desarrollo colaborativo*, GitHub Inc., 2025. [En línea]. Disponible: <https://github.com/>. [Accedido: 30-mar-2025].
- [12] *Pantalla táctil oficial Raspberry Pi (modelo 2)*, Raspberry Pi Ltd, 2025. [En línea]. Disponible: <https://www.raspberrypi.com/products/touch-display-2/>. [Accedido: 20-may-2025].
- [13] *Python General FAQ*, Python Software Foundation, 2025. [Online]. Disponible en: <https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place>. [Accedido: 28-abr-2025].
- [14] *RPi.GPIO*, PyPI, 2025. [En línea]. Disponible en: <https://pypi.org/project/RPi.GPIO/>. [Accedido: 05-may-2025].

- [15] *socket.io.esm.min.js (v4.8.1)*, CDN Documentation, 2025. [Online]. Available: <https://cdn.socket.io/4.8.1/socket.io.esm.min.js>. [Accedido: 14-abr-2025].
- [16] *TIOBE Index for May 2025*, TIOBE Software BV, 2025. [Online]. Disponible en: <https://www.tiobe.com/tiobe-index/>. [Accedido: 02-may-2025].
- [17] *Uso confiable de WebSocket en aplicaciones sin código*, AppMaster, 2025. [En línea]. Disponible en: <https://appmaster.io/es/blog/uso-confiable-de-websocket-en-aplicaciones-sin-codigo>. [Accedido: 25-abr-2025].
- [18] *Visual Studio Code: Entorno de desarrollo integrado*, Microsoft Corporation, 2025. [En línea]. Disponible: <https://code.visualstudio.com/>. [Accedido: 10-abr-2025].
- [19] *¿Qué es Electron.js?*, Bambu Mobile, 2025. [En línea]. Disponible en: <https://bambu-mobile.com/que-es-electron-js/>. [Accedido: 15-mar-2025].
- [30] *¿Qué es Electron.js?*, Raullg, 2025. [En línea]. Disponible en: <https://www.raullg.com/que-es-electron-js/>. [Accedido: 20-mar-2025].
- [31] *¿Qué es Electron y cómo funciona?*, KeepCoding, 2025. [En línea]. Disponible en: <https://keepcoding.io/blog/que-es-electron-y-como-funciona/>. [Accedido: 18-mar-2025].
- [32] *¿Qué son WebSockets?*, KeepCoding, 2025. [En línea]. Disponible en: <https://keepcoding.io/blog/que-son-websockets/>. [Accedido: 22-abr-2025].

6 Anexos

En este capítulo se mostrará el código desarrollado para la elaboración del trabajo de fin de grado. Primero se indicará una lista de los archivos.

```
TFG\package.json
TFG\requirements.txt
TFG\src\renderer\styles.css
TFG\src\renderer\renderer.js
TFG\src\renderer\index.html
TFG\src\main\preload.js
TFG\src\main\main.js
TFG\src\main\utils\clientSocketIO.js
TFG\resources\data.json
TFG\python\Simulator.py
TFG\python\serial_escucha.py
TFG\python\server\serverSIO.py
TFG\python\client\started_ClientSIO.py
TFG\python\client\serialEvent.py
TFG\python\client\GPIO_Event.py
TFG\python\client\clientSocketIO.py
TFG\python\client\commandHandlers\socketIOCommandHandler.py
TFG\python\client\commandHandlers\serialCommandHandler.py
TFG\python\client\commandHandlers\jsonCommandHandler.py
TFG\ESP8266\Emisor_receptor.ino
TFG\Help-files\autostart.desktop
TFG\Help-files\compiler
TFG\Help-files\CreateVenv
TFG\Help-files\start
```

TFG\package.json

```
1  {
2    "name": "tfg",
3    "version": "1.0.0",
4    "description": "TFG",
5    "main": "src/main/main.js",
6    "scripts": {
7      "start": "electron .",
8      "dev": "electron . --dev",
9      "pack": "electron-builder --dir",
10     "dist": "electron-builder",
11     "dist:win": "electron-builder --win",
12     "dist:linux": "electron-builder --linux",
13     "test": "echo \"Error: no test specified\" && exit 1"
14   },
15   "build": {
```

```

16  "appId": "com.tfg",
17  "productName": "TFG",
18  "win": {
19    "target": "nsis",
20    "icon": "src/assets/icon.ico"
21  },
22  "linux": {
23    "target": ["AppImage"],
24    "icon": "src/assets/icon.png"
25  },
26  "files": [
27    "**/*",
28    "!node_modules/",
29    "!.venv/**",
30    "!python/**",
31    "!Help-files/**"
32  ],
33  "extraResources": [
34    {
35      "from": "resources",
36      "to": "resources",
37      "filter": [
38        "**/*"
39      ]
40    },
41    {
42      "from": "bin",
43      "to": "bin",
44      "filter": [
45        "**/*"
46      ]
47    }
48  ],
49  },
50  "author": "Jesus Quirante <jesus.quirante.dominguez@gmail.com>",
51  "license": "ISC",
52  "devDependencies": {

```

```

53  "chai": "^5.2.0",
54  "electron": "^35.3.0",
55  "electron-builder": "^26.0.12",
56  "electron-installer-debian": "^3.2.0",
57  "electron-packager": "^17.1.2",
58  "mocha": "^11.1.0",
59  "standard": "^17.1.2"
60  },
61  "eslintConfig": {
62    "extends": "standar"
63  },
64  "repository": {
65    "type": "git",
66    "url": "git+https://github.com/MrJesu95/TFG.git"
67  },
68  "keywords": [],
69  "bugs": {
70    "url": "https://github.com/MrJesu95/TFG/issues"
71  },
72  "homepage": "https://github.com/MrJesu95/TFG#readme",
73  "dependencies": {
74    "socket.io-client": "^4.8.1"
75  }
76  }

```

TFG\requirements.txt

```

1  altgraph==0.17.4
2  bidict==0.23.1
3  blinker==1.9.0
4  certifi==2025.4.26
5  charset-normalizer==3.4.2
6  click==8.1.8
7  colorzero==2.0
8  Flask==3.1.0
9  Flask-SocketIO==5.5.1
10 future==1.0.0

```

```
11 gevent==25.4.2
12 gpiozero==2.0.1
13 greenlet==3.2.1
14 h11==0.16.0
15 idna==3.10
16 iso8601==2.1.0
17 itsdangerous==2.2.0
18 Jinja2==3.1.6
19 lgpio==0.2.2.0
20 MarkupSafe==3.0.2
21 packaging==25.0
22 pyinstaller==6.13.0
23 pyinstaller-hooks-contrib==2025.4
24 pyserial==3.5
25 python-engineio==4.12.0
26 python-socketio==5.13.0
27 PyYAML==6.0.2
28 requests==2.32.3
29 rpi-lgpio==0.6
30 simple-websocket==1.1.0
31 six==1.17.0
32 socketIO-client==0.7.2
33 urllib3==2.4.0
34 websocket-client==1.8.0
35 Werkzeug==3.1.3
36 wsproto==1.2.0
37 zope.event==5.0
38 zope.interface==7.2
39
```

TFG\src\renderer\styles.css

```
1 * {
2   user-select: none; /* Deshabilita la selección de texto */
3 }
4
5 body {
```

```

6  font-family: Arial, sans-serif;
7  display: flex;
8  justify-content: center;
9  align-items: center;
10 height: 100vh;
11 width: 100vw;
12 margin: 0;
13 padding: 20px;
14 box-sizing: border-box;
15 background-color: #f0f0f0;
16 position: relative;
17 }
18
19 .shutdown-btn-container {
20 position: absolute;
21 top: 40px;
22 left: 50%;
23 transform: translateX(-50%);
24 z-index: 10;
25 }
26
27 .shutdown-btn {
28 padding: 10px 20px;
29 font-size: 2.5rem;
30 background-color: #f44336; /* Rojo */
31 color: white
32 border: none;
33 border-radius: 5px;
34 cursor: pointer;
35 transition: background-color 0.3s;
36 white-space: nowrap; /* Evita que el texto se divida en varias líneas
37 */
38 }
39
40 .shutdown-btn:hover {
41 background-color: #d32f2f; /* Rojo más oscuro al hover */
42 }

```

```

42
43 /* Estilos para el modal de apagado */
44 .modal {
45     position: fixed;
46     top: 0;
47     left: 0;
48     width: 100%;
49     height: 100%;
50     background-color: rgba(0, 0, 0, 0.7);
51     display: flex;
52     justify-content: center;
53     align-items: center;
54     z-index: 2000; /* Mayor que todo lo demás */
55 }
56
57 .modal.hidden {
58     display: none;
59 }
60
61 .modal-content {
62     background-color: white
63     padding: 30px;
64     border-radius: 15px;
65     text-align: center;
66     max-width: 400px;
67     width: 90%;
68     box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
69 }
70
71 .modal-content p {
72     font-size: 1.5rem;
73     margin-bottom: 25px;
74     color: #333;
75 }
76
77 .modal-buttons {
78     display: flex;

```

```

79 justify-content: center;
80 gap: 20px;
81 }
82
83 .modal-btn {
84 padding: 12px 25px;
85 font-size: 1.2rem;
86 border: none;
87 border-radius: 8px;
88 cursor: pointer;
89 transition: all 0.3s ease;
90 }
91
92 .confirm-btn {
93 background-color: #f44336;
94 color: white
95 }
96
97 .confirm-btn:hover {
98 background-color: #d32f2f;
99 }
100
101 .cancel-btn {
102 background-color: #e0e0e0;
103 color: #333;
104 }
105
106 .cancel-btn:hover {
107 background-color: #bdbdbd;
108 }
109
110 .session-header {
111 position: absolute; /* Lo saca del flujo normal */
112 top: 50px; /* Ajusta según tu padding del body */
113 left: 50px; /* Ajusta según tu padding del body */
114 display: flex;
115 flex-direction: column;

```



```

116 align-items: center; /* Alinea a la derecha */
117 padding: 10px;
118 gap: 0;
119 z-index: 10; /* Asegura que esté por encima */
120 width: auto;
121 text-align: center;
122 border: 2px solid #666666;
123 border-radius: 15px;
124 background-color: rgba(255, 255, 255, 0.9);
125 }
126
127 .session-label {
128 font-family: 'Arial', sans-serif;
129 font-size: 2.4rem;
130 color: #333;
131 font-weight: bold;
132 text-align: center;
133 margin: 5px;
134 margin-bottom: 0; /* Elimina margen inferior */
135 line-height: 1; /* Reduce el espacio de línea */
136 width: 100%;
137 }
138
139 #nSession {
140 font-family: 'Arial', sans-serif;
141 font-size: 2rem;
142 color: #ffffff;
143 background-color: #4CAF50;
144 border: 2px solid #3e8e41;
145 border-radius: 8px;
146 padding: 5px 20px; /* Reducido padding vertical */
147 text-align: center;
148 width: fit-content;
149 box-shadow: 0 4px 6px rgba(0, 0, 0, 0);
150 animation: refresh 2s infinite ease-in-out;
151 margin-top: 20px; /* Mínimo espacio superior */
152 line-height: 1.2; /* Ajuste fino de altura de línea */

```

```

153 margin-left: auto;
154 margin-right: auto; /* Centrado horizontal */
155 }
156
157
158 .nserie-header {
159 position: absolute; /* Lo saca del flujo normal */
160 bottom: 50px; /* Ajusta según tu padding del body */
161 left: 50px; /* Ajusta según tu padding del body */
162 display: flex;
163 flex-direction: column;
164 align-items: center; /* Alinea a la derecha */
165 padding: 10px;
166 gap: 0;
167 z-index: 10; /* Asegura que esté por encima */
168 width: auto;
169 text-align: center;
170 border: 2px solid #666666;
171 border-radius: 15px;
172 background-color: rgba(255, 255, 255, 0.9);
173 }
174
175 .nserie-label {
176 font-family: 'Arial', sans-serif;
177 font-size: 2rem;
178 color: #333;
179 font-weight: bold;
180 text-align: center;
181 margin: 5px;
182 margin-bottom: 0; /* Elimina margen inferior */
183 line-height: 1; /* Reduce el espacio de línea */
184 width: 100%;
185 }
186
187 #nserie {
188 font-family: 'Arial', sans-serif;
189 font-size: 1.2rem;

```

```

190 color: #333;
191 font-weight: bold;
192 text-align: center;
193 margin: 5px;
194 margin-bottom: 0;
195 line-height: 1;
196 width: 100%;
197 }
198
199 .param-header {
200 position: absolute; /* Lo saca del flujo normal */
201 bottom: 50px; /* Ajusta según tu padding del body */
202 right: 50px; /* Ajusta según tu padding del body */
203 display: flex;
204 flex-direction: column;
205 align-items: center; /* Alinea a la derecha */
206 padding: 10px;
207 gap: 0;
208 z-index: 10; /* Asegura que esté por encima */
209 width: auto;
210 text-align: center;
211 border: 2px solid #666666;
212 border-radius: 15px;
213 background-color: rgba(255, 255, 255, 0.9);
214 }
215
216 .param-label {
217 font-family: 'Arial', sans-serif;
218 font-size: 2rem;
219 color: #333;
220 font-weight: bold;
221 text-align: center;
222 margin: 5px;
223 margin-bottom: 0; /* Elimina margen inferior */
224 line-height: 1; /* Reduce el espacio de línea */
225 width: 100%;
226 }

```

```

227
228 #param {
229 font-family: 'Arial', sans-serif;
230 font-size: 1rem;
231 color: #333;
232 font-weight: bold;
233 text-align: left;
234 margin: 5px;
235 margin-bottom: 0;
236 line-height: 1;
237 width: 100%;
238 }
239
240
241
242
243 /* Animación para simular un efecto refrescante */
244 @keyframes refresh {
245 0%, 100% {
246 transform: scale(1); /* Escala normal */
247 }
248 50% {
249 transform: scale(1.1); /* Ligeramente más grande */
250 }
251 }
252
253 .container {
254 position: relative;
255 text-align: center;
256 border: 15px solid #0004ff;
257 border-radius: 40px;
258 width: 100%;
259 height: 100%;
260 padding: 20px;
261 box-sizing: border-box;
262 transition: border-color 0.3s ease-in-out;
263 display: flex;

```

```

264 flex-direction: column;
265 justify-content: center; /* Centrado vertical principal */
266 align-items: center; /* Centrado horizontal */
267 }
268
269 /* Estados adicionales */
270 .container.running {
271 border-color: #4CAF50; /* Verde: Estado activo */
272 }
273
274 .container.paused {
275 border-color: #FFA500; /* Naranja: Estado en pausa/reposo */
276 }
277
278 .container.error {
279 border-color: #FF0000; /* Rojo: Estado de error */
280 }
281
282 .container.init {
283 border-color: #0004ff; /* Rojo: Estado de error */
284 }
285
286 .container.sessionState {
287 border-color: #ca32e9; /* Azul: Estado de error */
288 }
289
290 #countdown {
291 font-size: 8rem;
292 font-weight: bold;
293 margin: 0;
294 margin-top: 10%;
295 width: 100%;
296 text-align: center;
297 position: relative;
298 letter-spacing: 2px; /* Mejor legibilidad */
299
300 }

```

```

301
302 .buttons {
303     display: flex;
304     justify-content: center;
305     width: 100%;
306     margin-top: 40px; /* Espacio entre countdown y botones */
307     position: relative;
308     top: -10px; /* Ajuste fino si es necesario */
309 }
310
311 button {
312     width: 250px; /* Ancho fijo */
313     height: 80px;
314     padding: 15px 30px; /* Aumentado de 10px 20px */
315     font-size: 2.5rem; /* Aumentado de 1rem */
316     border: none;
317     border-radius: 8px; /* Bordes más redondeados */
318     cursor: pointer;
319     color: white
320     transition: all 0.3s ease;
321     margin: 10px;
322 }
323
324 button.init {
325     background-color: #2196F3; /* Azul - Empezar */
326 }
327
328 button.pause {
329     background-color: #FF9800; /* Naranja - Pausar */
330 }
331
332 button.continue {
333     background-color: #4CAF50; /* Verde - Continuar */
334 }
335
336 button.hidden {
337     display: none;

```

```

338 }
339 /* button: hover {
340 opacity: 0.8;
341 } */
342
343
344 /* Estilos para el bloque de leyenda */
345 .legend {
346 position: absolute;
347 top: 50px;
348 right: 50px;
349 background-color: rgba(255, 255, 255, 0.9);
350 border: 2px solid #666666;
351 border-radius: 15px;
352 padding: 15px;
353 z-index: 10;
354 box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
355 }
356
357 .legend h2 {
358 margin-top: 0;
359 margin-bottom: 10px;
360 font-size: 1.5rem;
361 color: #333;
362 text-align: center;
363 }
364
365 .legend-item {
366 display: flex;
367 align-items: center;
368 margin-bottom: 8px;
369 }
370
371 .legend-color {
372 width: 25px; /* Aumentado de 20px */
373 height: 25px; /* Aumentado de 20px */
374 border-radius: 4px;

```

```

375 margin-right: 10px;
376 border: 1px solid #333;
377 }
378
379 /* Colores específicos para cada ítem de leyenda */
380 .legend-item:nth-child(2) .legend-color {
381 background-color: #0004ff !important;
382 }
383
384 .legend-item:nth-child(3) .legend-color {
385 background-color: #4CAF50 !important;
386 }
387
388 .legend-item:nth-child(4) .legend-color {
389 background-color: #FFA500 !important;
390 }
391
392 .legend-item:nth-child(5) .legend-color {
393 background-color: #FF0000 !important;
394 }
395
396 .legend-item:nth-child(6) .legend-color {
397 background-color: #ca32e9 !important;
398 }
399
400 .legend-text {
401 font-size: 1rem;
402 color: #333;
403 min-width: 120px; /* Asegura espacio para el texto */
404 }
405
406
407 /* Pantalla de carga */
408 #loading-screen {
409 position: fixed;
410 top: 0;
411 left: 0;

```



```

412 width: 100%;
413 height: 100%;
414 background-color: #f0f0f0;
415 display: flex;
416 justify-content: center;
417 align-items: center;
418 flex-direction: column;
419 z-index: 1000; /* Asegúrate de que esté sobre otros elementos */
420 }
421
422 .hidden {
423 visibility: hidden; /* Mantiene el espacio pero oculta el contenido */
424 pointer-events: none; /* Evita la interacción mientras está oculto */
425 }
426
427 .visible {
428 visibility: visible;
429 pointer-events: auto;
430 }
431
432 .loading-spinner {
433 border: 8px solid #f3f3f3; /* Borde claro */
434 border-top: 8px solid #4caf50; /* Borde verde */
435 border-radius: 50%;
436 width: 50px;
437 height: 50px;
438 animation: spin 1s linear infinite;
439 }
440
441 /* Animación de carga */
442 @keyframes spin {
443 0% {
444 transform: rotate(0deg);
445 }
446 100% {
447 transform: rotate(360deg);
448 }

```

449 | }

TFG\src\renderer\renderer.js

```
1 // Importación del socket desde el archivo clientSocketIO
2 import { socket } from '../main/utils/clientSocketIO.js'
3
4 // Obtención de elementos del DOM
5 const countdownElement = document.getElementById("countdown")
6 const sesionElement = document.getElementById("nSession")
7 const nserieElement = document.getElementById("nserie")
8 const paramElement = document.getElementById("param")
9 const statusElement = document.getElementById("status")
10 const actionButton = document.getElementById('actionButton')
11 const shutdownButton = document.getElementById('shutdown-btn')
12 let dJSON
13 let countdownValueMax
14 let countdownValue
15 let countdownInterval
16
17 // Evento que se ejecuta cuando el DOM está completamente cargado
18 document.addEventListener("DOMContentLoaded", async () => {
19   dJSON = await window.electronAPI.loadJSON('data.json')
20   init()
21 })
22
23 // Configuración del modal de apagado
24 document.getElementById("shutdown-btn").addEventListener("click", () => {
25   {
26     document.getElementById("shutdown-modal").classList.remove("hidden")
27   }
28
29   document.getElementById("shutdown-confirm").addEventListener("click",
30     () => {
31       document.getElementById("shutdown-modal").classList.add("hidden")
32       window.electronAPI.shutdownComputer()
33     })
34 })
```

```

33 document.getElementById("shutdown-cancel").addEventListener("click", ()
    => {
34 document.getElementById("shutdown-modal").classList.add("hidden")
35 })
36
37 // Función de inicialización
38 function init() {
39     countdownValueMax = dJSON.timeSesion * 60
40     countdownValue = countdownValueMax
41     setNumberSession(dJSON.nSesiones)
42     setNumberSerie(dJSON.serialNumber)
43     setParam(dJSON.typeSignal, dJSON.amplitude, dJSON.frequency,
        dJSON.offset)
44     if (dJSON.nSesiones > 0) {
45         setReset()
46     } else {
47         setZeroSession()
48     }
49 }
50
51 // Función para obtener el estado actual
52 export function getStatus() {
53     return statusElement.className
54 }
55
56 // Función para establecer estado "listo" (oculta pantalla de carga)
57 export function setReady() {
58     const loadingScreen = document.getElementById("loading-screen")
59
60     // Ocultar la pantalla de carga
61     loadingScreen.classList.add("hidden")
62     loadingScreen.classList.remove("visible")
63 }
64
65 // Función para establecer estado "cargando" (muestra pantalla de
    carga)
66 export function setLoading() {

```

```

67  const loadingScreen = document.getElementById("loading-screen")
68
69  // Mostrar la pantalla de carga
70  loadingScreen.classList.add("visible")
71  loadingScreen.classList.remove("hidden")
72  }
73
74  // Función para establecer estado "en ejecución"
75  export function setRunning() {
76    if (sesionElement.textContent > 0 || countdownValue > 0){
77      updateCountdownDisplay(countdownValue)
78      startCountdownDisplay()
79      statusElement.className = 'container running'
80      swapButton()
81    }
82    else{
83      setZeroSession()
84    }
85  }
86
87  // Función para establecer estado "pausado"
88  export function setPaused() {
89    if (sesionElement.textContent > 0 || countdownValue > 0){
90      updateCountdownDisplay(countdownValue)
91      clearInterval(countdownInterval)
92      countdownInterval = null
93      statusElement.className = 'container paused'
94      swapButton()
95    }
96    else{
97      setZeroSession()
98    }
99  }
100
101  // Función para establecer estado "error"
102  export function setError() {
103    if (sesionElement.textContent > 0 || countdownValue > 0){

```

```

104 updateCountdownDisplay(countdownValue)
105 clearInterval(countdownInterval)
106 countdownInterval = null
107 statusElement.className = 'container error'
108 swapButton()
109 countdownElement.textContent = `Error de conexión`
110 // countdownElement.innerHTML = `${countdownElement.innerHTML}<br>Error
    de conexión`
111 }
112 else{
113     setZeroSession()
114 }
115 }
116
117 // Función para establecer estado "reset"
118 export function setReset() {
119     if (sesionElement.textContent > 0){
120         countdownValue = countdownValueMax
121         updateCountdownDisplay(countdownValue)
122         clearInterval(countdownInterval)
123         countdownInterval = null
124         statusElement.className = 'container init'
125         swapButton()
126     }
127     else{
128         setZeroSession()
129     }
130 }
131
132 // Función para establecer estado "cargando sesiones"
133 export function setSessionState() {
134     clearInterval(countdownInterval)
135     countdownInterval = null
136     statusElement.className = 'container sessionState'
137     countdownElement.textContent = 'CARGANDO SESIONES'
138     swapButton()
139 }

```

```

140
141 // Función para establecer estado "sin sesiones"
142 export function setZeroSession() {
143   updateCountdownDisplay(countdownValue)
144   clearInterval(countdownInterval)
145   countdownInterval = null
146   statusElement.className = 'container error'
147   countdownElement.textContent = 'SIN SESIONES'
148   swapButton()
149   setNumberSession(0)
150 }
151
152 // Función para actualizar el número de sesiones mostrado
153 export function setNumberSession(data) {
154   sesionElement.textContent = data
155 }
156
157 // Función para actualizar el número de serie mostrado
158 export function setNumberSerie(data) {
159   nserieElement.textContent = data
160 }
161
162 // Función para actualizar los parámetros mostrados
163 export function setParam(typeSignal, amplitude, frequency, offset) {
164   paramElement.innerHTML = `Señal: ${typeSignal}<br>Amplitud:
    ${amplitude}<br>Frecuencia: ${frequency}<br>Offset: ${offset}`
165 }
166
167 // Función para establecer el valor del contador
168 export function setCountDown(data) {
169   countdownValueMax = data * 60
170   countdownValue = countdownValueMax
171   updateCountdownDisplay(countdownValue)
172 }
173
174 // document.getElementById("actionButton").addEventListener("click", ()
=> {

```

```

175 // setRunning()
176 // })
177
178 // Función para iniciar y controlar la cuenta regresiva visual
179 function startCountdownDisplay() {
180 // Si no hay un intervalo activo, iniciamos uno
181 if (!countdownInterval) {
182   countdownInterval = setInterval(() => {
183     if (countdownValue > 0) {
184       countdownValue--
185       updateCountdownDisplay(countdownValue)
186     } else {
187       clearInterval(countdownInterval)
188       countdownInterval = null
189       if (sesionElement.textContent == 0) { setZeroSession() }
190       else {
191         setReset()
192         socket.emit('countdownFinished', { "countdownFinished": 0 })
193         countdownValue = countdownValueMax
194         updateCountdownDisplay(countdownValue)
195       }
196     }
197   }, 1000)
198 }
199 }
200
201 // Función para actualizar la visualización del contador
202 function updateCountdownDisplay(countdownValue) {
203   const minutes = Math.floor(countdownValue / 60)
204   const seconds = countdownValue % 60
205
206   countdownElement.textContent = `${minutes.toString().padStart(2,
207     '0')}:${seconds
208     .toString()
209     .padStart(2, '0')}`
210 }

```

```

211
212 // document.getElementById("stop").addEventListener("click", () => {
213 // clearInterval(countdownInterval)
214 // countdownInterval = null
215 // setPaused()
216 // })
217
218 // Función para cambiar el botón de acción según el estado
219 function swapButton() {
220 // Reset de clases
221 actionButton.classList.remove('init', 'continue', 'pause', 'hidden')
222 shutdownButton.classList.remove('shutdown-btn', 'hidden')
223
224 if (statusElement.classList.contains('init')) {
225 actionButton.textContent = "Empezar"
226 actionButton.classList.add('init')
227 shutdownButton.classList.add('shutdown-btn')
228 }
229 else if (statusElement.classList.contains('paused')) {
230 actionButton.textContent = "Continuar"
231 actionButton.classList.add('continue')
232 shutdownButton.classList.add('shutdown-btn')
233 }
234 else if (statusElement.classList.contains('running')) {
235 actionButton.textContent = "Pausar"
236 actionButton.classList.add('pause')
237 shutdownButton.classList.add('shutdown-btn')
238 }
239 else if (statusElement.classList.contains('sessionState')) {
240 actionButton.textContent = "Ocultar"
241 actionButton.classList.add('hidden')
242 shutdownButton.classList.add('hidden')
243 }
244 else if (statusElement.classList.contains('error')){
245 actionButton.textContent = "Ocultar"
246 actionButton.classList.add('hidden')
247 shutdownButton.classList.add('shutdown-btn')

```



```

248 }
249 }
250
251 // Asignar evento
252 //actionButton.addEventListener('click', swapButton)

```

TFG\src\renderer\index.html

```

1  <!DOCTYPE html>
2  <html lang="es">
3
4  <head>
5  <meta charset="UTF-8" />
6  <meta http-equiv="Content-Security-Policy"
7  content="default-src 'self'; connect-src ws://localhost:5000
8  http://localhost:5000; script-src 'self'; style-src 'self';">
9  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
10 <title>TFG</title>
11 <link rel="stylesheet" href="styles.css" />
12 <script type="module" src="../../main/utils/clientSocketIO.js"></script>
13
14 <body>
15 <!-- Pantalla de carga -->
16 <div id="loading-screen">
17 <div class="loading-spinner"></div>
18 <h1>Cargando...</h1>
19 </div>
20
21 <!-- Boton de apagado -->
22 <div class="shutdown-btn-container">
23 <button class="shutdown-btn" id="shutdown-btn">Apagar</button>
24 </div>
25
26 <!-- Modal de confirmación de apagado -->
27 <div id="shutdown-modal" class="modal hidden">
28 <div class="modal-content">
29 <p>¿Deseas apagar el equipo?</p>

```

```

30 <div class="modal-buttons">
31 <button id="shutdown-confirm" class="modal-btn confirm-
    btn">Aceptar</button>
32 <button id="shutdown-cancel" class="modal-btn cancel-
    btn">Cancelar</button>
33 </div>
34 </div>
35 </div>
36
37 <!-- Marco de sesiones -->
38 <div class="session-header">
39 <h1 class="session-label">Sesiones<br>Disponibles</h1>
40 <h1 id="nSession"></h1>
41 </div>
42
43 <!-- Marco de Numero de serie -->
44 <div class="nserie-header">
45 <h1 class="nserie-label">Número de serie</h1>
46 <h1 id="nserie"></h1>
47 </div>
48
49 <!-- Marco de parametros -->
50 <div class="param-header">
51 <h1 class="param-label">Parámetros<br>sesión</h1>
52 <h1 id="param"></h1>
53 </div>
54
55 <!-- Marco de leyendas -->
56 <div class="legend">
57 <h2>Estados</h2>
58 <div class="legend-item">
59 <div class="legend-color"></div>
60 <span class="legend-text">Estado inicial</span>
61 </div>
62 <div class="legend-item">
63 <div class="legend-color"></div>
64 <span class="legend-text">En ejecución</span>

```

```

65 </div>
66 <div class="legend-item">
67 <div class="legend-color"></div>
68 <span class="legend-text">En pausa</span>
69 </div>
70 <div class="legend-item">
71 <div class="legend-color"></div>
72 <span class="legend-text">Error</span>
73 </div>
74 <div class="legend-item">
75 <div class="legend-color"></div>
76 <span class="legend-text">Carga de sesiones</span>
77 </div>
78 </div>
79
80 <!-- Contenido principal -->
81 <div class="container" id="status">
82 <h1 id="countdown"></h1>
83 <div class="buttons">
84 <button class="init" id="actionButton">Empezar</button>
85 <!-- <button id="stop">Parar</button> -->
86 </div>
87 </div>
88
89 <script type="module" src="../../renderer/renderer.js"></script>
90 </body>
91
92 </html>

```

TFG\src\main\preload.js

```

1  const { contextBridge, ipcRenderer } = require('electron')
2
3  contextBridge.exposeInMainWorld('electronAPI', {
4    // Función cargar JSON
5    loadJSON: (filePath) => {
6      try {

```

```

7 // Solicita la carga del archivo JSON al proceso principal
8 const jsonData = ipcRenderer.invoke('load-json', filePath)
9 if (jsonData.error) {
10 console.error('Error desde el proceso principal:', jsonData.error)
11 return null;
12 }
13 return jsonData;
14 } catch (error) {
15 console.error('Error al comunicarse con el proceso principal:', error)
16 return null;
17 }
18 },
19 // Función para apagar el equipo
20 shutdownComputer: () => {
21 try {
22 return ipcRenderer.invoke('shutdown-computer')
23 } catch (error) {
24 console.error('Error al solicitar apagado:', error)
25 return Promise.reject(error)
26 }
27 }
28 });

```

TFG\src\main\main.js

```

1 const { app, BrowserWindow, ipcMain, ipcRenderer, globalShortcut } =
  require('electron')
2 const { spawn, exec } = require('child_process')
3 const path = require('path')
4 const fs = require('fs')
5 //const isDev = true;
6 const isDev = process.argv.includes('--dev') // Verificar si estamos en
  modo desarrollo
7
8 // Función para crear la ventana principal de la aplicación
9 const createWindow = () => {
10 const options = {
11 width: 1280,

```

```

12 height: 720,
13 icon: path.join(__dirname, '../assets/icon.png'), // Ruta del icono de
    la aplicación
14 frame: false, // Ventana sin marco
15 resizable: false, // No permitir redimensionamiento
16 webPreferences: {
17   preload: path.join(__dirname, 'preload.js'), // Archivo de precarga
18   devTools: isDev // Mostrar herramientas de desarrollo solo en modo dev
19 }
20 }
21
22 // Configuración adicional para producción
23 if (!isDev) {
24   Object.assign(options, {
25     frame: false,
26     movable: false,
27     fullscreen: true
28   }) // Pantalla completa en producción
29 }
30
31 const win = new BrowserWindow(options)
32 win.loadFile(path.join(__dirname, '../renderer/index.html')) // Cargar
    archivo HTML principal
33 }
34
35 // Deshabilitar atajos de teclado en producción
36 if (!isDev) {
37   app.on('browser-window-focus', () => {
38     globalShortcut.register("CommandOrControl+R", () => {
39       console.log("Ctrl + R está deshabilitado")
40     })
41   })
42
43   app.on('browser-window-blur', () => {
44     globalShortcut.unregister("CommandOrControl+R")
45   })
46 }

```

```

47
48 // Array para almacenar los procesos en ejecución
49 const pythonProcesses = []
50
51 // Scripts que se ejecutarán
52 const scriptNames = [
53   '/server/serverSIO.py',
54   '/client/started_ClientSIO.py'
55 ]
56
57 // Función para iniciar procesos ejecutables
58 const launchExecutableProcesses = (executables) => {
59   if (!Array.isArray(executables) || executables.length === 0) {
60     console.error("Debes proporcionar un array con nombres de ejecutables")
61     return
62   }
63
64   executables.forEach((execName, index) => {
65     // Determinar la ruta según si está empaquetado o no
66     const execDir = app.isPackaged
67       ? path.join(process.resourcesPath, 'bin')
68       : path.join(__dirname, '../..bin')
69
70     const execPath = path.join(execDir, execName)
71
72     // Establecer permisos de ejecución en sistemas Unix
73     if (process.platform !== 'win32') {
74       fs.chmodSync(execPath, 0o755)
75     }
76
77     // Iniciar el proceso
78     const childProcess = spawn(execPath, [], {
79       stdio: 'pipe',
80       shell: false
81     })
82
83     // Registrar el proceso

```

```

84 pythonProcesses.push({ name: execName, process: childProcess })
85
86 // Manejar salida estándar
87 childProcess.stdout.on('data', (data) => {
88   console.log(`Proceso ${index + 1} (${execName}): ${data}`)
89 })
90
91 // Manejar errores
92 childProcess.stderr.on('data', (data) => {
93   console.error(`Proceso ${index + 1} (${execName}): ${data}`)
94 })
95
96 // Manejar cierre del proceso
97 childProcess.on('close', (code) => {
98   console.log(`El proceso ${index + 1} (${execName}) se cerró con código:
99   ${code}`)
100 })
101
102 console.log(`Lanzados ${executables.length} procesos ejecutables`)
103 }
104
105 // Función para iniciar procesos Python
106 const launchPythonProcesses = (scripts) => {
107   if (!Array.isArray(scripts) || scripts.length === 0) {
108     console.error("Debes proporcionar un array con nombres de scripts")
109     return
110   }
111
112   scripts.forEach((scriptName, index) => {
113     // Determinar ruta según entorno
114     const pythonDir = app.isPackaged
115       ? path.join(process.resourcesPath, 'python')
116       : path.join(__dirname, '../..python')
117
118     const scriptPath = path.join(pythonDir, scriptName)

```

```

119 const pythonProcess = spawn('python', [scriptPath]) // Iniciar proceso
    Python
120
121 // Registrar el proceso
122 pythonProcesses.push({ name: scriptName, process: pythonProcess })
123
124 // Manejar salida estándar
125 pythonProcess.stdout.on('data', (data) => {
126   console.log(`Proceso ${index + 1} (${scriptName}): ${data}`)
127 })
128
129 // Manejar errores
130 pythonProcess.stderr.on('data', (data) => {
131   console.error(`Proceso ${index + 1} (${scriptName}): ${data}`)
132 })
133
134 // Manejar cierre del proceso
135 pythonProcess.on('close', (code) => {
136   console.log(`El proceso ${index + 1} (${scriptName}) se cerro con el
    codigo: ${code}`)
137 })
138 })
139
140 console.log(`Lanzados ${scripts.length} procesos Python`)
141 }
142
143 // Función para cerrar todos los procesos
144 const closePythonProcesses = () => {
145   pythonProcesses.forEach(( { name, process }, index) => {
146     console.log(`Cerrando proceso ${index + 1} (${name})...`)
147     process.kill() // Terminar proceso
148   })
149
150   pythonProcesses.length = 0 // Limpiar array
151   console.log("Todos los procesos Python han sido cerrados")
152 }
153

```



```

154 // Cuando la aplicación está lista
155 app.whenReady().then(async () => {
156   try {
157     if (isDev) {
158       launchPythonProcesses(scriptNames) // Usar scripts Python en desarrollo
159     } else {
160       launchExecutableProcesses(['server/serverSIO',
161         'client/started_ClientSIO']) // Usar ejecutables en producción
162     }
163     createWindow() // Crear ventana principal
164   } catch (error) {
165     console.error("Error:", error)
166   }
167   return
168 }
169 // Manejador para cargar archivos JSON
170 ipcMain.handle('load-json', (event, relativeFilePath) => {
171   try {
172     const jsonDir = app.isPackaged
173       ? path.join(process.resourcesPath, 'resources')
174       : path.join(__dirname, '../..resources')
175
176     const fullPath = path.join(jsonDir, relativeFilePath)
177     console.log(fullPath)
178
179     if (!fs.existsSync(fullPath)) {
180       throw new Error(`El archivo no existe: ${fullPath}`)
181     }
182
183     const data = fs.readFileSync(fullPath, 'utf8')
184     return JSON.parse(data) // Devolver contenido del JSON
185   } catch (error) {
186     console.error('Error al cargar el archivo JSON:', error)
187     return { error: error.message } // Devolver error
188   }
189 })

```

```

190
191 // Manejador para apagar el equipo
192 ipcMain.handle('shutdown-computer', () => {
193   return new Promise((resolve, reject) => {
194     let command
195
196     // Comando según sistema operativo
197     switch (process.platform) {
198       case 'win32': // Windows
199         command = 'shutdown /s /t 0'
200         break
201       case 'darwin': // macOS
202         command = 'osascript -e "tell app \"System Events\" to shut down"'
203         break
204       case 'linux': // Linux
205         command = 'shutdown now'
206         break
207       default:
208         reject(new Error('Sistema operativo no soportado'))
209         return
210     }
211
212     // Ejecutar comando
213     exec(command, (error, stdout, stderr) => {
214       if (error) {
215         console.error(`Error al apagar: ${error.message}`)
216         reject(error)
217         return
218       }
219       if (stderr) {
220         console.error(`Stderr: ${stderr}`)
221         reject(new Error(stderr))
222         return
223       }
224       console.log(`Apagado iniciado: ${stdout}`)
225       resolve(stdout)
226     })

```

```

227 })
228 })
229
230 // Función de prueba para apagado (comentada)
231 /*
232 function testShutdown() {
233   console.log("Iniciando prueba de apagado...")
234
235   new Promise((resolve, reject) => {
236     let command
237
238     switch (process.platform) {
239       case 'win32':
240         command = 'shutdown /s /t 10'
241         console.log("Comando Windows:", command)
242         break
243       case 'darwin':
244         command = 'osascript -e "tell app \"System Events\" to shut down"'
245         console.log("Comando macOS:", command)
246         break
247       case 'linux':
248         command = 'echo "Prueba: shutdown now"'
249         console.log("Comando Linux:", command)
250         break
251       default:
252         console.error("Sistema operativo no soportado")
253         return
254     }
255
256     exec(command, (error, stdout, stderr) => {
257       if (error) {
258         console.error(`Error en prueba: ${error.message}`)
259         return
260       }
261       if (stderr) {
262         console.error(`Stderr: ${stderr}`)
263         return

```

```

264 }
265 console.log(`Prueba exitosa. Salida: ${stdout}`)
266 })
267 })
268 }
269 */
270
271 // Evento cuando se cierran todas las ventanas
272 app.on('window-all-closed', () => {
273   closePythonProcesses() // Cerrar procesos
274   if (process.platform !== 'windows') {
275     app.quit() // Salir de la aplicación
276   }
277 })

```

TFG\src\main\utils\clientSocketIO.js

```

1  import { io } from "../socket.io.esm.min.js"
2  import {
3    setRunning,
4    setPaused,
5    setError,
6    setSessionState,
7    setNumberSession,
8    setNumberSerie,
9    setParam,
10   setLoading,
11   setCountDown,
12   setReady,
13   setReset,
14   getStatus,
15   setZeroSession
16 } from '../../renderer/renderer.js'
17
18 // Configuración del cliente Socket.IO con autenticación
19 export const socket = io("http://localhost:5000", {
20   extraHeaders: {

```

```

21 Authorization: "Bearer Tvhc]+@Ud*cvNsQwB|.2}I14%E|k3o)+nqz1l1~Tl-
    tox45?pZ"
22 }
23 })
24
25 // =====
26 // Manejadores de eventos del socket
27 // =====
28
29 // Conexión establecida
30 socket.on('connect', () => {
31   console.log('Conectado al servidor WebSocket')
32 })
33
34 // Respuesta genérica del servidor
35 socket.on('my_response', (data) => {
36   console.log('Mensaje recibido del backend:', data)
37   // Mostrar estado de carga o listo según conexiones
38   data.n_conex < 2 ? setLoading() : setReady()
39 })
40
41 // Manejo de estados principales
42 socket.on('setStatus', (data) => {
43   console.log('Mensaje recibido del backend:', data)
44   switch (data.status) {
45     case 'reset':
46       setReset()
47       break
48     case 'error':
49       // Obtener estado actual antes del error
50       socket.emit('getStatusBeforeError', { "getStatusBeforeError":
         getStatus()})
51       setError()
52       break
53     case 'paused':
54       setPaused()
55       break

```

```

56 case 'running':
57   setRunning()
58   break
59 case 'sessionState':
60   setSessionState()
61   break
62 default:
63   console.warn('Estado no reconocido:', data.status)
64 }
65 })
66
67 // Recepción de número de serie
68 socket.on('setNumSerie', (data) => {
69   console.log('Mensaje recibido del backend:', data)
70   setNumberSerie(data.numSerie)
71 })
72
73 // Recepción de parámetros
74 socket.on('setParam', (data) => {
75   console.log('Mensaje recibido del backend:', data)
76   setParam(data.typeSignal, data.amplitude, data.frequency, data.offset)
77 })
78
79 // Recepción de número de sesiones
80 socket.on('setNumSesion', (data) => {
81   console.log('Mensaje recibido del backend:', data)
82   setNumberSession(data.numSesion)
83 })
84
85 // Recepción de contador regresivo
86 socket.on('setCountDown', (data) => {
87   console.log('Mensaje recibido del backend:', data)
88   setCountDown(data.countDown)
89 })
90
91 // Manejo de desconexión
92 socket.on('disconnect', () => {

```

```

93 | console.log('Desconectado del servidor WebSocket')
94 | })
95 |
96 | // =====
97 | // Eventos de UI
98 | // =====
99 |
100 | // Manejador de clic para el botón de acción
101 | document.getElementById("actionButton").addEventListener("click", () =>
    {
102 | // Enviar estado actual al hacer clic
103 | socket.emit('buttonClick', { "buttonClick": getStatus()})
104 | })

```

TFG\resources\data.json

```

1 | {
2 |   "serialNumber": "01052025",
3 |   "nSesiones": 50,
4 |   "timeSesion": 20,
5 |   "typeSignal": "Cuadrada",
6 |   "amplitude": "0-5 V",
7 |   "frequency": "0.1-200 Hz",
8 |   "offset": "0"
9 | }

```

TFG\python\Simulator.py

```

1 | import time
2 | import socketio
3 | import os
4 | import platform
5 | from client.clientSocketIO import ClientSocketIO
6 |
7 | # Configuración inicial
8 | token = "Tvhc]+@Ud*cvNsQwB|.2}I14%E|k3o)+nqz1l1~Tl-tox45?pZ"
9 | client = ClientSocketIO(token=token) # Crear instancia del cliente
10 |

```

```

11 def limpiar_pantalla():
12     if platform.system() == "Windows":
13         os.system("cls") # Comando para Windows
14     else:
15         os.system("clear") # Comando para Linux y macOS
16
17 def send_status(opcion, status):
18     print(f"Has seleccionado la opción {opcion}.")
19     print(f"Enviando estado: {status}...\n")
20     client.send_data('setStatus', 'status', status)
21     time.sleep(1)
22
23 def send_contador(opcion, contador):
24     print(f"Has seleccionado la opción {opcion}.")
25     print(f"Enviando contador: {contador}...\n")
26     client.send_data('setNumSesion', 'numSesion', contador)
27     time.sleep(1)
28
29 def mostrar_menu():
30     print("==== MENÚ PRINCIPAL ====")
31     print("1. Send Running")
32     print("2. Send Pause")
33     print("3. Send Error")
34     print("4. Send Reset")
35     print("5. Send Cargar Sesiones")
36     print("6. Send Contador")
37     print("x. Exit")
38     print("=====")
39
40 def handle_incoming_data(data):
41     """Manejador para datos entrantes"""
42     print(f"\nDato recibido del servidor: {data}")
43     # Aquí puedes añadir más lógica para procesar los datos recibidos
44
45 def main():
46     # Conectar al servidor
47     client.conectar()

```



```

48
49 # Configurar el manejador de datos entrantes
50 client.receive_data('buttonClick', handle_incoming_data)
51
52 try:
53     while True:
54         limpiar_pantalla()
55         mostrar_menu()
56
57         opcion = input("Selecciona una opción: ")
58         print() # Línea vacía para separar visualmente las acciones
59
60         if opcion == '1':
61             send_status(opcion, "running")
62         elif opcion == '2':
63             send_status(opcion, "paused")
64         elif opcion == '3':
65             send_status(opcion, "error")
66         elif opcion == '4':
67             send_status(opcion, "reset")
68         elif opcion == '5':
69             send_status(opcion, "sessionState")
70         elif opcion == '6':
71             contador = input("Numero del contador: ")
72             send_contador(opcion, contador)
73         elif opcion.lower() == 'x':
74             print("Saliendo del simulador...")
75             break
76         else:
77             print("Opción inválida. Por favor, selecciona una opción correcta.\n")
78             time.sleep(3)
79     except KeyboardInterrupt:
80         print("\nInterrupción recibida, saliendo...")
81     finally:
82         client.desconectar()
83         print("Conexión cerrada correctamente")
84

```

```
85 | if __name__ == "__main__":
86 |     main()
```

TFG\python\serial_escucha.py

```
1 | from client.serialEvent import SerialEvent
2 | import time
3 | import os
4 | import platform
5 |
6 | def clear_screen():
7 |     """Limpia la terminal después de 2 segundos"""
8 |     time.sleep(2)
9 |     os.system('cls' if os.name == 'nt' else 'clear')
10 |
11 | if __name__ == "__main__":
12 |     # Detectar el sistema operativo y asignar el puerto serial
13 |     if platform.system() == 'Windows':
14 |         puerto = "COM3" # Puerto común en Windows
15 |     elif platform.system() == 'Linux':
16 |         puerto = "/dev/serial0" # Puerto común en Raspberry Pi
17 |     else:
18 |         raise OSError("Sistema operativo no soportado")
19 |     baudrate = 9600
20 |     evento = "serial_event"
21 |
22 |     listener = SerialEvent(puerto, baudrate, evento)
23 |
24 |     try:
25 |         listener.start(SerialEvent.handle_event)
26 |         print("Conexión serial establecida. Escribe tus mensajes (o 'exit' para salir):")
27 |
28 |         while True:
29 |             mensaje = input("> ") # Prompt para entrada de usuario
30 |
31 |             if mensaje.lower() == 'exit':
32 |                 break
```

```

33
34 listener.send(mensaje)
35 clear_screen()
36 print("Conexión serial activa. Escribe otro mensaje (o 'exit' para
    salir):")
37
38 except KeyboardInterrupt:
39 print("\nInterrupción por teclado detectada.")
40 except Exception as e:
41 print(f"\nError: {str(e)}")
42 finally:
43 listener.stop()

```

TFG\python\server\serverSI0.py

```

1 # ServerSI0.py
2 from threading import Lock
3 from flask import Flask, session, request, copy_current_request_context
4 from flask_socketio import SocketIO, emit, disconnect
5 from engineio.async_drivers import gevent
6
7 class ServerSI0:
8 """Servidor principal para manejar conexiones Socket.IO con
    autenticación y eventos en tiempo real."""
9
10 # Configuración del modo asíncrono (puede ser 'threading', 'eventlet' o
    'gevent')
11 async_mode = 'gevent'
12
13 # Aplicación Flask básica
14 app = Flask(__name__)
15 app.config['SECRET_KEY'] = "Tvhc]+@Ud*cvNsQwB|.2}I14%£|k3o)+nqz1l1~Tl-
    tox45?pZ"
16
17 # Configuración del servidor Socket.IO
18 socketio = SocketIO(app,
19 async_mode=async_mode,

```

```

20 cors_allowed_origins='*', # Permite CORS para todos los orígenes (#
    file://)
21 logger=False, # Desactiva logging estándar
22 engineio_logger=True) # Activa logging de Engine.IO
23
24 thread = None # Hilo para operaciones en segundo plano
25 thread_lock = Lock() # Lock para sincronización de hilos
26 n_conex = 0 # Contador de conexiones activas
27
28 def background_thread():
29     """Ejemplo de hilo en segundo plano para enviar eventos periódicos a
    los clientes."""
30     count = 0
31     while True:
32         ServerSIO.socketio.sleep(10)
33         count += 1
34         ServerSIO.socketio.emit('respuesta',
35             {'data': 'Server generated event', 'count': count})
36
37     # =====
38     # Manejadores de eventos principales
39     # =====
40
41     @socketio.event
42     def setStatus(message):
43         """Actualiza y difunde el estado del sistema a todos los clientes
    excepto al emisor."""
44         session['receive_count'] = session.get('receive_count', 0) + 1
45         emit('setStatus', message, broadcast=True, include_self=False)
46
47     @socketio.event
48     def setNumSession(message):
49         """Actualiza y difunde el número de sesiones disponibles."""
50         session['receive_count'] = session.get('receive_count', 0) + 1
51         emit('setNumSession', message, broadcast=True, include_self=False)
52
53     @socketio.event

```

```

54 def setNumSerie(message):
55     """Actualiza y difunde el número de serie del dispositivo."""
56     session['receive_count'] = session.get('receive_count', 0) + 1
57     emit('setNumSerie', message, broadcast=True, include_self=False)
58
59 @socketio.event
60 def setParam(message):
61     """Actualiza y difunde los parámetros de configuración."""
62     session['receive_count'] = session.get('receive_count', 0) + 1
63     emit('setParam', message, broadcast=True, include_self=False)
64
65 @socketio.event
66 def setCountDown(message):
67     """Actualiza y difunde el estado del contador regresivo."""
68     session['receive_count'] = session.get('receive_count', 0) + 1
69     emit('setCountDown', message, broadcast=True, include_self=False)
70
71 @socketio.event
72 def buttonClick(message):
73     """Procesa y difunde eventos de clic de botón desde los clientes."""
74     session['receive_count'] = session.get('receive_count', 0) + 1
75     emit('buttonClick', message, broadcast=True, include_self=False)
76
77 @socketio.event
78 def getStatusBeforeError(message):
79     """Maneja solicitudes para obtener el estado previo a un error."""
80     session['receive_count'] = session.get('receive_count', 0) + 1
81     emit('getStatusBeforeError', message, broadcast=True,
82         include_self=False)
83
84 @socketio.event
85 def countdownFinished(message):
86     """Notifica cuando el contador regresivo ha finalizado."""
87     session['receive_count'] = session.get('receive_count', 0) + 1
88     emit('countdownFinished', message, broadcast=True, include_self=False)
89
90 # @socketio.on('*')

```

```

90 # def catch_all(event, data):
91 # session['receive_count'] = session.get('receive_count', 0) + 1
92 # emit('my_response',
93 # {'data': [event, data], 'count': session['receive_count']})
94
95 # =====
96 # Manejadores de conexión/desconexión
97 # =====
98
99 @socketio.event
100 def disconnect_request():
101     """Maneja solicitudes de desconexión segura con confirmación."""
102     @copy_current_request_context
103     def can_disconnect():
104         """Función de callback para confirmar desconexión segura."""
105         disconnect()
106
107     session['receive_count'] = session.get('receive_count', 0) + 1
108     ServerSI0.n_conex -= 1
109     emit('my_response',
110         {'data': 'Disconnected!',
111          'count': session['receive_count'],
112          'n_conex': ServerSI0.n_conex},
113         callback=can_disconnect)
114
115 @socketio.event
116 def my_ping():
117     """Responde a mensajes ping para mantener la conexión activa."""
118     emit('my_pong')
119
120 @socketio.event
121 def connect():
122     """Maneja nuevas conexiones con autenticación por token Bearer."""
123     token = request.headers.get('Authorization')
124
125     # Validación del token de autorización

```

```

126 if not token or not token.startswith("Bearer ") or token.split(" ")[1]
    != ServerSIO.app.config['SECRET_KEY']:
127 print("Conexión rechazada: token inválido o ausente")
128 return False # Rechaza la conexión
129 # global thread
130 # with ServerSIO.thread_lock:
131 # if thread is None:
132 # thread = ServerSIO.socketio.start_background_tas-
    k(ServerSIO.queue_sender)
133 ServerSIO.n_conex += 1
134 emit('my_response', {'data': 'Connected', 'n_conex':
    ServerSIO.n_conex}, broadcast=True)
135 # origin = request.headers.get('Origin')
136 # print (f'// {origin}')
137
138 @socketio.on('disconnect')
139 def test_disconnect(reason):
140 """Maneja eventos de desconexión de clientes."""
141 ServerSIO.n_conex -= 1
142 emit('my_response', {'data': 'Connected', 'n_conex':
    ServerSIO.n_conex}, broadcast=True)
143 print('Client disconnected', request.sid, reason)
144
145 if __name__ == '__main__':
146 """Punto de entrada principal para ejecutar el servidor en
    localhost:5000."""
147 ServerSIO.socketio.run(ServerSIO.app, host='127.0.0.1', port=5000)

```

TFG\python\client\started_ClientSIO.py

```

1 import time
2 import socketio
3 import os
4 import platform
5 from clientSocketIO import ClientSocketIO
6 from commandHandlers.jsonCommandHandler import HandleJSON
7 from commandHandlers.socketIOCommandHandler import
    SocketIOCommandHandler

```

```

8  from commandHandlers.serialCommandHandler import SerialCommandHandler
9  from serialEvent import SerialEvent
10 from GPIO_Event import GPIO_Event
11 import sys
12
13 class Started:
14     """
15     Clase principal que inicia y coordina todos los componentes del
16     sistema.
17     Gestiona la comunicación entre Socket.IO, serial y GPIO.
18     """
19     # Configuración inicial de Socket.IO
20     sio = socketio.Client()
21     token = "Tvhc]+@Ud*cvNsQwB|.2}I14%£|k3o)+nqz1l1~Tl-tox45?pZ"
22
23     # Configuración del puerto serial según el sistema operativo
24     if platform.system() == 'Windows':
25         puerto = "COM3" # Puerto común en Windows
26     elif platform.system() == 'Linux':
27         puerto = "/dev/serial0" # Puerto común en Raspberry Pi
28     else:
29         raise OSError("Sistema operativo no soportado")
30     baudrate = 9600
31     evento = "serial_event"
32
33     @staticmethod
34     def get_base_path():
35         """
36         Determina la ruta base correcta tanto para ejecutables como para
37         scripts.
38
39         Returns:
40         str: Ruta base del proyecto
41         """
42         if getattr(sys, 'frozen', False):
43             # Si es un .exe, la ruta base es donde está el ejecutable

```



```

43 return os.path.dirname(sys.executable)
44 else:
45 # Si es un script, usa la ruta del script (__file__)
46 return os.path.dirname(__file__)
47
48 dir_actual = get_base_path() # Directorio del script actual
49 ruta_json = os.path.join(dir_actual, '../..', 'resources/data.json')
50 handleJSON = HandleJSON(ruta_json)
51
52 def start(self):
53 """
54 Método principal que inicia todos los componentes del sistema.
55 Configura y conecta Socket.IO, serial y GPIO con sus respectivos
56 handlers.
57 """
58 try:
59 # Redirigir stdout a stderr temporalmente para capturar logs
60 stdout = sys.stdout
61 sys.stdout = sys.stderr
62
63 # Inicialización de componentes principales
64 clientSocketIO = ClientSocketIO(token=self.token)
65 listener = SerialEvent(self.puerto, self.baudrate, self.evento)
66 gpio = GPIO_Event(TIME_WINDOW=0.5, ACTIVE=False)
67
68 # Configuración de handlers
69 handleJSON = HandleJSON(self.ruta_json)
70 handleSocketIO = SocketIOCommandHandler(handleJSON, gpio,
71 clientSocketIO)
72 handleSerial = SerialCommandHandler(handleJSON, handleSocketIO,
73 listener)
74
75 # Establecimiento de conexiones
76 clientSocketIO.conectar()
77
78 # Configuración de manejadores de eventos
79 clientSocketIO.receive_data('buttonClick', handleSocketIO.click_handle)

```

```

77 clientSocketIO.receive_data('getStatusBeforeError',
    handleSocketIO.error_handle)
78 listener.start(handleSerial.serial_handle)
79
80 # Configuración de GPIO
81 gpio.monitor_pulse_timeout(
82     timeout=0.7,
83     timeout_callback=handleSocketIO.order_error,
84     pulse_after_timeout_callback=handleSocketIO.after_error_handle
85 )
86 gpio.send_pulse(pulses=1)
87
88 # Mantener el programa en ejecución
89 listener.thread.join()
90
91 except Exception as e:
92     print(f"Error en la ejecución principal: {e}")
93 finally:
94     # Restaurar stdout original
95     sys.stdout = stdOut
96
97 if __name__ == "__main__":
98     """
99     Punto de entrada principal del programa.
100     Crea una instancia de Started y ejecuta el sistema.
101     """
102     started = Started()
103     started.start()

```

TFG\python\client\serialEvent.py

```

1 import serial
2 import time
3 import threading
4
5 class SerialEvent:
6     def __init__(self, port, baudrate, evento, timeout=1):
7         """

```

```

8  Inicializa el manejador de eventos seriales con configuración básica.
9
10 Args:
11 port (str): Puerto serial (ej: 'COM3' o '/dev/ttyUSB0')
12 baudrate (int): Velocidad en baudios (ej: 9600)
13 evento (str): Identificador del evento serial
14 timeout (float): Tiempo de espera para operaciones seriales (segundos)
15 """
16 self.port = port
17 self.baudrate = baudrate
18 self.evento = evento
19 self.timeout = timeout
20
21 self.lock = threading.Lock()
22 self.running = False
23 self.connected = False
24 self.ser = None
25 self.thread = None
26
27 def getConnected(self):
28     """
29     Obtiene el estado actual de conexión de forma thread-safe.
30
31     Returns:
32     bool: True si está conectado, False en caso contrario
33     """
34     with self.lock:
35         return self.connected
36
37 def conect(self):
38     """
39     Intenta establecer conexión serial de forma thread-safe.
40
41     Returns:
42     bool: True si la conexión fue exitosa, False en caso contrario
43     """
44     with self.lock:

```

```

45 if self.connected:
46     return True
47 try:
48     self.ser = serial.Serial(
49         self.port,
50         self.baudrate,
51         timeout=self.timeout
52     )
53     self.connected = True
54     print(f"Conexion establecida en {self.port} a {self.baudrate} bps.")
55     return True
56 except serial.SerialException:
57     print(f"Intento conectar en {self.port}.")
58     return False
59
60 def listen(self, func=None, *args, **kwargs):
61     """
62     Escucha continuamente datos del puerto serial y ejecuta callback.
63
64     Args:
65     func (callable): Función callback para procesar datos recibidos
66     *args: Argumentos posicionales adicionales para el callback
67     **kwargs: Argumentos clave adicionales para el callback
68     """
69     print(f"Escuchando evento: {self.evento}")
70     while True:
71         with self.lock:
72             if not self.connected or not self.ser or not self.ser.is_open:
73                 break
74             ser = self.ser
75
76         try:
77             if ser.in_waiting > 0:
78                 data = ser.readline().decode('utf-8').strip()
79                 func(data, *args, **kwargs)
80         except serial.SerialException:
81             print("Conexion perdida.")

```

```

82 break
83 except UnicodeDecodeError:
84     print("Error decodificando datos.")
85 except Exception as e:
86     print(f"Error inesperado: {str(e)}")
87 break
88
89 with self.lock:
90     if self.connected:
91         self.ser.close()
92         self.connected = False
93
94     def send(self, message):
95         """
96         Envía un mensaje por el puerto serial de forma thread-safe.
97
98         Args:
99         message (str): Mensaje a enviar (será codificado a UTF-8)
100         """
101         with self.lock:
102             if not self.connected or not self.ser or not self.ser.is_open:
103                 print("No se puede enviar: desconectado.")
104             return
105
106         try:
107             self.ser.write(message.encode('utf-8'))
108             print(f"Mensaje enviado: {message}")
109         except serial.SerialException as e:
110             print(f"Error enviando datos: {str(e)}")
111             self.connected = False
112             self.ser.close()
113
114     def start(self, *args, **kwargs):
115         """
116         Inicia el hilo principal que gestiona la conexión serial.
117         """
118         with self.lock:

```

```

119     if self.running:
120         return
121     self.running = True
122
123     self.thread = threading.Thread(
124         target=self._manage_connection,
125         args=args,
126         kwargs=kwargs,
127         daemon=True
128     )
129     self.thread.start()
130
131     def _manage_connection(self, *args, **kwargs):
132         """
133         Gestiona el ciclo de conexión/reconexión del puerto serial.
134         Método interno ejecutado en un hilo separado.
135         """
136         while self.running:
137             if self.conect():
138                 listen_thread = threading.Thread(
139                     target=self.listen,
140                     args=args,
141                     kwargs=kwargs,
142                     daemon=True
143                 )
144                 listen_thread.start()
145                 listen_thread.join()
146
147             with self.lock:
148                 if self.ser and self.ser.is_open:
149                     self.ser.close()
150                     self.connected = False
151
152             time.sleep(1)
153
154     def stop(self):
155         """

```

```

156 Detiene todos los hilos y cierra la conexión serial de forma segura.
157 """
158 with self.lock:
159     self.running = False
160     if self.ser and self.ser.is_open:
161         self.ser.close()
162         self.connected = False
163         self.ser = None
164     print("Conexion serial cerrada.")
165
166 @staticmethod
167 def handle_event(data):
168     """
169     Manejador por defecto para eventos seriales (puede ser sobrescrito).
170
171     Args:
172     data (str): Datos recibidos del puerto serial
173     """
174     print(f"Mensaje recibido: {data}")
175
176 # if __name__ == "__main__":
177 #     puerto = "COM3"
178 #     baudrate = 9600
179 #     evento = "serial_event"
180
181 #     listener = SerialEvent(puerto, baudrate, evento,
182 #                             SerialEvent.handle_event)
183
184 #     try:
185 #         listener.start()
186 #         while True:
187 #             listener.send("Hola desde la PC")
188 #             time.sleep(5)
189 #         except KeyboardInterrupt:
190 #             print("\nDeteniendo...")
191 #         finally:
192 #             listener.stop()

```

TFG\python\client\GPIO_Event.py

```
1  from gpiozero import DigitalInputDevice, DigitalOutputDevice
2  from gpiozero import Device
3  #from gpiozero.pins.rpigpio import RPiGPIOFactory
4  import time
5
6  class GPIO_Event():
7  def __init__(self, INPUT_PIN=17, OUTPUT_PIN=27, SAMPLE_RATE=0.01,
8  ACTIVATION_THRESHOLD=2, TIME_WINDOW=0.5, ACTIVE=True):
9  """
10
11  Inicializa el controlador de eventos GPIO con parámetros configurables.
12
13  Args:
14  INPUT_PIN (int): Pin GPIO para entrada (por defecto 17)
15  OUTPUT_PIN (int): Pin GPIO para salida (por defecto 27)
16  SAMPLE_RATE (float): Intervalo de muestreo en segundos (por defecto
17  0.01)
18  ACTIVATION_THRESHOLD (int): Umbral de pulsos para activación (por
19  defecto 2)
20  TIME_WINDOW (float): Ventana temporal para contar pulsos en segundos
21  (por defecto 0.5)
22  ACTIVE (bool): Habilitar/deshabilitar funcionalidad (por defecto True)
23  """
24  self.INPUT_PIN = INPUT_PIN
25  self.OUTPUT_PIN = OUTPUT_PIN
26  self.SAMPLE_RATE = SAMPLE_RATE # Muestreo (10 ms)
27  self.ACTIVATION_THRESHOLD = ACTIVATION_THRESHOLD # Número de pulsos
28  requeridos
29  self.TIME_WINDOW = TIME_WINDOW # Ventana de tiempo en segundos
30  self.ACTIVE = ACTIVE
31
32  def send_pulse(self, duration=0.3, pulses=1, interval=0.3):
33  """
34
35  Envía uno o varios pulsos por el pin de salida configurado.
36
37  Args:
```



```

32 duration (float): Duración de cada pulso en segundos (por defecto 0.3)
33 pulses (int): Número de pulsos a enviar (por defecto 1)
34 interval (float): Tiempo entre pulsos en segundos (por defecto 0.3)
35 """
36 if self.ACTIVE == False:
37     return
38 if self.OUTPUT_PIN is not None:
39     try:
40         output_device = DigitalOutputDevice(self.OUTPUT_PIN)
41         for i in range(pulses):
42             output_device.on()
43             time.sleep(duration)
44             output_device.off()
45             print(f"Pulso {i+1}/{pulses} enviado por el pin {self.OUTPUT_PIN}
46                 durante {duration} segundos")
47         # No esperar después del último pulso
48         if i < pulses - 1:
49             time.sleep(interval)
50         finally:
51             output_device.close()
52     else:
53         print("Error: No se ha configurado un pin de salida (OUTPUT_PIN)")
54
55 def start_INPUT_GPIO(self, callback=None, *args, **kwargs):
56     """
57     Monitorea pulsos de entrada y ejecuta callback al alcanzar umbral.
58
59     Args:
60     callback (function): Función a ejecutar al activarse
61     *args: Argumentos posicionales para el callback
62     **kwargs: Argumentos clave para el callback
63     """
64     if self.ACTIVE == False:
65         return
66
67     sensor = DigitalInputDevice(self.INPUT_PIN, pull_up=True)

```

```

68 last_state = sensor.value
69 pulse_times = [] # Almacenará tiempos en segundos
70
71 try:
72     while True:
73         current_state = sensor.value
74
75         # Detectar flanco de subida (LOW → HIGH)
76         if current_state and not last_state:
77             current_time = time.time()
78             print(f"¡Pulso detectado! (Tiempo: {current_time} s)")
79             pulse_times.append(current_time)
80
81         # Filtrar pulsos fuera de la ventana
82         pulse_times = [t for t in pulse_times if (current_time - t) <=
            self.TIME_WINDOW]
83
84         # Verificar si hay suficientes pulsos
85         if len(pulse_times) >= self.ACTIVATION_THRESHOLD:
86             print(f"¡Orden activada! ({len(pulse_times)} pulsos en
                {self.TIME_WINDOW} s)")
87
88         if callback:
89             callback(*args, **kwargs)
90
91         pulse_times = [] # Reiniciar contador
92
93         last_state = current_state
94         time.sleep(self.SAMPLE_RATE)
95
96     except KeyboardInterrupt:
97         print("\nPrograma terminado")
98     finally:
99         sensor.close()
100
101 def monitor_pulse_timeout(
102     self,

```

```

103 timeout=5.0, # Tiempo máximo permitido en estado LOW
104 timeout_callback=None, # Se ejecuta si LOW dura más del timeout
105 pulse_after_timeout_callback=None # Se ejecuta cuando vuelve a HIGH
    después del timeout
106 ):
107     """
108     Monitoriza la entrada GPIO y ejecuta callbacks según:
109     - Si el estado LOW dura más de 'timeout' segundos
110     - Cuando vuelve a HIGH después de un timeout
111
112     Args:
113     timeout (float): Máxima duración permitida en estado LOW (segundos)
114     timeout_callback (callable): Ejecutar cuando LOW supera el timeout
115     pulse_after_timeout_callback (callable): Ejecutar al volver a HIGH
    después de timeout
116     """
117     if self.ACTIVE == False:
118         print("Desactivado GPIO")
119         return
120
121     if self.INPUT_PIN is None:
122         print("Error: No se ha configurado INPUT_PIN")
123         return
124
125     sensor = DigitalInputDevice(self.INPUT_PIN, pull_up=True)
126     low_state_start = None
127     timeout_triggered = False
128     last_state = 1 # Asumimos estado inicial HIGH (pull-up)
129
130     try:
131         while True:
132             current_state = sensor.value
133             current_time = time.time()
134
135             # Detección de flanco de bajada (HIGH -> LOW)
136             if current_state == 0 and last_state == 1:
137                 low_state_start = current_time

```

```

138 timeout_triggered = False
139 print(f"Estado LOW detectado (Tiempo: {current_time})")
140
141 # Detección de flanco de subida (LOW -> HIGH)
142 elif current_state == 1 and last_state == 0:
143     low_duration = current_time - low_state_start if low_state_start else 0
144     print(f"Estado HIGH recuperado. LOW duró: {low_duration:.3f}s")
145
146 # Si habíamos tenido timeout y ahora recuperamos HIGH
147 if timeout_triggered and pulse_after_timeout_callback:
148     pulse_after_timeout_callback()
149     timeout_triggered = False
150
151 # Verificar timeout en estado LOW
152 if current_state == 0 and low_state_start and not timeout_triggered:
153     low_duration = current_time - low_state_start
154     if low_duration >= timeout:
155         print(f"Timeout LOW: Estado bajo por {low_duration:.3f}s")
156         if timeout_callback:
157             timeout_callback()
158             timeout_triggered = True
159
160 last_state = current_state
161 time.sleep(self.SAMPLE_RATE)
162
163 except KeyboardInterrupt:
164     print("\nMonitorización terminada")
165 finally:
166     sensor.close()
167
168 def visual_state():
169     """
170     Muestra cambios de estado en tiempo real para diagnóstico.
171     Monitorea el pin GPIO 17 y muestra transiciones HIGH/LOW con
172     duraciones.
173     """
174     sensor = DigitalInputDevice(17, pull_up=True)

```

```

174 last_state = sensor.value
175 last_change_time = time.monotonic() # Más preciso para mediciones de
    tiempo
176
177 print(f"Estado inicial: {'HIGH' if last_state else 'LOW'}")
178
179 while True:
180     current_state = sensor.value
181     if current_state != last_state:
182         current_time = time.monotonic()
183         duration = current_time - last_change_time
184
185     print(f"[{time.strftime('%H:%M:%S')}] Cambio: {'HIGH' if last_state
        else 'LOW'}→"
186           f"{'HIGH' if current_state else 'LOW'} | "
187           f"Duración: {duration:.3f}s")
188
189     last_state = current_state
190     last_change_time = current_time
191
192     time.sleep(0.01)
193     #GPIO_Event.visual_state()

```

TFG\python\client\clientSocketIO.py

```

1  # clientSocketIO.py
2
3  import time
4  import socketio
5
6
7  class ClientSocketIO:
8      def __init__(self, sio=socketio.Client(), token=None):
9          """
10         Inicializa el cliente Socket.IO con configuración básica.
11
12         Args:

```

```

13 sio (socketio.Client): Instancia del cliente Socket.IO (por defecto
nueva instancia)
14 token (str): Token de autenticación para conexión (opcional)
15 """
16 self.sio = sio
17 self.token = token
18
19 def conectar(self):
20     """
21     Establece conexión con el servidor Socket.IO con reintentos automáticos.
22     Intenta conectarse indefinidamente hasta lograr conexión exitosa.
23     """
24     intentos = 0
25     while True:
26         try:
27             print(f"Intentando conectar al servidor... (Intento {intentos + 1})")
28             self.sio.connect('http://localhost:5000', headers={
29                 'Authorization': f'Bearer {self.token}'
30             })
31             print("Conexion exitosa: SocketIO")
32             break
33         except Exception as e:
34             print(f"Error al conectar: {e}")
35             intentos += 1
36             time.sleep(1)
37
38     def receive_data(self, evento, func=None, *args, **kwargs):
39         """
40         Configura un manejador para eventos recibidos del servidor.
41
42         Args:
43         evento (str): Nombre del evento a escuchar
44         func (callable): Función callback para procesar los datos (opcional)
45         *args: Argumentos posicionales adicionales para el callback
46         **kwargs: Argumentos clave adicionales para el callback
47         """
48         @self.sio.on(evento)

```

```

49 def on_message(data):
50     print(f'\nI received a message! Datos: {data}')
51     if func is not None:
52         func(data, *args, **kwargs)
53
54     def send_data(self, fun, event, data):
55         """
56         Envía datos al servidor empaquetados en un diccionario.
57
58         Args:
59         fun (str): Nombre del evento a emitir
60         event (str): Clave para los datos en el payload
61         data: Valor a enviar (será convertido a string)
62         """
63         self.sio.emit(fun, {event: f'{data}'})
64
65     def send_data_full(self, fun, data):
66         """
67         Envía datos completos al servidor sin formato específico.
68
69         Args:
70         fun (str): Nombre del evento a emitir
71         data: Datos a enviar (cualquier formato serializable)
72         """
73         self.sio.emit(fun, data)
74
75     def desconectar(self):
76         """Cierra la conexión con el servidor Socket.IO."""
77         self.sio.disconnect()

```

TFG\python\client\commandHandlers\socketIOCommandHandler.py

```

1 import time
2
3 class SocketIOCommandHandler():
4     """Manejador de comandos para comunicaciones Socket.IO con el
5     frontend."""

```

```

6 control_repeat = False # Control para evitar repetición de comandos
7 getStatusBeforeError = 'container init' # Almacena el estado previo a
  un error
8 inError = False # Indica si el sistema está en estado de error
9
10 def __init__(self, handleJSON, gpio, clientSocketIO):
11     """
12     Inicializa el manejador de comandos.
13
14     Args:
15     handleJSON (HandleJSON): Instancia para manejar el archivo JSON de
        configuración
16     gpio (GPIO_Event): Instancia para controlar los pines GPIO
17     clientSocketIO (ClientSocketIO): Cliente para comunicación Socket.IO
18     """
19     self.handleJSON = handleJSON
20     self.gpio = gpio
21     self.clientSocketIO = clientSocketIO
22
23     def order_start(self):
24         """Envía comando para iniciar sesión y actualiza el contador."""
25         self.handleJSON.editJsonDec('nSesiones')
26         nSesiones = self.handleJSON.getJsonValue('nSesiones')
27         self.clientSocketIO.send_data('setStatus', 'status', "running")
28         self.clientSocketIO.send_data('setNumSesion', 'numSesion', nSesiones)
29         print("Se manda: running")
30
31     def order_pause(self):
32         """Envía comando para pausar la sesión."""
33         self.clientSocketIO.send_data('setStatus', 'status', "paused")
34         print("Se manda: paused")
35
36     def order_play(self):
37         """Envía comando para reanudar sesión pausada."""
38         self.clientSocketIO.send_data('setStatus', 'status', "running")
39         print("Se manda: running")
40

```



```

41 def order_error(self):
42     """Envía comando para indicar estado de error."""
43     self.clientSocketIO.send_data('setStatus', 'status', "error")
44     print("Se manda: error")
45
46 def order_sessionState(self):
47     """Envía comando para estado de carga de sesiones."""
48     self.clientSocketIO.send_data('setStatus', 'status', "sessionState")
49     print("Se manda: sessionState")
50
51 def order_SerialNumber(self, nSerie):
52     """Envía comando para actualizar número de serie.
53
54     Args:
55     nSerie (str): Nuevo número de serie
56     """
57     self.clientSocketIO.send_data('setNumSerie', 'numSerie', nSerie)
58     print("Se manda: setNumSerie")
59
60 def order_CountDown(self, timeSesion):
61     """Envía comando para actualizar tiempo de sesión.
62
63     Args:
64     timeSesion (str): Nuevo tiempo de sesión
65     """
66     self.clientSocketIO.send_data('setCountDown', 'countDown', timeSesion)
67     print("Se manda: setCountDown")
68
69 def order_reset(self, nSesiones=None):
70     """Envía comando para resetear el sistema.
71
72     Args:
73     nSesiones (int, optional): Número de sesiones a enviar
74     """
75     if nSesiones != None:
76         self.clientSocketIO.send_data('setNumSesion', 'numSesion', nSesiones)
77         self.clientSocketIO.send_data('setStatus', 'status', "reset")

```

```

78 self.control_repeat = False
79
80 def order_reset_with_error(self, nSesiones=None):
81     """Envía comando reset, considerando estado de error.
82
83     Args:
84     nSesiones (int, optional): Número de sesiones a enviar
85     """
86     if self.inError == True:
87         self.order_error()
88     else:
89         if nSesiones != None:
90             self.clientSocketIO.send_data('setNumSesion', 'numSesion', nSesiones)
91             self.clientSocketIO.send_data('setStatus', 'status', "reset")
92             self.control_repeat = False
93
94     def order_param(self, param):
95         """Envía comando para actualizar parámetros de sesión.
96
97     Args:
98     param (dict): Diccionario con parámetros a actualizar
99     """
100     self.clientSocketIO.send_data_full('setParam', param)
101     print("Se manda: setParam")
102
103     def click_handle(self, last_data):
104         """Maneja eventos de clic desde el frontend.
105
106     Args:
107     last_data (dict): Datos recibidos del frontend
108     """
109     try:
110         if self.control_repeat == False:
111             self.control_repeat = True
112             self.gpio.send_pulse(pulses=1)
113             if last_data['buttonClick'] == 'container init':
114                 self.order_start()

```

```

115 elif last_data['buttonClick'] == 'container running':
116     self.order_pause()
117 elif last_data['buttonClick'] == 'container paused':
118     self.order_play()
119     time.sleep(0.2)
120     self.control_repeat = False
121 except Exception as e:
122     print(f"Fallo en click_handle: {e}")
123
124 def error_handle(self, last_data):
125     """Maneja eventos de error desde el frontend.
126
127     Args:
128     last_data (dict): Datos recibidos del frontend
129     """
130     if self.control_repeat == False:
131         self.control_repeat = True
132
133     self.inError = True
134     if last_data['getStatusBeforeError'] != 'container error':
135         self.getStatusBeforeError = last_data['getStatusBeforeError']
136
137     time.sleep(0.2)
138     self.control_repeat = False
139
140 def after_error_handle(self):
141     """Maneja recuperación después de un error."""
142     nSesiones = self.handleJSON.getJsonValue('nSesiones')
143     try:
144         if self.control_repeat == False:
145             self.control_repeat = True
146             self.inError = False
147             if self.getStatusBeforeError == 'container init' or
148                 self.getStatusBeforeError == 'container sessionState':
149                 self.order_reset(nSesiones)
150             elif self.getStatusBeforeError == 'container running':
151                 self.order_play()

```

```

151 self.gpio.send_pulse(pulses=1)
152 elif self.getStatusBeforeError == 'container paused':
153     self.order_pause()
154     self.gpio.send_pulse(pulses=1)
155     time.sleep(0.2)
156     self.control_repeat = False
157 except Exception as e:
158     print(f"Fallo en click_handle: {e}")

```

TFG\python\client\commandHandlers\serialCommandHandler.py

```

1  # serial_commands.py
2  import time
3
4  class SerialCommandHandler:
5      """Manejador de comandos seriales para procesar y ejecutar operaciones
6      con JSON y Socket.IO."""
7
8      def __init__(self, handleJSON, handleSocketIO, listener):
9          """
10             Inicializa el manejador de comandos seriales con sus dependencias.
11
12             Args:
13             handleJSON (HandleJSON): Instancia para operaciones con archivos JSON
14             handleSocketIO (SocketIOCommandHandler): Instancia para comunicaciones
15             Socket.IO
16             listener (SerialEvent): Instancia de listener serial para enviar
17             respuestas
18             """
19             self.handleJSON = handleJSON
20             self.listener = listener # Listener como dependencia para enviar
21             respuestas
22             self.handleSocketIO = handleSocketIO
23
24             def serial_handle(self, command):
25                 """
26                 Método principal para procesar comandos seriales entrantes y dirigirlos
27                 a los manejadores adecuados.

```

```

23
24 Args:
25 command (str): Comando en texto plano recibido por conexion serial
26 """
27 parts = command.split()
28 if not parts or len(parts) <= 1:
29     print("Error: Comando vacio")
30     return
31
32 # Caso especial para el comando "Get SerialNumber"
33 if str(command).lower() == "get serialnumber":
34     return self._get_serial_number()
35
36 # Validacion básica de estructura del comando
37 if len(parts) < 3:
38     print("Error: Formato de comando incorrecto")
39     return
40
41 cmd_type, cmd, *values = parts # Descompone los valores restantes
42
43 cmd_type = str(cmd_type).lower()
44 cmd = str(cmd).lower()
45
46 values = [str(valor).replace('_', ' ') for valor in values]
47
48 handlers = {
49     'set': {
50         'serialnumber': lambda v: self._set_serial_number(v[0]),
51         'numbersession': lambda v: self._set_number_session(v[0]),
52         'countdown': lambda v: self._set_countdown(v[0]),
53         'param': lambda v: self._set_param(*v),
54     }
55 }
56
57 # Ejecucion del comando
58 try:
59     if cmd_type in handlers and cmd in handlers[cmd_type]:

```

```

60 handlers[cmd_type][cmd](values)
61 else:
62     print(f"Comando no reconocido: {command}")
63 except Exception as e:
64     print(f"Error al ejecutar '{command}': {str(e)}")
65
66 def _get_serial_number(self):
67     """
68     Maneja el comando 'Get SerialNumber' - obtiene y envia el numero de
        serie actual.
69
70     Returns:
71     str: Numero de serie actual del JSON
72     """
73     serial_number = self.handleJSON.getJsonValue('serialNumber')
74     self.listener.send(f"{serial_number}") # Usa el listener para enviar
        respuesta
75     print(f"Dato enviado por serial: {serial_number}")
76     return serial_number
77
78 def _set_serial_number(self, value):
79     """
80     Maneja el comando 'Set SerialNumber' - actualiza el numero de serie en
        el sistema.
81
82     Args:
83     value (str): Nuevo valor para el numero de serie
84     """
85     try:
86         self.handleSocketIO.order_SerialNumber(value)
87         self.handleJSON.editJson('serialNumber', value)
88         print(f"Numero de serie actualizado a: {value}")
89     except Exception as e:
90         print(f"Error al actualizar serial number: {e}")
91
92 def _set_number_session(self, value):
93     """

```

```

194 Maneja el comando 'Set NumberSession' - actualiza el contador de
195 sesiones.
196
197 Args:
198 value (str): Nuevo valor para el contador (debe ser convertible a int)
199 """
200 try:
201     n_sesion = int(value)
202     self.handleSocketIO.order_sessionState()
203     self.handleJSON.editJson('nSesiones', n_sesion)
204     time.sleep(2)
205     self.handleSocketIO.order_reset_with_error(n_sesion)
206     print(f"Numero de sesion actualizado a: {n_sesion}")
207 except ValueError:
208     print("Error: El valor debe ser un numero entero")
209
210 def _set_countdown(self, value):
211     """
212     Maneja el comando 'Set Countdown' - actualiza la duracion de sesion.
213
214     Args:
215     value (str): Nueva duracion (debe ser convertible a float)
216     """
217     try:
218         time_val = float(value)
219         self.handleSocketIO.order_CountDown(value)
220         self.handleJSON.editJson('timeSesion', time_val)
221         print(f"Tiempo de sesion actualizado a: {value}")
222     except ValueError:
223         print("Error: El valor debe ser un numero")
224
225 def _set_param(self, typeSignal, amplitude, frequency, offset):
226     """
227     Maneja el comando 'Set Param' - actualiza multiples parámetros de
228     señal.
229
230     Args:

```

```

129 typeSignal (str): Tipo de señal
130 amplitude (str): Amplitud de señal
131 frequency (str): Frecuencia de señal
132 offset (str): Offset de señal
133 """
134 try:
135     typeSignal = str(typeSignal)
136     amplitude = str(amplitude)
137     frequency = str(frequency)
138     offset = str(offset)
139     value={
140         "typeSignal": typeSignal,
141         "amplitude": amplitude,
142         "frequency": frequency,
143         "offset": offset
144     }
145     self.handleSocketIO.order_param(value)
146     self.handleJSON.editJson('typeSignal', typeSignal)
147     self.handleJSON.editJson('amplitude', amplitude)
148     self.handleJSON.editJson('frequency', frequency)
149     self.handleJSON.editJson('offset', offset)
150     print(f"Parámetros cambiados: Tipo Señal: {typeSignal}, Amplitud:
        {amplitude}, Frecuencia: {frequency}, Offset: {offset}")
151 except ValueError:
152     print("Error: Los valores deben ser texto")

```

TFG\python\client\commandHandlers\jsonCommandHandler.py

```

1 import json
2
3 class HandleJSON:
4     """Clase para manejar operaciones de lectura y escritura en archivos
        JSON."""
5
6     def __init__(self, json_path):
7         """
8         Inicializa el manejador JSON con la ruta al archivo.
9

```



```

10  Args:
11  json_path (str): Ruta completa al archivo JSON que se va a manejar
12  """
13  self.json_path = json_path
14
15  def editJson(self, key, value):
16  """
17  Modifica un valor específico en el archivo JSON.
18
19  Args:
20  key (str): Clave del valor a modificar
21  value (any): Nuevo valor a asignar (debe ser serializable en JSON)
22  """
23  with open(self.json_path, 'r', encoding='utf-8') as f:
24  datos = json.load(f)
25  datos[key] = value
26  with open(self.json_path, 'w', encoding='utf-8') as f:
27  json.dump(datos, f, indent=4)
28
29  def getJson(self):
30  """
31  Obtiene todos los datos del archivo JSON.
32
33  Returns:
34  dict: Diccionario con todos los datos del archivo JSON
35  """
36  with open(self.json_path, 'r', encoding='utf-8') as f:
37  datos = json.load(f)
38  return datos
39
40  def getJsonValue(self, key):
41  """
42  Obtiene un valor específico del archivo JSON.
43
44  Args:
45  key (str): Clave del valor a recuperar
46

```

```

47 Returns:
48 any: Valor asociado a la clave solicitada
49 """
50 with open(self.json_path, 'r', encoding='utf-8') as f:
51     datos = json.load(f)
52     return datos[key]
53
54 def editJsonDec(self, key):
55     """
56     Decrementa en 1 un valor numérico en el archivo JSON (si es mayor que
57     0).
58
59     Args:
60     key (str): Clave del valor numérico a decrementar
61
62     Returns:
63     int: Nuevo valor después del decremento, o 0 si ya era 0 o menor
64     """
65     with open(self.json_path, 'r', encoding='utf-8') as f:
66         datos = json.load(f)
67         if datos[key] > 0:
68             datos[key] = datos[key]-1
69         with open(self.json_path, 'w', encoding='utf-8') as f:
70             json.dump(datos, f, indent=4)
71         return datos[key]
72     return 0

```

TFG\ESP8266\Emisor_receptor.ino

```

1  const int inputPin = 4; // D2 - Pin para recibir señal externa
2  const int outputPin = 5; // D1 - Pin para enviar señal
3  const int ledPin = LED_BUILTIN; // LED integrado (GPIO2)
4
5  // Configuración de pulsos para el modo emisor
6  unsigned long pulseDuration = 500; // 500ms de pulso
7  unsigned long pulseInterval = 3000; // 3 segundos entre pulsos
8  unsigned long lastPulseTime = 0;
9  bool pulseActive = false;

```

```

10
11 // Para el modo receptor
12 unsigned long lastSignalTime = 0;
13 bool externalSignalActive = false;
14
15 void setup() {
16   Serial.begin(9600);
17
18   // Configuración de pines
19   pinMode(ledPin, OUTPUT);
20   pinMode(outputPin, OUTPUT);
21   pinMode(inputPin, INPUT);
22
23   // Estado inicial
24   digitalWrite(outputPin, LOW); // Inicia en HIGH (pulso inactivo)
25   digitalWrite(ledPin, HIGH); // LED apagado
26
27   Serial.println("\nSistema iniciado - Modo Emisor/Receptor Simultaneo");
28   Serial.println("Emisor: Generando pulsos cada " + String(pulseInterval)
29     + "ms");
30   Serial.println("Receptor: Monitoreando señal en pin D2");
31   }
32
33 void loop() {
34   unsigned long currentTime = millis();
35
36   // --- MODO EMISOR ---
37   // Genera pulsos periódicos
38   if (!pulseActive && (currentTime - lastPulseTime >= (pulseInterval +
39     pulseDuration))) {
40     // Inicia nuevo pulso
41     digitalWrite(outputPin, HIGH); // Activa el pulso (LOW para activar)
42     pulseActive = true;
43     lastPulseTime = currentTime;
44     Serial.println("Emisor: Pulso INICIADO");
45   }
46 }

```

```

45  if (pulseActive && (currentTime - lastPulseTime >= pulseDuration)) {
46  // Termina el pulso
47  digitalWrite(outputPin, LOW); // Desactiva el pulso
48  pulseActive = false;
49  Serial.println("Emisor: Pulso TERMINADO");
50  }
51
52  // --- MODO RECEPTOR ---
53  // Detecta señales externas
54  int signalState = digitalRead(inputPin);
55
56  if (signalState == HIGH && !externalSignalActive) {
57  // Señal externa detectada
58  externalSignalActive = true;
59  lastSignalTime = currentTime;
60  digitalWrite(ledPin, LOW); // Enciende el LED (activo bajo)
61  Serial.println("Receptor: Señal EXTERNA detectada - LED ON");
62  }
63  else if (signalState == LOW && externalSignalActive) {
64  // Señal externa terminó
65  externalSignalActive = false;
66  digitalWrite(ledPin, HIGH); // Apaga el LED
67  Serial.println("Receptor: Señal externa terminada - LED OFF");
68  }
69
70  // Pequeña pausa para evitar fluctuaciones
71  delay(10);
72  }

```

TFG\Help-files\autostart.desktop

[Desktop Entry]

Name=tfgr

Exec=/home/pi/Documentos/TFG-Dev-in-RPi/Help-files/start onboot

Icon=/home/pi/Documentos/TFG/src/assets/icon.png

Terminal=false

Type=Application

X-GNOME-Autostart-enabled=true

Hidden=false
NoDisplay=false

TFG\Help-files\compiler

```
1  #!/bin/bash
2
3  # Script to compile the client, server, and app components
4
5  # Get the absolute directory where this script is located
6  SCRIPT_DIR=$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)
7  PROJECT_ROOT=$(dirname "$SCRIPT_DIR")
8
9  # Activate Python virtual environment
10 VENV_PATH="$PROJECT_ROOT/.venv/bin/activate"
11 #VENV_PATH="/home/pi/Documentos/TFG 1504/TFG-New-
    features/.venv/bin/activate"
12 if [ -f "$VENV_PATH" ]; then
13     source "$VENV_PATH"
14 else
15     echo "Error: Virtual environment not found at $VENV_PATH"
16     exit 1
17 fi
18
19 # Compile client
20 CLIENT_SRC="$PROJECT_ROOT/python/client"
21 CLIENT_DEST="$PROJECT_ROOT/bin/client"
22 (
23     cd "$CLIENT_SRC" || exit 1
24     rm -rf build/ dist/
25     rm -f *.spec
26     pyinstaller --onefile started_ClientSI0.py
27     cp -f dist/started_ClientSI0 "$CLIENT_DEST/"
28     rm -rf build/ dist/
29     rm -f *.spec
30     echo "##### Client Compiled #####"
31 )
```

```

32 |
33 | # Compile server
34 | SERVER_SRC="$PROJECT_ROOT/python/server"
35 | SERVER_DEST="$PROJECT_ROOT/bin/server"
36 | (
37 | cd "$SERVER_SRC" || exit 1
38 | rm -rf build/ dist/
39 | rm -f *.spec
40 | pyinstaller --onefile serverSIO.py
41 | cp -f dist/serverSIO "$SERVER_DEST/"
42 | rm -rf build/ dist/
43 | rm -f *.spec
44 | echo "##### Server Compiled #####"
45 | )
46 |
47 | # Compile app
48 | (
49 | cd "$PROJECT_ROOT" || exit 1
50 | npm run dist:linux
51 | echo "##### App Compiled #####"
52 | )

```

TFG\Help-files\CreateVenv

```

1 | #!/bin/bash
2 |
3 | cd ..
4 | python -m venv .venv
5 | source .venv/bin/activate
6 | pip install -r requirements.txt
7 |
8 | #electron
9 | npm install electron

```

TFG\Help-files\start

```

1 | #!/bin/bash
2 |

```

```
3 | cd /home/pi/Documentos/TFG/dist/linux-arm64-unpacked
4 | ./tfg
5 |
```

Por último, la capa de comunicación emplea el protocolo Socket.IO, implementado en el frontend mediante la librería cliente oficial (archivo `socket.io.esm.min.js`, v4.8.1).

