Diseño de una interfaz gráfica para la visualización remota en fermentadores de café mediante IoT

Juan Esteban Mendez Diaz

Jhonatan Molina Muñoz

Asesor:

Janier Andres Ballesteros Rincon

Programa de Ingeniería de Sistemas

Escuela de Ciencias Básicas Tecnología e Ingeniería

Universidad Nacional Abierta y a Distancia

24 de mayo de 2025

Dedicatoria

Este proyecto fue dedicado con gratitud y amor a nuestras familias, quienes fueron el pilar fundamental durante todo el proceso académico y personal. A nuestras madres y padres, por enseñarnos el valor del esfuerzo y la perseverancia; a nuestros hermanos, por su apoyo constante, y a todos aquellos que creyeron en nuestra capacidad de alcanzar esta meta. También se dedicó este trabajo a los caficultores colombianos, especialmente a los del Huila, cuya labor representa la esencia del esfuerzo rural y la riqueza cultural del país. Su realidad inspiró el desarrollo de una herramienta pensada para mejorar su calidad de vida mediante el uso de la tecnología.

Agradecimientos

Este trabajo de grado fue posible gracias al apoyo incondicional de diversas personas e instituciones que acompañaron el proceso desde sus inicios hasta su culminación. En primer lugar, se agradeció profundamente a los tutores asignado, cuya guía y experiencia fueron fundamentales para orientar cada una de las fases del proyecto. De igual manera, se reconoció el respaldo recibido por parte de los docentes del programa de Ingeniería de Sistemas de la Universidad Nacional Abierta y a Distancia (UNAD), quienes brindaron herramientas conceptuales y técnicas esenciales para el desarrollo de esta investigación.

También se extendió un especial agradecimiento a los caficultores del departamento del Huila, quienes aportaron su conocimiento empírico sobre el proceso de fermentación del café, permitiendo contextualizar la solución tecnológica desarrollada. Agradecemos profundamente el apoyo emocional y moral de nuestras familias, quienes motivaron constantemente la culminación exitosa de este proyecto.

Resumen

El objetivo de este proyecto fue desarrollar una interfaz gráfica utilizando el lenguaje de programación C#, orientada al monitoreo del proceso de fermentación del café. La conexión entre los fermentadores y el sistema se estableció mediante tecnologías de Internet de las Cosas (iot), utilizando principalmente el protocolo de comunicación MQTT (Message Queuing Telemetry Transport), seleccionado por su eficiencia en la transmisión de datos en redes de bajo ancho de banda. Complementariamente, se implementó comunicación por puerto serie a través de dispositivos como la placa Arduino nano ESP32, a los cuales se conectaron sensores encargados de registrar variables críticas como la temperatura, el ph y la concentración de CO_2.

La interfaz se integró con una base de datos local desarrollada en SQL Server, encargada de almacenar y organizar la información recolectada, lo que permitió a los usuarios acceder a los datos en tiempo real y consultar registros históricos estructurados de forma eficiente.

El diseño de la aplicación se centró en la visualización intuitiva e interactiva de la información, incorporando gráficos dinámicos para facilitar la interpretación de los valores registrados. Además, se habilitó un sistema de notificaciones para alertar sobre posibles anomalías en los parámetros monitoreados.

La implementación del protocolo MQTT garantizó una comunicación ligera, eficaz y confiable entre los dispositivos remotos y la interfaz gráfica. Asimismo, el uso de estructuras de datos en lenguaje SQL permitió la creación y gestión eficiente de los registros.

Finalmente, se aplicaron principios de usabilidad y diseño centrado en el usuario (UX/UI), con el objetivo de ofrecer una experiencia de uso clara, accesible y funcional, adecuada al contexto técnico y operativo de los caficultores y operadores del sistema.

Palabras clave: Interfaz gráfica, iot, MQTT, fermentación del café, SQL Server, UX/UI.

Abstract

The objective of this project was to develop a graphical interface using the C# programming language, aimed at monitoring the coffee fermentation process. The connection between the fermenters and the system was established through Internet of Things (iot) technologies, primarily using the MQTT (Message Queuing Telemetry Transport) communication protocol, chosen for its efficiency in low-bandwidth data transmission environments. Additionally, serial communication was implemented via devices such as the ESP32 or Arduino Nano ESP32 boards, to which sensors were connected for recording critical variables such as temperature, ph, and CO_2 concentration.

The interface was integrated with a local database developed in SQL Server, responsible for storing and organizing the collected information. This allowed users to access real-time data and efficiently consult historical records.

The application design focused on intuitive and interactive data visualization, incorporating dynamic charts to facilitate the interpretation of recorded values. A notification system was also implemented to alert users of potential anomalies in the monitored parameters.

The use of MQTT ensured lightweight, reliable, and efficient communication between remote devices and the graphical interface. Moreover, SQL-based data structures enabled efficient creation and management of records.

Finally, usability and user-centered design (UX/UI) principles were applied to provide clear, accessible, and functional user experience, tailored to the technical and operational context of coffee growers and system operators.

Keywords: Graphical interface, iot, MQTT, coffee fermentation, SQL Server, UX/UI.

Tabla de Contenido

Lista de tablas

Lista de Figuras

Introducción

Como estudiantes de Ingeniería de Sistemas, hemos observado con gran interés el impacto que la tecnología puede tener en los procesos tradicionales del sector agrícola, especialmente en la producción de café. Este proyecto busca desarrollar una solución tecnológica que integrara conocimientos de programación, arquitectura de software, bases de datos y comunicaciones para optimizar el proceso de fermentación del café, una de las etapas más críticas en la determinación de la calidad final del producto.

La iniciativa consistió en diseñar e implementar de una interfaz gráfica —un dashboard— que permitirá la visualización en tiempo real de los valores cuantitativos y químicos monitoreados durante la fermentación, como son la temperatura, el ph y el nivel de dióxido de carbono (CO_2). Esta visualización se efectúa mediante la lectura de sensores conectados a un Microcontrolador Arduino Nano ESP32, que transmiten los datos usando el protocolo MQTT hacia una base de datos SQL Server, desde donde la interfaz desarrollada en C# los consulta, analiza y presenta de manera interactiva.

Se ha elegido este tema no solo por su relevancia técnica, sino también por su potencial social. Muchos pequeños y medianos caficultores carecen de herramientas tecnológicas que les permitan controlar con precisión sus procesos. Esta herramienta pretende ser una respuesta concreta a esa necesidad, aportando una solución de bajo costo, escalable y eficiente que contribuya al mejoramiento de la calidad del café colombiano.

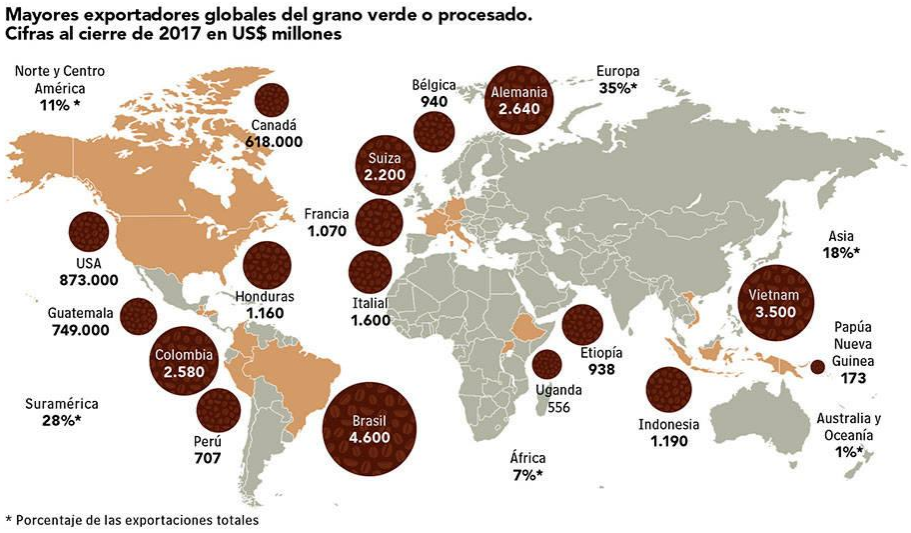
Árbol causa – efecto del problema

En el año 2024, los principales países productores de café a nivel mundial fueron Brasil, Colombia, Etiopía, Honduras y Vietnam, según datos reportados por Statista (Orús, 2025). Tal como se evidencia en las gráficas de la imagen 1, Colombia ocupó el cuarto lugar entre los mayores exportadores, con un total de USD 3.544,5 millones en exportaciones de café. Esta cifra representó un incremento respecto al año 2023, cuando el país ocupó la tercera posición con exportaciones valoradas en USD 2.792 millones (Fluctuante, 2024).

Datos de exportación de café a nivel mundial.

Figura 1

Mayores exportadores globales del grano verde o procesado



Nota. Principales países exportadores de café a nivel mundial en función del valor de las exportaciones en 2024. Fuente: Howmuch.net, con datos de la CIA 18 de septiembre de 2019.

A pesar de este aumento en el valor de exportación, el volumen exportado correspondió principalmente a café verde, lo que refleja un menor nivel de industrialización y valor agregado.

De acuerdo con la Federación Nacional de Cafeteros (Colombia, 2023), el 91% del café exportado fue en estado verde, mientras que solo el 9% correspondió a café procesado, específicamente “pergamino seco”. Esto indica que, aunque el volumen de producción es significativo, el bajo nivel de transformación limita las oportunidades de comercialización con mayor valor económico.

El departamento del Huila, reconocido como uno de los mayores productores de café del país (Hernán, Hugo, & Jakeline, 2020), enfrenta una paradoja: pese a su alta producción, también se ubicó en el duodécimo lugar entre los territorios con mayor índice de pobreza monetaria según el DANE (2024). Esta situación evidencia la necesidad de estrategias que impulsen la producción de café de alta calidad, con valor agregado y mayor capacidad de diferenciación en el mercado.

En este contexto, la implementación de un sistema de fermentación controlado mediante tecnologías iot permitió reproducir condiciones óptimas para mejorar la calidad del café. Esta solución favoreció a pequeños y medianos caficultores, al permitirles alcanzar estándares de calidad más altos y consistentes, lo cual representa una vía para reducir la pobreza en regiones productoras.

El uso de un fermentador inteligente, apoyado por sensores y monitoreo remoto a través de una interfaz gráfica desarrollada en C#, permitió estandarizar parámetros críticos como temperatura, ph y concentración de CO_2, los cuales han sido identificados como variables clave en el proceso de fermentación (Lanuza, Agurcia, Cornejo, & Matey, 2023). Gracias a esta automatización, fue posible observar en tiempo real la evolución del proceso, asegurando que los valores se mantuvieran dentro de los márgenes deseados, y promoviendo la producción de café con características sensoriales homogéneas.

Este control permitió no solo mejorar la calidad del producto final, sino también generar una mayor competitividad en el mercado internacional. A medida que se incrementó la capacidad de producir café de calidad superior, se incentivó su demanda, fortaleciendo el posicionamiento del café colombiano más allá de su volumen de exportación en estado verde.

Si bien las cifras de exportación continuaron en ascenso (Colombia, 2023), se concluyó que es prioritario enfocar los esfuerzos hacia la generación de valor agregado en origen. Exportar café con un mayor nivel de procesamiento podría aumentar los ingresos de los caficultores, favorecer la redistribución de ganancias dentro de la cadena productiva y contribuir a la reducción de los niveles de pobreza en las zonas rurales del país.

Árbol de problema

Figura 2

Árbol de problemas

Nota. Elaboración de un esquema de Árbol de Problemas, para la identificación de los efectos, problemas y causas de la implementación de esta interfaz gráfica. Foto: Fuente. Autoría Propia.

Planteamiento del problema

En muchas fincas cafeteras, el monitoreo de la fermentación se realiza manualmente, lo que impide identificar en tiempo real desviaciones críticas en parámetros como la temperatura o el ph. Esta ausencia de control puede derivar en una fermentación inadecuada, alterando la calidad del grano, reduciendo su valor comercial e incluso provocando pérdidas completas de lotes.

La necesidad surge de contextos rurales donde los caficultores, si bien tienen acceso a dispositivos de medición básicos, no cuentan con sistemas centralizados, conectados o automatizados. Además, las condiciones de conectividad en estos entornos exigen que la solución tecnológica sea ligera, eficiente y adaptable, requisitos que cumple el protocolo MQTT (HIVEMQ, 2024) al permitir la transmisión de datos incluso con ancho de banda limitado.

Gracias a la recopilación de datos de investigación se propone la implementación y desarrollo de una interfaz gráfica que se centralice en visualizar los datos de llegada, gestión de almacenamiento de datos por uso de terceros, la gestión de la implementación de alertas y presentar un diseño de intuitivo llamativo para la visualización del usuario descartando el desconocer valores técnicos avanzados.

Pregunta del Problema

¿Es necesario el diseño de una interfaz gráfica que permita la visualización de las variables en tiempo real de un fermentador de café mediante iot?

Sí, fue necesario diseñar una interfaz gráfica que permitiera la visualización en tiempo real de las variables críticas del proceso de fermentación del café mediante tecnologías iot. Esta necesidad surgió a partir de la falta de control preciso en procesos fermentativos realizados de forma empírica en muchas fincas cafeteras, lo cual derivaba en pérdidas económicas, lotes de baja calidad y ausencia de trazabilidad en el producto final (Lanuza, Agurcia, Cornejo & Matey, 2023).

La interfaz desarrollada permitió monitorear variables como el ph, la temperatura y la concentración de CO_2 de forma continua y remota, facilitando la toma de decisiones oportunas y reduciendo el margen de error durante el proceso. Además, su implementación en entornos rurales fue viable gracias al uso de tecnologías de bajo costo y protocolos eficientes como MQTT, que garantizaron la transmisión de datos aun en condiciones de baja conectividad (HIVEMQ, 2024; Karnouskos, 2011).

En conclusión, la implementación de este software en los sectores agrícola y tecnológico no solo enfoca a una necesidad técnica, sino que también busca integrar ambos ámbitos para ofrecer un servicio accesible, estable e intuitivo de usar. Esto permitiría brindar una herramienta esencial para los pequeños caficultores, ayudándoles a mejorar su productividad y aumentar su competitividad en el mercado.

Preguntas de investigación

Pregunta 1:

¿De qué manera una interfaz gráfica puede mejorar el monitoreo del proceso de fermentación del café en zonas rurales? La interfaz gráfica desarrollada permitió la visualización remota de variables críticas del proceso de fermentación, como la temperatura, el ph y el CO_2, lo que facilitó la toma de decisiones en tiempo real y redujo pérdidas por fermentaciones defectuosas.

Pregunta 2:

¿Qué tecnologías resultaron más adecuadas para implementar en contextos rurales con baja conectividad? Se empleó el protocolo MQTT por su bajo consumo de ancho de banda, junto con una arquitectura basada en C# y SQL Server, lo cual permitió operar de forma local y sincronizar datos cuando la conexión estuviera disponible (HIVEMQ, 2024; Karnouskos, 2011).

Pregunta 3:

¿Es viable implementar soluciones de monitoreo de bajo costo para pequeños caficultores? Sí. La solución desarrollada demostró ser técnicamente viable y económica, usando componente de bajo costo el Microcontrolador Arduino Nano ESP32 entorno de desarrollo Arduino IDE, además de software libre y código propio, permitiendo su replicabilidad en otras zonas cafetaleras.

Pregunta 4:

¿Qué impacto tuvo el sistema desarrollado sobre la calidad del proceso productivo? El sistema permitió estandarizar el proceso de fermentación mediante la monitorización continua de parámetros, lo cual mejoró la consistencia del producto final y ofreció a los productores una herramienta útil para el control de calidad.

Hipótesis

Hipótesis General:

Si se desarrolla una interfaz gráfica conectada a sensores mediante tecnologías IoT, entonces será posible monitorear en tiempo real el proceso de fermentación del café, mejorando la calidad del producto y reduciendo pérdidas asociadas a errores en dicho proceso.

Resultados obtenidos:

La hipótesis fue validada, ya que la implementación de la interfaz fermenttrack permitió una visualización continua y remota de los datos, generó alertas ante anomalías, y ayudó a mantener los parámetros de fermentación dentro de los rangos óptimos. Esto se tradujo en un mayor control del proceso y una mejor calidad del café fermentado.

Análisis y Limitaciones

Durante las últimas décadas, la industria cafetera ha evolucionado significativamente, impulsada por la necesidad de mejorar la calidad del grano y agregar valor a la cadena productiva. En Colombia, si bien el café es uno de los principales productos de exportación, la mayoría del grano exportado es café verde sin transformación (Colombia, 2023). Esta situación ha limitado el desarrollo de procesos tecnificados en zonas rurales, donde la fermentación del café continúa realizándose de forma empírica.

Diversas investigaciones señalaron que variables como la temperatura, el ph y la concentración de dióxido de carbono (CO_2) son determinantes en la calidad sensorial del café (Lanuza, Agurcia, Cornejo & Matey, 2023). A pesar de ello, los pequeños caficultores no contaban con sistemas automatizados que les permitieran monitorear estas variables en tiempo real, principalmente debido a limitaciones de conectividad, costo y acceso a tecnología.

Frente a este panorama, surgieron propuestas tecnológicas basadas en iot, como plataformas desarrolladas por empresas como cropin en India y Agrosmart en Brasil (Agrosmart, 2023; cropin, 2022), que, si bien ofrecieron soluciones robustas, no se ajustaban a las condiciones locales ni eran accesibles para pequeños productores. En Colombia, iniciativas como agroapp del ICA se enfocaron más en el diagnóstico agrícola que en el control en tiempo real.

Este trabajo retomó estos antecedentes para diseñar un sistema enfocado en los fermentadores de café, integrando sensores, microcontroladores y una interfaz gráfica amigable, con capacidad de operar en entornos de baja conectividad.

Justificación

El proceso de fermentación del café es fundamental para definir las propiedades organolépticas del grano. Sin embargo, en muchos contextos rurales, este proceso se realiza de manera empírica, sin monitoreo preciso, lo que deriva en lotes con calidades variables y pérdidas económicas. Como lo afirma enertic (2022), las tecnologías iot pueden reducir hasta en un 30% las pérdidas en la cadena de producción cafetera.

Al diseñar una interfaz gráfica que actúe como un panel de control visual —dashboard— los caficultores podrán observar en tiempo real las variables críticas del proceso y tomar decisiones informadas. Esta herramienta facilita la gestión de alertas, la consulta de datos históricos y la identificación de patrones que afectan la calidad del producto.

Desde la disciplina de la ingeniería de sistemas, este proyecto se fundamenta en principios de diseño de interfaces (UX/UI), arquitectura orientada a eventos, programación en C#, comunicación asíncrona y almacenamiento de datos estructurados. Esta integración de tecnologías me permitió poner en práctica competencias adquiridas a lo largo de la carrera y responder a una problemática real del entorno agrícola colombiano.

Objetivos

Objetivo general

Desarrollar una interfaz gráfica para el monitoreo de variables específicas en un fermentador de café mediante iot

Objetivos específicos

Diseñar una interfaz gráfica para visualizar las variables de temperatura, ph y CO2 de un fermentador de café mediante iot.

Gestionar una base de datos que permita conectar (GUI) del fermentador con la unidad de almacenamiento de información

Implementar un sistema GUI conectando con la base de datos que genere alertas y notificaciones sobre los cambios críticos presentes en el fermentador

Documentar la normatividad vigente que rigüe el funcionamiento y despliegue del software y el manual de usuario

Marco Teórico

Líneas y grupos de interés investigativo

Tabla 1

Relación de intereses investigativos, líneas y grupos de investigación

Intereses en ingeniería e investigación	**Línea de investigación y áreas temáticas**	**Grupo de investigación**
Visualización remota e interfaces graficas de usuario (GUI)	Gestión y visualización de variables de fermentación (PH, TEMP, CO2) Desarrollo de interfaces graficas en tiempo real Ingeniería de Software	GIDESTEC – Grupo de Investigación y Desarrollo en Sistemas Tecnológicos

Nota. Se realiza la gestión de definir la línea de investigación y sus áreas temáticas. Fuente. Autoría Propia

Fermentación del café:

El uso de tecnologías iot (Internet of Things) en la agroindustria ha revolucionado procesos tradicionales, permitiendo monitorear y controlar variables en tiempo real desde ubicaciones remotas. En el caso del café, uno de los cultivos más importantes de Colombia, la etapa de fermentación es crítica para la calidad del producto final. Sin embargo, muchos productores aún carecen de herramientas tecnológicas que les permitan visualizar y registrar adecuadamente estas variables.

Según Torres, Gómez y Castillo (2022), la implementación de plataformas digitales con sensores iot en procesos agrícolas mejora significativamente la eficiencia, calidad y trazabilidad de los productos. En el caso específico de los fermentadores, es fundamental monitorear parámetros como temperatura, ph, y concentración de CO_2, variables que afectan directamente la calidad sensorial del grano.

Molina y Herrera (2023) realizaron un estudio sobre el monitoreo de fermentación en microbeneficios cafeteros del Huila, concluyendo que la digitalización del proceso permite establecer estándares de calidad y evitar pérdidas. No obstante, también destacaron la necesidad de contar con interfaces gráficas simples, intuitivas y de bajo costo para facilitar su adopción por parte de pequeños caficultores.

La conectividad en zonas rurales sigue siendo un desafío. Según el mintic (2023), el 60% de las fincas cafeteras en Colombia tienen acceso intermitente o nulo a internet. Por ello, sistemas que usen tecnologías iot con almacenamiento local y sincronización remota, o bien sistemas híbridos, son esenciales.

Así, la visualización gráfica de datos desde una interfaz remota no solo mejora el proceso productivo, sino que también contribuye a la profesionalización del pequeño productor, promueve la toma de decisiones basada en datos y se alinea con las estrategias de transformación digital del agro colombiano.

Procesos, condiciones y criticas:

Monitoreo funcional y validación de requisitos:

Se realizó una evaluación continua de la interfaz gráfica desarrollada en C# para garantizar que cumpliera con los requisitos funcionales establecidos, tales como la visualización

en tiempo real de las variables críticas (temperatura, ph y CO_2), así como la integración con la base de datos local y los módulos de comunicación (puerto serie y MQTT).

Contextualización técnica de la interfaz para usuarios finales:

Se adaptaron los elementos visuales y textos informativos con base en las necesidades del público objetivo —caficultores y técnicos en zonas rurales—, asegurando que la terminología utilizada fuera clara, aplicable y contextualizada al entorno agrícola. Esto permitió una mayor apropiación de la herramienta digital, facilitando la toma de decisiones informadas en el proceso de fermentación.

Incorporación de elementos motivadores y de interacción:

Se evaluó la posibilidad de integrar elementos de gamificación —como indicadores de eficiencia, alertas positivas y seguimiento de lotes exitosos— con el objetivo de incentivar el uso continuo de la herramienta y promover el control de calidad como una práctica habitual. Esta estrategia estuvo alineada con enfoques de diseño motivacional en aplicaciones técnicas descritos por Baquero, Barbosa y Fernández (2023).

Estado del Arte

La digitalización de procesos agrícolas mediante tecnologías de monitoreo remoto ha cobrado gran relevancia en los últimos años, especialmente en sectores como la caficultura, donde la calidad del producto depende en gran medida del control de variables durante la postcosecha. A nivel internacional, existen plataformas como cropin (India), que integra herramientas de análisis predictivo, imágenes satelitales e iot para mejorar la trazabilidad y productividad de los cultivos (cropin, 2022). Por su parte, Agrosmart (Brasil) ofrece una solución completa de monitoreo climático, humedad del suelo y desarrollo fenológico, con el uso de inteligencia artificial y sensores distribuidos en campo (Agrosmart, 2023). Sin embargo, estas soluciones están orientadas a grandes productores y requieren infraestructura tecnológica robusta.

En el contexto colombiano, iniciativas como agroapp, desarrollada por el Instituto Colombiano Agropecuario (ICA), buscan asistir al productor con recomendaciones fitosanitarias, diagnósticos y alertas, aunque sin capacidades de integración con sensores físicos ni monitoreo en tiempo real (ICA, 2021). Esto evidencia una brecha en la disponibilidad de herramientas accesibles, específicas para la supervisión de procesos críticos como la fermentación del café, especialmente en zonas rurales con conectividad limitada.

Frente a este panorama, el sistema fermenttrack, propuesto en esta investigación, representa una solución innovadora al enfocarse en un entorno local, de bajo costo y adaptable a pequeños productores. A diferencia de otras plataformas, integra sensores físicos conectados a microcontroladores (Arduino Nano ESP32), con visualización gráfica en una interfaz desarrollada en C# y Visual Studio, usando SQL Server para el almacenamiento de datos y protocolo MQTT para la transmisión eficiente en redes de bajo ancho de banda. La interfaz

gráfica incorpora principios de diseño centrado en el usuario (ISO 9241-210, 2010), optimizando la interacción para caficultores sin experiencia tecnológica.

Esta comparación se evidencia que, si bien existen soluciones robustas a nivel global, pocas están diseñadas para atender las necesidades específicas del sector caficultor rural colombiano, lo que resalta la pertinencia y aporte diferencial de este proyecto.

Para ello diseñamos una tabla comparativa con soluciones similares que se presentan tanto en nuestro proyecto como en el exterior:

Tabla comparativa

Tabla 2

Digitalización de procesos agrícolas mediante tecnologías de IOT

Sistema / Proyecto	**Tecnología usada**	**Variables monitoreadas**	**Nivel de automatización**	**Público objetivo**
Fermenttrack (este trabajo)	C#, SQL Server, MQTT, ESP32, GUI	Ph, CO_2, Temperatura	Alto	Caficultores rurales
Cropin (India)	Web, iot, Cloud, Big Data	Clima, humedad, geoloc.	Medio	Grandes productores agrícolas
Agrosmart (Brasil)	Iot, Dashboard, IA	Clima, suelo, humedad	Alto	Agroindustria, investigación
Agroapp (Colombia – ICA)	App móvil	Diagnóstico de cultivo	Bajo	Caficultores y técnicos ICA

Nota. Se realiza una verificación de las tecnologías implementadas e investigadas. Fuente. Autoría Propia

Marco conceptual:

Fermentación del café:

Es un proceso bioquímico en el que los azúcares del mucílago del grano son descompuestos por microorganismos. Este proceso influye en el sabor, aroma y calidad del café. Requiere condiciones controladas de temperatura, tiempo, y ph.

IoT (Internet of Things):

Es un paradigma tecnológico que permite interconectar sensores y dispositivos a través de internet para recolectar, enviar y analizar datos en tiempo real. En agricultura, el iot permite crear sistemas inteligentes que monitorean variables ambientales y productivas sin intervención humana directa (Alvarez et al., 2021).

Visualización remota:

Hace referencia a la capacidad de observar en tiempo real datos de sensores o procesos desde una ubicación distinta a donde se produce el fenómeno. En el contexto del café, permite al caficultor tomar decisiones sin estar físicamente presente en la finca.

Interfaz gráfica de usuario (GUI):

Una GUI permite a los usuarios interactuar con sistemas mediante representaciones visuales como botones, gráficos y paneles. Para pequeños productores, una interfaz intuitiva y visual es esencial para la apropiación tecnológica (Gómez & Rivera, 2020).

Agricultura digital:

Es el uso de tecnologías de información para optimizar la producción agropecuaria. Incluye sensores, redes, inteligencia artificial, y visualización de datos, promoviendo prácticas sostenibles, eficientes y trazables.

Marco Legal

El desarrollo e implementación del proyecto fermenttrack, cuyo objetivo es monitorear y visualizar en tiempo real los valores cuantitativos enviados por sistemas embebidos en zonas cafeteras del Huila, se desarrolla dentro de un marco legal que prioriza la igualdad frente al acceso de la tecnología. Los siguientes puntos describen las consideraciones legales clave:

Derecho al desarrollo tecnológico en el sector rural:

Constitución Política de Colombia (1991):

El artículo 65 establece que la producción de alimentos gozará de especial protección del Estado. Este proyecto promueve mejoras en la calidad del café colombiano, directamente vinculado al desarrollo rural.

Ley 1341 de 2009 (Ley TIC):

Promueve el acceso, uso y apropiación de tecnologías de la información. La implementación de interfaces gráficas e IoT en zonas rurales promueve la democratización tecnológica y la inclusión digital.

Ley 1955 de 2019 (Plan Nacional de Desarrollo 2018–2022):

Incluye la transformación digital del agro como uno de sus pilares. Este proyecto se enmarca en dicha política al contribuir a la tecnificación del proceso de fermentación del café.

Protección de datos y tecnologías emergentes:

Ley 1581 de 2012 (Protección de Datos Personales):

El sistema debe garantizar la confidencialidad de los datos generados y su uso exclusivo para fines técnicos o de mejora productiva.

Resolución 254 de 2020 – minagricultura:

Promueve la implementación de tecnologías 4.0 en el campo, y respalda proyectos de automatización, trazabilidad y monitoreo remoto.

Marco Tecnológico

A partir del marco anterior, se proyecta el diseño de una interfaz gráfica conectada a sensores por iot para visualizar variables clave en tiempo real durante la fermentación del café. Esta interfaz deberá cumplir con los siguientes criterios:

Adaptabilidad: Accesible desde computadoras con baja resolución y con sistema operativo Windows.

Interactividad: Permitir la visualización de datos históricos y alertas automáticas.

Autonomía: Funcionamiento offline o semiconectado, con sincronización remota cuando haya acceso a red.

Facilidad de uso: GUI intuitiva pensada para caficultores sin conocimientos técnicos.

Escalabilidad: Posibilidad de extender a otros procesos postcosecha como secado o almacenamiento.

Metodología

El proyecto se desarrolló bajo un enfoque cuantitativo y aplicado, con un diseño cuasiexperimental orientado a validar la factibilidad técnica de un sistema de monitoreo remoto para fermentadores de café mediante tecnologías IoT. Se adoptó un modelo iterativo incremental, el cual permitió implementar, ajustar y evaluar progresiva cada uno de los distintos componentes (hardware, software y comunicación). Esta estrategia facilitó la integración modular de sensores, protocolos de transmisión y la interfaz gráfica, priorizando la usabilidad, confiabilidad de datos y estabilidad de la comunicación. (Karnouskos, 2020; ISO 9241-210, 2020).

Justificación metodológica:

Frente a otras alternativas como modelo de cascada (las cuales son menos flexibles para realizar justificaciones durante el desarrollo), el enfoque interactivo incremental resulto mucho mas adecuado para los componentes del hardware y software pues estos están en constante evolución, de la misma forma se tomo la selección de tecnologías (MQTT), enfocándose en su eficiencia en redes de bajo ancho de banda, mientras que el uso de C# y Windows Forms respondió con la necesidad de crear una interfaz robusta y accesible para entornos rurales con limitaciones técnicas.

La metodología está estructurada en las siguientes fases:

Análisis y definición de variables críticas.

En esta fase se realizó un análisis exhaustivo del proceso de fermentación del café, a fin de identificar las variables críticas que influyen directamente en la calidad del producto. Se consultó literatura técnica y científica (Lanuza et al., 2023) y se mantuvieron entrevistas con

productores locales del departamento del Huila. Esto permitió la identificación de variables clave en el proceso de la fermentación: pH, temperatura y concentración de CO_2. Adicionalmente, se analizaron los sensores disponibles en el mercado y su compatibilidad con los protocolos de comunicación IoT como MQTT y puertos serie.

Diseño e implementación del sistema IoT.

Se configuro un microcontrolador Arduino Nano ESP32 para la lectura y envío de datos desde los sensores de pH, temperatura y CO_2, junto con la implementación el protocolo MQTT para la transmisión eficiente de datos por las redes de bajo de ancho de banda hacia la interfaz gráfica, como complemento se habilito la comunicación por puerto serie para pruebas locales y entornos sin conectividad.

Desarrollo de la interfaz gráfica (FermentTrack).

Se realiza el diseño de la aplicación en C# con Windows Forms, organizada en las siguientes capas: Capa de presentación (UI), visualización en tiempo real, alertas y navegación. Capa de lógica de negocio, procesamiento de datos y validación de rangos. Capa de comunicación, integración con MQTT y puerto serie. Capa de datos, almacenamiento en SQL Server y consultas históricas.

Desarrollo de la interfaz gráfica (Fermentack)

Se diseño la aplicación en C# con Windows Forms, organizada en cuatro capaz:

- Capa de representación (UI): Visualización en tiempo real y navegación.
- Capa de lógica de negocio: Procesamiento y validación de rango crítico.
- Capa de comunicación: Integración con MQTT y puerto serial.
- Capa de datos: Almacenamiento en SQL Server (local) y consultas históricas.

Integración y pruebas del sistema.

Se realizan pruebas con datos simulados y reales para validar la transmisión, el almacenamiento y la visualización de la información, se analizó la latencia, estabilidad de conexión y precisión de los registros, mediante ensayos comparativos con valores de referencia para comprobar fidelidad de los datos visualizados.

Evaluación de usabilidad.

Para validar la usabilidad de la interfaz gráfica desarrollada, se aplica la System Usability Scale (SUS), una herramienta ampliamente utilizada para evaluar la percepción subjetiva de facilidad de uso de sistemas interactivos (Brooke, 1996), a un grupo de usuarios rurales identificando fortalezas y oportunidades de mejora en la interfaz (claridad, navegación, intención de uso).

Documentación y validación final

Se elabora manuales técnicos y recursos audiovisuales para usuarios finales, junto con un registro del proceso de desarrollo y recomendaciones para su implementación en entornos rurales con baja conectividad, asegurando la sostenibilidad y escalabilidad de la solución.

Desarrollo del proyecto

Diseño e implementación del sistema IoT

El diseño del sistema se fundamenta en una arquitectura de software orientada a la visualización remota de variables críticas en fermentadores de café mediante tecnologías IoT. Para el desarrollo de la interfaz gráfica se empleará el lenguaje de programación C# en el entorno de desarrollo Visual Studio, aprovechando sus capacidades para la creación de aplicaciones de escritorio interactivas y robustas. Esta interfaz permitirá el monitoreo en tiempo real de datos como temperatura, ph y concentración de CO_2, los cuales serán presentados mediante componentes gráficos como gráficas dinámicas y paneles de alerta.

La base de datos se gestionará mediante SQL Server, donde se almacena el historial de las mediciones, permitiendo el análisis posterior del comportamiento del proceso de fermentación. Esta base de datos se conectará a la aplicación C# mediante técnicas de acceso a datos como ADO.NET o Entity Framework, asegurando integridad, persistencia y trazabilidad de la información.

Para la transmisión de datos reales desde los sensores hacia la aplicación, se utilizará el protocolo MQTT (Message Queuing Telemetry Transport), a través del broker Mosquitto. Este protocolo ha sido seleccionado por su eficiencia en redes de bajo ancho de banda, su bajo consumo de recursos y su capacidad para operar de manera confiable en entornos con conectividad limitada, características propias de muchas zonas rurales cafetaleras. Mosquitto actuará como intermediario entre los dispositivos emisores (sensores iot) y el cliente receptor (interfaz en C#), asegurando la recepción oportuna de datos en tiempo real.

Adicionalmente, se realiza una fase de simulación de datos mediante una placa Arduino, que generará valores aleatorios para evaluar la estabilidad de la interfaz gráfica y verificar su

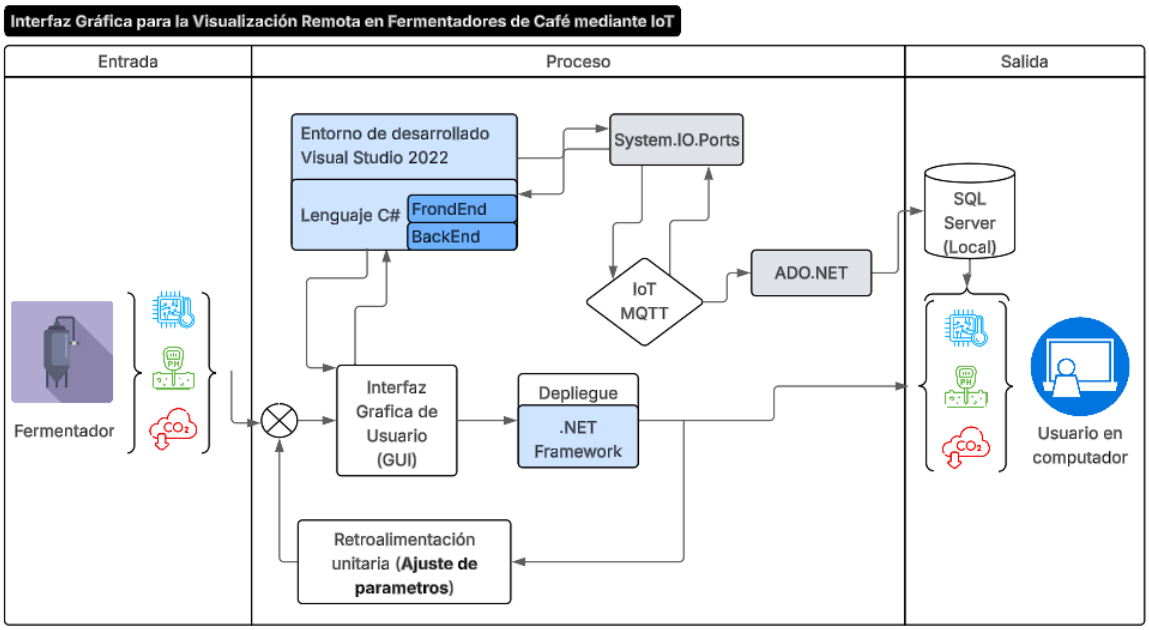
comportamiento ante posibles variaciones abruptas. Esta simulación permitirá pruebas de estrés y validación del sistema antes de su implementación con datos reales.

Desarrollo de la interfaz gráfica

Arquitectura del Sistema

Figura 3

Interfaz gráfica para la visualización remota en fermentadores de café mediante IoT



Nota. Diagrama de Arquitectura del software, principios del iot, sensores, conectividad, plataformas, etc. Fuente. Autoría Propia.

Arquitectura en Capas:

Figura 4

Diagrama de capas del proceso de los principios IoT

Nota. Implementación de diagrama de capas para el explicar los siguientes procesos: Principios del iot, Protocolos de comunicación iot (MQTT, HTTP, etc.), Tecnologías usadas.

Fuente. Autoría propia.

Para el desarrollo de esta interfaz gráfica se realizaron los siguientes pasos:

Capa de presentación (Interfaz gráfica - UI):

Tecnología: C# con Windows Forms (Visual Studio)

Responsabilidades:

- Mostrar los datos de fermentación en tiempo real y de forma histórica.
- Emitir alertas visuales ante valores fuera de rango.
- Interacción del usuario: login, cambio de fuente de datos, navegación.

Capa de lógica de negocio:

Tecnología: C# (.NET)

Responsabilidades:

- Procesamiento de los datos recibidos.
- Validación de rangos de seguridad.
- Conversión de datos para visualización.
- Control de conexión activa (MQTT o Serial).
- Control del ciclo de lectura y almacenamiento de datos.

Capa de comunicación:

Protocolos:

- MQTT (usando mqttnet)
- Serial Port (System.IO.Ports)

Responsabilidades:

- Escuchar mensajes desde un broker MQTT.
- Leer datos desde el puerto serial (Arduino Nano ESP32).
- Formatear y reenviar datos a la capa lógica.

Capa de acceso a datos:

Tecnología: ADO.NET + SQL Server

Estructura de BD:

- USUARIO (id_usuario, nombre, correo, contraseña) (futura actualización)
- FERMENTADOR (id_fermentador, nombre_alias, ubicacion, id_usuario)
- DATOS (id_dato, valor, fecha_hora, variable, id_fermentador)

Responsabilidades:

- Insertar registros nuevos de datos.

- Consultar valores históricos.
- Validar existencia de usuarios y fermentadores.

Flujo de datos

[Arduino Nano ESP32] → (MQTT o Serial) → [Capa de comunicación] → [Lógica de negocio] → [BD y UI]

Módulos principales

Tabla 3

Modelado de la distribución de la interfaz grafica

Módulo	Descripción breve
Formmenu	Pantalla principal. Permite navegación, muestra estado de conexión.
Conexion.cs	Selección entre MQTT y Serial. Cambia dinámicamente la fuente.
Form1.cs	Visualización gráfica de variables (pie charts, barras).
Form3.cs	Vista tipo tabla de los datos históricos desde SQL Server.
Mqttclient.cs	Cliente MQTT basado en la biblioteca mqttnet.

Serialreader.cs	Lectura y procesamiento de datos desde puerto serial.
Sqlhelper.cs	Conexión, inserción y consultas en la base de datos.

Nota. Modelado de los componentes implementados en la interfaz gráfica

Tecnologías utilizadas:

Tabla 4

Modelado de la tecnología implementada para el diseño de la interfaz grafica

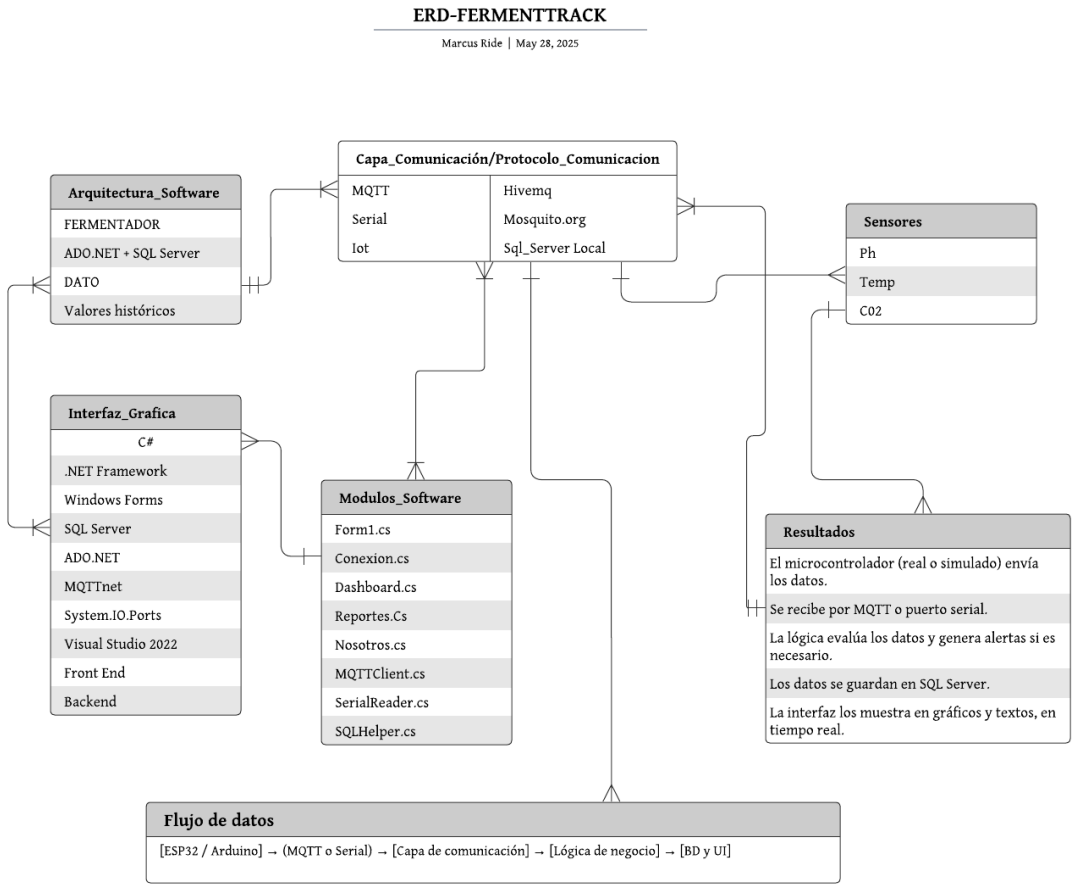
Tecnología	**Uso en el proyecto**
C#	Lenguaje principal
.NET Framework	Plataforma para la aplicación
Windows Forms	Interfaz gráfica de usuario
SQL Server	Base de datos relacional
ADO.NET	Acceso a datos
Mqttnet	Cliente MQTT en C#
System.IO.Ports	Comunicación Serial
Visual Studio 2022	IDE para desarrollo

Nota. en esta etapa se genera una tabla para verificar la implementación del diseño de la interfaz gráfica. Fuente. Autoría Propia

Diagrama (ERD):

Figura 5

Modelado ERD



Nota. Diseño de modelado ERD y BD para la gestión de datos. Fuente. Autoría Propia

El diseño de la interfaz gráfica fue realizado con base en principios de experiencia de usuario (UX) y usabilidad (UI), siguiendo lineamientos de la norma ISO 9241-210 (2010). Se empleó el lenguaje de programación C# dentro del entorno de desarrollo Visual Studio 2022. La interfaz incluyó gráficos dinámicos, alertas visuales por fuera de rango y paneles con datos en tiempo real. Se diseñaron prototipos funcionales en .NET utilizando Windows Forms, validando la navegación e interpretación con usuarios del sector caficultor mediante sesiones piloto.

El diseño contempla los siguientes elementos clave:

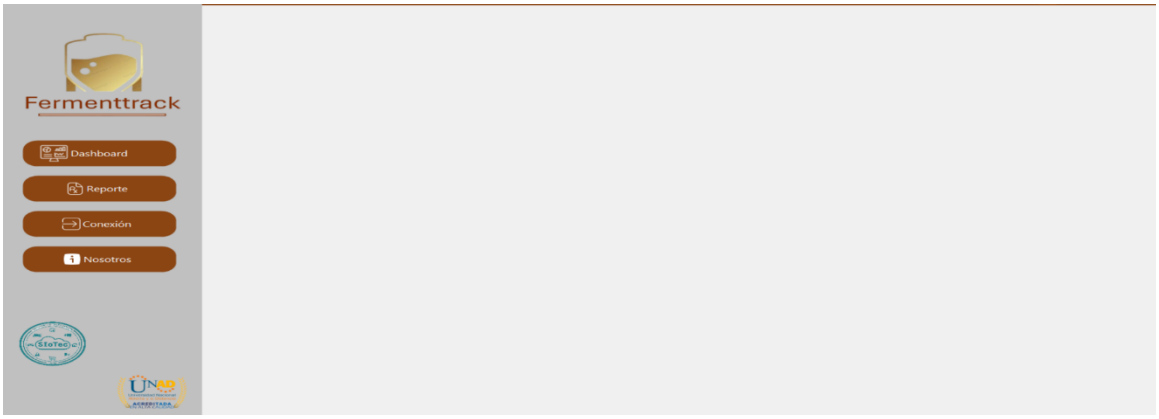
Durante esta fase se llevó a cabo la integración entre el software desarrollado y el hardware de recolección de datos. Se utilizo la placa Arduino Nano ESP32, configuradas para capturar datos a través de sensores especializados. La transmisión se realizó mediante el protocolo MQTT, utilizando el broker Mosquitto por su alta eficiencia en redes de bajo ancho de banda (Karnouskos, 2011). Los datos recibidos se almacenaron en una base de datos SQL Server, permitiendo su consulta en tiempo real y su análisis histórico. La arquitectura modular permitió alternar entre datos simulados y reales, facilitando pruebas controladas previas a su uso operativo.

Se implementan los siguientes parámetros a conocer:

Desarrollo del Dashboard: Se implementará una interfaz tipo panel de control, que muestre de forma gráfica y organizada las principales variables sensoriales capturadas (ph, temperatura, CO_2, etc.). Se utilizarán componentes visuales como gráficos de pastel, barras, medidores circulares, indicadores de nivel y tarjetas informativas.

Figura 6

Diseño del entorno visual del dashboard



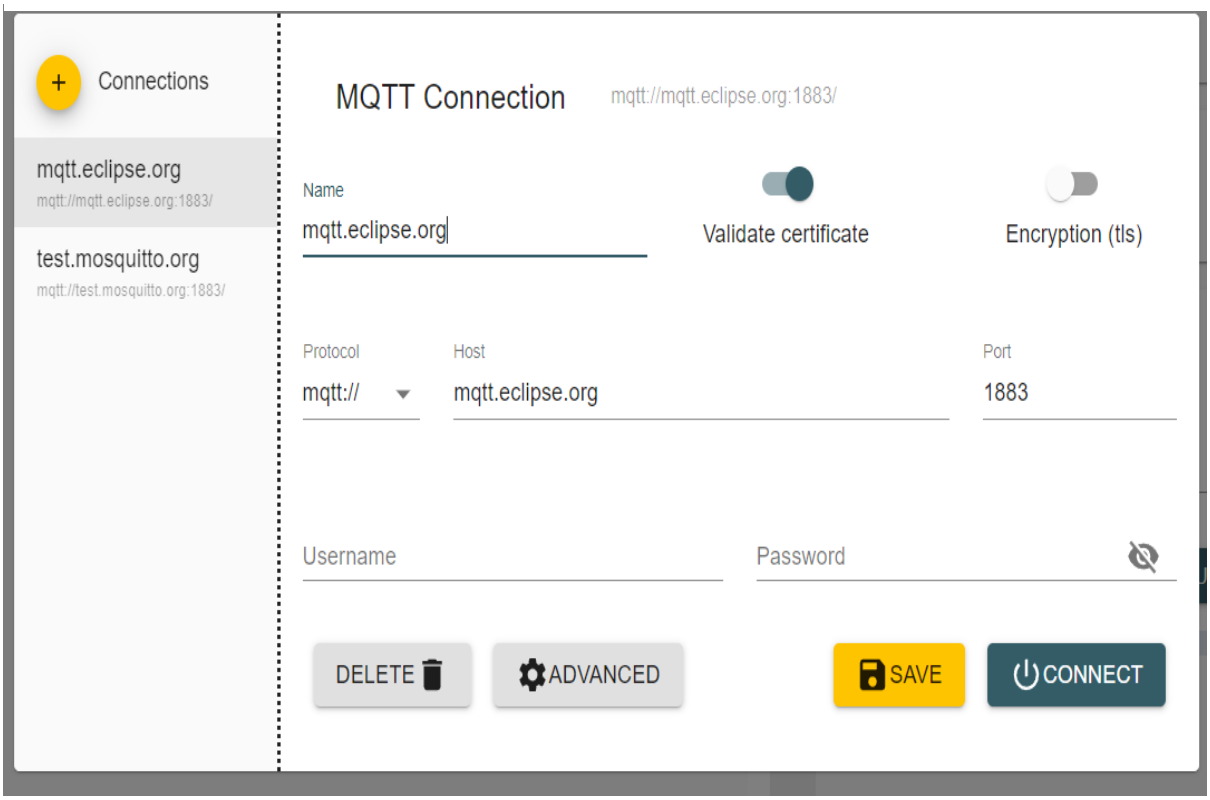
Nota. Diseño de Dashboard, Entorno principal y ejecución del programa. Fuente. Autoría Propia.

Modelado de Base de Datos: El diagrama entidad-relación (ERD) modelará cómo las entidades del sistema (usuarios, sensores, fermentadores, lecturas) interactúan entre sí. Este modelo permitirá optimizar las consultas SQL Server, mejorando el rendimiento del sistema al realizar búsquedas o análisis de los datos recolectados. Se implementarán índices en las columnas más consultadas para acelerar la respuesta de las consultas y mejorar la eficiencia del sistema.

Servidores y protocolos: Los datos se almacenarán y procesarán en servidores locales, utilizando protocolos de comunicación seguros como TCP/IP para garantizar la transmisión de los datos recopilados.

Figura 7

Protocolo de comunicación implementado MQTT



Nota. Medio de protocolo de transmisión de datos por MQTT. Fuente. Autoría Propia.

Definición de Vistas: La interfaz incluirá distintas vistas para acceder a:

- Datos en tiempo real, actualizados a través de MQTT.
- Registros históricos, extraídos desde la base de datos SQL Server, permitiendo análisis comparativos.
- Estado del sistema, incluyendo indicadores de conexión (con sensores y base de datos), alertas por fuera de rango, y logs de actividad.

Entorno de Desarrollo: Se desarrolla en el entorno de Visual Studio utilizando el lenguaje de programación C# con Windows Forms o Windows Presentation Foundation (WPF) como tecnología base. Esta elección se debe a su capacidad para desarrollar aplicaciones de escritorio robustas, con una amplia personalización de componentes gráficos, integración con bases de datos y conexión a protocolos de comunicación como MQTT.

Figura 8

Programa implementado

Nota. Entorno de desarrollo Visual Studio Basic 2022 C#. Fuente. Autoría propia

Integración y pruebas del sistema

Se implementaron pruebas funcionales, de rendimiento y usabilidad para garantizar que el sistema cumpliera con los objetivos planteados. Se evaluó la estabilidad de la comunicación MQTT, la precisión en la lectura de sensores y la capacidad de respuesta de la interfaz ante datos anómalos. Las pruebas de usabilidad se realizaron con usuarios del entorno rural mediante

cuestionarios semiestructurados y análisis observacional, siguiendo metodologías de diseño centrado en el usuario. La validez de los datos y la fidelidad de visualización fueron medidas mediante comparaciones controladas entre valores esperados y recibidos, asegurando consistencia técnica en la representación visual del proceso de fermentación. Como las siguientes mencionadas:

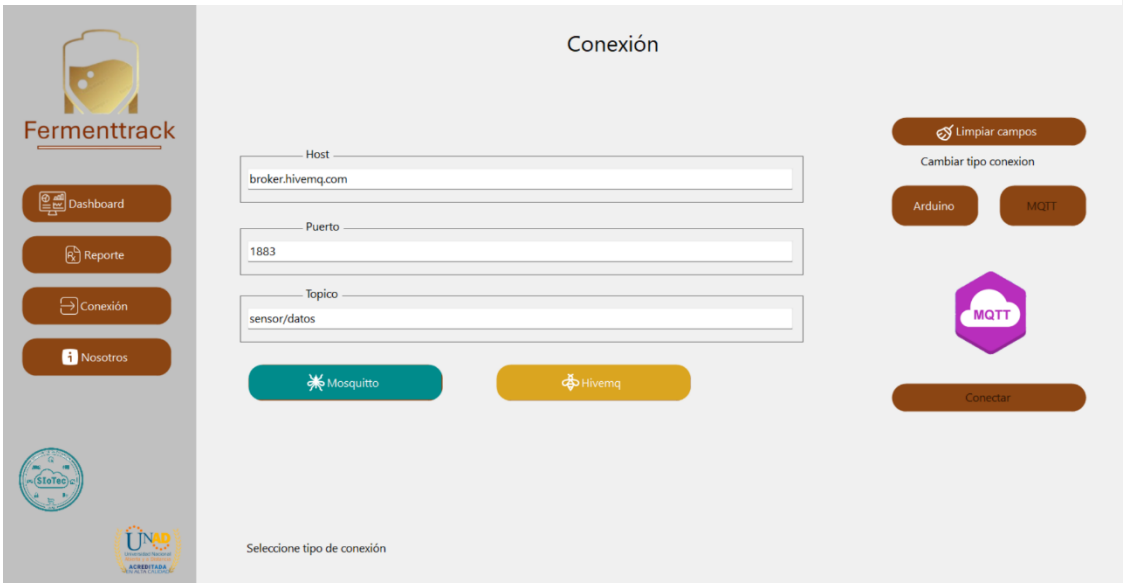
Integración del sistema de software (Interfaz gráfica)

Se realizó la integración completa de los módulos que componen la arquitectura de visualización, incluyendo:

- Conexión de la interfaz con el bróker MQTT Mosquitto, para la recepción de datos en tiempo real.
- Integración con la base de datos SQL Server para consulta y almacenamiento de datos históricos.
- Implementación de vistas gráficas dinámicas para mostrar indicadores en tiempo real y visualización histórica.

Figura 9

Conexión por protocolo MQTT



Nota. Entorno de conexión por Protocolo MQTT. Fuente. Autoría Propia

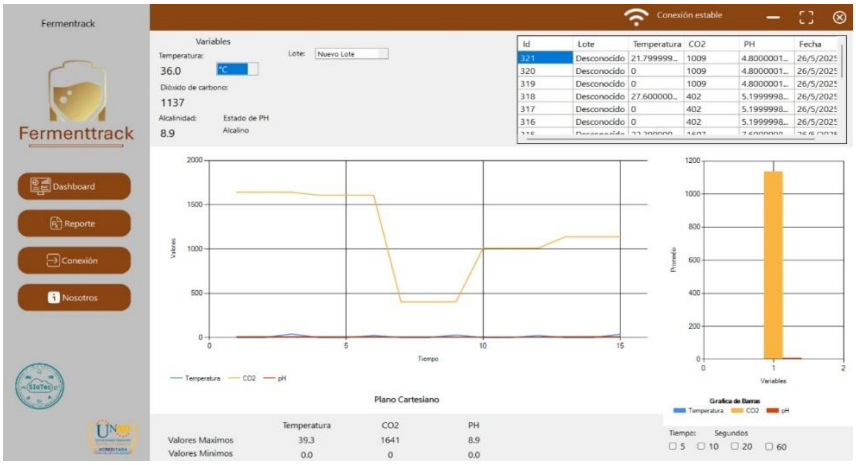
Pruebas de transmisión y recepción de datos

Para validar la comunicación de extremo a extremo en el software, se realizaron pruebas con datos simulados y reales:

- Pruebas con Arduino simulado: Validación del flujo de datos mediante la generación aleatoria de valores que imitan el comportamiento de los sensores físicos.
- Pruebas con datos reales por MQTT: Evaluación de la conectividad, latencia y visualización de valores transmitidos por los sensores reales en fermentadores de café.
- Se verificó la estabilidad de la recepción de mensajes, la frecuencia de actualización (cada 10 segundos) y la correcta inserción de datos en la base de datos tras cada ciclo de cinco lecturas.

Figura 10

Simulación de datos (Arduino UNO)



Nota. Entorno de conexión por Arduino (Simulación de datos). Fuente. Autoría propia.

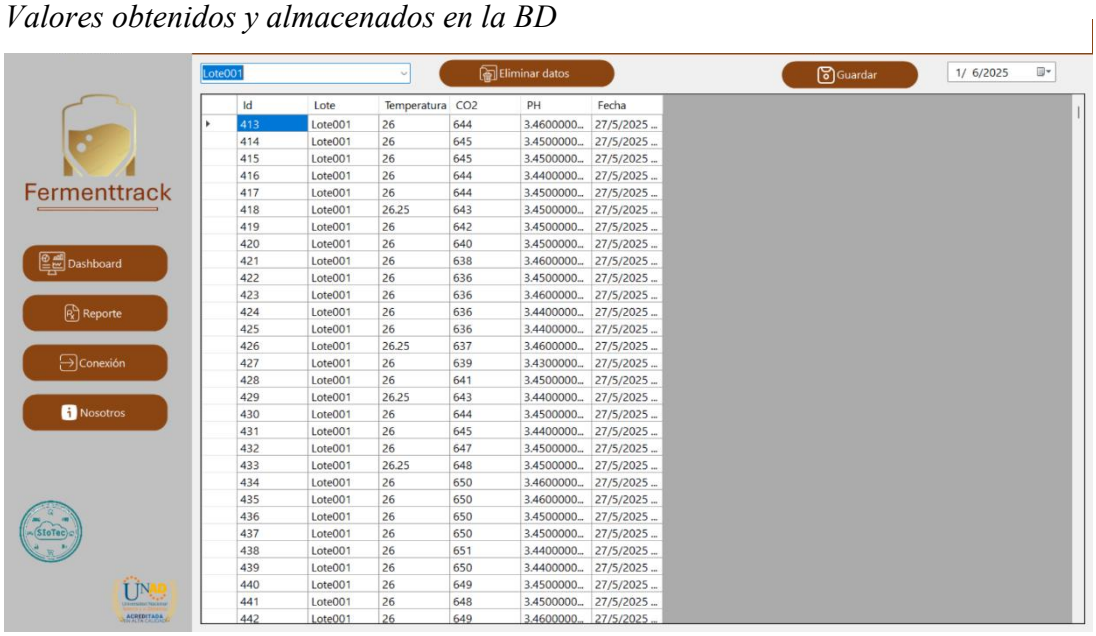
Evaluación de rendimiento y precisión visual

Las pruebas funcionales también incluyeron la validación de la actualización oportuna de los gráficos, tarjetas de datos, y alertas visuales dentro del Dashboard. Se controló:

- La visualización en tiempo real reflejara adecuadamente los cambios recientes en los valores.
- Las gráficas tipo pastel y de barras mostraran correctamente los datos actuales, mínimos, máximos y deseados.
- La latencia promedio en la actualización fuera menor a 1 segundo tras la recepción de los datos.

Figura 11

Valores obtenidos y almacenados en la BD



Id	Lote	Temperatura	CO2	PH	Fecha
413	Lote001	26	644	3.4600000...	27/5/2025 ...
414	Lote001	26	645	3.4500000...	27/5/2025 ...
415	Lote001	26	645	3.4500000...	27/5/2025 ...
416	Lote001	26	644	3.4400000...	27/5/2025 ...
417	Lote001	26	644	3.4500000...	27/5/2025 ...
418	Lote001	26.25	643	3.4500000...	27/5/2025 ...
419	Lote001	26	642	3.4500000...	27/5/2025 ...
420	Lote001	26	640	3.4500000...	27/5/2025 ...
421	Lote001	26	638	3.4600000...	27/5/2025 ...
422	Lote001	26	636	3.4500000...	27/5/2025 ...
423	Lote001	26	636	3.4600000...	27/5/2025 ...
424	Lote001	26	636	3.4400000...	27/5/2025 ...
425	Lote001	26	636	3.4400000...	27/5/2025 ...
426	Lote001	26.25	637	3.4600000...	27/5/2025 ...
427	Lote001	26	639	3.4300000...	27/5/2025 ...
428	Lote001	26	641	3.4500000...	27/5/2025 ...
429	Lote001	26.25	643	3.4400000...	27/5/2025 ...
430	Lote001	26	644	3.4500000...	27/5/2025 ...
431	Lote001	26	645	3.4400000...	27/5/2025 ...
432	Lote001	26	647	3.4500000...	27/5/2025 ...
433	Lote001	26.25	648	3.4500000...	27/5/2025 ...
434	Lote001	26	650	3.4600000...	27/5/2025 ...
435	Lote001	26	650	3.4600000...	27/5/2025 ...
436	Lote001	26	650	3.4500000...	27/5/2025 ...
437	Lote001	26	650	3.4500000...	27/5/2025 ...
438	Lote001	26	651	3.4400000...	27/5/2025 ...
439	Lote001	26	650	3.4400000...	27/5/2025 ...
440	Lote001	26	649	3.4500000...	27/5/2025 ...
441	Lote001	26	648	3.4500000...	27/5/2025 ...
442	Lote001	26	649	3.4600000...	27/5/2025 ...

Nota. Resultados cuantitativos obtenidos y registrados en la BD del Software. Fuente. Autoría Propia

Para esta fase se realizaron ciertas pruebas para comparar que los parámetros establecidos sean funcionales y presenten valores reales como los históricos registros nacionales subidos en la nube:

Tabla de pruebas funcionales del sistema

Tabla 5

Pruebas y validación del sistema

Prueba realizada	Resultado esperado	Resultado obtenido	Estado
Conexión MQTT (broker Mosquitto)	Mensajes recibidos cada 10 s	Tiempo promedio de llegada: 9.8 s	Aprobado
Visualización de variable ph	Gráfico actualiza con nuevos datos	Se actualiza correctamente sin latencia visible	Aprobado
Simulación desde Arduino	Datos aleatorios aparecen en la interfaz	Datos simulados registrados con precisión	Aprobado
Registro en SQL Server	Datos se almacenan con timestamp	Tablas muestran registros correctos	Aprobado
Alerta por valor crítico	Muestra alerta visual si ph < 4.0 o > 8.0	Alertas mostradas correctamente	Aprobado

Nota. Pruebas funcionales del sistema y verificación de funcionamiento. Fuente. Autoría Propia

Evaluación de usabilidad

Evaluación de usabilidad – Escala SUS:

Para validar la usabilidad de la interfaz gráfica desarrollada en fermenttrack, se aplicó la System Usability Scale (SUS), una herramienta ampliamente utilizada para evaluar la percepción subjetiva de facilidad de uso de sistemas interactivos (Brooke, 2020). Esta escala consta de 10 ítems con respuestas en formato Likert de 5 puntos, que permiten obtener un puntaje total entre 0 y 100. Según Sauro (2020), un puntaje superior a 85 es considerado "excelente" y refleja una alta satisfacción del usuario.

- Claridad en la presentación de la información.
- Facilidad de navegación entre vistas (real, histórico, estado del sistema).
- Comprensión de los íconos e indicadores visuales.
- Esta evaluación se realizó mediante observación directa y cuestionarios breves, recogiendo sugerencias para mejoras futuras en el diseño UI/UX.

Se aplicó la encuesta a un grupo reducido de usuarios (caficultores y técnicos), en sesiones de prueba en las que se simulaban diferentes escenarios de uso de la interfaz (monitoreo de variables, visualización de alertas, consulta histórica). Los resultados se consolidaron y promediaron por criterio evaluado, como se muestra a continuación:

Tabla de validación y evaluación de usabilidad método (SUS)

Tabla 6

Verificación y evaluación

Criterio evaluado	Puntaje promedio sobre 5	Observaciones
Facilidad de uso	4.2	Navegación intuitiva y lógica de interfaz clara
Claridad de los gráficos e indicadores	4.5	Buena interpretación de valores en tiempo real
Tiempo de respuesta de la interfaz	4.3	Fluida y estable con datos en transmisión
Interacción general con la plataforma	4.4	La experiencia fue positiva en su totalidad
Intención de uso continuo	4.6	Alta disposición de los usuarios a seguir usándola

Nota. Evaluación de usabilidad – Escala SUS. Fuente. Autoría Propia

Escala (SUS) total promedio: 88 / 100 → Excelente usabilidad (Brooke, 2020)

El puntaje total obtenido en la Escala SUS fue de 88 sobre 100, lo que ubica a la aplicación dentro del rango de excelente usabilidad, de acuerdo con los estándares internacionales (Bangor et al., 2008). Esto confirma que la interfaz es comprensible, funcional y apropiada para su público objetivo, incluso en entornos con baja familiaridad tecnológica

Documentación y validación final

Como resultado final, se elaboró una guía de interfaz gráfica (GIG) que documenta la arquitectura del sistema, el uso de la plataforma y los procedimientos para la interpretación de datos. Esta guía incluye: estructura visual de la interfaz, flujo de navegación, integración con sensores, gestión de datos en SQL Server, estrategias de validación, mantenimiento y potenciales actualizaciones futuras. También se contemplaron recomendaciones para su implementación en zonas rurales con conectividad limitada, y estrategias básicas para su escalamiento y comercialización en entornos agroindustriales.

Registro del proceso de desarrollo:

Durante el desarrollo del sistema, se documentaron detalladamente cada una de las etapas que conformaron el ciclo de vida del proyecto, desde la fase de diseño de la interfaz gráfica hasta su implementación e integración con los servicios de mensajería y bases de datos. Se llevaron registros técnicos del funcionamiento del dashboard, la comunicación vía protocolo MQTT, la interacción con la base de datos SQL Server y la experiencia del usuario durante las pruebas.

La documentación incluye diagramas de flujo de datos, descripciones de arquitectura, capturas de pantalla del entorno visual desarrollado en C# con Visual Studio, así como los resultados de las pruebas de rendimiento y usabilidad.

Ventajas, limitaciones y posibilidades de escalado (hablar sobre el problema de la energía)

El sistema desarrollado presenta diversas ventajas:

- Visualización clara y en tiempo real de variables críticas del proceso de fermentación (como ph, temperatura y CO_2).

- Capacidad para almacenar y consultar datos históricos.
- Integración eficiente con MQTT para sistemas iot de bajo ancho de banda.

Sin embargo, también se identificaron limitaciones, como:

- Dependencia de conectividad estable para la recepción de datos en tiempo real.
- Limitada flexibilidad gráfica en algunos componentes de Windows Forms frente a tecnologías más modernas como WPF o interfaces web.
- La simulación de datos con Arduino es útil en fase de pruebas, pero requiere sustitución por sensores físicos en la etapa productiva.

En cuanto a las posibilidades de escalado, el sistema está diseñado de forma modular, lo que permite:

- Ampliar el número de variables monitorizadas.
- Migrar la visualización a plataformas web o móviles para mayor accesibilidad.

Recomendaciones para su uso en entornos rurales:

Para su implementación en contextos rurales o de baja conectividad, se recomienda:

- Configurar el sistema para operar de forma local, con sincronización de datos cuando se recupere la conexión.
- Optimizar la transmisión MQTT con técnicas de compresión de mensajes o ajustes en la qos.
- Capacitar a los usuarios finales (productores y técnicos) mediante interfaces intuitivas, tutoriales y manuales de uso.
- Considerar el uso de dispositivos de bajo consumo energético y bajo costo, que faciliten su adopción en comunidades rurales.

Recursos necesarios e implementados

Tabla 7

Tabla de recursos necesarios para la implementación

RECURSO	DESCRIPCIÓN	PRESUPUESTO
Equipo Humano	Desarrolladores de software en C# y bases de datos, diseñadores de UX/UI,	**700.000**
Equipos y Software	Módulos de comunicación, SQL Server, MQTT Brokers, servidores locales.	**700.000**
Viajes y Salidas de Campo	Visita a sitios de cultivos de café para la implementación y pruebas.	**250.000**
Materiales y suministros	Materiales de cómputo, salvaguardas.	**550.000**
Bibliografía	Libros, artículos científicos, bases de datos para consulta	**0**
TOTAL: 2,200.000		

Nota. Presupuesto necesario para la implementación de la solución. Fuente. Autoría Propia.

Recursos y productos esperados

Tabla 8

Tabla de recursos o Productos Esperados

RESULTADO/PRODUCTO ESPERADO	INDICADOR	BENEFICIARIO
Interfaz gráfica funcional	Prototipo de la interfaz con visualización en tiempo real.	Operadores del fermentador de café.
Sistema de monitoreo validado	Pruebas de precisión y fiabilidad completadas.	Industria cafetera, con énfasis en el proceso de fermentación.
Documentación completa	Manual de usuario y guía grafica (gig) elaborado	Desarrolladores futuros y equipos técnicos encargados de mantenimiento.

Nota. Recursos necesarios para la implementación de la solución. Fuente. Autoría Propia.

Elaboración de manuales técnicos de uso:

Se desarrollaron guías digitales y físicas dirigidas a los usuarios finales (productores y técnicos), detallando el funcionamiento de la interfaz, la instalación del sistema y la interpretación de las gráficas. Las guías también incluyen protocolos de acción ante alertas emitidas por el sistema.

Producción de recursos audiovisuales de capacitación:

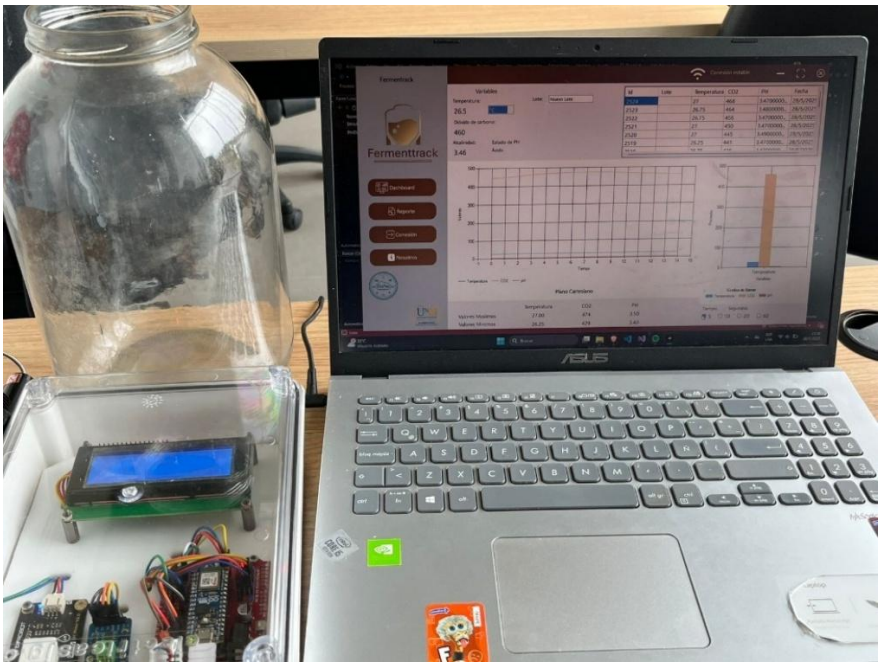
Se crearon archivos de audio y video con explicaciones del funcionamiento del sistema, enfocados en diferentes niveles de conocimiento técnico. Estos recursos permiten facilitar la capacitación remota o asincrónica de los usuarios, favoreciendo la inclusión digital y tecnológica.

Resultados

Se desarrolló una interfaz gráfica en C# que permitió el monitoreo en tiempo real de las variables críticas del fermentador de café (temperatura, pH y CO_2). La herramienta mostró datos mediante gráficos dinámicos y paneles interactivos, lo que facilitó la interpretación de los valores por parte de los usuarios.

Figura 12

Implementación de medio transmisor de datos y uso del programa

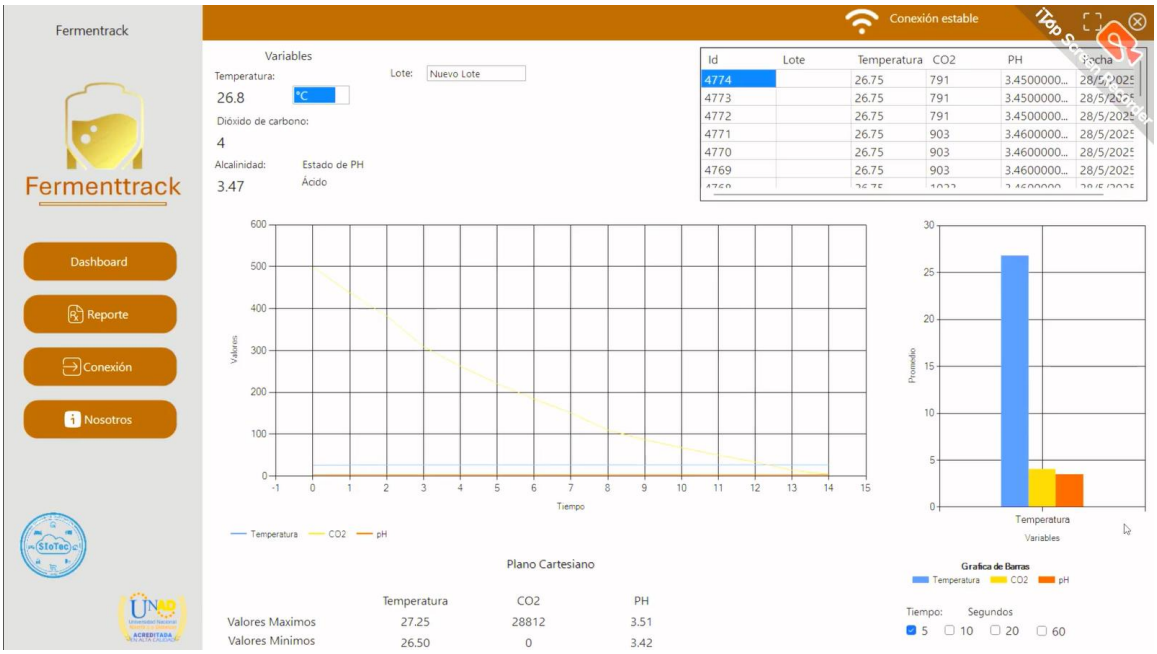


Nota. Implementación del Fermentador y Aplicación en revisión de su funcionabilidad. Fuente. Autoría propia.

Se gestionó una base de datos en SQL Server que garantizó la conexión entre la interfaz gráfica y la unidad de almacenamiento de información. Esta base de datos organizó de manera eficiente los registros históricos y aseguró la persistencia de los datos recolectados por los sensores.

Figura 13

Dashboard Fermentador de Café



Nota. Visualización funcional del form sección Dashboard. Fuente. Autoría propia.

Se implementó un sistema de notificaciones dentro de la interfaz gráfica, el cual generó alertas automáticas cuando los valores de las variables monitoreadas se encontraron fuera de los rangos establecidos. Este mecanismo contribuyó a la identificación oportuna de anomalías en el proceso de fermentación.

Figura 14

Equipo electrónico configurado con sensores: temperatura, CO2 y pH

Nota. Validación de datos en fermentador. Fuente. Autoría propia.

Se documentó la normatividad vigente relacionada con la implementación de soluciones tecnológicas en el sector agroindustrial y se elaboró un manual de usuario. Este documento ofreció instrucciones claras sobre el funcionamiento del sistema, su instalación y las acciones recomendadas ante las alertas generadas.

Conclusiones

La implementación de este software representa un avance importante en la digitalización del monitoreo agroindustrial al ofrecer una interfaz gráfica amigable y adaptable tanto para pequeños productores de café como para su uso en entornos empresariales. Esta interfaz, fue diseñada con un enfoque UX/UI, permitiendo la visualización remota en tiempo real de variables clave frente al proceso de fermentación —como lo son la temperatura, el pH y el CO_2— mediante gráficos dinámicos, garantizando una conexión confiable a través de protocolos IoT como MQTT y comunicación por puerto serie. Se implemento una arquitectura robusta para el sistema y una integración con bases de datos SQL Server para su almacenamiento, de igual forma se implementaron una serie de manuales técnicos y recursos audiovisuales cuyo objetivo es facilitar la interacción con nuestro software sin presentar perdida alguna de su entorno virtual, incluso en contextos con baja conectividad. Como resultado de funcionamiento, nuestro software da a conocer el cumplimiento de los objetivos específicos del proyecto planteado, consolidando una solución escalable, segura y centrada en el usuario, que mejora el proceso de monitoreo en la fermentación, y promueve una mayor apropiación tecnológica dentro del sector caficultor.

Referencias bibliográficas

Agrosmart. (2023). Soluciones en agricultura digital para Latinoamérica. Https://www.agrosmart.com

Arduino. (n.d.). Arduino reference. Https://www.arduino.cc/reference/en/

Bangor, A., Kortum, P. T., & Miller, J. T. (2020). An empirical evaluation of the System Usability Scale. International Journal of Human–Computer Interaction, 24(6), 574–594. Https://doi.org/10.1080/10447310802205776

Baquero, S., Barbosa, C., & Fernández, J. (2023). Gamificación en ambientes digitales de aprendizaje: estrategias de motivación e interacción. Revista de Tecnología Educativa, 15(2), 35–48. Https://doi.org/10.1234/rte.v15i2.2345

Brooke, J. (2020). SUS: A “quick and dirty” usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester & A. L. Mcclelland (Eds.), Usability Evaluation in Industry (pp. 189–194). London: Taylor & Francis.

Colombia. Federación Nacional de Cafeteros. (2023). Informe anual de exportaciones y calidad del café colombiano. Https://www.federaciondecafeteros.org

Cropin. (2022). Smartfarm: Digital Agriculture Solutions. Https://www.cropin.com

DANE. (2024). Índice de pobreza monetaria por departamentos en Colombia 2024. Departamento Administrativo Nacional de Estadística. Https://www.dane.gov.co

Date, C. J. (2020). An introduction to database systems (8th ed.). Addison-Wesley.

Espressif Systems. (n.d.). ESP32 technical reference manual. Https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/index.html

Gómez, A., & Rivera, J. (2020). Interfaces gráficas para el agro: diseño centrado en el usuario rural. Revista de Tecnología Agrícola, 10(1), 55–63.

Gutiérrez Jiménez, M., Aldana Bermúdez, C., & Montiel Buriticá, S. (2023). Diseño de contenidos educativos contextualizados para comunidades rurales. Revista Colombiana de Educación, 92(1), 71–93. Https://doi.org/10.17227/rce.num92-17054

Hernán, M., Hugo, R., & Jakeline, G. (2020). La caficultura huilense: desafíos y oportunidades en el mercado internacional. Editorial Universidad Surcolombiana.

HIVEMQ. (2024). Fundamentos de MQTT. Https://www.hivemq.com/mqtt

ICA. (2021). Agroapp ICA: Guía digital para productores agrícolas. Instituto Colombiano Agropecuario. Https://www.ica.gov.co

ISO 9241-210. (2020). Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems. International Organization for Standardization.

Karnouskos, S. (2021). MQTT and coap: Understanding two iot protocols. Industrial Informatics Magazine, 7(4), 27–35.

Karnouskos, S. (2021). Stuxnet worm impact on industrial cyber-physical system security. In IECON 2021 – 37th Annual Conference on IEEE Industrial Electronics Society.

Lanuza, G., Agurcia, A., Cornejo, M., & Matey, R. (2023). Análisis de variables clave en la fermentación del café: temperatura, ph y CO_2. Revista Científica de Agroindustria, 18(3), 44–59. Https://doi.org/10.5678/rca.v18i3.876

Microsoft. (2023, enero 6). Desktop guide (Windows Forms .NET). Https://learn.microsoft.com/enus/dotnet/desktop/winforms/overview/?View=netdesktop-6.0

Microsoft. (2024, julio 22). Novedades de SQL Server 2022. Https://learn.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server2022?View=sql-server-ver15

Microsoft. (n.d.). C# programming guide. Https://docs.microsoft.com/dotnet/csharp/programming-guide/

Mongodb, Inc. (n.d.). Mongodb: The definitive guide (3rd ed.). O'Reilly Media.

MQTT.org. (2022). MQTT.org. Https://mqtt.org/getting-started/

MQTT.org. (n.d.). MQTT 3.1.1 specification. Https://mqtt.org/specification

Orús, R. (2025). Producción mundial de café por país 2024. Statista. Https://www.statista.com/statistics/263311/worldwide-production-of-coffee-bycountry/

Redis. (n.d.). Redis documentation. Https://redis.io/documentation

Sauro, J. (2021). A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices. Measuring Usability LLC.

Sqlite Consortium. (n.d.). Sqlite: The complete 2023 documentation. Https://www.sqlite.org/docs.html

Torres, M., Gómez, D., & Castillo, L. (2022). Iot en la agricultura colombiana: retos y oportunidades. Revista de Innovación Agropecuaria, 13(1), 20–35.

W3schools. (2024). Tutorial de SQL. Https://www.w3schools.com/sql/

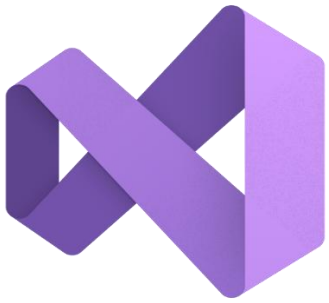
Anexos

Anexo 1 Entorno de desarrollo Integrado

El proyecto de la interfaz gráfica para la visualización remota en fermentadores de café mediante IoT (Luigi Atzori, 2010) esta desarrollado en lenguaje C# (Andrés Troelsen, 2021) en el entorno de desarrollo Visual Studio Community 2022. (IDE) (Microsoft, 2022)

Figura 15.

Visual Studio Community 2022.



Tomado de: *https://www.google.com/imgres?q=visual%20studio%20community%202022&imgurl=https%3A%2F%2Fimages-eds-ssl.xboxlive.com%2Fimage%3Furl%3D4rt9.lXDC4H_93laV1_eHHFT949fUipzkiFOBH3fAiZZUCdYojwUyX2aTonS1aIwMrx6NUIsHfUHSLz*

La interfaz se divide en cinco formularios:

1. Form1.cs
2. Conexión.cs
3. Dashboard.cs
4. Reporte.cs
5. Nosotros.cs

Anexo 2 Form1

El Form1.cs es el formulario principal donde se encuentra el menú para desplazarse a cada una de las secciones haciéndonos posible comunicarnos con cada uno de los formularios, haciendo de menú principal, ejecutándose y acoplándose cada uno de los otros formularios dentro del PnlVista de forma que no sean los formularios de nivel superior.

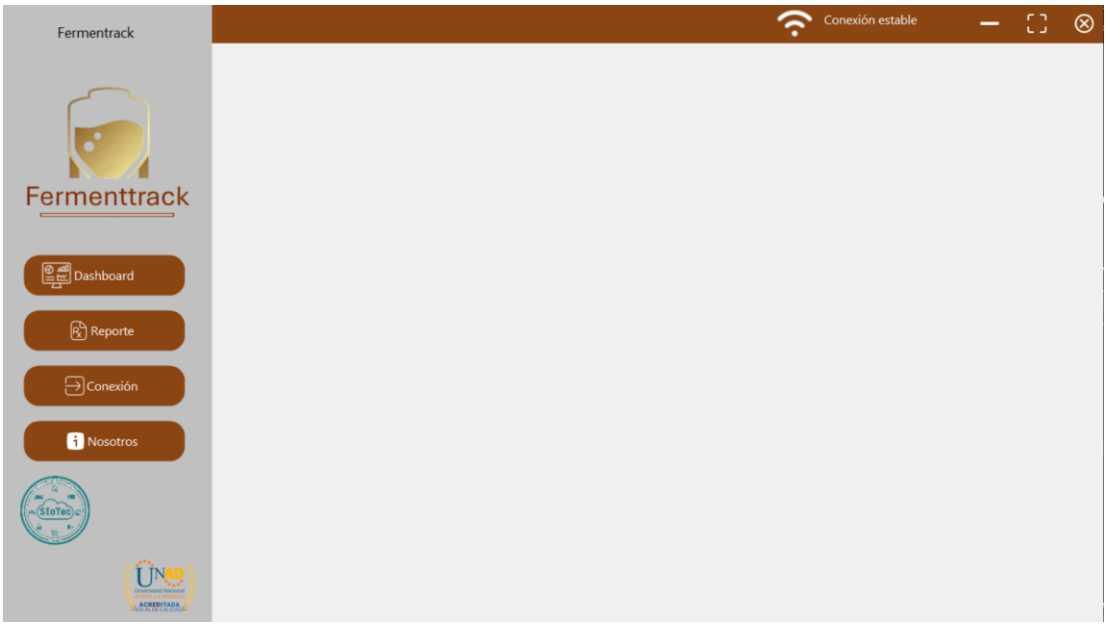


Figura 16. Interfaz gráfica para la visualización remota en fermentadores de café mediante IoT, FormMenu. Autoria Propia

Interfaz principal (Inicial)

En esta interfaz podemos observar diversas herramientas que nos permiten el desarrollo y buen funcionamiento de la aplicación de interfaz gráfica para la visualización remota en fermentadores de café mediante IoT, en este formulario tenemos los siguientes elementos:

1. Etiquetas:

a. LblTitulo = “Fermentrack”

b. LblEstado = “Estado de conexión a Internet”

2. Botones:

a. Botones Principales:

i. BtnDashboard

ii. BtnReporte

iii. BtnConexion

iv. BtnNosotros

b. Botones Secundarios:

i. BtnMinimizar

ii. BtnMaximizar

iii. BtnCerrar

3. Paneles:

a. Panel1

b. Panel2

c. Panel3

d. Panel4

e. Panel6

f. Panel9

g. Panel10

h. Panel14

i. Panel18

j. Panel20

k. Panel21

l. Panel22

m. Panel23

n. Panel26

o. Panel27

p. Panel28

q. Panel30

r. Panel31

s. Panel32

t. Panel33

u. Panel34

v. Panel35

w. PnlImagen

x. PnlVista

4. Caja de Imagen:

a. pictureBox1

b. pictureBox2

c. pictureBox3

d. PcbLogoWifi

Tabla 9.

Explicación código Form1.cs

Librerías Form1
using Interfaz_grafica.Controlador; *using System;* *using System.Net.NetworkInformation;* *using System.Threading.Tasks;* *using System.Windows.Forms;*
Estas son las librerías usadas en el código de la interfaz Form1 donde se establece el menú para el desplazamiento entre formularios. *using Interfaz_grafica.Controlador;* importa las clases y funciones del namespace Controlador, el cual está dentro del proyecto Interfaz_grafica. *using System;* Librería basica de .NET Framework *using System.Net.NetworkInformation;* Se usa para trabajar sobre información de la red tal como es el caso de Ping y NetworkInterface *using System.Threading.Tasks;* Permite el uso y ejecución de tareas asincrónicas tales como - Async. - Await. *using System.Windows.Forms;*

Libreria que contiene las clases necesarias para construir una interfaz gráfica con formularios.

- Form
- Button
- Label
- Etc.

Definició de Clase FormMenu

```
namespace Interfaz_grafica
{
    public partial class FormMenu : Form
    {
    }
}
```

namespace Interfaz_grafica

Esta dentro del namespace Interfaz_grafica

{public partial class FormMenu ...}

Declaración de clase FormMenu la cual heredada Form, representando así una ventana de aplicación.

Constructor del formulario

```
public FormMenu()
{
    InitializeComponent();
    VerificarConexion();
}
```

InitializeComponent();

Inicializa los componentes del formulario (Botones, paneles, imágenes, etc..)

VerificarConexion();

Llama a la función asincrónica que comprueba el estado de conexión a internet, esta también permite la actualización de la imagen PcbLogoWifi al igual que el Label LblEstado según el estado de conexión.

Método de Verificación de Conexión

```
//Wifi Imagenes
private async void VerificarConexion()
{
    while (true)
    {
        if (!NetworkInterface.GetIsNetworkAvailable())
        {
            PcbLogoWifi.Image = Properties.Resources.Sin_wifi;
            LblEstado.Text = "Sin conexión a Internet";
        }
        else if (await EsConexionInestable())
        {
            PcbLogoWifi.Image = Properties.Resources.inestable_wifi;
            LblEstado.Text = "Conexión inestable wifi";
        }
        else
        {
            PcbLogoWifi.Image = Properties.Resources.wifi;
            LblEstado.Text = "Conexión estable";
        }

        await Task.Delay(5000);
    }
```

}

private async void VerificarConexion()

{ ... }

Es un método asincrónico que corre en segundo plano sin interferir directamente en la interfaz.

Este se ejecuta infinitamente en un bucle while(true), verificando la conexión cada cinco segundos.

while (true)

{if (!NetworkInterface.GetIsNetworkAvailable())

{PcbLogoWifi.Image = Properties.Resources.Sin_wifi;

LblEstado.Text = "Sin conexión a Internet";}

Si no hay conexión a internet este inmediatamente proyecta una imagen “Sin_wifi” y se actualiza el label LblEstado a “Sin conexión a Internet”.

else if (await EsConexionInestable())

{PcbLogoWifi.Image = Properties.Resources.inestable_wifi;

LblEstado.Text = "Conexión inestable wifi";}

Si se presenta una conexión, pero inestable, esta hará que cambie la imagen a “inestable_wifi” al igual que el label LblEstado a “Conexión inestable wifi”.

else

{PcbLogoWifi.Image = Properties.Resources.wifi;

LblEstado.Text = "Conexión estable";}

Si todo está bien se muestra una imagen de “wifi”, y el label LblEstado dirá “Conexión estable”.

await Task.Delay(5000);

Se presenta un retraso de cinco segundos antes de volver a iniciar la verificación.

Método EsConexionInestable

```
//Verifica si la conexión a Internet es inestable
private async Task<bool> EsConexionInestable()
{
    try
    {
        var ping = new Ping();
        var reply = await ping.SendPingAsync("8.8.8.8", 1000); // Google DNS
        return reply.Status != IPStatus.Success || reply.RoundtripTime > 150;
    }
    catch
    {
        return true; // Si ocurre un error al hacer ping, se considera inestable
    }
}
```

//Verifica si la conexión a Internet es inestable

Comentario guía

private async Task<bool> EsConexionInestable()

{ ... }

Método asincrónico que me retorna un bool, este verifica si la conexión es inestable.

Try

{ ... }

Condicional try-catch

var ping = new Ping();

var reply = await ping.SendPingAsync("8.8.8.8", 1000); // Google DNS

Envía un Ping al servidor DNS de Google (8.8.8.8) con un timeout de un segundo.

return reply.Status != IPStatus.Success || reply.RoundtripTime > 150;

Si el Ping falla o si tarda más de 150 milisegundos, la conexión se considera inestable.

catch

{ return true; // Si ocurre un error al hacer ping, se considera inestable }

Ocurre lo que especifica el comentario, un ejemplo de ello es sin red o sin los permisos de esta, esto se considera inestable.

Botones de ventana (cerrar, maximizar, minimizar)

```
//Botones
 private void BtnCerrar_Click(object sender, EventArgs e)
 {
    this.Close();
 }
 private void BtnMaximizar_Click(object sender, EventArgs e)
 {
    // Verificar el estado actual de la ventana
    if (this.WindowState == FormWindowState.Maximized)
    {
        // Si está maximizada, restaurarla a su tamaño original
        this.WindowState = FormWindowState.Normal;
    }
    else
    {
        // Si está en tamaño original, maximizarla
        this.WindowState = FormWindowState.Maximized;
    }

 }
 private void BtnMinimizar_Click(object sender, EventArgs e)
 {
    this.WindowState = FormWindowState.Minimized;
```

}

private void BtnCerrar_Click(object sender, EventArgs e)

{this.Close();}

Este botón Cierra el formulario actual.

private void BtnMaximizar_Click(object sender, EventArgs e)

{ // Verificar el estado actual de la ventana

if (this.WindowState == FormWindowState.Maximized)

{// Si está maximizada, restaurarla a su tamaño original

this.WindowState = FormWindowState.Normal;}

else

{// Si está en tamaño original, maximizarla

this.WindowState = FormWindowState.Maximized;}}

Este botón altera la ventana, la llega a maximizar y volver a el tamaño normal.

private void BtnMinimizar_Click(object sender, EventArgs e)

{ this.WindowState = FormWindowState.Minimized; }

Minimiza la Ventana.

Botones de navegación entre formularios (Dashboard, Nosotros, etc.)

```
//BtnFormularios para Panel
private void BtnReporte_Click(object sender, EventArgs e)
{
    AbrirFormulario(new Reporte());
}

private void BtnDashboard_Click(object sender, EventArgs e)
{
    Conexion conexion = new Conexion();
    AbrirFormulario(new Dashboard(conexion));
}
private void BtnNosotros_Click(object sender, EventArgs e)
{
    AbrirFormulario(new Nosotros());
}
private void BtnConexion_Click(object sender, EventArgs e)
{
    AbrirFormulario(new Conexion());
}
```

private void BtnReporte_Click(object sender, EventArgs e)

{AbrirFormulario (new Reporte());}

Abre un formulario llamado Reporte dentro del panel PnlVista

private void BtnDashboard_Click(object sender, EventArgs e)

{ Conexion conexion = new Conexion(); *AbrirFormulario (new Dashboard(conexion));}* Crea un objeto de tipo Conexión y se lo pasa al formulario Dashboard *private void BtnNosotros_Click(object sender, EventArgs e)* *{AbrirFormulario (new Nosotros());}* Abre el formulario Nosotros *private void BtnConexion_Click (object sender, EventArgs e)* *{AbrirFormulario (new Conexion());}* Abre el formulario Conexión.
Método AbrirFormulario()

```
    //Codigo Conf Presentacion Forumularios
    private void AbrirFormulario(Form formulario)
    {
        // Limpiar el panel antes de abrir un nuevo formulario
        PnlVista.Controls.Clear();

        // Configurar el formulario hijo
        formulario.TopLevel = false;
        formulario.FormBorderStyle = FormBorderStyle.None;
        formulario.Dock = DockStyle.Fill;

        // Agregarlo al panel y mostrarlo
        PnlVista.Controls.Add(formulario);
        formulario.Show();
    }
  }
}
```

private void AbrirFormulario(Form formulario)

{ ... }

Este método se encarga de mostrar cualquier formulario dentro de un panel (PnlVista) en lugar de abrir una nueva ventana.

PnlVista.Controls.Clear();

Limpiar el panel antes de abrir un nuevo formulario

// Configurar el formulario hijo

formulario.TopLevel = false;

formulario.FormBorderStyle = FormBorderStyle.None;

formulario.Dock = DockStyle.Fill;

Configura el formulario para que no sea de nivel superior (Identifica que no sea una ventana independiente), no tenga bordes y se ajuste completamente al panel (PnlVista)

PnlVista.Controls.Add(formulario);

formulario.Show();

Agregarlo el formulario al panel y mostrarlo

Anexo 3 Conexión

En el formulario de conexión como bien lo dice se conecta a uno de los dos métodos de comunicación establecidos en la aplicación, estos son:

- MQTT (OASIS, 2019)

La conexión mediante MQTT se realiza a travez de brokers lo cual nos facilita la comunicación entre dispositivos mediante el Internet de las cosas (IoT), a su vez también usamos dos tipos de bróker por seguridad para la recepción de datos uno es HiveMQ y el otro es Mosquitto, en las imágenes podemos observar la conexión que se establece mediante un puerto, host y el tópico.

- HiveMQ (HIVEMQ, 2023)

Figura 17. Formulario HiveMQ.

Fuente: Autoria Propia

- Mosquitto: (Mosquitto, 2023)

Figura 18.Formulario Mosquitto.

Fuente: Autoria Propia

- USB

Figura 19. Formulario por Arduino.

Fuente: Autoría Propia

Este formulario se adapta a el espacio del panel (PnlVista), este permite la conexión por alguna de las formas establecidas, sin embargo, se debe tener en cuenta que la conexión al bróker se realiza con un internet de mediana o buena calidad pues de lo contrario no le permitirá una buena conexión a este y la aplicación puede llegar a presentar una breve desconexión presentándose lagunas de datos.

En este formulario se cuenta con los siguientes elementos:

1. Etiquetas:

a. LblTitulo

b. label2

c. label3

d. LblHost

e. label5

f. LblPuerto

g. LblTopico

h. LblCOM

i. LblBroker

j. LblTipoConexion

2. Botones:

a. BtnLimpiarCampos

b. BtnArduino

c. BtnMQTT

d. BtnConectar

e. BtnMosquitto

f. BtnHivemq

3. Caja de Imagen:

a. PcbConexion

4. Paneles:

a. panel1

b. panel2

c. panel3

d. panel4

5. Cajas de Texto:

a. TxtHost

b. TxtPuerto

c. TxtTopico

Tabla 10.

Explicación código Conexion.cs.

Fuente: Autoría Propia

Imports / Usings
using System; *using System.IO.Ports;* *using System.Windows.Forms;* *using uPLibrary.Networking.M2Mqtt;* *using uPLibrary.Networking.M2Mqtt.Messages;*
using System;

Funciones básicas del Sistema como lo son los String, Esception, Guid, ect.

using System.IO.Ports;

Libreria de clases del sistema que permite el manejo de puertos seriales (COM)

using System.Windows.Forms;

Permite la creación y manejo de formularios y sus controles en una aplicación de escritorio.

using uPLibrary.Networking.M2Mqtt;

Es una libreria usada para trabajar con MQTT en .NET (Cliente, conexión, publicación, ect.).

using uPLibrary.Networking.M2Mqtt.Messages;

Contiene los tipos de mensajes y eventos de MQTT.

Definición de la clase Conexion

```
namespace Interfaz_grafica.Controlador
{
    public partial class Conexion : Form
    {
    }
}
```

namespace Interfaz_grafica.Controlador

Define la clase Conexion dentro del namepace Interfaz_grafica.Controlador.

{ public partial class Conexion : Form ...}

Hereda de Form la ventana de aplicación

Delegado y eventos personalizados, así como variables de conexión y datos.

```
    public delegate void DatosRecibidosEventHandler(string temperatura, string co2,
string ph);
    public event DatosRecibidosEventHandler DatosRecibidos;
    private string _host;
    private string _puerto;
    private string _topico;
    private string _idCliente;
    private string _tipoConexion;
    private MqttClient _client;
    private string _temperatura;
    private string _co2;
    private string _ph;
    private SerialPort _serialPort;
```

public delegate void DatosRecibidosEventHandler(string temperatura, string co2, string ph);

Define un delegado (Tipo de método) que recibe Sting

public event DatosRecibidosEventHandler DatosRecibidos;

Define un evento que se dispara cuando llegan los datos (Temperatura, co2, ph)

Otros formularios se pueden suscribir a través de este evento, reaccionando a los datos recibidos como es el caso del formulario Dashboard que será presentado más adelante.

private string _host, _puerto, _topico, _idCliente, _tipoConexion;

_host, _puerto, _topico: Son los datos del servidor MQTT

_idCliente: Es el identificador único del cliente MQTT

_tipoConexion: Puede ser “MQTT” o “Arduino”

private MqttClient _client;

Objeto de tipo *MqttClient* es el cual maneja la conexión MQTT

private string _temperatura, _co2, _ph;

Son las variables para almacenar temporalmente los datos recibidos de los sensores.

Constructor

```
public Conexion()
{
    InitializeComponent();
}
```

Es el cual inicializa los componentes del formulario, esta línea configura la interfaz diseñada con el diseño de Visual Studio.

Botón que limpia los campos

```
private void BtnLimpiarCampos_Click(object sender, EventArgs e)
{
    TxtHost.Clear();
    TxtPuerto.Clear();
    TxtTopico.Clear();
}
```

Limpia los campos de texto en los cuales se enviará los datos al bróker para la conexión MQTT

Botón para seleccionar Conexión Arduino

```
private async void BtnArduino_Click(object sender, EventArgs e)
{
    InvisibleCampos();
    _tipoConexion = "Arduino";
    BtnArduino.Enabled = false;
    BtnMQTT.Enabled = true;
    PcbConexion.Image = Properties.Resources.Arduino;
}
```

InvisibleCampos();

Oculta los campos innecesarios para Arduino

_tipoConexion = "Arduino";

Guarda Arduino como tipo de conexión seleccionada

BtnArduino.Enabled = false;

Desactiva el boton de Arduino para indicar que ya está seleccionado

BtnMQTT.Enabled = true;

PcbConexion.Image = Properties.Resources.Arduino;

Cambia la imagen de *PcbConexion* a "*Arduino*"

Método para ocultar campos visibles

```
private void InvisibleCampos()
{
    label2.Visible = false;
    label3.Visible = false;
    label5.Visible = false;
    LblHost.Visible = false;
    LblPuerto.Visible = false;
    LblTopico.Visible = false;
    LblBroker.Visible = false;
    LblCOM.Visible = true;
    TxtHost.Visible = false;
    TxtPuerto.Visible = false;
    TxtTopico.Visible = false;
    BtnLimpiarCampos.Visible = false;
    BtnHivemq.Visible = false;
    BtnMosquitto.Visible = false;
}
```

Oculta las etiquetas, cajas de texto y botones relacionados con MQTT, solo muestra el campo LblCOM (En este nos dice el puerto a el cual está unido por conexión serial)

Botón para seleccionar conexión MQTT

```
private async void BtnMQTT_Click(object sender, EventArgs e)
{
    VisibleCampos();
    _tipoConexion = "MQTT";
    BtnMQTT.Enabled = false;
    BtnArduino.Enabled = true;
    PcbConexion.Image = Properties.Resources.MQTT;
}
```

VisibleCampos();

Hace lo contrario de *BtnArduino_click* mostrando los campos para MQTT y ocultando los de Arduino.

_tipoConexion = "MQTT";

Marca “MQTT” como el tipo de conexión

BtnMQTT.Enabled = false;

BtnArduino.Enabled = true;

Ajusta los botones

PcbConexion.Image = Properties.Resources.MQTT;

Implementa la imagen “MQTT” en *PcbConexion*

Método para mostrar los campos de MQTT

```
private void VisibleCampos()
{
    label2.Visible = true;
    label3.Visible = true;
    label5.Visible = true;
    LblHost.Visible = true;
    LblPuerto.Visible = true;
    LblTopico.Visible = true;
    LblBroker.Visible = true;
    LblCOM.Visible = false;
    TxtHost.Visible = true;
    TxtPuerto.Visible = true;
    TxtTopico.Visible = true;
    BtnLimpiarCampos.Visible = true;
    BtnHivemq.Visible = true;
    BtnMosquitto.Visible = true;
}
```

Muestra los elementos necesarios para la conexión mediante MQTT tales como lo son:

- Host
- Puerto
- Tópico
- Etc..

Oculta el campo LblCOM de Arduino

Botón para conexión (Para ambos modos)

```
private async void BtnConectar_Click(object sender, EventArgs e)
{
    if (_tipoConexion == "MQTT")
    {
        ConectarMQTT();
    }
    else if (_tipoConexion == "Arduino")
    {
        ConectarArduino();
    }
    else
    {
        MessageBox.Show("Seleccione un tipo de conexión");
    }
}
```

Se verifica el tipo de conexión seleccionada si es “MQTT” o “Arduino” esto llama al método establecido según sea el caso ConectarMQTT() o ConectarArduino(), y si se intenta conectar sin llegar a seleccionar ninguno entonces envía una advertencia.

Conectar por MQTT

```
    private void ConectarMQTT()
    {
        try
        {
            _host = TxtHost.Text;
            _puerto = TxtPuerto.Text;
            _topico = TxtTopico.Text;

            if (_puerto != "")
            {
                _client = new MqttClient(_host, Convert.ToInt32(_puerto), false, null, null,
MqttSslProtocols.None);
            }
            else
            {
                _client = new MqttClient(_host);
            }

            _client.MqttMsgPublishReceived += client_MqttMsgPublishReceived;
            string clientId = Guid.NewGuid().ToString();
            _client.Connect(clientId);
            if (_client.IsConnected)
```

```
            {
                _client.Subscribe(new string[] { _topico }, new byte[] {
MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE });
                BtnConectar.Enabled = false;
                MessageBox.Show("☑ Conexión a MQTT exitosa");
            }
            else
            {
                MessageBox.Show("❌ No se pudo establecer conexión con el servidor
MQTT.");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("❌ Error al conectar: " + ex.Message);
        }
    }
```

private void ConectarMQTT()

Obtiene los valores del host, puerto y tópico desde los Textbox (Cajas de Texto), esto crea un cliente MQTT y se conecta al bróker suscribiéndose a un tópico especifico y nos envía un mensaje según sea el caso pues puede llegar a ser un éxito o un error.

_client.MqttMsgPublishReceived += client_MqttMsgPublishReceived;

Asocia los eventos que se disparan cuando se recibe un mensaje MQTT. *string clientId = Guid.NewGuid().ToString();* Genera un identificador único para el cliente MQTT
Conector a Arduino por puerto serial

```
    private void ConectarArduino()
    {
        try
        {
            string puertoCOM = "COM3"; // Cambiar según el puerto que uses
            int velocidad = 9600; // La misma que uses en Arduino

            _serialPort = new SerialPort(puertoCOM, velocidad);
            _serialPort.DataReceived += SerialPort_DataReceived;
            _serialPort.Open();

            BtnConectar.Enabled = false;
            MessageBox.Show("☑ Conexión a Arduino exitosa por puerto " +
puertoCOM);
        }
        catch (Exception ex)
        {
            MessageBox.Show("✖ Error al conectar a Arduino: " + ex.Message);
        }
    }
```

private void ConectarArduino()
{ try { string puertoCOM = "COM3";

Simula una conexión con Arduino además de que define el puesto serial “COM3”, este puede cambiar de acuerdo con el que uses

int velocidad = 9600;

Define la velocidad de comunicación en baudios, esta debe ser la misma tanto en el Arduino como en el código 9600 en este caso.

_serialPort = new SerialPort(puertoCOM, velocidad);

Crea un nuevo objeto SerialPort el cual es almacenado en el atributo *_serialPort* siendo este accesible para todas las clases.

_serialPort.DataReceived += SerialPort_DataReceived;

Se suscribe al evento *DataReceived* este se dispara automáticamente cada vez que llega un nuevo dato desde Arduino.

_serialPort.Open();

Abre el puerto serial para que se comience a comunicar.

BtnConectar.Enabled = false;

Desactiva el botón

MessageBox.Show("Conexión a MQTT exitosa");

Envia un mensaje confirmando la conexión si esta es exitosa

} catch (Exception ex)

{ MessageBox.Show("Error al conectar a Arduino: " + ex.Message); }

Captura errores (si el puerto está ocupado o en su defecto no existe, etc.) mostrando un mensaje de error.

Evento para recibir datos desde Arduino

```
    private void SerialPort_DataReceived(object sender, SerialDataReceivedEventArgs
e)
    {
        try
        {
            string mensaje = _serialPort.ReadLine(); // Leer línea completa
            this.BeginInvoke(new Action(() => ProcesarMensaje(mensaje))); // Llama al
hilo de UI
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error al leer datos del Arduino: " + ex.Message);
        }
    }
```

private void SerialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)

{ try

{ string mensaje = _serialPort.ReadLine(); }

Es para leer los datos en una línea completa del puerto

this.BeginInvoke(new Action(() => ProcesarMensaje(mensaje)));

Llama a *ProcesarMensaje* desde el hilo de la interfaz gráfica (*UI*) porque el *DataReceived* trabaja en un hilo en segundo plano, este no puede modificar la interfaz directamente desde allí.

catch (Exception ex)

{ MessageBox.Show("Error al leer datos del Arduino: " + ex.Message); }

Captura errores mediante la duración de la lectura serial.

Mensaje recibido desde MQTT

```
void client_MqttMsgPublishReceived(object sender, MqttMsgPublishEventArgs e)
{
    string mensaje = System.Text.Encoding.UTF8.GetString(e.Message);
    ProcesarMensaje(mensaje);
}
```

Este evento se ejecuta cuando se recibe el mensaje por MQTT convirtiendo los datos binarios en mensajes en texto (UTF-8), para luego llamas al método *ProcesarMensaje*() para tratar los datos al igual que en Arduino

Procesar el mensaje recibido

```
    private void ProcesarMensaje(string mensaje)
    {
        try
        {
            dynamic datos = Newtonsoft.Json.JsonConvert.DeserializeObject(mensaje);
            _temperatura = datos.temperatura.ToString();
            _co2 = datos.co2.ToString();
            _ph = datos.ph.ToString();

            DatosRecibidos?.Invoke(_temperatura, _co2, _ph);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error al procesar mensaje: " + ex.Message + "\nMensaje recibido: " + mensaje);
        }
    }
```

private void ProcesarMensaje(string mensaje)

{try

{ dynamic datos = Newtonsoft.Json.JsonConvert.DeserializeObject(mensaje); }

Usa *Newtonsoft.Json* para de esta forma convertir el texto JSON recibido en un objeto dinámico.

_temperatura = datos.temperatura.ToString();

_co2 = datos.co2.ToString();

_ph = datos.ph.ToString();

Extrae y guarda los datos del mensaje recibido.

DatosRecibidos?.Invoke(_temperatura, _co2, _ph);

Dispara el evento *DatosRecibidos*() notificando a otros formularios o componentes que ya hay nuevos datos disponibles.

}catch (Exception ex)

{MessageBox.Show("Error al procesar mensaje: " + ex.Message + "\nMensaje recibido: " + mensaje); } }

Si el mensaje está mal formado, un ejemplo de esto es que no sea un JSON, muestra un error indicando el contenido del fallo.

Botón para autocompletar bróker MQTT públicos Mosquitto

```
private void BtnMosquitto_Click(object sender, EventArgs e)
{
    TxtHost.Text = "test.mosquitto.org";
    TxtPuerto.Text = "1883";
    TxtTopico.Text = "sensor/datos";
    LblBroker.Text = "Seleccione tipo de conexión";
    LblCOM.Visible = false;
    BtnLimpiarCampos.Visible = true;
}
```

Cuando se hace clic en este se llama automáticamente los campos configurados para que el bróker público Mosquitto.

HiveMQ

```
        private void BtnHivemq_Click(object sender, EventArgs e)
        {
            TxtHost.Text = "broker.hivemq.com";
            TxtPuerto.Text = "1883";
            TxtTopico.Text = "sensor/datos";
            LblBroker.Text = "Seleccione tipo de conexión";
            LblCOM.Visible = false;
            BtnLimpiarCampos.Visible = true;
        }
    }
}
```

Hace lo mismo que el anterior solo que en este caso con el bróker público de Hivemq.

Anexo 4 Dashboard

El formulario nombrado Dashboard es donde podemos observar las gráficas, el historial más reciente, los valores máximos y mínimos, así como los últimos recibidos, es donde se establece toda la lógica de la aplicación en base a los valores que llegan por la conexión establecida, dándonos lo valores de las variables temperatura, co2 y ph, siendo estos los considerados más importantes en el proceso de fermentación.

Figura 20. Dashboard.

Fuente: Autoría Propia

Este formulario cuenta con los siguientes elementos:

1. Etiquetas.

a. LblTitulo

b. LblLote

c. LblTemperatura

d. LblValTem

e. LblCO2

f. LblValCO2

g. LblPH

h. LblValpH

i. LblEstadoPH

j. LblPhEstado

k. LblTiempo

l. LblTieSeg

m. LblError

n. LblMyMTem

o. LblMyMCO2

p. LblMyMPH

q. LblValMax

r. LblMaxTem

s. LblMaxCO2

t. LblMaxPH

u. LblValMin

v. LblMinTem

w. LblMinCO2

x. LblMinPH

2. Botones:

a. BtnLote

3. Caja combinada:

a. CbxCambioTemp

b. CbxLote

4. Cajas de texto:

a. TxtLote

5. Casilla de verificación:

a. Cbx5Min

b. Cbx10Min

c. Cbx20Min

d. Cbx60Min

6. Vista de cuadrícula de datos:

a. DgvHistorial

7. Gráfico:

a. CtLineal

b. CtBarras

8. Paneles:

a. panel1

b. panel2

c. panel3

d. panel4

e. panel5

f. panel6

g. panel7

h. panel8

Tabla 11.

Explicación Codigo Dashboard.cs

Fuente: Autoría Propia

Espacio de nombres y declaraciones iniciales

```
using Interfaz_grafica.Controlador;
using MQTTnet;
using MQTTnet.Client;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Globalization;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

Importa las librerías necesarias para:

- Interfaz gráfica (Windows Forms)
- Comunicación serial (puerto COM)
- Conexión MQTT (Protocolo IoT)
- Acceso a bases de datos SQL Server

Clase principal

```
namespace Interfaz_grafica
{
    public partial class Dashboard : Form
    {
        ...
    }
}
```

Es la clase principal de Dashboard

Variables miembros

```
        private SerialPort serialPort;
        private IMqttClient mqttClient;
        private Timer historialTimer;
        private float maxTemp = float.MinValue;
        private float minTemp = float.MaxValue;
        private int maxCO2 = int.MinValue;
        private int minCO2 = int.MaxValue;
        private float maxPH = float.MinValue;
        private float minPH = float.MaxValue;
        private Conexion conexionForm;
        private string loteActual = "";
        private Timer barrasTimer;
        private List<float> tempHistory = new List<float>();
        private List<int> co2History = new List<int>();
        private List<float> phHistory = new List<float>();

        private readonly string dbConnection = "Data Source=.;Initial
Catalog=bd;Integrated Security=True";
        private bool isDisposing = false;
```

Componentes principales:

- *serialPort:* Se comunica con dispositivos seriales (*Arduino*)
- *mqttClient:* Para la comunicación MQTT con bróker IoT
- *historialTimer:* Temporizador para actualizar el historial periódicamente

Variables de seguimiento:

- Valores máximos y mínimos (*Temperatura, co2 y ph*).
- Historiales de lectura (*listas*)
- Conexión a base de datos SQL Server (Microsoft, Microsoft Ignite, 2015)
- Bandera *isDisposing* para manejo seguro de recursos.

Constructor

```
public Dashboard(Conexion conexion)
{
    InitializeComponent();
    conexionForm = conexion;

    conexionForm.DatosRecibidos += ConexionForm_DatosRecibidos;
    InitializeUI();
    InitializeDatabase();
    InitializeConnections();
}
```

Flujo de inicialización:

1. Configuración inicial de componentes de la interfaz
2. Suscripción a eventos de recepción de datos
3. Inicialización de la interfaz de usuario
4. Configuración de la base de datos.
5. Establecimiento de conexión (*serial y MQTT*)

Manejo de eventos de datos recibidos

```
    private void ConexionForm_DatosRecibidos(string temperatura, string co2,
string ph)
    {
        if (this.InvokeRequired)
        {
            this.Invoke(new Action(() => ConexionForm_DatosRecibidos(temperatura,
co2, ph)));
            return;
        }

        // Mostrar en labels
        LblTemperatura.Text = $"{temperatura} °C";
        LblCO2.Text = $"{co2} ppm";
        LblPH.Text = $"{ph} pH";

        // Agregar a gráficas
        float temp = float.Parse(temperatura, CultureInfo.InvariantCulture);
        int co2Value = int.Parse(co2);
        float phValue = float.Parse(ph, CultureInfo.InvariantCulture);

        //Engrosar las lineas
```

```
        CtLineal.Series["Temperatura"].BorderWidth = 12;
        CtLineal.Series["CO2"].BorderWidth = 12;
        CtLineal.Series["pH"].BorderWidth = 12;

        CtLineal.Series["Temperatura"].Points.AddY(temp);
        CtLineal.Series["CO2"].Points.AddY(co2Value);
        CtLineal.Series["pH"].Points.AddY(phValue);

        CtBarras.Series["Temperatura"].Points.Clear();
        CtBarras.Series["CO2"].Points.Clear();
        CtBarras.Series["pH"].Points.Clear();
        CtBarras.Series["Temperatura"].Points.AddY(temp);
        CtBarras.Series["CO2"].Points.AddY(co2Value);
        CtBarras.Series["pH"].Points.AddY(phValue);
    }
```

Características clave:

- Manejo seguro entre hilos con *InvokeRequired* se verifica si necesita invocarse en el hilo de UI
- Actualización de controles de la interfaz
- Procesamiento de valores numéricos con cultura invariante
- Actualización de gráficos lineales y de barras

Inicialización de componentes (Interfaz de usuario)

```
        private void InitializeUI()
        {
            // Configurar ComboBox de temperatura
            CbxCambioTemp.Items.AddRange(new string[] { "°C", "°F", "°K" });
            CbxCambioTemp.SelectedIndex = 0;
            CbxCambioTemp.SelectedIndexChanged +=
CbxCambioTemp_SelectedIndexChanged;

            // Configurar eventos de checkboxes
            if (Cbx5Min != null) Cbx5Min.CheckedChanged +=
CheckboxInterval_CheckedChanged;
            if (Cbx10Min != null) Cbx10Min.CheckedChanged +=
CheckboxInterval_CheckedChanged;
            if (Cbx20Min != null) Cbx20Min.CheckedChanged +=
CheckboxInterval_CheckedChanged;
            if (Cbx60Min != null) Cbx60Min.CheckedChanged +=
CheckboxInterval_CheckedChanged;

            // Inicializar gráficos si existen
            InitializeCharts();
        }
```

Configuración de combobox mediante las unidades de temperatura y se selecciona por defecto los "°*C*", configuración de checkbox mediante los intervalos e inicializados de gráficos
Inicializador de gráficos

```
    private void InitializeCharts()
    {
        try
        {
            if (CtLineal != null)
            {
                CtLineal.Series.Clear();
                var tempSeries = CtLineal.Series.Add("Temperatura");
                var co2Series = CtLineal.Series.Add("CO2");
                var phSeries = CtLineal.Series.Add("pH");

                tempSeries.ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                co2Series.ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                phSeries.ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
            }

            if (CtBarras != null)
            {
                CtBarras.Series.Clear();
                var tempBarSeries = CtBarras.Series.Add("Temperatura");
```

```
            var co2BarSeries = CtBarras.Series.Add("CO2");
            var phBarSeries = CtBarras.Series.Add("pH");

            tempBarSeries.ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Column;
            co2BarSeries.ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Column;
            phBarSeries.ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Column;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error inicializando gráficos: {ex.Message}");
    }
}
```

Se configura el grafico lineal:

- Limpian los campos existentes
- Crea series para cada métrica
- Configuración de tipos de gráficos

Se configura el grafico de barras de la misma forma en que se configuro el grafico lineal.

Inicializador de base de datos

```
    private void InitializeDatabase()
    {
        try
        {
            VerificarOCrearBaseDeDatos();
            VerificarOCrearTablaDatos();
            MostrarHistorial();
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error inicializando base de datos: {ex.Message}",
"Error",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
```

VerificarOCrearBaseDeDatos(): Crea BD si no existe

VerificarOCrearTablaDatos(): Crea tabla si no existe

MostrarHistorial(): Muestra datos históricos

catch (Exception ex): se atrapan posibles errores

Inicializador de Conexión serial (Arduino)

```
private async void InitializeConnections()
{
    InicializarSerial();
    await InicializarMQTT();
    IniciarTimer();
}
```

Se inicializa la conexión de la conexión serial y de IoT.

Conexión serial

```
    private void InicializarSerial()
    {
        try
        {
            // Verificar puertos disponibles
            string[] puertos = SerialPort.GetPortNames();
            string puertoSeleccionado = puertos.Contains("COM3") ? "COM3" :
                                        puertos.Length > 0 ? puertos[0] : null;

            if (puertoSeleccionado != null)
            {
                serialPort = new SerialPort(puertoSeleccionado, 115200);
                serialPort.DataReceived += SerialPort_DataReceived;
                serialPort.ReadTimeout = 1000;
                serialPort.WriteTimeout = 1000;
                serialPort.Open();
                Console.WriteLine($"Puerto serial {puertoSeleccionado} abierto
correctamente");
            }
            else
            {
                Console.WriteLine("No se encontraron puertos seriales disponibles");
            }
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error inicializando puerto serial: {ex.Message}");
        }
    }
```

Se obtienen los puertos disponibles, preferiblemente se toma el *COM3* o de lo contrario será el primero disponible, además de esto se configura el puerto serial y se toma el evento para los datos recibidos, estos se leen y se escriben cuando se abre la conexión.

Recibir y procesar datos de Arduino

```
    private void SerialPort_DataReceived(object sender,
SerialDataReceivedEventArgs e)
    {
        try
        {
            if (serialPort != null && serialPort.IsOpen && !isDisposing)
            {
                string mensaje = serialPort.ReadLine().Trim();
                if (!string.IsNullOrEmpty(mensaje) && this.IsHandleCreated)
                {
                    this.Invoke((MethodInvoker)(() => ProcesarDatos(mensaje)));
                }
            }
        }
        catch (TimeoutException)
        {
            // Timeout es normal, no hacer nada
        }
        catch (Exception ex)
        {
            if (!isDisposing)
            {
                Console.WriteLine($"Error leyendo puerto serial: {ex.Message}");
```

```
            }
        }
    }
```

Este es el evento manejador que se ejecuta automáticamente cuando llega un nuevo dato al *SerialPort*.

sender: es el objeto que envió el evento (normalmente el serialPort).

SerialDataReceivedEventArgs e: contiene datos del evento, aunque no siempre se usa.

Se verifica que *serialPort* no sea *null*.

El puerto esté abierto *(IsOpen == true).*

!isDisposing: significa que no se está cerrando o liberando el puerto (variable de control para evitar fallos si se está cerrando el formulario).

{ string mensaje = serialPort.ReadLine().Trim();

Lee una línea de texto enviada por Arduino, hasta encontrar un salto de línea *(\n o \r\n).*

Trim() elimina espacios o saltos extra al principio o final.

if (!string.IsNullOrEmpty(mensaje) && this.IsHandleCreated)

Verifica que:

- El mensaje no esté vacío o nulo.
- El formulario ya tenga un *handle* creado (es decir, que está completamente cargada).

{ this.Invoke((MethodInvoker)(() => ProcesarDatos(mensaje)));

Usa *Invoke* para ejecutar el código en el hilo de la interfaz gráfica, ya que el evento *DataReceived* corre en un hilo en segundo plano.

ProcesarDatos(mensaje): llama a tu método para interpretar y usar los datos recibidos *catch (TimeoutException)* Ignora errores por tiempo de espera (cuando no hay datos en el tiempo límite). Esto es común y pues no representa un fallo grave. *catch (Exception ex)* Captura los errores inesperados como lecturas corruptas o puertos desconectados, esto envía el error por consola solo si no está cerrado el programa (!isDisposing)
Verificación y creación de bases de datos

```
        private void VerificarOCrearBaseDeDatos()
        {
            string masterConnection = "Data Source=.;Initial
Catalog=master;Integrated Security=True;Connection Timeout=30;";
            try
            {
                using (SqlConnection conn = new SqlConnection(masterConnection))
                {
                    conn.Open();
                    string checkDbQuery = "IF DB_ID('bd') IS NULL CREATE DATABASE
bd;";
                    using (SqlCommand cmd = new SqlCommand(checkDbQuery, conn))
                    {
                        cmd.ExecuteNonQuery();
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception($"Error creando base de datos: {ex.Message}");
            }
        }
```

Se verifica si esta creada la base de datos y si a esta esta creada entonces no se hace nada, sin embargo, si no está creada se crea pues es donde estarán ubicadas las tablas.
Verificación o creación de tablas

```
private void VerificarOCrearTablaDatos()
{
    try
    {
        using (SqlConnection conn = new SqlConnection(dbConnection))
        {
            conn.Open();
            string createTableQuery = @"
            IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'Datos')
            BEGIN
                CREATE TABLE Datos (
                    Id INT IDENTITY(1,1) PRIMARY KEY,
                    Lote NVARCHAR(100),
                    Temperatura FLOAT,
                    CO2 INT,
                    PH FLOAT,
                    Fecha DATETIME DEFAULT GETDATE()
                );
            END";
            using (SqlCommand cmd = new SqlCommand(createTableQuery, conn))
            {
                cmd.ExecuteNonQuery();
            }
```

```
            }
        }
        catch (Exception ex)
        {
            throw new Exception($"Error creando tabla: {ex.Message}");
        }
    }
```

Se verifica que este creada la tabla si no está creada entonces esta se crea para guardar los datos de las variables temperatura, co2 y ph

Conexión MQTT

```
private async Task InicializarMQTT()
{
    try
    {
        var factory = new MqttFactory();
        mqttClient = factory.CreateMqttClient();

        var opciones = new MqttClientOptionsBuilder()
            .WithClientId(Guid.NewGuid().ToString())
            .WithTcpServer("broker.hivemq.com", 1883)
            .WithCleanSession()
            .WithKeepAlivePeriod(TimeSpan.FromSeconds(60))
            .Build();

        mqttClient.ConnectedAsync += async e =>
        {
            try
            {
                var subscribeOptions = new MqttClientSubscribeOptionsBuilder()
                    .WithTopicFilter("sensor/datos")
                    .Build();
                await mqttClient.SubscribeAsync(subscribeOptions);
```

```
                Console.WriteLine("Conectado y suscrito a MQTT");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error suscribiéndose a MQTT: {ex.Message}");
            }
        };

        mqttClient.DisconnectedAsync += async e =>
        {
            if (!isDisposing)
            {
                Console.WriteLine("Desconectado de MQTT, intentando
reconectar...");
                await Task.Delay(5000);
                try
                {
                    await mqttClient.ConnectAsync(opciones);
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Error reconectando MQTT: {ex.Message}");
                }
```

```
            }
        };

        mqttClient.ApplicationMessageReceivedAsync += e =>
        {
            try
            {
                string mensaje =
Encoding.UTF8.GetString(e.ApplicationMessage.Payload);
                if (this.IsHandleCreated && !isDisposing)
                {
                    this.Invoke((MethodInvoker)(() => ProcesarDatos(mensaje)));
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error procesando mensaje MQTT:
{ex.Message}");
            }
            return Task.CompletedTask;
        };

        await mqttClient.ConnectAsync(opciones);
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error inicializando MQTT: {ex.Message}");
        }
    }
```

Es un **método asíncrono** (async) que devuelve una Task, ideal para operaciones de red como MQTT, donde hay espera, esto se puede ejecutar con *await*

InicializarMQTT();

var factory = new MqttFactory();

mqttClient = factory.CreateMqttClient();

Crea una fábrica de objetos MQTT y luego crea el cliente MQTT (*mqttClient*) que usaremos para conectarnos al bróker, esto se configura en este caso con conexión por hivemq con sus respectivos datos (ID único, Bróker publico), después de esto se dispara el evento de conexión y se suscribe al tópico.

mqttClient.ConnectedAsync += async e =>

Este bloque se ejecuta automáticamente una vez el cliente se conecta exitosamente, si se desconecta se inicializa *mqttClient.DisconnectedAsync += async e =>* el cual espera cinco segundos y reinicia la conexión, una vez se confirma entonces se inicializa la conexión

isDisposing es una bandera que evita reconectar si la aplicación se está cerrando.

mqttClient.ApplicationMessageReceivedAsync += e =>

Cuando llega un mensaje MQTT:

- Lo convierte de bytes a texto UTF-8.
- Verifica que el formulario esté listo *(IsHandleCreated)* y no se esté cerrando *(!isDisposing).*
- Llama a *ProcesarDatos(mensaje)* usando *Invoke()* para que se ejecute en el hilo de la UI (es seguro para modificar controles como *Label, Textbox, etc).*

await mqttClient.ConnectAsync(opciones);

Intenta conectarse al bróker con las opciones configuradas.

catch (Exception ex)

Captura errores generales durante la configuración o conexión.

Método de procesamiento de datos

```
    public void ProcesarDatos(string mensaje)
    {
        try
        {
            if (string.IsNullOrWhiteSpace(mensaje) || isDisposing) return;

            Console.WriteLine($"Mensaje recibido: {mensaje}");

            float temperatura = 0;
            int co2 = 0;
            float ph = 0;
            bool tempValida = false, co2Valido = false, phValido = false;

            if (mensaje.StartsWith("{")) // Posible JSON
            {
                try
                {
                    dynamic datos =
Newtonsoft.Json.JsonConvert.DeserializeObject(mensaje);
                    if (datos != null)
                    {
                        if (datos.temperatura != null)
```

```
            {
                temperatura = (float)datos.temperatura;
                tempValida = true;
            }
            if (datos.co2 != null)
            {
                co2 = (int)datos.co2;
                co2Valido = true;
            }
            if (datos.ph != null)
            {
                ph = (float)datos.ph;
                phValido = true;
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error al interpretar JSON: " + ex.Message);
    }
}
else // Estilo Arduino: clave:valor,clave:valor
{
```

```
                string[] partes = mensaje.Split(',');
                foreach (string parte in partes)
                {
                    string[] subpartes = parte.Split(':');
                    if (subpartes.Length != 2) continue;

                    string clave = subpartes[0].Trim().ToLower();
                    string valor = subpartes[1].Trim();

                    if (clave == "temperatura" || clave == "temp")
                    {
                        if (float.TryParse(valor, NumberStyles.Float,
CultureInfo.InvariantCulture, out temperatura))
                            tempValida = true;
                    }
                    else if (clave == "co2")
                    {
                        if (int.TryParse(valor, out co2))
                            co2Valido = true;
                    }
                    else if (clave == "ph")
                    {
```

```
                    if (float.TryParse(valor, NumberStyles.Float,
CultureInfo.InvariantCulture, out ph))
                        phValido = true;
                }
            }
        }

        // Solo actualizamos si hay algún dato válido
        if (tempValida || co2Valido || phValido)
        {
            ActualizarUI(temperatura, co2, ph, tempValida, co2Valido, phValido);
            GuardarEnBaseDeDatos(temperatura, co2, ph, tempValida, co2Valido,
phValido);
        }
        else
        {
            Console.WriteLine("Mensaje recibido sin datos válidos.");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error general procesando datos: {ex.Message}");
    }
```

}
if (string.IsNullOrWhiteSpace(mensaje) || isDisposing) return;
Validación inicial: mensaje vacío o aplicación cerrándose
Se imprimen mensajes por consola y se almacenan los valores de las variables
if (mensaje.StartsWith("{"))
Verifica si el mensaje parece ser JSON
dynamic datos = Newtonsoft.Json.JsonConvert.DeserializeObject(mensaje);
Deserializa el JSON (Newtonsoft, 2022) dinámicamente
Else { ... }
Formato estilo Arduino "clave:valor"
if (tempValida || co2Valido || phValido)
Si se obtuvo al menos un dato válido
Método Actualizar UI

```
    private void ActualizarUI(float temperatura, int co2, float ph, bool tempValida,
bool co2Valido, bool phValido)
    {
        try
        {
            if (tempValida && LblValTem != null)
            {
                LblValTem.Text = temperatura.ToString("F2"); //  ◇ Dos decimales
                ConvertirTemperatura(temperatura);
            }

            if (co2Valido && LblValCO2 != null)
            {
                LblValCO2.Text = co2.ToString("D"); //  ◇ Entero
            }

            if (phValido && LblValpH != null)
            {
                LblValpH.Text = ph.ToString("F2"); //  ◇ Dos decimales

                if (LblPhEstado != null)
                {
```

```
                LblPhEstado.Text = ph < 7 ? "Ácido" : ph > 7 ? "Alcalino" :
"Neutro";
            }
        }

        // Valores máximos y mínimos
        ActualizarMinMax(temperatura, co2, ph, tempValida, co2Valido,
phValido);

        // Historial
        ActualizarHistorial(temperatura, co2, ph, tempValida, co2Valido,
phValido);

        Console.WriteLine($"UI actualizada - Temp: {(tempValida ?
temperatura.ToString("F2") : "N/A")}," +
            $" CO2: {(co2Valido ? co2.ToString() : "N/A")}, pH: {(phValido ?
ph.ToString("F2") : "N/A")}");
      }
      catch (Exception ex)
      {
        Console.WriteLine($"Error actualizando UI: {ex.Message}");
      }
    }
```

private void ActualizarUI(float temperatura, int co2, float ph, bool tempValida, bool co2Valido, bool phValido)

Este método actualiza los elementos gráficos (UI) con los nuevos valores de temperatura, CO2 y pH, siempre que los valores sean válidos

Try { ... }

- Si la temperatura es válida y el label de temperatura *(LblValTem)* no es nulo.
- Se muestra el valor del pH y se determina si es Ácido, Neutro o Alcalino.
- Actualiza los valores máximos y mínimos históricos
- Guarda los datos en listas para graficarlos
- Muestra los datos en consola

Método ActualizarMinMax(...)

```
    private void ActualizarMinMax(float temperatura, int co2, float ph, bool
tempValida, bool co2Valido, bool phValido)
    {
        // Actualizar temperatura
        if (tempValida)
        {
            maxTemp = maxTemp == float.MinValue ? temperatura :
Math.Max(maxTemp, temperatura);
            minTemp = minTemp == float.MaxValue ? temperatura :
Math.Min(minTemp, temperatura);
            if (LblMaxTem != null) LblMaxTem.Text = maxTemp.ToString("F2");
            if (LblMinTem != null) LblMinTem.Text = minTemp.ToString("F2");
        }

        // Actualizar CO2
        if (co2Valido)
        {
            maxCO2 = maxCO2 == int.MinValue ? co2 : Math.Max(maxCO2, co2);
            minCO2 = minCO2 == int.MaxValue ? co2 : Math.Min(minCO2, co2);
            if (LblMaxCO2 != null) LblMaxCO2.Text = maxCO2.ToString();
            if (LblMinCO2 != null) LblMinCO2.Text = minCO2.ToString();
        }
```

```
        // Actualizar pH
        if (phValido)
        {
            maxPH = maxPH == float.MinValue ? ph : Math.Max(maxPH, ph);
            minPH = minPH == float.MaxValue ? ph : Math.Min(minPH, ph);
            if (LblMaxPH != null) LblMaxPH.Text = maxPH.ToString("F2");
            if (LblMinPH != null) LblMinPH.Text = minPH.ToString("F2");
        }
    }
```

Este método actualiza los valores mínimos y máximos de temperatura, CO2 y pH, comparando los nuevos datos con los actuales.

if (tempValida) { ... } Si aún no hay max, se usa el valor actual; si ya hay, se compara

Similar lógica se aplica para *co2* y *ph*.

Método: ActualizarHistorial(...)

```
    private void ActualizarHistorial(float temperatura, int co2, float ph, bool
tempValida, bool co2Valido, bool phValido)
    {
        const int maxHistorySize = 15;

        // Agregar valores al historial - mantener sincronizadas las listas
        // Si no hay valor válido, usar el último valor conocido o 0
        float tempParaHistorial = tempValida ? temperatura : (tempHistory.Count >
0 ? tempHistory.Last() : 0);
        int co2ParaHistorial = co2Valido ? co2 : (co2History.Count > 0 ?
co2History.Last() : 0);
        float phParaHistorial = phValido ? ph : (phHistory.Count > 0 ?
phHistory.Last() : 0);

        tempHistory.Add(tempParaHistorial);
        co2History.Add(co2ParaHistorial);
        phHistory.Add(phParaHistorial);

        // Mantener solo los últimos valores
        if (tempHistory.Count > maxHistorySize) tempHistory.RemoveAt(0);
        if (co2History.Count > maxHistorySize) co2History.RemoveAt(0);
        if (phHistory.Count > maxHistorySize) phHistory.RemoveAt(0);
```

```
        Console.WriteLine($"Historial actualizado - Temp: {tempHistory.Count}
valores, CO2: {co2History.Count} valores, pH: {phHistory.Count} valores");

        // Actualizar gráficos
        ActualizarGraficoLineal();
        if (DebeActualizarGraficoBarras())
        {
            ConfigurarTimerBarras();
            ActualizarGraficoBarras();
        }
    }
```

Este método guarda los últimos 15 valores de temperatura, CO2 y pH en listas *(tempHistory, co2History, phHistory)* para graficar después de esto se llama *ActualizarGraficoLineal();* esto me tendrá los últimos datos en barras.

```
if (DebeActualizarGraficoBarras()){
   ConfigurarTimerBarras();
   ActualizarGraficoBarras(); }
```

Si algún checkbox de intervalo está seleccionado, entonces:

- Configura el temporizador con el tiempo correspondiente.
- Actualiza inmediatamente el gráfico de barras.

Método: DebeActualizarGraficoBarras()

```
private bool DebeActualizarGraficoBarras()
{
    ConfigurarTimerBarras();
    return (Cbx5Min?.Checked == true) ||
           (Cbx10Min?.Checked == true) ||
           (Cbx20Min?.Checked == true) ||
           (Cbx60Min?.Checked == true);
}
```

Este método revisa si alguno de los checkbox de intervalos está marcado

Método: ConfigurarTimerBarras()

```
private void ConfigurarTimerBarras()
{
    // Detener si ya está corriendo
    barrasTimer?.Stop();

    int intervalo = 0;

    if (Cbx5Min?.Checked == true)
        intervalo = 5 * 60 * 1000; // 5 minutos
    else if (Cbx10Min?.Checked == true)
        intervalo = 10 * 60 * 1000;
    else if (Cbx20Min?.Checked == true)
        intervalo = 20 * 60 * 1000;
    else if (Cbx60Min?.Checked == true)
        intervalo = 60 * 60 * 1000;

    if (intervalo > 0)
    {
        if (barrasTimer == null)
        {
            barrasTimer = new Timer();
            barrasTimer.Tick += (s, e) => ActualizarGraficoBarras();
        }
```

```
            barrasTimer.Interval = intervalo;
            barrasTimer.Start();

            Console.WriteLine($"Timer de barras configurado a cada {intervalo / 60000} min.");
        }
        else
        {
            // Si no hay checkbox seleccionado, detener el timer
            barrasTimer?.Stop();
            Console.WriteLine("Timer de barras detenido (sin checkbox activo).");
        }
    }
```

barrasTimer?.Stop() Si el timer ya estaba corriendo, lo detiene primero

int intervalo = 0; Se inicializa la variable que guardará el intervalo en milisegundos

Calcula el tiempo según el checkbox marcado (en milisegundos).

if (intervalo > 0) { ... }

- Si aún no existe el *barrasTimer*, se crea.
- Se le asigna el intervalo determinado.
- Se configura para llamar a *ActualizarGraficoBarras()* cada vez que se cumpla el tiempo.

Método ConvertirTemperatura(float celsius)

```
private void ConvertirTemperatura(float celsius)
{
    try
    {
        if (CbxCambioTemp?.SelectedItem == null || LblValTem == null) return;

        string tipo = CbxCambioTemp.SelectedItem.ToString();
        float valor;
        if (tipo == "°F")
        {
            valor = (celsius * 9f / 5f) + 32f;
        }
        else if (tipo == "°K")
        {
            valor = celsius + 273.15f;
        }
        else
        {
            valor = celsius;
        }

        LblValTem.Text = valor.ToString("F1");
    }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"Error convirtiendo temperatura: {ex.Message}");
        }
    }
```

Este método convierte una temperatura en Celsius a Fahrenheit o Kelvin dependiendo de la opción seleccionada en un *ComboBox*

demaci

Verifica que el ComboBox CbxCambioTemp tenga una opción seleccionada y que la etiqueta LblValTem no sea nula.

El operador ?. evita errores si CbxCambioTemp es nulo

Se puede cambiar también los valores de “°C” a “°K” o “°F”

Metodo ActualizarGraficoLineal()

```
    private void ActualizarGraficoLineal()
    {
        try
        {
            if (CtLineal?.Series == null) return;

            Console.WriteLine($"Actualizando gráfico lineal - Datos disponibles:
Temp={tempHistory.Count}, CO2={co2History.Count}, pH={phHistory.Count}");

            // Temperatura
            if (CtLineal.Series.FindByName("Temperatura") != null &&
tempHistory.Count > 0)
            {
                var serieTemp = CtLineal.Series["Temperatura"];
                serieTemp.Points.Clear();
                for (int i = 0; i < tempHistory.Count; i++)
                {
                    serieTemp.Points.AddXY(i, tempHistory[i]);
                }
                Console.WriteLine($"Serie Temperatura: {serieTemp.Points.Count}
puntos añadidos");
            }
```

```
            // CO2
            if (CtLineal.Series.FindByName("CO2") != null && co2History.Count >
0)
            {
               var serieCO2 = CtLineal.Series["CO2"];
               serieCO2.Points.Clear();
               for (int i = 0; i < co2History.Count; i++)
               {
                  serieCO2.Points.AddXY(i, co2History[i]);
               }
               Console.WriteLine($"Serie CO2: {serieCO2.Points.Count} puntos
añadidos");
            }

            // pH
            if (CtLineal.Series.FindByName("pH") != null && phHistory.Count > 0)
            {
               var seriePH = CtLineal.Series["pH"];
               seriePH.Points.Clear();
               for (int i = 0; i < phHistory.Count; i++)
               {
                  seriePH.Points.AddXY(i, phHistory[i]);
               }
```

```
                Console.WriteLine($"Serie pH: {seriePH.Points.Count} puntos
añadidos");
            }
            // Forzar actualización del gráfico
            CtLineal.Invalidate();
            CtLineal.Update();
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error actualizando gráfico lineal: {ex.Message}");
        }
    }
```

if (CtLineal?.Series == null) return;

esta línea de código me comprueba si el objeto del grafico (CtLineal) ya existe y que contiene este, además de esto si no hay una configuración no hace nada y finaliza el método.

Console.WriteLine($"Actualizando gráfico lineal - Datos disponibles: Temp={tempHistory.Count}, CO2={co2History.Count}, pH={phHistory.Count}");

Muestra en la consola cuántos datos hay en las listas *tempHistory*, *co2History* y *phHistory*

if (CtLineal.Series.FindByName("Temperatura") != null && tempHistory.Count > 0)

{

var serieTemp = CtLineal.Series["Temperatura"];

```
    serieTemp.Points.Clear();
    for (int i = 0; i < tempHistory.Count; i++)
    {
        serieTemp.Points.AddXY(i, tempHistory[i]);
    }
    Console.WriteLine($"Serie Temperatura: {serieTemp.Points.Count} puntos añadidos");
}
```

Este codigo busca la serie llamada “Temperatura”, verifica su existencia y si hay datos en *tempHistorial* limpia los puntos anteriores (*Points.Clear()*), también recorre la lista *tempHistorial* agregando los nuevos valores al grafico con su índice como eje X y finalmente, imprime cuantos puntos se añadieron.

La serie de CO2 hace exactamente lo mismo al igual que la serie de pH.

```
CtLineal.Invalidate();
CtLineal.Update();
```

Invalidate() indica que el control debe repintarse.

Update() fuerza el repintado inmediato para que los nuevos puntos aparezcan en pantalla sin esperar al próximo ciclo de refresco.

```
catch (Exception ex)
{
    Console.WriteLine($"Error actualizando gráfico lineal: {ex.Message}");
}
```

Con esta excepción se atrapan los errores por ejemplo los problemas de datos, escribiendo el error en consola en lugar de cerrar directamente la aplicación.
Metodo ActualizarGraficoBarras()

```
        private void ActualizarGraficoBarras()
        {
            try
            {
                if (CtBarras?.Series == null) return;

                Console.WriteLine("Actualizando gráfico de barras");

                // Temperatura
                if (CtBarras.Series.FindByName("Temperatura") != null &&
tempHistory.Count > 0)
                {
                    var serieTemp = CtBarras.Series["Temperatura"];
                    serieTemp.Points.Clear();
                    serieTemp.Points.AddXY("Temperatura", tempHistory.LastOrDefault());
                    Console.WriteLine($"Barra Temperatura:
{tempHistory.LastOrDefault()}");
                }

                // CO2
                if (CtBarras.Series.FindByName("CO2") != null && co2History.Count >
0)
                {
```

```
            var serieCO2 = CtBarras.Series["CO2"];
            serieCO2.Points.Clear();
            serieCO2.Points.AddXY("CO2", co2History.LastOrDefault());
            Console.WriteLine($"Barra CO2: {co2History.LastOrDefault()}");
        }

        // pH
        if (CtBarras.Series.FindByName("pH") != null && phHistory.Count > 0)
        {
            var seriePH = CtBarras.Series["pH"];
            seriePH.Points.Clear();
            seriePH.Points.AddXY("pH", phHistory.LastOrDefault());
            Console.WriteLine($"Barra pH: {phHistory.LastOrDefault()}");
        }

        // Forzar actualización del gráfico
        CtBarras.Invalidate();
        CtBarras.Update();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error actualizando gráfico de barras:
{ex.Message}");
```

```
        }
    }
```

if (CtBarras?.Series == null) return;

Se asegura de que el CtBarras exista y tenga las series configuradas pero si no las tiene este método no hace nada eso es lo que dice este condicional.

Console.WriteLine("Actualizando gráfico de barras");

Nos imprime por consola cuando la función fue llamada.

```
if (CtBarras.Series.FindByName("Temperatura") != null && tempHistory.Count > 0)
{
   var serieTemp = CtBarras.Series["Temperatura"];
   serieTemp.Points.Clear();
   serieTemp.Points.AddXY("Temperatura", tempHistory.LastOrDefault());
   Console.WriteLine($"Barra Temperatura: {tempHistory.LastOrDefault()}");
}
```

Este código busca la serie llamada “Temperatura”, rectifica que exista así como si esta vacía con *tempHistorial,* borra cualquier barra precia y añade las barras en los ejes X y Y, imprimiendo así los calores gráficos, esto mismo sucede con CO2 y pH.

```
CtBarras.Invalidate();
CtBarras.Update();
```

Fuerza que el control se redibuje con las nuevas barras

Finalmente se crea una excepción para atrapar cualquier error presente en la ejecución.

```
catch (Exception ex)
{
```

Console.WriteLine($"Error actualizando gráfico de barras: {ex.Message}"); *}*
Método GuardarEnBaseDeDatos(...)

```
    private void GuardarEnBaseDeDatos(float temperatura, int co2, float ph, bool
tempValida, bool co2Valido, bool phValido)
    {
        try
        {
            // Solo guardar si al menos uno de los valores es válido
            if (!tempValida && !co2Valido && !phValido) return;

            using (SqlConnection conn = new SqlConnection(dbConnection))
            {
                conn.Open();
                string insertQuery = "INSERT INTO Datos (Lote, Temperatura, CO2,
PH) VALUES (@Lote, @Temp, @CO2, @PH)";
                using (SqlCommand cmd = new SqlCommand(insertQuery, conn))
                {
                    cmd.Parameters.AddWithValue("@Lote", loteActual);
                    cmd.Parameters.AddWithValue("@Temp", tempValida ?
(object)temperatura : DBNull.Value);
                    cmd.Parameters.AddWithValue("@CO2", co2Valido ? (object)co2 :
DBNull.Value);
                    cmd.Parameters.AddWithValue("@PH", phValido ? (object)ph :
DBNull.Value);
                    cmd.ExecuteNonQuery();
```

```
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error guardando en base de datos: {ex.Message}");
    }
}
```

if (!tempValida && !co2Valido && !phValido) return; Si ninguno de los datos es válido, no guarda nada.

using (SqlConnection conn = new SqlConnection(dbConnection))

{ conn.Open(); Abre la conexión a la base de datos.

Inserta los valores válidos en la tabla *Datos*. Si un dato no es válido, guarda *NULL*.

Métodos relacionados al Timer e Historial

```
private void IniciarTimer()
{
    historialTimer = new Timer();
    historialTimer.Interval = 60000; // 1 minuto
    historialTimer.Tick += (s, e) => MostrarHistorial();
    historialTimer.Start();
}
```

Crea un temporizador que cada minuto ejecuta *MostrarHistorial*().

Método MostrarHistorial()

```
    private void MostrarHistorial()
    {
        try
        {
            if (DgvHistorial == null || isDisposing) return;

            using (SqlConnection conn = new SqlConnection(dbConnection))
            {
                conn.Open();
                SqlDataAdapter da = new SqlDataAdapter(
                    "SELECT TOP 20 Id, Lote, Temperatura, CO2, PH, Fecha FROM
Datos ORDER BY Fecha DESC",
                    conn);
                DataTable dt = new DataTable();
                da.Fill(dt);

                if (this.IsHandleCreated)
                {
                    this.Invoke((MethodInvoker)(() => DgvHistorial.DataSource = dt));
                }
            }
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"Error mostrando historial: {ex.Message}");
        }
    }
```

if (DgvHistorial == null || isDisposing) return; Si el *DataGridView* es nulo o el programa está cerrando, no hace nada.

using (SqlConnection conn = new SqlConnection(dbConnection))

Obtiene los 20 registros más recientes de la base de datos

if (this.IsHandleCreated)

Asigna los datos al *DataGridView* de forma segura desde el hilo principal.

Eventos de UI

```
    private void CbxCambioTemp_SelectedIndexChanged(object sender, EventArgs
e)
    {
        if (tempHistory.Count > 0)
        {
            ConvertirTemperatura(tempHistory.LastOrDefault());
        }
    }
```

Convierte la temperatura al cambiar el tipo *(°F, °K, °C)* en el ComboBox

Método CheckboxInterval_CheckedChanged

```
private void CheckboxInterval_CheckedChanged(object sender, EventArgs e)
{
    // Actualizar gráfico de barras cuando cambie la selección de intervalos
    if (DebeActualizarGraficoBarras())
    {
        ConfigurarTimerBarras();
        ActualizarGraficoBarras();
    }
}
```

Cuando cambias la selección de intervalos (5, 10, etc.), se actualiza el gráfico de barras.

Método Dispose(...)

```
protected override void Dispose(bool disposing)
{
    isDisposing = true;

    if (disposing)
    {
        try
        {
            historialTimer?.Stop();
            historialTimer?.Dispose();

            if (serialPort?.IsOpen == true)
            {
                serialPort.Close();
            }
            serialPort?.Dispose();

            if (mqttClient?.IsConnected == true)
            {
                mqttClient.DisconnectAsync().Wait(1000);
            }
            mqttClient?.Dispose();
        }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"Error durante dispose: {ex.Message}");
        }
    }

    base.Dispose(disposing);
}
```

Este método limpia los recursos *(Timers, puertos, conexiones)* antes de cerrar el formulario

Métodos para gestionar lotes

```
    private void BtnLote_Click(object sender, EventArgs e)
    {
        // Tomar el valor escrito en el TextBox
        string nuevoLote = TxtLote.Text.Trim();

        if (!string.IsNullOrEmpty(nuevoLote))
        {
            try
            {
                // Guardar el lote en la base de datos
                using (SqlConnection conn = new SqlConnection(dbConnection))
                {
                    conn.Open();

                    // Creamos la tabla de lotes si no existe
                    string createTableQuery = @"IF NOT EXISTS ( SELECT * FROM
sys.tables WHERE name = 'Lotes') BEGIN CREATE TABLE Lotes (Id INT
IDENTITY(1,1) PRIMARY KEY,
                        Nombre NVARCHAR(100) UNIQUE ); END";
                    using (SqlCommand cmd = new SqlCommand(createTableQuery,
conn))
                    {
                        cmd.ExecuteNonQuery();
```

```
            }

            // Insertar nuevo lote si no existe
            string insertQuery = "IF NOT EXISTS (SELECT 1 FROM Lotes
WHERE Nombre = @Nombre) " +
                                 "INSERT INTO Lotes (Nombre) VALUES (@Nombre)";
            using (SqlCommand cmd = new SqlCommand(insertQuery, conn))
            {
                cmd.Parameters.AddWithValue("@Nombre", nuevoLote);
                cmd.ExecuteNonQuery();
            }
        }

        // Reflejar en el ComboBox si no estaba
        if (!CbxLote.Items.Contains(nuevoLote))
        {
            CbxLote.Items.Add(nuevoLote);
        }

        // Seleccionarlo como lote actual
        CbxLote.SelectedItem = nuevoLote;
        //  ◇ Muy importante para que GuardarEnBaseDeDatos use este valor
        loteActual = nuevoLote;
```

```
            // Limpiar y ocultar campos de creación
            TxtLote.Clear();
            TxtLote.Visible = false;
            BtnLote.Visible = false;

            MessageBox.Show("Lote guardado correctamente.");
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error guardando lote: {ex.Message}");
        }
    }
    else
    {
        MessageBox.Show("Por favor, ingrese un nombre de lote.");
    }
}
```

Este método permite agregar un nuevo lote al ComboBox si no existe ya.

CbxLote

```
    private void CbxLote_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (CbxLote.Text == "Nuevo Lote")
        {
            TxtLote.Visible = true;
            BtnLote.Visible = true;
            BtnLote.Text = "Guardar Lote";
        }
        else
        {
            TxtLote.Visible = false;
            BtnLote.Visible = false;
        }
    }
  }
}
```

Muestra los controles para agregar un nuevo lote si el usuario selecciona la opción "Nuevo Lote".

Anexo 5 Reporte

En el formulario denominado Reporte.cs podemos observar los datos que se han pasado con anterioridad por el Dashboard generándonos reportes sobre los lotes guardados y las fechas seleccionadas, estos datos podemos guardarlos en varios tipos de datos, pueden ser PDF, JSON,

TXT el formato PDF nos permite vislumbrar al final del reporte una imagen de la gráfica de los datos guardados.

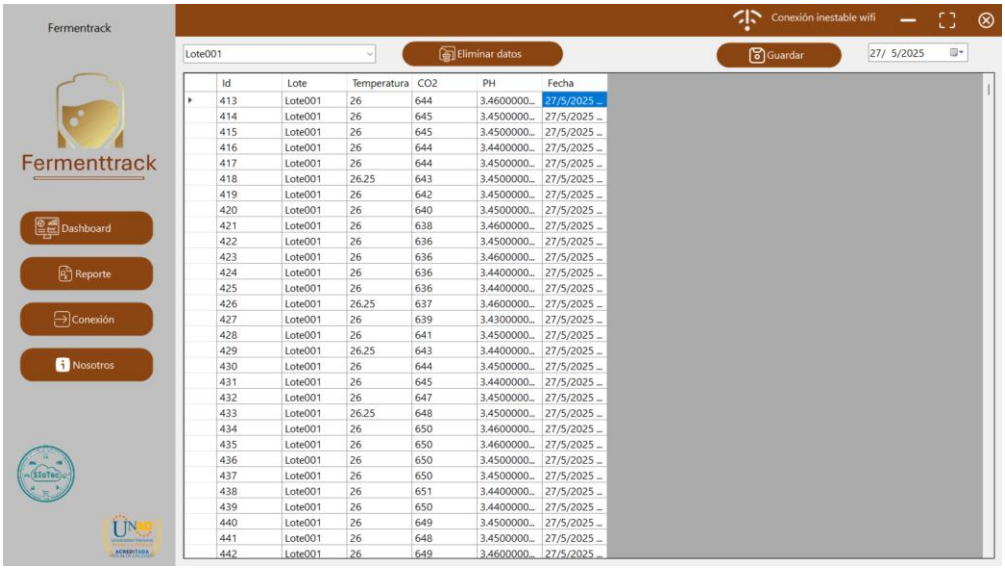


Figura 21. Reporte, Autoría Propia

En este formulario podemos buscar los datos por los tipos de lotes por los cuales estamos guardando o hemos guardado los mismo en el Dashboard también podemos filtrar los datos por fechas siendo estos métodos efectivos a la hora de hacer una búsqueda rápida.

En el formulario se presentan los siguientes elementos:

6. Caja de texto:

a. CbxLotes

7. Botones:

a. BtnLimpiar

b. BtnGuardar

8. Selector de tiempo de datos:

a. DtpFecha

9. Paneles:

a. panel1

b. panel2

c. panel3

d. panel4

10. Vista de cuadricula de datos:

a. DgvReporte

Tabla 12.

Explicación código Reporte.cs

Fuente: Autoría Propia

Sección de importaciones (using)
using iTextSharp.text; *using iTextSharp.text.pdf;* *using System;* *using System.Data;* *using System.Data.SqlClient;* *using System.Drawing;* *using System.IO;* *using System.Linq;* *using System.Windows.Forms;* *using System.Windows.Forms.DataVisualization.Charting;* *using PdfImage = iTextSharp.text.Image;*
using iTextSharp.text;

importa el espacio de nombres principales de *iTextSharp,* este contiene las clases para trabajar con documentos PDF *(*como *Document, Paragraph, Font, etc.)*

using iTextSharp.text.pdf;

Importa las clases necesarias para crear archivos PDF más complejos, tal como son las tablas *(PdfPTable),* celdas *(PdfPCell),* escritores *(PdfWriter)* y eventos *(PdfPageEventHelper).*

using System;

Importa las funciones básicas de .NET, tales como los tipos de datos *(DateTime, Exception, etc.)* así como sus clases comunes

Using System.Data;

Permite trabajar con estructuras de datos desconocidas, estos son algunos:

- *DataTable*
- *DataRow*
- *DataColumn*
- *Ect.*

using System.Data.SqlClient;

Importa clases para conectarse y ejecutar consultas en una base de datos SQL Server *(SqlConnection, SqlCommand, SqlDataAdapter...)*

using System.Drawing;

Importa clases para manejar gráficos, colores, fuentes, imágenes *(Bitmap, Color, Font, etc.).*

using System.IO;

Permite leer y escribir archivos desde el sistema de archivos *(FileStream, StreamWriter, File, etc.).*

using System.Linq;

Importa extensiones *LINQ* para hacer consultas sobre colecciones *(Select, Where, Average, etc.)*

using System.Windows.Forms;

Importa los controles para construir interfaces gráficas de *Windows Forms (como Form, Button, MessageBox, etc.).*

using System.Windows.Forms.DataVisualization.Charting;

Permite generar gráficas *(barras, líneas, etc.)* en Windows Forms usando objetos como *Chart, Series, ChartArea, etc.*

using PdfImage = iTextSharp.text.Image;

Crea un alias *PdfImage* para la clase *iTextSharp.text.Image* (evita confusión con *System.Drawing.Image*).

Estructura base del formulario Reporte

```
namespace Interfaz_grafica
{
   public partial class Reporte : Form
   {
   ...
   }
}
```

namespace Interfaz_grafica

Define el espacio de nombres del proyecto. Agrupa las clases para evitar conflictos de nombres.

public partial class Reporte : Form

Declara la clase *Reporte*, que hereda de *Form* (es un formulario de Windows Forms). El modificador *parcial* indica que esta clase puede estar dividida en varios archivos

Variables Claves

```
private DataTable datosLotes = new DataTable();
private readonly string dbConnection = "Data Source=.;Initial
Catalog=bd;Integrated Security=True";
```

private DataTable datosLotes = new DataTable();

Crea una tabla en memoria para almacenar los datos que se mostrarán o manipularán

private readonly string dbConnection = "Data Source=.;Initial Catalog=bd;Integrated Security=True";

Cadena de conexión a una base de datos SQL Server local *(. = localhost)*, con autenticación de Windows *(Integrated Security=True)* y usando la base de datos llamada *bd*.

Constructor

```
public Reporte()
{
    InitializeComponent();
    CargarLotes();
    CargarDatosDesdeBD();
}
```

public Reporte()

Constructor de la clase Reporte.cs se ejecuta automáticamente cuando se crea una instancia del formulario

{ InitializeComponent();

Incializa los componentes del formulario (botones, etiqueta, tablas, ect.), este metodo es generado por el diseño visual de Windows forms.

CargarLotes();

Llama al método *CargarLotes()* para obtener los lotes disponibles en la base de datos y llenar el *ComboBox (Selector).*

CargarDatosDesdeBD(); }

Llama al método *CargarDatosDesdeBD()* para cargar todos los datos desde la tabla *Datos* de la base de datos y mostrarlos en el *DataGridView.*

Método CargarDatosDesdeBD()

```
private void CargarDatosDesdeBD()
{
    datosLotes.Clear();

    using (SqlConnection conn = new SqlConnection(dbConnection))
    {
        conn.Open();
        SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Datos", conn);
        da.Fill(datosLotes);
    }

    DgvReporte.DataSource = datosLotes;
}
```

private void CargarDatosDesdeBD()

Define un método privado (no accesible desde fuera de la clase) que carga todos los registros desde la base de datos hacia el formulario.

Este método no recibe parámetros ni devuelve valores.

{ datosLotes.Clear();

Limpia la tabla *datosLotes*, eliminando cualquier dato anterior que pudiera haberse cargado previamente.

using (SqlConnection conn = new SqlConnection(dbConnection))

Crea un objeto *SqlConnection* con la cadena de conexión *dbConnection*.

El bloque *using* asegura que la conexión se cierre y libere automáticamente cuando termine el bloque.

{ conn.Open();

Abre la conexión a la base de datos SQL Server.

*SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Datos", conn);*

Crea un adaptador de datos *(SqlDataAdapter)* que ejecuta una consulta SQL *(SELECT * FROM Datos)* para seleccionar todas las filas y columnas de la tabla *Datos.*

da.Fill(datosLotes); }

Llena la tabla en memoria *datosLotes* con los datos obtenidos desde la base de datos

DgvReporte.DataSource = datosLotes; }

Asigna el *DataTable (datosLotes)* como fuente de datos del *DataGridView* llamado *DgvReporte*, para mostrar los datos en la interfaz gráfica.

Método BtnGuardar_Click

```
private void BtnGuardar_Click(object sender, EventArgs e)
{
    DataTable datosFiltrados = ((DataTable)DgvReporte.DataSource);

    if (datosFiltrados == null || datosFiltrados.Rows.Count == 0)
    {
        MessageBox.Show("No hay datos para guardar.");
        return;
    }

    SaveFileDialog sfd = new SaveFileDialog
    {
        Filter = "CSV (*.csv)|*.csv|TXT (*.txt)|*.txt|PDF (*.pdf)|*.pdf",
        FileName = "reporte"
    };

    if (sfd.ShowDialog() != DialogResult.OK) return;

    string path = sfd.FileName;
    try
    {
        if (path.EndsWith(".csv")) GuardarCSV(path, datosFiltrados);
```

```
            else if (path.EndsWith(".txt")) GuardarTXT(path, datosFiltrados);
            else if (path.EndsWith(".pdf")) GuardarPDF(path, datosFiltrados);
            else MessageBox.Show("Formato no soportado.");

            MessageBox.Show("Archivo guardado correctamente en:\n" + path);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error al guardar archivo: " + ex.Message);
        }
    }
```

private void BtnGuardar_Click(object sender, EventArgs e)

Este método es un manejador de evento que se ejecuta cuando el usuario hace clic en el botón “Guardar”.

{ DataTable datosFiltrados = ((DataTable)DgvReporte.DataSource);

Obtiene los datos actualmente visibles en el *DataGridView (DgvReporte)* y los convierte a un objeto *DataTable*, que se almacenará en la variable *datosFiltrados*.

if (datosFiltrados == null || datosFiltrados.Rows.Count == 0)

Verifica si la variable *datosFiltrados* está vacía *(null)* o si no tiene *filas (Rows.Count == 0).*

{MessageBox.Show("No hay datos para guardar.");

Si no hay datos, muestra un mensaje de advertencia al usuario.

return;}

Finaliza la ejecución del método para no continuar con el proceso de guardado.

SaveFileDialog sfd = new SaveFileDialog

Crea un cuadro de diálogo para guardar un archivo. Este cuadro le permite al usuario elegir el nombre, la ubicación y el formato del archivo.

{Filter = "CSV (.csv)|*.csv|TXT (*.txt)|*.txt|PDF (*.pdf)|*.pdf",*

Establece los tipos de archivo disponibles para guardar: *.csv, .txt o .pdf.*

FileName = "reporte"};

Sugiere un nombre predeterminado para el archivo: *"reporte".*

if (sfd.ShowDialog() != DialogResult.OK) return;

Muestra el cuadro de diálogo al usuario y espera su respuesta.

Si el usuario *cancela*, el método termina sin hacer nada.

string path = sfd.FileName;

Guarda la ruta y el nombre del archivo que el usuario eligió.

try{ if (path.EndsWith(".csv")) GuardarCSV(path, datosFiltrados);

else if (path.EndsWith(".txt")) GuardarTXT(path, datosFiltrados);

else if (path.EndsWith(".pdf")) GuardarPDF(path, datosFiltrados);

Si la ruta termina en *.csv,* llama al método *GuardarCSV* y le pasa la ruta y los datos.

Si el archivo es *.txt*, llama a *GuardarTXT*.

Si es *.pdf,* llama a *GuardarPDF*.

else MessageBox.Show("Formato no soportado.");

Si el formato no es ninguno de los anteriores, muestra un mensaje de error.

MessageBox.Show("Archivo guardado correctamente en:\n" + path);

Si todo va bien, muestra un mensaje de éxito con la ubicación del archivo guardado.

} catch (Exception ex)

Captura cualquier excepción *(error)* que ocurra durante el proceso de guardado.

{ MessageBox.Show("Error al guardar archivo: " + ex.Message); } }

Muestra el mensaje de error detallado al usuario si algo salió mal.

Método GuardarCSV()

```
private void GuardarCSV(string path, DataTable datos)
{
    using (StreamWriter sw = new StreamWriter(path))
    {
        foreach (DataColumn col in datos.Columns)
            sw.Write(col.ColumnName + ";");
        sw.WriteLine();

        foreach (DataRow row in datos.Rows)
            sw.WriteLine(string.Join(";", row.ItemArray));
    }
}
```

private void GuardarCSV(string path, DataTable datos)

Este método guarda los datos de una tabla (*DataTable*) en un archivo CSV.

- *path*: ruta del archivo donde se guardará.
- *datos*: datos que se escribirán en el archivo.

{using (StreamWriter sw = new StreamWriter(path))

Se crea un *StreamWriter* para escribir texto en el archivo ubicado en *path*. *using* asegura que el recurso se libere correctamente cuando termine de usarse. *{foreach (DataColumn col in datos.Columns)* Inicia un ciclo que recorre todas las columnas del *DataTable*. *sw.Write(col.ColumnName + ";");* Escribe en el archivo el nombre de cada columna, seguido de un *;* (separador de tipo CSV), esto forma la fila del encabezado del archivo CSV *sw.WriteLine();* Agrega un salto de línea final de la final de encabezado para pasar a la siguiente línea, es donde estarán ubicados los datos. *foreach (DataRow row in datos.Rows)* Inicia un ciclo que recorre todas las filas del *DataTable*. *sw.WriteLine(string.Join(";", row.ItemArray));}}* Convierte todos los valores de la fila *(row.ItemArray)* en una sola cadena separada por *;* y la escribe como una línea en el archivo.
Método GuardarTXT()

```
    private void GuardarTXT(string path, DataTable datos)
    {
        using (StreamWriter sw = new StreamWriter(path))
        {
            foreach (DataRow row in datos.Rows)
            {
                sw.WriteLine($"Lote: {row["Lote"]}, Temp: {row["Temperatura"]}, CO2:
{row["CO2"]}, pH: {row["pH"]}, Fecha: {row["Fecha"]}");
            }
        }
    }
```

private void GuardarTXT(string path, DataTable datos)

Este método guarda los datos de la tabla en un archivo de texto plano *(formato .txt)*.

path: ruta del archivo donde se guardará.

datos: parámetro recibido, aunque en esta implementación no se utiliza (se usa *datosLotes* en su lugar).

{using (StreamWriter sw = new StreamWriter(path))

Se crea un *StreamWriter* para escribir en el archivo de texto en la ruta *path*.

El *using* asegura el cierre y liberación del archivo al finalizar.

{foreach (DataRow row in datosLotes.Rows)

Se recorre cada fila de la tabla *datosLotes.*

{sw.WriteLine($"Lote: {row["Lote"]}, Temp: {row["Temperatura"]}, CO2:

{row["CO2"]}, pH: {row["pH"]}, Fecha: {row["Fecha"]}");}}}

Escribe una línea de texto con los datos de cada fila en el orden establecido en este código.

Clase PageEventHelper()

```
public class PageEventHelper : PdfPageEventHelper
{
   public override void OnEndPage(PdfWriter writer, Document document)
   {
      PdfPTable table = new PdfPTable(1);
      table.TotalWidth = document.PageSize.Width;
      table.DefaultCell.Border = iTextSharp.text.Rectangle.NO_BORDER;



      PdfPCell cell = new PdfPCell(new Phrase(writer.PageNumber.ToString()));
      cell.Border = iTextSharp.text.Rectangle.NO_BORDER;
      cell.HorizontalAlignment = Element.ALIGN_RIGHT;
      table.AddCell(cell);


      table.WriteSelectedRows(0, -1, 0, document.PageSize.Height - 20,
writer.DirectContent);
   }
}
```

public class PageEventHelper : PdfPageEventHelper

Se declara una clase publica llamada *PageEventHelper que hereda de PdfPageEventHelper,* esto permite sobrescribir métodos que se ejecutan en ciertos momentos del ciclo de vida del *PDF* (Al final de la página)

{public override void OnEndPage(PdfWriter writer, Document document)

Sobrescribe el método *OnEndPage,* este ejecuta automáticamente al terminar una página del PDF:

- write: el escritor del documento
- document: el documento PDF actual

{PdfPTable table = new PdfPTable(1);

Crea una tabla con una sola columna, esto se usará para colocar el número de página.

table.TotalWidth = document.PageSize.Width;

Establece que el ancho total de la tabla será igual al ancho de la página del documento, ayudando a que la tabla ocupe el ancho completo y se alinee correctamente.

table.DefaultCell.Border = iTextSharp.text.Rectangle.NO_BORDER;

Quitar al borde predeterminado de las celdas de la tabla para que no se dibujen líneas alrededor.

PdfPCell cell = new PdfPCell(new Phrase(writer.PageNumber.ToString()));

Crea una celda con el texto del número de página actual (*write.PageNumber*).

cell.Border = iTextSharp.text.Rectangle.NO_BORDER;

Quita los bordes de la celda especifica.

cell.HorizontalAlignment = Element.ALIGN_RIGHT;

Alinea el texto del número de página a la derecha dentro de la celda.

table.AddCell(cell);

Agrega las celdas a la tabla

table.WriteSelectedRows(0, -1, 0, document.PageSize.Height - 20,

writer.DirectContent);}}

Dibuja la tabla en una posición específica del documento:

- Desde la primera fila (índice 0) hasta la última (-1).
- Coordenadas:

- *x = 0* (borde izquierdo de la página).
- *y = document.PageSize.Height - 20* (un poco debajo del borde superior).

Método para generar gráfica lineal

```
private Bitmap GenerarGraficaLineal(string campo, DataTable datos)
{
    var chart = new Chart
    {
        Size = new Size(700, 500),
        BackColor = Color.White
    };
    chart.ChartAreas.Add(new ChartArea());

    var lotes = datos.AsEnumerable()
                .Select(r => r["Lote"].ToString())
                .Distinct();

    Random rand = new Random();
    foreach (var lote in lotes)
    {
        var filasPorLote = datos.AsEnumerable()
                           .Where(r => r["Lote"].ToString() == lote)
                           .OrderBy(r => Convert.ToDateTime(r["Fecha"]));

        Series serie = new Series(lote)
        {
            ChartType = SeriesChartType.Line,
```

```
            BorderWidth = 2,
            Color = Color.FromArgb(rand.Next(256), rand.Next(256), rand.Next(256))
        };

        foreach (var fila in filasPorLote)
        {
            DateTime fecha = Convert.ToDateTime(fila["Fecha"]);
            double valor = Convert.ToDouble(fila[campo]);
            serie.Points.AddXY(fecha, valor);
        }

        chart.Series.Add(serie);
    }

    Bitmap bmp = new Bitmap(chart.Width, chart.Height);
    chart.DrawToBitmap(bmp, new System.Drawing.Rectangle(0, 0, bmp.Width,
bmp.Height));
    return bmp;
}
```

private Bitmap GenerarGraficaLineal(string campo, DataTable datos)

Este método es privado y devuelve un objeto *Bitmap*.

Recibe:

- *campo*: el nombre de la columna *("Temperatura", "CO2" o "pH").*

- *datos*: la tabla de datos de donde se extrae la información para graficar.

{var chart = new Chart

{Size = new Size(700, 500),

BackColor = Color.White};

Crea un nuevo objeto de tipo *Chart(Grafico)* (inc., 2024) con el tamaño especifico 700, 500 px y con fondo blanco *Color.White*

chart.ChartAreas.Add(new ChartArea());

Agrega un área de gráficos al objeto *Chart* (Microsoft, Microsoft Ignite, 2025), lo cual es necesario para poder renderizar datos.

var lotes = datos.AsEnumerable()

.Select(r => r["Lote"].ToString())

.Distinct();

Extrae todos los valores únicos de la columna "*Lote*" del *DataTable*, *AsEnumerable*() permite usar LINQ sobre *DataTable*.

Random rand = new Random();

Crea una instancia de *Random* para generar colores aleatorios más adelante, siendo uno por cada lote

foreach (var lote in lotes)

Itera por cada lote único encontrado

{var filasPorLote = datos.AsEnumerable()

.Where(r => r["Lote"].ToString() == lote)

.OrderBy(r => Convert.ToDateTime(r["Fecha"]));

Filtra las filas de *DataTable* que pertenecen al lote actual y las ordena por fecha ascendente

Series serie = new Series(lote)

{ChartType = SeriesChartType.Line,

BorderWidth = 2,

Color = Color.FromArgb(rand.Next(256), rand.Next(256), rand.Next(256))};

Crea una nueva serie de datos con:

- Tipo de grafico: Linea (*line*)
- Grosor de la línea: 2 px
- Color aleatorio para distinguir visualmente cada lote

foreach (var fila in filasPorLote)

Itera sobre las filas filtradas del lote actual.

{DateTime fecha = Convert.ToDateTime(fila["Fecha"]);

double valor = Convert.ToDouble(fila[campo]);

serie.Points.AddXY(fecha, valor);}

Convierte la fecha y valores del campo seleccionado a sus tipos adecuados luego se agregan como puntos (*x= Fecha, y=Valor*) en la serie.

chart.Series.Add(serie);}

Agrega la serie construida al gráfico

Bitmap bmp = new Bitmap(chart.Width, chart.Height);

chart.DrawToBitmap(bmp, new System.Drawing.Rectangle(0, 0, bmp.Width, bmp.Height));

Crea un objeto *Bitmap* con el mismo tamaño del gráfico y este dibuja el grafico completo dentro del *Bitmap* *return bmp;}* Devuelve el gráfico como una imagen *Bitmap.*
Método para generar gráfica de barras

```
    private Bitmap GenerarGraficaBarras(DataTable datos)
    {
        var chart = new Chart
        {
            Size = new Size(700, 500),
            BackColor = Color.White
        };
        chart.ChartAreas.Add(new ChartArea());

        var mediaTemp = datosLotes.AsEnumerable().Average(r =>
Convert.ToDouble(r["Temperatura"]));
        var mediaCO2 = datosLotes.AsEnumerable().Average(r =>
Convert.ToDouble(r["CO2"]));
        var mediapH = datosLotes.AsEnumerable().Average(r =>
Convert.ToDouble(r["pH"]));

        Series serie = new Series("Media")
        {
            ChartType = SeriesChartType.Bar,
            Color = Color.DodgerBlue
        };
        serie.Points.AddXY("Temp", mediaTemp);
        serie.Points.AddXY("CO2", mediaCO2);
```

```
        serie.Points.AddXY("pH", mediapH);
        chart.Series.Add(serie);

        Bitmap bmp = new Bitmap(chart.Width, chart.Height);
        chart.DrawToBitmap(bmp, new System.Drawing.Rectangle(0, 0, bmp.Width,
bmp.Height));
        return bmp;
    }
```

private Bitmap GenerarGraficaBarras(DataTable datos)

Este método es privado y devuelve una imagen *Bitmap*.

Recibe un *DataTable* como parámetro (aunque no lo utiliza directamente, sino que usa *datosLotes*, probablemente por decisión de diseño).

{var chart = new Chart

{Size = new Size(700, 500),

BackColor = Color.White};

Crea un nuevo objeto *Chart* con un tamaño de 700x500 píxeles y fondo blanco, que contendrá la gráfica.

chart.ChartAreas.Add(new ChartArea());

Agrega una zona de dibujo al gráfico. Sin esto, no se puede graficar.

var mediaTemp = datosLotes.AsEnumerable().Average(r =>

Convert.ToDouble(r["Temperatura"]));

var mediaCO2 = datosLotes.AsEnumerable().Average(r =>

Convert.ToDouble(r["CO2"]));

var mediapH = datosLotes.AsEnumerable().Average(r =>

Convert.ToDouble(r["pH"]));

Calcula los promedios de las columnas *Temperatura*, *CO2* y *pH* a partir del *DataTable datosLotes*, usando LINQ y *Convert.ToDouble* para hacer los cálculos

Series serie = new Series("Media")

{ChartType = SeriesChartType.Bar,

Color = Color.DodgerBlue};

Crea una nueva serie de barras llamada “Media” y le asigna el color Azul (*DodgerBlue*)

serie.Points.AddXY("Temp", mediaTemp);

serie.Points.AddXY("CO2", mediaCO2);

serie.Points.AddXY("pH", mediapH);

Agrega tres puntos al gráfico: uno para cada promedio calculado (*Temperatura, CO2 y pH*), usando los nombres como etiquetas del eje X.

chart.Series.Add(serie);

Agrega la serie al gráfico.

Bitmap bmp = new Bitmap(chart.Width, chart.Height);

chart.DrawToBitmap(bmp, new System.Drawing.Rectangle(0, 0, bmp.Width,

bmp.Height));

Crea una imagen en blanco con el mismo tamaño del gráfico, y dibuja el contenido del gráfico dentro de esa imagen.

return bmp;}

Devuelve la imagen (*Bitmap*) con la gráfica de barras ya renderizada.

Clase para encabezado y pie de página

```
    public class EncabezadoPiePagina : PdfPageEventHelper
    {
        PdfContentByte cb;
        BaseFont bf = null;
        PdfTemplate template;

        public override void OnOpenDocument(PdfWriter writer, Document document)
        {
            try
            {
                bf = BaseFont.CreateFont(BaseFont.HELVETICA, BaseFont.CP1252,
BaseFont.NOT_EMBEDDED);
                cb = writer.DirectContent;
                template = cb.CreateTemplate(50, 50);
            }
            catch (DocumentException) { }
            catch (IOException) { }
        }

        public override void OnEndPage(PdfWriter writer, Document document)
        {
            if (writer.PageNumber == 1)
                return; // No encabezado ni número en la primera página
```

```
            float leftMargin = document.LeftMargin;
            float rightMargin = document.RightMargin;
            float topMargin = document.Top + 10;

            // Encabezado izquierdo
            ColumnText.ShowTextAligned(writer.DirectContent, Element.ALIGN_LEFT,
                new Phrase("Proyecto de grado: Datos tomados de una Interfaz Gráfica
para la Visualización Remota en Fermentadores de Café mediante IoT",
                new iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.HELVETICA,
8)),
                document.LeftMargin, document.PageSize.Height - topMargin, 0);

            // Número de página derecho
            ColumnText.ShowTextAligned(writer.DirectContent, Element.ALIGN_RIGHT,
                new Phrase("Página " + writer.PageNumber, new
iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.HELVETICA, 8)),
                document.PageSize.Width - document.RightMargin,
document.PageSize.Height - topMargin, 0);
        }
    }
```

```
public class EncabezadoPiePagina : PdfPageEventHelper
```

Esta clase hereda de *PdfPageEventHelper*, lo que le permite interceptar eventos durante la generación del PDF, como abrir el documento, terminar una página, etc

{PdfContentByte cb;

BaseFont bf = null;

PdfTemplate template;

- Cb: Canal para dibujar directamente en el PDF.
- Bf: Fuente base usada para escribir texto.
- Templete: Plantilla grafica que puede realizarse (aunque aquí no se usa luego del *CreateTemplate*)

public override void OnOpenDocument(PdfWriter writer, Document document)

Método *OnOpenDocument,* este método se ejecuta cuando se abre el documento PDF.

{try

{bf = BaseFont.CreateFont(BaseFont.HELVETICA, BaseFont.CP1252,

BaseFont.NOT_EMBEDDED);

Crea una Fuente *Helvetica*, codifica en *CP1252* (Latino Basico), esto no se encrustara en el archivo.

cb = writer.DirectContent;

Obtiene el canal de contenido directo, que permite dibujar manualmente en el documento.

template = cb.CreateTemplate(50, 50);}

Crea un espacio de dibujo de 50x50 unidades.

catch (DocumentException) { }

catch (IOException) { }}

Captura errores de creación del documento o fuente.

public override void OnEndPage(PdfWriter writer, Document document)

Método *OnEndPage, este método se ejecutará al final de cada página, es ideal para agregar encabezados o pies de página.*

{if (writer.PageNumber == 1)

return; // No encabezado ni número en la primera página

Evita que se agregue encabezado o número de página en la primera página del documento.

float leftMargin = document.LeftMargin;

float rightMargin = document.RightMargin;

float topMargin = document.Top + 10;

Obtiene márgenes del documento. Se suma 10 al margen superior para posicionar el texto un poco más abajo del borde.

// Encabezado izquierdo

ColumnText.ShowTextAligned(writer.DirectContent, Element.ALIGN_LEFT,

new Phrase("Proyecto de grado: Datos tomados de una Interfaz Gráfica para la Visualización Remota en Fermentadores de Café mediante IoT",

new iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.HELVETICA, 8)),

document.LeftMargin, document.PageSize.Height - topMargin, 0);

Dibuja un encabezado alineado a la izquierda, con tamaño de fuente 8, en la parte superior de la página *(ajustado por topMargin).*

// Número de página derecho

ColumnText.ShowTextAligned(writer.DirectContent, Element.ALIGN_RIGHT,

new Phrase("Página " + writer.PageNumber, new iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.HELVETICA, 8)), document.PageSize.Width - document.RightMargin, document.PageSize.Height - topMargin, 0);}} Dibuja el número de página actual, alineado a la derecha, también en la parte superior.
Método GuardadoPDF

```
private void GuardarPDF(string path, DataTable datos)
{
    string imagePathLineTemp = Path.ChangeExtension(path, "_line_temp.png");
    string imagePathLineCO2 = Path.ChangeExtension(path, "_line_co2.png");
    string imagePathLinePH = Path.ChangeExtension(path, "_line_ph.png");
    string imagePathBar = Path.ChangeExtension(path, "_bar.png");

    Bitmap graficoLinealTemp = GenerarGraficaLineal("Temperatura", datos);
    Bitmap graficoLinealCO2 = GenerarGraficaLineal("CO2", datos);
    Bitmap graficoLinealPH = GenerarGraficaLineal("pH", datos);
    Bitmap graficoBarras = GenerarGraficaBarras(datos);


    try
    {
        graficoLinealTemp.Save(imagePathLineTemp);
        graficoLinealCO2.Save(imagePathLineCO2);
        graficoLinealPH.Save(imagePathLinePH);
        graficoBarras.Save(imagePathBar);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error al guardar imágenes: " + ex.Message);
```

```
            return;
        }

        Document doc = new Document(PageSize.A4);
        PdfWriter writer = PdfWriter.GetInstance(doc, new FileStream(path,
FileMode.Create));
        writer.PageEvent = new EncabezadoPiePagina(); // clase personalizada para
encabezado y pie
        doc.Open();

        // Página 1 - Título grande centrado
        doc.NewPage();
        iTextSharp.text.Font tituloFont = FontFactory.GetFont("Times-Roman", 72,
iTextSharp.text.Font.BOLD, new BaseColor(139, 69, 19));
        Paragraph titulo = new Paragraph("Fermentrack", tituloFont)
        {
            Alignment = Element.ALIGN_CENTER,
            SpacingBefore = doc.PageSize.Height / 2 - 72 // para centrar verticalmente
        };
        doc.NewPage();
        doc.Add(titulo);

        // Pie de página
```

```
        PdfPTable footerTable = new PdfPTable(1);
        footerTable.WidthPercentage = 100;
        footerTable.DefaultCell.Border = iTextSharp.text.Rectangle.NO_BORDER;

        PdfPCell cellFooter = new PdfPCell(new Phrase("Datos suministrados
por:\nJuan Esteban Mendez Diaz\nJhonatan Molina Muñoz\nFecha: "
            + DateTime.Now.ToString("dd/MM/yyyy")));
        cellFooter.Border = iTextSharp.text.Rectangle.NO_BORDER;
        cellFooter.HorizontalAlignment = Element.ALIGN_CENTER;
        footerTable.AddCell(cellFooter);

        doc.Add(footerTable);

        // Evento para agregar número de página
        writer.PageEvent = new PageEventHelper();

        doc.NewPage(); // Página 2: aquí empieza la data con encabezado


        // Tabla con datos del DataGridView
        PdfPTable table = new PdfPTable(DgvReporte.Columns.Count);
        foreach (DataGridViewColumn column in DgvReporte.Columns)
        {
```

```
    table.AddCell(new Phrase(column.HeaderText));
}

foreach (DataGridViewRow row in DgvReporte.Rows)
{
    if (!row.IsNewRow)
    {
        foreach (DataGridViewCell cell in row.Cells)
        {
            table.AddCell(cell.Value?.ToString() ?? "");
        }
    }
}

doc.Add(table);

// Gráficas
doc.NewPage();
doc.Add(new Paragraph("1. Gráfica lineal de Temperatura"));
PdfImage imgLineTemp = PdfImage.GetInstance(imagePathLineTemp);
imgLineTemp.ScaleToFit(550, 400);
imgLineTemp.Alignment = Element.ALIGN_CENTER;
doc.Add(imgLineTemp);
```

```
doc.NewPage();
doc.Add(new Paragraph("2. Gráfica lineal de CO2"));
PdfImage imgLineCO2 = PdfImage.GetInstance(imagePathLineCO2);
imgLineCO2.ScaleToFit(550, 400);
imgLineCO2.Alignment = Element.ALIGN_CENTER;
doc.Add(imgLineCO2);

doc.NewPage();
doc.Add(new Paragraph("3. Gráfica lineal de pH"));
PdfImage imgLinePH = PdfImage.GetInstance(imagePathLinePH);
imgLinePH.ScaleToFit(550, 400);
imgLinePH.Alignment = Element.ALIGN_CENTER;
doc.Add(imgLinePH);

doc.NewPage();
doc.Add(new Paragraph("4. Gráfica de barras"));
PdfImage imgBar = PdfImage.GetInstance(imagePathBar);
imgBar.ScaleToFit(550, 400);
imgBar.Alignment = Element.ALIGN_CENTER;
doc.Add(imgBar);

// Cierre del documento
```

```
        doc.Close();
        writer.Close();

        // Eliminar imágenes temporales
        File.Delete(imagePathLineTemp);
        File.Delete(imagePathLineCO2);
        File.Delete(imagePathLinePH);
        File.Delete(imagePathBar);
    }
```

private void GuardarPDF(string path, DataTable datos)

{string imagePathLineTemp = Path.ChangeExtension(path, "_line_temp.png");

string imagePathLineCO2 = Path.ChangeExtension(path, "_line_co2.png");

string imagePathLinePH = Path.ChangeExtension(path, "_line_ph.png");

string imagePathBar = Path.ChangeExtension(path, "_bar.png");

Rutas de las imágenes a guardar:

Estas líneas definen las rutas temporales donde se guardan las imágenes generadas por PDF, cambiando la extensión del archivo de salida original (*path*) por archivos PDF

Bitmap graficoLinealTemp = GenerarGraficaLineal("Temperatura", datos);

Bitmap graficoLinealCO2 = GenerarGraficaLineal("CO2", datos);

Bitmap graficoLinealPH = GenerarGraficaLineal("pH", datos);

Bitmap graficoBarras = GenerarGraficaBarras(datos);

Generación de graficas:

Se generan las gráficas usando métodos personalizados, en base al *DataTable* recibido:

- Tres gráficas lineales (*por campo*)
- Una gráfica de barras con promedios

try

{graficoLinealTemp.Save(imagePathLineTemp);

graficoLinealCO2.Save(imagePathLineCO2);

graficoLinealPH.Save(imagePathLinePH);

graficoBarras.Save(imagePathBar);}

Se guardan las imágenes como archivos PNG temporales.

catch (Exception ex)

{MessageBox.Show("Error al guardar imágenes: " + ex.Message);

return;}

Si ocurre un error al guardar, este notificara al usuario y se detiene la ejecución del método.

Document doc = new Document(PageSize.A4);

PdfWriter writer = PdfWriter.GetInstance(doc, new FileStream(path, FileMode.Create));

writer.PageEvent = new EncabezadoPiePagina(); // clase personalizada para encabezado y pie

doc.Open();

- Se crea un documento tamaño A4.
- Se configura el *PdfWriter* para escribir en el archivo de salida.
- Se asigna una clase (*EncabezadoPiePagina*) que añadirá encabezado y pie de página.

- Se abre el documento.

// Página 1 - Título grande centrado

doc.NewPage();

iTextSharp.text.Font tituloFont = FontFactory.GetFont("Times-Roman", 72,

iTextSharp.text.Font.BOLD, new BaseColor(139, 69, 19));

Paragraph titulo = new Paragraph("Fermentrack", tituloFont)

{Alignment = Element.ALIGN_CENTER,

SpacingBefore = doc.PageSize.Height / 2 - 72 // para centrar verticalmente};

doc.NewPage();

doc.Add(titulo);

Se crea una nueva página con un título grande y centrado verticalmente, luego se añade al documento

// Pie de página

PdfPTable footerTable = new PdfPTable(1);

footerTable.WidthPercentage = 100;

footerTable.DefaultCell.Border = iTextSharp.text.Rectangle.NO_BORDER;

Se crea una tabla de una sola celda (*fila*), sin bordes, que se usara como pie de página.

PdfPCell cellFooter = new PdfPCell(new Phrase("Datos suministrados por:\nJuan

Esteban Mendez Diaz\nJhonatan Molina Muñoz\nFecha: "

+ DateTime.Now.ToString("dd/MM/yyyy")));

Se crea una celda con los nombres y la fecha actual.

cellFooter.Border = iTextSharp.text.Rectangle.NO_BORDER;

cellFooter.HorizontalAlignment = Element.ALIGN_CENTER;

footerTable.AddCell(cellFooter);

doc.Add(footerTable);

La celda se centra, se elimina su borde y se añade al documento.

// Evento para agregar número de página

writer.PageEvent = new PageEventHelper();

Se cambia el evento de paginación para que a partir de aquí se añadan los números de página.

doc.NewPage(); // Página 2: aquí empieza la data con encabezado

// Tabla con datos del DataGridView

PdfPTable table = new PdfPTable(DgvReporte.Columns.Count);

Se inicia una nueva página y se prepara una tabla con tantas columnas como columnas visibles tiene el DgvReporte.

foreach (DataGridViewColumn column in DgvReporte.Columns)

{table.AddCell(new Phrase(column.HeaderText));}

Se añaden los nombres de las columnas como encabezados de la tabla.

foreach (DataGridViewRow row in DgvReporte.Rows)

{if (!row.IsNewRow)

{foreach (DataGridViewCell cell in row.Cells)

{table.AddCell(cell.Value?.ToString() ?? "");}}}

Se recorren las filas y se añaden sus celdas a la tabla, se omiten nuevas filas si están vacías.

doc.Add(table);

Se añade la tabla al documento PDF.

```
// Gráficas

doc.NewPage();

doc.Add(new Paragraph("1. Gráfica lineal de Temperatura"));

PdfImage imgLineTemp = PdfImage.GetInstance(imagePathLineTemp);

imgLineTemp.ScaleToFit(550, 400);

imgLineTemp.Alignment = Element.ALIGN_CENTER;

doc.Add(imgLineTemp);
```

Agrega graficas al documento pues cada grafica se añade en su propia página, con su respectivo título, esto se repite para CO2, pH y gráficos de barras.

```
doc.NewPage();

doc.Add(new Paragraph("2. Gráfica lineal de CO2"));

PdfImage imgLineCO2 = PdfImage.GetInstance(imagePathLineCO2);

imgLineCO2.ScaleToFit(550, 400);

imgLineCO2.Alignment = Element.ALIGN_CENTER;

doc.Add(imgLineCO2);

doc.NewPage();

doc.Add(new Paragraph("3. Gráfica lineal de pH"));

PdfImage imgLinePH = PdfImage.GetInstance(imagePathLinePH);

imgLinePH.ScaleToFit(550, 400);

imgLinePH.Alignment = Element.ALIGN_CENTER;

doc.Add(imgLinePH);

doc.NewPage();

doc.Add(new Paragraph("4. Gráfica de barras"));
```

PdfImage imgBar = PdfImage.GetInstance(imagePathBar);

imgBar.ScaleToFit(550, 400);

imgBar.Alignment = Element.ALIGN_CENTER;

doc.Add(imgBar);

Cada imagen se escala y se centra para presentar bien en PDF.

// Cierre del documento

doc.Close();

writer.Close();

Cerrar y eliminar temporales, se cierra correctamente el documento y el escritor.

// Eliminar imágenes temporales

File.Delete(imagePathLineTemp);

File.Delete(imagePathLineCO2);

File.Delete(imagePathLinePH);

File.Delete(imagePathBar);}

Se eliminan las imágenes temporales que fueron insertadas en PDF.

Método CargarLotes()

```
    private void CargarLotes()
    {
        DateTime fechaSeleccionada = DtpFecha.Value.Date;

        // Filtrar el DataGridView por fecha
        using (SqlConnection conn = new SqlConnection(dbConnection))
        {
            conn.Open();
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Datos WHERE
CAST(Fecha AS DATE) = @fecha", conn);
            da.SelectCommand.Parameters.AddWithValue("@fecha", fechaSeleccionada);

            DataTable dt = new DataTable();
            da.Fill(dt);
            DgvReporte.DataSource = dt;

            if (dt.Rows.Count == 0)
            {
                MessageBox.Show("No se encontraron datos para la fecha seleccionada.");
                DgvReporte.DataSource = null;
                return;
            }
```

```
        }

        // También actualizar el ComboBox de lotes para esa fecha
        CbxLotes.Items.Clear();
        using (SqlConnection conn = new SqlConnection(dbConnection))
        {
            conn.Open();
            string query = "SELECT DISTINCT Lote FROM Datos WHERE CAST(Fecha
AS DATE) = @fecha ORDER BY Lote";
            SqlCommand cmd = new SqlCommand(query, conn);
            cmd.Parameters.AddWithValue("@fecha", fechaSeleccionada);
            SqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                CbxLotes.Items.Add(reader["Lote"].ToString());
            }
        }

    }
```

private void CargarLotes()

Declara el método *CargarLotes*, que no recibe parámetros y se encarga de filtrar los datos por fecha y llenar el *ComboBox* con los lotes correspondientes

{DateTime fechaSeleccionada = DtpFecha.Value.Date;

Obtiene la fecha seleccionada desde el control *DateTimePicker* llamado *DtpFecha*, y la guarda sin la parte de la hora *(.Date)*.

// Filtrar el DataGridView por fecha

using (SqlConnection conn = new SqlConnection(dbConnection))

Cargar datos al *DataGridView* según la fecha seleccionada, crea una conexión temporal (*conn*) a la base de datos usando la cadena *dbConnection*.

{conn.Open();

Abre la conexión a la base de datos.

*SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Datos WHERE CAST(Fecha AS DATE) = @fecha", conn);*

- Crea un adaptador de datos que consulta todos los registros de la tabla *Datos* cuyo campo *Fecha* coincide con la fecha seleccionada.
- Se usa *CAST(Fecha AS DATE)* para eliminar la hora y comparar solo la fecha.

da.SelectCommand.Parameters.AddWithValue("@fecha", fechaSeleccionada);

DataTable dt = new DataTable();

Asigna valores reales al parámetro @fecha en la consulta

da.Fill(dt);

Crea una tabla temporal *dt* y la llena con los resultados obtenidos de la consulta.

DgvReporte.DataSource = dt;

Asigna el contenido de dt al *DataGridView* llamado *DgvReporte*.

if (dt.Rows.Count == 0)

{MessageBox.Show("No se encontraron datos para la fecha seleccionada.");

DgvReporte.DataSource = null;

return;}}

Verifica si no se encontraron registros. Si es así:

- Muestra un mensaje de advertencia.
- Limpia el *DataGridView*.
- Sale del método.

// También actualizar el ComboBox de lotes para esa fecha

CbxLotes.Items.Clear();

Limpia todos los elementos previamente añadidos al *ComboBox* llamado *CbxLotes*.

using (SqlConnection conn = new SqlConnection(dbConnection))

{conn.Open();

Crea y abre una nueva conexión con la base de datos (se reutiliza la misma variable *conn*).

string query = "SELECT DISTINCT Lote FROM Datos WHERE CAST(Fecha AS DATE) = @fecha ORDER BY Lote";

Consulta todos los lotes únicos (*DISTINCT*) que coinciden con la fecha seleccionada, ordenados alfabéticamente.

SqlCommand cmd = new SqlCommand(query, conn);

cmd.Parameters.AddWithValue("@fecha", fechaSeleccionada);

Prepara el comando SQL y le asigna el valor del parámetro *@fecha*

SqlDataReader reader = cmd.ExecuteReader();

Ejecuta la consulta y guarda el resultado en un lector (*reader*) para recorrer fila por fila.

while (reader.Read())

{CbxLotes.Items.Add(reader["Lote"].ToString());}}} Recorre cada fila del resultado y añade el valor del campo *Lote* al *ComboBox*
Método DtpFecha_ValueChanged

```
    private void DtpFecha_ValueChanged(object sender, EventArgs e)
    {
        DateTime fechaSeleccionada = DtpFecha.Value.Date;

        CbxLotes.Items.Clear();
        using (SqlConnection conn = new SqlConnection(dbConnection))
        {
            conn.Open();
            string query = "SELECT DISTINCT Lote FROM Datos WHERE CAST(Fecha
AS DATE) = @fecha ORDER BY Lote";
            SqlCommand cmd = new SqlCommand(query, conn);
            cmd.Parameters.AddWithValue("@fecha", fechaSeleccionada);
            SqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                CbxLotes.Items.Add(reader["Lote"].ToString());
            }
        }
    }
```

private void DtpFecha_ValueChanged(object sender, EventArgs e)

Este es un manejador de eventos que se activa automáticamente cada vez que el usuario cambia la fecha en el *DateTimePicker* llamado *DtpFecha*.

{DateTime fechaSeleccionada = DtpFecha.Value.Date;

Obtiene la fecha seleccionada del *DateTimePicker* y elimina la hora, dejando solo la parte de la fecha (*.Date*).

CbxLotes.Items.Clear();

Limpia todos los elementos actualmente mostrados en el *ComboBox CbxLotes*, para actualizarlo con los lotes correspondientes a la nueva fecha seleccionada.

using (SqlConnection conn = new SqlConnection(dbConnection))

Bloque *using* para consulta SQL, crea una conexión con la base de datos utilizando la cadena *dbConnection*

{conn.Open();

Abre la conexión

string query = "SELECT DISTINCT Lote FROM Datos WHERE CAST(Fecha AS DATE) = @fecha ORDER BY Lote";

Define la consulta SQL:

- Selecciona lotes únicos *(DISTINCT)* de la tabla Datos.
- Compara solo la fecha *(sin hora)* usando CAST*(Fecha AS DATE)*.
- Ordena los lotes alfabéticamente *(ORDER BY).*

SqlCommand cmd = new SqlCommand(query, conn);

Prepara el commando SQL con la consulta y la conexión abierta

cmd.Parameters.AddWithValue("@fecha", fechaSeleccionada);

Asigna el valor real del parámetro *@fecha* que se usará en la consulta.

SqlDataReader reader = cmd.ExecuteReader();

Ejecuta la consulta y obtiene un lector (*reader*) para recorrer los resultados.

while (reader.Read()) *{CbxLotes.Items.Add(reader["Lote"].ToString());}}}* Mientras haya filas en el resultado: - Extrae el valor del campo *Lote*. - Lo convierte a texto y lo añade al *ComboBox CbxLotes*.
Método BtnLimpiar_Click

```
        private void BtnLimpiar_Click(object sender, EventArgs e)
        {
            if (CbxLotes.SelectedItem == null)
            {
                MessageBox.Show("Por favor, seleccione un lote para eliminar.");
                return;
            }

            string loteSeleccionado = CbxLotes.SelectedItem.ToString();
            DialogResult resultado = MessageBox.Show($"¿Está seguro de eliminar los datos del lote {loteSeleccionado}?",
                                                     "Confirmar eliminación",
                                                     MessageBoxButtons.YesNo,
                                                     MessageBoxIcon.Warning);

            if (resultado == DialogResult.Yes)
            {
                try
                {
                    using (SqlConnection conn = new SqlConnection(dbConnection))
                    {
                        conn.Open();
                        string query = "DELETE FROM Datos WHERE Lote = @Lote";
```

```
                        SqlCommand cmd = new SqlCommand(query, conn);
                        cmd.Parameters.AddWithValue("@Lote", loteSeleccionado);
                        int filasAfectadas = cmd.ExecuteNonQuery();

                        if (filasAfectadas > 0)
                        {
                            MessageBox.Show($"Se eliminaron {filasAfectadas} registros
del lote {loteSeleccionado}.");
                            CargarDatosDesdeBD();
                            CargarLotes();
                        }
                        else
                        {
                            MessageBox.Show($"No se encontraron registros para el lote
{loteSeleccionado}.");
                        }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error al eliminar datos: " + ex.Message);
                }
            }
```

}

private void BtnLimpiar_Click(object sender, EventArgs e)

Este método se ejecuta cuando se hace clic en el boton "*Limpiar",* el cual tiene asignado el siguiente evento

{if (CbxLotes.SelectedItem == null)

Verifica si el *ComboBox CbxLotes* no tiene ningún ítem seleccionado.

{MessageBox.Show("Por favor, seleccione un lote para eliminar.");

return;}

Si no hay selección, muestra un mensaje al usuario y termina el método para evitar errores.

string loteSeleccionado = CbxLotes.SelectedItem.ToString();

Obtiene el lote seleccionado, se convierte el ítem seleccionado en el *ComboBox* a una cadena y lo guarda en *loteSeleccionado*.

DialogResult resultado = MessageBox.Show($"¿Está seguro de eliminar los datos del lote {loteSeleccionado}?",

"Confirmar eliminación",

MessageBoxButtons.YesNo,

MessageBoxIcon.Warning);

Confirma la acción con el usuario, esto muestra un cuadro de dialogo que pregunta al usuario si realmente quiere eliminar ese lote, este resultado puede ser sí o no

if (resultado == DialogResult.Yes)

Verifica si el resultado seleccionado es "si" en la confirmación

{try

Bloque que captura errores si ocurre un problema durante la eliminación

{using (SqlConnection conn = new SqlConnection(dbConnection))

Establece una conexión con la base de datos.

{conn.Open();

Abre la conexión.

string query = "DELETE FROM Datos WHERE Lote = @Lote";

Define la consulta SQL para eliminar todas las filas donde el campo *lote* coincida.

SqlCommand cmd = new SqlCommand(query, conn);

Crea un comando SQL con la consulta y la conexión activa.

cmd.Parameters.AddWithValue("@Lote", loteSeleccionado);

Agrega el valor real del parámetro *@Lote* usando el lote seleccionado.

int filasAfectadas = cmd.ExecuteNonQuery();

Ejecuta el comando y devuelve la cantidad de filas eliminadas

if (filasAfectadas > 0)

Verifica si se eliminó algo, pues comprueba si se eliminó al menos un registro.

{MessageBox.Show($"Se eliminaron {filasAfectadas} registros del lote {loteSeleccionado}.");

CargarDatosDesdeBD();

CargarLotes();}

Informa al usuario y actualiza la tabla y el combo de lotes.

else

{MessageBox.Show($"No se encontraron registros para el lote {loteSeleccionado}.");}}}

Si no se encontró nada, avisa al usuario.

catch (Exception ex) *{MessageBox.Show("Error al eliminar datos: " + ex.Message);}}}* Si ocurre algún error (por ejemplo, problema con la base de datos), lo muestra en un *MessageBox*.
Método manejador del evento de cambio de selección

```
            private void CbxLotes_SelectedIndexChanged(object sender, EventArgs e)
            {
                if (CbxLotes.SelectedItem == null) return;
                string loteSeleccionado = CbxLotes.SelectedItem.ToString();

                using (SqlConnection conn = new SqlConnection(dbConnection))
                {
                    conn.Open();
                    SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Datos
WHERE Lote = @Lote", conn);
                    da.SelectCommand.Parameters.AddWithValue("@Lote",
loteSeleccionado);

                    DataTable filtro = new DataTable();
                    da.Fill(filtro);
                    DgvReporte.DataSource = filtro;
                }
            }
        }
    }
```

private void CbxLotes_SelectedIndexChanged(object sender, EventArgs e)

Este método se ejecuta automáticamente cuando el usuario cambia la selección en el *ComboBox* llamado *CbxLotes*

{if (CbxLotes.SelectedItem == null) return;

Si no hay ningún ítem seleccionado, el método termina inmediatamente para evitar errores

string loteSeleccionado = CbxLotes.SelectedItem.ToString();

Convierte el ítem seleccionado del *ComboBox* a una cadena y lo guarda en *loteSeleccionado*

using (SqlConnection conn = new SqlConnection(dbConnection))

Crea una nueva conexión a la base de datos usando la cadena de conexión *dbConnection*.

{conn.Open();

Abre la conexión.

*SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Datos WHERE Lote = @Lote", conn);*

Crea un adaptador que ejecutará una consulta SQL para obtener todos los datos del lote seleccionado.

da.SelectCommand.Parameters.AddWithValue("@Lote", loteSeleccionado);

Agrega el valor del parámetro *@Lote* a la consulta para filtrar los registros por ese lote.

DataTable filtro = new DataTable();

da.Fill(filtro);

Crea un nuevo *DataTable* llamado *filtro* y lo llena con los datos obtenidos de la consulta.

DgvReporte.DataSource = filtro;}}

Asigna el *DataTable* como origen de datos del *DataGridView DgvReporte*, lo que actualiza su contenido visualmente.

Anexo 6 Nosotros

En el formulario Nosotros.cs se encuentra información sobre la aplicación, sus creadores y políticas de uso e intelectuales, con esta información también se encuentran los requerimientos necesarios para el equipo.

Figura 22. Nosotros.cs.

Fuente: Autoría Propia

La interfaz cuenta con los siguientes elementos:

1. Etiquetas:

a. LblTitulo_1

b. LblMensaje_2

c. LblMensaje_4

d. LblMensaje_5

e. LblMensaje_6

f. LblTitulo

g. LblTexto

h. LblTexto1

i. LblMensajeTerminos

j. LblTitulo_2

k. LblMensaje_7

l. LblMensaje_8

m. LblMensaje_9

n. LblMensaje_10

o. LblMensaje_11

p. LblMensaje_12

q. LblMensaje_13

r. LblMensaje_14

s. LblMensaje_15

t. LblMensaje_16

u. LblMensaje_17

v. LblMensaje_18

w. LblMensaje_19

x. LblMensaje_20

y. LblMensaje_21

z. LblMensaje_22

2. Cajas de imágenes:

a. pictureBox1

b. pictureBox2

c. pictureBox3

3. Paneles:

a. panel4

b. panel6

c. panel7

d. panel8

e. panel9

f. panel10

g. panel11

h. panel12

i. panel13

4. Casilla de verificación:

a. CbxAceptarTerminosyCondiciones

5. Caja combinada:

a. CbxTerminos

Tabla 13.

Explicación Código Nosotros.cs.

Fuente: Autoría Propia

Importaciones y espacios de nombre
using System; using System.Windows.Forms;
using System; permite usar clases básicas de .NET, como *String, DateTime, etc.* *using System.Windows.Forms;* permite trabajar con formularios *(Form),* controles *(Label, ComboBox, etc.).*
Clase Nosotros (Formulario)

```
namespace Interfaz_grafica.Controlador
{
    public partial class Nosotros : Form
    {
        …
    }
}
```

namespace Interfaz_grafica.Controlador

Define el espacio de nombres donde está contenida esta clase. Esto ayuda a organizar tu proyecto.

public partial class Nosotros : Form

Es la declaración de la clase parcial llamada *Nosotros* que se hereda de *Form*, es una ventana de la aplicación

Constructor

```
            public Nosotros()
            {
                InitializeComponent();
                MostrarTexto();
                CbxTerminos.Text = "\"Acuerdos\" de términos y condiciones";


                // Verifica si ya se aceptaron los términos
                if (Properties.Settings.Default.AceptoTerminos)
                {
                    CbxAceptarTerminosyCondiciones.Checked = true;
                    CbxAceptarTerminosyCondiciones.Enabled = false; // ya no se puede
desmarcar
                }
            }
```

public Nosotros()

Método que se ejecuta automáticamente al crear una instancia del formulario

InitializeComponent();

Inicializa todos los componentes visuales definidos en el diseñador (.*Designer.cs*).

MostrarTexto();

Llama al método que carga el texto informativo en los *Labels*.

CbxTerminos.Text = "\"Acuerdos\" de términos y condiciones";

Establece el texto por defecto para el *Combobox CbxTerminos*.

if (Properties.Settings.Default.AceptoTerminos)

Verifica si el usuario ya aceptó los términos previamente (se guarda en las configuraciones del programa). *CbxAceptarTerminosyCondiciones.Checked = true;* *CbxAceptarTerminosyCondiciones.Enabled = false;* Si ya los aceptó, marca el *CheckBox* y lo desactiva para evitar que se desmarque.
Método MostrarTexto()

```
            private void MostrarTexto()
            {
                LblTexto.Text = "Esta aplicación se desarrolló en base a el departamento
\n" +
                    "del Huila, Colombia, especíalizado para los pequeños productores \n"
+
                    "donde se recolectaron muestras de café para la medición de las \n" +
                    "variables temperatura, pH y dióxido de carbono (CO₂) presentes \n" +
                    "en el proceso de fermentación. El objetivo principal es mejorar \n" +
                    "el monitoreo de la fermentacion mediante el diseño de una \n" +
                    "aplicación, que permite visualizar en tiempo real los datos \n" +
                    "capturados por sensores conectados a placas como ESP32 o Arduino.
\n" +
                    "Esta solución tecnológica busca facilitar la toma de decisiones\n" +
                    "en el proceso de fermentación local, promoviendo el uso de \n" +
                    "herramientas digitales accesibles y eficientes para el control y \n" +
                    "análisis de parámetros en el proceso de fermentacion en el café.";

                LblTexto1.Text = "Se encuentran estas variedades de café en los lotes
demarcados\n" +
                    "en la toma de muestras en la imagen:\n" +
                    "1. Caturra\n" +
                    "2. Borbon Rosado\n" +
```

```
                "3. San Bernardo\n" +
                "4. Catuay\n" +
                "5. Supremo\n";
        }
```

private void MostrarTexto()

Método privado que carga texto explicito en los labels del formulario

LblTexto.Text = "Esta aplicación se desarrolló...";

Asignar un texto descriptivo a LblTexto, con información sobre el propósito de la app.

LblTexto1.Text = "Se encuentran estas variedades de café...";

Asigna un texto sobre los tipos de café utilizados en la recolección de datos

Evento CbxTerminos_SelectedIndexChanged

```
        private void CbxTerminos_SelectedIndexChanged(object sender, EventArgs
e)
        {

            if (CbxTerminos.Text == "1. Política de Privacidad")
            {
                LblMensajeTerminos.Text =
                    "La presente Política de Privacidad regula el tratamiento de los datos
personales recolectados por esta aplicación, conforme a la Ley \n" +
                    "1581 de 2012, el Decreto 1377 de 2013, y demás normas aplicables
en la República de Colombia.\r\n\r\n" +
                    "1. Responsable del Tratamiento:\r\n\r\n" +
                    "- Nombre del Responsable: Estudinates Unadistas\n" +
                    "   - Jhonatan Molina Muñoz\n" +
                    "   - Juan Esteban Mendez Díaz\r\n\r\n" +
                    "- Correo de contacto: \n" +
                    "   - yomoz2320@gmail.com\n" +
                    "   - Marcusride52@gmail.com\r\n\r\n" +
                    "2. Datos recolectados:\r\nLa aplicación podrá recolectar los
siguientes datos: nombres, apellidos, correo electrónico, número de \n" +
                    "identificación, ubicación, información de contacto, entre
otros.\r\n\r\n" +
                    "3. Finalidades del Tratamiento:\r\n\r\n" +
```

```
                    "- Gestión de usuarios registrados.\r\n\r\n" +
                    "- Mejora de la experiencia de usuario.\r\n\r\n" +
                    "- Envío de notificaciones, alertas o actualizaciones.\r\n\r\n" +
                    "- Cumplimiento de obligaciones legales.\r\n\r\n" +
                    "4. Derechos del Titular:\r\n" +
                    "Usted tiene derecho a:\r\n\r\n" +
                    "- Conocer, actualizar y rectificar sus datos.\r\n\r\n" +
                    "- Solicitar la supresión de los datos o revocar la
autorización.\r\n\r\n" +
                    "- Presentar quejas ante la Superintendencia de Industria y
Comercio.\r\n\r\n" +
                    "5. Medidas de Seguridad:\r\n" +
                    "Los datos serán tratados de forma segura, impidiendo el acceso,
modificación o eliminación no autorizada.\r\n\r\n" +
                    "6. Conservación de los datos:\r\n" +
                    "Sus datos se conservarán por el tiempo necesario para cumplir con
las finalidades mencionadas.";
                }

                if (CbxTerminos.Text == "2. Terminos y Condiciones del uso de la
aplicación")
                {
                    LblMensajeTerminos.Text =
```

```
                "1. Aceptación de los Términos\r\n\r\n" +
                "Al instalar, acceder o utilizar esta aplicación desarrollada en C#, usted acepta los presentes Términos y Condiciones. Si no está de acuerdo \n" +
                "con ellos, absténgase de utilizar la aplicación.\r\n\r\n" +
                "2. Uso Autorizado\r\n\r\n" +
                "El usuario se compromete a utilizar esta aplicación únicamente para los fines permitidos por la ley, estos Términos y Condiciones y cualquier \n" +
                "normativa aplicable. Cualquier uso indebido, reproducción no autorizada o ingeniería inversa del software está prohibido.\r\n\r\n" +
                "3. Propiedad Intelectual\r\n\r\n" +
                "Todo el contenido, código fuente, diseño, interfaz y demás componentes de esta aplicación son propiedad del desarrollador o titular de los \n" +
                "derechos. Se prohíbe su reproducción total o parcial sin autorización previa y por escrito.\r\n\r\n" +
                "4. Protección de Datos Personales\r\n\r\n" +
                "El uso de esta aplicación puede implicar el tratamiento de datos personales. En cumplimiento de la Ley 1581 de 2012 y sus decretos \n" +
                "reglamentarios, se informa al usuario que:\r\n\r\n" +
                "- Los datos personales suministrados serán tratados conforme a nuestra Política de Privacidad.\n" +
                "- El titular de los datos tiene derecho a conocer, actualizar, rectificar y suprimir sus datos, así como a revocar la autorización otorgada \n" +
                "para su tratamiento.\n" +
```

```
                "- La información recolectada será utilizada exclusivamente para los
fines funcionales de la aplicación.\r\n\r\n" +
                "5. Responsabilidad del Usuario\r\n\r\n" +
                "El usuario es responsable de mantener la confidencialidad de su
cuenta, contraseña o cualquier dato de \n" +
                "acceso utilizado dentro de la aplicación. El desarrollador no se hace
responsable por el uso indebido que terceros hagan de dicha información.\r\n\r\n" +
                "6. Limitación de Responsabilidad\r\n\r\n" +
                "La aplicación se proporciona \"tal cual\", sin garantías de ningún tipo.
El desarrollador no será responsable por daños directos o \n" +
                "indirectos, pérdida de datos, fallas en el sistema o cualquier otro
inconveniente derivado del uso de la aplicación.\r\n\r\n" +
                "7. Actualizaciones y Cambios\r\n\r\n" +
                "El desarrollador podrá actualizar o modificar estos Términos y
Condiciones en cualquier momento. Es responsabilidad del usuario revisarlos \n" +
                "periódicamente. El uso continuado de la aplicación después de dichos
cambios implica la aceptación de los mismos.\r\n\r\n" +
                "8. Legislación Aplicable y Jurisdicción\r\n\r\n" +
                "Estos Términos y Condiciones se rigen por las leyes de la República
de Colombia. En caso de controversias, el usuario y el desarrollador \n" +
                "aceptan someterse a la jurisdicción de los jueces y tribunales de
Colombia.\r\n\r\n" +
                "9. Contacto\r\n\r\n" +
```

```
                    "Para ejercer sus derechos relacionados con el tratamiento de datos o
para cualquier inquietud sobre estos Términos, puede contactarnos al \n" +
                    "correo:\n" +
                    "Fermentrack@fermentando.com";
                }


                if (CbxTerminos.Text == "3. Aviso de Privacidad")
                {
                    LblMensajeTerminos.Text =
                        "Al registrarse o usar esta aplicación, usted autoriza el tratamiento de
sus datos personales para fines relacionados \n" +
                        "con el uso y funcionamiento de la misma, conforme a nuestra
\"Política de Privacidad\". Tiene derecho a conocer, actualizar \n" +
                        "y suprimir sus datos.";
                }


                if (CbxTerminos.Text == "4. Licencia de Uso del Software")
                {
                    LblMensajeTerminos.Text =
                        "Este software es propiedad de los esti+udiantes de Ing. de sistemas
\" Jhonatan Molina M. y Juan Esteban Mendez D\"\n" +
                        "y se entrega bajo los siguientes términos:\r\n\r\n" +
```

```
                    "- El usuario recibe una licencia de uso no exclusiva, revocable e
intransferible para acceder y utilizar esta aplicación.\r\n\r\n" +
                    "- Queda prohibida la modificación, reproducción o distribución no
autorizada del código fuente o binario.\r\n\r\n" +
                    "- No se permite la ingeniería inversa, descompilación o uso del
software con fines ilícitos.\r\n\r\n" +
                    "- Esta licencia no transfiere ningún derecho de propiedad intelectual
al usuario.";
                }

                if (CbxTerminos.Text == "5. Acuerdo de Nivel de Servicio (SLA)")
                {
                    LblMensajeTerminos.Text =
                    "Este Acuerdo establece los niveles de servicio ofrecidos por la
aplicación:\r\n\r\n" +
                    "- Disponibilidad del servicio: 99% mensual.\r\n\r\n" +
                    "- Soporte técnico: Disponible en horario lunes a viernes, 9:00 a.m. a
4:00 p.m., vía correo electrónico: Fermentrack@fermentando.com\r\n\r\n" +
                    "- Tiempo de respuesta: máximo 48 horas hábiles.\r\n\r\n" +
                    "- Actualizaciones: periódicas y automáticas, sin afectar el uso
normal.";
                }
```

```
                if (CbxTerminos.Text == "6. Consentimiento Informado")
                {
                    LblMensajeTerminos.Text =
                        "Al utilizar esta aplicación, usted otorga su consentimiento
informado para el tratamiento de sus datos personales, \n" +
                        "conforme a la Ley 1581 de 2012 y la Política de Privacidad. Tiene
derecho a conocer, actualizar y suprimir sus datos.";
                }

                if (CbxTerminos.Text == "7. Política de Seguridad de la Información")
                {
                    LblMensajeTerminos.Text =
                        "La seguridad de la información es fundamental. Se implementan
medidas técnicas y organizativas para proteger los datos personales \n" +
                        "contra acceso no autorizado, pérdida o destrucción. Sin embargo, no
se garantiza la seguridad absoluta de la información transmitida.";
                }
            }
```

private void CbxTerminos_SelectedIndexChanged(object sender, EventArgs e)

Se activa cuando el usuario selecciona un nuevo valor en el *ComboBox CbxTerminos*

if (CbxTerminos.Text == "....")\{...\}

Se asigna el texto de acuerdo con la opción seleccionada este patrón es repetido siete veces

Evento CbxAceptarTerminosyCondiciones_CheckedChanged

```
            private void CbxAceptarTerminosyCondiciones_CheckedChanged(object
sender, EventArgs e)
            {
                if (CbxAceptarTerminosyCondiciones.Checked)
                {
                    // Guardar que ya aceptó los términos
                    Properties.Settings.Default.AceptoTerminos = true;
                    Properties.Settings.Default.Save();

                    MessageBox.Show("Gracias por aceptar los términos y condiciones.");
                    CbxAceptarTerminosyCondiciones.Enabled = false; // evita que se
desmarque
                }
            }
        }
    }
```

private void CbxAceptarTerminosyCondiciones_CheckedChanged(object sender, EventArgs e)

Se activa cuando el usuario marca o desmarca el *CheckBox* *CbxAceptarTerminosyCondiciones*.

if (CbxAceptarTerminosyCondiciones.Checked)

Verifica si se ha marcado.

Properties.Settings.Default.AceptoTerminos = true;

Properties.Settings.Default.Save();

Guarda el valor en la configuración del programa para recordar que ya aceptó los términos

MessageBox.Show("Gracias por aceptar los términos y condiciones.");

CbxAceptarTerminosyCondiciones.Enabled = false;

Muestra un mensaje de agradecimiento y desactiva el *CheckBox* para evitar modificaciones futuras.

Guía espacio de ejecución del programa, funcionamiento de botones y funciones agregadas.

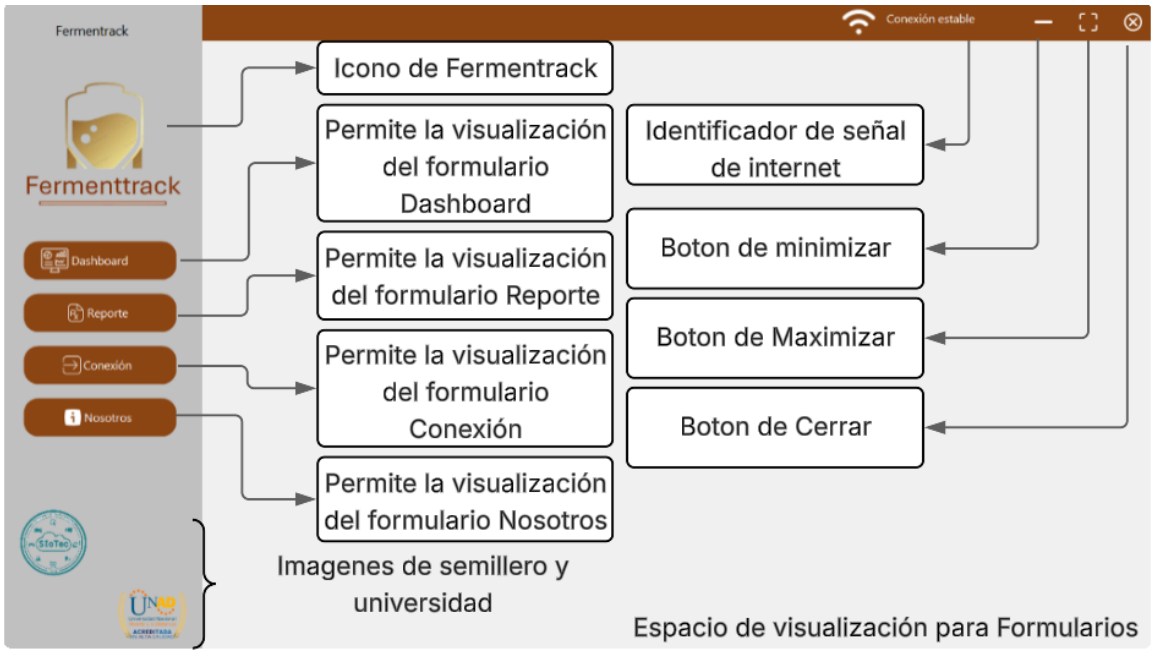


Figura 23. Guía de funcionamiento Form1.cs, Autoria Propia

Figura 24. Guía de funcionamiento Dashboard.cs, Autoría Propia

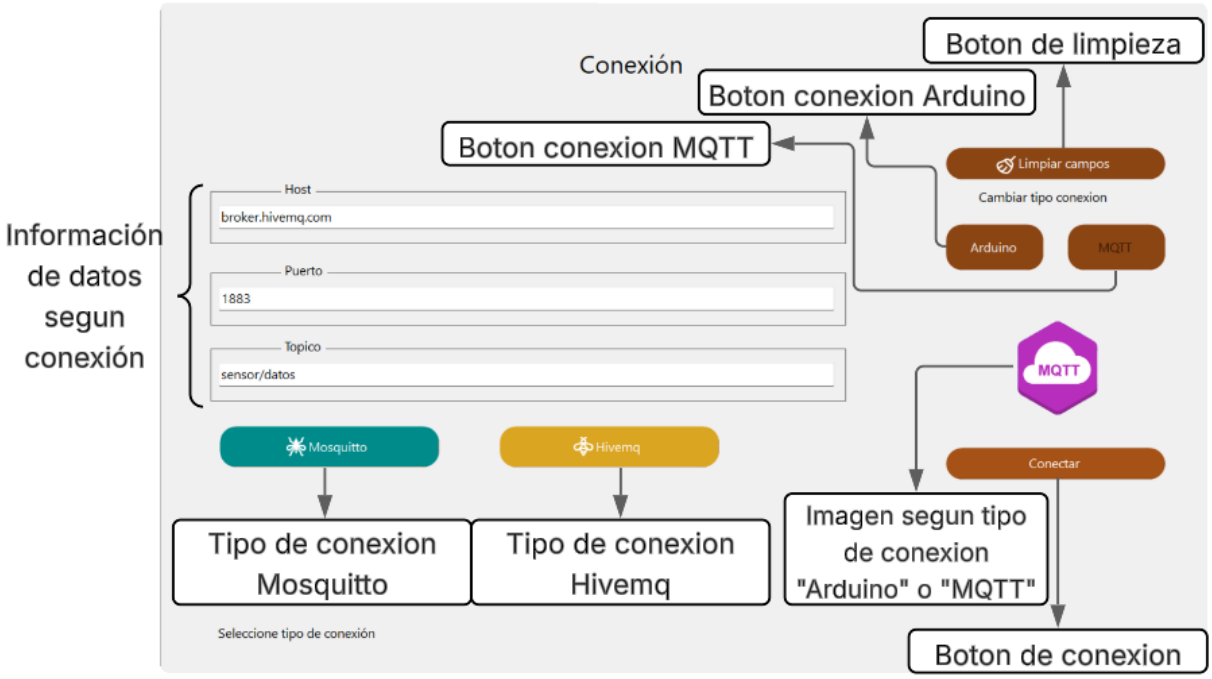
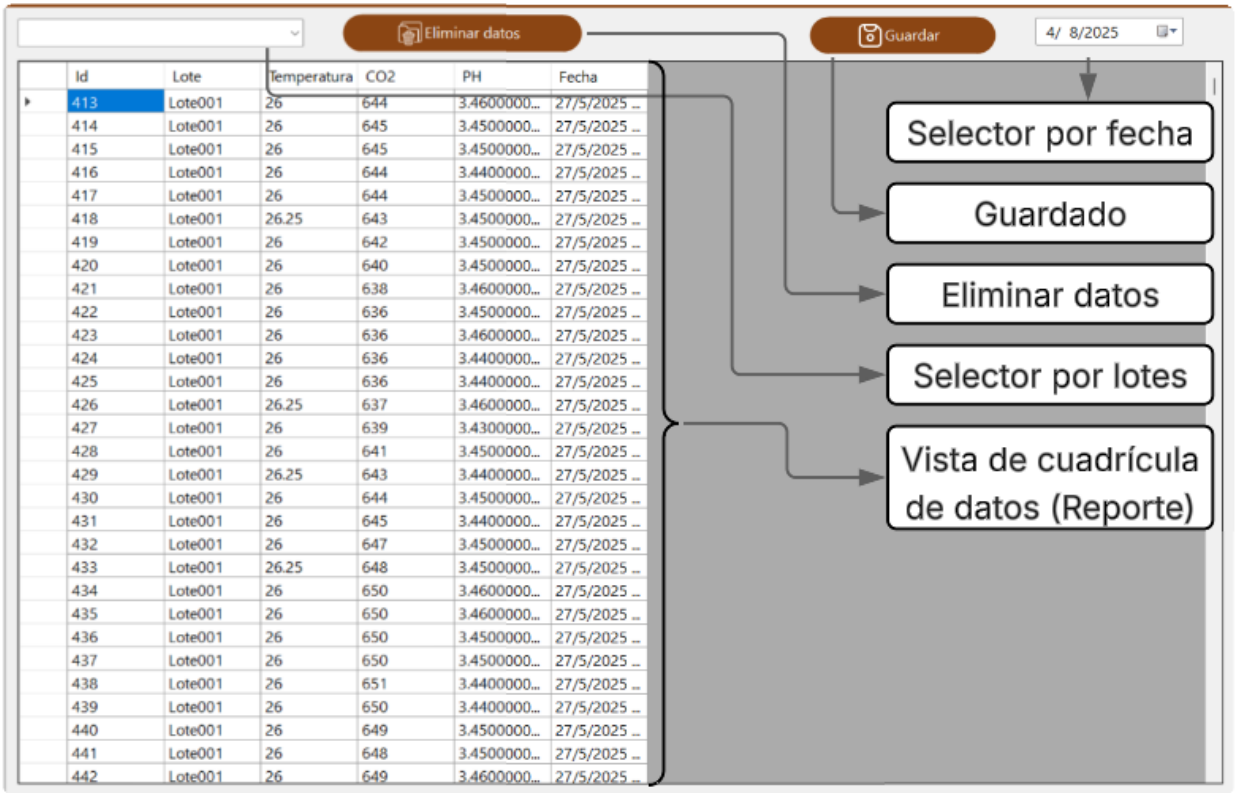


Figura 25. Guía funcionamiento Conexion.cs, Autoría Propia



Id	Lote	Temperatura	CO2	PH	Fecha
413	Lote001	26	644	3.4600000...	27/5/2025 ...
414	Lote001	26	645	3.4500000...	27/5/2025 ...
415	Lote001	26	645	3.4500000...	27/5/2025 ...
416	Lote001	26	644	3.4400000...	27/5/2025 ...
417	Lote001	26	644	3.4500000...	27/5/2025 ...
418	Lote001	26.25	643	3.4500000...	27/5/2025 ...
419	Lote001	26	642	3.4500000...	27/5/2025 ...
420	Lote001	26	640	3.4500000...	27/5/2025 ...
421	Lote001	26	638	3.4600000...	27/5/2025 ...
422	Lote001	26	636	3.4500000...	27/5/2025 ...
423	Lote001	26	636	3.4600000...	27/5/2025 ...
424	Lote001	26	636	3.4400000...	27/5/2025 ...
425	Lote001	26	636	3.4400000...	27/5/2025 ...
426	Lote001	26.25	637	3.4600000...	27/5/2025 ...
427	Lote001	26	639	3.4300000...	27/5/2025 ...
428	Lote001	26	641	3.4500000...	27/5/2025 ...
429	Lote001	26.25	643	3.4400000...	27/5/2025 ...
430	Lote001	26	644	3.4500000...	27/5/2025 ...
431	Lote001	26	645	3.4400000...	27/5/2025 ...
432	Lote001	26	647	3.4500000...	27/5/2025 ...
433	Lote001	26.25	648	3.4500000...	27/5/2025 ...
434	Lote001	26	650	3.4600000...	27/5/2025 ...
435	Lote001	26	650	3.4600000...	27/5/2025 ...
436	Lote001	26	650	3.4500000...	27/5/2025 ...
437	Lote001	26	650	3.4500000...	27/5/2025 ...
438	Lote001	26	651	3.4400000...	27/5/2025 ...
439	Lote001	26	650	3.4400000...	27/5/2025 ...
440	Lote001	26	649	3.4500000...	27/5/2025 ...
441	Lote001	26	648	3.4500000...	27/5/2025 ...
442	Lote001	26	649	3.4600000...	27/5/2025 ...

Figura 26. Guía de funcionamiento Reporte.cs, Autoría Propia

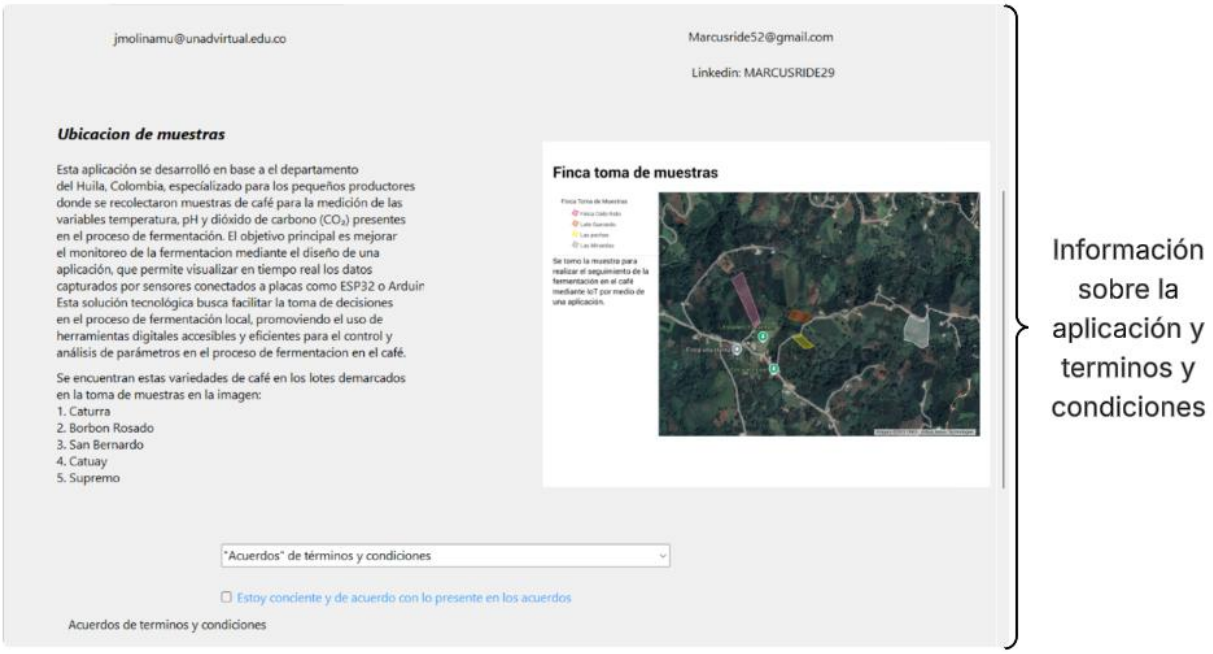


Figura 27. Guia funcionamiento de Nosotros.cs, Autoría Propia