



Educación
Secretaría de Educación Pública



TECNOLÓGICO
NACIONAL DE MÉXICO



TESIS

CON EL TEMA:

“BIORREACTOR SISTEMA DE CONTROL”

QUE PRESENTAN:

**OSCAR MANUEL AGUILAR HERNANDEZ
LUIS ERNESTO ROMAN CHITALA**

ASESORA

DRA. MAYRA ITZCALOTZIN MONTERO CORTES

REVISORES

**MTI. VIOLETA HAIDE PLAZOLA SOLTERO
DR. HECTOR MIGUEL GASTELUM GONZALEZ**

COMO REQUISITO PARCIAL PARA OBTENER EL TÍTULO DE:
INGENIEROS EN SISTEMAS COMPUTACIONALES

TLAJOMULCO DE ZÚÑIGA, JALISCO. MARZO, 2025.



Instituto Tecnológico de Tlajomulco
Dirección
Subdirección Académica

Tlajomulco de Zúñiga, Jalisco, 03/Marzo/2025
No. DE OFICIO: D.SA/DEP/346/2025

ASUNTO: Autorización de impresión
definitiva y digitalización.

C. OSCAR MANUEL AGUILAR HERNANDEZ
C. LUIS ERNESTO ROMAN CHITALA
PASANTES DE LA CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES
P R E S E N T E

Dado que el Comité dictaminó como **APROBADA** su TITULACIÓN INTEGRAL: OPCIÓN I (TESIS), con el tema
"**BIORREACTOR SISTEMA DE CONTROL**" y determinó que dan cumplimiento con los requisitos
establecidos, se les notifica que tienen la autorización para su impresión definitiva y digitalización.

Sin otro particular quedo de usted.

A T E N T A M E N T E

Excelencia en Educación Tecnológica®

Educando para la sociedad Actual y los Retos del futuro®

C. MARÍA ISABEL BECERRA RODRÍGUEZ
DIRECTORA DEL PLANTEL



C.c.p.- Coordinación Titulación. - Edificio
C.c.p.- Minutario. -
MIBR/RNP/VHPS/mjhc*



2025
Año de
La Mujer
Indígena

KM. 10 carretera a San Miguel Cuyutlán, Cto. Vicente Fernández
Gómez C.P.45640 Tlajomulco de Zúñiga, Jal. Tel. 33 29 02 11 30 ext.1015
e-mail: plan_tlajomulco@tecnm.mx www.tlajomulco.tecnm.mx





Instituto Tecnológico de Tlajomulco
Dirección
Subdirección Académica
Departamento de Ingenierías

Tlajomulco de Zúñiga, Jalisco, 01/Marzo/2025
No. DE OFICIO: D.SA/DING/055/2025

ASUNTO: Liberación de proyecto para
la titulación integral.

MTI. VIOLETA HAIDE PLAZOLA SOLTERO
JEFA DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES
P R E S E N T E

Por este medio informo que ha sido liberado el siguiente proyecto para la titulación integral:

NOMBRE DEL ESTUDIANTE Y/O EGRESADO:	OSCAR MANUEL AGUILAR HERNANDEZ
NO. DE CONTROL:	20940332
PRODUCTO:	OPCIÓN I (TESIS)
CARRERA:	INGENIERÍA EN SISTEMAS COMPUTACIONALES
NOMBRE DEL PROYECTO:	"BIORREACTOR SISTEMA DE CONTROL"

Agradezco de antemano su valioso apoyo en esta importante actividad para la formación profesional de nuestros egresados.

ATENTAMENTE

Excelencia en Educación Tecnológica®
Educando para la sociedad Actual y los Retos del futuro®



MC. FABIOLA BEJARANO REBOLLEDO
JEFA DEL DEPARTAMENTO DE INGENIERÍAS

 DRA. MAYRA ITZCALOTZIN MONTERO CORTES Nombre y firma del asesor	 MTI. VIOLETA HAIDE PLAZOLA SOLTERO Nombre y firma del revisor	 DR. HECTOR MIGUEL GASTELUM GONZALEZ Nombre y firma del revisor
---	---	--

C.c.p.- Coordinación Titulación. - Edificio
C.c.p.- Minutario. -
FBR/mjhc*



2025
Año de
La Mujer
Indígena

KM. 10 carretera a San Miguel Cuyutlán, Cto. Vicente Fernández
Gómez C.P.45640 Tlajomulco de Zúñiga, Jal. Tel. 33 29 02 11 30 ext.1015
e-mail: plan_tlajomulco@tecnm.mx www.tlajomulco.tecnm.mx





Instituto Tecnológico de Tlajomulco
Dirección
Subdirección Académica
Departamento de Ingenierías

Tlajomulco de Zúñiga, Jalisco, 01/Marzo/2025
No. DE OFICIO: D.SA/DING/054/2025

ASUNTO: Liberación de proyecto para
la titulación integral.

MTI. VIOLETA HAIDE PLAZOLA SOLTERO
JEFA DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES
P R E S E N T E

Por este medio informo que ha sido liberado el siguiente proyecto para la titulación integral:

NOMBRE DEL ESTUDIANTE Y/O EGRESADO:	LUIS ERNESTO ROMAN CHITALA
NO. DE CONTROL:	20940385
PRODUCTO:	OPCIÓN I (TESIS)
CARRERA:	INGENIERÍA EN SISTEMAS COMPUTACIONALES
NOMBRE DEL PROYECTO:	"BIORREACTOR SISTEMA DE CONTROL"

Agradezco de antemano su valioso apoyo en esta importante actividad para la formación profesional de nuestros egresados.

ATENTAMENTE

Excelencia en Educación Tecnológica®

Educando para la sociedad Actual y los Retos del futuro®

MC. FABIOLA BEJARANO REBOLLEDO
JEFA DEL DEPARTAMENTO DE INGENIERIAS

DRA. MAYRA ITZCALOTZIN NONTERO
CORTES
Nombre y firma del asesor

MTI. VIOLETA HAIDE PLAZOLA SOLTERO
Nombre y firma del revisor

DR. HECTOR MIGUEL GASTELUM GONZALEZ
Nombre y firma del revisor

C.c.p.- Coordinación Titulación. - Edificio
C.c.p.- Minutario. -
FBR/mjhc*



2025
Año de
La Mujer
Indígena

KM. 10 carretera a San Miguel Cuyutlán, Cto. Vicente Fernández
Gómez C.P.45640 Tlajomulco de Zúñiga, Jal. Tel. 33 29 02 11 30 ext.1015
e-mail: plan_tlajomulco@tecnm.mx www.tlajomulco.tecnm.mx



Agradecimientos

Queremos manifestar nuestro más profundo agradecimiento a todas aquellas personas que han desempeñado un papel esencial en la realización de este proyecto de tesis.

En primer lugar, extendemos nuestro agradecimiento al Instituto Tecnológico de Tlajomulco por habernos permitido llevar a cabo este proyecto en sus instalaciones y por el respaldo constante durante todo el proceso. De manera especial, reconocemos la invaluable contribución de la Dra. Mayra Itzcalotzin Montero Cortes, nuestra asesora, y de la M.C. Ana Julia Cardoso Magaña, cuya dedicación, orientación y consejos fueron fundamentales para el desarrollo y éxito de este trabajo. Su experiencia y sabiduría resultaron indispensables en cada etapa.

Asimismo, agradecemos a nuestros compañeros y colegas, José Alonso Figueroa Martínez e Iván Uriel González Segundo, por su colaboración y el intercambio de ideas que enriquecieron nuestro aprendizaje. Su apoyo fue determinante para el progreso de cada fase de este proyecto.

A nuestras familias y amigos, les expresamos nuestro agradecimiento por su comprensión, paciencia y respaldo incondicional durante esta etapa. Su motivación nos brindó la fortaleza necesaria para enfrentar los desafíos y finalizar este esfuerzo.

Por último, nos reconocemos a nosotros mismos, tanto de manera individual como en equipo, por el esfuerzo, la dedicación y la colaboración que nos permitieron superar obstáculos y alcanzar los objetivos propuestos.

Dedicatorias

A mi madre y a todos los que me apoyaron a lo largo de este camino.

Luis Ernesto Román Chitala

Este proyecto y toda mi carrera se la dedico a Rosa Hernandez mi madre.

Oscar Manuel Aguilar Hernandez

Índice

Portada	I
Hoja de firmas	II
Agradecimientos	VI
Dedicatorias	VII
Índice	VIII
Índice de figuras	XIII
Resumen	XIV
Abstract	XV
Introducción	1
Marco teórico	2
Biorreactor	2
Sistemas de Inmersión.....	2
Categorías de Sistemas de Inmersión	2
Sistemas de inmersión parcial y un mecanismo de renovación del medio líquido	2
Esterilización.....	3
Autoclave	3
Fotosíntesis	3
Fotoperiodo.....	3
<i>In Vitro</i>	4
Micropropagación	4
Tarjeta de desarrollo	4
Arduino Uno	4
Orange PI	5
ESP-32	5
Relé	5
Electroválvula (Mando indirecto)	5
Redes informáticas	6
Protocolo HTTP	6
TCP/IP	6
Red de área local.....	6
Sistema Linux	7
Distribución Linux.....	7

Ubuntu	7
Consola Bash	8
Script Shell.....	8
Cron.....	8
Crontab	8
Desarrollo multiplataforma	9
Lenguaje de programación	9
Lenguaje de programación compilado	9
Lenguaje de programación interpretado.....	9
Patrones de diseño	9
Arquitectura Limpia	10
Arquitectura hexagonal	10
Inyección de dependencias.....	10
Sistemas de control de versiones	10
Git.....	11
Github	11
Bases de datos	11
Sistema de gestión de bases de datos.....	11
Modelo relacional.....	11
MySQL.....	12
ORM	12
Lenguaje de programación GO	12
Lenguaje de programación JavaScript	12
Framework de aplicación web.....	12
Librería	13
Gin.....	13
Wails.....	13
Bootstrap	13
React.js.....	13
Antecedentes	14
Problemática	16
Justificación	17
Objetivos	19

General.....	19
Específicos	19
Hipótesis	20
Materiales y métodos	21
Evaluación inicial de la solución existente.....	21
Hardware.....	21
Tecnologías.....	22
Implementación.....	24
Diagrama de sistema.....	24
Programación de los módulos	25
Preparación del entorno.....	25
Librerías usadas	25
Flujo del código.....	25
Compilación y flasheo	26
Resultados y discusión	28
Evaluación inicial de la solución existente.....	28
Planificación de la solución	28
Integración de la red local.....	29
Desarrollo de la aplicación.....	29
Diseño de la base de datos.....	30
Sensor.....	32
Module	32
SensorType.....	32
ModulePin	32
Activity.....	32
User	32
Configuration.....	33
Execution	33
Funcionamiento de un proyecto de Wails	34
Renderizado de componentes web en una aplicación nativa de escritorio	34
Comunicación entre la interfaz del cliente y la lógica del lado del “servidor”	35
Configuración de un entorno de Wails (Entorno Ubuntu)	35
Instalación de Golang.....	35

Instalación de Node.js y npm.....	35
Instalación de dependencias adicionales.....	36
Instalación y verificación de Wails	36
Creación de un proyecto de wails	36
Organización de un proyecto de Wails.....	37
Desarrollo del backend	38
Punto de Entrada del proyecto	38
Instanciación de las aplicaciones mediante la inyección de sus dependencias	39
Capa de aplicación.....	40
Capa de controlador.....	41
Capa de servicio.....	42
Capa de repositorio	43
Capa de integraciones.....	44
Capa de api.....	45
Capa de middleware	46
Casos de uso generales.....	47
Casos de uso principales	47
Ejecución de trabajos programados	50
Interacción de los componentes del sistema.....	50
Desarrollo de la interfaz	51
Rutas.....	51
Páginas del sistema	52
Modales de control de inmersión manual	54
Modal de control de inmersión programada.....	55
Modal de control de fotoperiodo programado	56
Etapas de construcción del ejecutable.....	57
Consumo de recursos.....	58
Producto final.....	58
Conclusiones	61
Desarrollo de un software eficiente y modular.....	61
Optimización de la interacción entre software y hardware.....	61
Interfaz de usuario intuitiva y accesible	61
Automatización y optimización de procesos.....	62

Flexibilidad y capacidad de expansión	62
Referencias bibliográficas	63

Índice de figuras

Figura 1. Arquitectura de la solución del sistema de control de biorreactores	23
Figura 2. Diagrama del sistema	24
Figura 3. Router	29
Figura 4. Arquitectura Hexagonal Créditos (Padilla Delgado, 2021)	30
Figura 5. Diagrama entidad relación creado automáticamente con el software DBeaver	34
Figura 6. Distribución actual del sistema	38
Figura 7. Punto de entrada de un proyecto de Wails	39
Figura 8. Ejemplo visual de la inyección de dependencias en el sistema	40
Figura 9. Ejemplo actual de la inyección de dependencias para la instancia final de la capa de aplicación	40
Figura 10. Descripción de la estructura de la capa de aplicación de un proyecto de Wails usando el dominio Ejecución como ejemplo	41
Figura 11. Estructura de la capa de controlador	42
Figura 12. Estructura de la capa de servicio	43
Figura 13. Estructura de la capa de repositorio	44
Figura 14. Implementación del cliente HTTP para interactuar con los módulos ESP-32	45
Figura 15. Implementación del api del sistema del biorreactor utilizando gin	46
Figura 16. Implementación del middleware de autorización basado en verificación de headers	47
Figura 17. Primera parte del método de creación del script de ejecución de intervalo de inmersión	49
Figura 18. Segunda parte del método de creación del script de ejecución de intervalo de inmersión	49
Figura 19. Diagrama de procesos	50
Figura 20. HashRouter de la aplicación del lado del cliente	52
Figura 21. Ruta raíz	52
Figura 22. Página de inicio	53
Figura 23. Página de listado de módulos	53
Figura 24. Página detalle de módulo	54
Figura 25. Modal de ejecución de inmersión manual	55
Figura 26. Modal de programación de intervalo de inmersión	55
Figura 27. Modal de programación de intervalo de fotoperiodo	56
Figura 28. Línea de biorreactores con luz fría	58
Figura 29. Líneas de biorreactores	59
Figura 30. Sistema instalado en la placa de desarrollo	60

Resumen

En los últimos años, el desarrollo de tecnologías para el control y monitoreo de sistemas biotecnológicos, como biorreactores para cultivos de importancia económica, ha cobrado gran relevancia, especialmente en el ámbito de la investigación y en empresas dedicadas a la multiplicación de plantas a gran escala. Sin embargo, uno de los principales desafíos radica en el alto costo y la complejidad de los equipos comerciales, lo que limita su accesibilidad. Para enfrentar esta problemática, han surgido diversas alternativas basadas en software y hardware de código abierto, así como en metodologías de desarrollo que reducen costos y optimizan la eficiencia de los sistemas de multiplicación de plantas en condiciones *in vitro*.

El núcleo del sistema está basado en la placa Orange Pi 5, que ejecuta un software desarrollado para gestionar las operaciones de los módulos mediante peticiones HTTP en una red local. La comunicación inalámbrica permite ejecutar acciones tanto manuales como programadas, mientras que el software aprovecha las capacidades del kernel de Linux para orquestar tareas automatizadas. Estas tareas se implementan mediante scripts de bash programados con expresiones cron, lo que garantiza la ejecución precisa de las inmersiones y los ciclos de luz según la configuración establecida.

Entre los logros más destacados se encuentra la mejora en la fiabilidad del sistema y su escalabilidad. El software permite controlar múltiples líneas de biorreactores de manera simultánea y con configuraciones independientes, lo que optimiza la producción de plantas *in vitro*. Además, la arquitectura del sistema facilita la integración de nuevos módulos, adaptándose a las necesidades de capacidad instalada y conexiones hidráulicas disponibles.

Palabras clave: Biorreactor, micropropagación, automatización,

Abstract

This work describes the improvements made to the prototype of a medium exchange immersion system, with a main focus on the development and optimization of the control software. The proposal integrates a modular design that separates the bioreactor lines, managed by independent modules controlled by a multiplatform application. This software, designed with low computational requirements, has an intuitive graphical interface operated through a touch screen, facilitating the programming of the photoperiod and the immersion without the need for additional hardware or advanced technical knowledge.

The core of the system is based on the Orange Pi 5 board, which runs a software developed to manage the operations of the modules through HTTP requests on a local network. Wireless communication allows to execute both manual and scheduled actions, while the software takes advantage of the capabilities of the Linux kernel to orchestrate automated tasks. These tasks are implemented through bash scripts programmed with cron expressions, ensuring the precise execution of the immersions and light cycles according to the established configuration.

Among the most notable achievements is the improvement in the reliability of the system and its scalability. The software allows for the control of multiple bioreactor lines simultaneously and with independent configurations, which optimizes the production of in vitro plants. In addition, the system architecture facilitates the integration of new modules, adapting to the needs of installed capacity and available hydraulic connections.

Keywords: Bioreactor, micropropagation, automation.

Introducción

El desarrollo de software para controlar biorreactores de inmersión temporal representa un avance clave en biotecnología y cultivo de plantas *in vitro*. Este proyecto tiene como objetivo crear una aplicación automatizada que gestione con precisión los tiempos de inmersión y otros parámetros críticos, superando las limitaciones de los sistemas actuales, como la falta de exactitud en la regulación. La aplicación se ejecuta en una Orange Pi, que actúa como núcleo de control, y opera junto con módulos relés y dispositivos ESP-32 S3 conectados en una red local, permitiendo una programación precisa y adaptable.

La aplicación cuenta con una interfaz intuitiva que facilita la configuración y monitoreo de los procesos, eliminando la necesidad de conocimientos técnicos avanzados. Su arquitectura modular permite integrar múltiples biorreactores con configuraciones independientes, optimizando la eficiencia y escalabilidad del sistema. Esto asegura un control más preciso y adaptable a las necesidades específicas de cada cultivo.

Entre las ventajas del software destacan la reducción de costos operativos, la mejora en la confiabilidad del proceso y su capacidad para adaptarse a diferentes entornos de producción. Estas características no solo aumentan la precisión en el control de los biorreactores, sino que también mejoran la calidad y consistencia en la producción de plantas *in vitro*.

Marco teórico

Biorreactor

Un biorreactor, puede considerarse como un espacio delimitado en el que, en presencia y con la intervención de un biocatalizador, ocurre una transformación de materia. Normalmente ocurre un intercambio activo de materia y calor con el entorno. (Chmiel et al., 2018, 197)

Sistemas de Inmersión

Son sistemas que permiten el contacto entre un ente biológico y un medio líquido así como también, la renovación completa de la atmósfera del cultivo mediante el uso de presión, la cual impulsa el líquido hacia el material vegetal. (Herve & M, 2002, 219)

Categorías de Sistemas de Inmersión

Según (Hervé & M, 2002) los sistemas de inmersión temporal para propagación de plantas se dividen en 4 categorías según su funcionamiento:

- Máquinas de inclinación y oscilación
- Inmersión completa del material vegetal y renovación del medio de cultivo
- Inmersión parcial y un mecanismo de renovación del medio nutritivo líquido
- Inmersión completa mediante transferencia neumática del medio líquido y sin renovación del medio nutritivo

Sistemas de inmersión parcial y un mecanismo de renovación del medio líquido

En este sistema el ente vegetal primeramente se coloca sobre un soporte de cultivo (tal como un sustrato o agar agar), para después aplicar el medio de cultivo con la frecuencia establecida hasta ser retirado mediante una acción de drenado, considerando que la vida útil del medio de cultivo puede verse afectado por los

componentes del sistema de transferencia del mismo aunque la base vegetal sea sumergida parcialmente. (Herve & M, 2002, 219)

Esterilización

Se define esterilización como el proceso que elimina de manera efectiva agentes tales como hongos, bacterias y virus de elementos tales como superficies, equipos o alimentos. Esta se mide mediante el nivel de garantía de esterilidad (SAL) (Vishal Gupta & Shukshith, 2016, 220)

Autoclave

El autoclave es una cámara de presión utilizada para realizar esterilización mediante la aplicación de temperaturas y presiones altas, este se aplica en diferentes industrias, tales como la química y la médica. (Vishal Gupta & Shukshith, 2016, 220) Cita Corta

Fotosíntesis

Proceso físico-químico mediante el cual las plantas y otros organismos fotosintéticos utilizan la energía solar para sintetizar compuestos orgánicos. El proceso a grandes rasgos conlleva la liberación de oxígeno como resultado del consumo de dióxido de carbono para lograr así la síntesis de sus compuestos orgánicos. (Pérez-Urria Carril, 2009, 1)

Fotoperiodo

Periodo temporal durante el cual se expone a las plantas a una fuente luz como parte fundamental para el proceso de fotosíntesis, este se puede manipular controlando la irradiación y la duración comúnmente medido en ciclos y relacionado al periodo de luz solar durante el día. (Jiménez-Ruiz et al., 2023, 77)

In Vitro

Dentro del contexto de las plantas, es un método de cultivo dentro de un frasco de vidrio dentro de un ambiente artificial. Dicho método se caracteriza de dos conceptos fundamentales:

- Asepsia (ausencia de gérmenes)
- Control de factores que afectan al crecimiento.

Este método ha permitido un estudio acelerado de plantas a diferentes niveles, así como también permitir la reproducibilidad de experimentos.(Castillo, 2004, 1)

Micropropagación

Es un método de propagación realizado en condiciones de laboratorio y que puede servir para la etapa de plantación, el cual tiene un impacto de mejora en las plantas, entre otras áreas. Se puede aplicar utilizando medios sexuales y asexuales, comúnmente utilizado para mejorar el material biológico, en cuestiones de homogeneidad y salubridad. (Kumar et al., 2019, 44)

Tarjeta de desarrollo

Circuito electrónico que contiene un microcontrolador o dispositivo lógico y elementos como puertos, conectores y reguladores que permiten acceder a las interfaces de alto nivel sin necesidad de tanta configuración pero a la vez también brindan acceso a los componentes de bajo nivel (Palma & Osorio, 2018, 45)

Arduino Uno

Placa de microcontrolador basada en el ATmega328P que cuenta con 14 pines de entrada/salida digital además de 6 entradas analógicas, cuenta con conexión usb, una toma de alimentación, un encabezado ICSP y un botón de reinicio. Puede alimentarse con un cable USB o un adaptador de corriente alterna a continua. (Arduino, S.f)

Orange Pi

Placa de desarrollo equipada con el procesador de 8 núcleos Rockchip RK3588S (4 núcleos A76 + 4 núcleos A55), con una frecuencia de hasta 2.4 GHz y un diseño de proceso de 8 nm. Integra una GPU ARM Mali-G610, un módulo de aceleración de imágenes 3D/2D y una NPU con capacidad de cálculo de 6 Tops. Ofrece opciones de memoria de 4GB, 8GB o 16GB (LPDDR4/4X) y soporte para salida de vídeo 8K. Sus interfaces incluyen HDMI, GPIO, M.2 PCIe 2.0, USB 2.0 y 3.0, puerto LAN Gigabit, entre otros. Es compatible con sistemas operativos como Orange Pi OS, Android 12 y Debian 11, y se utiliza en aplicaciones de inteligencia artificial, computación en la nube, seguridad inteligente, hogares inteligentes y más. (Orange Pi, n.d.)

ESP-32

Chip combinado que cuenta principalmente con Wi-Fi y Bluetooth de 2.4 GHz, diseñado con la tecnología de bajo consumo de 40 nm de TSMC. Está diseñado para lograr el mejor rendimiento en cuanto a energía y RF, mostrando robustez, versatilidad y fiabilidad en una amplia variedad de aplicaciones y escenarios de potencia. (EXPRESSIF, n.d.)

Relé

Término general con el que se refiere a todos los dispositivos que utilizan una pequeña señal eléctrica para activar una carga más grande de alta corriente. Este término va más allá de las aplicaciones automotrices, abarcando una amplia gama de aplicaciones eléctricas. (Littelfuse, n.d.)

Electroválvula (Mando indirecto)

Está provista de una electroválvula de mando directo que bajo tensión alimenta un operador neumático. Se trata de un distribuidor neumático accionado por una señal eléctrica. De las cuales se encuentran dos variantes:

- Servoasistidas (o autoalimentadas): el operador se alimenta del aire de la conexión de alimentación principal 1 del distribuidor

- Alimentadas externamente: conceptualmente iguales a las servoasistidas pero con la alimentación externa del operador (PNEUMAX, n.d.)

Redes informáticas

Permite la conexión de varios ordenadores o computadores entre sí, mediante hardware y software, permitiendo que exista un receptor y emisor. (Oviedo Bayas et al., 2018, 3)

Protocolo HTTP

El protocolo de transferencia de hipertexto (HTTP, Hypertext Transfer Protocol) es el protocolo base de la web, permite las aplicaciones cliente-servidor que permitan la utilización de hipertextos, este protocolo permite transmitir la información con la eficiencia necesaria para efectuar saltos de hipertexto. (Stallings, 2004, 780)

TCP/IP

Arquitectura de protocolos resultado de la investigación originada de la red experimental de conmutación de paquetes ARPANET. Inicialmente con fines militares Consisten en una extensa colección de protocolos que se han especificado como estándares de Internet por parte de IAB (Internet Architecture Board). Resuelve el problema de la comunicación en cinco capas relativamente independientes entre sí

- Capa física.
- Capa de acceso a la red.
- Capa internet.
- Capa extremo-a-extremo o de transporte.
- Capa de aplicación (Stallings, 2004, 40)

Red de área local

También conocida como LAN es una red de comunicaciones que interconecta varios dispositivos y proporciona un medio para el intercambio de información entre ellos, cuenta con las siguientes características:

- La cobertura es pequeña,

- Suele ser de propiedad de la organización la cual conecta los dispositivos a la red
- Usualmente las velocidades de transferencia suelen ser mayores a las de otras redes

Elas pueden utilizar diferentes tipos de canales tales como:

- Cableado
- Comunicación inalámbrica (Stallings, 2004, 17)

Sistema Linux

El sistema básico o kernel de Linux es un entorno estándar para aplicaciones y programación de usuarios, pero no impone un medio estándar para gestionar la funcionalidad disponible en su totalidad. Su evolución a través del tiempo ha permitido que surjan diferentes distribuciones acorde a los diferentes tipos de necesidades de usuarios. (Silberschatz, 2017, 776)

Distribución Linux

Una distribución de Linux incluye todos los componentes estándar del kernel de Linux, además de un conjunto de herramientas administrativas para simplificar la instalación inicial, las actualizaciones posteriores de Linux y la gestión de la instalación y eliminación de otros paquetes en el sistema. Una distribución moderna también suele incluir herramientas para la gestión de sistemas de archivos, creación y administración de cuentas de usuario, administración de redes, navegadores web, procesadores de texto, entre otros. (Silberschatz, 2017, 776)

Ubuntu

Distribución de Linux creada y distribuida por la empresa Canonical. Ofrecen el sistema operativo de forma gratuita, pero cobran por el soporte comercial del producto. Ampliamente usada por organizaciones en su versión de servidor, y comúnmente utilizada como distribución de entrada para usuarios de escritorio.. (Neil, 2016, 6)

Consola Bash

Consola predeterminada utilizada en muchas distribuciones de linux. Desarrollada por el proyecto GNU como un reemplazo de la shell estándar de Unix, conocida como Bourne shell (en honor a su creador). El nombre "Bash" es un juego de palabras que significa "Bourne again shell" (shell de Bourne renacida). (Blum, 2022, 17)

Script Shell

Archivo ejecutable mediante el cual en linux, permite colocar múltiples comandos ejecutables desde la línea de comandos. Conocido como script de shell, ya que agrupa comandos ejecutables por la consola.

Los archivos de script de shell son archivos de texto plano. Para crear un archivo de script de shell, solo se necesita usar cualquier editor de texto. (Blum, 2022, 141)

Cron

Herramienta utilizada por el kernel de linux para la programación de tareas de forma regular. El programa cron se ejecuta en segundo plano y revisa unas tablas especiales, llamadas cron tables (tablas de cron), para encontrar las tareas programadas que deben ejecutarse. (Blum, 2022, 226)

Crontab

Son las tablas de referencia pertenecientes a una distribución Linux así como a sus usuarios separadamente, para la programación de tareas. El formato para la programación de tareas es la siguiente:

minutos horas díadelmes mes díadelasemana comando

Permitiendo especificar las entradas como valores específicos, rangos de valores (como 1–5) o mediante un carácter comodín (el asterisco). (Blum, 2022, 227)

Desarrollo multiplataforma

Ventaja o modo de crear aplicaciones de software que utilizando una misma base de código permiten tras su compilación ejecutarse en múltiples sistemas operativos o plataformas (como Android, iOS, Windows, macOS, Linux, etc.). (Ambedkar, 2024, 3)

Lenguaje de programación

Sistema estructurado que se basa en principios matemáticos y lógicos, cuyo diseño tiene el fin de expresar algoritmos y cálculos matemáticos de forma comprensible para humanos. Este puede interactuar con el hardware y es clasificado con base a sus diversos paradigmas. (Yinong Chen, 2016, 2)

Lenguaje de programación compilado

Lenguajes en los cuales, su código fuente se traduce a instrucciones máquina usando un programa, denominado compilador. El compilador genera un programa objeto, que básicamente consiste en instrucciones máquina y referencias a otro código. (Urquía Moraleda et al., 2011, 68)

Lenguaje de programación interpretado

Lenguajes cuyo código base es analizado, leyendo y ejecutando sentencia a sentencia de código fuente, análogamente a como el ordenador ejecuta las mismas sentencias en código máquina. (Urquía Moraleda et al., 2011, 67)

Patrones de diseño

Solución recurrente a un problema de código dentro de un contexto específico. Altamente abstractos para mantener la facilidad de implementación ante diversas situaciones. Se destacan por contar con 4 características principales:

- Nombre
- Problema que propicia su aplicación
- Solución

- Consecuencias que implica aplicarlo (Gamma et al., 1995, 3,4)

Arquitectura Limpia

La arquitectura limpia es una modalidad de diseño de software que prioriza las preocupaciones separándolas en secciones, que se conocen más apropiadamente como "capas".

El principal objetivo de la arquitectura limpia es asegurarse de que todas las preocupaciones presentadas hayan sido atendidas y priorizadas al separarlas en capas distintivas, una encima de la otra. (Lewis, 2020, 27)

Arquitectura hexagonal

La idea, a grandes rasgos, se basa en la construcción de sistemas de software basados en una serie de puertos (interfaces de acceso público) y adaptadores (implementaciones de esas interfaces para un contexto específico) que se comunican con el núcleo de la aplicación. Allí es donde reside toda la lógica de negocio.

Esto busca implementar la lógica de negocio (el núcleo) de manera aislada e independiente de cualquier agente externo. Los puertos y adaptadores son los puntos de entrada, salida y conversión de datos. (Lewis, 2020, 18)

Inyección de dependencias

Patrón de diseño estructural que delega a un ensamblador la responsabilidad de proporcionar las implementaciones necesarias para sus dependencias, apegándose a la interfaz o contrato requerido. (Fowler, 2004)

Sistemas de control de versiones

Sistema el cual registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de manera que se pueda consultar diferentes versiones a través del tiempo de una manera segura y eficiente. (Chacon & Straub, 2014, 27)

Git

Sistema de control de versiones creado por Linux Torvalds, altamente utilizado en la actualidad y considerado como un estándar en el área del desarrollo y las TI, surgió tras la revocación del acceso gratuito a BitKeeper. Actualmente es capaz de gestionar proyectos como el kernel de linux y destaca su característica de sistema de ramas.(Chacon & Straub, 2014, 31)

Github

Propiedad de Microsoft y considerado como el principal alojamiento de repositorios Git y punto central de colaboración para millones de desarrolladores y proyectos. Cuenta con características gratuitas o limitadas a una capa de prueba y brinda una versión empresarial muy utilizada por las organizaciones.. (Chacon & Straub, 2014, 193)

Bases de datos

Colección estructurada de datos, que representa objetos, relaciones y restricciones del mundo real. Adaptable a diversos casos y cuya estructura depende del sistema de gestión de base de datos. (Celma Giménez et al., 2003, 2).

Sistema de gestión de bases de datos

Un sistema de gestión de bases de datos (SGBD) es una herramienta de software (conjunto de programas) que permite la creación y manipulación de bases de datos. (Celma Giménez et al., 2003, 2)

Modelo relacional

Propuesto por E.F Codd en 1970, se estableció como estándar desde los años 80, y actualmente es el más utilizado en el ámbito comercial. Su estructura básica es la relación, que representa los objetos y sus relaciones mediante los atributos inherentes a su dominio. (Celma Giménez et al., 2003, 36, 37)

MySQL

Sistema de gestión de bases de datos SQL Open Source más popular, desarrolla, distribuye y soporta MySQL AB. MySQL AB es una compañía comercial, fundada por los desarrolladores de MySQL. Es una compañía Open Source de segunda generación que une los valores y metodología Open Source con un exitoso modelo de negocio. (Oracle, 2014)

ORM

Un mapeador objeto-relacional (ORM) es una biblioteca de código que automatiza la transferencia de datos almacenados en tablas de bases de datos relacionales a objetos que se usan más comúnmente en el código de las aplicaciones. (Zimáyi et al., 2018)

Lenguaje de programación GO

Go es un lenguaje de propósito general diseñado con un enfoque en la programación de sistemas. Es fuertemente tipado, cuenta con recolección de basura y tiene soporte explícito para la programación concurrente. Los programas se construyen a partir de paquetes, cuyas propiedades permiten una gestión eficiente de las dependencias. (go.dev, 2024)

Lenguaje de programación JavaScript

Lenguaje de programación de alto nivel, dinámico e interpretado, que se adapta bien a los estilos de programación orientada a objetos y funcional. (Flanagan, 2011, 1)

Framework de aplicación web

Un Framework de Aplicación Web o simplemente "framework web" es un marco de software diseñado para apoyar el desarrollo de aplicaciones web, incluyendo servicios web, recursos web y APIs web. Los frameworks son, en resumen, bibliotecas que te ayudan a desarrollar tu aplicación de manera más rápida e inteligente. (Zimáyi et al., 2018)

Librería

En programación de computadoras, una librería se refiere a una colección de archivos, programas, rutinas, scripts o funciones que pueden ser referenciados en el código de programación. (Computer Hope, 2024)

Gin

Gin es un framework web desarrollado en Golang. Ofrece una API similar a Martini, pero con un rendimiento hasta 40 veces más rápido que Martini. Si buscas alto rendimiento y productividad, Gin es una excelente opción. (Gin Team, n.d.)

Wails

Framework que “Proporciona la capacidad de envolver tanto el código Go como un frontend web en un solo binario. Se proporcionan herramientas para facilitar esto, manejando la creación del proyecto, la compilación y el empaquetado” (Lea, n.d.)

Bootstrap

Bootstrap inicialmente creado por Twitter es un conjunto de código CSS y JavaScript que permite acelerar el desarrollo web de una manera estandarizada mediante el uso de clases estándares y medidas basadas en el sistema Grid de CSS, comúnmente relacionado con JQuery. (Matsinopoulos, 2020, 4)

React.js

Librería de JavaScript diseñada para la creación de interfaces modernas facilitando el manejo de estado y almacenamiento local, además de que su principal fortaleza es la creación de componentes reutilizables e independientes.

Cuenta con un enfoque declarativo, mediante el cual los desarrolladores describen el estado deseado de la interfaz de usuario y React se encarga de actualizar la interfaz real para que coincida con ese estado. Esto simplifica enormemente el proceso de desarrollo y mejora la legibilidad del código. (Poornima & Kavita, 2023, 1)

Antecedentes

En los últimos años, el desarrollo de tecnologías para controlar y monitorear sistemas biotecnológicos, como biorreactores y sistemas de cultivo, ha tomado mucha importancia, especialmente en el ámbito académico y de investigación. Sin embargo, hay un problema grande: los equipos comerciales son costosos y suelen ser complicados de usar. Esto ha hecho que muchas universidades y centros de investigación, sobre todo los que no tienen tantos recursos, no puedan acceder a estas herramientas. Por eso, han surgido varias alternativas que buscan solucionar esto, usando software y hardware de código abierto, así como metodologías de desarrollo que permiten abaratar costos y hacer que estos sistemas sean más eficientes.

A continuación, se presentan diversas investigaciones que ejemplifican el uso de biorreactores en el ámbito comercial. En este marco, el Prototipo de software para un biorreactor de tanque agitado habla sobre un prototipo de software creado para controlar y monitorear un biorreactor de tanque agitado en la Universidad EIA. Este proyecto surgió porque los biorreactores comerciales son muy costosos. El software que desarrollaron permite controlar cosas clave como la temperatura, el pH, la velocidad de agitación y la concentración de oxígeno, que son importantes para los cultivos celulares. Para llevar a cabo el control de biorreactores, siguieron una metodología de cuatro fases: primero recopilaron información, posteriormente realizaron un diseño arquitectónico, después lo implementaron y finalmente lo desplegaron. Usaron tecnologías como Java, JavaFX, PostgreSQL y Amazon Web Services (AWS), y aplicaron patrones de diseño como Singleton y Modelo-Vista-Controlador (MVC) para que el sistema fuera modular y fácil de mantener. Además, le agregaron funcionalidades como la gestión de alarmas, el almacenamiento de datos y el control en tiempo real, lo que hace que el monitoreo del biorreactor sea seguro y eficiente (Londoño Londoño, 2023).

Por otro lado, Open Source Software to Control Bioflo Bioreactors presenta un software de código abierto hecho en LabVIEW para controlar biorreactores de la familia Bioflo. Este sistema usa una arquitectura llamada QSM-PC (Queued State Machine-Producer Consumer), que permite ejecutar varios procesos al mismo tiempo sin que uno bloquee al otro. El software se comunica con el control primario de los biorreactores

usando un protocolo llamado AFS (Advanced Function Set) y tiene funciones para registrar datos, ejecutar protocolos personalizados y controlar parámetros como la temperatura, el pH y el oxígeno disuelto. También permite integrar hardware adicional mediante plugins y controlar varios biorreactores desde una sola interfaz, lo que hace que sea más escalable y eficiente en el uso de recursos (Burdge & Libourel, 2014)

Por último, la automatización de un sistema de inmersión temporal con base en plataformas abiertas de hardware y software, habla de un sistema de inmersión temporal (SIT) para micropropagación de plantas, que usa hardware y software de código abierto. Este sistema es una alternativa a los equipos comerciales, que son de alto costo y complicados, en el que automatizan el proceso de aplicación de nutrientes y el monitoreo constante, usando sensores, actuadores y placas Arduino para controlar el ambiente de cultivo. El software lo desarrollaron en Java, con herramientas como NetBeans IDE y Arduino IDE, y también hicieron una app móvil en Android para controlar el sistema desde cualquier lugar. Usaron una base de datos SQLite para guardar información sobre las condiciones del cultivo, lo que ayuda a tomar decisiones basadas en datos. Probando el prototipo en un cultivo de lináloe, los resultados fueron buenos: obtuvieron 131 brotes a partir de 10 explantes en solo un mes, algo que supera por mucho lo que se logra con sistemas tradicionales (Lugo Espinosa *et al.*, 2017, 271)

Problemática

Actualmente, encontrar una técnica biotecnológica que facilite la micropropagación de plántulas de importancia económica representa un gran desafío. Aunque existen diversos sistemas que favorecen el crecimiento y desarrollo de las plantas, muchos de ellos son costosos y de alta complejidad. Por esta razón, se desarrolló un software para biorreactores, diseñado para apoyar la micropropagación de plantas de alta calidad de manera más accesible y eficiente.

El sistema previo presentaba limitaciones significativas debido a su dependencia de Arduino como plataforma central. Aunque ampliamente utilizado en proyectos de automatización, su arquitectura rígida y su interacción basada en controladores físicos restringían la flexibilidad del sistema. Cualquier ajuste operativo requería modificaciones directas en el código fuente y su carga en el dispositivo, un proceso técnico complejo que dependía de personal especializado. Esta falta de accesibilidad ralentizaba la optimización y dificultaba la adaptación a nuevas necesidades.

Además, el sistema no contaba con mecanismos de persistencia de información, lo que significaba que todos los parámetros y configuraciones se perdían tras un reinicio o apagado. Esto obligaba a los usuarios a reconfigurar manualmente cada variable en cada uso, incrementando la posibilidad de errores y afectando la continuidad operativa. La ausencia de un almacenamiento interno o remoto impedía el análisis histórico de datos, lo que limitaba la capacidad de optimización y diagnóstico basado en el rendimiento pasado.

Sumado a esto, la falta de una API o mecanismos de comunicación con otras plataformas restringía la integración del sistema con herramientas externas. El monitoreo y la configuración solo podían realizarse mediante interacción directa con el hardware, lo que hacía inviable la supervisión remota. En conjunto, estas deficiencias afectan la confiabilidad del sistema, volviéndolo poco práctico y costoso de mantener a largo plazo.

Justificación

La creciente demanda de cultivos de importancia económica impulsa el diseño de diversas técnicas de cultivo, como los métodos hidropónicos, aeropónico y en sustrato sólido, entre otros. No obstante, el cultivo *in vitro* presenta múltiples ventajas, especialmente en el ámbito de la biotecnología y la agricultura. Algunas de las principales ventajas incluyen:

1. Control total del ambiente permitiendo regular factores clave como temperatura, luz, humedad y la composición del medio de cultivo.
2. Facilita la propagación rápida de plantas o tejidos, garantizando obtener un gran número de clones genéticamente idénticos en un periodo corto, ideal para cultivos comerciales.
3. Al realizarse en condiciones asépticas y controladas, se eliminan los riesgos asociados a patógenos, plagas o enfermedades que podrían afectar el desarrollo del cultivo.
4. Es posible inducir mutaciones, estudiar el comportamiento genético o realizar modificaciones genéticas en un entorno controlado antes de introducir las plantas en el campo.
5. Requiere menos agua y nutrientes en comparación con los cultivos tradicionales, garantizando la optimización de recursos.
6. Es escalable, una vez optimizado el proceso, puede ampliarse fácilmente para producir grandes cantidades de material biológico con características uniformes.
7. No depende de las estaciones o condiciones climáticas externas, lo que permite una producción continua durante todo el año.

Es por eso que los biorreactores de inmersión temporal son herramientas fundamentales en la producción de plantas *in vitro*, ya que permiten tener un mejor control de las condiciones ambientales, mayor eficiencia, producción a gran escala, mejor oxigenación, menor contaminación, mayor asepsia, ciclos optimizados y mayor productividad uniforme.

Sin embargo, las deficiencias detectadas en el sistema previo, como la inestabilidad de la aireación, acumulación de presión, errores en el intercambio de medio de cultivo y

un sistema limitante, comprometen la eficiencia, la confiabilidad y el rendimiento general del proceso.

Es crucial optimizar y mejorar estos sistemas para garantizar condiciones óptimas para el crecimiento de las plantas, reducir riesgos operativos y minimizar el impacto negativo en términos de tiempo, recurso humano y pérdida de material biológico. Al abordar las problemáticas actuales y desarrollar un control automatizado modular, se podrá no solo resolver los desafíos técnicos, sino también facilitar la operación del sistema mediante una interfaz más intuitiva y funcional.

Además, la implementación de un enfoque de mejora continua y escalabilidad permitirá que el sistema se adapte a futuras necesidades tecnológicas, aumentando su flexibilidad y utilidad en distintos escenarios de producción.

Esto asegura no solo la solución a las deficiencias detectadas, sino también un avance sostenible y escalable que contribuirá al fortalecimiento de los procesos de producción de cultivos *in vitro* por medio de biorreactores de inmersión temporal, impulsando la eficiencia operativa, mejorando la calidad del producto final y fomentando la innovación en los medios de cultivo.

Objetivos

General

Desarrollar un software basado en tecnologías open source para el control de biorreactores de inmersión temporal (SIT).

Específicos

- Diseñar e implementar una arquitectura de software robusta y modular que permita la integración con diferentes tipos de hardware para el control de los biorreactores.
- Desarrollar una interfaz de usuario amigable que no requiera de intervención técnica o especializada.
- Implementar controles versátiles para la regulación automática de los parámetros clave, como tiempos de inmersión, control de fotoperiodo y ciclos de intercambio de medio de cultivo.
- Optimizar la interacción entre el software y el hardware mediante protocolos de comunicación eficientes, asegurando la confiabilidad del sistema y la mínima latencia en la transmisión de datos.

Hipótesis

La implementación de un software modular y open source basado en Wails (Go + React) para el control de biorreactores de inmersión temporal (SIT), optimizará la gestión de estos sistemas, mejorando su eficiencia, escalabilidad y facilidad de uso.

Materiales y métodos

Evaluación inicial de la solución existente

Evaluar el funcionamiento del sistema de inmersión temporal que se tenía en el cuarto de incubación, identificar limitaciones y establecer soluciones de mejora del sistema

Hardware

Al principio del desarrollo del proyecto se consideró utilizar Arduino y conexiones físicas, pero de una manera más organizada e independiente, lo cual, por común acuerdo se consideró como una opción no escalable, dados los requerimientos reales en los procesos de los usuarios finales.

Por ende, la definición de la arquitectura se realizó considerando los siguientes puntos principales:

- No se necesita tener conocimientos especializados en programación para interactuar con el sistema.
- La interacción con el sistema debe ser intuitiva.
- Debe haber independencia en las interacciones (inmersión y fotoperiodo)
- La solución debe escalar horizontalmente para el control de N número de líneas de biorreactores.

El primer punto marca en gran medida la elección del hardware, ya que pese a las ventajas de las diferentes placas de Arduino, estas mismas terminan teniendo limitantes que no permitirían por ejemplo; alojar aplicaciones complejas con una UI basada en componentes web, al mismo tiempo que establecer conexiones a base de datos y controlar módulos con sensores. Por lo que se optó subir de rango dentro de las placas de desarrollo disponibles, seleccionando la placa de desarrollo Orange Pi en su quinta versión.

Esta placa brinda notables ventajas, siendo una de las principales la posibilidad de correr una distribución de linux basada en arquitectura ARM, aunado a las mejoras en cantidad de almacenamiento y memoria RAM.

Además, se seleccionó la placa ESP-32 S3 para la modularización de las líneas de los biorreactores, esta elección, más allá de subir el costo brindará los siguientes beneficios:

- Evitar un punto único de fallo.
- Simplificar y reducir las distancias entre los módulos y sus sensores.
- Reducir los tiempos de mantenimiento

Lista final de hardware para la realización del proyecto

- Centro de control
 - Placa Orange PI 5
 - Pantalla touch de 7 Pulgadas
 - Antena WI-FI USB
- Módulos de control de línea de biorreactor
 - Placa ESP-32 S3
 - Módulo de 4 relays

Tecnologías

Al contar con una placa con las capacidades de la Orange PI 5, los lenguajes de programación ya eran más a preferencia del equipo, pero se consideraron solo lenguajes de alto nivel para la aplicación de control, considerando el código Arduino/C para la interacción con los sensores en la placa ESP-32 S3, de los módulos de control.

Para la selección final se consideraron lenguajes tales como Java, Python o Rust, pero se optó por la combinación de Golang y JavaScript, unidos mediante el framework de aplicaciones multiplataforma Wails.

Wails permite el desarrollo de aplicaciones usando Golang como tecnología del lado del servidor y la elección libre de cualquier framework/librería de JavaScript para el desarrollo de la aplicación de lado del cliente.

Wails entra en la categoría de frameworks de aplicaciones multiplataforma como su competidor Electron, el cual permite un desarrollo parecido a Wails pero usando JavaScript como único lenguaje de programación. Se eligió Wails como tecnología final

debido a que tiene mejor desempeño y las aplicaciones tienen un peso final menor a las realizadas con Electron,

Para la realización de la interfaz de usuario se utilizó la librería de JavaScript React Js en su versión 18 mientras que del lado del servidor Golang en su versión 1.21 y como base de datos MySQL 8.0.

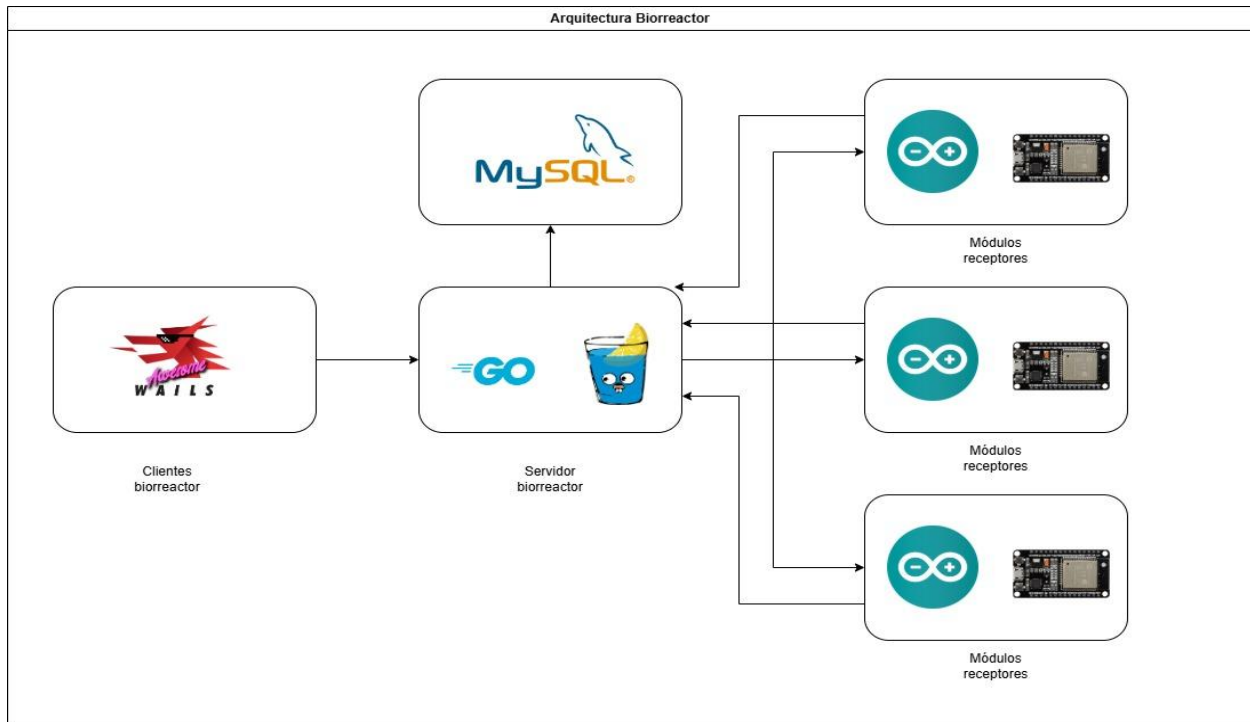


Figura 1. Arquitectura de la solución del sistema de control de biorreactores

Implementación

Diagrama de sistema

En la *Figura 2* se puede observar el sistema, sus componentes y las conexiones del biorreactor, el cual cuenta con una alimentación principal de 120V. Incluye componentes como una computadora Orange Pi con una pantalla LED de 7 pulgadas.

En el sistema de control se encuentran los ESP-32 y los relés, que a su vez controlan las electroválvulas. Estas se alimentan de un compresor y están conectadas a un manifold, encargado de distribuir el flujo de aire de manera uniforme entre los recipientes.

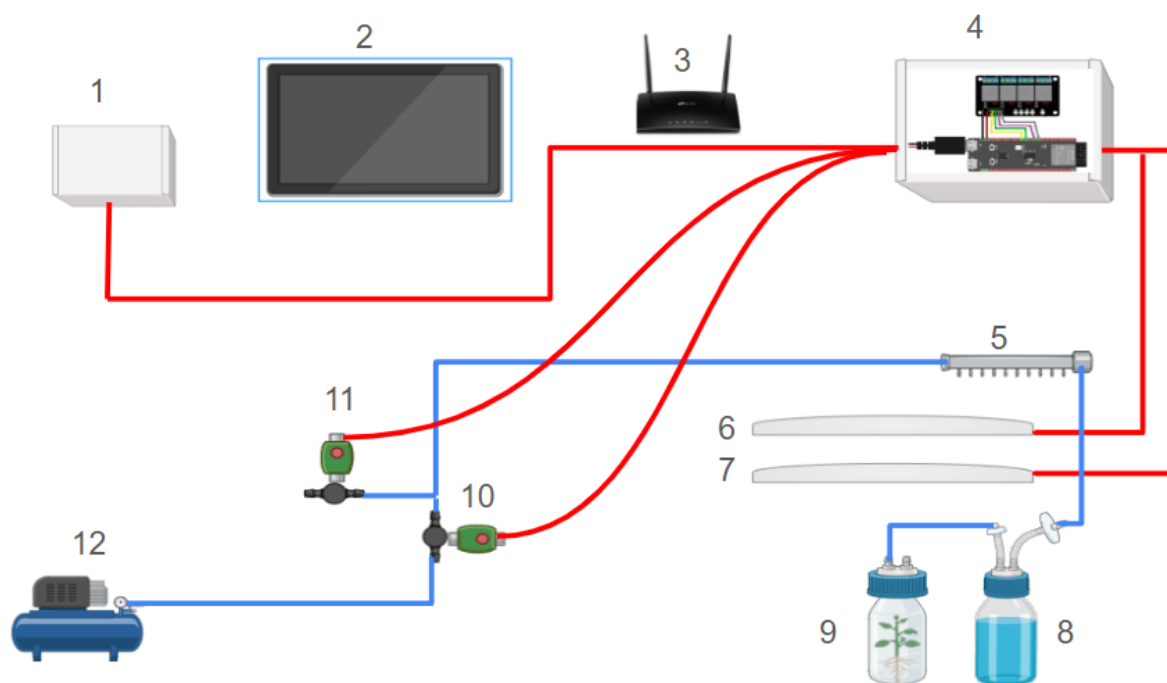


Figura 2. Diagrama del sistema

(1)Alimentación principal.(2)Computadora.(3)Router.(4)SEP-32 y Relé.(5)Manifold.(6)Lámparas led luz fria.(7)Lamparas led color(8)Recipiente con los nutrientes.(9)Recipiente con la planta.(10)Válvula de inyección.(11)Válvula de alivio.(12)Compresor

Programación de los módulos

Preparación del entorno

Para la programación del ESP32-S3, se requiere de ciertas herramientas de software y hardware:

- ESP-IDF: Framework oficial de Expressif para desarrollo de aplicaciones complejas en ESP32. Incluye soporte para C/C++ y características avanzadas.
- Arduino Core: Se usó en este código, ya que resulta ser más amigable y permite usar funciones simplificadas con librerías como *WiFi.h* y *ArduinoJson.h*.
- Esptool: Herramienta de línea de comandos para depurar y flashear el firmware al ESP32.
- Visual Studio Code: Se utiliza como IDE junto con la extensión ESP-IDF.
- ESP32-S3 Dev Kit: Placa de desarrollo.
- Conexión USB-C: Se usa para conectar la placa de desarrollo para cargar el firmware y monitorear el puerto serial.

Librerías usadas

WiFi.h

- Proporciona funciones para la conexión WiFi.
- En el código:
 - *WiFi.config()*: Configura una IP estática para el ESP32.
 - *WiFi.begin(ssid, password)*: Conecta el ESP32 a la red WiFi especificada.
 - *WiFi.localIP()*: Obtiene la dirección IP asignada.

ArduinoJson.h

- Usada para crear y manejar respuestas JSON fácilmente.

WiFi Server

- Nos permitió crear un servidor HTTP que escucha en el puerto 1337.
- En el código:
 - *server.begin()*: Inicial el servidor.
 - *server.available()*: Verifica si un cliente ha enviado una solicitud.

Flujo del código

Conexión WiFi con IP estática

- El ESP32 intenta conectarse a una red WiFi usando las credenciales proporcionadas.
- Se configura una dirección IP estática para garantizar que el dispositivo mantenga siempre la misma IP.
- Si no logra conectarse, el código imprime mensajes por el puerto serial hasta que la conexión sea exitosa.

Control de relés

- Se configuraron los pines de los relés (relay1Pin, relay2Pin, relay3Pin, relay4Pin) como salidas.
- Al iniciar el programa, se desactivan todos los relés enviando un estado lógico alto (HIGH), lo que hace que se apaguen.

Servidor HTTP

- El servidor se inicializa para escuchar solicitudes HTTP en el puerto 1337.
- Se programaron varios endpoints:
 - */health*: Responde con “Healthy” para verificar el estado del dispositivo.
 - */data*: Devuelve la dirección MAC e IP del ESP32 en formato JSON.
 - */lightInteraction*: Permite encender o pagar un relé específico para las luces blancas y de color, según los parámetros enviados.
 - */startAction*: Controla los intervalos de activación y desactivación del sistema de aireación.

Gestión de las solicitudes

- Se programó la función `getQueryParam()`, que extrae los parámetros de las solicitudes GET que recibe el ESP32 para poder determinar las acciones a realizar.
- El programa envía respuestas HTTP al cliente con códigos de estado (200 para éxito, 400 para error o 404 en caso de que el endpoint no exista).

Compilación y flasheo

Para la compilación del programa se utiliza la extensión ESP-IDF en Visual Studio Code, seleccionando la opción Build o con el comando `idf.py build:`

Para el flasheo del ESP32-S3:

- Se selecciona la placa en este caso es la ESP32-S3.
- Se conecta la placa al equipo de computo.
- Se selecciona el puerto COM en el que se encuentra conectada la placa.
- Se selecciona el método de flasheo en nuestro caso es UART.
- Se selecciona la opción de flasheo y monitoreo o con el comando *idf.py flash*.

Resultados y discusión

Evaluación inicial de la solución existente

Al momento de iniciar el proyecto se evaluó el funcionamiento real del sistema de inmersión existente, tras el cual se identificaron las siguientes limitaciones de los controles y su unidad de “procesamiento”.

- No se contaba con documentación ni código base del sistema.
- Pese a que la solución implementa una placa Arduino UNO, toda la ejecución dependía de un componente “añadir nombre” de Steren que se accionaba desde una aplicación móvil.
- El Arduino UNO limitaba la cantidad de módulos los cuales podían ser implementados debido a la cantidad de pines con la que cuenta.
- Los intervalos de inmersión no podían ser menores a 1 minuto.
- La configuración de las inmersiones era poco intuitiva.

Planificación de la solución

Una vez quedaron claras las limitaciones del sistema anterior se decidió separar esfuerzos dentro del equipo, siguiendo cada quien su especialidad y utilizando la metodología ágil Kanban. Se comenzó con la creación de tareas siguiendo tres ejes principales:

- Desarrollo y conexión de los sistemas lógicos, con las interfaces de software.
- Desarrollo de la aplicación de control de inmersión y fotosíntesis.

Desde un enfoque de ingeniería de software, la primera etapa de evaluación del proyecto nos ayudó a recabar los requisitos de los usuarios, dejando claras las reglas de negocio para la solución final.

El desarrollo del proyecto se previó para completarse en 1 año, teniendo sesiones semanales los días sábado de manera presencial. Por lo que siguiendo los principios de las metodologías ágiles, se acordaron entregables que fueran funcionales desde el primer momento, para ir iterando hacia la mejora de la solución.

Integración de la red local

- Se instaló un router WiFi *Figura 3* dentro del laboratorio donde opera el sistema biorreactor, garantizando que toda la red de dispositivos ESP-32 y el Orange Pi estén conectados a una red local estable.



Figura 3. Router

- Se asignaron direcciones IP de manera estática en cada controlador ESP-32 asegurando que siempre tengan la misma dirección IP en la red y evitar problemas de conectividad.
- Se previó la posibilidad de agregar más módulos en el futuro. Por lo tanto, la red local se diseñó para que fuera escalable, lo que permite agregar dispositivos sin necesidad de cambios significativos en la estructura de la red.

Desarrollo de la aplicación

Dadas las tecnologías elegidas, se determinó el uso de una de las consideradas arquitecturas limpias, la arquitectura hexagonal, en la cual se prioriza la separación de responsabilidades entre la lógica de negocio y cada una de las otras capas, separándolas independientemente usando interfaces e inyección de dependencias catalogados como patrones de diseño estructurales, esta misma se puede observar en la *Figura 4*.

Para la creación de la base de datos se implementó el ORM llamado GORM, el cual nos permitió generar la base de datos definiendo las entidades de la aplicación las cuales generan automáticamente las migraciones que se traducen a código SQL y se ejecutan en el servidor de base de datos.

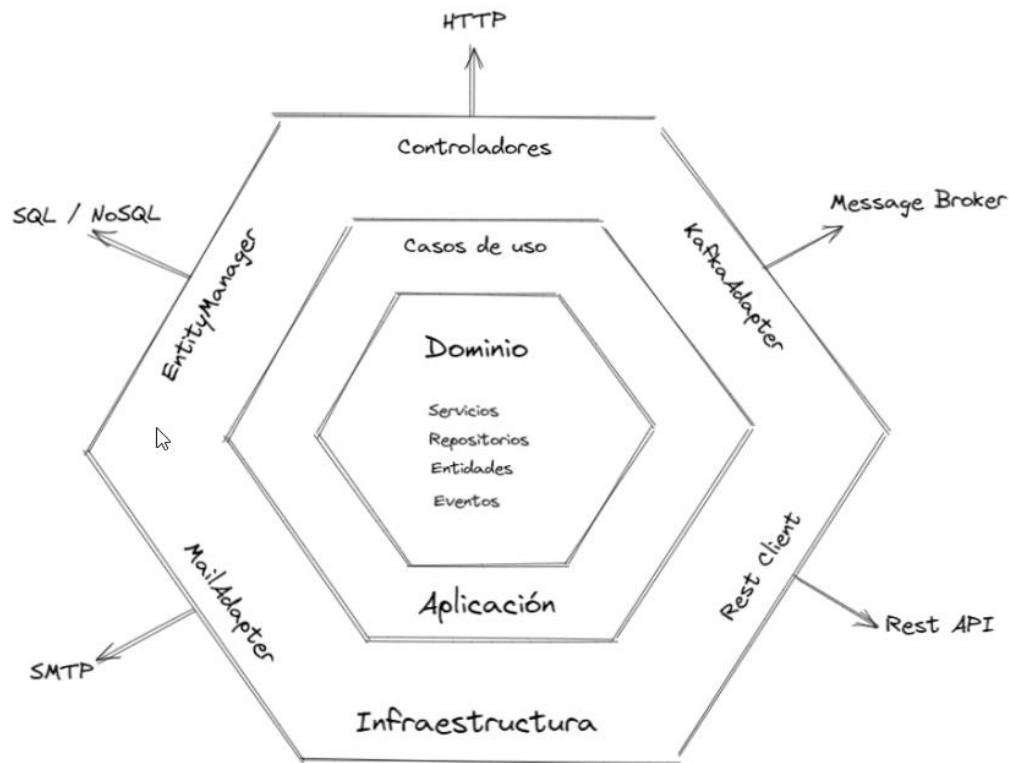


Figura 4. Arquitectura Hexagonal Créditos (Padilla Delgado, 2021)

Diseño de la base de datos

La base de datos se realizó utilizando el motor de base de datos MySQL. Esta consta de 8 tablas entre las cuales se contemplan los principales dominios: usuarios, módulos, sensores, configuraciones y ejecuciones.

El presente diseño se determinó debido al escalamiento modular que se tendrá permitiendo en un futuro, agregar más cantidad de módulos, completamente independientes y configurables. Pese a que el enfoque del desarrollo completo fue principalmente la portabilidad, se reafirmó la elección de MySQL como una solución más robusta que puede ser aprovechada para futuras implementaciones, tales como sincronización con bases de datos en la nube, o réplicas de información en infraestructura en sitio.

- Sensor: Tabla que representa los sensores y el pin en formato entero al que va asignado dentro de la placa (en desuso versión anterior).
- Module: Tabla que guarda la información de los módulos que controlan cada línea de biorreactores, es importante destacar el atributo tipo de módulo ya que no

descarta la utilización de módulos locales, los cuales hacen uso de los pines de la placa principal de llegar a ser necesario. Aunque el principal de caso de uso se apoya de los datos de la placa, tales como la dirección MAC y el código interno asignado para llevar el control mediante un inventario cuando llegue a ser necesario, así como la ip que será asignada para la comunicación por protocolo HTTP.

- **SensorType:** Tabla de catálogo para los tipos de sensores, la cual nos permite hacer la especificación de la relación entre un módulo y los sensores que implementa.
- **ModulePin:** Representa una relación más precisa entre el objeto módulo y el pin al cual está conectado un tipo de sensor.
- **Activity:** Tabla que registra la ejecución de intervalos de fotoperiodo e inmersiones.
- **User:** Tabla que registra al usuario y su código de acceso.
- **Configuration:** Tabla que registra las configuraciones para la ejecución programada de intervalos de fotoperiodo e inmersiones, los atributos más destacables son:
 - Intervalos de ejecución
 - Intervalo de inmersión
 - Expresión Cron
 - Ruta al script
 - Tipo (fotoperiodo, inmersión)
- **Execution:** Tabla que guarda los datos de ejecución de intervalos, (en desuso versión anterior).

El siguiente diccionario de datos muestra a detalle los campos presentes en las entidades del sistema, teniendo como formato su nombre y el tipo de dato, en algunos casos las restricciones que tienen ciertos tipos.:

Sensor

id-uint64
name-string(varchar(50))
pin - int (integer)

Module

id-uint64
uuid-string(char(36))
name-string(varchar(50))
description-string(varchar(25))
code-string(varchar(6))
output-int(integer)
is_remote-bool(boolean)
ip-string(varchar(15), nullable)
mac - string (varchar(17), nullable)

SensorType

id-uint64
name - string (varchar(50))

ModulePin

id-uint64
module_id-uint64
sensor_type_id-uint64
type-string(varchar(50))
pin - int (integer)

Activity

id-uint64
output-int(integer)
module_id-uint64
user_id-uint64
configuration_id-uint64
is_scheduled-bool(boolean)

User

id-uint64
name-string(varchar(255))
code-string (varchar(6))

Configuration

id-uint64
module_id-uint64

configuration_uuid-string(varchar(36),unique)
user_id-uint64
duration-int(integer)
interval-int(integer)
is_active-bool(boolean)
deleted-bool(boolean,default:false)
is_scheduled-bool(boolean)
execution_start_date-time.Time(date,nullable)
execution_end_date-time.Time(date,nullable)
cron_expression-string(varchar(255),nullable)
script_path-string(varchar(255),nullable)
type-string (enum:'nutritional_period', 'photo_period', default: 'nutritional_period')

Execution

id - uint64

configuration_uuid-string(varchar(36),unique)
configuration_id-uint64(nullable)
success - bool (boolean, default: true)

Diagrama entidad relación

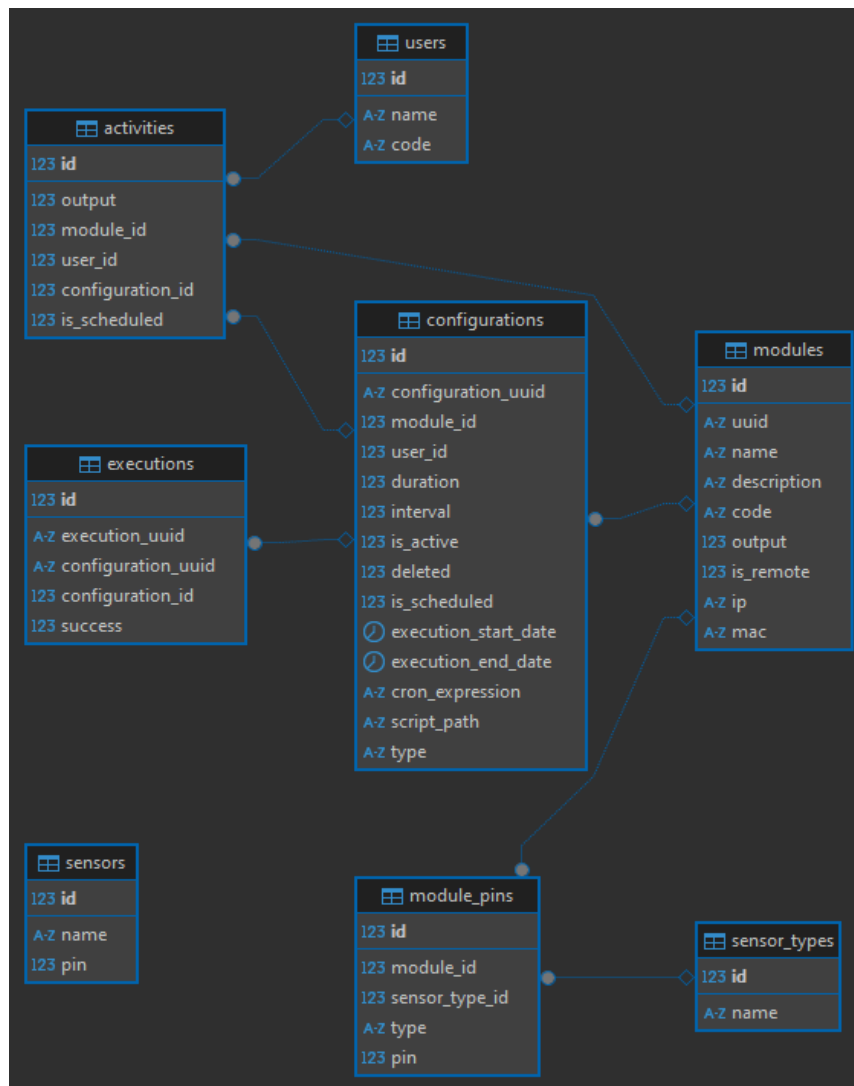


Figura 5. Diagrama entidad relación creado automáticamente con el software DBeaver

Funcionamiento de un proyecto de Wails

Durante la creación de un proyecto de Wails es importante resaltar dos etapas principales que describen a fondo su funcionamiento.

Renderizado de componentes web en una aplicación nativa de escritorio

Wails utiliza diferentes motores de renderizado web dependiendo de los sistemas operativos (Windows, macOS y Linux), lo cual permite ejecutar el código JavaScript con los estilos CSS y el maquetado de HTML en una ventana nativa de escritorio. (Lea, n.d.)

Comunicación entre la interfaz del cliente y la lógica del lado del “servidor”

La comunicación que establece entre los componentes renderizados por el motor, se realiza mediante llamadas directas a funciones entre JavaScript y Golang, construyendo este puente mediante una función tipada escrita en TypeScript que encapsula los atributos para solo hacer referencia a llamada, no dependiendo de una llamada a un servidor HTTP. (Lea, n.d.)

Configuración de un entorno de Wails (Entorno Ubuntu)

Ejecutar los siguientes comandos en la terminal

Instalación de Golang

```
# Descargar la última versión de Go
wget https://golang.org/dl/go1.21.2.linux-amd64.tar.gz

# Extraer el archivo descargado
sudo tar -C /usr/local -xvzf go1.21.2.linux-amd64.tar.gz

# Agregar Go al PATH
echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.profile
source ~/.profile

# Verificar la instalación
go version
```

Instalación de Node.js y npm

```
# Agregar el repositorio de Node.js
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -

# Instalar Node.js y npm
sudo apt install -y nodejs

# Verificar la instalación
node -v
npm -v
```

Instalación de dependencias adicionales

```
sudo apt install -y cmake git libwebkit2gtk-4.0-dev
```

Instalación y verificación de Wails

Instalar Wails usando Go

```
go install github.com/wailsapp/wails/v2/cmd/wails@latest
```

Verificar la instalación

```
wails doctor
```

Creación de un proyecto de wails

Crear un nuevo proyecto

```
wails init -n MyApp
```

Navegar al directorio del proyecto

```
cd MyApp
```

Instalar las dependencias del frontend

```
npm install
```

Compilar el proyecto

```
wails dev
```

Organización de un proyecto de Wails

Por defecto un proyecto de Wails se constituye de la siguiente estructura:

```
MyApp/
├── go.mod
├── go.sum
├── wails.json
├── backend/
│   ├── main.go
│   ├── app/
│   │   ├── app.go
│   │   └── handlers.go
│   ├── assets/
│   ├── models/
│   │   └── user.go
├── frontend/
│   ├── package.json
│   ├── src/
│   │   ├── App.vue
│   │   ├── main.js
│   │   ├── components/
│   │   └── assets/
│   ├── public/
│   └── dist/
├── build/
│   └── wails_build.json
├── node_modules/
└── .gitignore
```

Pero se optó por implementar la siguiente distribución como lo muestra la *Figura 6*, en el que se mueve la lógica del backend a la carpeta internal, para dentro de esta aplicar los principios de la arquitectura hexagonal. Respecto al frontend y los componentes web estos se mantuvieron en su carpeta llamada frontend frontend y mantiene la distribución por defecto. Además de que se agregaron unos cuantos archivos extras tales como el archivo .env que contiene las variables de entorno y más importante el punto de entrada en el archivo main.go

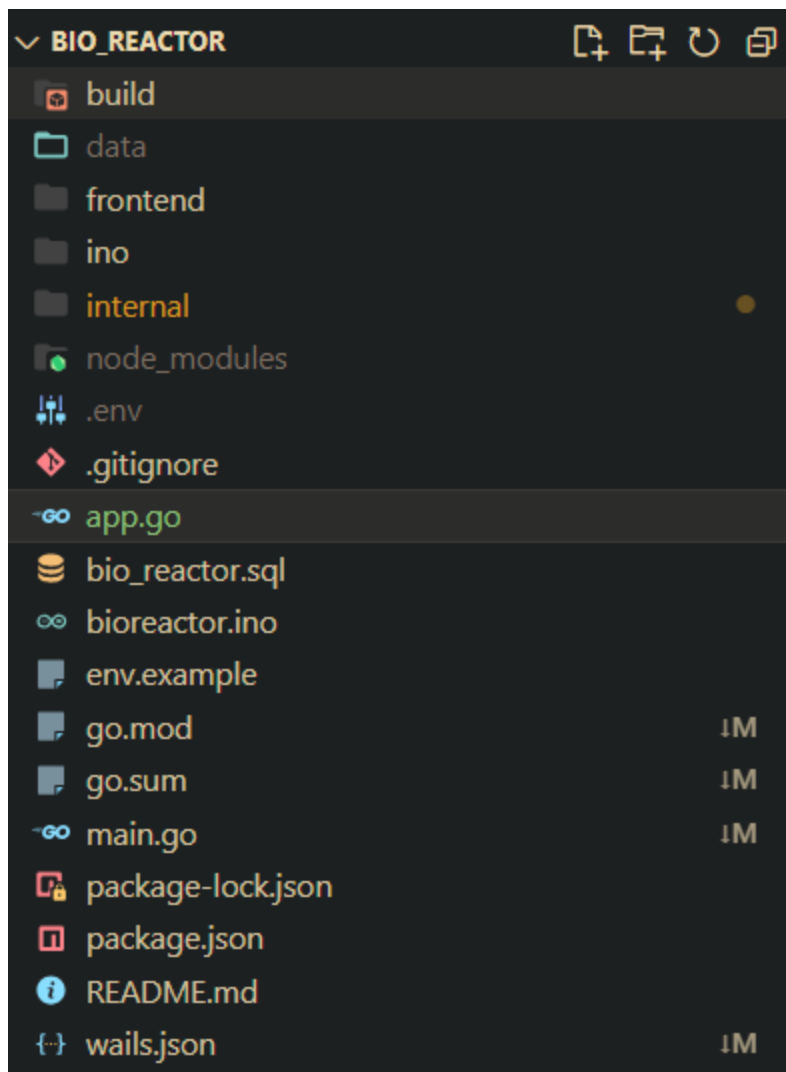


Figura 6. Distribución actual del sistema

Desarrollo del backend

Punto de Entrada del proyecto

Lo primero que se optó por realizar fue la programación de la lógica de negocio del lado del servidor, para tener toda la interacción con base de datos y los casos de uso de los cuales estos dependen. Para comenzar, en un proyecto de Wails el punto de entrada de la aplicación es el archivo main.go, el cual se definen las configuraciones de la ventana, el directorio final de donde se consumirá la aplicación web compilada y la sección más importante que es la de Bind en la cual se agregarán las aplicaciones cuyas

funciones se transpilarán a código TypeScript para estar disponible del lado del cliente, como se puede observar en la *Figura 7*.

```
main.go > main
48 func main() {
49     defer config.CloseDatabaseConnection(db)
50     // Load environment variables from the .env file
51     err := godotenv.Load()
52     if err != nil {
53         log.Fatal("Error cargando el archivo .env")
54     }
55     // Create an instance of the app structure
56     app := NewApp()
57     // Create application with options
58     err = wails.Run(&options.App{
59         Title: "reactor",
60         Width: 1024,
61         Height: 768,
62         AssetServer: &assetserver.Options{
63             Assets: assets,
64         },
65         BackgroundColour: &options.RGBA{R: 27, G: 38, B: 54, A: 1},
66         OnStartup: app.startup,
67         Bind: []interface{}{
68             app,
69             userApp,
70             moduleApp,
71             executionApp,
72         },
73     })
74 }
75
76 if err != nil {
77     println("Error:", err.Error())
78     os.Exit(1)
79 }
80 }
81
```

Figura 7. Punto de entrada de un proyecto de Wails

Instanciación de las aplicaciones mediante la inyección de sus dependencias

El otro apartado principal por el cual se aplica el patrón de arquitectura hexagonal justificadamente, es la instanciación de las aplicaciones que el atributo Bind termina importando, cada una de las aplicaciones sigue una estructura estandarizada: siendo la creación de una configuración única de acceso a la base de datos como dependencia inicial, que es inyectada a la capa de repositorio que simplifica la importación de la entidad que abarca, para posteriormente inyectar a la capa de servicio y esta a su vez inyectar a la capa de controlador, finalizando la creación de aplicación con todas sus dependencias. Este patrón de inyección de dependencias se puede observar en las *Figuras 8 y 9*.

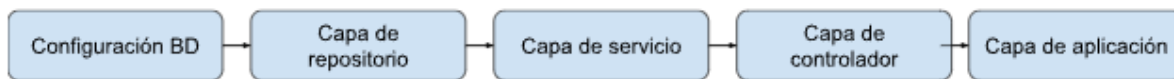


Figura 8. Ejemplo visual de la inyección de dependencias en el sistema

```

var (
    db *gorm.DB = config.SetupDatabaseConnection()
    userRepository repository.UserRepository = repository.NewUserRepository(db)
    userService service.UserService = service.NewUserService(userRepository)
    userController controller.UserController = controller.NewUserController(userService)
    // Configuration
    configurationRepository repository.ConfigurationRepository = repository.NewConfigurationRepository(db)
    configurationService service.ConfigurationService = service.NewConfigurationService(configurationRepository)
    configurationController controller.ConfigurationController = controller.NewConfigurationController(configurationService)
    // Execution
    executionRepository repository.ExecutionRepository = repository.NewExecutionRepository(db)
    executionService service.ExecutionService = service.NewExecutionService(executionRepository)
    executionController controller.ExecutionController = controller.NewExecutionController(executionService)
    // Activity
    activityRepository repository.ActivityRepository = repository.NewActivityRepository(db)
    activityService service.ActivityService = service.NewActivityService(activityRepository)
    // Module
    moduleRepository repository.ModuleRepository = repository.NewModuleRepository(db)
    moduleService service.ModuleService = service.NewModuleService(moduleRepository)
    esp32Client integration.NewClient("")
    moduleController controller.ModuleController = controller.NewModuleController(moduleService, activityRepository, executionService,
    userApp = apps.NewUserApp(userController)
    moduleApp = apps.NewModuleApp(moduleController)
    executionApp = apps.NewExecutionApp(moduleService, executionController, configurationController)
)
  
```

Figura 9. Ejemplo actual de la inyección de dependencias para la instancia final de la capa de aplicación.

Una vez mostrada la estructura y punto de entrada del proyecto se puede detallar como es que la capa de aplicación se genera siendo esta comparada “a manera de analogía” con lo que sería un enrutador en un sistema de lado del servidor.

Capa de aplicación

Para ejemplificar podemos mostrar la capa de aplicación del dominio de Ejecución cuyo código se puede observar en la *Figura 10*. Esta se define en primer lugar utilizando un struct en Golang, determinando en éste los componentes de los cuales depende, tales como el servicio y controladores necesarios. En seguida se crea su método constructor el cual se inicializan estas dependencias. Para cada uno de los métodos definidos en esta capa no es necesario crear interfaces ya que los métodos se transpilarán a su definición en TypeScript automáticamente por Wails.

Por último, pero no menos importante esta capa es fundamental para la obtención de datos provenientes del lado del cliente, por lo que cada uno de sus métodos funcionan como métodos de controlador en una arquitectura MVC, pero con la mejora que relegan

la lógica de negocio a la capa de controlador en el contexto de arquitectura hexagonal, para no mezclar la lógica de negocio con la capa de recepción HTTP, como se llega a realizar en ocasiones en la arquitectura MVC.

```
type ExecutionApp struct {
    ctx          context.Context
    moduleService service.ModuleService
    executionController controller.ExecutionController
    configurationController controller.ConfigurationController
}

// You, 7 months ago • feat: Add get executions go method
// NewExecutionApp crea una nueva instancia de ExecutionApp
func NewExecutionApp(moduleService service.ModuleService, executionController controller.ExecutionController, configurationController controller.ConfigurationController) *ExecutionApp {
    return &ExecutionApp{
        moduleService:    moduleService,
        executionController: executionController,
        configurationController: configurationController,
    }
}

// startup se llama cuando la aplicación se inicia. Se guarda el contexto
// para poder llamar a los métodos de runtime
func (a *ExecutionApp) startup(ctx context.Context) {
    a.ctx = ctx
}

func (a *ExecutionApp) GetExecutions() ([]entity.Execution, error) {
    var executions []entity.Execution
    executions, err := a.executionController.List()
    if err != nil {
        return executions, err
    }
    return executions, nil
}
```

Figura 10. Descripción de la estructura de la capa de aplicación de un proyecto de Wails usando el dominio Ejecución como ejemplo

Capa de controlador

La siguiente capa la del controlador como se muestra en la *Figura 11*, comienza a tener una estructura diferente, aparte de que se define con un struct y que cuenta con un método constructor, implementa interfaces, las cuales cumplen una función principalmente; servir como contrato entre la capa superior que la consume, independientemente de su implementación por lo que logramos el objetivo principal el cual es el desacoplamiento de las capas.

En la capa controlador del dominio ejecución, el método List realiza esencialmente una consulta a la base de datos para mostrar todas las ejecuciones existentes, por lo que para lograrlo; más allá de ejecutar lógica dentro del controlador en su lugar se llama a la capa de servicio.

Capa de servicio

```
1 package controller
2
3 import (
4     "bio_reactor/internal/entity"
5     "bio_reactor/internal/service"
6 )
7
8 You, 7 months ago | 1 author (You)
9 type ExecutionController interface {
10     List() ([]entity.Execution, error)
11 }
12 You, 7 months ago • feat: Add get executions go method
13
14 You, 7 months ago | 1 author (You)
15 type executionController struct {
16     executionService service.ExecutionService
17 }
18
19 func NewExecutionController(executionService service.ExecutionService) ExecutionController {
20     return &executionController{
21         executionService: executionService,
22     }
23 }
24
25 func (c *executionController) List() ([]entity.Execution, error) {
26     executions, err := c.executionService.List()
27     if err != nil {
28         // Manejar el error, podrías loguearlo o devolver un error personalizado.
29         return nil, err
30     }
31     return executions, nil
32 }
```

Figura 11. Estructura de la capa de controlador

La siguiente capa, de servicio, sigue utilizando la misma estructura de implementar interfaces, definir las dependencias mediante un struct y contar con un método constructor. Pero además se puede observar otro de los beneficios con casos comúnmente realistas, supongamos que en algún punto se cambia de base de datos, la capa siguiente que es la de repositorio podrá cambiar o recibir actualizaciones, esto más allá que parecer preocupante o añadir carga para refactorizar la capa de servicio, no debe de preocupar bajo esta arquitectura, ya que la capa de servicio solo llamará al método de la interfaz o llamado también contrato, esto se puede observar en la *Figura 12*.

```

1 package service
2
3 import (
4     "bio_reactor/internal/entity"
5     "bio_reactor/internal/repository"
6 )
7
8 You, 7 months ago | 1 author (You)
9 type ExecutionService interface {
10     CreateExecution(execution *entity.Execution) error
11     List() ([]entity.Execution, error)
12 }
13 You, 7 months ago * feat: Add get executions go method
14 You, 7 months ago | 1 author (You)
15 type executionService struct {
16     executionRepo repository.ExecutionRepository
17 }
18
19 func NewExecutionService(executionRepo repository.ExecutionRepository) ExecutionService {
20     return &executionService{
21         executionRepo: executionRepo,
22     }
23 }
24
25 func (service *executionService) CreateExecution(execution *entity.Execution) error {
26     return service.executionRepo.CreateExecution(execution)
27 }
28
29 func (service *executionService) List() ([]entity.Execution, error) {
30     executions, err := service.executionRepo.GetExecutions()
31     if err != nil {
32         return nil, err
33     }
34     return executions, nil
35 }

```

Figura 12. Estructura de la capa de servicio

Capa de repositorio

Para cerrar con el ciclo de arquitectura hexagonal implementando inyección de dependencias, llegamos al final pero prácticamente volvemos al principio, a la capa de repositorio, la cual depende únicamente de la conexión a la base de datos y utiliza a su vez la entidad correspondiente a su dominio, ya que es en esta en donde se realizan las operaciones en base de datos, como se puede observar en la *Figura 13*, mediante la conexión a la base de datos que tiene como parámetro, realiza operaciones mediante GORM, que para el método GetExecutions estaría realizando una consulta en MySQL tal como:

```
SELECT * FROM executions;
```

```

1 package repository
2
3 import (
4     "bio_reactor/internal/entity"
5
6     "gorm.io/gorm"
7 )
8
9 type ExecutionRepository interface {
10     CreateExecution(execution *entity.Execution) error
11     GetExecutions() ([]entity.Execution, error)
12 }
13
14 type executionRepository struct {
15     connection *gorm.DB
16 }
17
18 func NewExecutionRepository(db *gorm.DB) ExecutionRepository {
19     return &executionRepository{
20         connection: db,
21     }
22 }
23
24 func (r *executionRepository) CreateExecution(execution *entity.Execution) error {
25     return r.connection.Create(execution).Error
26 }
27
28 func (r *executionRepository) GetExecutions() ([]entity.Execution, error) {
29     var executions []entity.Execution
30     if err := r.connection.Find(&executions).Error; err != nil {
31         return nil, err
32     }
33     return executions, nil
34 }
35

```

Figura 13. Estructura de la capa de repositorio

Capa de integraciones

Para interactuar con los módulos ESP-32 dentro de la red local del laboratorio es necesario crear una integración con que mediante una interfaz provea los métodos que se mapean a peticiones HTTP. La implementación mostrada en la *Figura 14*, muestra cómo el método `StartManualExecution`, toma sus parámetros y los envía junto con el endpoint al que el módulo ESP-32 espera ser llamado para dicha interacción.

```

1 // integration/client.go
2 package integration
3
4 import (
5     "fmt"
6     "net/http"
7 )
8
9 // ESP32Client defines the methods for interacting with an ESP32 device.
10 type ESP32Client interface {
11     TurnLight(moduleID int, modulePin int, state int) (bool, error)
12     StartManualExecution(interval int) (bool, error)
13 }
14
15 // Client represents an HTTP client for interacting with ESP32 devices.
16 type Client struct {
17     BaseURL string
18 }
19
20 // NewClient creates a new ESP32 HTTP client with the specified base URL.
21 func NewClient(baseURL string) *Client {
22     return &Client{
23         BaseURL: baseURL,
24     }
25 }
26
27 func (c *Client) StartManualExecution(interval int, transferenceInterval int) (bool, error) {
28     endpoint := "/startAction"
29     params := map[string]interface{}{
30         "pumpInterval":    transferenceInterval,
31         "InmersionInterval": interval,
32     }
33     return c.sendRequest(endpoint, params)
34 }

```

Figura 14. Implementación del cliente HTTP para interactuar con los módulos ESP-32

Capa de api

La capa de API, implementada con **Gin** en Go, actúa como el punto de entrada para las solicitudes externas, permitiendo que múltiples clientes interactúen con el sistema del biorreactor de manera escalable. Siguiendo la **arquitectura hexagonal**, desacopla la lógica del negocio de la infraestructura, donde los controladores procesan las solicitudes, los servicios manejan la lógica y los repositorios gestionan la persistencia. Esto garantiza la reutilización del código y facilita futuras integraciones sin afectar el núcleo del sistema.

```

internal > api > server.go > main
1  package main
2
3  import (
4      "fmt"
5      "net/http"
6
7      "bio_reactor/internal/config"
8      "bio_reactor/internal/handlers"
9      "bio_reactor/internal/middleware"
10     "bio_reactor/internal/repository"
11     "bio_reactor/internal/service"
12
13     "github.com/gin-gonic/gin"
14     "gorm.io/gorm"
15 )
16
17 var (
18     apidb          *gorm.DB                = config.SetupDatabaseConnection()
19     executionRepository repository.ExecutionRepository = repository.NewExecutionRepository(apidb)
20     configurationRepository repository.ConfigurationRepository = repository.NewConfigurationRepository(apidb)
21     executionService service.ExecutionService = service.NewExecutionService(executionRepository)
22     configurationService service.ConfigurationService = service.NewConfigurationService(configurationRepository)
23     executionHandler *handlers.ExecutionHandler = handlers.NewExecutionHandler(executionService, configurationService)
24 )
25
26 func main() {
27     defer config.CloseDatabaseConnection(apidb)
28
29     // Configuración de Gin
30     r := gin.Default()
31
32     r.GET("/healthcheck", func(c *gin.Context) {
33         c.JSON(http.StatusOK, gin.H{
34             "status": "healthy",
35             "message": "Service is up and running",
36         })
37     })
38
39     r.Use(middleware.ValidateHeaders())
40
41     r.GET("/executions", executionHandler.List)
42     r.POST("/executions", executionHandler.Create)
43
44     // Iniciar el servidor
45     fmt.Println("Starting API server on port 8080...")
46     if err := r.Run(":8080"); err != nil {
47         fmt.Println("Failed to start API:", err)
48     }
49 }

```

Figura 15. Implementación del api del sistema del biorreactor utilizando gin

Capa de middleware

En este proyecto, el middleware actúa como una capa intermedia que intercepta las solicitudes HTTP antes de que lleguen a los controladores. Se utiliza para realizar validaciones,

autenticación, manipulación de cabeceras y otras funciones transversales necesarias para el correcto funcionamiento de la API.

```
internal > middleware > auth.go > ValidateHeaders
You, 2 hours ago | 1 author (You)
1 package middleware
2
3 import (
4     "net/http"
5     "os"
6
7     "github.com/gin-gonic/gin"
8 )
9
10 func ValidateHeaders() gin.HandlerFunc {
11     return func(c *gin.Context) {
12         authHeader := c.GetHeader("x-service-auth")
13         if authHeader == "" {
14             c.JSON(http.StatusUnauthorized, gin.H{"error": "Missing x-service-auth header"})
15             c.Abort()
16             return
17         }
18
19         expectedAuth := os.Getenv("X_SERVICE_AUTH")
20         if authHeader != expectedAuth {
21             c.JSON(http.StatusForbidden, gin.H{"error": "Invalid x-service-auth value"})
22             c.Abort()
23             return
24         }
25
26         c.Next()
27     }
28 }
```

Figura 16. Implementación del middleware de autorización basado en verificación de headers

Casos de uso generales

Dentro de los dominios principales, se pueden realizar operaciones CRUD y funciones específicas, para el resto como son los catálogos, simplemente se definen operaciones de lectura, ya que estas entidades (debido al alcance actual), se insertan mediante seeders.

Casos de uso principales

- Método de ejecuciones manuales para probar el correcto funcionamiento de los módulos y sus sensores (Implementa cliente ESP-32)
- Creación de la configuración para los intervalos de fotoperiodo e inmersión para cada módulo

- Creación del script en bash que disparará los llamados de ejecución a los módulos mediante una expresión cron. Cada uno de estos tanto para fotoperiodo e inmersión, utilizan una función base, la cual mapea los datos obtenidos desde la aplicación del cliente y con ellos establece las franjas de ejecución, hablando de la fecha inicial, horas de ejecución y las restricciones que cada caso pueda presentar. Para ejemplificar se muestra en las *figuras 17 y 18*, la función para el script de inmersión, la *figura 17* Muestra la declaración de la función con los atributos anteriormente mencionados, y mediante el uso del método Sprintf se crea el contenido del script a generar, la primera parte muestra la validación de rangos de fechas y posteriormente la de rangos de horas. Por otro lado, en la *figura 18* se muestra el resto de la función en donde se mapean los atributos para construir la petición HTTP que este caso se hará nativa desde el script de bash, una vez generado el contenido del script, se crea con este un archivo nombrado de acuerdo a un UUID versión 4 creado aleatoriamente, con la extensión sh.

```

133 func generatePath(ip string, pumpInterval int, immersionInterval int, transferenceTime int, startDate, endDate, startTime, endTime string) (string, error) {
134     // Obtener el directorio raíz actual
135     rootDir, err := os.Getwd()
136     if err != nil {
137         return "", err
138     }
139
140     // Generar un UUID aleatorio para el archivo
141     uuid := uuid.New().String()
142     fileName := fmt.Sprintf("%s.sh", uuid)
143     filePath := filepath.Join(rootDir, "data", fileName)
144
145     // Construir el script como texto con validación de fecha y hora
146     script := fmt.Sprintf(`#!/bin/bash
147
148     # Fecha y hora actuales
149     current_date=$(date +%Y-%m-%d)
150     current_time=$(date +%H:%M)
151
152     # Fechas y horas de inicio y fin
153     start_date="%s"
154     end_date="%s"
155     start_time="%s"
156     end_time="%s"
157
158     # Validar rango de fechas
159     if [[ "$current_date" < "$start_date" || "$current_date" > "$end_date" ]]; then
160         echo "La ejecución está fuera del rango de fechas permitidas."
161         exit 1
162     fi
163
164     # Validar rango de tiempo
165     if [[ "$current_time" < "$start_time" || "$current_time" > "$end_time" ]]; then
166         echo "La ejecución está fuera del rango de horas permitidas."
167         exit 1
168     fi

```

Figura 17. Primera parte del método de creación del script de ejecución de intervalo de inmersión

```

133 func generatePath(ip string, pumpInterval int, immersionInterval int, transferenceTime int, startDate, endDate, startTime, endTime string) (string, error) {
134
135     # Dirección IP y parámetros
136     ip="%s"
137     pumpInterval=%d
138     immersionInterval=%d
139     transferenceTime=%d
140
141     # Construir la URL
142     url="http://%s:1337/startAction?pumpInterval=%d&transferenceTime=%d&immersionInterval=%d"
143
144     # Hacer la petición GET usando curl
145     response=$(curl -s -o /dev/null -w "%{http_code}" "$url")
146
147     # Comprobar la respuesta HTTP
148     if [ "$response" = 200 ]; then
149         echo "La petición se ha realizado con éxito."
150     else
151         echo "Error: La petición ha fallado con código de estado HTTP $response."
152         exit 1
153     fi
154
155     echo "El proceso ha terminado con pumpInterval=%d, immersionInterval=%d y transferenceTime=%d."
156     , startDate, endDate, startTime, endTime, ip, pumpInterval, immersionInterval, transferenceTime
157
158     // Crear y escribir el archivo
159     err = os.WriteFile(filePath, []byte(script), 0755)
160     if err != nil {
161         return "", err
162     }
163
164     return filePath, nil
165 }

```

Figura 18. Segunda parte del método de creación del script de ejecución de intervalo de inmersión

Ejecución de trabajos programados

Uno de los argumentos de la función de generar configuración de ejecuciones, es la expresión cron, la cual nos permite insertarla en la crontab general del sistema, la cual se encargará de ejecutar las configuraciones generadas para que cada interacción tanto de fotoperiodo e inmersión correctamente en tiempo y en forma.

Interacción de los componentes del sistema

Después de haber implementado la lógica del servidor y estructurado los componentes según la arquitectura hexagonal, se logró reutilizar la base de código para exponer los métodos esenciales a través de una interfaz HTTP. Esto permitió que tanto los módulos internos como los clientes externos interactuaran de manera eficiente con la base de datos y la gestión de tareas programadas en la tabla de cronjob.

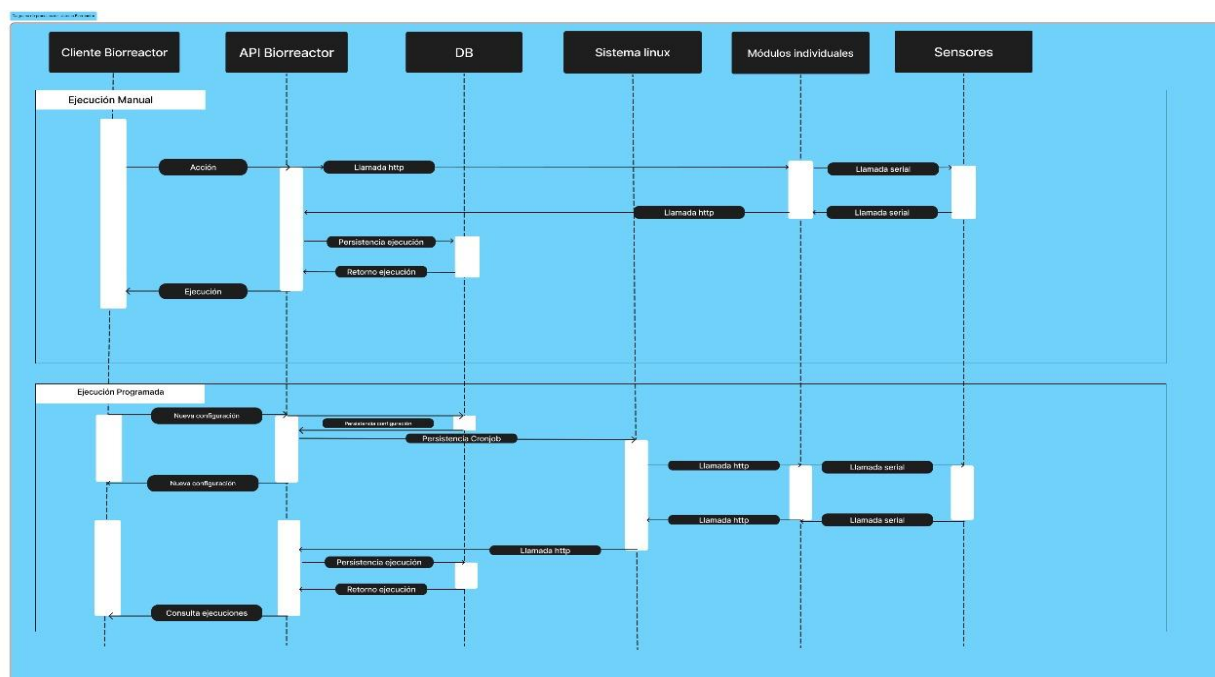


Figura 19. Diagrama de procesos

Desarrollo de la interfaz

Para la realización de la aplicación del lado del cliente, se eligió la librería de interfaces web React.js en su versión 18, así que tomando su estructura por defecto se implementó junto con la librería de estilos Bootstrap 5, para estandarizar la imagen de los componentes utilizados.

Rutas

Para la construcción de las rutas de la aplicación se implementó la librería react-router-dom, esta librería nos permite mediante su HashRouter definir la ruta interna al que en teoría la url del navegador accede para renderizar un componente, como se muestra en la *figura 20*, se observan 4 rutas principales:

- “/” O ruta base, la cual renderiza el panel de autenticación *figura 21*
- “/app” Renderiza la página de inicio en la que se muestra la barra de navegación superior, *figura 22*
- “/app/modules” Renderiza la página que lista los componentes, *figura 23*
- “app/modules/{id}” Renderiza el detalle del módulo desde el cual se puede realizar la interacción, *figura 24*

La estructura de las páginas se organiza de la siguiente manera: El enrutador relaciona una ruta con un componente de tipo página y esta importa su respectivo componente para que usando un layout o un componente base, se añaden componente estandarizados tales como la barra de navegación, lo cual nos permite modularizar el código correctamente y solo cambiar una sección específica de los componentes haciendo más eficiente y fluida esta aplicación de tipo SPA.

Páginas del sistema

```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import { Routes, Route, HashRouter } from 'react-router-dom';
4 import { HomeComponent } from '../components/HomeComponent';
5 import { HomePage } from '../pages/HomePage';
6 import { ModuleListPage } from '../pages/modules/ModuleListPage';
7 import { ModuleDetailPage } from '../pages/modules/ModuleDetailPage';
8
9 export const AppRouter = () => {
10   return (
11     ReactDOM.render(
12       <HashRouter basename="/">
13         <Routes>
14           <Route path="/" element={<HomeComponent />} exact />
15           <Route path="/app" element={<HomePage />} />
16           <Route path="/app/modules" element={<ModuleListPage />} />
17           <Route path="/app/modules/:id" element={<ModuleDetailPage />} />
18         </Routes>
19       </HashRouter>,
20       root
21     )
22   );
23 }
```

Figura 20. HashRouter de la aplicación del lado del cliente

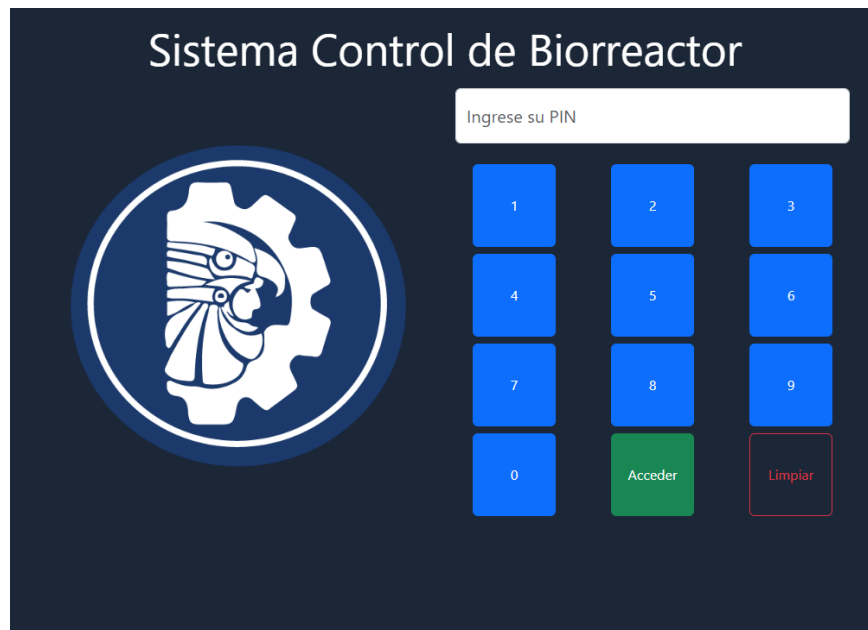


Figura 21. Ruta raíz

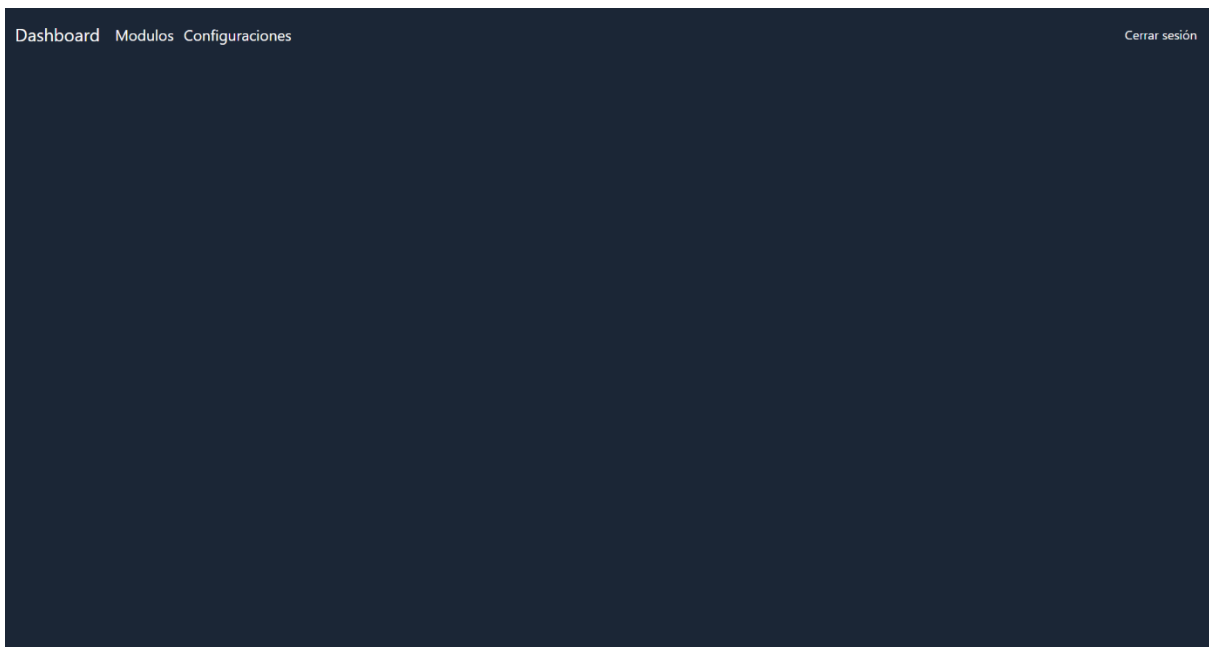


Figura 22. Página de inicio

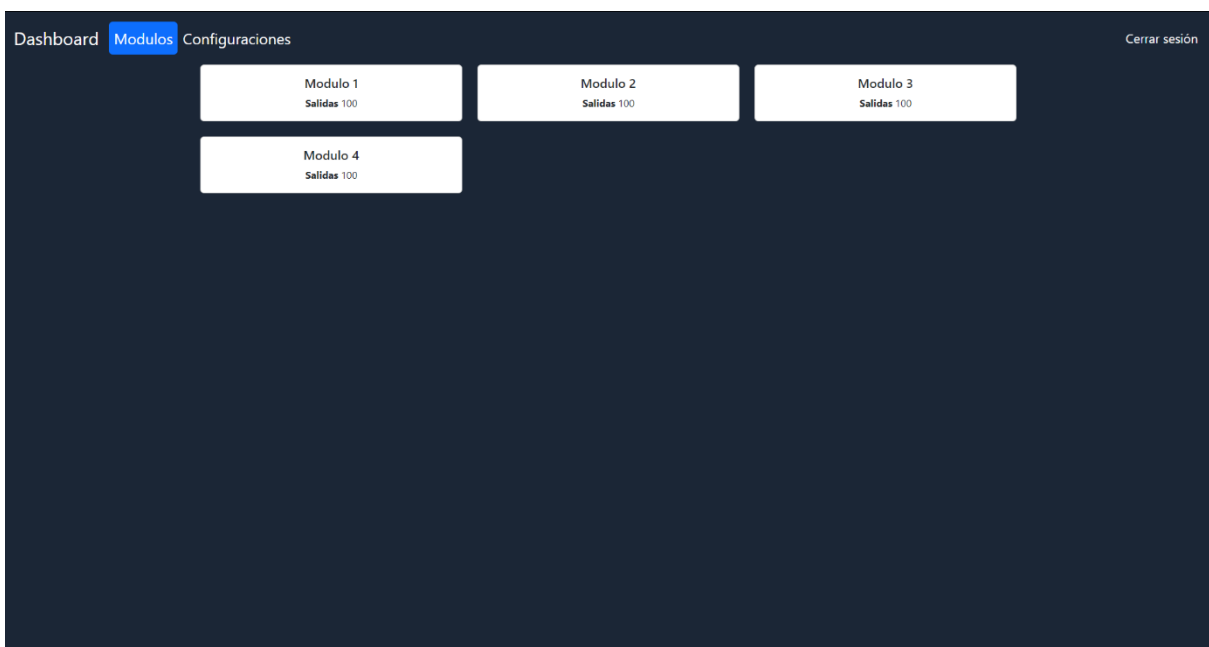


Figura 23. Página de listado de módulos

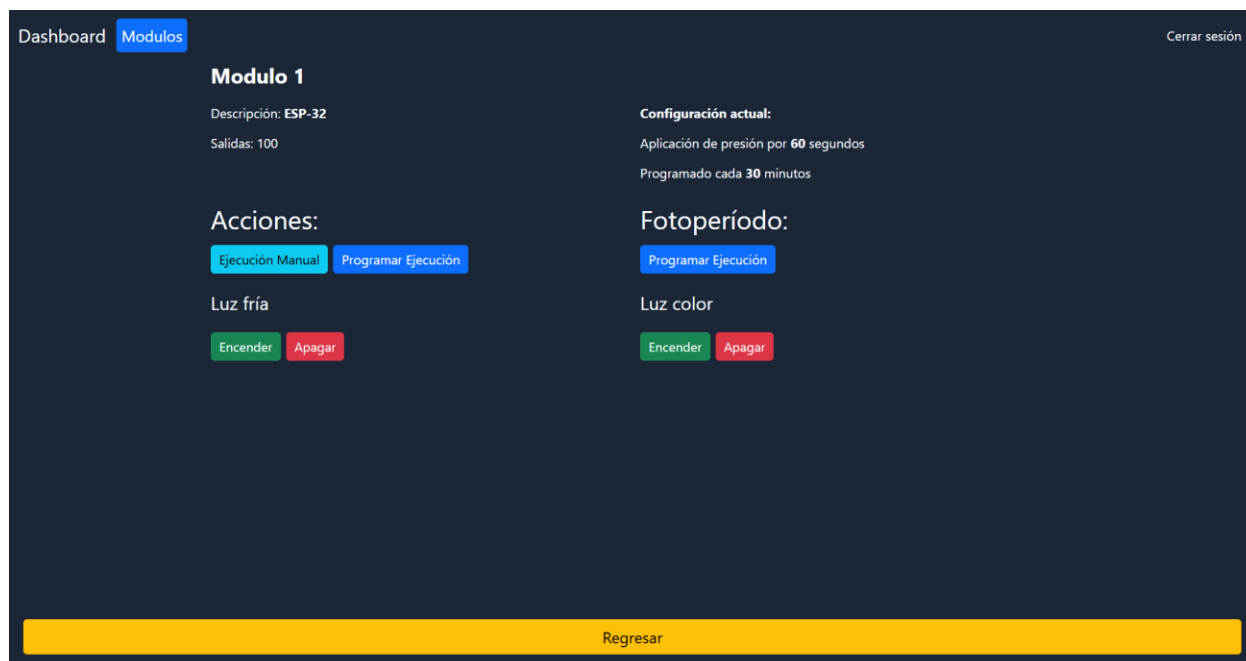


Figura 24. Página detalle de módulo

La figura 24 muestra el componente con el que más se interactúa, este cuenta con los detalles principales de cada módulo, tales como su nombre, su tipo y si cuenta con una programación actual, además de dos conjuntos de botones para interactuar directamente con las luces frías y cálidas.

Modales de control de inmersión manual

Dentro de los dos módulos principales, la ejecución manual de inmersión es fundamental para revisar que la solución esté trabajando como se debe o para cuando quieran realizar ejecuciones fuera de una programación establecida, por lo que se optó por realizar un modal simplificado con los parámetros mínimos para que interactúe con su respectiva acción, la figura 25 muestra cómo con dos selectores se consigue enviar los parámetros necesarios para la función de ejecución manual, la cual llega a la base de código de Golang y ejecuta su función homónima.

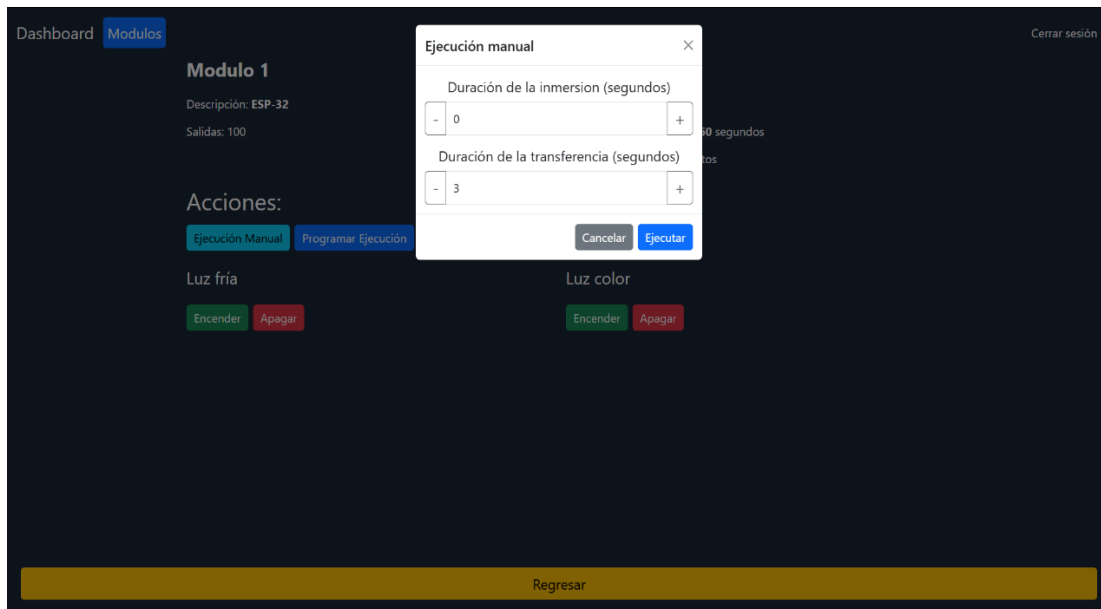


Figura 25. Modal de ejecución de inmersión manual

Modal de control de inmersión programada

Los componentes más importantes son sin duda los mostrados en las figuras 26 y 27, dichos modales nos permiten tomar todos los parámetros que sus respectivas funciones de programación de acción necesitan para interactuar tanto con la inmersión y fotoperiodo.

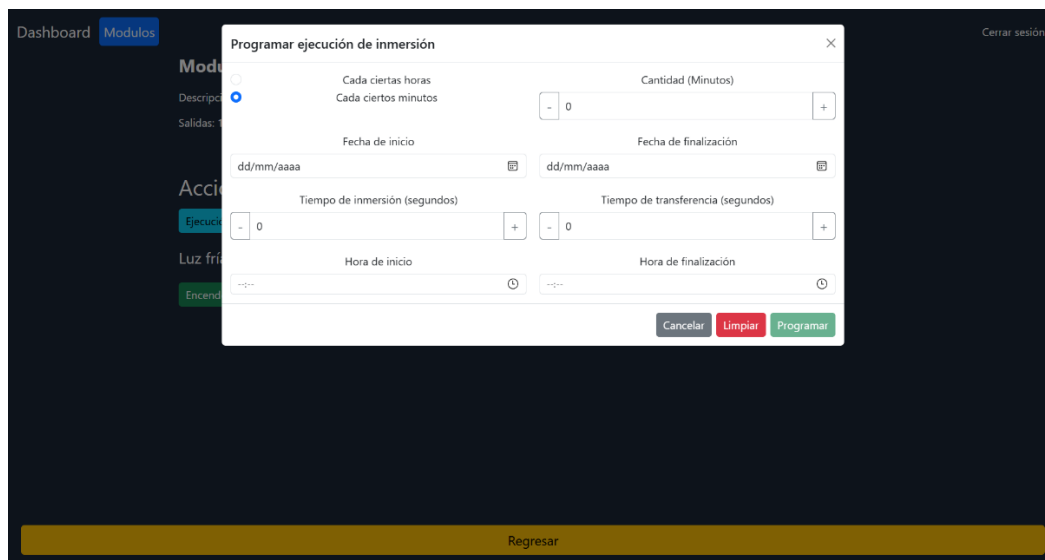


Figura 26. Modal de programación de intervalo de inmersión

Modal de control de fotoperiodo programado

The image shows a web application interface with a dark blue sidebar on the left containing navigation links: 'Dashboard', 'Modulos', 'Descripción', 'Salidas: 1', 'Acción', 'Ejecución', 'Luz fría', and 'Encendido'. The main content area is dark blue. A white modal titled 'Programación de fotoperiodo' is open in the center. The modal contains the following fields and controls:

- Horas de luz:** A numeric input field with a value of 0, flanked by minus and plus buttons.
- Horas de oscuridad:** A numeric input field with a value of 24.
- Seleccione el tipo de luz:** Two radio buttons. The first is selected and labeled 'Luz Fría (Pin: 25)'. The second is labeled 'Luz Color (Pin: 26)'.
- Hora de inicio:** A time selection field showing '--:--' with a clock icon.
- Fecha de inicio:** A date selection field showing 'dd/mm/aaaa' with a calendar icon.
- Fecha de finalización:** A date selection field showing 'dd/mm/aaaa' with a calendar icon.
- Buttons:** At the bottom right of the modal are three buttons: 'Cancelar' (grey), 'Limpiar' (red), and 'Programar' (green).

Below the modal, a yellow bar at the bottom of the page contains a 'Regresar' button.

Figura 27. Modal de programación de intervalo de fotoperiodo

Etapa de construcción del ejecutable

Una vez terminado el desarrollo de la aplicación es necesario compilarla a su versión ejecutable y nativa, para realizarlo es necesario cumplir con los siguientes pasos (para el sistema ubuntu), para macOS y Windows consultar la documentación oficial de Wails.

1. Abrir una terminal y situarse en la raíz del proyecto
2. Ejecutar el comando de compilación
 - a. `wails build -platform linux/amd64 -tags webkit2_41`
3. Verificar que la compilación se haya realizado correctamente, buscando el ejecutable en el directorio `/build`
4. Una vez confirmada la existencia del ejecutable se recomienda moverlo a la carpeta `bin` usando el siguiente comando
 - a. `sudo cp nombre-del-ejecutable /usr/local/bin/`
5. En seguida, tenemos que situarnos en el directorio compartido de las aplicaciones
 - a. `cd ~/.local/share/applications`
6. Para poder ejecutar el programa es necesario crear un archivo `.desktop`, usando el siguiente comando
 - a. `nano nombre-del-ejecutable.desktop`
7. Una vez dentro del editor se tienen que llenar los siguientes campos acorde a su aplicación, se puede añadir un ícono, habilitar la terminal entre otras configuraciones, para finalmente guardar el contenido del archivo

[Desktop Entry]

Name=Nombre de la Aplicación

Exec=/usr/local/bin/nombre-del-ejecutable

Icon=/ruta/a/un/icono.png

Type=Application

Terminal=false

Categories=Utility;

8. Por último, se le tiene que dar permisos de ejecución mediante el siguiente comando
 - a. `chmod +x nombre-del-ejecutable.desktop.`

Consumo de recursos

En los resultados obtenidos, se observó que tanto la API como el cliente del sistema de control del biorreactor consumen en promedio 25 MB de RAM cada uno, lo que refleja un uso eficiente de los recursos del sistema. El cliente, al ser construido como un archivo ejecutable, tiene un tamaño de aproximadamente 10 MB, lo que lo hace ligero y adecuado para su implementación. En cuanto a la velocidad de respuesta, la API muestra tiempos de respuesta en el orden de microsegundos, e incluso en algunos casos, llegando a tiempos de respuesta cercanos a los nanosegundos, lo que garantiza una interacción rápida y sin demoras notables entre los componentes del sistema.

Producto final

La distribución de los biorreactores, dividida por niveles, se muestra en la *figura 28*. Cada una de las conexiones proporcionará de forma programada o manual la presión necesaria para alimentar de solución nutritiva a la planta.



Figura 28. Línea de biorreactores con luz fría

Se ilustran los 2 tipos de lámparas (luz blanca, luz LED de espectro completo con 3 opciones de color violeta, rojo, violeta-rojizo) que pueden implementarse en cada módulo, conforme a las especificaciones de cada producto destinado a su uso en el biorreactor *figura 29*.

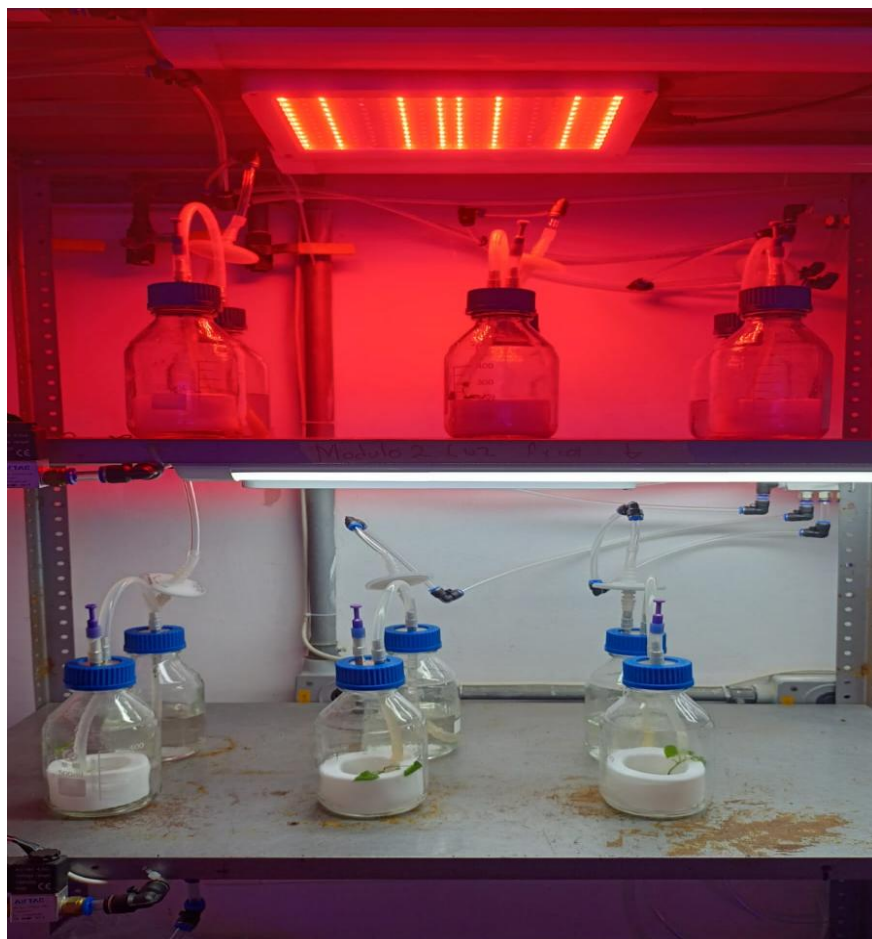


Figura 29. Líneas de biorreactores

Como se puede observar en la *figura 30*, el sistema de administración de biorreactores se encuentra instalado y ejecutándose en la placa de desarrollo OrangePi conectada mediante cable WIFI a una red escolar con acceso a internet a una red local junto con los módulos individuales que accionan las válvulas y encienden las luces.

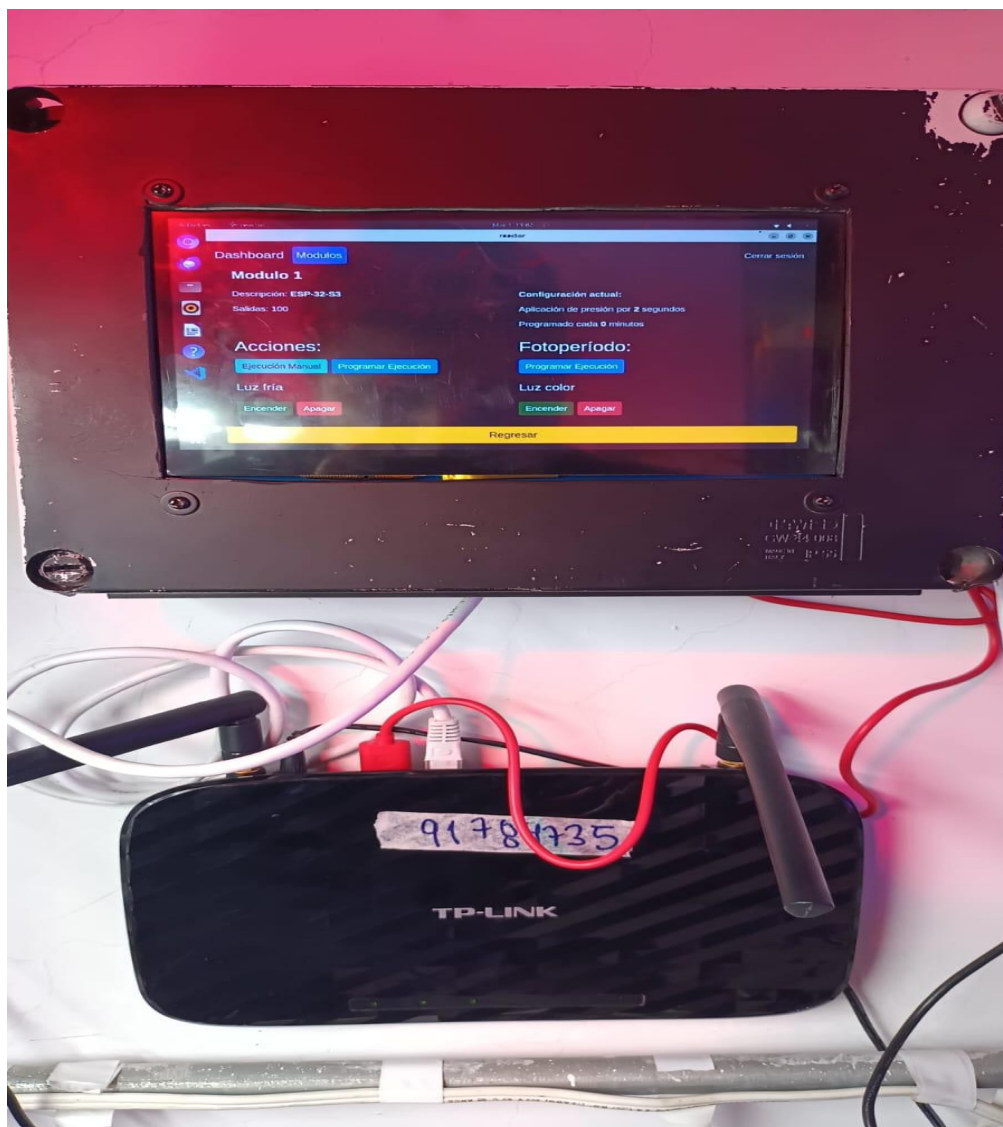


Figura 30. Sistema instalado en la placa de desarrollo

Conclusiones

Desarrollo de un software eficiente y modular

El presente trabajo ha dado lugar a una solución innovadora para el control de biorreactores de inmersión temporal (SIT), fundamentada en una arquitectura modular y escalable. Mediante el uso del stack Wails (Go + React), se ha desarrollado un sistema capaz de adaptarse a diversas configuraciones operativas, asegurando estabilidad y eficiencia a largo plazo. La implementación de una estructura modular facilita el mantenimiento del software y permite futuras mejoras sin afectar su operatividad.

Optimización de la interacción entre software y hardware

El diseño del sistema posibilita la integración con nuevos módulos y bases de datos locales, optimizando la comunicación con el hardware a través de protocolos eficientes que minimizan la latencia y garantizan la confiabilidad del sistema. Gracias a la descentralización de la gestión de sensores y la independencia de los módulos de control, el software reduce el riesgo de fallos al evitar la dependencia de una única unidad de procesamiento. Este enfoque incrementa la tolerancia a errores y facilita la escalabilidad, permitiendo la incorporación de nuevos dispositivos sin comprometer la estabilidad del sistema.

Interfaz de usuario intuitiva y accesible

Con el objetivo de mejorar la experiencia del usuario, se diseñó una interfaz gráfica intuitiva basada en React, la cual simplifica la interacción con el sistema y facilita la configuración y monitoreo del biorreactor. Este diseño ha reducido significativamente la curva de aprendizaje para los operadores, minimizando errores humanos y optimizando la eficiencia en la gestión del sistema.

Automatización y optimización de procesos

La automatización de los procesos dentro del biorreactor ha sido un elemento clave en el desarrollo del software. La implementación de mecanismos de control automático permite regular con precisión parámetros críticos, como los tiempos de inmersión y el flujo de aireación, lo que contribuye a mejorar la estabilidad y calidad de los cultivos in vitro. Asimismo, la reducción del riesgo de fallos operativos asegura un desempeño más eficiente y confiable del sistema en entornos de producción biotecnológica.

Flexibilidad y capacidad de expansión

La adopción de estándares abiertos para la integración de sensores y actuadores de diversos fabricantes ha dotado al sistema de una gran flexibilidad. Esta característica permite su implementación en distintos entornos sin requerir modificaciones significativas, ampliando así su aplicabilidad y potencial de expansión. Gracias a estas capacidades, el software no solo responde a las necesidades actuales del control de biorreactores, sino que también sienta las bases para futuras mejoras y desarrollos, consolidándose como una solución robusta y adaptable dentro del campo de la biotecnología.

Referencias bibliográficas

- Ambedkar, B. (2024). *Cross Platform Application Development* (1st ed.). Dr. Babasaheb Ambedkar Open University, Ahmedabad. <https://baou.edu.in/assets/pdf/MCA-303%20Cross%20Platform%20Application%20Development.pdf>
- Arduino. (S.f). *Arduino Uno*. Arduino Uno Rev3. Retrieved 11 26, 2024, from <https://store.arduino.cc/products/arduino-uno-rev3>
- Blum, R. (2022). *Linux Fundamentals*. Jones & Bartlett Learning.
- Burdge, D., & Libourel, I. (2014). *Open Source Software to Control Bioflo Bioreactors* (Vol. 9). pLoS one. <https://doi.org/10.1371/journal.pone.0092108>
- Castillo, A. (2004). *Propagación de plantas por cultivo in vitro: una biotecnología que nos acompaña hace mucho tiempo*. Unidad de Biotecnología, INIA Las Brujas.
- Celma Giménez, M., Casamayor Ródenas, J. C., & Mota Herranz, L. (2003). *Bases de datos relacionales*. Pearson Educación.
- Chacon, S., & Straub, B. (2014). *Pro Git*. Apress.
- Chmiel, H., Takors, R., & Weuster-Botz, D. (Eds.). (2018). *Bioprozesstechnik*. Springer Berlin Heidelberg.
- Computer Hope. (2024, 09 19). *What's a Library*. Computer Hope. Retrieved 12 2, 2024, from <https://www.computerhope.com/jargon/l/library.htm>
- ESPRESSIF. (n.d.). *ESP32 Series Datasheet* (4.7). ESPRESSIF. Retrieved 11 26, 2024, from https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- Flanagan, D. (2011). *JavaScript: The Definitive Guide*. O'Reilly Media, Incorporated.
- Fowler, M. (2004, January 23). *Inversion of Control Containers and the Dependency Injection pattern*. Martin Fowler. Retrieved December 2, 2024, from <https://martinfowler.com/articles/injection.html>

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns*. Addison-Wesley.

Gin Team. (n.d.). *Gin Documentation*. Gin Web Framework. Retrieved 02 23, 2025, from <https://gin-gonic.com/>

go.dev. (2024, June 13). *The Go Programming Language Specification*. The Go Programming Language. Retrieved December 2, 2024, from <https://go.dev/ref/spec>

Herve, E., & M, B. (2002). Temporary immersion systems in plant micropropagation. *Plant Cell, Tissue and Organ Culture*, 69, 215–231.

Jiménez-Ruiz, E. F., Martínez-Solís, J., Mejía-Muñoz, J. M., Sosa-Montes, E., & Juárez-Hernández, M. d. J. (2023). EL FOTOPERIODO COMO MODULADOR DEL CRECIMIENTO EN *Dahlia campanulata* SAAR, P. D. SØRENSEN & HJERT. *Fitotec*, 46(1), 77-83. https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S0187-73802023000100077

Kumar, M., Muthusamy, A., Kumar, V., & Bhalla-Sarin, N. (Eds.). (2019). *In Vitro Plant Breeding Towards Novel Agronomic Traits: Biotic and Abiotic Stress Tolerance*. Springer Nature Singapore.

Lea, A. (n.d.). *Wails Documentation*. Retrieved 12 2, 2024, from <https://github.com/wailsapp/wails>

Lewis, E. (2020). *Clean Architecture: A Comprehensive Beginner's Guide to Learn the Realms of Clean Architecture from A-Z*. Independently Published.

Lewis, E. (2020). *Clean Architecture: Advanced and Effective Strategies Using Clean Architecture Principles*. Independently Published.

Littelfuse. (n.d.). *RELAYS* [Definition, Specifications and Applications]. Littelfuse. [https://www.mouser.com/pdfDocs/LF-Relays-](https://www.mouser.com/pdfDocs/LF-Relays-Whitepaper.pdf?srsId=AfmBOorb7FDXJEeliXGnXtDcE5Fgzl054CEKM4wn0cDZEJMQrCq5z_y)

[Whitepaper.pdf?srsId=AfmBOorb7FDXJEeliXGnXtDcE5Fgzl054CEKM4wn0cDZEJMQrCq5z_y](https://www.mouser.com/pdfDocs/LF-Relays-Whitepaper.pdf?srsId=AfmBOorb7FDXJEeliXGnXtDcE5Fgzl054CEKM4wn0cDZEJMQrCq5z_y)

Londoño Londoño, S. (2023). *Prototipo de software para un biorreactor de tanque agitado*. Universidad EIA. <https://repository.eia.edu.co/entities/publication/ffae63be-486b-4440-8034-4b45281bf8cd>

Lugo Espinosa, O., Arellano Ostoa, G., & Hernández Cote, D. (2017). Automatización de un sistema de inmersión temporal con base en plataformas abiertas de hardware y software. *Terra Latinoamericana*, 35(3), 269-277.

https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S0187-57792017000300269

Matsinopoulos, P. (2020). *Practical Bootstrap: Learn to Develop Responsively with One of the Most Popular CSS Frameworks*. Apress.

Neil, N. J. (2016). *Learning Ubuntu: A Beginners Guide to Using Linux*. CreateSpace Independent Publishing Platform.

Oracle. (2014, 05 24). *MySQL 5.0 Reference Manual*. Retrieved 12 2, 2024, from <https://downloads.mysql.com/docs/refman-5.0-es.pdf>

Orange Pi. (n.d.). *Orange Pi 5*. Orange Pi - Orangepi. Retrieved November 26, 2024, from <http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5.html>

Oviedo Bayas, B., Samaniego Mena, E., & Murillo Oviedo, J. (2018). *Fundamentos de redes* (Primera ed.). Ediciones Grupo Compás. https://www.researchgate.net/publication/358105030_Libro_de_Fundamentos_de_RED_ES

Padilla Delgado, R. (2021, 04 30). *Arquitectura Hexagonal*. Retrieved 12 03, 2024, from <https://raulpadilladelgado.github.io/blog/p/arquitectura-hexagonal/>

Palma, C., & Osorio, S. (2018). Tarjetas de Desarrollo: Herramientas para el diseño. *Letras ConCiencia Tecnológica*, 43-52.

Pérez-Urria Carril, E. (2009). Fotosíntesis: Aspectos Básicos. *Reduca (Biología)*, 2(3), 1-47.

PNEUMAX. (n.d.). *Válvulas Electroválvulas* [Información Técnica]. PNEUMAX.

Retrieved 11 26, 2024, from

<https://rodavigo.net/catalogos/PNEUMAX/01%20Introducci%C3%B3n/04%20Informaci%C3%B3n%20t%C3%A9cnica%20III.pdf>

Poornima, G. N., & Kavita, S. O. (2023). *Awesome React.js (Unleash the Power of Modern UI Building)* (Vol. 1). International Institute of Organized Research. 978-81-961331-4-6

Silberschatz. (2017). *Operating System Concepts 10E*. John Wiley & Sons, Incorporated.

Stallings, W. (2004). *Comunicaciones y redes de computadores* (J. E. Díaz Verdejo, Trans.). Pearson Educación.

Urquía Moraleda, A., Martín Villalba, C., & Rubio González, M. Á. (2011). *LENGUAJES DE PROGRAMACIÓN*. UNED.

Vishal Gupta, N., & Shukshith, K.S. (2016). Qualification of Autoclave. *International Journal of PharmTech Research*, 9(4), 220-226.

Yinong Chen. (2016). *Introduction to Programming Languages: Programming in C C++ Scheme Prolog C# and SOA*. Kendall Hunt Publishing Company.

Zimáyi, E., Jallow, B., & Kashef, S. (2018, 12 18). *Object Relational Mapping and Entity Framework* [Advanced Databases Project]. Retrieved 12 03, 2024, from https://cs.ulb.ac.be/public/_media/teaching/entityframework_2017.pdf