Informe de análisis
Compilatio Magister+ | ZEGEL-PE

TAP Moises_Roberto_Linconao

ID : 456c5608dd43a396c18d41acef466748c5589cbe



8%
Textos sospechosos

Nombre del fichero : TAP Moises_Roberto_Linconao.txt
Tamaño del archivo original : 7,89 MB
Número de palabras : 15.111

Depositante : Jerson Sanchez Ramos
Fecha de depósito : 18 de septiembre de 2026
Tipo de carga : interface
fecha de fin de análisis : 18 de septiembre de 2026

Resumen (sección 1/3)

Localización de los textos sospechosos en el documento :



Incluido en el porcentaje de textos sospechosos :

Similitudes — **8%**

Sintáctica 7% | Semántica 1%

Pasajes con similitudes a fuentes encontradas en diferentes colecciones.



Detección de IA — **9%**

Textos estilísticamente próximos a un texto generado por una IA.
Este índice es un indicador y no una prueba. Comprueba con el autor si domina los conocimientos mencionados en el documento.



Idiomas no reconocidos — **12%**

Pasajes en los que parte del vocabulario utilizado no forma parte del diccionario de la lengua.
Puede tratarse de un intento del autor de modificar el texto para evitar ser detectado.



No incluido en el porcentaje de textos sospechosos :

Textos entre comillas — **1%**

Pasajes entre comillas, a menudo indicativos de una cita.

Fuentes de similitudes (sección 2/3)

Similitudes

8%

Pasajes con similitudes a fuentes encontradas en diferentes colecciones.



Fuente principal detectada

N°	Descripciones	Similitudes	Ubicaciones
1	E4 Entregable_Patricio Perez... #78fd50 Detección semántica Viene de de mi grupo	3%	
2	G3_Sistema web y aplicación móvil para la... #684ca9 Detección semántica Viene de de mi grupo	2%	
3	Documento de otro usuario #bdf7bb Detección semántica Viene de de otro grupo	<1%	
4	Misión y Visión - Vivero Cuatro Vientos viverocuatrovientos.cl/mision-y-vision/	<1%	
5	Aplicativo de inteligencia de mercados para... Detección semántica repositorio.puce.edu.ec/handle/123456789/48361	<1%	
6	¿Qué es el patrón Observer y cómo se usa? Detección semántica keepcoding.io/blog/patron-observer-y-como-se-usa/	<1%	
7	Análisis FODA: Herramienta clave para la toma de... Detección semántica unade.edu.mx/articulos/analisis-foda-herramienta-clave-para-la-toma-d...	<1%	
9	Vivero Los Inkas – Vivero Los Inkas www.viverolosinkas.com.pe/wp/	<1%	
10	Documento de otro usuario #2d2b3b Viene de de otro grupo	<1%	

Fuente con similitudes fortuitas

N°	Descripciones	Similitudes	Ubicaciones
8	repositorio.puce.edu.ec Detección semántica repositorio.puce.edu.ec/server/api/core/bitstreams/5e8ec154-cb80-41a2-...	<1%	
11	Documento de otro usuario #92f11d Viene de de otro grupo	<1%	
12	GitHub - fastapi/fastapi: FastAPI framework, high performance,... github.com/FastAPI/FastAPI	<1%	
13	Documento de otro usuario #6d6617 Viene de de otro grupo	<1%	

N°		Descripciones	Similitudes	Ubicaciones
14		Ejemplos de análisis de situación: Definición según Autor, qué es,... definicionwiki.com/ejemplos-de-analisis-de-situacion-definicion-segun-au...	<1%	
15		Documento de otro usuario #7a29ab Viene de de otro grupo	<1%	
16		Documento de otro usuario #147c00 Viene de de otro grupo	<1%	
17		Atik yazılım geliştirme - Wikipedia tr.wikipedia.org/wiki/Atik_yaz%C4%B1l%C4%B1m_geli%C5%9Ftirme	<1%	

Fuente mencionada (sin similitudes detectadas)

N°		Descripciones
1		https://www.autentia.com/libros/
2		https://doi.org/10.17993/3ctecno/2019.v8n1e29/30-41
3		https://fastapi.tiangolo.com/
4		https://fourweekmba.com/kanban
5		https://www.istqb.org
6		https://www.lean6sigmatraining.co.uk/kanban-tools-are-now-used-by-pixar-spotify
7		https://lucide.dev/
8		https://learn.microsoft.com/en
9		https://dev.mysql.com/doc/refman/8.0/en/
10		https://docs.python.org/3/library/asyncio.html
11		https://react.dev/
12		https://refactoring.guru/es/design-patterns/strategy
13		https://www.sqlalchemy.org/
14		https://vitejs.dev/
15		https://www.espressif.com/sites/default/files/documentation/esp32-wroom
16		https://www.electromania.pe/producto/protoboard-transparente-de
17		https://theyosh.github.io/TerrariumPI/hardware/sensor/sensirion
18		https://www.espressif.com
19		https://www.rfc-editor.org/rfc/rfc6455
20		https://thonny.org/
21		https://docs.micropython.org/en/latest/
22		https://www.nxp.com/docs/en/user-guide/UM10204.pdf

N°		Descripciones
23		https://sensirion.com/media/documents/213E6A3B/63A5A569/Datasheet_SHT3x_DIS.pdf
24		https://www.youtube.com/watch?v=ODITzqixBy8

Puntos de interés (sección 3/3)

INSTITUTO DE EDUCACIÓN SUPERIOR PRIVADA IDAT PROGRAMA DE ESTUDIOS EN DESARROLLO DE SISTEMAS DE INFORMACIÓN

Desarrollar un Sistema Informático para optimizar la visualización de Sensores del

vivero Los Winkas

Trabajo de aplicación profesional para obtener el título de Desarrollo de Sistemas de Información

Moises Roberto, Linconao

0009-0007-2728-6837

Lima – Perú

2026

Dedicatoria

A Dios, por ser mi guía constante, quien me fortalece en **Cristo** y me brinda su protección en cada paso que doy.

A mi familia, por su amor y apoyo constante, incluyendo **Kiki,** mi gato, por su compañía silenciosa.

Moises Roberto Linconao.

3 6 7 8 9 10 11 11 12 12 12 13 13 13 13 13 13 13 13 13

TABLA DE CONTENIDO

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

Resumen Ejecutivo

Introducción

Capítulo I ANÁLISIS DEL NEGOCIO

Descripción de la Empresa

Antecedentes de la empresa

Visión empresarial

Misión empresarial

Organigrama y Funciones de las Principales Áreas

Identificación de la Necesidad (Problema u Oportunidad)

Análisis FODA (Proceso)

Elicitación de Requisitos

Análisis del Problema

Propuesta de Solución

Alternativas de Solución

Factibilidad del Proyecto

Factibilidad Técnica

Factibilidad Operativa

13 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 15 15

Factibilidad Económica

Capítulo II PLANIFICACIÓN DEL PROYECTO

Enfoque de Desarrollo y del Ciclo de Vida

Enfoque a aplicar

Justificación de metodología a utilizar

Arquitectura de Software a utilizar

Modelos, métodos y artefactos a aplicar

Planificación del Proyecto

Plan del Proyecto

Declaración y Alcance del Proyecto

Objetivos del Proyecto

Objetivo General

Objetivos Específicos

Beneficios del Proyecto

Cronograma

Interesados

Supuestos

Factores Críticos de Éxito

Riegos

Comunicaciones

Capítulo III EJECUCIÓN DEL PROYECTO

Fase Inicio

15 15 15 15 15 15 16 16 16 16 16 16 16 16 16 16 17 17 17 18 18 19

Presentación del Lanzamiento de Trabajo (Kick Off)

Fuentes de obtención de Requerimientos

Fase Planificación

EDT / Especificación de Casos de Uso de Alto nivel Priorizados

Modelo de Casos de uso

Modelo de Análisis

Cronograma

Paquetes de Trabajo/Descomposición de Tareas

Fase Ejecución

Seguimiento y Control/Validación de las Iteraciones

Carta Gantt/ Tablero Kanban, Burn Down Chart

Gestión de Cambio/Mantenimiento de Product Backlog

Pruebas de Aceptación

Fase Transición y Cierre

Lecciones Aprendidas

Actas de Conformidad de los Entregables

Capítulo IV PROGRAMACIÓN

Implementación de la Arquitectura de Software

Modelo de Diseño

Creación de la Base de Datos y Generación del Script

Implementación de Librerías y Dependencias

Codificación del Front End

20 20 21 22 23 23 24 24 25 25 25 25 26 27

Codificación del Back End

Listado de Consultas y Reportes

Codificación de Mantenedores (CRUD)

Procesos Transaccionales

Capítulo V PRUEBAS DE CALIDAD DE SOFTWARE

Pruebas Unitarias

Pruebas Integrales

Otros (Casos de Prueba)

Capítulo VI CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

RECOMENDACIONES

REFERENCIAS BIBLIOGRAFICAS (Referencias según APA)

Glosario de Términos

ANEXOS

Resumen Ejecutivo

Esta investigación se centró en el desarrollo de un sistema de monitoreo de sensores Iot, diseñado para integrar diferentes fuentes de datos ambientales. Luego del análisis en el vivero "Los Winkas", se identificó que el registro manual de mediciones ambientales generaba datos inconsistentes y una lentitud de respuesta ante contingencias climáticas, generando pérdidas económicas debido al daño en los cultivos. Debido a lo anterior, se implementó una solución informática que permite la visualización de mediciones en tiempo cercano al real mediante un tablero de control, además de poder configurar los umbrales de mínimo y máximo para cada sensor. El desarrollo concluyó en una arquitectura de backend robusta que fue validada mediante pruebas con sensores IoT físicos. Los resultados demuestran una mejora inmediata en los procesos operativos, detectando alertas tempranas, generación de reportes históricos de seguimiento y una mayor precisión en la interpretación de datos ambientales, eliminando las confusiones y falta de registro al poder tener almacenado tanto las alarmas generadas como las mediciones tomadas. De esta manera los directivos pueden interpretar la “salud” del vivero basándose en hechos concretos y no en suposiciones. El sistema es escalable y desacoplado, lo que garantiza que la lógica de negocio sea independiente del hardware

utilizado. En conclusión, este sistema resuelve el problema operativo de "Los Winkas", y transforma su modelo productivo en innovación tecnológica, permitiendo evaluar decisiones basada en evidencias, permitiéndoles crecer en el mercado nacional apostando a las nuevas tecnologías.

Introducción

El presente trabajo se describe el desarrollo de un sistema informático para mejorar la visualización y el monitoreo de sensores ambientales en casi tiempo real, se ha elegido con fines académicos el vivero “Los Winkas” como objetivo de estudio. La idea nace a partir de la necesidad de contar con una herramienta que centralice la supervisión de las condiciones del vivero de forma más ordenada y eficiente, ya que el proceso actual presenta limitaciones para un correcto control, además de la pérdida de información.

El propósito principal es implementar una solución tecnológica que ayude a tomar mejores decisiones operativas por ejemplo: decidir cuándo regar en base a la humedad, determinar la ventilación si es necesario, o colocar mallas de sombreado si las temperaturas son muy elevadas, adecuar la cantidad de insumos (agua, fertilizantes, entre otros) basándose en las necesidades climáticas, a partir de datos obtenidos por mediciones casi en tiempo real.

Para ello, se utilizó una arquitectura en capas, empleando FastAPI para el backend y React para el frontend, incorporando comunicación mediante WebSockets, de esta manera podemos obtener los datos rápidamente y actualizaciones casi en el instante.

El trabajo se encuentra organizado en seis capítulos: el Capítulo I aborda el análisis del negocio y la problemática identificada; el Capítulo II desarrolla la planificación y gestión del proyecto; el Capítulo III presenta el análisis de los requerimientos; el Capítulo IV describe el diseño técnico y de software; el Capítulo V explica la construcción e implementación del sistema; y finalmente, el Capítulo VI expone las conclusiones y recomendaciones finales.

Capítulo I

ANÁLISIS DEL NEGOCIO

Descripción de la Empresa

Antecedentes de la empresa

Vivero Los WInkas es una empresa peruana especializada en la producción y

comercialización de plantas ornamentales y productos afines además de contar con

invernaderos de propagación. Brinda servicios paisajistas profesionales y ofrece flores y

follajes de corte en el mercado nacional e internacional.

Figura 1.

Logo de del vivero Los Winkas

Nota. La empresa es ficticia y ha sido creada con fines académicos.

Figura 2.

Ubicación Geográfica del vivero Los WInkas

Nota. Adaptado de Google Maps

Visión empresarial

Producir plantas con los más altos estándares de calidad y sanidad. Entregando un servicio confiable a pequeños, medianos y grandes agricultores a lo largo del país, el cual se enfoca en la innovación y desarrollo para nuevos desafíos productivos.

Misión empresarial

Llegar a convertirnos en un real medio de desarrollo para la producción de plantas para el mercado nacional e internacional, es nuestro principal objetivo. A través de la implementación de novedosas técnicas productivas y comerciales, queremos generar un cambio radical a la experiencia de la compra de plantas.

Organigrama y Funciones de las Principales Áreas

Figura 4

Organigrama del vivero Los Winka

Nota. El organigrama está adaptado acorde a las funciones y responsables de cada área.

Mapa de Procesos y Proceso(s) a Automatizar

El vivero “Los Winkas” cuenta con un sistema de sensores los cuales permite monitorizar los cambios ambientales, en la actualidad el proceso por el cual se realiza el seguimiento es el siguiente:

En un horario, no definido durante el día, el Encargado del vivero se acerca al área donde se encuentran los medidores, revisa el acta de mediciones, en la cual escribe a mano el detalle de lo que observa en cada una de sus visitas. Luego carga la lectura de cada uno de los sensores que posee, y realiza las anotaciones pertinentes.

Durante la validación de los datos verifica si estos se encuentran dentro de los parámetros aceptables o hay inconsistencias. En el caso de que los datos sean acordes a lo estipulado, se continúa generando un reporte para informar al área de Coordinación y Mantenimiento, los cuales recepcionan el informe y lo archivan.

En el caso de que alguno de los valores no se encuentre dentro de los parámetros aceptables, se comunica al área de Mantenimiento, quienes llaman a los Operarios del vivero, estos realizan los ajustes y correcciones ambientales necesarias. Al finalizar, generan un reporte comentando las tareas realizadas y las nuevas mediciones tomadas luego de solventar la anomalía. El área de Mantenimiento recepcionan el informe y en caso de ser correcto finaliza el proceso o en caso contrario solicita una nueva revisión.

Figura 5

Mapa de Procesos

Nota. Esta imagen describe el proceso de negocio correspondiente al proceso de control de sensores del vivero Los Wincas, elaborado en base al análisis del negocio.

Identificación de la Necesidad (Problema u Oportunidad)

Análisis FODA

El Análisis FODA es una matriz que se utiliza para evaluar la situación actual de una organización, proyecto o individuo, identificando factores internos y externos clave que pueden influir en el logro de los objetivos. El valor del FODA radica en su simplicidad para combinar estos factores (por ejemplo, usar **Fortalezas** para aprovechar **Oportunidades** o corregir **Debilidades** para evitar **Amenazas),** lo que lleva a la formulación de estrategias específicas.

Figura 6

Análisis Foda

Nota. Elaboración propia con base en la información recopilada

Elicitación de Requisitos

Análisis del Problema

En la actualidad, el Vivero “Los Winka" presenta el problema de no contar con una plataforma para el monitoreo de los sensores ambientales instalados. Además, no dispone de un sistema de alarmas ni de reportes automáticos de seguimiento, lo que limita la capacidad de control y reacción ante posibles variaciones en las condiciones ambientales.

Esta situación genera los siguientes inconvenientes:

• Dificultad para detectar cambios críticos en tiempo real, debido a la falta de un monitoreo continuo.

• Ausencia de alarmas visuales, lo que provoca una fuerte dependencia de la supervisión manual.

• Inexistencia de un registro histórico de eventos, impidiendo analizar o rastrear incidentes pasados.

• Imposibilidad de reaccionar de manera rápida y oportuna ante anomalías ambientales.

• Deficiencias en la comunicación interna, que retrasan la coordinación entre las áreas involucradas.

Estos problemas afectan la eficiencia operativa del vivero y la calidad del control

•Cambios rápidos en la tecnología •Factores de índolo económico •Competencia de otros locales

•Implementar un sistema de monitoreo automatizado •Incremento de la productividad al reducir tareas manuales •Ventaja Competitiva

•Implementación de un sistema automatizado •Uso de tecnologías •Ampliar los servicios

•Experiancia del Personal. •Infraestructura •Compromiso del Personal

F O

ambiental, evidenciando la necesidad de implementar un sistema automatizado de monitoreo y alerta temprana.

Figura 7

Árbol de Problemas

Figura 8

Curva del Problema

Propuesta de Solución

Se propone como solución la creación de un sistema informático que permita supervisar de forma centralizada los sensores ambientales en el vivero, mediante una plataforma intuitiva. Esta permitirá a los usuarios visualizar en semi tiempo real, las mediciones captadas por los sensores, configurar alarmas personalizadas basadas en parámetros críticos y acceder a un historial detallado de eventos.

El objetivo principal de esta solución es la posibilidad de controlar las condiciones ambientales del vivero de manera activa. Al centralizar la supervisión, se reducirán los tiempos de respuesta ante posibles incidencias, asegurando un registro de los datos de manera completa y confiable al ser almacenados mediante el sistema. Esto no solo mejorará la eficiencia operativa, sino que también facilitará la toma de decisiones informadas. Además, se garantizará un entorno adecuado para el crecimiento y conservación de las especies cultivadas en el vivero.

Alternativas de Solución

Alternativa 1: Integración de un sistema de monitoreo automatizado con aplicación móvil

Se plantea como segunda opción, el desarrollo de un sistema Mobile, de comunicación directa con los sensores en el vivero, de esta manera el monitoreo puede realizarse de manera más rápida y con notificaciones mediante mensajes de texto o alertas visuales.

Alternativa 2: Implementar un sistema IoT (Internet de las Cosas), conectado a un modelo de IA para la toma de decisiones.

Esta alternativa se plantea como una solución más avanzada y costosa que las propuestas, la integración del monitoreo a un modelo de Inteligencia Artificial conectado a la nube, donde se procesen y análisis los datos con el fin de detectar patrones y posibles riesgos. Mediante el uso de técnicas predictivas, el sistema podría anticipar las condiciones desfavorables, adelantándose a la emisión de alertas e informes más precisos.

Factibilidad del Proyecto

La fiabilidad del Proyecto se sustenta bajo el análisis de las principales dimensiones descritas a continuación:

Factibilidad Técnica

Se cuentan con los recursos tecnológicos para poder llevar el proyecto a cabo. El vivero cuenta con una infraestructura de sensores ya instaladas y funcionando, lo que facilita la integración con el sistema informático. Los sensores Iot poseen una interfaz que puede ser programada en Python, eso lo hace mucho más adaptable a cualquier capa de que pueda responder a la interfaz que se programe. Desde este punto es factible que pueda implementarse de manera óptima.

Factibilidad Operativa

El vivero cuenta con el personal idóneo y suficiente para adecuarse al nuevo sistema de monitoreo. No solo resolverá el inconveniente de las visualizaciones, sino que simplificará tareas, el proceso se volverá más corto, y menos tedioso al no tener que registrar de manera manual los eventos, evitando errores humanos. Mejorará la comunicación entre

las áreas de mantenimiento, supervisión y coordinación. Por estas razones, la factibilidad operativa es alta y favorable.

Factibilidad Económica

Análisis de Factibilidad del Proyecto:

Cálculo de Van y Tir

Al realizar la estimación de los costos iniciales de desarrollo, la adquisición de sensores con los cuales ya cuenta el vivero por lo que no representarán un gasto, además del mantenimiento, y soporte. Comparado con los beneficios esperados, como la reducción en la pérdida de plantas por las anomalías ambientales no detectadas a tiempo, la disminución del personal en supervisar manualmente el vivero, pudiéndose encargar de otras tareas más relevantes, determinan que el proyecto es económicamente viable.

Para el análisis financiero se consideran los siguientes indicadores:

• **VAN (Valor Actual Neto):** el proyecto presenta un VAN positivo, lo que indica que los beneficios futuros descontados superan la inversión inicial (Sapag Chain et al., 2014).

• **TIR (Tasa Interna de Retorno):** la TIR estimada se encuentra por encima del costo de oportunidad del capital, demostrando que el proyecto es rentable (Sapag Chain et al., 2014).

En conclusión, el proyecto es económicamente factible, sustentando una inversión acorde a los beneficios que aporta el control del vivero.

Figura 9

Tabla de Costos

Flujo neto anual: 8,000 – 2,000 = 6,000 S/. por año

Figura 10.

Tabla de flujo monetario

VAN (Valor Actual Neto) S/ +7,745

TIR (Tasa Interna de Retorno) = 20%

Periodo de recuperación: 3 Años

En conclusión, el “Sistema de Monitoreo del Vivero Winka” es económicamente viable, ya que presenta un VAN positivo (S/ 7,745) y una TIR del 20%.

Capítulo II

PLANIFICACIÓN DEL PROYECTO

Enfoque de Desarrollo y del Ciclo de Vida

Kanban Proviene del japonés y significa “Tarjeta” es una metodología que surge en Toyota a finales de 1940, por el

Ingeniero Taiichi Ohno. Se basa en el principio “just-in-time”, la cual se traduce en un sistema que produce lo que requiere, en el momento que se necesita, con la calidad especificada, y sin desperdiciar recursos, generando los componentes necesarios para satisfacer la demanda en el tiempo requerido. El sistema Kanban se basa en los siguientes principios: - Visualización: Kanban, permite tener una manera clara y visual de todo el desarrollo, pudiéndose identificar de manera rápida las tareas y su clasificación, como pueden ser las de usuario, de servicio, y manuales. - Calidad: El enfoque plantea, que lo realizado se haga bien desde el comienzo. (A diferencia de técnicas como TDD en la que parte de pruebas para validar el código), Kanban promueve la prevención de errores y el enfoque en la calidad continua durante todo el proceso. - Priorización: Gestionar el tiempo y los recursos de manera ordenada, priorizando las tareas que se enfocan en el producto final. - Facilita la mejora continua: Promueve la continuidad de las actividades

En un mundo en constante cambio, donde el tiempo es limitado y los requerimientos son cada vez más exigentes, resulta fundamental hacer las cosas bien desde el principio. No hay espacio para desperdiciar ese recurso tan valioso en tareas que no aportan al núcleo del sistema o que no son realmente prioritarias. Saber priorizar y reconocer los elementos esenciales permite visualizar el producto final como el resultado coherente de todas sus partes. En este contexto, el sistema Kanban se convierte en una herramienta clave dentro de la filosofía de producción “Just in Time”, optimizar los procesos, reducir desperdicios y garantizar un flujo de trabajo eficiente y enfocado constituye un elemento esencial dentro de las metodologías de producción **Just in Time.**

Implementar este sistema, requiere un análisis de requerimientos profundo, para comprender las prioridades, del sistema. Sin embardo, no es sencillo, se requiere que las condiciones sean estables, y haya una baja variación en lo solicitado.

A medida que se desarrolla, se visualizan nuevos problemas y desafíos, retrasos, y cuellos de botellas en tareas alternas al núcleo. Kanban, nos ayuda en la correcta coordinación y visualización de los desafíos mencionados.

Muchas organizaciones como: Toyota (pionera), Microsoft, Spotify, Intel, Amazon, entre otras, demuestran que esta metodología es un éxito. Por ello y por todo lo anteriormente dicho, opto utilizar Kanban como gestión visual y ágil de productividad.

Arquitectura de Software por utilizar

El sistema estará desarrollado utilizando una **arquitectura por capas,** lo que permite organizar las funcionalidades en módulos con responsabilidades bien definidas, facilitando la re utilización de componentes y el mantenimiento. Este enfoque promueve la separación de la lógica de negocio, de la interfaz de usuario y los datos, contribuyendo de esta manera a la evolución del sistema y mejorando la calidad en el desarrollo.

En el **Capítulo IV: Programación,** se describe la implementación de la arquitectura de software del sistema de monitoreo.

Planificación del Proyecto

Plan del Proyecto

El Plan del Proyecto se describe en las siguientes etapas: Análisis del Negocio, Diseño de la solución, Implementación de la solución propuesta y Monitoreo conjunto a la resolución de posibles conflictos.

Figura 12.

Planificación de Proyecto

Declaración y Alcance del Proyecto

El presente proyecto tiene como finalidad el desarrollo de un Sistema de Monitoreo de Sensores para el vivero “Los Winkas”, para poder visualizar en semi tiempo real, (near real time), los datos emitidos por diferentes sensores ambientales, registrar las mediciones, y generar alertas en caso de superar umbrales configurables.

El alcance va desde el diseño, y la implementación de una arquitectura en capas como he mencionado anteriormente, desarrollando una interfaz web para la visualización de la información.

Objetivos del Proyecto

Objetivo General

Desarrollar e implementar un sistema de monitoreo que permita observar y analizar en tiempo real las condiciones ambientales del vivero “Los Winkas”, mediante una arquitectura por capas que asegure modularidad, eficiencia y escalabilidad.

Objetivos Específicos

1- ***Análisis de los requerimientos funcionales:*** determinar las variables y parámetros ambientales a controlar e informar.

2- ***Diseño de la arquitectura de software:*** crear las diferentes capas para garantizar el procesamiento de la lógica de negocio.

3- ***Desarrollo de la interfaz de usuario:*** Creación de una interfaz web con React para visualizar los datos de los sensores en casi tiempo real.

4- ***Configuración de Base de Datos:*** Integrar MySQL como histórico y almacén de la información.

5- ***Disposición de interfaz para la conexión de múltiples fuentes:*** Realizar una interfaz que permita desacoplar la lógica de la fuente de datos. En el sistema utilizaremos un api como pruebas de datos, la interfaz definirá un contrato, para permitir integrar cualquier sistema externo como fuente de datos.

Beneficios del Proyecto

La implementación de este sistema de monitoreo IoT tiene una importancia positiva en la administración del proceso de Revisión de los parámetros ambientales en el vivero "Los Winkas" siendo los siguientes los beneficios principales:

• Precisión de Datos: Al registrar los datos mediante un Sistema Informático, se garantiza que la información almacenada sea fidedigna, y esté disponible para auditorías o como datos históricos. • Respuesta Temprana: Al poder configurar umbrales, el Sistema actúa como un Centinela que detecta desviaciones dentro de los parámetros normales lo cual posibilita una intervención temprana antes de que se produzcan daños en los cultivos. • Reducción de Pérdidas Económicas: Al minimizar el Riesgo de mortalidad en las plantas por condiciones ambientales no controladas, el Proyecto ayuda a la protección de la inversión en plantas. • Optimización de Recursos: El personal puede enfocarse en tareas de mayor criticidad e importancia en lugar de dedicar tiempo a la recolección de datos, optimizando los tiempos operativos. • Escalabilidad Tecnológica: El diseño de la solución desacoplado el software del hardware utilizado, en el Vivero permitiendo en el futuro, en caso de requerir, integrar las nuevas tecnologías sin necesidad de una inversión completa en nuevo software. • Toma de Decisiones Estratégicas: Los directivos cuentan con evidencias, digitales para poder planificar y tomar decisiones con mayor precisión y seguridad de la información

Cronograma

Figura 13

Cronograma del proyecto

Interesados

Los interesados (Steakholders), del proyecto son los propietarios del vivero “Los Winkas”, quienes son los clientes principales, y usuarios finales.

Los Docentes, quienes supervisan y evalúan que el proyecto cumpla con los estándares académicos.

Supuestos

Factores Críticos de Éxito

Contamos con los Factores de éxito que proporcionan una mejora en el proceso de monitore de variables ambientales, a continuación, se describen.

Figura 14. *Cuadro de Factores de éxito.*

Riegos

Como en todo sistema, siempre contamos con Riesgos, estos se deben considerar y detallar a continuación

Figura 15.

Cuadro de Riesgos

Comunicaciones

La comunicación es una parte muy importante en toda organización, describiré a continuación de qué manera se llevará a cabo las diferentes reuniones e información importante para los interesados.

Figura 16.

Cuadro de Comunicaciones

Capítulo III

EJECUCIÓN DEL PROYECTO

Fase Inicio

Luego de haber definido los objetivos, planteado la solución y el alcance del proyecto,

la ejecución comienza formalmente luego de la reunión de lanzamiento(Kick Off), durante la

misma se da formalidad al camino a seguir, y los interesados (Steakholders) comprenderán su

rol, las herramientas y los canales de comunicación que se llevará a cabo.

Presentación del Lanzamiento de Trabajo (Kick Off)

Acta de Constitución

A continuación, se presenta el Acta del Proyecto donde se formaliza el inicio de este, definiendo la descripción del producto a entregar y los lineamientos declarados, además de los objetivos, el propósito, y el alcance, conjunto a las Fases

Figura 17

Acta de Constitución del Proyecto

Fuentes de obtención de Requerimientos

Descripción de requerimientos funcionales priorizados

El sistema permitirá la visualización y monitoreo de datos ambientales en tiempo cercano al real, podrá generar reportes, alertas y notificaciones visuales. Además, se habilitará la administración de sensores, usuarios, así como la integración con fuentes externas.

Funcionalidades Principales:

• Visualización de datos near real-time. • Monitoreo del estado de sensores. • Notificaciones visuales y alertas. • Generación de reportes. • Simulación de datos. • Integración con fuentes de datos externas • Visualización gráfica de información

Descripción de requerimientos no funcionales

Dentro de los requerimientos no funcionales encontramos los siguientes:

2 • El Sistema debe soportar al menos 10 sensores. • El sistema debe almacenar las contraseñas de manera encriptada. • El sistema debe ser compatible con los navegadores web: Chrome y Edge. • El backend deberá ser capaz de integrarse con diferentes fuentes de datos • El sistema debe visualizar los cambios de la información en un máximo 2 de 5 segundos.

Fase Planificación

1,2 ***EDT / Especificación de Casos de Uso de Alto nivel Priorizados***

Modelo de Casos de uso

La siguiente figura representa el Diagrama General de Casos de Uso del sistema, donde

detallamos los actores conjuntos a las Acciones que puede realizar cada uno de ellos.

Figura 18

Diagrama de Casos de Uso del Sistema

Descripción de los Actores:

En esta sección, describiremos los actores que interactúan con el sistema de monitoreo de sensores, definimos las características y los Casos de Uso que puede realizar. De esta manera se ven los perfiles destacados.

ACTOR Administrador ACT1

Descripción

Persona que puede conectar la fuente de datos, ver los datos ambientales y las alertas generadas, crear nuevos usuarios, y realizar configuraciones sobre los sensores.

Características Puede visualizar datos ambientales y realizar configuraciones además de visualizar los reportes

Relaciones: Se relaciona con el Encargado al poder crear usuarios con diferentes perfiles

Referencias:

Caso de Uso: Acceder al Sistema Caso de Uso: Cerrar Sesión Caso de Uso: Conectar Fuente de Datos Caso de Uso: Visualizar Datos Ambientales Caso de Uso: Configurar Alias/Nombre Caso de Uso: Configurar Visibilidad Caso de Uso: Configurar Umbral Caso de Uso: Visualizar Usuarios Caso de Uso: Crear Usuarios Caso de Uso: Eliminar Usuario Caso de Uso: Visualizar Reporte de Alertas Caso de Uso: Visualizar Reporte de Auditoria Caso de Uso: Visualizar Reporte de Mediciones Promedio Caso de Uso: Imprimir Reporte Autor: Moises Linconao

ACTOR Encargado ACT2

Descripción Persona que puede conectar la fuente de datos, ver los datos ambientales y las alertas generadas. Características Puede visualizar datos ambientales.

Relaciones: Se relaciona con el Administrador para poder controlar las mediciones de los sensores.

Referencias:

Caso de Uso: Acceder al Sistema Caso de Uso: Cerrar Sesión Caso de Uso: Conectar Fuente de Datos Caso de Uso: Visualizar Datos Ambientales Autor: Moises Linconao

Especificación de Casos de Uso de Alto Nivel.

CU Caso de Uso CU-001 Acceder al Sistema CU-002 Cerrar Sesión CU-003 Conectar Fuente de Datos CU-004 Visualizar Datos ambientales CU-005 Configurar Alias/Nombre CU-006 Configurar Visibilidad CU-007 Configurar Umbral CU-008 Visualizar Usuarios CU-009 Crear Usuario CU-010 Eliminar Usuario CU-011 Visualizar Reportes de Alertas CU-012 Visualizar Reporte de Auditoria CU-013 Visualizar Reporte de Mediciones Promedio CU-014 Imprimir Reporte

Descripción de los Casos de Uso

A continuación definimos los escenarios, donde se describe como el Actor interactúa

con el sistema.

Caso de Uso: Acceder al Sistema **CU-001 Actores:** Administrador, Encargado **Tipo:** Principal **Pre-Condición:** El Administrador o Encargado debe estar registrado en el Sistema **Post Condición:** El Administrador o Encargado pudo acceder al sistema **Flujo Normal:** 1. El Usuario ingresa su usuario y contraseña. 2. El Usuario selecciona Ingresar: El sistema valida los datos e ingresa al sistema.

Flujo Alternativo:

2A. El usuario ingresó un usuario inexistente: El sistema no debe permitir el ingreso, y debe informar que el usuario o contraseña son incorrectos. 2B. El usuario ingresó de manera incorrecta el user name o contraseña: El sistema no debe permitir el ingreso y visualizar un mensaje indicando que se ingresó usuario o contraseña incorrectos.

Caso de Uso: Cerrar Sesión **CU-002 Actores:** Administrador, Encargado **Tipo:** Extend de CU-002 Acceder al Sistema **Pre-Condición:** El usuario debe encontrarse dentro de su Sesión **Post Condición:** El usuario salió de su Sesión **Flujo Normal:** 1. El usuario selecciona "Cerrar Sesión" 2. El sistema solicita confirmación para Cerrar la Sesión 3. El usuario confirma la solicitud. 4. El sistema cierra la sesión 5. El usuario se encuentra en la página de Login

Flujo Alternativo:

3A. El usuario cancela la solicitud de confirmación: El sistema no cierra la sesión y se mantiene en la vista donde se encuentra.

Caso de Uso: Conectar Fuente de Datos **CU-003 Actores:** Administrador **Tipo:** Principal **Pre-Condición:**

El Administrador debe estar Logueado. Debe existir una fuente de datos conectada al Sistema. El Backend debe contar con la estrategia correspondiente **Post Condición:** El Administrador pudo conectarse a una Fuente de Datos Externa **Flujo Normal:** 1. El Encargado selecciona "Conectar Sensores" 2. El sistema se encarga de conectarse mediante WebSocket al backend y establece una conexión, comienza a escuchar la fuente de datos, guarda la conexión, mapea el jsons que recibe y realice un broadcast a todos los que están subscritos al WebSocket. 3. El Sistema establece la conexión y visualiza los datos de los sensores en "near" tiempo real. 4. Se ejecuta el "CU-004 Visualizar Datos Ambientales"

Flujo Alternativo:

2A. El Sistema no logra conectar a la fuente de datos: El sensor puede estar apagado o no estar emitiendo datos 2B. El WebSocket es erróneo: El sistema no establece conexión e informa que la dirección de la fuente de datos es inexistente.

Especial:

Para este Casos de Uso es necesario que se cuente con un sensor Iot el cual emita los datos a un servidor mediante WebSocket. El Backend debe contar con la Estrategia (Strategy) para adaptar los datos recibido al formato que requiere el front. Y contar con la url del servidor el cual debe emitir mediante WebSocket

Caso de Uso: Visualizar Datos ambientales **CU-004 Actores:** Administrador, Encargado **Tipo:** Extend de CU-003 Conectar Fuente de Datos **Pre-Condición:** El Administrador o Encargado debe encontrarse logueado en el sistema

Post Condición:

El Administrador o Encargado puede visualizar los datos emitidos por los sensores **Flujo Normal:** 1. El Encargado ingresa al Tablero. 2. El sistema visualiza los siguientes datos para los sensores activos: El Sistema visualiza "Nombre del Sensor", "Mac del Sensor", "Valor de la medición", Gráfico visualizando la medición

Flujo Alternativo:

1A. No se pudo establecer conexión con la Fuente de Datos: El sistema no debe visualizar datos. 2A. No hay sensores activos: El Sistema no debe visualizar datos.

Caso de Uso: Configurar Alias/Nombre **CU-005 Actores:** Administrador **Tipo:** Extend de CU-003 Conectar Fuente de Datos **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema **Post Condición:** EL Administrador pudo modificar el Alias/Nombre del Sensor **Flujo Normal:** 1- El Administrador selecciona "Editar" de un Sensor. 2- El Sistema habilita la edición del "Alias/Nombre" del Sensor. 3- El Administrador cambia el "Alias/Nombre" y selecciona Guardar. 4- El sistema guarda y actualiza el "Alias/Nombre" del Sensor.

Flujo Alternativo:

3A. El usuario deja en blanco el campo: El sistema informa que no es permitido y no guarda el cambio. 3B. El usuario selecciona cancelar: El sistema cancela la modificación y no genera la modificación.

Caso de Uso:

Configurar Visibilidad **CU- 006 Actores:** Administrador **Tipo:** Extend de CU-003 Conectar Fuente de Datos **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema **Post Condición:** EL Administrador pudo cambiar la visibilidad del sensor **Flujo Normal:** 1- El Administrador selecciona "Visible" de un sensor. 2- El sistema cambia al modo "Oculto" el sensor. 3- El sistema no lo visualiza en el tablero

Flujo Alternativo:

1A. El sensor se encuentra "No Visible": El sistema lo cambia a "Visible": Y lo visualiza en el tablero.

Caso de Uso: Configurar Umbral **CU-007 Actores:** Administrador **Tipo:** Extend de CU-003 Conectar Fuente de Datos **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema **Post Condición:** EL Administrador pudo cambiar el umbral de un sensor **Flujo Normal:** 1- El Administrador selecciona "Umbrales". 2- El Sistema visualiza los umbrales configurados para cada sensor. 3- El Administrador selecciona "Modificar" de un Sensor. 4- El Sistema habilita la modificación del "Minimo" y "Máximo" del Sensor. 5- El administrador modifica los valores "Mínimo" y/o "Máximo" y selecciona "Guardar". 6- El Sistema actualiza los umbrales y guarda los cambios. Envía al backend una actualización. Se registra para la auditoría.

Flujo Alternativo:

5A. El usuario deja un campo vacío: El sistema no lo permite y no guarda la edición. 5B El usuario ingresa en el campo "Máximo" un valor inferior al mínimo: El sistema informa que el máximo no puede ser menor al mínimo y no guarda el cambio. 5C El usuario intenta ingresar letras: El sistema no lo permite. 5D. El usuario cancela la edición: El sistema no guarda los cambios y visualiza los valores que tenía.

Caso de Uso:

Visualizar Usuarios **CU- 008 Actores:** Administrador **Tipo:** Principal **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema

Post Condición:

EL Administrador pudo visualizar todos los usuarios registrados en el Sistema **Flujo Normal:** 1- El Administrador selecciona "Usuarios". 2- El Sistema visualiza el personal registrado. De cada usuario visualiza: "Nombre y Apellido", "Email", "Rol". Al lado de cada uno el botón Eliminar.

Flujo Alternativo:

Caso de Uso: Crear Usuario **CU- 009 Actores:** Administrador **Tipo:** Extend de CU-009 Visualizar Usuarios **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema **Post Condición:** EL Administrador pudo Registrar un Nuevo Usuario. **Flujo Normal:** 1- El Administrador selecciona "Usuarios". 2- El Sistema visualiza los usuarios y los campos para crear un nuevo usuario con los campos: "Nombre y Apellido", "Email", "Contraseña", Combo box de "Rol" y el botón "Crear Usuario". 3- El Administrador completa los campos, y selecciona "Crear Usuario". 4- El sistema informa que el usuario se ha creado y lo registra en la Base de Datos.

Flujo Alternativo:

3A El usuario no completa ningún campo y selecciona "Crear Usuario": El sistema informa que los campos son obligatorios. 3B El usuario completa un email de manera incorrecta: El sistema informa que el email no es válido. 3C El usuario ingresa un email existente: El sistema informa que el email existe y debe ingresar otro.

Caso de Uso: Eliminar Usuario **CU-010 Actores:** Administrador **Tipo:** Extend de CU-009 Visualizar Usuarios **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema **Post Condición:** EL Administrador pudo eliminar un Usuario. **Flujo Normal:** 1- El Administrador selecciona "Usuarios". 2- El Sistema visualiza los usuarios y los campos para crear un nuevo usuario con los campos: "Nombre y Apellido", "Email", "Contraseña", Combo box de "Rol" y el botón "Eliminar" junto a la lista de usuarios.

3- El Administrador selecciona "Eliminar". 4- El sistema solicita la confirmación antes de eliminar. 5- El Administrador Confirma la solicitud. 6- El Sistema elimina al usuario y actualiza la lista de usuarios.

Flujo Alternativo:

3A El Administrador selecciona su propio usuario: El sistema no lo permite y no lo elimina. 4A El Administrador cancela la confirmación: El sistema no elimina al usuario y visualiza la lista.

Caso de Uso: Visualizar Reportes de Alertas **CU- 011 Actores:** Administrador **Tipo:** Principal **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema **Post Condición:** EL Administrador pudo visualizar los reportes de alertas **Flujo Normal:** 1- El Administrador selecciona "Reportes" -> "Alertas". 2- El Sistema visualiza una tabla con las alertas generadas por los diferentes sensores visualizando los siguientes campos: "Fecha /Hora",

"Sensor", "Alerta", "Valor /Umbral", "Descripción".

Flujo Alternativo:

2. No hay alertas generadas: El sistema no visualiza datos.

Caso de Uso: Visualizar Reportes de Auditoria **CU- 012 Actores:** Administrador **Tipo:** Principal **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema **Post Condición:** EL Administrador pudo visualizar los reportes de auditoria **Flujo Normal:** 1- El Administrador selecciona "Reportes" -> "Auditoría". 2- El Sistema visualiza una tabla con las acciones realizadas en el sistema, se visualizan los siguientes campos "Fecha/Hora", "Acción" y Descripción detallada"

Flujo Alternativo:

Caso de Uso: Visualizar Reporte de Mediciones Promedio **CU- 013 Actores:** Administrador **Tipo:** Principal **Pre-Condición:** El Administrador debe encontrarse logueado en el sistema

Post Condición: EL Administrador pudo visualizar el reporte de mediciones promedio **Flujo Normal:** 1- El Administrador selecciona "Reportes" -> "Mediciones". 2- El Sistema visualiza una tabla con las mediciones promedio de los últimos dos minutos los campos que se visualizan son los siguientes: "Fecha/Hora", "Alias", "Mac", "Promedio", "Rango (Max/Min)"

Flujo Alternativo:

Caso de Uso: Imprimir Reporte **CU- 014 Actores:** Administrador **Tipo:** Extend de los CU-011, CU-012 y CU-13

Pre-Condición:

El Administrador debe encontrarse logueado en el sistema El Administrador debe encontrarse en alguno de los reportes. **Post Condición:** EL Administrador pudo imprimir el reporte **Flujo Normal:** 1- El Administrador selecciona "Imprimir Reporte". 2- El Sistema visualiza el impresor genérico de Windows dando la opción de "Guardar como pdf" o "Imprimir" físicamente.

Flujo Alternativo:

2. El usuario cancela la acción: El sistema cierra la ventana y visualiza las alertas

Diagrama de Paquetes

De manera de generar un desarrollo ordenado, y destacar la mutabilidad del sistema, la arquitectura se organiza en paquetes donde cada uno contiene los diferentes componentes de software.

Figura 19

Diagrama de Paquetes

Diagrama de Actores El siguiente diagrama identifica los roles y entidades que interactúan con el sistema. Tenemos como entidad superior el Usuario General el cual es una generalización, de la misma por herencia tenemos dos usuarios: Administrador y Encargado. La siguiente figura representa el Diagrama de Actores del Sistema.

Figura 20

Diagrama de Actores

Diagrama de Casos de Uso por Paquete

El diagrama de Casos de Caso por Paquete muestra la división de los requerimientos funcionales del software, en una estructura modular, permitiendo identificar dentro de cada paquete que función perteneces a cada uno.

Figura 21

Diagrama de Casos de Uso por Paquete

Atributos de Casos de Uso

A continuación, encontraremos el detalle de cada Caso de Uso.

Atributo Detalle

ID CU-01

Nombre Acceder al Sistema

Prioridad Alta

Precondición El usuario debe estar registrado en el Sistema

Actor Principal Administrador

Actor Secundario Encargado

Postcondición El usuario Ingresó al Sistema

Atributo Detalle

ID CU-03

Nombre Conectar Fuente de Datos

Prioridad Alta

Precondición

El Administrador debe estar Logueado. Debe existir una fuente de datos conectada al Sistema. El Backend debe contar con la estrategia correspondiente

Actor Principal

Administrador

Actor Secundario -

Postcondición El Administrador pudo conectarse a una Fuente de Datos Externa

Atributo Detalle

ID CU-04

Nombre Visualizar Datos ambientales

Prioridad Alta

Precondición

El Administrador o Encargado debe encontrarse logueado en el sistema. Se debe haber establecido conexión con la Fuente de Datos

Actor Principal

Administrador

Actor Secundario Encargado

Postcondición

El Administrador o Encargado puede visualizar los datos emitidos por los sensores

Atributo

Detalle

ID CU-05

Nombre Configurar Alias/Nombre

Prioridad Media

Precondición

El sistema debe estar conectado a la Fuente de Datos

Actor Principal

Administrador

Actor Secundario -

Postcondición

EL Administrador pudo modificar el Alias/Nombre del Sensor

Atributo Detalle

ID CU-06

Nombre Configurar Visibilidad

Prioridad Media

Precondición

El Administrador debe encontrarse logueado en el sistema

Actor Principal

Administrador

Actor Secundario Encargado

Postcondición

EL Administrador pudo cambiar la visibilidad del sensor

Atributo Detalle

ID CU-07

Nombre Configurar Umbral

Prioridad Alta

Precondición

El Administrador debe encontrarse logueado en el sistema. Debe estar conectado a una fuente de datos. Debe haber sensores activos

Actor Principal

Administrador

Actor Secundario Encargado

Postcondición

EL Administrador pudo cambiar el umbral de un sensor

Atributo Detalle

ID CU-08

Nombre

Visualizar Usuarios

Prioridad Baja

Precondición

El Administrador debe encontrarse logueado en el sistema

Actor Principal

Administrador

Actor Secundario -

Postcondición

EL Administrador pudo visualizar todos los usuarios registrados en el Sistema

Atributo

Detalle

ID CU-09

Nombre Crear Usuario

Prioridad Media

Precondición

El Administrador debe encontrarse logueado en el sistema

Actor Principal

Administrador

Actor Secundario -

Postcondición

EL Administrador pudo Registrar un Nuevo Usuario.

Atributo Detalle

ID CU-10

Nombre Eliminar Usuario

Prioridad Media

Precondición

El Administrador debe encontrarse logueado en el sistema

Actor Principal

Administrador

Actor Secundario -

Postcondición

EL Administrador pudo eliminar un Usuario.

Atributo Detalle

ID CU-11

Nombre

Visualizar Reportes de Alertas

Prioridad Media

Precondición

El Administrador debe encontrarse logueado en el sistema

Actor Principal

Administrador

Actor Secundario -

Postcondición

EL Administrador pudo visualizar los reportes de alertas

Atributo Detalle

ID CU-12

Nombre

Visualizar Reportes de Auditoria

Prioridad Media

Precondición

El Administrador debe encontrarse logueado en el sistema

Actor Principal

Administrador

Actor Secundario -

Postcondición

EL Administrador pudo visualizar los reportes de auditoria

Atributo Detalle

ID CU-13

Nombre

Visualizar Reporte de Mediciones Promedio

Prioridad Media

Precondición

El Administrador debe encontrarse logueado en el sistema

Actor Principal

Administrador

Actor Secundario

Postcondición

EL Administrador pudo visualizar el reporte de mediciones promedio

Atributo Detalle

ID CU-14

Nombre

Imprimir Reporte

Prioridad Baja

Precondición

El Administrador debe encontrarse logueado en el sistema El Administrador debe encontrarse en alguno de los reportes.

Actor Principal

Administrador

Actor Secundario

Postcondición

EL Administrador pudo imprimir el reporte

Modelo de Análisis

Diagrama de Arquitectura de Análisis

Se ha utilizado una Arquitectura por capas como se comentó anteriormente, en la cual se cuentan con las siguientes capas:

La siguiente figura representa el Diagrama de la Arquitectura por Capas.

Figura 22

Diagrama de Arquitectura por Capas

Diagrama de Clases del Análisis

Para la elaboración del presente Proyecto se ha programado en base al siguiente diagrama de

Clases, donde cada una tiene una responsabilidad específica y encapsula el funcionamiento.

Figura 22 – *Diagrama de Clases del Sistema*

Clases:

Sesion: Es un modelo de SQLAlchemy el cual gestiona el estado de autenticación de los usuarios. Archivo ***models.py.***

Representa la tabla "sesión" en la base de datos. Almacena el "token_uuid", la IP de origen y si la sesión está activa. Mantiene una relación con la clase "Persona".

Redis Client (redis_client):

Es la instancia global que maneja la comunicación con el servidor Redis para el cacheo de

configuraciones. Archivo ***redis_client.py.***

Nos provee de un cliente asíncrono para leer y escribir configuraciones de sensores de forma extremadamente rápida, evitando consultas constantes a la base de datos principal de no ser necesario.

SHT3x (Modelado de Hardware):

En esta clase reside en el código que se encarga de conectarse al microcontrolador (ESP32) y su sensor. La Clase está modelada para mostrar cómo un sensor físico (IoT) interactúa con el hardware. Lee temperatura y humedad a través

del protocolo I2C y envía los datos al backend mediante un cliente WebSocket.

Patrones de Diseño

Patrón Estrategia (Strategy Pattern)

Es un Patrón del tipo de Comportamiento, tiene como objetivo definir una familia de estrategias encapsuladas de manera que sea intercambiable cada una de ellas. Permite que el algoritmo varíe acorde al cliente que lo utiliza.

Tiene siguiente diagrama:

Figura 22

Diagrama del Patron Strategy

Nota. Esta imagen describe las clases intervinientes en el Patron Strategy. Fue tomada del Libro

10

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Patrones de diseño: Elementos de software orientado a objetos reutilizable.* Addison-Wesley.

El Patron Strategy permite cambiar el comportamiento de un objeto en tiempo de ejecución.

Gamma et al. (1995) señalan que "las *estrategias de división en líneas no están implementadas en la clase* ***Composición,*** sino que se implementan separadamente por las subclases de la clase abstracta ***Componedor,*** las cuales definen distintas estrategias" (p. 289).

Para resolver la gestión de datos, implementamos el patrón Strategy, lo que nos permite independizar la lógica del sistema de la fuente de donde provienen los datos. En lugar de anclar el código a una conexión fija, definimos una interfaz que actúa como un 'traductor' universal.

Esto significa que si el día de mañana queremos integrar sensores de calidad de aire como el MH- Z19B o el SCD30, o incluso cambiar la infraestructura para recibir datos desde una cola de MQTT, Kafka o una API externa, no tenemos que reescribir el backend. Basta con crear una nueva estrategia que herede de DataStrategy y normalice los datos para el frontend. Además, añadimos un endpoint que permite conmutar entre estas fuentes en tiempo real, dándole al sistema una flexibilidad total para adaptarse a cualquier entorno de hardware.

En nuestro sistema aquí encontramos el Patron Strategy:

Figura 23

Diagrama de Clases del Patron Strategy

Nota. Esta imagen visualiza el Patrón Strategy en nuestro sistema. En este caso contamos con dos estrategias: Para utilizar una Api y un Sensor IoT físico.

Diagrama de Secuencia

El siguiente diagrama visualiza la interacción entre Clases en el Patron Strategy y la secuencia del flujo de programa durante la ejecución

Figura 23 – *Diagrama de Secuencia del Patron Strategy*

Nota. Esta imagen representa el diagrama de Secuencia desde que llega una lectura, como la

procesa la estrategia y visualiza el frontend los datos.

Interfaz (Clase Base): DataStrategy, Define el método obligatorio “procesar()”.

Estrategia Concreta API: ApiStrategy, Procesa datos que vienen formateados desde una fuente externa como una API.

Estrategia Concreta Sensor: SensorSHT3xStrategy, Procesa los datos crudos que envía el microcontrolador ESP32 que se encuentra conectado a un Sensor Sght3x.

Patron Observer:

Define una dependencia entre objetos de forma que cuando un objeto cambia su estado, todos los objetos que dependen de él son notificados y pueden reaccionar si lo desean a una acción. Propósito: Definir una dependencia de uno-a-muchos entre objetos, de manera que cuando cambia el estado de uno se notifique a todos los que dependen. Conocido también Como Dependientes o Publish/Subscribe (Publicador/Suscriptor).

Aplicabilidad:

Cuando una abstracción depende de otro.

Cuando un objeto quiere cambiar otros y no sabemos cuántos son

Cuando un objeto debe notificar sin conocer a los otros (Fuerte desacoplamiento).

Define una dependencia entre objetos de forma que cuando un objeto cambia su estado, todos los objetos que dependen de él son notificados y pueden reaccionar si lo desean a una acción.

Figura 24

Diagrama de Clases del Patron Observer

Nota. Esta imagen fue tomada de **Autentia.** (s.f.). *Software Design: Principios y patrones del desarrollo de software, Primera parte: Guía para directivos y técnicos* (V.1). Autentia. https://www.autentia.com/libros/

La siguiente figura visualiza el Diagrama de Secuencia y como interactúa con el Patron Strategy

Figura 25

Diagrama Secuencia del Patron Observer

WebSocketOrchestrator: Esta clase es la encargada de conectarse con la fuente de datos (puede ser un sensor IoT, un api, cola MQTT, etc.) y decide a quién pasarle la información, acorde a la estrategia (anteriormente explicado) configurada para que sea procesada y luego enviada al frontend.

ConnectionManager: Mantiene una lista de todos los “clientes” (frontend) que están “observando” el backend en ese momento. Cuando llega un nuevo dato, se encarga de "gritarlo" (hacer un broadcast) a todos los observadores para que se actualicen.

Redis Client (redis_client): Maneja la comunicación con el servidor Redis para el cacheo de

configuraciones. De esta manera evitamos consultar a la Base de Datos constantemente los parámetros de configuración de los sensores activos, evitamos el flujo de datos constante lo cual sería demasiado lento generando cuellos de botella. Al utilizar Redis, contamos con un cliente asíncrono para leer y escribir datos de manera extremadamente rápida.

En conclusión, como bien indica Fowler (2018), el patrón Observer se apoya en los modelos orientados a eventos para ganar flexibilidad. En este proyecto, en lugar de que el backend gestione los sensores (o fuente de datos) uno a uno a sus suscriptores, implementamos un Orquestador que centraliza la comunicación. Esto significa que podemos conectar o desconectar nuevos módulos (como otros frontend, Dashboard, etc.) sin tener que tocar ni una línea de código del back de sensores.

Diagrama de Comunicación del Análisis

En el siguiente diagrama detallaremos el CU “Conectar Fuente de Datos” como es la comunicación entre componentes

CU003-Conectar Fuente de Datos

Figura 27

Diagrama Comunicación del CU003-Conectar Fuente de Datos

CU004 - Visualizar Datos ambientales

En el Caso de Uso 04 se detalla la Visualización de los Datos Ambientales, en el diagrama de Comunicación se describe como la información viaja desde una fuente de datos hasta visualizarse en el tablero

Figura 28

Diagrama Comunicación del CU004 - Visualizar Datos ambientales

Cronograma / Planificación (Iteraciones Kanban)

En este proyecto utilizo Kanban para gestionar el flujo de trabajo, dividiendo cada parte del desarrollo en Iteraciones Funcionales. A diferencia de Scrum, los tiempos no son cerrados, pero para mantener un urden, describiré un flujo continúo basado en prioridades y colocando límite al trabajo en cada fase.

Iteración 1 Duración Semanas de la 1 a la 4 Descripción Base del Sistema y Conectividad

Objetivo

Establecer la arquitectura base y la comunicación con la fuente de datos en semi tiempo real. Backlog priorizado

Configuración del entorno FastAPI y generar los modelos (Persona, Rol). Implementación de la clase base DataStrategy. Desarrollo del WebSocketOrchestrator y ConnectionManager. Implementación de ApiStrategy para lectura desde una Api. Casos de Uso: CU-001 Acceder al Sistema CU-002 Cerrar Sesión CU-003 Conectar Fuente de Datos

Iteración 2 Duración Semanas de la 5 a la 7 Descripción Configuración de Sensores y Lógica de Alertas Objetivo Implementar la lógica de negocio para el manejo de los Sensores y las Alertas, además de continuar con la persistencia de datos

Backlog priorizado

Implementar el Patrón Observer para la emisión (broadcast) de los datos. Desarrollo de la lógica de umbrales (valor_min_umbral vs valor_max_umbral). Implementación de Redis como caché Creación de Tablero y Usuarios Casos de Uso: CU-004 Visualizar Datos ambientales CU-005 Configurar Alias/Nombre CU-006 Configurar Visibilidad

Iteración 3 Duración Semanas de la 8-11

Descripción

Visualización del Tablero, generación de Reportes e Integración con Sensores Reales. Objetivo Implementar la visualización de los tableros, refinando el frontend y generando las interfaces faltantes. Backlog priorizado

Desarrollo del SensorContext en el frontend. Implementar gráficos en semi tiempo real (Chart.js/Recharts). Generar resumen de mediciones para el historial. Implementar el patrón Strategy para la conexión a diferentes fuentes de datos Casos de Uso: CU-007 Configurar Umbral CU-011 Visualizar Reportes de Alertas CU-006 Configurar Visibilidad CU-013 Visualizar Reporte de Mediciones Promedio CU-008 Visualizar Usuarios CU-009 Crear Usuario

Iteración 4 Duración Semana 12 Descripción Pruebas Funcionales Objetivo Realizar las pruebas Funcionales, de Regresión y Ajustes Backlog priorizado

Realizar las Pruebas acorde a los Casos de Prueba. Revisar el sistema íntegramente. Realizar Ajustes necesarios.

Casos de Uso:

Descomposición de Tareas de las HU

A continuación, se describen las tareas a desarrollar para cada Caso de Uso, junto a su tipo y el estado en que se

encuentran. La finalidad es detallar lo que implica el desarrollo de cada funcionalidad

Fase Ejecución

Seguimiento y Control/Validación de las Iteraciones

El desarrollo del sistema no fue lineal, sino que se construyó a través de Iteraciones de trabajo. En esta sección se detalla cada una de ellas, validando el avance desde las primeras conexiones con los sensores, hasta lograr la estabilidad y visualización de los datos ambientales en el tablero de control. El seguimiento nos permitió detectar errores y retrasos a tiempo y ajustar las funciones conjunto a las estrategias de desarrollo.

Figura 29

Seguimiento y Control de la Interacción 1

Figura 30

Seguimiento y Control de la Interacción 2

Figura 29

Seguimiento y Control de la Interacción 3

Figura 29

Seguimiento y Control de la Interacción 4

Tablero Kanban, Burn Down Chart

Tablero Kanban

Se implementó un Tablero Kanban para visualizar el flujo de trabajo. En esta herramienta se dividió las tareas en columnas (Backlog (pendiente), En Desarrollo, Listo para Pruebas, en QA y Finalizado(Done)), facilitando la detección de cuellos de botella y asegurando que ninguna funcionalidad quedara estancada.

Figura 30

Tablero Kanban de la Interacción 1

Figura 31

Tablero Kanban de la Interacción 2

Figura 32

Tablero Kanban de la Interacción 3

Figura 33

Tablero Kanban de la Interacción 4

Burn Down Chart

Este gráfico muestra el contrastar entre el trabajo pendiente y el tiempo que tenemos disponible. El valor que nos dio fue la visión realista del ritmo de trabajo, al observar los pendientes, la curva de desarrollo, pudiendo ajustar los esfuerzas y estrategias en cada iteración para poder garantizar que el sistema de monitoreo estuviera finalizado para la fecha de entrega

Figura 34

Burn Down Chart de la Interacción 1

Figura 35

Porcentaje de las tareas para la Iteración 1

Estado de la Tarea Cantidad (N°) Porcentaje (%)

Done (Finalizado) 9 50% En Desarrollo 4 22% Listo para Pruebas 3 17% En QA 2 11% TOTAL 18 100% Se ajustaron las prioridades y esfuerzos para la siguiente iteración

Figura 36

Burn Down Chart de la Interacción 2

Figura 37

Porcentaje de las tareas para la Iteración 2

Estado de la Tarea Cantidad (N°) Porcentaje (%)

Done (Finalizado) 13 52%

En QA 5 20%

En Desarrollo 4 16%

Listo para Pruebas 3 12% TOTAL 25 100%

Se ajustan las prioridades para poder finalizar en la siguiente Iteración. Se ha consumido tiempo de la Iteración 2 para finalizar tareas de la anterior fase.

Figura 38

Burn Down Chart de la Interacción 3

Figura 39

Porcentaje de las tareas para la Iteración 3

Estado de la Tarea Cantidad (N°) Porcentaje (%)

Done (Finalizado) 25 67.60%

En QA 5 13.50% En Desarrollo 4 10.80% Listo para Pruebas 3 8.10%

TOTAL 37 100%

Acciones para Finalizar en la Próxima Iteración

• **Finalización de Pendientes:** Antes de iniciar nuevas tareas, se priorizan el cierre de las actividades que se encuentran en las columnas de **QA** y **Listo para Pruebas** para evitar la acumulación de deuda técnica.

• **Refinamiento del Backlog:** Revisar y detallar las tareas de la siguiente iteración, asegurando que cada una tenga criterios de aceptación claros.

• **Análisis de Cuellos de Botella:** Identificar las razones por las cuales las tareas se estancan en fases de validación y ajustar la capacidad para agilizar el flujo de trabajo.

Figura 40

Burn Down Chart de la Interacción 4

Figura 41

Porcentaje de las tareas para la Iteración 4

Estado de la Tarea Cantidad (N°) Porcentaje (%)

Done (Finalizado) 14 77.80%

En QA 4 22.20% En Desarrollo 0 0% Listo para Pruebas 0 0% TOTAL 18 100%

En Resumen, podemos observar que para la iteración 4 se ha llegado a completar el 100% de las actividades. Gracias al tablero Kanban se pudo detectar los cuellos de botella y tomar las medidas necesarias para evitar retrasos.

Figura 42

Resumen final por Iteración

Gestión de Cambio/Mantenimiento de Product Backlog

El Product Backlog es el eje del proyecto, agrupa las funcionalidades, pendientes de desarrollo y nos da una visión para la revisión y refinamiento. A medida que las Iteraciones avanzaban, surgieron diferentes prioridades y ajustes. Este mantenimiento constante permitió que el esfuerzo del desarrollo se enfoque en las características de mayor valor que den valor al negocio.

Figura 43

Vista del Product Backlog con todas las historias de usuario finalizadas. Se ha utilizado

Clic Up como herramienta de gestión.

Figura 44

Vista del Backlog priorizado con las asignaciones

Priorización del Backlog

Para la priorización del backlog, se aplica la técnica **WSJF (Weighted Shortest Job First),** la cual permite ordenar las actividades considerando el valor económico que aportan como el tiempo necesario para su entrega. De esta manera, se favorece la priorización en la implementación de los elementos que generan mayor beneficio en el menor tiempo posible, colaborando en la toma de decisiones y alineando la priorización con objetivos de negocio claros. Este enfoque está respaldado por los principios de Kanban y por las prácticas de escalado ágil propuestas en SAFe, que promueven decisiones basadas en valor y eficiencia del flujo de trabajo.

Se establece que

Figura 45

Fórmula del WSJF

Explicado de otra manera =

WSJF = Costo del Retraso / Duración del trabajo

Costo del Retraso = Valor del Negocio + Urgencia Temporal + Reducción de Riesgos

David J. Anderson explica el concepto de Costo del Retraso(Cost of Delay) como base para tomar decisiones económicas en sistemas Kanban. En SAFe se toma este concepto y se formaliza en la fórmula comentada.

Iteración 1: Infraestructura

En esta etapa, el foco es el armado de la arquitectura del Sistema

Figura 46

Fórmula del WSJF para los CU de la Iteración 1

Iteración 2: Visualización y Operación

En esta etapa, el foco está en las funcionalidades de utilidad para el usuario final

Figura 47 –

Fórmula del WSJF para los CU de la Iteración 2

Nota: *Se prioriza la visualización (CU-004) porque es el núcleo en el tablero de control.*

Iteración 3: Reportes y Control de Sensores y Umbrales

Utilización de los datos tomados por los sensores además de aplicar la lógica de negocio.

Figura 48

Fórmula del WSJF para los CU de la Iteración 3

Iteración 4: Pruebas del Sistema

En esta fase la prioridad es asegurar la calidad del sistema

Figura 49

Fórmula del WSJF para los CU de la Iteración 4

Fase Transición y Cierre

Lecciones Aprendidas/Retrospectivas (Iteraciones y del Product Backlog)

A diferencia de Scrum, donde la retrospectiva se realiza en tiempos definidos, en Kanban este espacio se utiliza para revisar el flujo de trabajo en el tablero. Su propósito es identificar cuellos de botella y tareas innecesarias, con el fin de mejorar el Lead Time. Según David J. Anderson, estas revisiones ayudan a ajustar los límites de trabajo en progreso (WIP) y a mejorar continuamente la calidad de las entregas.

Retrospectiva por Iteración

Iteración 1: Establecimiento del Flujo

Se completó el 50% de las tareas, lo que permitió dejar establecida la base y conectividad del sistema. Sin embargo, una parte importante del trabajo quedó detenida en la etapa de desarrollo, lo que evidencia dificultades en la configuración inicial del entorno. Para corregir esta situación, se propuso mejorar las definiciones de "Listo para Probar", de modo que las tareas avancen con mayor fluidez desde el inicio del proceso.

Iteración 2: Visualización y Operación

Se logró completar un mayor número de tareas, aun cuando los casos de uso presentaron mayor complejidad. La etapa de calidad (QA) se ha convertido en el cuello de botella del flujo. Para mejorar esta situación, se plantea incorporar revisiones de código y de pares antes de pasar a QA con el fin de tener una mejor calidad en lo entregado.

Iteración 3: Reportes y Control de Sensores y Umbrales

En esta etapa se alcanzó el mayor nivel de productividad, se logró completar el 68% de las tareas y se cerró 25. Sin embargo, la alta complejidad y la carga de trabajo, puso al límite todos los tiempos, aumentó considerablemente la presión y las horas de trabajo incorporadas. Se dividieron las tareas en unidades más pequeñas para reducir los riesgos y llegar al cierre.

Iteración 4: Pruebas del Sistema

En esta última iteración se logró completar la totalidad de las tareas, dejando sin pendientes los estados de desarrollo

y calidad. Me enfoqué en realizar ajustes y en revisar el sistema de forma integral mediante los casos de prueba definidos. Se logró resolver la deuda técnica y finalizar con un producto de calidad.

Actas de Conformidad de los Entregables

Figura 50

Acta de conformidad firmada

Capítulo IV

PROGRAMACIÓN

Implementación de la Arquitectura de Software

El sistema de Monitoreo de Sensores para el vivero Los Winkas está diseñado con una **Arquitectura por Capas,** ya que nos permite separa las funcionalidades en módulos con responsabilidades bien definidas, además de la reutilización de componentes.

"La arquitectura en capas es un modelo común para sistemas distribuidos, ya que permite aislar la lógica de negocio de la interfaz de usuario y los mecanismos de acceso a datos." (Sommerville, 2011, p. 219)

"El diseño en capas proporciona una estructura que soporta la división del trabajo y facilita la evolución controlada del sistema." (Pressman & Maxim, 2020)

Los autores mencionados anteriormente, determinan que, al tener módulos por separados, se desacopla la lógica y mejora la manera de desarrollar y probar, aportando calidad a nuestro sistema.

En una Arquitectura en capas, se divide la aplicación en "capas", con la intención de que cada una tenga una funcionalidad independiente. La arquitectura se centra en la separación y comunicación.

Figura 51.

Comunicación entre capas

Nota. Tomada de *Introducción a la arquitectura de software* (Blancarte Iturralde, 2020), Página 145.

Contaremos con las siguientes Capas:

Capa de Presentación: Se encargará de generar las vistas con la información necesaria para la

interacción con el usuario, en nuestro caso será una Web genera con el Framework de JS **React.** El cual visualizará los datos de los sensores en tiempo cercano al real.

Capa de Negocio: Contendremos toda la lógica de negocio. Se procesarán los datos acordes a la lógica, conectaremos la fuente de datos con la Base de Datos y se realizará la comunicación con la Capa de Presentación (envío y recepción de datos por parte del Frontend). Como lenguaje principal será **Python,** con servicios asíncronos para optimizar las respuestas.

Capa de Datos: Esta capa, responde a donde estará la información. Se utilizará **MySQL** como motor de Base de Datos relacional.

Capa de Integración: Esta capa representa la fuente externa de información. Para el proyecto con fines educativos se utilizará una **Api** que simulará el envío de datos de los diferentes sensores, generando de esta manera un escenario ficticio para la obtención de datos de una fuente externa. Además, se utilizará **Redis** como sistema intermedio para gestionar de manera eficiente la transmisión de datos en tiempo casi real.

Además, se incorpora una **interfaz de fuente de datos (DataSource),** la cual define los métodos para conectar y desconectarnos a fuentes externas, permitiéndonos abstraernos de la lógica, facilitando la integración futura a servicios o fuentes de datos.

Figura 52.

Arquitectura del sistema

Referencias

Adaptado de *Introducción a la arquitectura de software: Conceptos fundamentales e implementación*

(Blancarte Iturralde, 2020).

Modelo de Diseño

Diagrama de Componentes

Figura 53

Diagrama de Componentes

Descripción de la Arquitectura

La estructura del proyecto sigue el principio de arquitectura en capas. Como lo describe Robert C. Martin en su libro Clean Architecture (2017).

Señala que cada carpeta tiene una responsabilidad única y el flujo de dependencia se mantiene de manera interna.

"Los *modelos, esquemas, servicios, controladores y capa de persistencia están organizados para mantener el código desacoplado y extensible."*

1. Capa de Presentación Frontend: Se comunica con el backend mediante protocolos HTTP (para datos estáticos) y WebSockets (para datos en tiempo cercano al real).} **2. Capa de Negocio (Backend):** *Servicios:* Es el núcleo de la lógica. Orquesta el flujo entre la persistencia, la

cache (Redis) y las fuentes de datos.

WebRouter: Punto de entrada de las peticiones que se derivan a las fuentes de datos.

WebSocketSource: Implementa la interfaz "Data Source" para recibir flujos continuos de información.

DatosEnCache(Redis): Almacena de manera temporal datos para acceso rápido.

3. Capa de Datos

Persistencia De Datos: Gestiona la lógica de guardado mediante SQLAlchemy.

Conexión a BD: Interfaz que abstrae la base de datos física (MySQL).

FuenteDeDatos: Representa los sensores externos (Simulador, ESP32, etc.) enviando paquetes JSON.

Diagrama de Despliegue

Figura 54

Diagrama de Despliegue

Creación de la Base de Datos y Generación del Script

Para la Creación de la BD con sus respectivas tablas utilizaremos el siguiente script:

```
CREATE DATABASE IF NOT EXISTS sensores_db;

USE sensores_db;

-- 1. ROLES DE USUARIO

CREATE TABLE IF NOT EXISTS rol (

id_rol BIGINT AUTO_INCREMENT PRIMARY KEY,

nombre VARCHAR(50) NOT NULL -- 'Supervisor', 'Encargado', 'Técnico'

);

CREATE TABLE IF NOT EXISTS alerta (

id_alerta BIGINT AUTO_INCREMENT PRIMARY KEY,

fk_sensor BIGINT NOT NULL,

tipo_alerta VARCHAR(20), -- 'ALTA' o 'BAJA'

descripcion TEXT, -- Aquí se guardará el texto: "ALERTA BAJA: Sensor..."

valor_leido FLOAT, -- El valor que disparó la alerta (ej: 17.9)

umbral_limite FLOAT, -- El límite que se rompió (ej: 50.0)

fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

FOREIGN KEY (fk_sensor) REFERENCES sensor(id_sensor) ON DELETE CASCADE

);

-- 2. PERSONAS / USUARIOS
```

```
CREATE TABLE IF NOT EXISTS persona (
id_persona BIGINT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(100) NOT NULL,
apellido VARCHAR(100) NOT NULL,
email VARCHAR(150) UNIQUE NOT NULL,
password_hash VARCHAR(255) NOT NULL,
fk_rol BIGINT NOT NULL,
FOREIGN KEY (fk_rol) REFERENCES rol(id_rol)
);
-- 3. SESIONES
CREATE TABLE IF NOT EXISTS sesion (
id_sesion BIGINT AUTO_INCREMENT PRIMARY KEY,
fk_persona BIGINT NOT NULL,
token_uuid VARCHAR(255) UNIQUE NOT NULL,
ip_origen VARCHAR(45),
fecha_inicio TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
ultima_actividad TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
activa TINYINT(1) DEFAULT 1,
FOREIGN KEY (fk_persona) REFERENCES persona(id_persona)
);
-- 4. SENSORES (Información estática y estado actual)
CREATE TABLE IF NOT EXISTS sensor (
id_sensor BIGINT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(100) NOT NULL,
mac VARCHAR(50) UNIQUE NOT NULL,
tipo_medicion VARCHAR(50) NOT NULL, -- 'Temperatura', 'Humedad', etc.
estado_conexion VARCHAR(20) DEFAULT 'OFFLINE', -- 'ONLINE', 'OFFLINE'
ubicacion VARCHAR(100),
metadata_extra JSON, -- Para guardar datos que no todos los sensores tienen
fk_registrado_por BIGINT NOT NULL,
```

```
fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

FOREIGN KEY (fk_registrado_por) REFERENCES persona(id_persona)

);

-- 5. CONFIGURACIÓN Y ALARMAS

CREATE TABLE IF NOT EXISTS configuracion_sensor (

id_configuracion BIGINT AUTO_INCREMENT PRIMARY KEY,

fk_sensor BIGINT NOT NULL,

valor_min_umbral INT,

valor_max_umbral INT,

tiempo_tolerancia_seg INT DEFAULT 0,

notificar_email TINYINT(1) DEFAULT 0,

activo TINYINT(1) DEFAULT 1,

ultima_modificacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE

CURRENT_TIMESTAMP,

fk_usuario_gestion BIGINT, -- Quién hizo el último cambio

FOREIGN KEY (fk_sensor) REFERENCES sensor(id_sensor),

FOREIGN KEY (fk_usuario_gestion) REFERENCES persona(id_persona)

);

-- 6. MEDICIONES AGREGADAS (Resumen por minuto)

CREATE TABLE IF NOT EXISTS medicion_resumen (

id_medicion BIGINT AUTO_INCREMENT PRIMARY KEY,

fk_sensor BIGINT NOT NULL,

valor_promedio FLOAT NOT NULL,

valor_max FLOAT NOT NULL,

valor_min FLOAT NOT NULL,

fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

FOREIGN KEY (fk_sensor) REFERENCES sensor(id_sensor)

);

-- 7. EVENTOS Y AUDITORÍA

-- Logins, Alertas, Desconexiones, Cambios de Configuración

USE monitoreo_sensores;
```

```
CREATE TABLE IF NOT EXISTS evento_sistema (

id_evento INT AUTO_INCREMENT PRIMARY KEY,

tipo_entidad VARCHAR(50) NOT NULL,

id_entidad_afectada INT NOT NULL,

tipo_accion VARCHAR(50) NOT NULL,

descripcion TEXT,

fecha_hora DATETIME DEFAULT CURRENT_TIMESTAMP,

fk_persona INT NULL,

CONSTRAINT fk_evento_persona FOREIGN KEY (fk_persona) REFERENCES persona(id_persona)

) ENGINE=InnoDB;

USE sensores_db;

-- 8. Inserts iniciales

INSERT INTO rol (id_rol, nombre) VALUES

(1, 'Supervisor'),

(2, 'Encargado'),

-- 9. Insert de usuarios:

INSERT INTO persona (nombre, apellido, email, password_hash, fk_rol)

VALUES ('Moises', 'Linconao', 'moi.linconao@gmail.com', '1234', 1);
```

Para el sistema de monitoreo de sensores iot, se optó por utilizar MySQL como motor de base de datos principal, por su solidez, velocidad en el manejo de relaciones, lo cual es importante a la hora de almacenar datos anidados. MySQL permite organizar la información de manera ordenada dándonos la confianza de que los datos son íntegros, seguros y consistentes, lo cual es clave para manejar mediciones, umbrales y alertas.

MySQL se integra muy bien con arquitecturas en Capas como la que estoy presentando, y además tiene soporte para varios lenguajes de programación, lo que lo hace muy versátil. Esto resulta muy útil a la hora de querer escalar nuestro sistema o migrar a otros paradigmas, en caso de ser necesario.

Rendimiento en Operaciones de Lectura/Escritura

En nuestro sistema utilizamos **Casos de Uso** como "Visualizar Reporte de Mediciones" (CU-013) y "Visualizar Datos Ambientales" (CU-004), allí la velocidad es crítica.

Con MySQL la **Optimización de Consultas** es extremadamente eficiente en operaciones de lectura, lo cual es vital para el dashboard donde se visualizan promedios y alertas. Esto nos permite tener una transición fluida en el desarrollo, desde la Iteración 1 hasta la carga máxima de datos (Iteración 3), sin necesidad de re estructurar el modelo.

Por último, MYSQL nos da confiabilidad al cumplir con las propiedades **ACID** (Atomicidad, Consistencia, Aislamiento y Durabilidad), garantizando que las transacciones de las mediciones se procesen de forma segura, evitando la pérdida de información crítica durante fallos del sistema.

Modelo Lógico de la BD

Figura 55

Modelo de la BD

Tablas Maestras

Las tablas Maestras son las siguientes:

rol: Tipo de usuarios del sistema.

persona: Guarda los usuarios registrados y se les asocia un rol.

sensor: Contiene los datos principales de los sensores ambientales registrados.

configuracion_sensor: Guarda los parámetros de configuración y umbrales asociados a cada sensor, sirve para generar las alertas.

Tablas Transaccionales

sesion: Registra las sesiones de los usuarios al sistema.

medicion_resumen: Almacena las mediciones agregadas de los sensores en un periodo de tiempo.

alerta: Registra las alertas generadas cuando los valores de las mediciones superan un umbral o están por debajo del minimo permitido.

evento_sistema: Guarda los eventos para la auditoría del sistema, como inicios de sesión, cambios de configuración.

Implementación de Librerías y Dependencias

Las librerías y dependencias utilizadas en la elaboración del proyecto se detallan a continuación.

Backend

Para el desarrollo de la arquitectura, se optó por herramientas que permitan operaciones asíncronas y conexiones persistentes. Las herramientas seleccionadas facilitan la comunicación bidireccional con los sensores y aseguran fluidez con la base de datos. Asegurando el flujo de información desde las fuentes de datos hasta el tablero.

Figura 56

Librerías Utilizadas en el Backend

Frontend

La interfaz de usuario se diseñó priorizando la navegación fluida y visualización de datos de manera dinámica. Para lograr un tablero se integraron librerías modernas que permiten transformar los datos crudos del backend en información visual.

A continuación, se detallan las dependencias clave utilizadas para la construcción de la interfaz

Figura 57

***Librerías** Utilizadas en el Frontend*

Figura 58

Comunicación entre librerías – Backend

Frontend desarrollado con React, está optimizado con Vite. La interfaz utiliza React- Router-Dom para la navegación sin recargas y Chart.js para la visualización gráfica de las mediciones de los sensores. El Backend, orquestado por

FastAPI y ejecutado sobre el servidor ASGI Uvicorn, gestiona las peticiones y la comunicación bidireccional a través de WebSockets.

La capa de servicios orquestadora interactúa con:

Gestión de Datos: Utiliza SQLAlchemy para la persistencia en la Base de Datos y Redis para el manejo de caché mediante un RedisManager.

El componente WebSocketSource actúa como puente para recibir DataJson desde la Fuente de Datos externa.

Servidor y Framework: Uvicorn actúa como servidor ASGI para ejecutar FastAPI, mientras que Pydantic se encarga de validar los esquemas de datos de entrada.

Seguridad: Se implementa Bcrypt para el cifrado de información sensible.

Driver de Base de Datos:

Aiomysql: Driver asíncrono utilizado específicamente para recibir datos de sensores vía WebSockets sin bloquear el hilo principal del sistema.

PyMySQL: Driver síncrono que sirve como motor estándar para SQLAlchemy, encargado de tareas de seguridad y acceso secuencial a los datos.

Codificación del Front End

Se mostrarán las pantallas principales:

Figura 59

Dashboard principal

Figura 60

Componente Gráfico

Figura 61

Componente del Medidor de valores

Figura 62

Centraliza el estado global de los sensores

Programación en el Backend

En el siguiente diagrama se visualiza por cada Capa las clases implementadas para la obtención de datos de diferentes fuentes, la persistencia de datos y la re transmisión de la información (broadcast)

Figura 63

Tabla con las Clases correspondientes por Capa

Detalle de Clases por Capa

1. Capa de Negocio

SensorService: Funciones para procesar mediciones, calcular promedios y gestionar el registro de sensores.

AuthService: Lógica para validación de credenciales, generación de tokens y manejo de sesiones.

PersonaService: Lógica para la administración de usuarios y roles.

2. Capa de Datos

Modelos SQLAlchemy: Define las entidades como "Persona", "Sensor", "MedicionResumen", etc. para mapearlas de objetos a relacional.

Schemas Pydantic: Clases para validación de entrada/salida de datos (DTOs).

Redis Client: Objeto encargado de la comunicación con la base de datos en memoria para acceso rápido.

3. Capa de Integración

API Routers: Controladores que exponen los servicios vía HTTP.

sensor_sht3x_strategy: Script encargado de traducir los datos recibidos por WebSockets y generar la estrategia concreta para el sensor.

Middleware: Manejo de CORS.

Codificación del Back End

Figura 64

Parte de la codificación de la clase SensorService

Figura 65

Parte de la codificación de la clase WebSocketOrchestrator

Figura 66

Parte de la codificación del archivo sensor_router.py

Listado de Consultas y Reportes

Para poder realizar los diferentes reportes y consultas en la base de datos se utilizó los siguientes elementos con el fin de obtener información de manera efectiva, y poder utilizarla en el frontend.

Figura 67

Tabla con las consultas y Reportes

Figura 68

Lista de Reportes del sistema

NRO NOMBRE DESCRIPCIÓN 1 Auditoría de Actividad Registro de acciones. 2 Alertas del Sistema Historial de notificaciones de umbrales 3 Alertas Detalladas Informe de alertas por tipo de sensor.

4 Medición Resumen

Reporte de promedios, máximos y mínimos técnicos.

5 Historial de Mediciones Últimas 100 lecturas

Codificación de Mantenedores (CRUD)

El sistema debió utilizar un CRUD de Usuarios, pero para los Sensores solo lectura y actualización ya que no los crearemos, sino que tomaremos los que vienen en la fuente de datos emitidos a través del WebSocket, a continuación, se detallan los mantenedores.

Figura 69

Tabla de Mantenedores

Procesos Transaccionales

Los Casos de Uso del sistema, se transforman en procesos de software, a continuación, se describe la funcionalidad que cumple cada uno.

Figura70

Tabla con los procesos del sistema por cada Caso de Uso

Capítulo V

PRUEBAS DE CALIDAD DE SOFTWARE

El proceso de aseguramiento de la calidad no es una etapa que ejecutamos al final del proyecto, sino que es el eje que permite transformar el código en soluciones que aportan valor al usuario final. La calidad se define de forma sencilla: significa aportar valor a las personas interesadas en nuestro producto. Sin embargo, alcanzar esta satisfacción, es compleja, debido a que la calidad es subjetiva; lo que hoy se considera un sistema eficiente y seguro, mañana podría no serlo según cambien las expectativas y quien lo utiliza.

En nuestro proyecto, las pruebas de software actúan como el marco de verificación para garantizar que el sistema no solo funcione, sino que cumpla con las funcionalidades más importantes, como la seguridad, el rendimiento y la usabilidad. El objetivo central del testing es agregar calidad al producto en cada iteración, identificando desviaciones de manera temprana para asegurar que el software final sea exactamente lo que se solicitó. Es nuestra forma de validar que estamos entregando una herramienta útil y confiable para el monitoreo de datos ambientales.

Pruebas Unitarias

Estas pruebas tienen como finalidad la ejecución de componentes de manera aislada, para verificar que cada elemento por separado se ejecute correctamente. Estas pruebas son llevadas a cabo por el equipo de desarrollo.

Figura71

Tabla con las pruebas Unitarias

Pruebas Integrales

Estas pruebas se centran en verificar el correcto funcionamiento entre interfaces y sus

interacciones con los componentes.

Figura72

Tabla con las pruebas Integrales

Otros (Casos de Prueba)

Caso de Prueba

De acuerdo con el Estándar IEEE 610 (1990) un caso de prueba se define como:

> "Un *conjunto de entradas de prueba, condiciones de ejecución, y resultados esperados desarrollados con un objetivo particular, tal como el de ejercitar un camino en particular de un programa o el verificar que cumple con un requerimiento específico."*

Los Casos de Prueba definen las validaciones que realizaremos en nuestro sistema, son la guía que determina que cosas debe probar el Tester, son la referencia de ejecución.

A continuación, se presentan los Casos de Prueba Funcionales:

Figura73

Tabla con los Casos de Prueba por cada Caso de Uso

Evidencias de algunos Casos de Prueba

El sistema ha sido probado y verificado cada una de sus funcionalidades por lo que podemos asegurar que se presenta un software con alto nivel de calidad, a continuación, presentaré algunas evidencias de las pruebas realizadas.

Figura74

Evidencia OK del CP Configuración de Umbrales

Figura74

Evidencias del Casos de Prueba 27: Configuración de Umbrales

Figura75

Evidencias del Casos de Prueba 28: Configuración de Umbrales

Figura76

Evidencias del Casos de Prueba: Reporte de Alertas

Figura77

Evidencias del Casos de Prueba: Reporte de Alertas

Capítulo VI

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

Se ha llegado a concluir un proyecto que implementa un sistema de monitoreo de sensores, el mismo integra una tecnología asincrona (ASGI) la cual nos permite administrar en "semi" tiempo real el flujo de datos desde diferentes fuentes.

El desarrollo del presente proyecto implicó un alto nivel de exigencia técnica, ya que fue necesario incorporar y dominar nuevas tecnologías fuera del área de conocimiento inicial. Este proceso incluyó el aprendizaje sobre sensores IoT, microcontroladores y los mecanismos de interconexión entre el hardware y el backend, así como la transmisión y procesamiento de los datos generados en tiempo casi real.

Para poder implementar la solución, se diseñó una arquitectura de software escalable y extensible, destacando el uso del patrón de diseño Strategy. Este enfoque permite que el sistema procese los datos de forma uniforme, independientemente de su origen. De esta manera, es posible integrar nuevas fuentes de información, como colas MQTT, APIs externas, Kafka u otros mecanismos de mensajería, implementando la estrategia concreta bajo la interfaz definida. El sistema reconoce automáticamente las nuevas fuentes y envía la información mediante broadcast a los distintos clientes conectados a través de WebSockets.

La arquitectura implementada, proporciona un alto grado de transparencia y desacoplamiento entre el backend y el frontend, ya que nos permite re diseñar nuevas interfaces, incorporando nuevos gráficos, o reportes, sin necesidad de modificar la lógica de negocio del sistema.

Finalmente, se llevaron a cabo las pruebas de software correspondientes, validando la calidad y confiabilidad del sistema. Las pruebas incluyeron tanto el consumo de datos generados de manera aleatoria a través de una API, como la integración con sensores IoT reales (SHT3x) controlados por el microcontrolador ESP32. Los resultados obtenidos confirmaron el correcto funcionamiento del sistema en escenarios reales, garantizando su estabilidad al trabajar con dispositivos de medición de alta calidad.

RECOMENDACIONES

Se recomienda desarrollar la conexión una Api de clima externa para poder visualizar los datos ambientales de manera más rápida.

Implementar un Módulo de Análisis por IA: Se recomienda integrar un motor de Inteligencia Artificial para procesar los parámetros recolectados por los sensores. El objetivo es que genere diagnósticos inmediatos y sugiera acciones correctivas inteligentes.

Protocolo de Mantenimiento de sensores IoT: Es deseable establecer un programa de revisión técnica (podría ser trimestral), de los componentes físicos IoT, debido al deterioro del hardware en ambientes de alta humedad.

Gestión de Datos Masivos (Big Data): El crecimiento y volumen de las mediciones es exponencial, y la información almacenada puede ser insostenible, se sugiere implementar rutinas automatizadas que consoliden los datos históricos, permitiendo liberar la carga del sistema sin perder la trazabilidad histórica.

Mejora en la Capa de Persistencia: Para tener un mejor rendimiento del sistema, se propone la migración de la base de datos a MongoDB esto generaría robustes al ser un motor que destaca en la gestión de datos no estructurados. Esta migración garantizaría el crecimiento optimo del proyecto en la utilización de muchos sensores ambientales.

REFERENCIAS BIBLIOGRAFICAS (Referencias según APA)

Abrial, J. R. (1974). Data semánticas. En J. W. Klimbie & K. L. Koffeman (Eds.), *Data base management.* North-Holland.

Anderson, D. J. (2010). *Kanban: Successful evolutionary change for your technology business.* Blue Hole Press.

Anderson, D. J., & Carmichael, A. (2016). *Essential Kanban condensed* (Digital ed.). Blue Hole Press.

Autentia. (s. f.). *Software design: Principios y patrones del desarrollo de software. Primera parte: Guía para directivos y técnicos* (Vol. 1). Autentia. https://www.autentia.com/libros/

5, 16, 17

Castellano Lendínez, L. (2019). Kanban: Metodología para aumentar la eficiencia de los procesos. *3C Tecnología. Glosas de innovación aplicadas a la pyme.* https://doi.org/10.17993/3ctecno/2019.v8n1e29/30-41

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson.

FastAPI Documentation. (2024). *FastAPI: High performance, easy to learn, fast to code, ready for production.* https://fastapi.tiangolo.com/

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* [Tesis doctoral, University of California, Irvine].

Fowler, M. (2018). *Patterns of enterprise application architecture.* Addison-Wesley.

FourWeekMBA. (s. f.). *Kanban framework in a nutshell.* https://fourweekmba.com/kanban

Freeman, E., & Robson, E. (2020). *Head first design patterns: Building extensible and maintainable object-oriented software.* O'Reilly Media.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley.

International Organization for Standardization. (2015). *ISO 9001:2015 quality management systems — Requirements.* ISO.

International Software Testing Qualifications Board. (2023). *ISTQB® certified tester foundation level syllabus* (Version 4.0). https://www.istqb.org

Lean Six Sigma Training. (s. f.). *Kanban tools are now used by Pixar, Spotify and Zara.* https://www.lean6sigmatraining.co.uk/kanban-tools-are-now-used-by-pixar-spotify- and-zara

Lucide-React. (2024). *Lucide: Beautiful & consistent icons made by the community.* https://lucide.dev/

Microsoft Learn. (s. f.). *What is Kanban?* Microsoft. https://learn.microsoft.com/en- us/azure/devops/learn/agile/what-is-kanban

MySQL. (2024). *MySQL 8.0 reference manual.* Oracle Corporation.

https://dev.mysql.com/doc/refman/8.0/en/

Porter, M. E. (1980). *Competitive strategy: Techniques for analyzing industries and competitors.* Free Press.

Project Management Institute. (2021). *A guide to the project management body of knowledge (PMBOK® Guide)* (7th ed.). PMI.

Python Software Foundation. (2024). *Asyncio: Asynchronous I/O.* https://docs.python.org/3/library/asyncio.html

React Documentation. (2024). *React: The library for web and native user interfaces.* Meta Platforms, Inc. https://react.dev/

Refactoring.guru. (s. f.). *Patrón Strategy (Estrategia).* https://refactoring.guru/es/design-patterns/strategy

Sánchez, C. (2020, 29 de enero). *Figuras.* Normas APA (7ma edición). https://normas- apa.org/estructura/figuras/

Sapag Chain, N., Sapag Chain, R., & Sapag Puelma, J. M. (2014). *Preparación y evaluación de proyectos* (6ta ed.). McGraw-Hill.

Scaled Agile, Inc. (2021). *Weighted shortest job first (WSJF).* En SAFe Lean-Agile glossary.

SQLAlchemy Documentation. (2024). *SQLAlchemy: The Python SQL toolkit and object relational mapper.* https://www.sqlalchemy.org/

Toledo Rodríguez, F., Curcio, A., & Scuoteguazza, G. (2014). *Introducción a las pruebas de sistemas de información.*

Abstracta.

Vite. (2024). *Vite: Next generation frontend tooling.* https://vitejs.dev/

Glosario de Términos

ASGI (Asynchronous Server Gateway Interface): Especificación que permite a los servidores Python manejar múltiples conexiones simultáneas, como WebSockets.

Cardinalidad: Relación de instancias entre una entidad y otra.

Dashboard: Interfaz gráfica que resume y visualiza los indicadores.

ESP32: El ESP32 es un microcontrolador de bajo consumo desarrollado por Espressif Systems, diseñado para proyectos de Internet de las Cosas (IoT). Integra conectividad Wii-Fi y Bluetooth.

IoT: Del inglés “Internet of Things” (internet de las cosas), son dispositivos físicos con sensores y software que permiten la recopilación de datos y mediante internet su utilización.

ORM (Object-Relational Mapping): Es una técnica de programación que permite mapear entidades que son objetos a una base de datos relacional.

WebSocket: Protocolo de comunicación que proporciona canales de comunicación bidireccional sobre una única conexión TCP.

ANEXOS

Anexo 1: Manual de Usuario

Anexo 2: Conexión con Sensores Iot físicos

El presente proyecto emplea sensores IoT de temperatura y humedad para la toma de variables ambientales. El dispositivo utiliza conectividad inalámbrica mediante Wi-Fi para la transmisión de datos hacia un servidor local. Dicha comunicación se efectúa a través del protocolo WebSocket, garantizando el envío de información en tiempo casi real con el backend. Este anexo detalla los procedimientos de configuración, el sensor utilizado y las pruebas de validación realizadas para asegurar la correcta lectura de las mediciones.

Componentes utilizados:

Microcontrolador ESP32

El ESP32 es un microcontrolador diseñado por Espressif Systems, cuyo propósito principal es la integración con aplicaciones móviles y soluciones de Internet de las Cosas (IoT). Se trata de un chip de bajo consumo energético, capaz de integrar y gestionar múltiples sensores IoT de forma simultánea, lo que lo convierte en una solución versátil para proyectos embebidos.

Entre sus principales características se destacan: Wi-Fi 802.11 b/g/n integrado Bluetooth v4.2 (BR/EDR y BLE) Procesador Xtensa® LX6 de 32 bits, disponible en versión single-core o dual- core Frecuencia de reloj de hasta 240 MHz 448 KB de ROM 520 KB de SRAM Soporte para memoria externa (Flash y PSRAM) Hasta 34 pines GPIO programables Convertidores ADC y DAC integrados Interfaces de comunicación: UART, SPI, I2C, I2S, PWM, SDIO Aceleradores de hardware para criptografía (AES, SHA, RSA, TRNG) Relojes internos de alta precisión (hasta 8 MHz) Múltiples modos de ahorro de energía (sleep, deep sleep, hibernación)

Gracias a su bajo consumo energético, alta velocidad de respuesta y amplias capacidades de conectividad, el ESP32 es una pieza ideal para el desarrollo de proyectos IoT tales como: Casas inteligentes, Agricultura inteligente, Monitoreo ambiental, Reconocimiento de voz e imágenes, Cámaras de video y sistemas de streaming, Dispositivos portátiles, etc.

El ESP32 se adapta a diversos escenarios tecnológicos, debido a su equilibro, optimo rendimiento, bajo costo, consumo energético y escalabilidad.

Figura 1 Especificación de las interfaces de entrada y salida del *ESP32 DevKit V1*

Nota: Imagen tomada de Espressif Systems. (2024). *ESP32-WROOM-32 Datasheet* (v3.4). https://www.espressif.com/sites/default/files/documentation/esp32-wroom- 32_datasheet_en.pdf

Protoboard: Plataforma para probar diseños electrónicos, construir circuitos de manera

temporal de manera rápida y sin soldadura. La interconexión se logra conectando los

componentes en los agujeros y utilizando cables de conexión.

Figura 2

La siguiente imagen es un Protobard

Nota: imagen tomada de https://www.electromania.pe/producto/protoboard-transparente-de-

830-puntos/

Sensor SHT3X: Sensor de monitoreo de variables ambientales como Humedad y Temperatura

en tiempo real.

Figura3:

La siguiente imagen muestra un Sensor SHT3X

Nota: imagen tomada de https://theyosh.github.io/TerrariumPI/hardware/sensor/sensirion-

sht3x-sensor/

Cables Dupont (o cables puente): es un componente electrónico que crea conexiones

temporales y sin soldaduras.

Armado

Como podemos ver a continuación, en la Protoboard se ha conectado el microcontrolador **ESP32**

y el sensor **SHT3X,** con los cables puente (macho-macho), se han realizado las conexiones

mediante la siguiente tabla:

Figura 4:

La siguiente *tabla muestra las conexiones de los pines del SHT3X*

Figura 5:

Foto real del Proyecto, como queda la ESP32 conectada al sensor

Figura 6:

Foto del microcontrolador con los cables que conectan al sensor.

Figura7:

Vista del Sensor SHT3X

Pruebas configuraciones Previas

Para la realización de las pruebas, la placa debe ser conectada mediante un cable usb de 5 v tipo C, a nuestro pc de esta manera podemos saber a qué puerto se conecta y realizar la instalación de micro Python para nuestro proyecto

Figura8:

Administrador de dispositivos del pc, se visualiza el ESP32 en el puerto COM8

Thonny

Thonny es un **entorno de desarrollo integrado (IDE)** diseñado para programar en Python, en los microcontroladores como el **ESP32.** Posee una interfaz fácil de uso, es limpio no abruma al usuario de opciones, y gestiona paquetes, librerías de manera rápida y sencilla:

Figura9:

Vista del Programa Thonny ejecutando el programa que lee los datos del sensor físico

Programa para la Lectura de Datos y emisión mediante WebSocket

Los datos no se envían directamente al backend, ya que, en la arquitectura propuesta, no representa un escenario realista en sistemas productivos. Técnicamente es posible, pero implicaría acoplar la lógica del backend directamente al sensor. Es por eso por lo que se prefirió diseñar el sistema de forma desacoplada, permitiendo que el backend pueda consumir información de múltiples fuentes de datos, y no exclusivamente de un único dispositivo.

Es por ello que para leer los datos emitidos por el sensor, armamos un json y los enviamos a un servidor local, desde donde posteriormente son consumidos por el backend

Se programó el siguiente código:

Figura 10

Código fuente explicado parte 1

Figura 11

Código fuente explicado parte 2

Figura 12

Código fuente explicado parte 3

Código:

```
import network
import uasyncio as asyncio
from machine import I2C, Pin, unique_id
import json
import ubinascii
from async_websocket_client.ws import AsyncWebsocketClient as lo_client
# --- CONFIGURACIÓN ---
WIFI_SSID = 'USUARIO WI FI'
WIFI_PASS = 'CLAVE WI FI'
# URL del Puente
WS_URL = "ws://192.168.100.52:7777/sensor-real"
# Identificador Único
```

```
id_hex = ubinascii.hexlify(unique_id()).decode().upper()

MAC_ADR = ":".join([id_hex[i:i+2] for i in range(0, len(id_hex), 2)])

# Sensor SHT3x

i2c = I2C(0, scl=Pin(22), sda=Pin(21))
```



```
ADDR = 0x44

async def leer_sensor():

try:

i2c.writeto(ADDR, b'\x2C\x06')

await asyncio.sleep_ms(100)

data = i2c.readfrom(ADDR, 6)

# Fórmulas SHT3x

temp = round(-45 + (175 * ((data[0] << 8) | data[1]) / 65535), 2)

hum = round(100 * ((data[3] << 8) | data[4]) / 65535, 2)

return temp, hum

except:

return None, None

async def conectar_wifi():

wlan = network.WLAN(network.STA_IF)

wlan.active(True)

if not wlan.isconnected():

print("Conectando a WiFi...")

wlan.connect(WIFI_SSID, WIFI_PASS)

for _ in range(20):

if wlan.isconnected(): break
```

```
await asyncio.sleep(0.5)
return wlan.isconnected()
async def main():
while True:
if not await conectar_wifi():
await asyncio.sleep(5)
continue
ws = lo_client()
try:
print("Conectando al Puente:", WS_URL)
await ws.handshake(WS_URL)
print("Conectado")
while True:
t, h = await leer_sensor()
if t is not None:
payload = [
{"MAC": f"{MAC_ADR}_T", "Tipo": "Temperatura", "Valor": t},
{"MAC": f"{MAC_ADR}_H", "Tipo": "Humedad", "Valor": h}
]
await ws.send(json.dumps(payload))
print("Enviado:", payload)
await asyncio.sleep(2)
except Exception as e:
print("Reconectando...", e)
await asyncio.sleep(5)
# --- INICIO ---
try:
asyncio.run(main())
except:
print("Detenido")
```

Servidor

Como se explicó anteriormente utilizamos un servidor local para leer los datos emitidor por el ESP32 y luego el Backend los toma desde aquí.

El flujo es el siguiente:

El ESP32 se conecta a la puerta /stream-datos. El responde “ok” y envía los datos recibidos

El **ESP32** se conecta a la puerta /sensor-real y envía por ejemplo: *"Temperatura: 25°C".*

El **Servidor** recibe esos "25°C" y, como ya sabe quién está conectado a él, le re envía la información recibida del esp32.

Figura 13

Código fuente del servidor local explicado parte 1

Figura 14

Código fuente del servidor local explicado parte 2

Código fuente del servidor:

```
import asyncio

import fastapi

from fastapi import WebSocket, WebSocketDisconnect

"""Librerias necesarias para crear un servidor web rápido que permite conexiones

WebSockets"""

app = FastAPI() """Aquí creamos el "servidor"."""

# Referencia global al socket del backend

backend_ws = None

@app.websocket("/sensor-real")
```

"""Puerta de entrada llamada "sensor-real".

Solo el ESP32 puede entrar por aca."""

async def endpoint_esp32(websocket: WebSocket):

#Datos del esp32

global backend_ws

await websocket.accept() """acepta la conexión del ESP32."""

print("EsP32 conectada")

try:

"""Es un ciclo infinito. Mientras el ESP32 esté encendido, el servidor estará escuchando."""

while True:

data = await websocket.receive_text()

"""SI EL BACKEND ESTÁ CONECTADO LE PASAMOS LA DATA RECIBIDA DEL ESP32"""

if backend_ws:

await backend_ws.send_text(data) """Si la respuesta es sí, el servidor la bola de

datos"""

except WebSocketDisconnect:

"""si por alun motivo se desconecta el esp32 el programa informa y termina"""

print("Esp32 esconectado")

@app.websocket("/stream-datos")

"""se Define otra puerta distinta "stream-datos", aca es donde se conecta el back"""

async def endpoint_backend(websocket: WebSocket):

global backend_ws

await websocket.accept() """aceptamos la conexión del backend"""

```
backend_ws = websocket """el servidor guarda la conexión"""

print("Backend vinculado")

try:

while True:

# Mantenemos la conexión abierta esperando

await websocket.receive_text()

except WebSocketDisconnect:

backend_ws = None

print("Backend desvinculado")

if __name__ == "__main__":

import uvicorn

uvicorn.run(app, host="0.0.0.0", port=7777)
```

Figura 15: ESP32 emitiendo datos a la izquierda y el servidor escuchando a la derecha

Conexión con el Backend y Explicación del Patron Strategy

1. La Clase Base (DataStrategy): "El Contrato"

Como se ha mencionado en el Proyecto, utilizamos el patrón **Strategy** con la finalidad de poder conectarnos a diferentes fuentes de datos, independientemente de cuál sea (un api, sensores físicos como es nuestro proyecto, cola kafta, mqtt, etc), para ellos creamos en primer lugar la “Estrategia Base”.

Figura 16:

Clase base de la estrategia ubicada en websocket_logic/strategies/base_strategy.py

Esta clase es una **clase base abstracta** (indicada por ABC), funciona como un **contrato** o un "esqueleto" que define cómo deben comportarse todas las estrategias de manejo de datos que tengamos.

2. La Estrategia Concreta

Cómo no sabemos desde donde recibiremos los datos, solo necesitamos saber que deben ser en formato JSON, y estos pueden ser distintos. Debido a ello, en la clase **DataStrategy** obligamos a que, sin importar de dónde venga el dato, siempre tengamos un método llamado procesar, el cual limpie y deje uniforme la información para que el Frontend no tenga errores. En la clase concreta **sensor_sht3x_strategy.py** recibimos los datos tal cual los envía el ESP32 y los transformamos a un formato que el Frontend pueda entender sin errores.

Figura 16:

*Clase concreta para procesar el **sensor,** dentro de*

websocket_logic/strategies/sensor_sht3x_strategy.py

4. El Orquestador Ahora que tenemos la estrategia que procesa la fuente de datos, debemos explicar quien la usa. **El Orquestador.** Esta clase no sabe leer sensores, solo sabe gestionar las conexiones con el websocket y una vez que tiene los datos limpios (proporcionados por la estrategia), los re envía (broadcast) a los que están escuchando al backend. **Figura 18** *Clase Orquestador*

Uso de la estrategia concreta. El Orquestador NO sabe de donde provienen los datos, sino que

le indica a la estrategia seteada que los procese (los limpia) y luego hace el broadcas

Figura 19 *Orquestador emisión de datos*

5. Seteo de estrategia

Dentro del enviroment del backend declaramos la fuente de datos

Figura 20

*Archivo **.env** enviroment de python*

Dentro del archivo main seleccionamos la estrategia por defecto acorde a la que esté indicada en el archivo de

enviroment

Figura 21

Fragmento del main

Luego instanciamos al Orquestados y le indicamos la estrategia a utilizar.

Beneficios

• **Desacoplamiento:** El backend no necesita saber cómo calcular los datos o nombres que usa el sensor, api, u otra fuente, solo recibe el "paquete" y la estrategia lo devuelve listo. • **Estandariza el Formato:** Si en el futuro se decide cambiar el nombre del campo "medición", “valor”, “nombre” a otro, solo lo cambiamos en el archivo de la Estrategia, y automáticamente todo tu sistema se actualiza. • **Manejo de Errores:** Si la fuente de datos envía datos corruptos o mal formados podemos identificarlos y corregirlos, el sistema devolverá una lista vacía. • **Escalabilidad:** La clase Orquestador no sabe, y no le interesa, qué sensor está leyendo. Solo sabe que la estrategia tiene un método .procesar(). Si se cambia el sensor, o se conecta a otros dispositivos, no hay que reprogramar el Orquestador, solamente creamos una nueva estrategia y se la "inyecta".

Referencias:

Espressif Systems. (2023). ESP32 series datasheet (Version 5.2): 2.4 GHz Wi-Fi + Bluetooth® + Bluetooth LE SoC. https://www.espressif.com

Fette, I., & Melnikov, A. (2011). The WebSocket protocol (RFC 6455). Internet Engineering Task Force (IETF). https://www.rfc-editor.org/rfc/rfc6455

Kuzmin, A. (2024). Thonny: Python IDE for beginners. https://thonny.org/

MicroPython Project. (2024). MicroPython documentation (v1.22). https://docs.micropython.org/en/latest/

NXP Semiconductors. (2021). UM10204: I2C-bus specification and user manual (Version 7.0). https://www.nxp.com/docs/en/user-guide/UM10204.pdf

Ramírez, T. (2024). FastAPI framework: High performance, easy to learn, fast to code, ready for production. https://fastapi.tiangolo.com/

Sensirion AG. (2024). Datasheet SHT3x-DIS: Humidity and temperature sensor (Version 2.0). https://sensirion.com/media/documents/213E6A3B/63A5A569/Datasheet_SHT3x_DIS.pdf

YouTube. (s. f.). [Video explicativo sobre ESP32 y MicroPython]. https://www.youtube.com/watch?v=ODITzqixBy8