UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Diseño e implementación de un sistema software modular para gestión de protocolos en una plataforma de biorreactores ChiBio

Trabajo Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

AUTOR/A: Martinez Gomez, Eloy

Tutor/a: Picó Marco, Jesús Andrés

Cotutor/a: Díaz Iza, Harold José

CURSO ACADÉMICO: 2023/2024

RESUMEN

La automatización de protocolos en el ámbito de la biología sintética es fundamental para optimizar el rendimiento experimental, garantizar la reproducibilidad y estandarizar los procesos. Los biorreactores a pequeña escala se han posicionado como herramientas esenciales para la caracterización de circuitos biológicos sintéticos y la evaluación de rendimientos y productividades en una etapa pre-industrial. Estos dispositivos permiten realizar experimentos en modos de operación lote, semi-lote y continuo, todo ello a costes reducidos. Además, el uso de baterías con múltiples biorreactores posibilita la ejecución de protocolos avanzados de cultivo celular, facilitando la optimización experimental de construcciones biológicas sintéticas y la implementación de estrategias de evolución dirigida.

Para aprovechar plenamente estas capacidades, es crucial contar con un software robusto de gestión de protocolos que facilite la comunicación entre los biorreactores y el ordenador de control. Este debe integrar diversas estrategias de control realimentado y sensores (observadores) para las variables pertinentes. La programación modular es esencial para definir, implementar y ejecutar protocolos complejos de manera eficiente. Es necesario un sistema que permita la rápida definición e integración de nuevos módulos de control y observación, garantizando así la flexibilidad y adaptabilidad a diferentes necesidades experimentales.

Palabras clave: biorreactores ChiBio, Django, Python

ÍNDICE

DOCUMENTOS CONTENIDOS EN EL TFG

- MEMORIA
- PRESUPUESTO
- ANEXO I

ÍNDICE DE LA MEMORIA

ÍNDICE DEL PRESUPUESTO

RESUMEN EJECUTIVO

CONCEPT (ABET)	CONCEPTO (traducción)	¿Cumple? (S/N)	¿Dónde? (paginas)
1. IDENTIFY:	1. IDENTIFICAR:	S	10-16
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	10
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Restricciones (normas, códigos, necesidades, requisitos y especificaciones)	S	11-12
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	14-16
2. FORMULATE:	2. FORMULAR:	S	12
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	12
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	12
3. SOLVE:	3. RESOLVER:	S	19-35
3.1. Fulfilment of goals	3.1. Cumplimiento de objetivos	S	19-35
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Impacto global y alcance (contribuciones y recomendaciones prácticas)	S	19-35

CAPÍTULO 1. INTRODUCCIÓN

Actualmente, la automatización juega un papel crucial en todos los ámbitos que incluyen procesos manuales. Reducir errores, tiempo y costes son las principales razones por las que se ha vuelto tan importante. Podemos separar la automatización en dos términos: hardware y software. Este Trabajo de Fin de Grado (TFG) se centra en el último. El software es la forma de comunicarse con los ordenadores mediante la configuración de instrucciones específicas en un lenguaje de programación particular. El hardware se refiere a los componentes físicos del sistema que ejecutan las instrucciones del software.

En el primer capítulo, trataremos de presentar la motivación detrás de este proyecto, tanto a nivel personal como científico y técnico. Discutiremos los objetivos específicos que se pretenden alcanzar y la metodología empleada para lograrlos. Además, detallaremos las herramientas y librerías utilizadas en el desarrollo del sistema, así como la estructura del trabajo.

1.1. MOTIVACIÓN

1.1.1. Motivación personal

Desde niño, siempre he tenido curiosidad por construir cosas como coches, bicicletas, edificios, etc. Esta curiosidad me llevó a estudiar una carrera en ingeniería. Una vez que comencé, debido a las asignaturas de automatización, programación e ingeniería eléctrica, me di cuenta de que había una parte no física llamada programación. Sin embargo, tanto las partes físicas como las no físicas no pueden existir una sin la otra. Luego, me inscribí en un curso online de programación en Python después de investigar cuál era el lenguaje que mejor se adapta a mis conocimientos. Primero, hice algunos códigos básicos para ayudarme con mis tareas, como crear hojas de cálculo automáticas en Excel y resolver problemas de ingeniería estructural más rápidamente. Después de algunos programas, supe que estaba listo para hacer scripts más complejos para resolver problemas.

Cuando llegó el momento de elegir el tema de mi TFG, decidí que debía ser en software. Estaba buscando algo relacionado con la ingeniería estructural y de materiales, pero el código utilizado es muy complejo y completo, por lo que mi trabajo no habría resuelto ningún problema que aún no se hubiera resuelto. En ese momento, estaba hablando con el departamento de automatización buscando algún tema también. Hablé con mi tutor sobre este proyecto y era muy desafiante, así que decidí aceptarlo. Un nuevo programa que puede ayudar a aumentar la eficiencia en las áreas de laboratorio.

Mi objetivo personal era aprender a codificar programas de control de ordenadores y conectarlos con otros dispositivos para recibir datos y ejecutar instrucciones.

1.1.2. Motivación profesional

Mi motivación profesional está impulsada por una fuerte aspiración a convertirme en emprendedor y tener un impacto significativo en el campo del software. Me atrae el camino del emprendimiento porque me ofrece la oportunidad de materializar mis ideas innovadoras, crear mis

propias empresas y contribuir al desarrollo de tecnologías revolucionarias. Este proyecto de TFG sobre el diseño e implementación de un sistema modular para la gestión de protocolos en un sistema automatizado de biorreactores se alinea perfectamente con mis objetivos empresariales. Al completar con éxito este proyecto, pretendo adquirir habilidades valiosas en desarrollo de software, automatización y gestión de proyectos, que son esenciales para establecer y dirigir mi propia startup de tecnología en el futuro. Esta experiencia me proporcionará una comprensión profunda de los desafíos de la automatización en laboratorios.

Si bien mi objetivo principal es el emprendimiento, también estoy abierto a oportunidades de trabajo de alto perfil que me permitan hacer contribuciones significativas al campo. Este proyecto de TFG puede servir como un trampolín hacia esas oportunidades, ya que demostrará mi competencia técnica, habilidades para resolver problemas y compromiso con la innovación. Al mostrar mi experiencia en el desarrollo de soluciones de software para la automatización de laboratorios y mi capacidad para coordinar e integrar diferentes dispositivos, puedo posicionarme como un activo valioso para empresas biotecnológicas e instituciones de investigación que buscan individuos con una sólida base en tecnologías de automatización.

1.2. OBJETIVOS

El objetivo principal de este TFG es diseñar e implementar un sistema modular para la gestión de protocolos en una plataforma de biorreactores ChiBio. Esta plataforma actualmente cuenta con un software no modular e ineficiente que presenta dificultades para implementar nuevas estrategias de control realimentado, sensorización y protocolos que requieran la coordinación entre los biorreactores. Este proyecto busca superar estas limitaciones mediante el desarrollo de un nuevo sistema que cumpla con los siguientes requisitos.

El sistema será programado en Python, utilizando librerías estándar para asegurar la accesibilidad y facilidad de mantenimiento. Se diseñará e implementará una estructura modular que permita la rápida definición e integración de nuevos módulos, como controladores y observadores, facilitando así la expansión y adaptación a futuras necesidades.

Otro objetivo clave es la gestión en tiempo real de los biorreactores desde un ordenador de control central. Esto permitirá la supervisión y ajuste inmediato de los experimentos, mejorando la eficiencia y precisión de los procesos. Para lograr esto, se implementarán módulos básicos con estrategias de control realimentado y observadores de las tasas de reacción, garantizando un control adecuado de las variables experimentales críticas.

La gestión de datos es otro componente fundamental del proyecto. Se implementarán herramientas para la visualización y almacenamiento de datos, permitiendo a los usuarios monitorear y analizar los resultados en tiempo real y acceder a registros históricos para evaluar el rendimiento y realizar ajustes basados en datos concretos.

Finalmente, se realizará una validación experimental del funcionamiento de la plataforma. Esta etapa será crucial para asegurar que el software cumple con todos los requisitos establecidos y funciona de manera eficiente en un entorno de laboratorio real. La validación incluirá pruebas

exhaustivas con los biorreactores para verificar la integración de los módulos, la precisión de los datos y la eficacia de los protocolos definidos.

En resumen, este TFG tiene como objetivo desarrollar un sistema software modular y eficiente que mejore significativamente la gestión de protocolos en la plataforma de biorreactores ChiBio, facilitando así la investigación y experimentación en biología sintética.

1.3. METODOLOGÍA

La metodología empleada en este TFG se ha diseñado para garantizar el cumplimiento de los objetivos establecidos de manera eficiente y estructurada. Esta no sólo guiará el desarrollo del sistema software modular, sino que también determinará la estructura del trabajo y la memoria. A continuación, se detallan los pasos propuestos para el cumplimiento de los objetivos, así como la justificación de la elección de Python y las librerías utilizadas.

1.3.1. Python

La elección de Python como lenguaje de programación principal para este proyecto se basa en su versatilidad, facilidad de uso y amplia adopción en la comunidad científica y de ingeniería. Python ofrece una sintaxis clara y concisa, lo que facilita la escritura y mantenimiento del código, así como una extensa colección de librerías que permiten abordar diversas tareas de forma eficiente.

El primer paso consiste en el análisis de requisitos, donde se identifican las necesidades específicas de la plataforma de biorreactores ChiBio y se definen los requisitos funcionales y no funcionales del sistema software. A continuación, se realiza el diseño del sistema, creando una arquitectura modular que permita la fácil integración y expansión de nuevos módulos. Este diseño incluye la definición de interfaces claras entre los módulos para asegurar la cohesión y reducir el acoplamiento.

Una vez definido el diseño, se procede al desarrollo del núcleo del sistema. En esta etapa, se implementa la base del sistema en Python, utilizando principios de programación orientada a objetos. Esto incluye la creación de funciones y métodos para la gestión de biorreactores, protocolos y controladores. Posteriormente, se desarrollan módulos específicos, implementando estrategias de control realimentado y observadores de tasas de reacción. Estos módulos permiten la gestión en tiempo real de los biorreactores y la sincronización de eventos entre ellos.

La integración y pruebas constituyen la siguiente fase, donde se integran todos los módulos desarrollados y se realizan pruebas exhaustivas para asegurar su correcto funcionamiento. Esto incluye pruebas unitarias, de integración y de sistema, así como la validación experimental en un entorno de laboratorio real. También se implementan herramientas para la visualización y almacenamiento de datos, utilizando librerías específicas para el manejo de datos y la generación de gráficos, permitiendo a los usuarios monitorear y analizar los resultados en tiempo real.

1.3.2. Librerías: Django

El uso de librerías específicas es fundamental para cumplir con los objetivos del proyecto de manera eficiente. Las seleccionadas permiten abordar tareas complejas de forma simplificada, asegurando la calidad y robustez del sistema.

El primer paso es su selección, identificando aquellas que mejor se adapten a las necesidades del proyecto. Estas deben ser de código abierto, ampliamente utilizadas y bien documentadas para asegurar su fiabilidad y facilidad de integración.

Django se utiliza como framework principal para la gestión del servidor web y la interfaz de usuario. Este facilita la creación de aplicaciones web robustas y seguras, permitiendo una rápida implementación de funcionalidades y una buena organización del código.

Un framework es una estructura predefinida que proporciona un conjunto de herramientas y componentes reutilizables para facilitar el desarrollo de software, promoviendo la eficiencia y el uso de buenas prácticas de programación.

Entre las ventajas que ofrece Django sobre cualquier otro en programación web en Python tenemos:

- *Arquitectura Modular y Escalable*: sigue el principio de "baterías incluidas", proporcionando una gran cantidad de funcionalidades integradas, como un ORM (Object-Relational Mapping), administración automática, autenticación, y más. Esto permite un desarrollo más rápido y reduce la necesidad de integrar manualmente múltiples componentes. La arquitectura modular facilita la creación de aplicaciones complejas y escalables, permitiendo la integración de nuevas funcionalidades sin afectar negativamente al resto del sistema.

- *Estructura y Convenciones*: impone una estructura y convenciones claras para el desarrollo del proyecto, lo que facilita la colaboración entre múltiples desarrolladores y el mantenimiento del código a largo plazo. Esta estructura bien definida ayuda a mantener el código organizado y coherente, reduciendo la posibilidad de errores y mejorando la eficiencia del desarrollo. Sus convenciones también facilitan la integración de nuevos desarrolladores en el proyecto, ya que pueden entender rápidamente la estructura y las prácticas utilizadas.

- *Seguridad*: incluye por defecto muchas características de seguridad que protegen contra ataques comunes como la inyección SQL, la falsificación de solicitudes entre sitios (CSRF) y la falsificación de scripts entre sitios (XSS). Estas características integradas permiten a los desarrolladores centrarse en la funcionalidad de la aplicación sin preocuparse constantemente por las vulnerabilidades de seguridad. Además, sigue las mejores prácticas de seguridad recomendadas por la OWASP (Open Web Application Security Project), proporcionando una base sólida para el desarrollo de webs seguras.

- *Ecosistema y Comunidad*: cuenta con una gran comunidad de desarrolladores y una amplia documentación, lo que facilita la resolución de problemas y la búsqueda de soluciones. Su

ecosistema incluye una gran cantidad de paquetes y extensiones que pueden ser fácilmente integrados en el proyecto para añadir funcionalidades adicionales. La comunidad activa también asegura que el framework esté constantemente mejorado y actualizado, proporcionando acceso a las últimas tecnologías y prácticas de desarrollo.

La integración de las librerías seleccionadas en el desarrollo del sistema es un paso crítico. Esto incluye la configuración y pruebas de cada una de ellas para verificar su desempeño en el entorno del proyecto. Finalmente, se redacta la documentación específica sobre el uso y configuración de cada una en el sistema, facilitando el mantenimiento y futuras expansiones del software.

En resumen, la metodología propuesta asegura que el desarrollo del sistema modular se realice de manera estructurada y eficiente, utilizando Python y un conjunto de librerías seleccionadas para cumplir con los objetivos del proyecto. Esto garantiza que el software desarrollado sea robusto, escalable y adaptable a futuras necesidades, proporcionando una solución eficiente para la gestión de protocolos en la plataforma de biorreactores ChiBio.

1.4 ESTRUCTURA

La estructura del presente TFG está organizada de manera que el lector pueda seguir de forma clara y lógica el desarrollo del proyecto, desde su planteamiento inicial hasta las conclusiones finales. A lo largo de los diferentes capítulos, se aborda el proceso completo de diseño e implementación del sistema software modular para la gestión de protocolos en la plataforma de biorreactores ChiBio.

El siguiente capítulo, denominado Análisis del Problema, se enfoca en la identificación y análisis de las posibles soluciones al problema planteado. Aquí se describen las limitaciones del software actual y se justifica la necesidad de desarrollar un nuevo sistema. También se presenta la solución propuesta y se detalla el plan de trabajo que se seguirá para llevar a cabo el proyecto.

En el capítulo 3, titulado Diseño de la Solución, se detalla la arquitectura del sistema, describiendo cómo se estructura el software de manera modular. Se discuten las tecnologías utilizadas y se justifica su elección, asegurando que se adecuan al ámbito del trabajo y a los objetivos del TFG.

El capítulo 4, denominado Desarrollo de la Solución, abarca el proceso de implementación del sistema software. Aquí se describen los distintos módulos desarrollados, las estrategias de control realimentado implementadas y la integración de los observadores de las tasas de reacción. También se aborda la gestión en tiempo real de los biorreactores y la coordinación de eventos entre ellos.

Finalmente, el capítulo 5, titulado Conclusiones, resume los hallazgos principales del TFG, reflexiona sobre los logros alcanzados y discute las posibles mejoras futuras. También se sugieren líneas de trabajo adicionales que podrían explorarse para continuar avanzando en el desarrollo del software.

Además de los capítulos mencionados, el documento incluye anexos que proporcionan información adicional relevante para el proyecto. Estos pueden contener detalles técnicos específicos, ejemplos de código, datos experimentales y cualquier otro material que complemente y enriquezca el contenido principal del TFG. Los mismos permiten al lector profundizar en aspectos particulares del proyecto, ofreciendo una visión más completa y detallada del trabajo realizado.

CAPÍTULO 2. ANÁLISIS DEL PROBLEMA

El primer paso en el desarrollo del sistema software modular para la gestión de protocolos en la plataforma de biorreactores ChiBio es la especificación detallada de los requisitos. Este proceso implica identificar y documentar las necesidades específicas de los usuarios y las funcionalidades que debe ofrecer el sistema para satisfacer dichas necesidades. Para ello, se utilizarán técnicas estándar ampliamente aceptadas en el ámbito de la informática.

2.1. DEFINICIÓN DEL PROBLEMA: ESTADO INICIAL

La plataforma de biorreactores ChiBio enfrenta importantes limitaciones debido a la falta de un sistema software eficiente y modular. Esta carencia dificulta la implementación de nuevas estrategias de control realimentado y sensorización, así como la coordinación efectiva entre los cinco biorreactores disponibles. Como resultado, los investigadores y técnicos de laboratorio tienen dificultades para gestionar experimentos de manera eficiente, lo que afecta la reproducibilidad y estandarización de los protocolos en biología sintética. Además, el software actual no facilita la integración de nuevos módulos ni la gestión en tiempo real de los biorreactores, presentando un obstáculo significativo para la optimización experimental y la implementación de estrategias avanzadas como la evolución dirigida.

El software existente, aunque funcional en sus aspectos básicos, carece de la flexibilidad y escalabilidad necesarias para satisfacer las demandas actuales de los usuarios. Desarrollado con un enfoque no modular, presenta problemas significativos en términos de mantenimiento y expansión. La infraestructura actual está basada en un código monolítico, lo que dificulta la incorporación de nuevos elementos y la adaptación a las necesidades cambiantes de los experimentos en biología sintética. Esta situación ha sido ampliamente discutida en reuniones con técnicos y usuarios del sistema, quienes han aportado valiosos insights sobre las limitaciones actuales y los requisitos esenciales para el nuevo desarrollo. Estas discusiones han identificado la necesidad de una solución que no solo mejore la eficiencia operativa, sino que también permita una mayor adaptabilidad y robustez en la gestión de protocolos experimentales.

Para abordar estas limitaciones, se ha propuesto el diseño de una arquitectura modular utilizando Django, detallado en el Capítulo 3. Esta nueva estructura no solo mejora la eficiencia y escalabilidad del sistema, sino que también facilita la incorporación de nuevas estrategias de control y sensorización. Posteriormente, en el Capítulo 4, se describe la implementación detallada de módulos y controladores, optimizando la gestión de protocolos y garantizando una mayor flexibilidad y eficiencia operativa en la plataforma ChiBio.

2.2. ESPECIFICACIÓN DE REQUISITOS Y MODELADO CONCEPTUAL

En este apartado, se detallarán los requisitos necesarios para el desarrollo del sistema software modular para la gestión de protocolos en la plataforma de biorreactores ChiBio. La especificación es un paso crucial en el desarrollo de cualquier sistema de software, ya que define las expectativas y necesidades que el sistema debe cumplir. Para asegurar la claridad y exhaustividad, se utilizarán técnicas estándar y ampliamente aceptadas en el ámbito de la informática.

La identificación de requisitos comienza con una comprensión profunda de las necesidades de los usuarios y las limitaciones del sistema actual. En el contexto de la plataforma de biorreactores ChiBio, los principales usuarios son los investigadores y técnicos de laboratorio que requieren un sistema eficiente y fiable para gestionar sus experimentos. Los requisitos identificados se pueden clasificar en dos categorías principales: funcionales y no funcionales.

Para identificar estos requisitos, se llevaron a cabo reuniones con los principales usuarios del sistema: los investigadores y técnicos de laboratorio. Estas reuniones se centraron en recoger información sobre las necesidades y expectativas específicas para el nuevo sistema. Durante estas reuniones, se discutieron las limitaciones del sistema actual y se identificaron las mejoras necesarias para optimizar la eficiencia y precisión en la gestión de los experimentos. Además, se realizaron análisis de las tareas diarias y se observaron los procesos en acción para entender mejor las necesidades operativas.

2.2.1. Requisitos funcionales

Los requisitos funcionales especifican las funciones que el sistema debe realizar. En primer lugar, el sistema debe estar desarrollado en Python utilizando librerías estándar, lo que facilitará la accesibilidad y el mantenimiento del código. Además, el software debe tener una estructura modular, permitiendo la fácil integración y expansión de nuevos módulos, tales como controladores y observadores. Es crucial que el sistema permita la gestión en tiempo real de los biorreactores desde un ordenador de control central. También se deben desarrollar módulos básicos con estrategias de control realimentado y observadores de las tasas de reacción. Otro aspecto importante es que el software debe permitir la definición y gestión de protocolos que coordinen eventos entre los biorreactores. Finalmente, el sistema debe proporcionar herramientas para la visualización y almacenamiento de datos.

Durante las reuniones con los usuarios, se identificaron varias necesidades específicas que debían ser atendidas por el sistema. Los usuarios, que son principalmente investigadores y técnicos de laboratorio, destacaron la importancia de que el sistema permite la supervisión y ajuste inmediato de los parámetros de los biorreactores. Esto es fundamental para mejorar la eficiencia y precisión de los procesos experimentales de este tipo. Además, se subrayó la necesidad de que el sistema sea modular. Esto significa que debe ser posible integrar fácilmente nuevos módulos de control y observación para adaptarse a futuros requerimientos. La modularidad también facilita la actualización y expansión del sistema sin afectar negativamente a su funcionamiento general.

Otra necesidad clave identificada fue la capacidad de definir y gestionar protocolos que coordinen eventos entre múltiples biorreactores. Los usuarios enfatizaron que esta funcionalidad es crucial para realizar experimentos complejos de manera sincronizada, lo que permite una mayor flexibilidad y precisión en la ejecución de estos. En resumen, los requisitos funcionales del sistema no solo deben centrarse en la gestión en tiempo real y la modularidad, sino también en la capacidad de coordinar eventos y proporcionar herramientas robustas para la visualización y almacenamiento de datos.

2.2.2. Requisitos no funcionales

Los requisitos no funcionales describen los atributos del sistema, como su usabilidad, fiabilidad y rendimiento. El sistema debe ser fácil de usar, con una interfaz intuitiva que permita a los usuarios gestionar los biorreactores sin dificultad. Además, el software debe ser fiable, garantizando la correcta ejecución de los experimentos y la integridad de los datos. La arquitectura modular debe permitir la escalabilidad del sistema, facilitando la adición de nuevos módulos y funcionalidades en el futuro. El sistema también debe ser capaz de gestionar múltiples biorreactores simultáneamente sin comprometer el rendimiento. Por último, el mismo debe implementar medidas de seguridad para proteger los datos y el acceso al sistema.

Durante las reuniones con los usuarios, se destacó la necesidad de que el sistema sea altamente usable. Los investigadores y técnicos de laboratorio requieren una interfaz de usuario que sea intuitiva y fácil de navegar, de manera que puedan gestionar los experimentos sin necesidad de un entrenamiento extenso. La facilidad de uso es esencial para garantizar que el sistema sea adoptado y utilizado de manera efectiva por todos los usuarios.

Además de la usabilidad, la fiabilidad del sistema fue otro punto crítico mencionado por los usuarios. Es fundamental que el software garantice la correcta ejecución de los experimentos y la integridad de los datos generados. Esto incluye la implementación de medidas de seguridad robustas para proteger los datos sensibles. Los usuarios también enfatizaron la importancia de la escalabilidad del sistema. La arquitectura modular debe permitir la fácil adición de nuevos módulos y funcionalidades, asegurando que el sistema pueda evolucionar y adaptarse a futuras necesidades sin comprometer su rendimiento.

En resumen, los requisitos no funcionales se centran en asegurar que el sistema sea fácil de usar, fiable y escalable, con la capacidad de gestionar múltiples biorreactores simultáneamente y proteger los datos de manera efectiva. Estas características son esenciales para proporcionar una herramienta que no solo cumpla con las necesidades actuales de los usuarios, sino que también pueda adaptarse y crecer con ellos en el futuro.

Para enfrentar las diversas soluciones posibles, se realizó una evaluación y selección de tecnologías y metodologías adecuadas, detalladas en el Capítulo 3, específicamente en los apartados 3.1 sobre la arquitectura del sistema y 3.2 sobre la tecnología utilizada. En estos apartados, se analiza cómo cada tecnología contribuye a la solución modular propuesta, asegurando su eficiencia y escalabilidad. Además, en el Capítulo 4, en los apartados 4.1 y 4.2, se describe el proceso detallado de desarrollo, que incluye la configuración del entorno y la creación de la estructura del proyecto, garantizando una implementación coherente y estructurada de los módulos necesarios.

2.3. IDENTIFICACIÓN Y ANÁLISIS DE POSIBLES SOLUCIONES

A partir de los requisitos analizados, se considera que no existe una solución única para resolver el problema planteado. Por lo tanto, se presenta un abanico de soluciones posibles, a estudiar una por una para determinar sus defectos y virtudes. Una de las soluciones propuestas es la mejora del software existente. Esta solución implicaría la refactorización y optimización del anterior,

que no es modular, para mejorar su eficiencia y capacidad de implementación de nuevas estrategias de control y sensorización. Sin embargo, este enfoque podría estar limitado por las restricciones inherentes al diseño original y podría no ofrecer la flexibilidad necesaria para futuras expansiones. La refactorización del código actual requeriría un análisis profundo del existente, identificando y resolviendo los problemas de diseño y arquitectura que limitan su eficiencia y escalabilidad. Además, esta solución demandaría una revisión exhaustiva de las funcionalidades existentes para asegurar que no se introduzcan errores ni se comprometan las operaciones actuales del sistema.

Otra solución es el desarrollo de un nuevo sistema desde cero. Esta opción permite diseñar un software completamente nuevo, concebido desde el principio para ser modular y eficiente, utilizando Python y librerías estándar. Este enfoque garantiza que el sistema sea altamente adaptable y escalable, permitiendo la integración de nuevas funcionalidades de manera más sencilla. Esta solución permite aprovechar las últimas tecnologías y prácticas de desarrollo de software, asegurando que el sistema cumpla con los estándares actuales de calidad y rendimiento. Además, diseñar un nuevo sistema desde cero ofrece la oportunidad de implementar una arquitectura moderna y flexible, que facilite el mantenimiento y una posible expansión futura. No obstante, desarrollar un sistema desde cero requiere un mayor esfuerzo inicial en términos de tiempo y recursos. Además de requerir altos conocimientos en mecánica molecular, lo que queda fuera de nuestro alcance.

La tercera solución es la adopción y adaptación de un sistema software modular existente. Esta opción implica buscar un sistema ya desarrollado que pueda ser adaptado a las necesidades específicas de la plataforma ChiBio. La ventaja de este enfoque es que podría reducir significativamente el tiempo de desarrollo al aprovechar componentes ya probados. Sin embargo, encontrar un sistema que se ajuste perfectamente a las necesidades específicas puede ser desafiante, y la adaptación podría requerir una considerable cantidad de trabajo. Este enfoque también conlleva el riesgo de depender de soluciones de terceros, que podrían no estar perfectamente alineadas con los requisitos del proyecto o que podrían tener limitaciones en cuanto a personalización y escalabilidad. La adaptación de un sistema existente requeriría un análisis detallado de las capacidades y limitaciones existentes, así como la implementación de ajustes y modificaciones para asegurar que cumple con los requisitos específicos de la plataforma ChiBio. Cualquier código que pueda ser utilizado tendría entonces que ser abierto como en la primera solución.

Cada una de estas soluciones se evaluará en términos de viabilidad técnica, coste, tiempo de implementación y capacidad para satisfacer los requisitos definidos. Para ello, se establecerá un criterio de selección basado en estos factores y se aplicará para determinar cuál será la solución que finalmente se desarrolle en este TFG. El análisis comparativo de las soluciones permitirá una elección informada que optimice los recursos disponibles y maximice los beneficios del proyecto. Este proceso de evaluación incluirá pruebas de concepto para validar la viabilidad técnica de cada solución propuesta, así como la consulta con expertos y usuarios finales para recoger feedback y ajustar las propuestas según sea necesario.

Para la mejora del software existente, se ha decidido refactorizar para incorporar nuevas estrategias de control y sensorización, y hacer el sistema más flexible y escalable. Este proceso se detalla en el Capítulo 4, especialmente en los apartados 4.1 y 4.2, donde se describe el proceso de refactorización y las mejoras implementadas en el sistema actual. Estas secciones ofrecen una visión exhaustiva de cómo se optimizó la estructura del código y se integraron nuevas funcionalidades, garantizando una mayor eficiencia y escalabilidad en la plataforma ChiBio.

2.4. SOLUCIÓN PROPUESTA

La solución elegida es la mejora del software existente. Este enfoque implica la refactorización y optimización del no modular actual para mejorar su eficiencia y capacidad de implementación de nuevas estrategias de control y sensorización. Esta decisión se basa en la necesidad de aprovechar la base ya existente, reduciendo el tiempo y los recursos necesarios para desarrollar un sistema completamente nuevo desde cero. Además, permite mantener la continuidad de las funcionalidades actuales mientras se introducen mejoras significativas en la arquitectura y el rendimiento. Está decisión nos aportará las ventajas de la tercera solución propuesta sin la negativa de no tener extensos conocimientos en mecánica molecular.

El proceso de mejora del software existente se dividirá en varias fases. En primer lugar, se realizará un análisis exhaustivo del código actual para identificar los principales problemas de diseño y arquitectura que limitan su eficiencia y escalabilidad. Esta fase incluirá la revisión del código fuente, la documentación y las entrevistas con los desarrolladores originales y los usuarios del sistema. El objetivo es obtener una comprensión clara de las debilidades y áreas de mejora. Durante esta fase, se identificarán los componentes que necesitan ser refactorizados, así como las funcionalidades que deben ser añadidas o mejoradas. Se prestará especial atención a las partes del código que presentan problemas de rendimiento, dificultades de mantenimiento o limitaciones para la integración de nuevas funcionalidades.

Este enfoque soluciona el problema identificado en el apartado 2.1 de varias maneras. Al realizar un análisis exhaustivo del código actual, se abordarán las limitaciones en flexibilidad y escalabilidad, permitiendo identificar áreas críticas que requieren mejoras. La revisión detallada y las entrevistas con los usuarios aseguran que las necesidades y expectativas de los investigadores y técnicos de laboratorio se integren en el nuevo diseño, resolviendo las dificultades actuales en la gestión de experimentos.

Posteriormente, se procederá a la refactorización del código. Esto implica reestructurar el código existente para mejorar su legibilidad, modularidad y eficiencia sin cambiar su comportamiento externo. Durante esta fase, se implementarán principios de diseño orientado a objetos para mejorar la modularidad del sistema, permitiendo una mejor separación de preocupaciones y facilitando la integración de nuevos módulos. Se utilizarán patrones de diseño cuando sea apropiado para resolver problemas comunes y mejorar la estructura del código. La refactorización también incluirá la eliminación de código redundante, la optimización de algoritmos y la mejora de la gestión de recursos. Uno de los cambios más significativos en esta fase será la migración de Flask a Django. Flask es un microframework ligero y flexible, ideal para proyectos pequeños y medianos. Sin embargo, a medida que el proyecto crece en complejidad, Flask puede

volverse difícil de manejar debido a la falta de una estructura predeterminada y la necesidad de integrar manualmente muchas funcionalidades adicionales. Django, por otro lado, es un framework de desarrollo web de alto nivel que promueve el desarrollo rápido y el diseño limpio y pragmático. Este ofrece una estructura predeterminada que facilita la organización del código y la implementación de nuevas funcionalidades de manera modular y escalable. Entre las ventajas sobre Flask se incluyen la arquitectura modular y escalabilidad, seguridad y ecosistema y comunidades, entre otras descritas en el apartado 1.3.2 de la memoria.

Una parte crucial de la refactorización será la implementación de una arquitectura modular. Esta arquitectura permitirá la fácil integración y expansión de nuevos módulos, tales como controladores y observadores. Cada módulo será diseñado como una unidad independiente con interfaces bien definidas, lo que facilitará su desarrollo, prueba y mantenimiento. La modularización del sistema permitirá que las nuevas funcionalidades se añadan sin afectar negativamente al resto del sistema, mejorando la flexibilidad y escalabilidad del software.

La siguiente fase del proyecto implica la implementación de módulos específicos necesarios para el sistema. Esto incluye la creación de controladores que implementen estrategias de control realimentado y observadores que monitoreen las tasas de reacción. Estos módulos son esenciales para la funcionalidad del sistema, ya que permiten la gestión precisa y eficiente de los experimentos en tiempo real. La implementación de estos módulos requerirá la integración de algoritmos de control avanzados y técnicas de sensorización software. Además, se desarrollarán herramientas para la definición y gestión de protocolos experimentales que coordinen eventos entre los biorreactores, asegurando que los experimentos se lleven a cabo de manera sincronizada y eficiente.

La integración de todos los componentes desarrollados es una etapa crucial para asegurar que el sistema funcione de manera cohesiva y eficiente. Se realizarán pruebas exhaustivas, incluyendo pruebas unitarias, de integración y de sistema, para verificar que todas las partes del sistema operan correctamente y se cumplen los requisitos especificados. Estas pruebas permitirán identificar y corregir cualquier problema antes de la implementación final del sistema. Se utilizarán técnicas de pruebas automatizadas para asegurar una cobertura de pruebas adecuada y facilitar la detección de errores regresivos. Durante esta fase, se monitorea el rendimiento del sistema, se recogerán datos y se evaluará su desempeño en situaciones teóricas. Los resultados de estas pruebas serán analizados para asegurar que el sistema cumple con todos los requisitos funcionales y no funcionales definidos en el apartado 2.1 de la memoria.

Además de la refactorización y mejora del código existente, se llevará a cabo una actualización de la documentación técnica del sistema. Esto incluye la creación de documentación detallada sobre la arquitectura del sistema, las interfaces de los módulos y las funcionalidades implementadas. La documentación es crucial para facilitar el mantenimiento futuro del sistema y para proporcionar una guía clara para los desarrolladores que trabajen en el sistema en el futuro. La documentación también abordará las mejores prácticas para la implementación de nuevos módulos y la integración con el sistema existente.

Este enfoque integral garantiza que el sistema no solo resolverá los problemas actuales de eficiencia, flexibilidad y escalabilidad, sino que también estará preparado para futuras expansiones y

mejoras. La incorporación de las observaciones y necesidades recogidas en las reuniones con los usuarios asegura que el sistema final cumpla con las expectativas y requisitos específicos de los investigadores y técnicos de laboratorio, mejorando significativamente la gestión de protocolos en biología sintética.

2.5. PLAN DE TRABAJO

El plan de trabajo para la mejora del software existente se define mediante técnicas de planificación y estimación de esfuerzo, indicando las fases en las que se dividirá el desarrollo del TFG. Este plan asegura que todas las fases del proyecto se desarrollen de manera estructurada y eficiente, permitiendo alcanzar los objetivos establecidos y asegurar el éxito del mismo.

La primera fase es el análisis y diseño. Durante esta fase, se realizará un análisis exhaustivo del código actual para identificar los principales problemas de diseño y arquitectura que limitan su eficiencia y escalabilidad. Este paso es crucial para establecer una base sólida para el desarrollo del software, asegurando que todas las necesidades y restricciones se entienden completamente antes de comenzar la refactorización. Durante esta fase, se llevarán a cabo reuniones con los expertos para recoger sus requisitos y expectativas, así como para discutir y revisar el diseño propuesto. Se elaborarán documentos detallados de especificaciones que describen la arquitectura y el comportamiento del sistema. Este análisis incluirá la revisión del código fuente existente en Flask, la evaluación de sus limitaciones y la planificación de la migración a Django. Además, se evaluarán las funcionalidades actuales del sistema y se definirán las mejoras necesarias para cumplir con los requisitos identificados.

Este análisis detallado permitirá identificar las áreas donde el sistema actual falla en términos de flexibilidad y escalabilidad. Las reuniones con los usuarios proporcionarán información crucial sobre los problemas actuales y las necesidades específicas, asegurando que las soluciones diseñadas aborden directamente estas cuestiones.

La siguiente fase es la refactorización del código. En esta fase, se reestructurará el código existente para mejorar su legibilidad, modularidad y eficiencia sin cambiar su comportamiento externo. La refactorización incluirá la implementación de principios de diseño orientado a objetos, la eliminación de código redundante, la optimización de algoritmos y la mejora de la gestión de recursos. Durante esta fase, se utilizarán herramientas de desarrollo ágil y revisiones de código para asegurar que el sistema sea robusto y mantenible. La migración de Flask a Django será un componente clave de esta fase. Este último proporciona una estructura predeterminada y muchas funcionalidades integradas que facilitan el desarrollo y la escalabilidad del sistema. La migración incluirá su configuración, la reorganización del proyecto para seguir las convenciones, y la reimplementación de las funcionalidades actuales utilizando las características avanzadas. Además, se implementarán nuevas funcionalidades y mejoras identificadas durante la fase de análisis, asegurando que el sistema sea capaz de soportar las nuevas estrategias de control y sensorización.

La refactorización del código y la migración a Django abordarán los problemas de modularidad y escalabilidad, permitiendo una integración más sencilla de nuevos módulos y funcionalidades. Esto resolverá las limitaciones actuales y mejorará la eficiencia del sistema.

La fase siguiente es la implementación de nuevos módulos específicos, como los controladores y observadores. Estos módulos son esenciales para la funcionalidad del sistema, permitiendo la gestión en tiempo real de los biorreactores y la implementación de estrategias de control realimentado. Durante esta fase, se desarrollarán e integrarán algoritmos de control avanzados y técnicas de sensorización software. Los módulos de control se encargarán de ajustar los parámetros operativos de los biorreactores en respuesta a las condiciones observadas, mientras que los observadores monitorizarán variables clave como las tasas de reacción y la concentración de nutrientes. Se prestará especial atención a la creación de interfaces claras y bien definidas para cada módulo, facilitando su integración y reutilización en futuras expansiones del sistema. Esta fase también incluirá la creación de protocolos experimentales detallados que describen cómo se deben coordinar los eventos entre los biorreactores, asegurando que los experimentos se lleven a cabo de manera sincronizada y eficiente.

La implementación de estos módulos permitirá una gestión más precisa y eficiente de los experimentos, abordando las necesidades de supervisión y ajuste en tiempo real destacadas por los usuarios. La capacidad de definir y gestionar protocolos que coordinen eventos entre biorreactores resolverá las dificultades actuales en la ejecución de experimentos complejos.

La fase de integración y pruebas, incluye la integración de todos los componentes desarrollados y la realización de pruebas exhaustivas para asegurar su correcto funcionamiento. Se realizarán pruebas unitarias para verificar la correcta implementación de cada módulo individualmente, y pruebas de integración para asegurar que los módulos interactúan correctamente entre sí. Además, se llevarán a cabo pruebas de sistema para evaluar el rendimiento y la estabilidad del sistema completo bajo condiciones operativas. Durante esta fase, se monitorea el rendimiento del sistema, se recogerán datos y se evaluará su desempeño en situaciones teóricas. Los resultados de estas pruebas serán analizados para asegurar que el sistema cumple con todos los requisitos funcionales y no funcionales definidos en la etapa de especificación. Se utilizarán herramientas de pruebas automatizadas para asegurar una cobertura de pruebas adecuada y facilitar la detección de errores regresivos.

Las pruebas exhaustivas garantizarán que el sistema final sea robusto y fiable, cumpliendo con los requisitos funcionales y no funcionales identificados, y asegurando la integridad de los datos y la correcta ejecución de los experimentos.

Finalmente, la documentación y presentación del TFG. Esta etapa incluye la redacción de la documentación técnica y la memoria, detallando el proceso de desarrollo, los resultados obtenidos y las conclusiones. La documentación técnica incluirá manuales de usuario y guías de instalación, así como documentación del código para facilitar el mantenimiento y la futura expansión del sistema. La documentación también abordará las mejores prácticas para la implementación de nuevos módulos y la integración con el sistema existente. Además, se realizarán ensayos de la presentación para asegurar que todos los puntos clave del proyecto se comunicaran de manera efectiva, y se prepararán respuestas a posibles preguntas del tribunal evaluador.

Este plan de trabajo asegura que todas las fases del proyecto se desarrollen de manera estructurada y eficiente, permitiendo alcanzar los objetivos establecidos y asegurar el éxito del TFG.

Además, se incluirán anexos que proporcionarán información adicional relevante, como detalles técnicos específicos, ejemplos de código, datos experimentales y cualquier otro material que complemente y enriquezca el contenido principal. Estos anexos permitirán al lector profundizar en aspectos particulares del proyecto, ofreciendo una visión más completa y detallada del trabajo realizado. Este enfoque integral y detallado asegura que el desarrollo del sistema software modular para la plataforma de biorreactores ChiBio se realice de manera eficiente, cumpliendo con todos los requisitos y expectativas de los usuarios. La planificación meticulosa y la ejecución cuidadosa de cada fase del proyecto garantizan que el sistema final sea robusto, escalable y capaz de mejorar significativamente la gestión de protocolos en biología sintética. La validación experimental y la documentación exhaustiva proporcionan una base sólida para futuras mejoras y expansiones del sistema, asegurando su relevancia y utilidad a largo plazo.

CAPÍTULO 3. DISEÑO DE LA SOLUCIÓN

Este capítulo se centra en el diseño de la solución propuesta para el sistema de gestión de protocolos en la plataforma de biorreactores ChiBio. A través de un análisis detallado y sistemático, se han definido los requisitos del sistema y se ha desarrollado un modelo conceptual para su implementación. La arquitectura del sistema se ha dividido en componentes clave, cada uno con su rol específico, y se han seleccionado las tecnologías más adecuadas para llevar a cabo el proyecto. El objetivo de este capítulo es proporcionar una visión integral del diseño del sistema, asegurando que cada parte de la solución está bien fundamentada y justificada.

3.1. ARQUITECTURA DEL SISTEMA

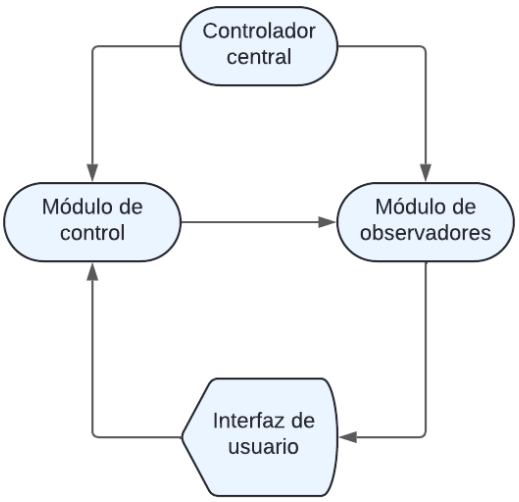
La arquitectura del sistema se ha diseñado con un enfoque modular para facilitar la gestión y expansión de la funcionalidad. El sistema se basa en la arquitectura de Django, un framework de alto nivel de Python que promueve el desarrollo rápido y el diseño limpio y pragmático. El proyecto se organiza en varios componentes esenciales:

El Controlador Central es el núcleo del sistema, encargado de coordinar la comunicación entre los biorreactores y el ordenador de control. Este módulo es esencial para asegurar que los protocolos se ejecuten de manera eficiente, gestionando la sincronización y la transmisión de datos en tiempo real entre los diferentes componentes del sistema. Utiliza canales de comunicación para garantizar una respuesta rápida a las condiciones cambiantes durante los experimentos.

Los Módulos de Controladores implementan diversas estrategias de control realimentado. Estos módulos permiten ajustar las condiciones de operación de los biorreactores basándose en datos en tiempo real, optimizando los procesos biológicos. Cada módulo puede ser desarrollado de forma independiente, lo que facilita la incorporación de nuevas estrategias sin necesidad de modificar el sistema central.

Los Módulos de Observadores funcionan como sensores virtuales, monitorizando parámetros críticos como las tasas de reacción. Estos módulos analizan los datos en tiempo real y proporcionan información crucial para la toma de decisiones, asegurando que los experimentos se desarrollen conforme a los protocolos establecidos. Al igual que los módulos de controladores, los observadores pueden ser desarrollados y ajustados de manera modular.

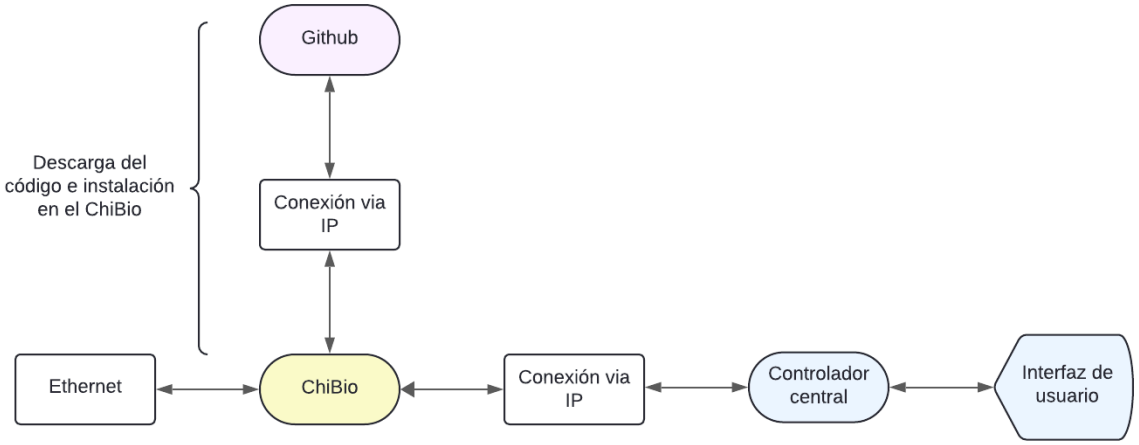
La Interfaz de Usuario es el componente que permite a los investigadores definir y gestionar los protocolos de manera intuitiva. Esta interfaz proporciona herramientas para configurar los experimentos, monitorear su progreso y visualizar los resultados. Está diseñada para ser accesible y fácil de usar, minimizando la curva de aprendizaje y maximizando la eficiencia operativa.



3.1.a Diagrama de bloques: Módulos de la arquitectura

En conjunto, estos componentes interactúan de manera coordinada para proporcionar un sistema robusto y flexible, capaz de adaptarse a una amplia variedad de protocolos y condiciones experimentales. La modularidad de la arquitectura asegura que el sistema puede evolucionar con el tiempo, incorporando nuevas tecnologías y metodologías conforme se desarrollen.

Para la instalación y conexión del sistema, se ha diseñado un proceso que garantiza una integración sin problemas y una comunicación eficiente entre los componentes. El código fuente del sistema se descarga e instala en el ChiBio mediante una conexión SSH desde GitHub. Una vez instalado, el ChiBio se conecta a través de Ethernet al ordenador de control central, que gestiona los protocolos y la interfaz de usuario. La conexión IP permite la transmisión de datos en tiempo real entre el ChiBio, el controlador central y la interfaz de usuario, asegurando una operación coordinada y eficiente del sistema.



3.1.b Diagrama de bloques: Arquitectura de conexión

3.2. TECNOLOGÍA UTILIZADA

La elección de tecnologías para este proyecto ha sido un proceso deliberado, centrado en la eficacia, facilidad de uso y robustez. El lenguaje de programación principal es Python, debido a su simplicidad y extensas capacidades de librerías que facilitan tanto el desarrollo como la implementación de funcionalidades avanzadas. Python se destaca por su sintaxis clara y su amplio soporte comunitario, lo que lo hace especialmente adecuado para aplicaciones científicas y de investigación. La comunidad activa de Python ofrece muchos recursos, documentación y soporte, lo que acelera la resolución de problemas y la implementación de soluciones innovadoras. Además, la vasta colección de bibliotecas y paquetes disponibles en Python permite a los desarrolladores integrar fácilmente funcionalidades complejas sin tener que reinventar la rueda, asegurando que el desarrollo sea eficiente y mantenible.

El framework Django es el núcleo del desarrollo del sistema. Ofrece una estructura organizada y componentes reutilizables, lo que permite un desarrollo rápido y un mantenimiento sencillo. Este sigue el principio de "No te repitas" (Don't Repeat Yourself, DRY), lo que minimiza la redundancia y facilita la gestión del código. Su arquitectura orientada a objetos y su manejo eficiente de bases de datos lo hacen ideal para la implementación de sistemas modulares como el requerido en el proyecto. Django proporciona herramientas integradas para la creación de formularios, validación de datos y administración, lo que reduce significativamente el tiempo de desarrollo y asegura que las aplicaciones sean seguras y escalables.

Para la base de datos, se optó inicialmente por SQLite durante la fase de desarrollo debido a su simplicidad y facilidad de configuración. Esta base de datos es liviana y de código abierto que es perfecta para la fase de desarrollo debido a su portabilidad y cero configuración. Permite a los desarrolladores comenzar rápidamente sin preocuparse por la administración de la base de datos. Sin embargo, para la fase de producción, se migrará a PostgreSQL, una base de datos potente y escalable que ofrece capacidades avanzadas de manejo de datos y es conocida por su fiabilidad y rendimiento. PostgreSQL soporta operaciones complejas y consultas avanzadas, lo que es esencial para manejar grandes volúmenes de datos y asegurar la integridad y consistencia de los datos en un entorno de producción. Además, PostgreSQL ofrece características como la replicación y el clustering, que son cruciales para aplicaciones que requieren alta disponibilidad y rendimiento.

El entorno de desarrollo se gestiona mediante *'virtualenv'*, que permite crear entornos aislados y manejar dependencias de manera ordenada. Virtualenv asegura que cada proyecto tenga su propio entorno de dependencias independiente, evitando conflictos entre diferentes versiones de librerías y paquetes utilizados en distintos proyectos. Esto asegura que el proyecto puede desarrollarse de manera consistente en diferentes máquinas y facilita la colaboración entre varios desarrolladores. Al utilizar virtualenv, los desarrolladores podemos replicar exactamente el entorno de desarrollo en cualquier máquina, lo que simplifica la configuración del entorno y asegura que el código funcione de manera idéntica en todos los entornos de desarrollo y producción. La distribución dentro del entorno virtual sigue un orden de carpetas. Las carpetas 'bin' e 'include', importan el lenguaje de programación python, para ejecutar todos los códigos. En la carpeta 'lib' se incluyen todas las librerías instaladas en el entorno virtual.

Además, el proyecto utiliza consultas AJAX para llamar asíncronamente a funciones, mejorando la interactividad y la eficiencia del sistema. AJAX (Asynchronous JavaScript and XML) permite a la aplicación web enviar y recibir datos del servidor de manera asíncrona sin recargar la página completa. Esto mejora significativamente la experiencia del usuario al permitir interacciones más rápidas y fluidas. Por ejemplo, los datos pueden actualizarse dinámicamente en la interfaz de usuario en respuesta a eventos específicos, como cambios en los parámetros del experimento, sin necesidad de una recarga completa de la página. Esta capacidad de actualización en tiempo real es crucial para la gestión eficiente de los protocolos de biorreactores, donde los cambios y ajustes rápidos son necesarios para optimizar los resultados experimentales.

Adicionalmente, para la gestión se utiliza Docker, que permite contenerizar la aplicación y sus dependencias, asegurando que se ejecute de manera consistente en cualquier entorno. Docker utiliza volúmenes para almacenar datos de manera persistente, lo que es crucial para mantener la integridad de los datos entre reinicios y actualizaciones. El uso de volúmenes de Docker permite una gestión eficiente de los datos, facilitando el desarrollo y el despliegue continuo. Además, durante la fase de migración a PostgreSQL, Docker se utilizará para levantar la base de datos, asegurando que todas las dependencias y configuraciones necesarias estén encapsuladas en un contenedor, lo que simplifica el proceso de configuración y garantiza que la base de datos esté lista para su uso en un entorno de producción.

En resumen, la combinación de estas tecnologías crea una base sólida y flexible para el desarrollo del sistema de gestión de protocolos en la plataforma de biorreactores ChiBio. Python y Django proporcionan un entorno de desarrollo eficiente y organizado, mientras que PostgreSQL asegura una gestión robusta de datos. Virtualenv facilita la gestión de dependencias y la replicabilidad del entorno, y las consultas AJAX mejoran la interactividad y la eficiencia del sistema. El uso de Docker añade una capa adicional de consistencia y facilidad de gestión, asegurando tanto su funcionalidad actual como su capacidad de evolución futura.

CAPÍTULO 4. DESARROLLO DE LA SOLUCIÓN

En este capítulo se describe el proceso de desarrollo de la solución para la gestión de protocolos en la plataforma de biorreactores ChiBio. Se abordarán las implementaciones más relevantes, complejas y críticas del sistema, proporcionando una comprensión detallada de los aspectos técnicos esenciales. A lo largo del desarrollo, se han seguido principios de diseño modular y buenas prácticas de programación para asegurar la eficiencia, mantenibilidad y escalabilidad del sistema.

4.1. DESARROLLO DE LA SOLUCIÓN

Para el desarrollo de la solución, se siguió un enfoque sistemático y modular, utilizando el framework Django por su robustez y flexibilidad en la creación de aplicaciones web. A continuación, se describe el proceso de creación de la estructura del proyecto:

4.1.1. Configuración del entorno

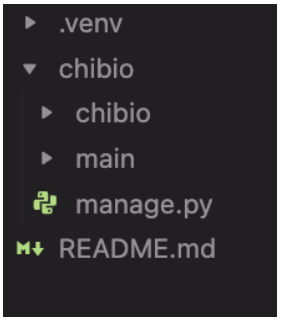
El primer paso, fue configurar el entorno de trabajo. Para ello, debemos crear primero el entorno de python, dónde instalaremos todos las librerías necesarias mencionadas en el apartado 1.3.2. Seguido, crearemos la aplicación con el framework Django, siguiendo los pasos de la documentación de dicha librería.

```
# —--Creación del entorno virtual—-------------------------
python3 -m venv .venv
source .venv/bin/activate
# —-----------------------------------------------------------
# —-Instalando la librería Django y las demás dependencias—-
pip install django
pip install -r requirements.txt
# —-----------------------------------------------------------
# —--Creando la aplicación Django—-------------------------
django-admin startproject chibio
cd chibio
django-admin startapp main
# —-----------------------------------------------------------
```

4.1.1.a Bloque de código para configurar el entorno de trabajo.

Se utiliza un archivo ‘requirements.txt’ en el que podemos encontrar todas las librerías que queremos instalar. De esta manera, evitamos tener que introducir todas ellas por consola.

Una vez ejecutadas las líneas de código, observamos que ya se han creado las carpetas y archivos básicos para el funcionamiento de una aplicación creada con Django en Python.



4.1.1.b Estructura básica de las carpetas del proyecto.

4.1.2. Configuración de la estructura del proyecto

Para mantener una estructura organizada y modular, se crean una serie de directorios para tener, según su funcionalidad, los archivos de código correspondientes, tanto en Python, como HTML, CSS y JavaScript. Todas las carpetas y archivos se crean desde la herramienta Visual Studio Code, permitiendo visualizar una estructura completa y editar todo tipo de archivos de código. En el siguiente apartado (4.2) se detalla el esquema de las carpetas y archivos, además de su contenido.

4.1.3. Configuración de la contenerización con Docker

Para facilitar un entorno consistente y facilitar el despliegue, se utiliza docker. En primer lugar, se debe crear un DockerFile y seguido un archivo '.yaml' llamado 'docker-compose.yaml', dónde se ejecuta la gestión de servicios. Para configurarlo, se ejecutan los siguientes pasos. En el apartado 4.2 se detalla su uso.

```
FROM python:3.x
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "chibio.wsgi:application"]
```

4.1.3.a Archivo DockerFile

```yaml
version: '3.8'
services:
  web:
    build: .
    command: gunicorn chibio.wsgi:application --bind 0.0.0.0:8000
    volumes:
      - .:/app
    ports:
      - "8000:8000"
    env_file:
      - .venv
    depends_on:
      - db
  db:
    image: postgres:13
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      POSTGRES_DB: chibio
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    env_file:
      - .env
```

4.1.3.b Archivo docker-compose.yaml

En el primer archivo se configura el entorno de forma similar al paso 4.1.1. Docker descarga un entorno de python para poder correr la aplicación. Luego instala las librerías y establece los comandos necesarios para poner en producción la aplicación. Todo esto estableciendo el directorio de trabajo '/app' y copiando el actual dentro de la contenerización.

El segundo archivo es para establecer los servicios. En este caso, tenemos dos: 'app' y 'db'. El primero se construye en base al DockerFile, con el comando 'build: .', por lo tanto se trata de un "constructor" y establece que el servicio de la aplicación debe salir por el puerto 8000 de nuestra red local. Es importante elegir bien el puerto, ya que puede sobreescribir otros servicios que estén en funcionamiento y provocar errores. El puerto 8000 es por defecto. Se marca el entorno virtual de python y se establece, con el comando 'depends_on' que no se ejecute el servicio hasta que el otro servicio 'db' no se haya creado correctamente. De esta forma nos aseguramos que la aplicación no funcione sin la base de datos correctamente inicializada. Después tenemos el servicio 'db', el cuál crea la base de datos. Usamos postgres como hemos explicado en apartados anteriores y se declaran el nombre de la base, el usuario y la contraseña.

4.1.4. Desarrollo

Una vez configurada toda la base del proyecto, se inicia la migración de la aplicación de Flask a Django. Este proceso es esencial para aprovechar las características avanzadas y la escalabilidad que ofrece Django. La migración se lleva a cabo en varias etapas para asegurar una transición correcta.

En primer lugar, nos enfocamos en los archivos estáticos, que incluyen HTML, CSS y JavaScript. Estos archivos son fundamentales para el correcto funcionamiento y la presentación de la aplicación. Los archivos estáticos principales en este proyecto son 'index_new.html' y 'main.css', que se encargan de la estructura de la página y del diseño, respectivamente. La migración de estos archivos asegura que la apariencia y la usabilidad de la aplicación se mantengan coherentes con la versión original en Flask. Uno de los objetivos clave de estas actualizaciones es mejorar la experiencia del usuario. Esto implica no solo mantener la funcionalidad existente, sino también optimizarla para hacerla más intuitiva y accesible. Se pone un énfasis particular en crear un diseño responsive, lo que permitirá a los usuarios interactuar con la aplicación de manera efectiva sin importar el tamaño de la pantalla o el dispositivo que estén utilizando. Además, se utilizan estilos de Bootstrap, una fuente altamente utilizada en el diseño web para mejorar la apariencia de las páginas.

```
// DEVICES
$(function() { // ScanDevices
  $('#ScanDevices').click(function(){
    var targetURL = '/scanDevices/all';
     $.ajax({
         type:'GET',
         url: targetURL,
         success:function(response){
         getSysData();
        }
   });
   })
});

$(function() { // Devices0
  $('#Device0').click(function(){
    var targetURL = '/changeDevice/M0';
     $.ajax({
         type:'GET',
         url: targetURL,
         success:function(response){
         getSysData();
        }
   });
   })
});
```

4.1.4.a Configuración de consultas AJAX - 'link.js'

Después de haber migrado los archivos estáticos, nos centramos en la migración de todas las funcionalidades de la aplicación, que están escritas en Python. Este paso es crucial ya que implica trasladar las rutas, vistas y modelos de Flask a Django. Nuestro objetivo es asegurarnos de que todas las operaciones y procesos se realicen de manera eficiente y sin errores, manteniendo la integridad y el rendimiento de la aplicación. La migración de las rutas es uno de los primeros pasos en esta fase. En Flask, las rutas se definen utilizando decoradores directamente en las funciones de vista. En Django, sin embargo, las rutas se configuran en un archivo de 'urls.py', lo que permite una mayor

organización y separación de las preocupaciones. Este cambio requiere una cuidadosa reescritura de las rutas para asegurarse de que todas las URL de la aplicación continúen funcionando como se espera.

```
urlpatterns = [
    path('', views.index, name='index'),
    path('template', views.template, name='template'),
    path('getSysdata/', views.getSysdata, name='getSysdata'),
    # DEVICES FUNCTIONS
    path('scanDevices/<str:which>/', views.scanDevices_view, name='scanDevices'),
    path('changeDevice/<str:wich>/', views.changeDevice_view, name='changeDevice'),
    path('CharacteriseDevice/<str:M>/<str:Program>', views.CharacteriseDevice_view, name='CharacteriseDevice'),
    # ------------------------------------------------------------------------------------------------------
```

4.1.4.b Configuración de rutas URL - 'urls.py'

Las vistas en Flask suelen estar definidas en funciones, mientras que en Django es común utilizar clases basadas en vistas, lo que proporciona una mayor modularidad y reutilización de código. Estas se encuentran en el archivo 'views.py'. Migrar las vistas implica convertir las funciones de Flask en funciones de Django, adaptando la lógica para que funcione en el nuevo framework. Este proceso también permite aprovechar características adicionales de Django, que facilitan la implementación de funcionalidades comunes. Durante todo el proceso de migración, se realizan pruebas exhaustivas para garantizar que cada componente funcione correctamente en el nuevo entorno de Django. Estas pruebas incluyen pruebas unitarias para verificar que las funciones individuales se comporten como se espera, pruebas de integración para asegurar que los diferentes componentes del sistema interactúen correctamente entre sí, y pruebas de usuario para evaluar la experiencia desde la perspectiva del usuario final. Identificar y resolver problemas en esta etapa es fundamental para asegurar una transición suave. Además de las pruebas, se implementa un riguroso control de calidad para revisar el código migrado y asegurar que sigue las mejores prácticas de desarrollo en Django. Esto incluye revisiones del código para reestructurar a una versión más actualizada siguiendo una metodología de buenas prácticas (best practices) de Python. Este paso lo estudiaremos más en detalle en el apartado 5.2.

```
def index(request):
    #Function responsible for sending appropriate device's data to user interface.
    outputdata=sysData[sysItems['UIDevice']]
    for M in ['M0','M1','M2','M3','M4','M5','M6','M7']:
            if sysData[M]['present']==1:
                outputdata['presentDevices'][M]=1
            else:
                outputdata['presentDevices'][M]=0

    return render(request, 'main/index_new.html', outputdata)
```

4.1.4.c Vista principal - 'views.py'

```python
# EXPERIMENT FUNCTIONS
# These functions are responsible for setting up and running experiments.
# ------------------------------------------------------------------------------------------------------------------------
def ExperimentReset_view(request):
    #Resets parameters/values of a given experiment.
    try:
        ExperimentReset()
    except:
        return JsonResponse({'status': 'error'}, status=500)

    return JsonResponse({'status': 'success'}, status=204)

def ExperimentStartStop_view(request, M,value):
    #Stops or starts an experiment.
    try:
        ExperimentStartStop(M,value)
    except:
        return JsonResponse({'status': 'error'}, status=500)

    return JsonResponse({'status': 'success'}, status=204)
# ------------------------------------------------------------------------------------------------------------------------

```

4.1.4.d Ejemplo funcionalidad y manejo de errores - 'views.py'

El objetivo final de este proceso de migración no es solo replicar la funcionalidad existente, sino también mejorarla. La estructura más robusta y las características avanzadas de Django nos permiten optimizar el rendimiento de la aplicación, mejorar la escalabilidad y proporcionar una base más sólida para el desarrollo futuro. En última instancia, esto resulta en una aplicación más robusta y eficiente, que ofrece una mejor experiencia al usuario.

En resumen, la migración de la aplicación de Flask a Django se realiza de manera meticulosa y sistemática. Nos enfocamos en mantener y mejorar la experiencia del usuario, asegurando al mismo tiempo que todas las funcionalidades se trasladen de manera efectiva y eficiente. Este proceso incluye la reescritura cuidadosa de rutas, la adaptación de vistas y la redefinición de modelos, respaldado por un exhaustivo proceso de pruebas y control de calidad.

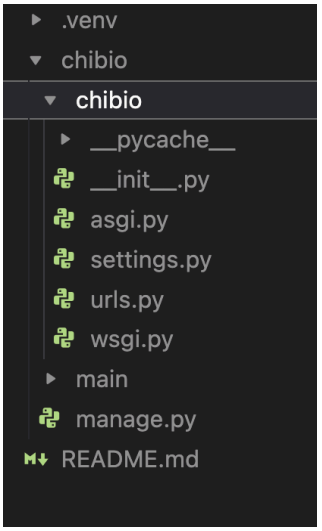
4.2. ESTRUCTURA DEL PROYECTO

El proyecto está organizado de manera que se facilite tanto el desarrollo como el mantenimiento del sistema. Para lograr esto, se ha adoptado una estructura de proyecto clara y modular dentro del framework Django. En el directorio raíz del proyecto, encontramos la configuración general del entorno y los módulos compartidos, los cuales son esenciales para el correcto funcionamiento de la aplicación en su conjunto. Además, encontraremos el Virtualenv, dónde se encuentra todo el entorno de librerías de Python. Esta configuración inicial incluye los archivos necesarios para la gestión de dependencias y la configuración del servidor.

4.2.a Estructura entorno virtual

La aplicación principal se encuentra dentro del directorio *chibio/*. Este directorio alberga la lógica central del sistema, incluyendo los modelos, vistas y controladores que gestionan la interacción entre los usuarios y los biorreactores. Dentro de *chibio/*, podremos observar dos carpetas y un archivo Python. La carpeta *chibio/* contiene toda la configuración del framework Django. Desde esta ruta, se pueden realizar todas las modificaciones necesarias de configuración. En la carpeta *main/*, encontramos la aplicación principal. En ella, se pueden observar diversos archivos que detallaremos más adelante. Por último, el archivo manage.py permite ejecutar comandos administrativos, como la migración de la base de datos, la ejecución del servidor de desarrollo y la creación de nuevas aplicaciones dentro del proyecto. Esta configuración puede visualizarse en la imagen 4.1.1.b.

Dentro de la carpeta *chibio/*, se encuentra la configuración del proyecto Django, que incluye archivos como 'settings.py', 'urls.py', y 'wsgi.py'. El archivo 'settings.py' contiene todas las configuraciones globales de la aplicación, como la configuración de la base de datos, las aplicaciones instaladas, los middleware, y otras configuraciones importantes. El archivo 'urls.py' define las rutas URL para la aplicación, permitiendo que las solicitudes sean dirigidas a las vistas correspondientes. El archivo 'wsgi.py' se utiliza para desplegar la aplicación en servidores de producción, facilitando la interacción entre el servidor web y la misma.



4.2.b Estructura 'chibio'

Dentro de la carpeta *main/*, la estructura se organiza en varios subdirectorios y archivos que manejan diferentes aspectos de la funcionalidad de la aplicación. El subdirectorio *config/* incluye configuraciones adicionales, como las simulaciones de *'GPIO'*, *'I2C'* y *'SMBus'* en el subdirectorio *arduino/*. Estos archivos ('mockgpio.py', 'mocki2c.py', 'mocksmbus.py') simulan el comportamiento del hardware real, permitiendo pruebas y desarrollo sin necesidad de acceso físico a los dispositivos. Una vez lanzada la aplicación a producción, se sustituirán por las librerías adecuadas de python para controlar dispositivos, como 'Adafruit_GPIO.I2C' y 'Adafruit_BBIO.GPIO'.

Además, los archivos Python en *config/* contienen la lógica central de la aplicación. El archivo 'controls.py' maneja la lógica de control para los biorreactores. En él podemos observar funciones de medición como '*MeasureTemp()*' o '*GetLight()*' *(ejemplo 4.2.c*), las cuales se usan posteriormente para ajustar los parámetros del sistema mediante control. El archivo 'custom.py' alberga todas las funciones relacionadas con la generación de lógica personalizada para la creación de programas. Está diseñado para que la personalización del código sea sencilla y accesible, permitiendo a los usuarios ajustar rápidamente las funcionalidades según sus necesidades específicas. Esta capacidad de personalización se ha mejorado significativamente con la migración a Django, facilitando el acceso y modificación de estas funciones. El archivo 'export.py' se encarga de las funciones que generan un archivo exportable en formato '.csv'. Esta funcionalidad es crucial para la documentación y análisis de los datos experimentales, permitiendo a los usuarios exportar resultados y procesarlos en otras herramientas de análisis. El archivo 'init.py' se encarga de todas las funciones de inicialización del sistema. Proporciona una lógica que permite inicializar todo el programa con una sola llamada, así como reiniciarlo o apagarlo. También incluye lógica específica para la inicialización de experimentos individuales, asegurando que cada experimento pueda comenzar con una configuración limpia y predefinida. El archivo 'run.py' contiene la función '*runExperiment()*', que es responsable de ejecutar los experimentos con un parámetro de entrada que es el multiplexor. Esta función está aislada en un archivo específico debido a la posible necesidad de configuraciones adicionales y su importancia crítica en el proceso experimental. El archivo 'setup.py' es un archivo de configuración que establece las bases para las funciones de los demás archivos. Entre sus funcionalidades se incluyen el escaneo

de aparatos, cambio de configuraciones y borrado del terminal, todas esenciales para preparar el entorno de ejecución y asegurar que todos los componentes están correctamente configurados antes de iniciar los experimentos. El archivo 'sys.py' es el archivo de configuración de los parámetros de entrada. Sus funciones se encargan de cargar los datos en la aplicación desde archivos JSON (JavaScript Object Notation), como 'sysdata.json', 'sysdevices.json' y 'syitems.json'. Este enfoque permite realizar modificaciones en los parámetros de configuración sin necesidad de alterar el código fuente, facilitando la flexibilidad y adaptabilidad del sistema a diferentes condiciones experimentales. Por último, el archivo 'watchdogs.py' se encarga de monitorizar el estado actual del hardware, mostrando datos relevantes a través del terminal, como los estados de *'GPIO'*, lo que es esencial para la detección temprana de fallos y la garantía de un funcionamiento continuo y seguro del sistema.

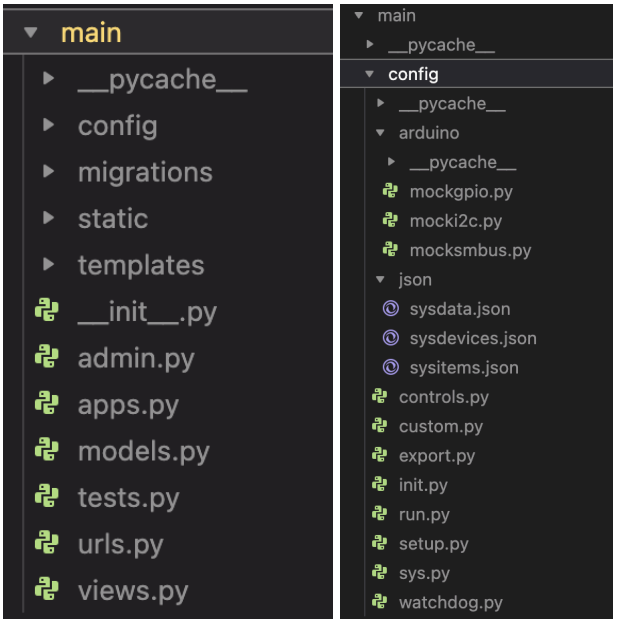
```python
def GetLight(M,wavelengths,Gain,ISteps):
    #Runs spectrometer measurement and puts data into appropriate structure.
    global sysData
    M=str(M)
    channels=['nm410','nm440','nm470','nm510','nm550','nm583','nm620', 'nm670','CLEAR','NIR','DARK','ExtGPIO', 'ExtINT' , 'FLICKER']
    for channel in channels:
        sysData[M]['AS7341']['channels'][channel]=0 #First we set all measurement ADC indexes to zero.
    index=1;
    for wavelength in wavelengths:
        if wavelength != "OFF":
            sysData[M]['AS7341']['channels'][wavelength]=index #Now assign ADCs to each of the channel where needed.
        index=index+1

    success=0
    while success<2:
        try:
            AS7341Read(M,Gain,ISteps,success)
            success=2
        except:
            print(str(datetime.now()) + 'AS7341 measurement failed on ' + str(M))
            success=success+1
            if success==2:
                print(str(datetime.now()) + 'AS7341 measurement failed twice on ' + str(M) + ', setting unity values')
                sysData[M]['AS7341']['current']['ADC0']=1
                DACS=['ADC1', 'ADC2', 'ADC3', 'ADC4', 'ADC5']
                for DAC in DACS:
                    sysData[M]['AS7341']['current'][DAC]=0

    output=[0.0,0.0,0.0,0.0,0.0,0.0]
    index=0
    DACS=['ADC0', 'ADC1', 'ADC2', 'ADC3', 'ADC4', 'ADC5']
    for wavelength in wavelengths:
        if wavelength != "OFF":
            output[index]=sysData[M]['AS7341']['current'][DACS[index]]
        index=index+1

    return output
```

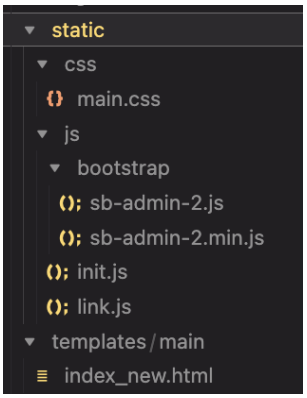
4.2.c Función ejemplo 'GetLight()' - 'controls.py'



4.2.d Estructura carpeta 'main'

El directorio *static/* contiene los archivos estáticos como CSS, JavaScript e imágenes que son necesarios para la apariencia y el comportamiento de la interfaz de usuario. Por ejemplo, el subdirectorio *css/* alberga el archivo 'main.css' que define los estilos visuales de la aplicación, mientras que el subdirectorio *js/* contiene scripts de JavaScript, incluidos aquellos necesarios para las bibliotecas de gráficos y tablas, así como archivos de inicialización y enlaces ('init.js', 'link.js'). El primer archivo se encarga de establecer los parámetros que luego se representan en la página principal. En cambio el segundo se encarga de todas las consultas AJAX que se realizan en función de los procesos.

El directorio *templates/* alberga las plantillas HTML utilizadas para generar el contenido de la interfaz de usuario de manera dinámica. Dentro de *templates/main/*, se encuentran el archivo index.html, que es la principal plantilla para la interfaz de usuario, diseñada para presentar los datos y permitir la interacción del usuario con el sistema.



4.2.d Estructura archivos estáticos y html

El directorio *migrations/* se utiliza para gestionar las migraciones de la base de datos, permitiendo actualizar el esquema de la base de datos a medida que se desarrollan nuevas características y se realizan cambios en los modelos.

Para asegurar la consistencia del entorno de desarrollo y facilitar el despliegue, se utiliza Docker para contenerizar la aplicación y sus dependencias. Docker permite crear entornos aislados que aseguran que se ejecute de manera consistente en cualquier entorno. La configuración de Docker incluye un Dockerfile que define cómo se crea la imagen de Docker para la aplicación, instalando las dependencias necesarias y configurando el servidor Gunicorn para servir la aplicación Django. Además, Docker se utiliza para levantar la base de datos PostgreSQL, asegurando que todas las dependencias y configuraciones necesarias estén encapsuladas en contenedores, lo que simplifica el proceso de configuración y garantiza que la base de datos esté lista para su uso en un entorno de producción.

El uso de Docker en el proyecto permite mantener un entorno de desarrollo limpio y replicable, evitando los típicos problemas de "funciona en mi máquina" al asegurar que todos los desarrolladores trabajen en un entorno idéntico. Docker también facilita la integración continua y el despliegue continuo (CI/CD), permitiendo que las nuevas versiones de la aplicación se prueben y desplieguen de manera automatizada. La gestión del despliegue de la aplicación ChiBio utilizando Docker asegura que el sistema sea robusto, escalable y fácil de mantener, permitiendo una transición suave desde el desarrollo hasta la producción y asegurando la integridad y consistencia del entorno de ejecución.

CAPÍTULO 5. IMPLEMENTACIÓN DE LA SOLUCIÓN

5.1. DESPLIEGUE DE LA APLICACIÓN

El despliegue de la aplicación ChiBio se ha diseñado para ser robusto y eficiente, utilizando Docker para contenerizar tanto la aplicación como sus dependencias. Este enfoque garantiza que la aplicación se ejecute de manera consistente en cualquier entorno, facilitando tanto el desarrollo como el despliegue en producción.

Para contenerizar la aplicación, se utiliza un archivo Dockerfile que define cómo se crea la imagen de Docker. Este archivo incluye instrucciones para instalar todas las dependencias necesarias y configurar el entorno de ejecución. Por ejemplo, se instalan las librerías de Python requeridas y se configura Gunicorn como servidor WSGI para servir la aplicación Django. Gunicorn se elige por su eficiencia y capacidad para manejar múltiples solicitudes concurrentes, lo que es crucial para una aplicación de alto rendimiento como ChiBio. Además del Dockerfile, se utiliza Docker Compose para gestionar múltiples contenedores. Docker Compose permite definir y ejecutar aplicaciones multi-contenedor, lo cual es ideal para configurar no solo la aplicación Django, sino también otros servicios necesarios como la base de datos PostgreSQL. La configuración de Docker Compose asegura que todos los componentes del sistema estén correctamente orquestados y que las dependencias entre ellos se manejen de manera automática.

Para la base de datos, se levanta un contenedor con PostgreSQL. Este enfoque asegura que la base de datos esté encapsulada en su propio entorno, lo que facilita la administración y el escalado. Todas las configuraciones necesarias, como las variables de entorno para la configuración de la base de datos, se definen en el archivo de Docker Compose. Esto incluye la configuración del volumen de Docker para la persistencia de datos, asegurando que los datos de la base de datos se mantengan intactos entre reinicios y actualizaciones del contenedor. El uso de volúmenes de Docker es crucial para mantener la persistencia de los datos. Los volúmenes permiten que los datos generados y utilizados por los contenedores se almacenen fuera del ciclo de vida de los contenedores, garantizando que no se pierdan cuando los contenedores se detengan o se reinicien.

Para desplegar la aplicación en un entorno de producción, se utilizan las siguientes etapas: construcción de la imagen de Docker utilizando el Dockerfile, despliegue de la aplicación y la base de datos con Docker Compose, y configuración de volúmenes para la persistencia de datos. Este proceso asegura que la aplicación esté lista para manejar cargas de trabajo reales y que pueda escalar según sea necesario. Además, se configura un proxy inverso para manejar las solicitudes HTTP y HTTPS, proporcionando una capa adicional de seguridad y mejorando el rendimiento mediante la gestión de la carga de tráfico y la implementación de certificados SSL para el cifrado de datos. Esta configuración se define en el archivo de Docker Compose, permitiendo que todos los servicios se integren de manera fluida.

El uso de Docker y Docker Compose simplifica significativamente el despliegue de la aplicación, asegurando que todos los componentes del sistema estén correctamente configurados y que las dependencias se gestionen de manera automática. Este enfoque también facilita el desarrollo

colaborativo, ya que los desarrolladores pueden replicar el entorno de producción en sus máquinas locales, asegurando que las pruebas y el desarrollo se realicen en un entorno que refleja con precisión el entorno de producción.

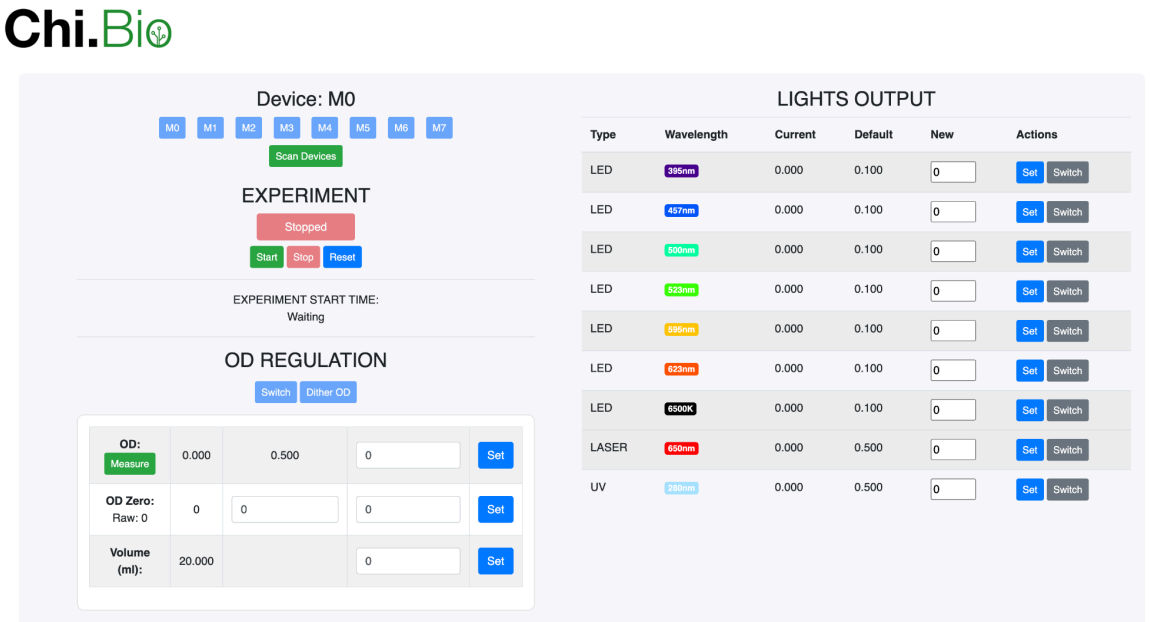
5.2. PRUEBAS DE USO

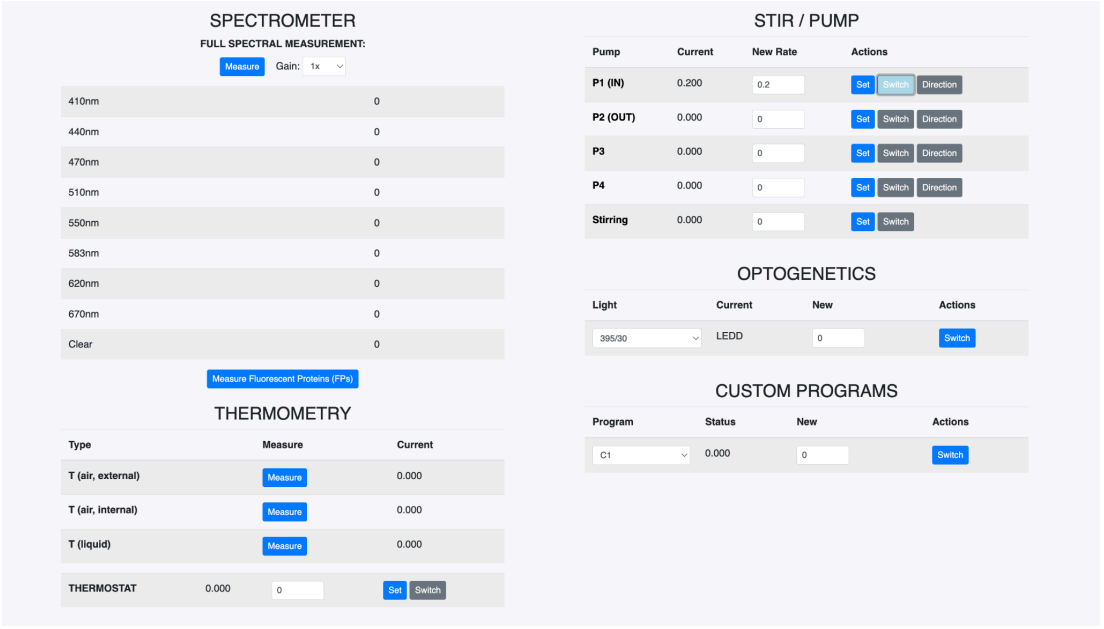
El primer paso para comprobar su funcionamiento es lanzar el servidor de Django. Para ello, ejecutamos los comandos correspondientes y comprobamos los logs obtenidos. Una vez el servidor está en marcha, tenemos que obtener siempre los datos del sistema, que se encuentran en el archivo JSON 'sysdata.json'.

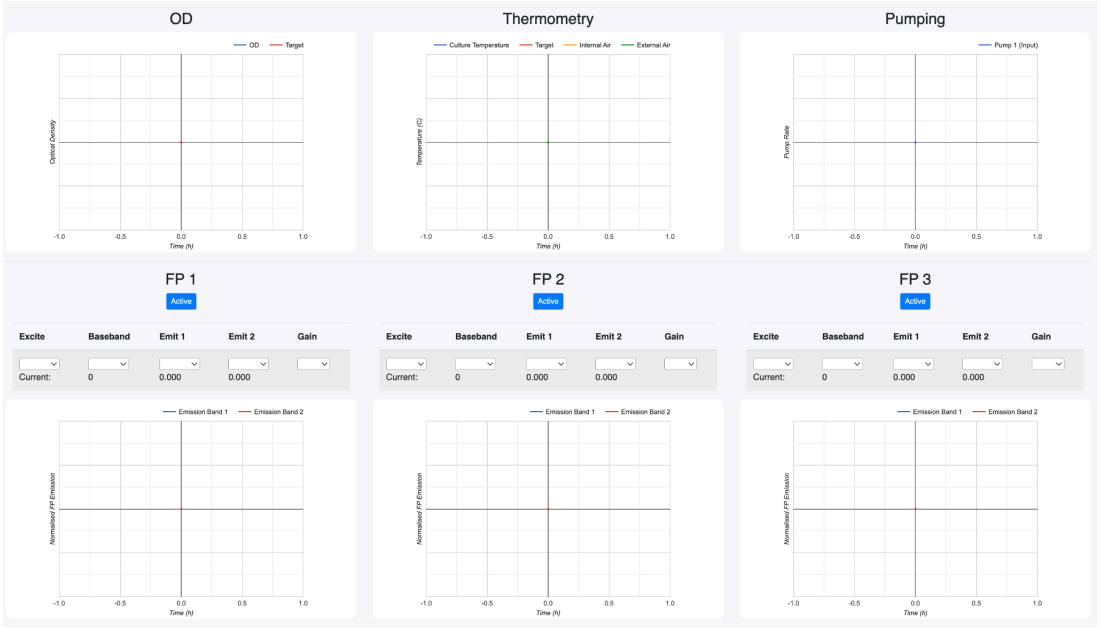
```
10.236.62.224 - - [20/Jun/2024 10:50:16] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:50:16] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:50:19] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:50:21] "GET /getSysdata/ HTTP/1.1" 200 -
```

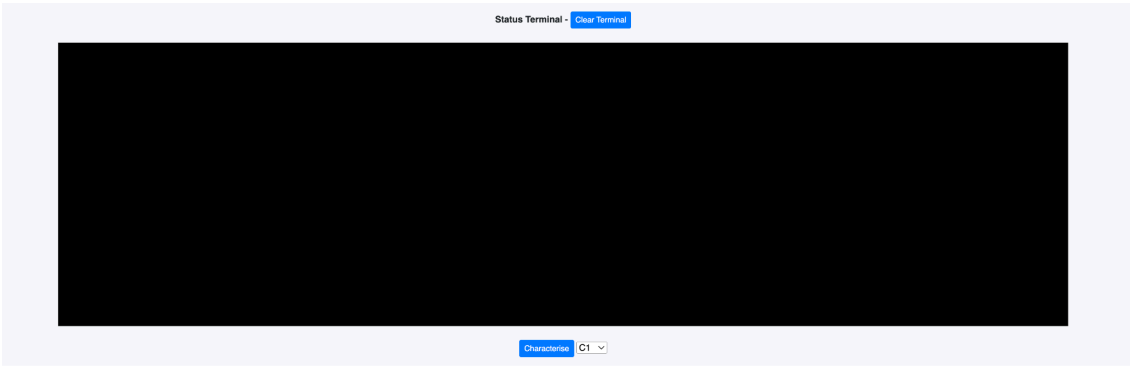
5.2.a Inicialización aplicación

Con el servidor en funcionamiento, ya podemos acceder a la URL donde está configurado el proyecto. Si se utiliza en local, se puede acceder a través de 'localhost:8000'. En un entorno de producción, se configurará una URL específica en el momento del despliegue. Durante las pruebas, se utilizó la IP '158.42.16.121', seleccionada según la configuración de Chibio en ese momento. Conectándonos desde nuestro navegador a la dirección mencionada, podremos ver la página de la aplicación que se ha diseñado específicamente para tener un uso más claro y fácil. Se pueden observar las diferencias con la antigua visualización en el anexo I.







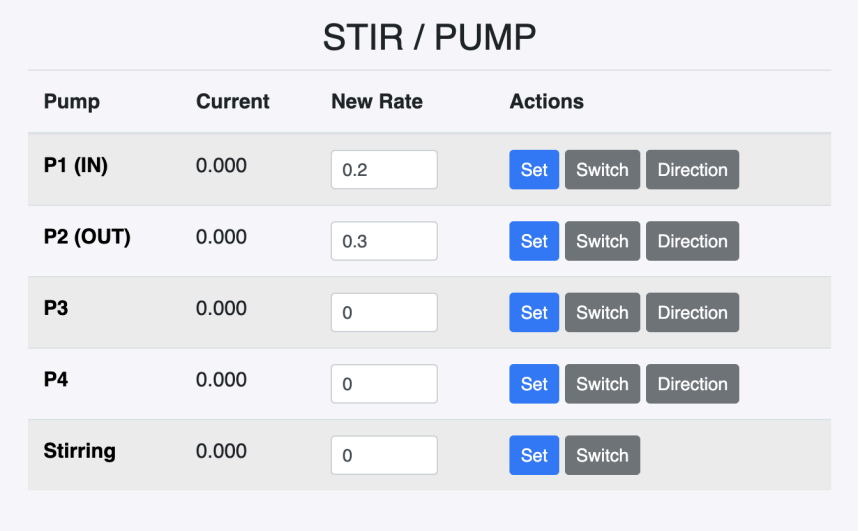


5.2.b Visualización de la página principal de la aplicación - 'index_new.html'

Luego de configurar el servidor y acceder a la interfaz web, procedemos a realizar una serie de pruebas exhaustivas para verificar la funcionalidad del proyecto. Estas pruebas son esenciales para asegurar que el software se comunica correctamente con el hardware y que todas las funciones operan como se espera. Las pruebas abarcan varias áreas clave del sistema, incluyendo la óptica, termometría, bombeo, y regulación de OD (densidad óptica), así como la gestión de luces y programas personalizados.

Para garantizar la correcta funcionalidad del proyecto, realizamos una serie de pruebas exhaustivas centradas en la comunicación y conexión entre el software y el hardware. Primero, en la configuración y pruebas de dispositivos, verificamos que todos los dispositivos conectados, desde M0 hasta M7, sean detectados correctamente por el sistema. Utilizamos la función “Scan Devices” para identificar y listar todos los dispositivos disponibles. Además, aseguramos que cada dispositivo pueda cambiar de estado (activo/inactivo) y que estos cambios se reflejan tanto en la interfaz web como en los logs del sistema.

En cuanto a las pruebas de interacción con el hardware, comprobamos que el programa envía las instrucciones correctas al hardware. Esto incluye comandos para activar luces, bombas y otros componentes del sistema. También verificamos que los datos enviados desde el hardware sean recibidos correctamente por el software.



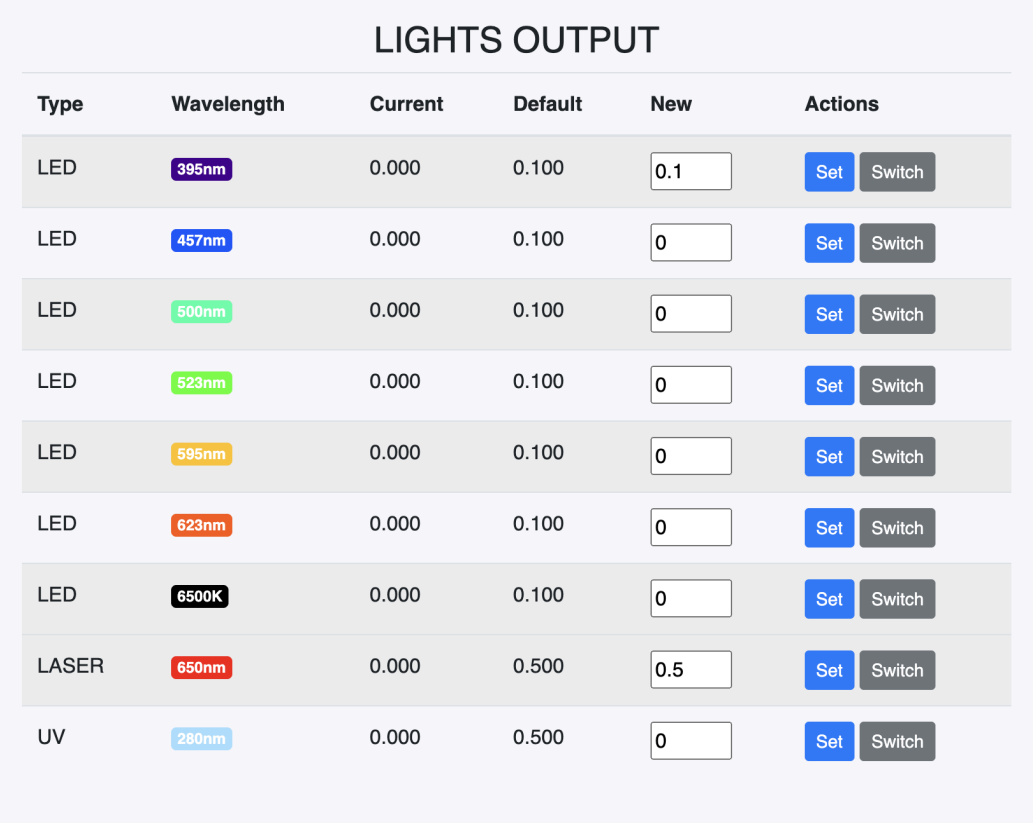

```
10.236.62.224 - - [20/Jun/2024 10:50:08] "POST /SetOutputTarget/Pump2/0/0.3 HTTP/1.1" 204 -
10.236.62.224 - - [20/Jun/2024 10:50:08] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:50:08] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:50:09] "POST /SetOutputOn/Pump2/2/0 HTTP/1.1" 204 -
10.236.62.224 - - [20/Jun/2024 10:50:09] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:50:10] "GET /getSysdata/ HTTP/1.1" 200 -
```

```
10.236.62.224 - - [20/Jun/2024 10:50:13] "POST /SetOutputTarget/Pump1/0/0.2 HTTP/1.1" 204 -
10.236.62.224 - - [20/Jun/2024 10:50:13] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:50:13] "POST /SetOutputOn/Pump1/2/0 HTTP/1.1" 204 -
10.236.62.224 - - [20/Jun/2024 10:50:13] "GET /getSysdata/ HTTP/1.1" 200 -
```

5.2.c Ejemplo de funcionamiento de las bombas

En el ejemplo mostrado en las imágenes 5.2.c, podemos observar la configuración de las bombas 1 y 2 del sistema. Al pulsar el botón 'Set' y activar el 'Switch', se genera un log que registra la configuración de la velocidad de la bomba. Este log se puede visualizar en el primer registro de cada imagen, con el formato 'Pump${numero_bomba}/{encendido_apagado}/${velocidad_bomba}'. En cuanto a la variable 'encendido_apagado', cuando se recibe un 0 es para iniciarlo, cuando se recibe un 2 es para apagarlo. La próxima vez que recibimos datos de la bomba es cuando volvemos a pulsar el botón 'Switch', esta vez para apagarlas, por lo que la velocidad registrada en el log será 0.

Finalmente, realizamos pruebas de funcionalidad específicas para asegurarnos de que el sistema opera correctamente en situaciones particulares. En el ámbito de la optogenética y control de luz, configuramos y probamos diferentes longitudes de onda y corrientes de luz LED para asegurarnos de que el sistema puede controlar y ajustar la iluminación correctamente. Validamos cada ajuste de luz mediante pruebas físicas y comparación de resultados en el laboratorio. Además, en la regulación de OD, realizamos pruebas para medir y ajustar la densidad óptica de las muestras. Utilizamos la opción "Measure" para obtener lecturas de OD y ajustamos los parámetros para ver cómo responde el sistema, asegurando que el control de OD sea preciso y confiable.




```
10.236.62.224 - - [20/Jun/2024 10:56:27] "POST /SetOutputTarget/LASER650/0/0.5 HTTP/1.1" 204 -
10.236.62.224 - - [20/Jun/2024 10:56:27] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:56:28] "POST /SetOutputOn/LASER650/2/0 HTTP/1.1" 204 -
10.236.62.224 - - [20/Jun/2024 10:56:28] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:56:28] "GET /getSysdata/ HTTP/1.1" 200 -
```

```
10.236.62.224 - - [20/Jun/2024 10:55:43] "POST /SetOutputTarget/LEDA/0/0.1 HTTP/1.1" 204 -
10.236.62.224 - - [20/Jun/2024 10:55:43] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:55:44] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:55:44] "POST /SetOutputOn/LEDA/2/0 HTTP/1.1" 204 -
10.236.62.224 - - [20/Jun/2024 10:55:44] "GET /getSysdata/ HTTP/1.1" 200 -
10.236.62.224 - - [20/Jun/2024 10:55:46] "GET /getSysdata/ HTTP/1.1" 200 -
```

5.2.d Ejemplo de funcionamiento de los LED

En las imágenes 5.2.d, se muestra cómo se configuran las luces LED y el láser. En este caso específico, se seleccionaron un LED de 395nm y un láser de 650nm. Al presionar el botón ‘Set’ y activar el ‘Switch’, se genera un log que documenta la configuración de la intensidad de las luces. Este registro aparece en el primer log de cada imagen, con el formato ‘${nombre}/${encendido_apagado}/${intensidad}’. Cuando volvemos a recibir datos de las luces, es debido a que se ha vuelto a pulsar el botón ‘Switch’ para apagarlas, resultando en un registro de intensidad de 0 en el log.

Esta documentación detallada del proceso de encendido y apagado tanto de las luces como de las bombas proporciona una clara evidencia de la correcta operación y respuesta del sistema a los comandos de usuario. Los registros generados y verificados físicamente permiten asegurarnos de que las instrucciones enviadas desde la interfaz de usuario son ejecutadas correctamente por el hardware, y que las respuestas recibidas son coherentes con las acciones realizadas. Esto no solo garantiza la precisión en el control de los dispositivos, sino que también facilita la detección y solución de posibles problemas en la interacción entre el software y el hardware.

CAPÍTULO 6. CONCLUSIONES

El desarrollo del sistema de gestión de protocolos para la plataforma de biorreactores ChiBio ha logrado cumplir con los objetivos planteados inicialmente en el proyecto. Desde el inicio, se estableció la meta de diseñar e implementar un software modular que permitiera una gestión eficiente y flexible de los protocolos experimentales en biología sintética. A lo largo del proyecto, se han superado diversos retos técnicos y se ha logrado integrar una serie de tecnologías avanzadas para alcanzar estos objetivos.

Uno de los principales logros del proyecto ha sido la migración del software existente a un framework más robusto y escalable como es Django. Esta migración no solo ha mejorado la estructura del código, haciéndolo más modular y mantenible, sino que también ha permitido la incorporación de nuevas funcionalidades de manera más sencilla. La utilización de este ha facilitado la organización del proyecto, la gestión de dependencias y la implementación de una interfaz de usuario más intuitiva y funcional.

Otro aspecto destacado ha sido la implementación de un sistema de control y observación en tiempo real. Este sistema permite ajustar los parámetros operativos de los biorreactores de manera dinámica, basándose en datos obtenidos de diversos sensores. La capacidad de monitorear y controlar en tiempo real es crucial para optimizar los procesos biológicos y asegurar la reproducibilidad de los experimentos. La integración de algoritmos de control avanzados y técnicas de sensorización software ha sido fundamental para lograr esta funcionalidad.

El uso de Docker para contenerizar la aplicación y sus dependencias ha sido un elemento clave para garantizar la consistencia del entorno de desarrollo y facilitar el despliegue en producción. Este ha permitido crear entornos aislados, asegurando que la aplicación se ejecute de manera consistente en cualquier entorno, evitando los problemas típicos de "funciona en mi máquina". La configuración de Docker Compose para gestionar múltiples contenedores, incluyendo la base de datos PostgreSQL, ha simplificado significativamente la administración del sistema y ha mejorado la escalabilidad.

La implementación de consultas AJAX ha mejorado la interactividad y eficiencia del sistema, permitiendo la comunicación asincrónica con el servidor y la actualización dinámica de la interfaz de usuario. Esta capacidad ha sido crucial para proporcionar una experiencia de usuario fluida y responsiva, mejorando la eficiencia operativa en la gestión de protocolos.

A lo largo del proyecto, se han aprendido y aplicado numerosos conocimientos y habilidades que no se habían abordado en profundidad durante los estudios. El dominio de Django, la configuración y uso de Docker, la implementación de consultas AJAX y la integración de algoritmos de control en tiempo real son algunas de las competencias técnicas adquiridas. Además, se ha aprendido a trabajar con bases de datos como PostgreSQL y a gestionar entornos de desarrollo de manera eficiente utilizando herramientas como virtualenv.

El proyecto también ha presentado varios desafíos, como la migración del código existente a una nueva estructura y la integración de múltiples tecnologías. Estos retos se han superado mediante

un enfoque sistemático y colaborativo, utilizando metodologías ágiles y revisiones de código para asegurar la calidad y robustez del sistema.

Desde una perspectiva personal y profesional, este proyecto ha sido una oportunidad valiosa para aplicar y expandir los conocimientos adquiridos durante la carrera. Ha permitido desarrollar habilidades en áreas clave de la ingeniería de software, como el diseño de sistemas, la programación orientada a objetos, y la gestión de proyectos. También ha proporcionado una visión práctica de los desafíos y oportunidades en la automatización de laboratorios y la biología sintética. En términos de competencias transversales, el proyecto ha requerido la aplicación de habilidades de resolución de problemas, gestión del tiempo y trabajo en equipo. La capacidad de comunicar ideas de manera clara y efectiva, tanto de forma escrita como verbal, ha sido crucial para el éxito del proyecto. Además, la necesidad de adaptarse a nuevas tecnologías y metodologías ha fomentado una actitud de aprendizaje continuo y mejora constante.

En conclusión, el proyecto de desarrollo de un sistema software modular para la gestión de protocolos en la plataforma de biorreactores ChiBio ha sido exitoso y ha cumplido con los objetivos establecidos. Se han superado diversos retos técnicos y se han adquirido habilidades valiosas que contribuirán significativamente al desarrollo profesional futuro. Este trabajo no solo ha mejorado la eficiencia y flexibilidad del sistema de gestión de protocolos, sino que también ha proporcionado una base sólida para futuras investigaciones y desarrollos en el campo de la biología sintética y la automatización de laboratorios.

PRESUPUESTO

Es importante destacar que uno de los objetivos del Trabajo de Fin de Grado (TFG) es valorar económicamente el trabajo realizado. Este presupuesto ayudará a estimar el costo total del proyecto, proporcionando una visión clara de los recursos utilizados y facilitando la planificación financiera. Se establece un precio la hora competitivo, obtenido de un análisis riguroso del mercado Español actual para un desarrollador de software junior. Este precio será de 20 euros por hora.

1. LISTADO DE MEDICIONES Y PRESUPUESTO

Fase 1 - Objetivos de la solución

Concepto	Cantidad	Coste € / Ud.	Total
Definición y planificación de objetivos	10	20	200,00 €
Identificación de requisitos funcionales y no funcionales	15	20	300,00 €
Investigación y análisis de soluciones	25	20	500,00 €
Documentación y presentación de la solución propuesta	10	20	200,00 €
Fase 1 - Total:	**60**	**80**	**1.200,00 €**

Fase 2 - Diseño de la solución

Concepto	Cantidad	Coste € / Ud.	Total
Diseño de la estructura del proyecto	25	20	500,00 €
Diseño de la arquitectura del proyecto	30	20	600,00 €
Planificación detallada del proyecto	10	20	200,00 €
Selección de las tecnologías utilizadas	5	20	100,00 €
Fase 2 - Total:	**70**	**80**	**1.400,00 €**

Fase 3 - Desarrollo de la solución

Concepto	Cantidad	Coste € / Ud.	Total
Configuración de las tecnologías utilizadas	10	20	200,00 €
Limpieza y reestructuración de las funciones	30	20	600,00 €
Clasificación de la funcionalidad	20	20	400,00 €
Departamentalización del código según su funcionalidad	20	20	400,00 €
Modularidad	25	20	500,00 €
Fase 3 - Total:	**105**	**100**	**2.100,00 €**

Fase 4 - Implementación de la solución

Concepto	Cantidad	Coste € / Ud.	Total
Paquetización de la solución con Docker	10	20	200,00 €
Pruebas de funcionamiento en local	5	20	100,00 €
Despliegue de la aplicación	10	20	200,00 €
Pruebas de funcionamiento en producción	5	20	100,00 €
Fase 4 - Total:	**30**	**80**	**600,00 €**

2. RESUMEN DEL PRESUPUESTO POR CAPÍTULOS

Concepto	Cantidad	Coste € / Ud.	Total
Capítulo 1: Introducción	0	20	- €
Capítulo 2: Análisis del problema	60	20	1.200,00 €
Capítulo 3: Diseño de la solución	70	20	1.400,00 €
Capítulo 4: Desarrollo de la solución	105	20	2.100,00 €
Capítulo 5: Implementación de la solución	30	20	600,00 €
Capítulo 6: Conclusión	0	20	- €
Total presupuesto:	**265**	**120**	**5.300,00 €**

Este presupuesto proporciona una estimación clara y detallada del costo total del proyecto, destacando la inversión necesaria en términos de tiempo y dinero para cada fase y capítulo. Con un coste total de 5.400 euros y 270 horas de trabajo, se ha valorado adecuadamente el esfuerzo y los recursos empleados en el desarrollo del proyecto de software del TFG.

Anexo I. Ejemplos comparativos

Este anexo expone las diferencias en cuanto a la visualización y el código de la aplicación antes y después del proceso de transformación con el fin de obtener una interfaz de usuario más intuitiva y cómoda de usar. Además de mostrar las mejoras visuales, se incluirán extractos del código fuente anterior y del nuevo para ilustrar los cambios técnicos implementados. Este análisis permitirá apreciar no solo la evolución en la estética y usabilidad de la interfaz, sino también las optimizaciones y refactorizaciones del código que han mejorado la eficiencia y la mantenibilidad del sistema. Las comparaciones detalladas proporcionarán una visión clara de cómo se ha mejorado la experiencia del usuario y la estructura del software, destacando los beneficios de las nuevas prácticas y tecnologías adoptadas.

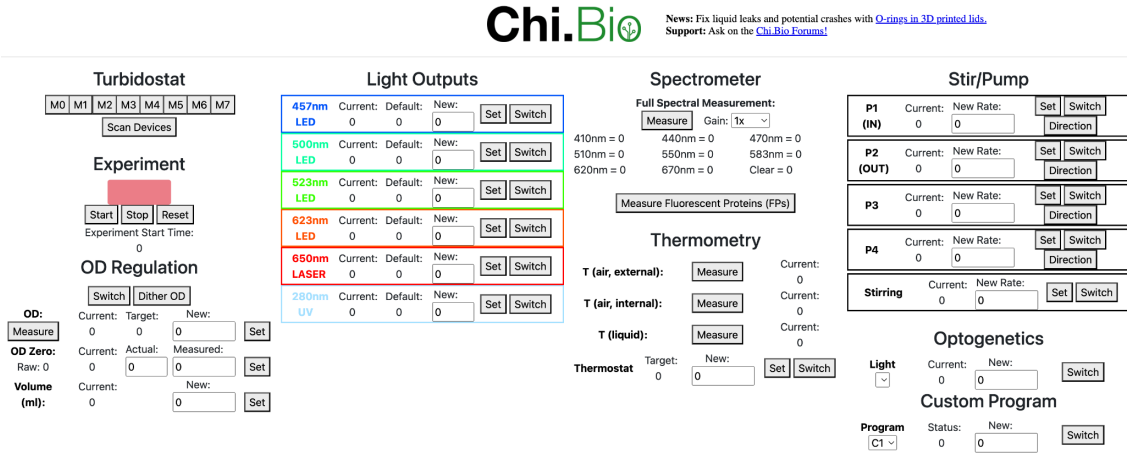


Fig.1. Visualización de la página principal de la aplicación antigua

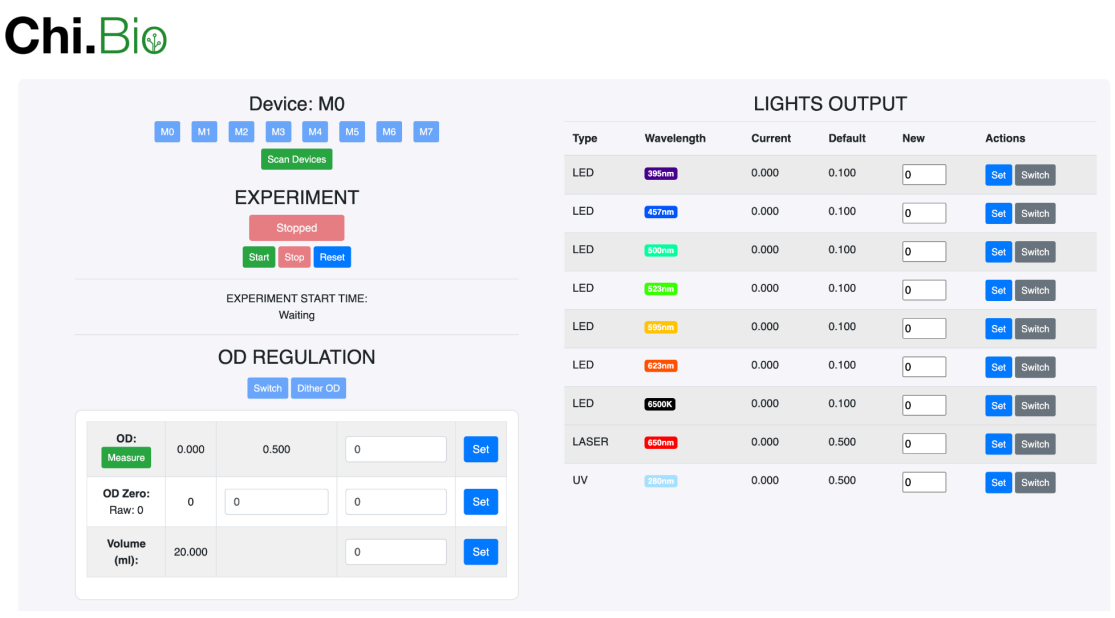


Fig.2. Visualización de la página principal de la aplicación nueva

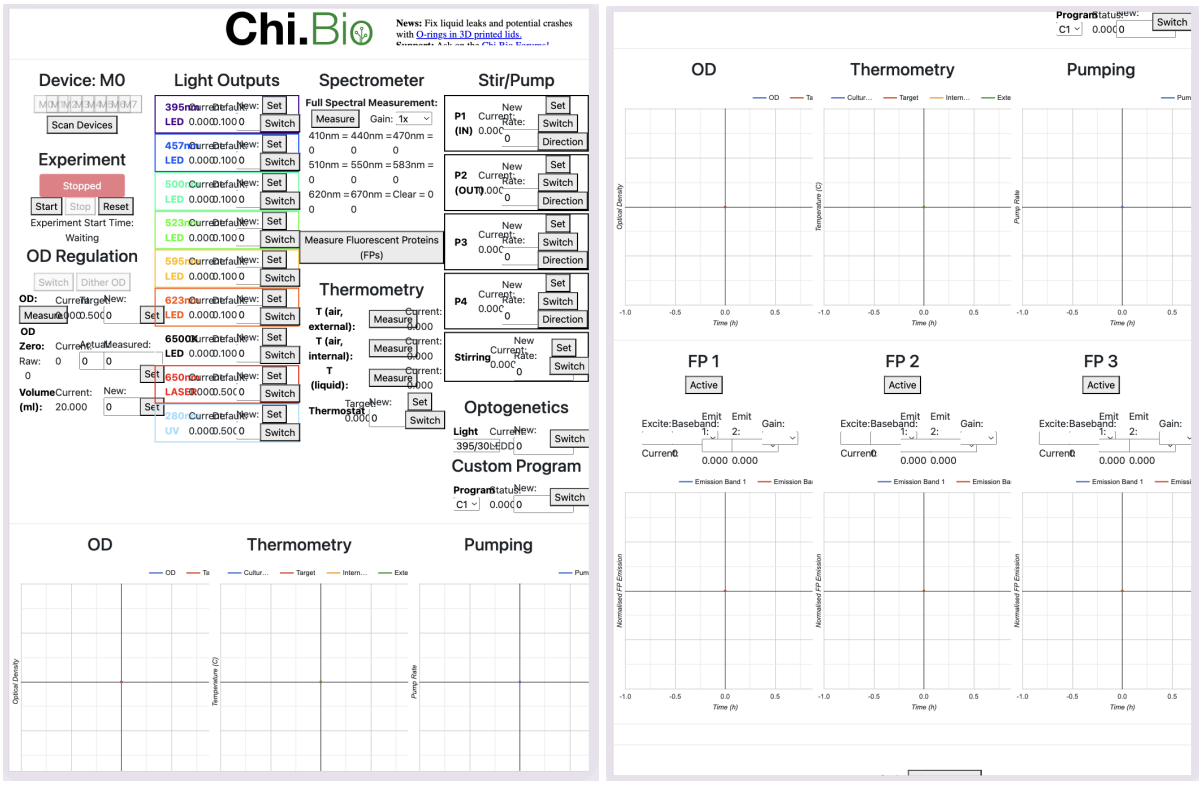


Fig.3. Visualización responsive para Ipad Pro de la aplicación antigua

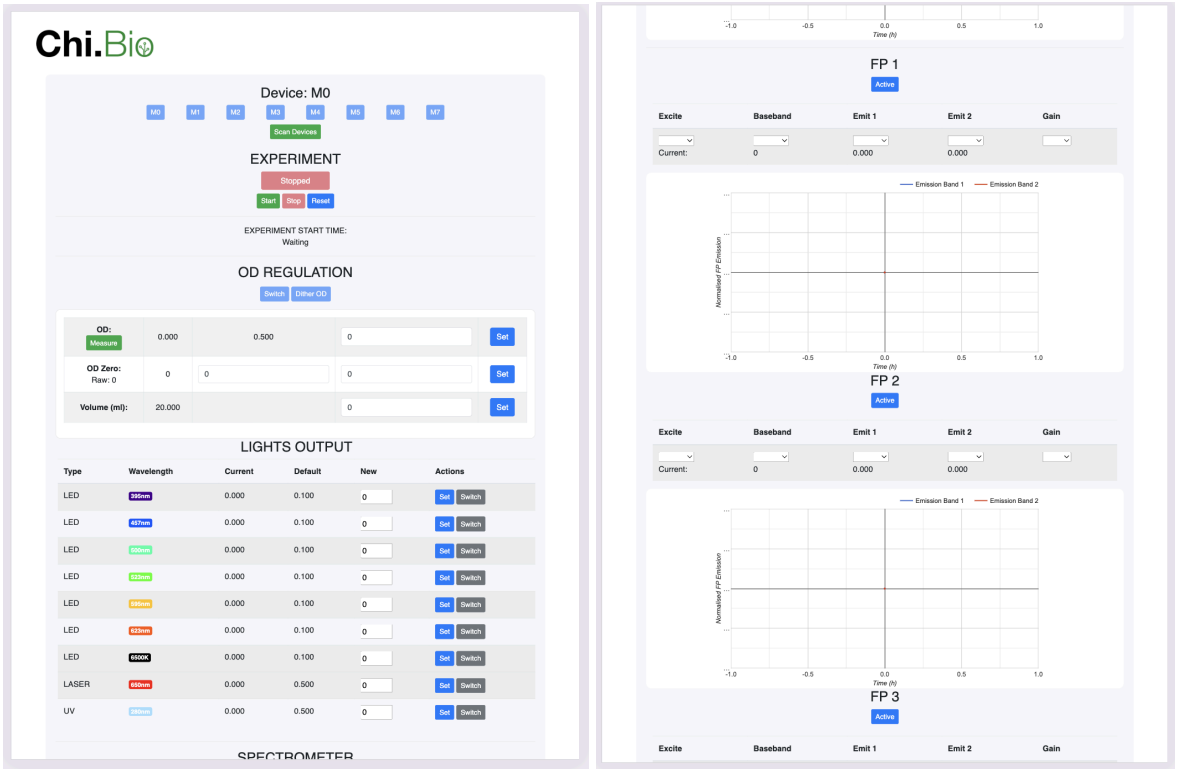


Fig.4. Visualización responsive para Ipad Pro de la aplicación nueva

```
             

             </div>
          </div>
          </div>

          <div id="chart_div6"></div>

        </div>

      </div>
      <hr>

      <div class="row">
        <div class="col-4">

        <div id="chart_div7"></div>

        </div>
      </div>

      <hr>

    <br>

    <!-- This section deals with the terminal, to send something to the terminal assign its value to TerminalText then run addTerminal() -->

    <center>
```

```
      </div>

      <div class="row">
            <div class="col-5 align-self-center">
             <b><font color="000000"> T (liquid): </font></b>

          </div>
          <div class="col-3 align-self-center">
              <button id="TempMeasure3">Measure</button> <br>

          </div>

            <div class="col-4 align-self-center">
              Current: <div id="TempCurrent3">0</div>

          </div>

      </div>

        <div class="row">
            <div class="col-3 align-self-center">
             <b><font color="000000">Thermostat</font></b>

          </div>
          <div class="col-2 align-self-center">
              Target: <br>
              <div id="ThermostatTarget">0</div>

          </div>

          <div class="col-3 align-self-center">
              New: <br>
              <input id="ThermostatInput" type="number" size="8" value=0 style="width: 6em"/>

          </div>
          <div class="col-4 align-self-center">
              <button id="ThermostatSet">Set</button> <button id="ThermostatSwitch" >Switch</button>

          </div>

      </div>


      </div>
  </div>
```

Fig.5. Estructura del código HTML anterior

Fig.6. Estructura del código HTML nueva

En la figura 5 y 6, podemos observar como el código se reestructura de forma clara y concisa, evitando que el usuario no pueda seguir una lógica a la hora de analizarlo. Además, de esta manera, favorecemos posibles modificaciones de este, mejorando así la escalabilidad de la aplicación.