UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Universidad Internacional de La Rioja

MÁSTER UNIVERSITARIO EN INDUSTRIA 4.0 (MI) - PER1172 2020-2021

Propuesta de transformación digital de una empresa de producción de minerales: "Ecuaminerales GB"

Trabajo Fin de Máster

Presentado por: García Alcalde, Enrique

Director/a: Alcalde Delgado, Roberto

Ciudad: Madrid
Fecha: martes, 10 de febrero de 2021

Resumen

Las nuevas tecnologías de la industria 4.0 pueden suponer un cambio de paradigma en la industria a nivel mundial e impulsar los procesos de digitalización en las medianas y pequeñas empresas. Ubicada en Ecuador, Ecuaminerales GB es una empresa dedicada a la fabricación de productos elaborados a partir de minerales no metálicos. La baja productividad y eficiencia de su proceso de fabricación pone en riesgo la continuidad del negocio. El presente Trabajo Fin de Máster tiene como objetivo impulsar la automatización de sus procesos por medio de una plataforma de internet de las cosas industrial. Para realizarlo, se ha tomado como marco metodológico la "Arquitectura de Referencia de Internet Industrial" desarrollada por "The Industrial Internet Consortium". El resultado de esta investigación es el diseño de una plataforma IIoT y la implementación de sus componentes principales. La implementación se lleva a cabo mediante el despliegue de un servidor Bróker MQTT Mosquitto y el desarrollo en el lenguaje de programación Python de distintos clientes MQTT. Esta plataforma es capaz de gestionar las órdenes de producción generadas por "Odoo ERP", realizar la tratabilidad del proceso y registrar medidas de calidad de los productos.

Palabras Clave: Industria 4.0, IIRA, MES, MQTT, Python

Abstract

The new Industry 4.0 technologies may mean a change in the paradigm of industry worldwide and may boost digitalization processes in small and medium-sized enterprises. Ecuaminerales GB is a small company headquartered in Ecuador that manufactures non-metallic mineral products, but whose automatization levels are close to none. The low productivity and lack of efficiency in the manufacturing processes jeopardize the business continuity. The goal of the present master's thesis is to boost the automatization of the company's processes with the help of an industrial internet of things platform. To do so, the Industrial Internet reference architecture developed by The Industrial Internet Consortium was used as methodological framework. As a result, an IIoT platform was designed and its main components were implemented. A Broker MQTT server was deployed and various MQTT clients were developed in Phyton. This platform allows to manage the production orders generated by the "Odoo ERP" and the traceability of the processes, and to track the quality standards of the products.

Keywords: Industry 4.0, IIRA, MES, MQTT, Python

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

El presente trabajo tiene por objetivo plantear un reto de innovación y transformación digital en una empresa privada que carece de procesos automatizados mediante la integración de tecnologías de la industria 4.0.

Ecuaminerales GB es una empresa familiar ubicada en Ecuador que lleva más de 20 años dedica a la producción y comercialización de minerales no metálicos. Ecuaminerales GB cuenta con 12 empleados y factura en torno a 1 millón de dólares anuales. Sus productos se destinan a arenas para mascotas y a los sectores agrícola, pecuario e industrial. Disponen de una fábrica en la que realizan el proceso de fabricación formada por dos líneas de procesamiento con diferentes molinos controlados mediante cuadros de mando eléctricos para tratar la materia prima. La empresa dispone de un sistema ERP (Enterprise-Resource-Planning) recién implantado que integra la información comercial y financiera. Los registros en el sistema son manuales y no existe ningún tipo de conexión con las líneas de producción. Ecuaminerales GB se encuentra en un momento en el cual necesita realizar la digitalización de los procesos. Su estrategia empresarial abarca los siguientes aspectos:

1. Innovación y mejora de procesos productivos y administrativos,
2. Incremento en ventas y posicionamiento de marca,
3. Desarrollo de productos con mayor valor agregado.

El Trabajo Fin de Máster “Propuesta de transformación digital de una empresa de producción de minerales: Ecuaminerales GB” se enfoca en apoyar al cumplimiento del primer objetivo de la estrategia empresarial, en concreto, en los aspectos relativos a la mejora de los procesos productivos.

1.1. Justificación

La nueva revolución industrial, la globalización y el impacto generado por la pandemia de la COVID-19 pone en riesgo la continuidad de la empresa ante un mercado cada vez más competitivo. La falta de conocimiento de los costes reales de producción hace incurrir a la empresa en grandes pérdidas económicas sin conocer las causas o el origen de éstas.

Este proyecto impulsa la transformación de la organización modernizándola y preparándola para el futuro. Supone una novedad en la manera de operar en los procesos productivos y

busca propiciar un cambio cultural en las personas que la integran. Esta transformación facilita la toma decisiones estratégicas, por medio del control de calidad y la trazabilidad de los procesos, respecto a políticas organizacionales y comerciales, generando valor.

El objetivo del proyecto es impulsar la transformación digital en la empresa basándose en la Industria 4.0, como un proceso de innovación en una empresa con nulos niveles de automatización y en un país en desarrollo.

1.2. Planteamiento del trabajo

Para poder mejorar la productividad de los procesos y reducir los costes es fundamental conocer el tiempo de ejecución de las tareas e identificar donde existen los posibles errores humanos o fallos que perjudiquen la calidad de los productos. Para conseguirlo, es necesario identificar y conocer el proceso en detalle por medio del control de la calidad de los productos y la trazabilidad de los procesos.

El Trabajo Fin de Máster “Propuesta de transformación digital de una empresa de producción de minerales: Ecuaminerales GB” presenta una solución mediante la implantación de las nuevas tecnologías de la información y la industria 4.0, la cual permite recoger información de la trazabilidad y la calidad de los procesos y los productos. La integración de sistemas ciberfísicos (CPS) a través de una plataforma de Internet de las cosas industrial (IIoT) posibilita obtener conocimiento del proceso, crear indicadores de calidad y reducir los costes de producción.

1.3. Estructura de la memoria

A lo largo de este trabajo fin de Máster se ha realizado una contextualización del estado del arte donde se resume la situación actual de los sistemas de trazabilidad y calidad en la Industria 4.0. A continuación, se han definido los objetivos a alcanzar y la metodología a seguir para la consecución de estos objetivos. Posteriormente, se ha analizado el proceso actual y diseñado una solución para el control de la calidad y la trazabilidad de los procesos. Adicionalmente, se han implementado los componentes principales de la solución mediante una prueba de concepto. Finalmente, se han analizado los resultados y propuesto posibles alternativas de futuro para la ampliación y mejora del sistema propuesto.

2. Contexto y estado del arte

2.1. Introducción

Las empresas industriales están cambiando sus métodos de producción, pasando de una producción basada en el producto a otra orientada a los servicios, en los que el cliente está en el centro de las decisiones. Esto es hoy en día posible gracias a la digitalización y a las tecnologías habilitadoras de la Industria 4.0 como el IIoT o los CPS (Liu et al., 2019). Los sistemas de control de la producción son capaces de monitorizar y documentar la gestión de una planta, reduciendo los costes y los tiempos de entrega, mejorando la productividad y aumentando la trazabilidad y la calidad de los productos (De Ugarte et al., 2009). Estos sistemas son fácilmente desplegables gracias a su integración con sistemas ciber-físicos mejorando la toma de decisiones. En la industria se denominan sistemas de ejecución de la fabricación del inglés Manufacturing Execution System (MES) (Iarovyi et al., 2016). Tecnologías como IIoT o Bigdata permiten recoger una gran cantidad de datos a lo largo de toda la cadena de valor del producto (Marjani et al., 2017). La aparición de estas y otras tecnologías posibilitan nuevas soluciones innovadoras para el desarrollo de estos sistemas de manera asequible y sencilla de implantar para pequeñas y medianas empresas (PYMES) (Almada-Lobo, 2016).

2.2. MES, CPS, IIoT e Industria 4.0

La aparición de la industria 4.0 y los CPS supone un cambio de paradigma en la automatización de los procesos promoviendo la interconexión entre el mundo físico y el virtual y convirtiendo las fábricas en ecosistemas inteligentes gracias a la interconexión y monitorización de las materias primas, los procesos y los productos en todos los niveles de la organización. Este alto grado de monitorización permite transformar todos esos datos en información para el análisis de máquinas y procesos aumentando su eficiencia para reducir los errores y predecir contingencias. Además, posibilita la creación de modelos digitales minimizado los costes de implantación y mantenimiento y permitiendo simular y optimizar los sistemas. La colaboración hombre-máquina toma un papel relevante dedicando a las máquinas a las tareas más repetitivas y a los operadores las tareas más creativas o la parametrización de los procesos a partir de la información extraída del sistema. Finalmente, toda esta información agregada se convierte en conocimiento facilitando la toma de decisiones de manera autónoma o a través de la supervisión y control que ofrecen estos sistemas a los niveles más altos de las organizaciones (Lee et al., 2015).

IIoT explota el potencial que ofrece Internet y amplía su capacidad para recopilar, analizar y distribuir datos que se pueden convertir en información y conocimiento. En la industria, estos sistemas ofrecen soluciones que permiten mejorar y ampliar la capacidad de monitorización y trazabilidad de los sistemas. IIoT es capaz de crear una red de comunicaciones para habilitar la implantación de CPS (Grgić et al., 2016). La combinación de tecnologías de CPS e IIoT permiten recoger información en tiempo real y ser compartida a diferentes niveles de los procesos de organizaciones de fabricación industrial (Zhang et al., 2018).

Los sistemas MES surgieron como una herramienta para satisfacer la necesidad de las empresas de adaptar su producción en términos de volumen, calidad, estandarización, reducción de costes y plazos. Existen diversas organizaciones que han definido estándares que recogen las principales funciones que deben cumplir estos sistemas y como se deben de conectar los sistemas de producción en fábrica con los niveles superiores de la organización (Iarovyi et al., 2016). En la Tabla 1 se pueden ver las funciones definidas para estos sistemas por las organizaciones Manufacturing Execution Systems Association (MESA), International Society of Automation (ISA) y Verein Deutsche Ingenieure (VDI):

Tabla 1. Funciones MES

MESA	ISA-95	VDI
Operaciones / planificación detallada	Producción detallada	Planificación detallada
Localización y estado de recursos	Planificación	Gestión de operación y recursos
Documentación y control	Recopilación de datos	Gestión de material
Despacho de producción	Gestión de recursos de producción	Proceso y adquisición de datos
Análisis de rendimiento	Gestión de definición de productos	Interfaz de gestión
Gestión del mantenimiento	Despacho de productos	Análisis del rendimiento
Gestión del proceso	Ejecución de productos	Gestión de la calidad y la información
Gestión de la calidad	Análisis de rendimiento de producción	
Recopilación y adquisición de datos		
Trazabilidad de productos		

Fuente: (Iarovyi et al., 2016)

La trazabilidad y la gestión de los datos de producción pueden repercutir muy favorablemente en la eficiencia de los procesos de las empresas. Sin embargo, los sistemas MES son habitualmente soluciones propietarias a menudo asequibles tan solo para las grandes empresas lo cual requiere una gran inversión en equipos dedicados y licencias, así como personal especializado para gestionarlas. En la Figura 1, se muestra un cuadrante elaborado por la consultora Gartner (Rick Franzosa, 2019) en la que se pueden ver los principales proveedores comerciales de soluciones MES en el mercado.

Figura 1. Cuadrante mágico para sistemas de ejecución de fabricación (Rick Franzosa, 2019)

Este informe recoge las principales características y funcionalidades que los sistemas MES comerciales comparten, entre las que destacan el despacho y envío de productos, gestión de la producción, recopilación de datos, bases de datos operativas, procesos de gestión de la calidad relacionados con la fabricación, ejecución de procesos, trazabilidad y generación de informes e indicadores clave de rendimiento (KPIs). Estos sistemas tratan de incluir, cada vez, una mayor integración con los ecosistemas de producción desde los sistemas ERP (Software planificación de recursos empresarial) a los sistemas de control en campo. Cabe destacar que, esta integración se orienta a fomentar el uso de sus sistemas propietarios. En la actualidad, HTML5 se ha convertido en el lenguaje de programación mayoritario para las tecnologías de interfaz de usuario de MES, aunque su adopción varía mucho de un proveedor a otro. Los servicios en la nube han comenzado a incorporarse en los sistemas MES, aunque su uso sigue siendo residual. Existen numerosos proveedores de sistemas MES en el mercado, los cuales se agrupan en distintas categorías: Proveedores de sistemas ERP como

Oracle o SAP, Proveedores PLM (Gestión de vida de productos) que ofrecen aplicaciones y tecnología en todo el ciclo del proceso como Dassault Systèmes o Siemens Digital Industries, proveedores de sistemas de automatización que también ofrecen soluciones SCADA (Sistemas de control, supervisión y adquisición de datos) como AVEVA o Rockwell Automation y proveedores específicos de sistemas MES como FORCAM que normalmente se dedican a nichos específicos del negocio.

La dificultad de integrar estos sistemas supone una gran barrera para las PYMES (De Ugarte et al., 2009; Urbina Coronado et al., 2018). Muchos de estos sistemas ni siquiera responden a las necesidades de las pequeñas empresas que requieren de soluciones menos complejas y adaptadas a volúmenes de producción más pequeños (Iarovyi et al., 2016). Las nuevas tecnologías habilitadoras de la industria 4.0 como IIoT permiten el desarrollo de estos sistemas de producción de manera sencilla y asequible. Diferentes experiencias basadas en estas tecnologías ofrecen arquitecturas y soluciones para la trazabilidad y la recopilación de datos en entornos industriales. Zhang et al. (2018) ofrecen un marco de referencia para la producción inteligente basada en CPS e internet de las cosas (IoT) para sistemas logísticos. Wan et al. (2018) proponen una solución basada en OPC UA mediante CPS e IoT para lograr una interacción y una gestión dinámica de los recursos. Grgić et al. (2016) presentan una solución IoT basada en la web para monitorear datos usando el protocolo Message Queuing Telemetry Transport (MQTT) en la agricultura. Iarovyi et al. (2016) plantean sistemas CPS para MES basados en tecnologías y software abiertos. Mourtzis et al. (2016) presentan una metodología para la recolección de datos mediante la adopción de IoT gracias al uso de dispositivos móviles. Park (2015) y Urbina Coronado et al. (2018) evalúan la utilización de dispositivos móviles para recoger datos de producción y examinan su viabilidad en las PYMES.

Las arquitecturas ofrecen una visión simplificada y abstracta de una implementación en un sistema IIoT. La Industrial Internet Consortium presenta 3 diseños de arquitecturas de referencia para soluciones IIoT de alto nivel (Lin et al., 2017):

- Arquitectura de tres niveles (Edge, Plataforma, Empresa),
- Patrón de administración y conectividad perimetral mediado por puerta de enlace,
- Patrón de bus de datos en capas.

Las arquitecturas basadas en tecnologías IoT se integran a través de plataformas. Estas plataformas proporcionan a los sistemas los servicios y características necesarios para realizar la conectividad entre dispositivos y los sistemas de back-end. La selección de una

plataforma debe considerar factores tales como como, administración de dispositivos seguridad, integración, protocolos de datos, gestión y análisis de datos, sistemas de visualización y aspectos relacionados con el negocio o la viabilidad técnica y financiera (Choi et al., 2018; Hejazi et al., 2018).

Existen numerosas plataformas a nivel industrial y comercial que provienen de distintos ámbitos y ofrecen distintos tipos de servicios. Las plataformas Cloud de grandes proveedores de servicios en internet ofrecen desde servicios de infraestructuras, plataformas o incluso aplicaciones, las plataformas industriales que ofrecen soluciones específicas para la industria y las plataformas de código abierto impulsadas por instituciones, comunidades o fundaciones que pretende ofrecer un marco base para el desarrollo de plataformas dedicadas. En ocasiones, las soluciones pueden combinar servicios de distintas plataformas, dado que la mayoría de ellas están disponibles de manera modular y son interoperables (Gartner, 2019; Guth et al., 2017).

En la Tabla 2, pueden verse las principales plataformas IoT:

Tabla 2. Plataformas IoT

Plataformas Cloud	Plataformas industriales	Plataformas open-source
AWS IoT (Amazon Web Services).	ThingWorx (PTC).	FIWARE
Azure IoT (Microsoft).	Cumulocity (Software AG)	IoT Eclipse Foundation
Google IoT Core (Google Cloud Platform).	Lumada (Hitachi)	SiteWhere
Watson IoT Platform (IBM).	Bosch IoT Suite (Bosch).	
	Predix (General Electric).	
	MindSphere (Siemens).	

Fuente: Adaptado(Gartner, 2019; Guth et al., 2017; Pelino & Miller, 2019)

La selección de las tecnologías de comunicación estándar y en tiempo real es clave a la hora de diseñar una solución para la industria 4.0. Estas soluciones usan protocolos ligeros que han sido desarrollados para cumplir con las nuevas necesidades tecnológicas. Estos protocolos están diseñados para reducir la carga de las redes de comunicación y proporcionar comunicaciones seguras. Además, su implementación suele ser más sencilla que la de protocolos industriales clásicos (Naik, 2017). El uso de estos protocolos también ha contribuido a la proliferación del uso de sensores inalámbricos en el ámbito de IoT (Thangavel et al., 2014). Diferentes protocolos destacan actualmente por su flexibilidad, seguridad y

disponibilidad en soluciones para escenarios basados en CPS. OPC UA ha resultado de la evolución OPC Classic y es uno de los protocolos más extendidos en la industria y aceptado por la comunidad científica ofreciendo flexibilidad, interoperabilidad y la capacidad de comunicar información con gran riqueza semántica (Graube et al., 2017). El uso de OPC UA permite conectar dispositivos a distintos niveles de la empresa y es compatible con diversos protocolos de campo (Luo et al., 2017). MQTT es junto OPC UA el protocolo más usado para aplicaciones de IoT. MQTT es un protocolo ligero que gracias a su arquitectura servidor/ bróker es ideal para integrar arquitecturas distribuidas con sensores inalámbricos (Naik, 2017; Thangavel et al., 2014). Otros protocolos como CoAP, AMQP y HTTP, también son ampliamente utilizados en este tipo de aplicaciones. Su selección dependerá de la aplicación a llevar a cabo y sus necesidades. Así, aspectos como longitud de los telegramas, consumo de energía, ancho de banda requerido, latencia en las comunicaciones en tiempo real, disponibilidad, interoperabilidad, aprovisionamiento, seguridad, capacidad de estandarización o usabilidad son claves a la hora de su selección (Naik, 2017).

2.3. Conclusiones

Muchas experiencias basadas en CPS e IIoT han sido desplegadas y probadas en entornos reales. Pese a ello, el uso de CPS e IIoT en sistemas de ejecución de la fabricación todavía no son predominantes en la industria y continúa habiendo una brecha en la literatura científica que ofrezca un claro marco de referencia para la adopción de estas soluciones (Liu et al., 2019; Marjani et al., 2017). Sin embargo, el éxito de estas experiencias muestra la tendencia en el estado del arte de los sistemas de calidad y trazabilidad de los procesos y el potencial de su aplicación en la Industria 4.0. Además, estas experiencias muestran como estas nuevas tecnologías tienen la capacidad de democratizar los sistemas de ejecución de la fabricación y hacerlos accesibles a las pequeñas y medianas empresas.

Los conceptos y tecnologías revisados en este estado del arte son el marco conceptual de referencia sobre el cual ha sido desarrollado este trabajo fin de máster. La elección de una solución adecuada requiere de un estudio profundo en un escenario complejo en el que diferentes tecnologías aparecen como alternativas en un mercado muy fragmentado y con un bajo nivel de estandarización.

3. Descripción general de la contribución del TFM

3.1. Descripción general de la propuesta

Este trabajo fin de máster propone una solución basada en las tecnologías habilitadoras de la industria 4.0 para garantizar la trazabilidad y calidad de un proceso de fabricación de minerales no metálicos. Esta solución es capaz de recopilar información sobre el proceso y sobre la calidad de los productos. Su aplicación pretende demostrar como el uso de tecnologías como CPS o IIoT pueden favorecer la implantación de sistemas asequibles de gestión de la producción en PYMES con bajos o nulos niveles de automatización.

3.2. Objetivos específicos

Los objetivos específicos se plantean a fin de cumplir el objetivo general de gestionar la trazabilidad y la calidad del proceso. Estos objetivos se detallan a continuación:

- Analizar las necesidades del proceso.
- Diseñar una arquitectura capaz gestionar las órdenes de producción generadas por un ERP, realizar la tratabilidad del proceso y recopilar mediciones del proceso para el control de la calidad de los productos.
- Seleccionar los componentes tecnológicos que hagan posible la implementación real de la arquitectura propuesta.
- Implementar los componentes claves de la plataforma y simular una prueba de concepto que permita probar la arquitectura diseñada.

3.3. Metodología del trabajo

En primer lugar, la información esencial de este Trabajo Fin de Máster proviene de: (i) una revisión profunda de literatura sobre los sistemas comerciales de ejecución de la producción y de los sistemas basados en las nuevas tecnologías habilitadoras de la industria 4.0; (ii) revisión de la documentación facilitada por la empresa Ecuaminerales GB para estudiar el proceso de fabricación y definir sus necesidades; y, (iii) reuniones de trabajo con la empresa estudiada para poder definir con detalle sus prioridades y capacidades. En segundo lugar, para el diseño de arquitectura se ha utilizado como marco de referencia metodológico IIRA (Internet Industrial Reference Arquitecture), desarrollada por “The Industrial Internet Consortium” que tienen como propósito proporcionar orientación y asistencia en el desarrollo,

documentación, comunicación y despliegue de sistemas IIoT (Industrial Internet Consortium (IIC), 2015). A partir de cuatro puntos de vista negocio, uso, vista y funcional IIRA, permite crear un análisis detallado para cada punto de vista con el objetivo de satisfacer las necesidades identificadas por medio de un sistema IIoT.

A partir de esta base, se realiza una triangulación de la información recopilada y analizada para identificar las principales líneas de acción del TFM y las potenciales aplicaciones de la tecnología a fin de facilitar el proceso de automatización de Ecuaminerales GB.

4. Sistema de control de la calidad y trazabilidad de los procesos

4.1. Proceso actual e identificación requisitos

4.1.1. Descripción proceso actual

El proceso analizado realiza la fabricación de distintos productos de minerales no metálicos. En concreto, la línea objeto de análisis usa como materia prima bentonita cálcica proveniente de una explotación minera. El proceso de fabricación se realiza de manera secuencial y se basa en el secado y la molienda de la materia prima para obtener una humedad y una densidad granulométrica determinadas para la aplicación final de cada producto. En el siguiente gráfico se puede observar el flujo del proceso, en el que se identifican las etapas principales que lo componen y sus flujos de información:

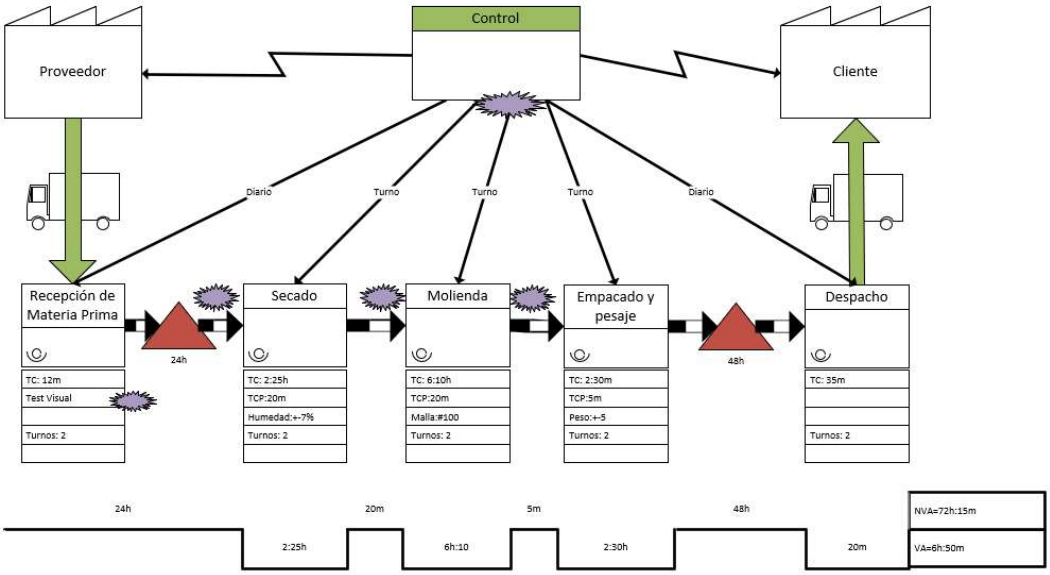


Figura 2. Value Stream Mapping. Elaboración propia

La lista de etapas del proceso y sus actividades se detallan en la Tabla 3:

Tabla 3. Lista de etapas del proceso

PROCESO	ACTIVIDADES	RESPONSABLE
Recepción de Materia Prima	Test visual (color y tamaño)	Jefe de Producción
	Pesaje de camión recibido	
	Inspección visual del material	
	clasificación y separación en caso de contaminación	
	Descarga y almacenaje	
	Registro documental administración	
Transporte al Área de Procesamiento	Transporte con maquinaria	Operario
Secado	Carga Horno	Operario
	Proceso de secado por ventilación forzada y calor en tubo de secado	
	Test de humedad en Laboratorio	
	Almacenaje de materia prima seca	
Adición Insumos	Adición manual Zeolita (verde)	Operario
Molienda	Carga Molino	Operario
	Proceso de trituración de la piedra, entrada de material de 30/60cm por la boca de la trituradora y salida del material 2/5cm	
Empacado y Pesaje	Llenado de la pala del montacargas con materia prima triturada	Operario
	Test granulometría Laboratorio	
	Llenado sacos	
	Pesado	
	Sellado bolsa plástica interna (zuncho)	
	Sellado (cosido)	
Transporte Bodega de Producto Terminado	Almacenaje sobre pallets	Operario
	Ubicación en zona para despacho	
Despacho	Carga de productos al Camión	Jefe producción y administración
	Facturación y procesos tributarios	

Fuente: Elaboración propia

Actualmente, los procesos comerciales y de facturación se realizan a través del ERP Odoo. Odoo ERP es un software Open source que se comercializa por módulos y permite realizar una gestión integral de los recursos de la empresa en las áreas de ventas y operaciones. Además, Odoo dispone de una comunidad de desarrollo en el que distintas empresas ofrecen nuevas soluciones y aplicaciones que complementan sus funcionalidades.

En este momento Ecuaminerales usa los módulos de contabilidad, CRM y comercio electrónico. El software ERP genera órdenes de producción, rutas de material y recetas que son transmitidas a producción de manera manual.

Adicionalmente, la empresa está desarrollando una aplicación en Power BI para analizar la información que obtiene del ERP y otra información adicional sobre mediciones realizadas sobre el proceso y que se registran de manera manual en Excel, como por ejemplo el tiempo de funcionamiento de máquina.

4.1.2. Identificación de requisitos

La identificación de requisitos se ha definido teniendo en cuenta el análisis detallado del proceso realizado en el apartado anterior, a partir del estudio de la documentación facilitada por la empresa y de diversas reuniones de trabajo mantenidas con sus responsables. En la Tabla 4 se establecen los requisitos y funciones a cumplir por el sistema para poder realizar la trazabilidad del proceso y obtener medidas que permitan obtener un conocimiento minucioso del proceso.

Tabla 4. Requisitos proceso

PROCESO	TRAZABILIDAD	MEDICIÓN
Gestión de la fabricación	Orden producción	No aplica
	Receta (Productos necesarios y cantidades)	
	Lote producción	
	Ruta producción (origen-Destino)	
Recepción de Materia Prima	Registro llegada camión	Peso
		Humedad
		Temperatura
Transporte al Área de Procesamiento	Orden producción	Peso

Secado	Orden producción	Humedad
	Control arranque/paro y velocidad	Temperatura
Molienda	Orden producción	Humedad
	Control arranque/paro y velocidad	Temperatura
		Granulometría
Empacado y Pesaje	Orden producción	Peso
	Marcado QR	
Transporte Bodega de Producto Terminado	Orden producción	Operario
Despacho	Orden producción	No aplica

4.2. Solución propuesta

4.2.1. Justificación y presentación de la solución

En este apartado se presenta la solución diseñada a partir del marco metodológico de la arquitectura de referencia de Internet Industrial IIRA desarrollada por “The Industrial Internet Consortium”. Este marco de referencia ha servido para definir las características a satisfacer por la plataforma IIot y para definir la arquitectura a implementar. Esta plataforma trata de satisfacer dos de las funcionalidades definidos por MESA para los sistemas MES: gestión de calidad y trazabilidad de los productos.

4.2.2. Arquitectura de la solución

La aplicación de la metodología de IIRA permite definir los actores, las actividades y los objetivos que la solución basada en IIoT debe cumplir para garantizar y satisfacer las necesidades del sistema estudiado. Para ello se han analizado los diferentes puntos de vista propuestos por IIRA, puntos de vista de negocio, uso, funcional e implementación.

4.2.2.1 Punto de vista del negocio

El punto de vista de negocio recoge la visión, los valores y los objetivos de las partes interesadas del negocio en su contexto empresarial y regulatorio. Para llevarlo a cabo se evalúan aspectos orientados a los negocios, como el valor comercial, el rendimiento esperado de la inversión, el costo de mantenimiento y la responsabilidad del producto.

- Actores relevantes:
    - Socio fundador: En la actualidad no trabaja en la empresa, pero participa de la toma de decisiones estratégicas de la empresa.
    - Director Marketing: Lidera la toma de decisiones sobre los cambios organizacionales y tecnológicos en la empresa.
    - Director Administración y Humanos: Realiza la gestión y el control financiero y es clave para la asignación de presupuestos.
    - Director Ingeniería: Es el actor con mayor conocimiento del proceso productivo y sus necesidades.
- Visión: La visión empresarial está enfocada en:
    - Innovación y mejora de procesos productivos y administrativos.
    - Mejora de la calidad de los productos.
    - Desarrollo de productos con mayor valor agregado.
- Valores: El sistema debe perseguir los siguientes valores:
    - Aumentar el conocimiento del proceso.
    - Reducción de costes.
    - Aumentar productividad.
- Objetivos Clave:
    - Automatizar los procesos.
    - Integrar la información de producción y administración en un sistema informatizado para poder realizar la trazabilidad del proceso de fabricación.
    - Controlar la calidad de los productos.
- Capacidades fundamentales:
    - Obtener información para el control de calidad proceso.
    - Trazabilidad del proceso de producción.
    - Conectar el ERP con producción.
    - Capacidad para arrancar y parar automáticamente las máquinas.

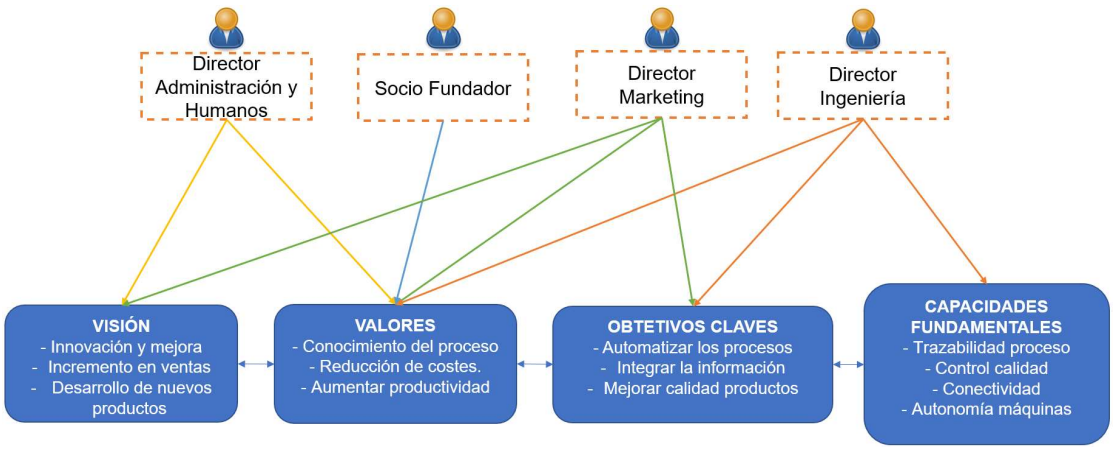


Figura 3. Punto de vista del negocio. Elaboración propia

4.2.2.2 Punto de vista de uso

El punto de vista de uso identifica las distintas actividades tareas, roles, partes y actividades involucradas en el proceso de producción que llevará a cabo el sistema IIot, que para el caso de uso se detallan en la Tabla 5. Una actividad es una consecución de tareas específicas realizadas por una parte o varias partes que asumen un rol determinado en dicha tarea.

Tabla 5. Punto de vista de uso

Actividad	Rol	Parte	Tarea
Recepción de Materia Prima	Tomar datos	Operario	Recepción albarán
	Medir	Sensor	Test Humedad
	Identificar	ERP	Asignar número lote
Transporte al Área de secado	Ordenar producción	ERP	Generar orden de producción
	Enviar Orden	MES	Enviar orden a operario
Secado	Confirmar carga	Operario	Carga hornos
	Enviar receta	MES	Enviar receta a horno
	Secar	Autómata	Secado
	Arranque	Autómata	Enviar evento
	Paro	Autómata	Enviar evento
	Medir	Sensor	Test Humedad
Transporte área de Molienda	Ordenar producción	ERP	Generar orden de producción
	Enviar Orden	MES	Enviar orden a operario
Molienda	Confirmar carga	Operario	Carga Molinos
	Enviar receta	MES	Enviar receta a Molino
	Secar producto	Autómata	Secado
	Arranque	Autómata	Enviar evento
	Paro	Autómata	Enviar evento
	Medir Humedad	Sensor	Test Humedad
Transporte área Empacado	Ordenar producción	ERP	Generar orden de producción
	Enviar Orden	MES	Enviar orden a operario
Empacado y Pesaje	Medir granulometría	Laboratorio	Enviar medida
	Medir peso	Sensor	Enviar evento
	Escanear QR	Operario	Enviar evento
Transporte Bodega de Producto Terminado	Transportar	Operario	Transporte
	Almacenar	Operario	Almacenaje sobre pallets
Despacho	Cargar	Operario	Carga de productos al Camión
	Facturar	Administrativo	Facturación

Toda actividad tiene un desencadenante, un flujo de trabajo, un efecto y unas restricciones. Para este caso de uso se han definido las siguientes actividades que coinciden con las distintas etapas del proceso (ver Figura 4 y 5):

1- Recepción de Materia Prima:
    - Desencadenante: Llegada de camión a la fábrica
    - Flujo de trabajo:
        - Escáner albarán
        - Envío albarán por e-mail
        - Se registra medida de peso
    - Efecto: Asignación lote a materia prima y destino en almacén
    - Restricciones: El albarán no se registra en el sistema el e-mail será procesado por un administrativo y no se alojará en la base de datos
2- Transporte al Área de secado
    - Desencadenante: Orden de producción en ERP
    - Flujo de trabajo:
        - MES envía a operario mensaje con orden de producción
        - Operario confirma que la operación ha sido realizada
        - MES envía a ERP confirmación
    - Efecto: Materia prima en área de secado preparada para cargar
    - Restricciones: La confirmación la realizará el operario en pulsador electromecánico situado a pie de línea y conectado a un PLC (Controlador lógico programable) ubicado en la estación de secado
3- Secado
    - Desencadenante: Orden de producción en ERP
    - Flujo de trabajo:
        - MES recibe orden de producción de ERP y lo envía a operario

        - Operario recibe orden de carga
        - Operario confirma horno cargado en pulsador electromecánico
        - MES envía orden a horno y comienza a funcionar
        - Medida de humedad continua hasta alcanzar humedad objetivo
        - Horno termina secado
        - Operario recibe orden de descarga
        - MES envía a ERP confirmación orden finalizada
    - Efecto: Producto seco
    - Restricciones: La confirmación la realizará el operario en pulsador electromecánico situado a pie de línea y conectado al PLC de secado

4- Transporte área de Molienda

    - Desencadenante: Orden de producción en ERP
    - Flujo de trabajo:
        - MES envía a operario mensaje con orden de producción
        - Operario confirma que la operación ha sido realizada
        - MES envía a ERP confirmación
    - Efecto: Materia prima en área de secado preparada para cargar
    - Restricciones: La confirmación la realizará el operario en pulsador electromecánico situado a pie de línea y conectado a un PLC ubicado en la estación de Molino

5- Molienda

    - Desencadenante: Orden de producción en ERP
    - Flujo de trabajo:
        - MES recibe orden de producción de ERP y lo envía a operario
        - Operario recibe orden de carga

        - Operario confirma horno cargado en pulsador electromecánico
        - MES envía orden a horno y comienza a funcionar
        - Medida de humedad continua
        - Horno termina secado
        - Operario recibe orden de descarga
        - Operario recoge muestra para test granulometría en laboratorio.
        - MES envía a ERP confirmación orden finalizada
    - Efecto: Producto Molido
    - Restricciones: La confirmación la realizará el operario en pulsador electromecánico situado a pie de línea y conectado al PLC de Molino

6- Transporte área Empacado

    - Desencadenante: Orden de producción en ERP
    - Flujo de trabajo:
        - MES envía a operario mensaje con orden de producción
        - Operario confirma que la operación ha sido realizada en pulsador
        - MES envía a ERP confirmación
    - Efecto: Materia prima en área de secado preparada para cargar
    - Restricciones: La confirmación la realizará el operario pulsando un pulsador a pie de línea conectado al PLC de secado

7- Empacado y Pesaje

    - Desencadenante: Orden de producción en ERP
    - Flujo de trabajo:
        - MES recibe orden de producción de ERP y lo envía a operario
        - Operario recibe orden de llenado
        - Operario escanea los sacos una vez llenados sobre la bascula

        - MES registra código QR y peso del saco
        - Operario confirma pallet completo escaneando código QR en Pallet
        - MES envía a ERP confirmación orden finalizada
    - Efecto: Producto empacado y paletizado
    - Restricciones: El operario deberá escanear en el orden indicado los códigos para que la operación sea realizada correctamente

8- Transporte Bodega de Producto Terminado

    - Desencadenante: Orden de producción en ERP
    - Flujo de trabajo:
        - MES recibe orden de producción de ERP y lo envía a operario
        - Operario apila pallet escanea QR de área en almacenaje y escanea pallet
        - MES envía a ERP confirmación orden finalizada
    - Efecto: Producto en bodega
    - Restricciones: El operario deberá escanear en el orden indicado los códigos para que la operación sea realizada correctamente

9- Despacho

    - Desencadenante: Llegada de camión a la fábrica
    - Flujo de trabajo:
        - ERP genera orden de trabajo
        - MES enviar orden de carga trabajo a operario
        - Operario carga camión y escanea pallets
        - Confirma fin del trabajo por aplicación móvil.
    - Efecto: Camión cargado con material

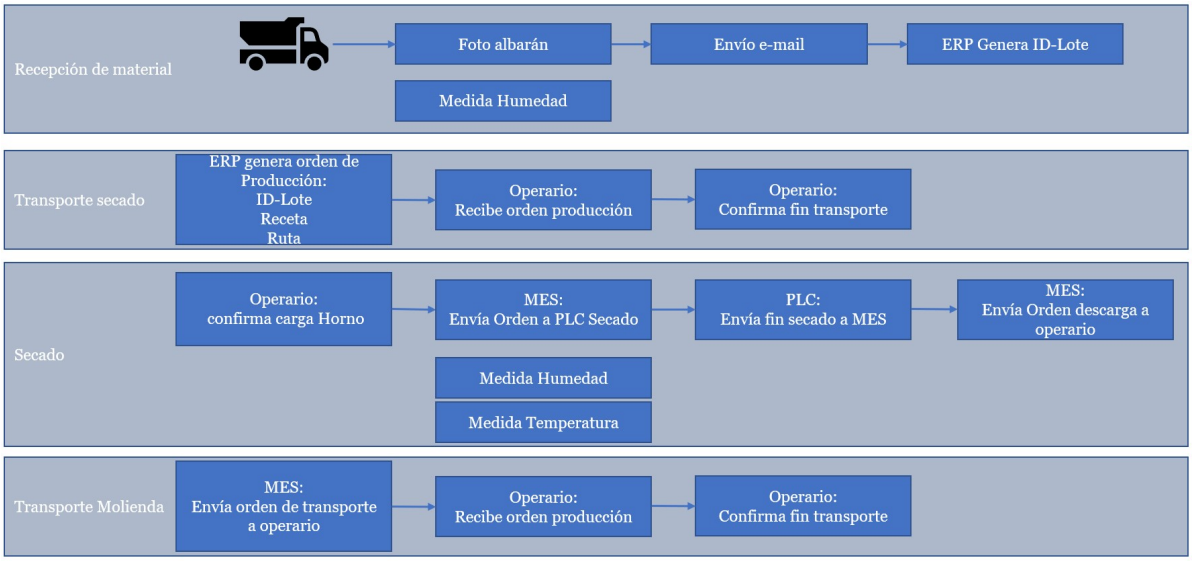


Figura 4. Flujo de trabajo por actividades 1. Elaboración propia

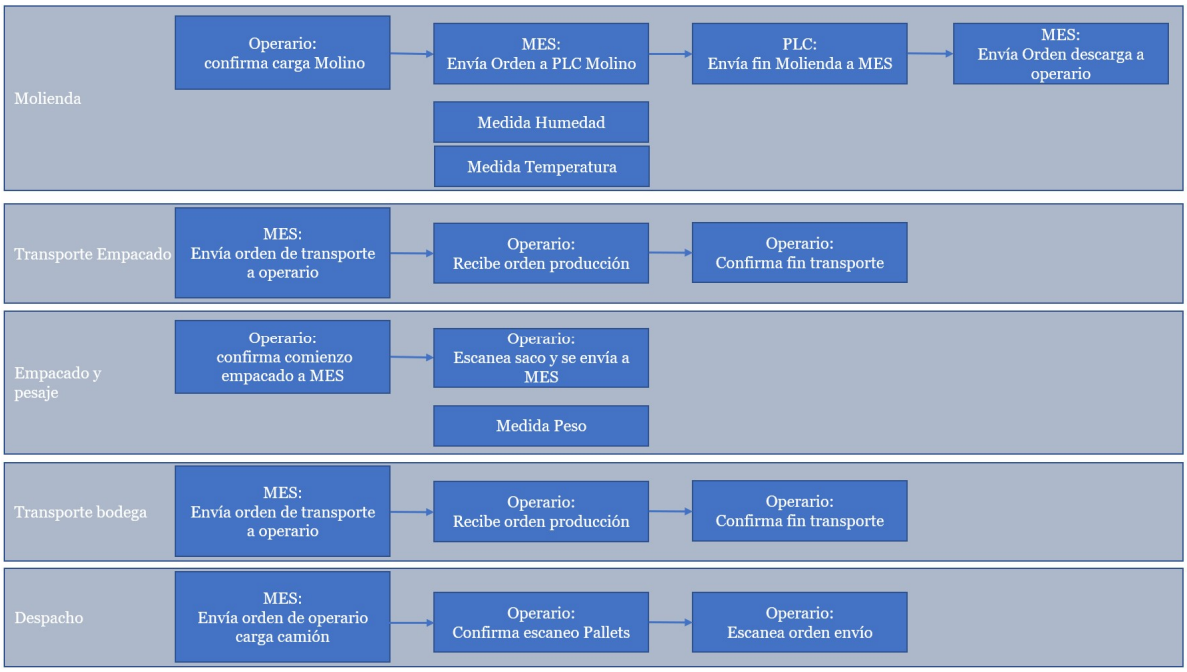


Figura 5. Flujo de trabajo por actividades 2. Elaboración propia

4.2.2.3 Punto de vista funcional

Con el punto de vista funcional se trata de identificar los intereses de los actores respecto al sistema IIoT en los distintos dominios definidos por IIRA:

- Dominio de control:

  El dominio de control tiene como objetivo recoger información del sistema mediante el uso de sensores inteligentes. El sistema recoge información de la línea de producción y controla, mediante autómatas industriales, el arranque paro de las máquinas y su

velocidad de funcionamiento. Los sensores podrán estar conectados a los autómatas de línea o directamente al sistema en función de los protocolos de comunicación que utilicen.

- Dominio de operaciones:

  El dominio de operaciones permite el control y supervisión de los elementos del dominio de control, como el control manual de las máquinas o generación de eventos y alarmas.

- Dominio de información:

  El dominio de información recoge y sirve los datos tanto del sistema físico como de las órdenes realizadas por el sistema ERP y MES.

- Dominio de aplicación:

  Este dominio realiza la conectividad con los sistemas ERP y de cuadro de mandos para la gestión de la información de alto nivel.

- Dominio de negocio:

  El dominio de negocio aloja el actual ERP y el sistema de inteligencia de negocio existentes.

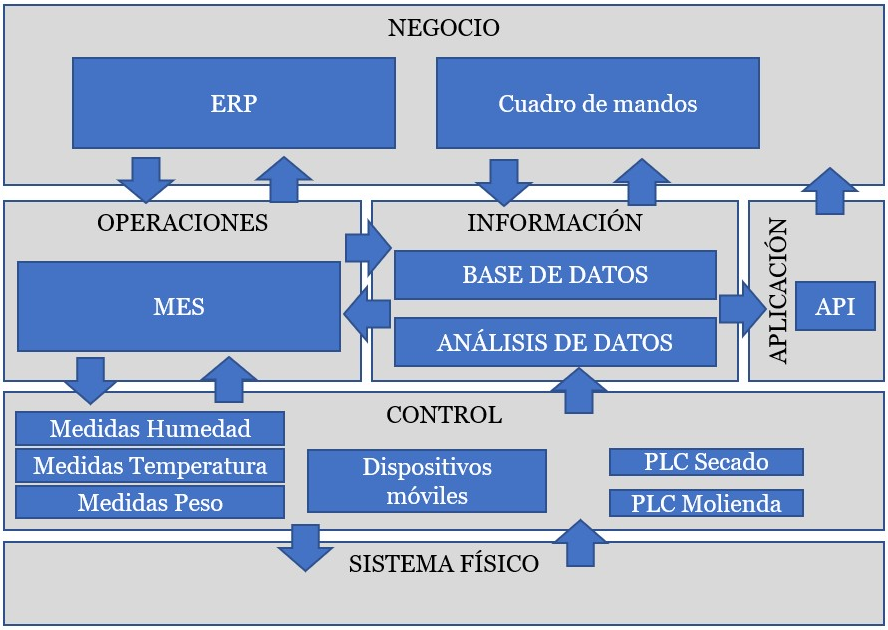


Figura 6. Punto de vista funcional. Elaboración propia

4.2.2.4 Punto de vista de implementación

El punto de vista de implementación ofrece una representación técnica del sistema IIoT, así como, las tecnologías y componentes necesarios para implementar las actividades y funciones descritas en los tres apartados anteriores. Para llevarlo a cabo se ha realizado, tal y como describe IIRA, la arquitectura general de un sistema IIoT respecto a su estructura, la distribución de los componentes y la topología a través de la cual están interconectados. Esta arquitectura se ha realizado utilizando el patrón de arquitectura de tres niveles propuesto por IIRA, el mismo que comprende los niveles de borde (Edge), plataforma y empresa. Estos niveles representan los roles específicos en el procesamiento de los flujos de datos y los flujos de control involucrados en las actividades de uso anteriormente descritas.

En la Figura 7 se representa un mapa de implementación que muestra las características clave del sistema y sus interacciones a través de una arquitectura de tres niveles y los distintos dominios funcionales. En el nivel de empresa quedan integrados los niveles de negocio y aplicación que se implementan a través del software ERP de Odoo y Power BI. En el nivel de plataforma se integran los dominios de información y operaciones que se encargan de recopilar la información del sistema, analizarla y dar soporte para realizar tareas de mantenimiento y configuración de los equipos. Finalmente, en el nivel "edge" se encuentran los dispositivos provistos de sensores y actuadores encargados de interactuar con el mundo físico y un Gateway que sirve como punto de conexión entre dispositivos y con los niveles superiores del sistema.

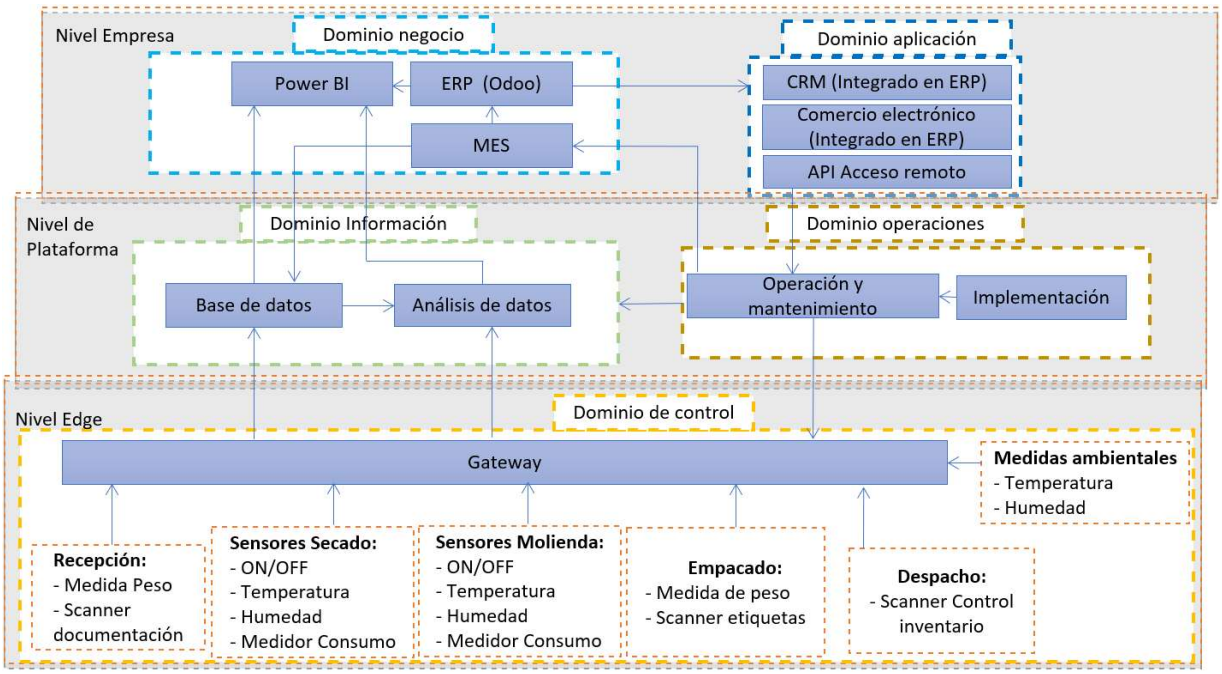


Figura 7. Punto de vista de implementación. Patrón de arquitectura de tres niveles (3-tier). Elaboración propia

A continuación, se definen técnicamente los componentes, así como sus interfaces, protocolos y principales funcionalidades o propiedades:

- Nivel Empresa: El nivel empresarial implementa los dominios de negocio y aplicación que dan soporte a las decisiones ejecutivas y de gestión y proporciona interfaces de visualización a los usuarios finales. En él se reciben flujos de datos del nivel de borde y plataforma y se envían comandos de control al nivel de plataforma y al nivel de borde.
    - ERP Odoo: El ERP ejecuta las órdenes de producción, las rutas de los materiales, así como, las recetas de los productos. Además, integra funciones como un CRM para la gestión de los clientes y un servicio de comercio electrónico con una tienda online dirigida a la venta y marketing de nuevos clientes.
    - Power BI: Power BI proporciona un interfaz gráfico para mostrar estadísticas y analizar los datos proporcionados por el dominio de información. Esto permitirá tomar decisiones y evaluar la productividad y el rendimiento del proceso.
    - API acceso remoto: Mediante un interfaz grafico se podrá acceder a los distintos niveles de plataforma para la visualización de datos y el estado del proceso o recibir alarmas.
- Nivel Plataforma: rocesa y sirve de interfaz de los comandos de control del nivel empresarial al nivel Edge. Consolida los procesos y analiza los flujos de datos del nivel Edge. Además, permite la gestión para dispositivos y activos.
    - Base de datos: Se aloja en un servidor local y recolecta los datos del sistema y los sirve a los distintos dominios para su análisis y visualización.
    - Análisis de datos: Este módulo realizará análisis predictivos de los datos de proceso con el objetivo de mejorar el rendimiento y predecir fallos.
    - Operación mantenimiento e implementación: El nivel de plataforma habilita la conectividad a todos los dispositivos del proceso para su configuración, diagnóstico y mantenimiento.

- Nivel Edge:
    - Gateway: Realiza la conexión con los dispositivos de campo recogiendo la información y sirviéndola al nivel de plataforma. Realiza la comunicación a través de Ethernet TPC/IP con las capas superiores de la arquitectura y utilizará Wifi e Industria Ethernet utilizando el protocolo MQTT con los dispositivos de campo. Actúa como bróker para la suscripción y publicación de temas de los distintos dispositivos de la red. En el caso de existir un dispositivo que no soporte MQTT podrá realizarse un agente que traduzca la información desde protocolo nativo del dispositivo a MQTT.
    - Autómatas Secado y Molienda. Dos autómatas industriales permiten controlar los arranques y paros del proceso y la velocidad de funcionamiento de máquina. Se conectan al sistema a través de Ethernet Industrial usando el protocolo MQTT. Además, estos autómatas permitirán conectar al sistema aquellos sensores que no soportan MQTT como es el caso de los sensores de humedad.
    - Sensores nivel de peso. Miden el peso de los camiones en la recepción de material y miden el peso de los sacos de productos fabricados. Se comunican de manera inalámbrica a través de wifi y usan el protocolo MQTT.
    - Sensor de condiciones ambientales: Este sensor permite obtener la temperatura y humedad ambiente para poder observar como las condiciones del entorno afectan al proceso de producción. Se conecta por Wifi a través de MQTT.
    - Dispositivos Móviles: Los dispositivos móviles permiten interactuar con el sistema a los operarios. Reciben órdenes de producción y utilizan la cámara para escanear los albaranes en el proceso de escaneado y para los códigos QR de los sacos en el proceso de empacado.

4.2.3. Tecnología seleccionada para solución

Una vez definida la arquitectura del sistema y definidas sus funcionalidades y tecnologías, se han seleccionado los componentes industriales para satisfacer las especificaciones definidas en el apartado anterior. El criterio principal a seguir es el de garantizar la compatibilidad con las tecnologías existentes en la empresa y la de utilizar herramientas open-source que no requieran licencias o cánones de uso.

En la Figura 8 se muestra la arquitectura con los componentes seleccionados, los distintos elementos del sistema y sus protocolos e interfaces de comunicación.

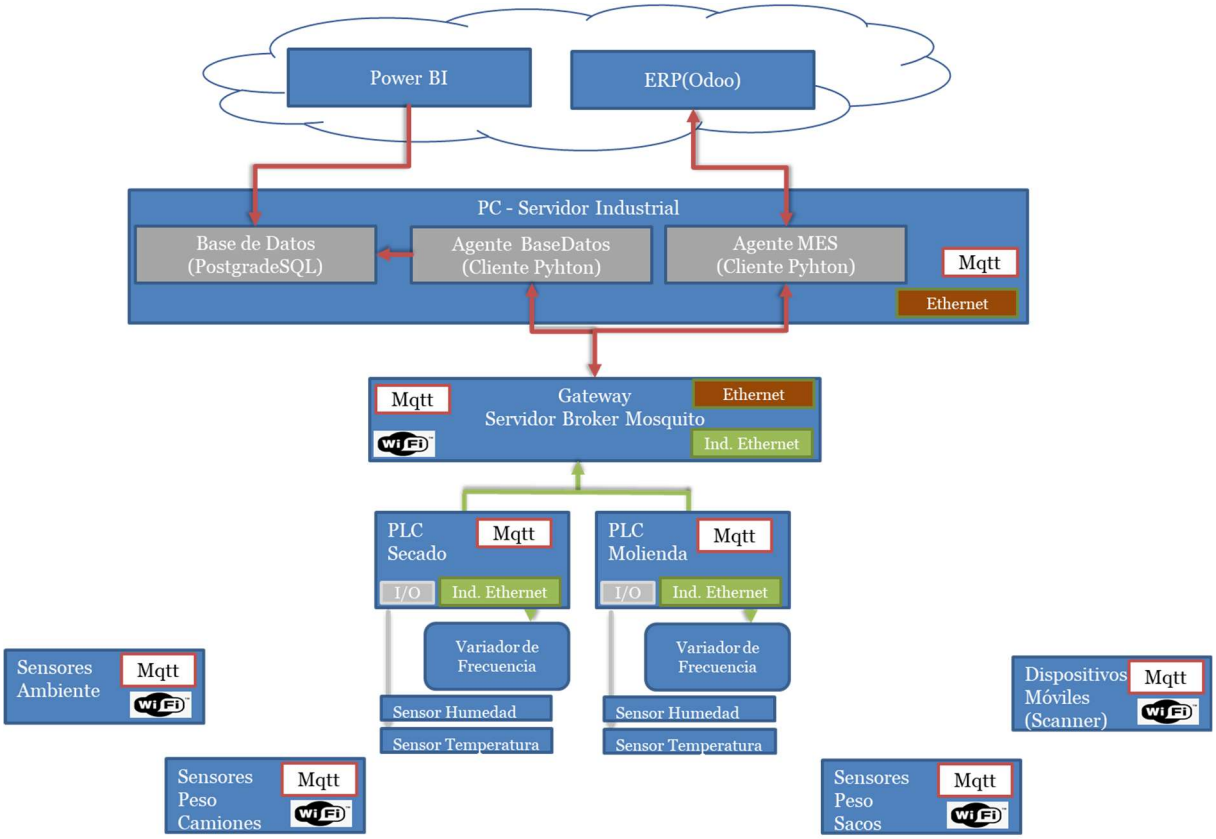


Figura 8. Punto de vista de implementación con dispositivos y protocolos de comunicaciones. Elaboración propia

4.2.3.1 Tecnologías y componentes

- Servidor Bróker: Eclipse Mosquitto es un agente de mensajes de código abierto que implementa el protocolo MQTT. El protocolo MQTT proporciona un método ligero para realizar mensajes mediante un modelo de publicación / suscripción. Esto lo hace adecuado para la mensajería IIoT, en sensores de baja potencia, dispositivos móviles como teléfonos, computadoras integradas o microcontroladores. MQTT es uno de los protocolos más utilizados actualmente en IoT. Mosquitto ha sido desarrollado por la Eclipse Foundation la cual conforma una comunidad global de personas y organizaciones creando un entorno maduro y estable ofreciendo un software de código abierto creado por medio de la colaboración y la innovación (Mosquitto.org, 2020).

- Lenguaje desarrollo: El software de desarrollo utilizado es Python (Python.org, 2020). Python es uno de los lenguaje de programación con mayor crecimiento en la actualidad, tiene una baja curva de aprendizaje para su uso, dispone de un gran número de librerías open source y una gran comunicad de desarrolladores (Millman & Aivazis, 2011). Adicionalmente es el lenguaje nativo en el que está desarrollado el actual ERP de Odoo por lo puede facilitar su integración. Se ha utilizado Pycharm 2020.3 Community como entorno de desarrollo al ser compatible con Windows. Se han desarrollado dos clientes MQTT con el fin de permitir la ingesta de datos en la base de datos y de implementar un MES capaz de recibir las órdenes de producción del ERP y transmitirlo al resto de clientes del sistema. Además, se han desarrollado: un cliente con Eclipse Paho que permite utilizar los dispositivos móviles para escanear códigos QR, enviar e-mails y varios clientes para simular el resto de principales componentes del sistema.

- Base de datos: Gestor de base de datos relacional de código abierto gratuito (PostgreSQL.org, 2020). Se ha seleccionado esta base de datos al ser compatible con Power BI el cual incorpora un conector para esta base de datos de manera directa. Además, es la base de datos usada por el ERP de Odoo, lo que podría facilitar la integración o unificación de ambas bases de datos, así como la interoperabilidad de éstas con otros sistemas en el futuro. Adicionalmente, dispone de un conector con Python que facilita su conectividad.

- Gateway: FATBOX G3 Edge IIoT Gateway (Edge IIoT Gateway, 2020). Soporta procesamiento de datos, es programable en Python y ofrece conectividad Wifi, Ethernet y estándares de comunicación industrial como Modbus y el protocolo MQTT.

- Sensor Humedad Línea: Dos sensores de humedad NDC Series 9 para medida en línea compatible con bentonita cálcica. Señal- 4-20mA conectado a autómata NDC (NDC Technologies, 2020).
- Sensor de temperatura y humedad: Permite estudiar como las condiciones ambientales afectan al proceso. Conectividad Wifi y soporta MQTT (NDC Technologies, 2020).
- Autómatas secado, molienda: Autómata industrial CX9020 Beckhoff para el control del orno de secado y el molino. Soporta MQTT, industrial Ethernet y otros buses industriales. Controlan el arranque-paro de las máquinas y la velocidad de máquina, además permitirán conectar al sistema elementos de campo que no soportan MQTT como es el caso de los sensores de humedad (Beckhoff Cx9020, 2020).
- Sensor Temperatura de Máquina: Sensor PT-100. Conectado a autómata mediante cableado (Jumo Temperature sensor, 2020).
- Variador de frecuencia de máquina: Permitirá modificar la velocidad de máquina y optimizar su consumo y rendimiento de proceso (ABB ACS380, 2020).
- ERP Odoo. ERP existente actualmente.
- Microsoft Power BI: Herramienta de visualización de datos existente actualmente.

4.3 Evaluación y prueba de concepto (POC)

Finalmente, y dado que la implementación del sistema propuesto se escapa al alcance de este TFM, se ha comprobado la viabilidad del sistema en una prueba de concepto.

- Objetivos: Validar la viabilidad la plataforma IIoT propuesta como solución en el aparatado 4.2
- Alcance: El alcance queda limitado a implementar los componentes claves de la plataforma:
    - Servidor Bróker
    - Agente Base de Datos y base de datos
    - Agente MES
    - Interfaz operarios
- Configuración: Se simularán mediante clientes MQTT:
    - Sensores y Autómatas: En el caso de los elementos de campo se han simulado sus señales de funcionamiento con varios clientes MQTT.
    - Para probar la conectividad con el sistema se ha creado una cuenta en el ERP Odoo, creando diferentes productos y órdenes de producción.
    - Power BI: La base de datos se ha conectado a una cuenta de Power BI para poder comprobar la compatibilidad del software con la base de datos.

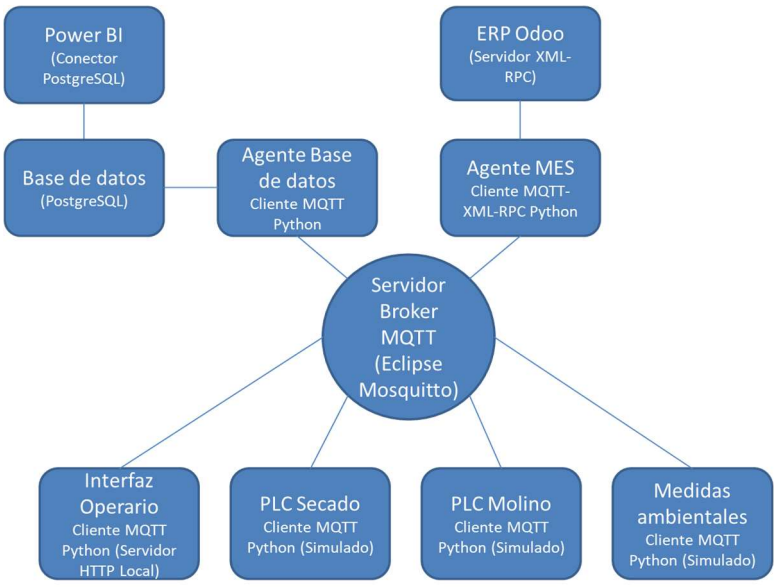


Figura 9. Esquema prueba de concepto

4.3.1. POC

4.3.2.1 Servidor bróker

Se ha instalado un servidor mosquitto-2.0.3 (Mosquitto.org, 2020) y montado sobre un equipo Windows 10. El servidor bróker transmite toda la información publicada por los distintos clientes de la plataforma y cada cliente recibe la información a la que se ha suscrito específicamente.

4.3.2.2 Base de datos y Agente base de datos

Se ha creado una base de datos en PostgreSQL. Esta base de datos contiene una tabla por cada elemento del sistema.

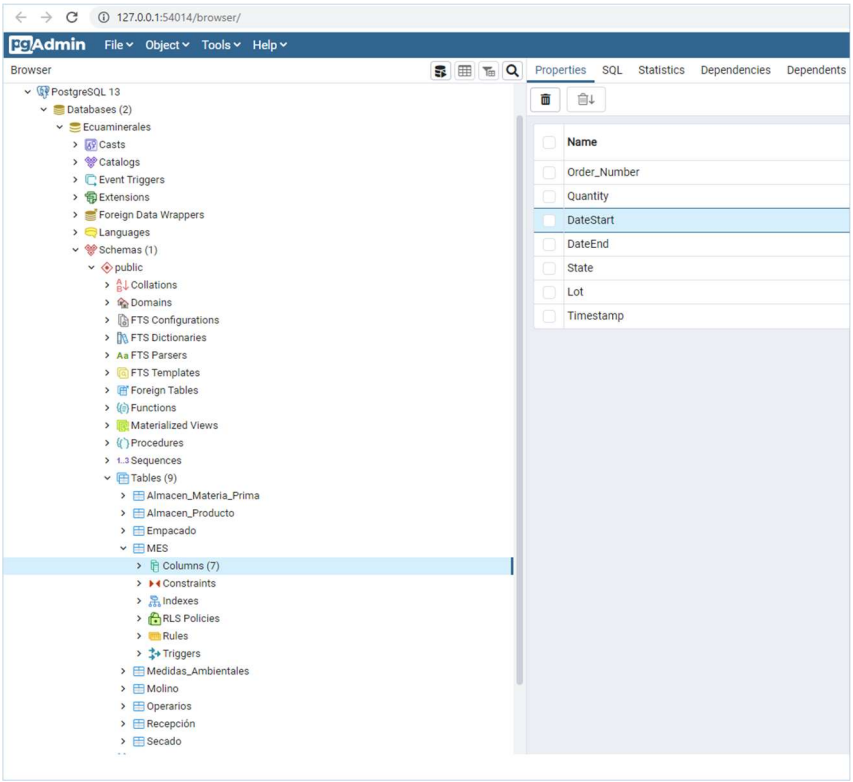


Figura 10. Tablas base de datos PostgreSQL

El “Agente base de datos” actúa como un cliente MQTT el cual recibe todos los mensajes del sistema. La función del agente base de datos es recoger la información del sistema e insertarla en la base de datos en el campo y formato adecuado. Para implementar el cliente MQTT se ha utilizado la librería paho-mqtt 1.5.1 (Roger Light, 2020) que permite crear un cliente para conectarse a un agente MQTT, publicar mensajes, suscribirse a temas y recibir mensajes publicados. La conexión con la base de datos se ha realizado utilizando la librería Psycopg (Gregorio, 2020) que funciona como un adaptador de base de datos PostgreSQL para el lenguaje de programación Python, el cual permite realizar consultas y editar la base de datos (ver Anexo I).

4.3.2.3 Agente MES y ERP

El agente MES es el encargado de conectar con el ERP y enviar las órdenes de producción a las distintas estaciones de trabajo, además envía de nuevo al ERP el estado actual de la orden de trabajo (Confirmed, Progress, Done). De esta manera dota al sistema de la capacidad de realizar la trazabilidad de los productos.

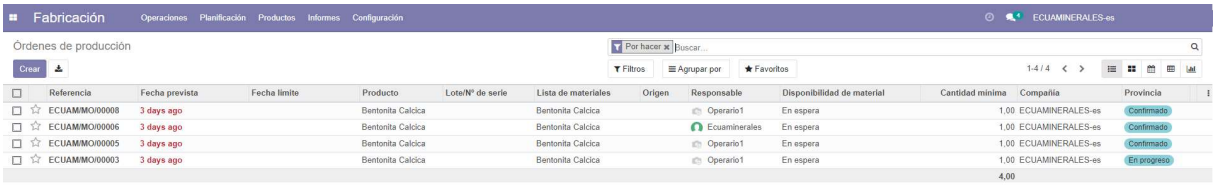


Figura 11. Órdenes de producción en ERP Odoo

Para su implementación se ha utilizado de nuevo la librería la librería paho-mqtt 1.5.1 para el cliente MQTT. La conexión al ERP de Odoo se ha establecido mediante la librería XML-RPC de Pyhton (Na, 2020). XML-RPC es un protocolo que permite llamar a un procedimiento remoto (RPC) que utiliza XML para codificar sus llamadas y utiliza HTTP como mecanismo de transporte (ver Anexo II).

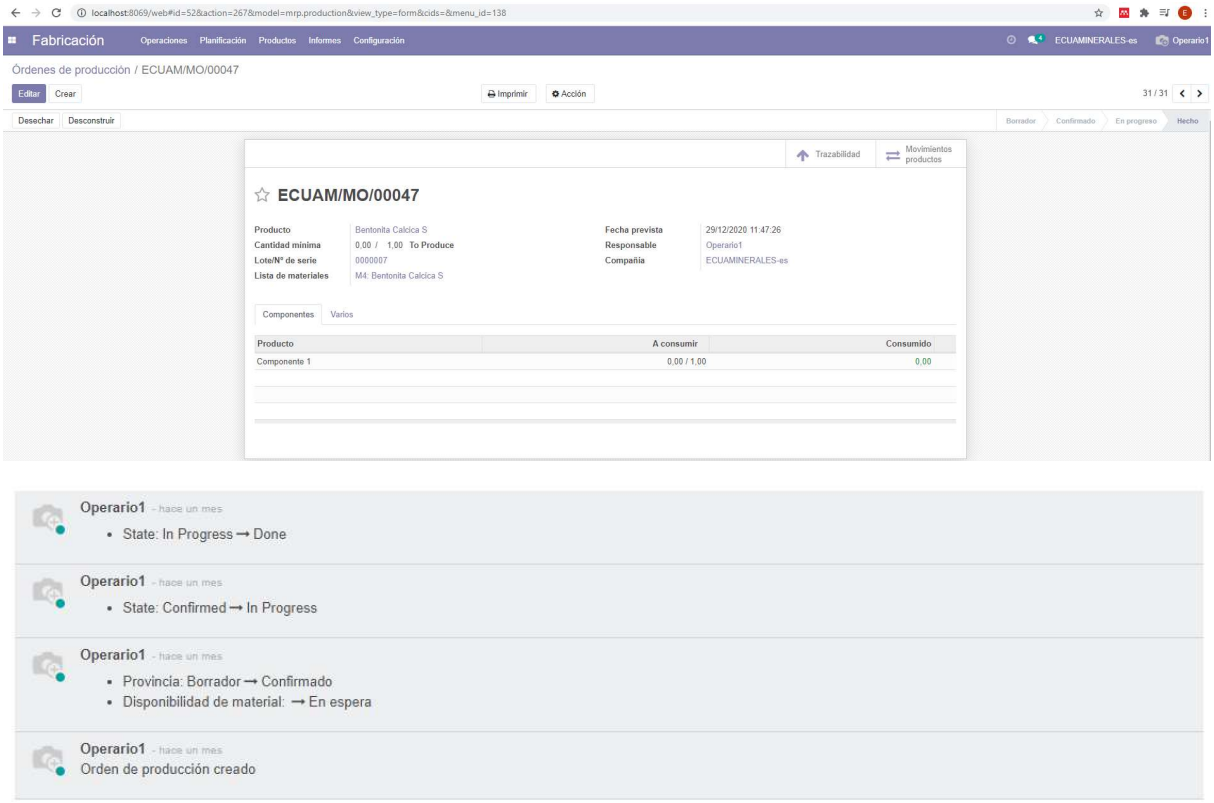


Figura 12. Detalle progreso orden de producción en ERP Odoo

4.3.2.4 Interfaz operarios

La interfaz de operarios utiliza un servidor web HTML. Esta interfaz se ha desarrollado en Python mediante la librería Flask (Ronacher, 2020), un framework ligero que permite desarrollar aplicaciones WSGI (Web Server Gateway Interface) de manera sencilla. Adicionalmente, se ha utilizado Flask-MQTT (S.Lehmann, 2020) que es una extensión de Flask para la implementación de clientes MQTT basada en Eclipse Paho. La interfaz Web ha sido desarrollada en HTML y permite a cualquier operario o personal de administración acceder a los datos proporcionados por el sistema MES en las distintas etapas del proceso. Mediante esta interfaz se puede observar las órdenes de producción, confirmar la realización de dichas órdenes, realizar fotografías y escanear códigos QR (ver Anexo III).

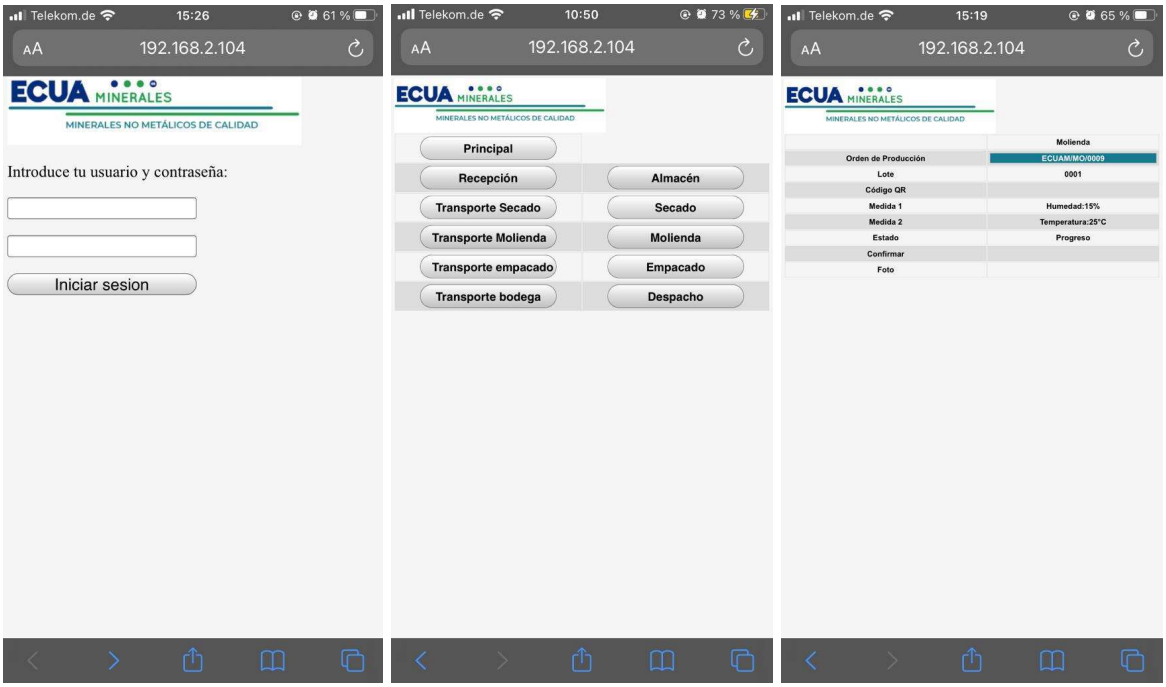
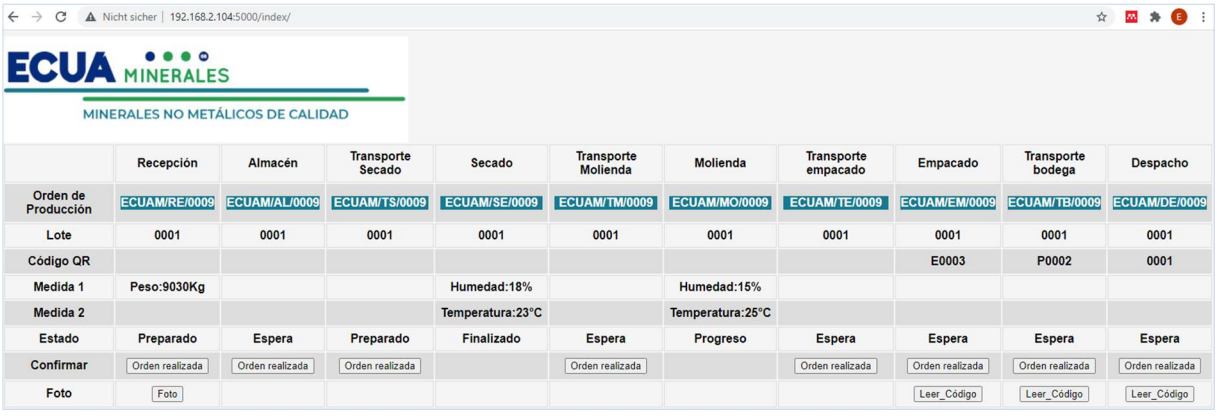


Figura 13. Vista pantallas gestión de acceso a usuarios, menú principal y Molienda desde móvil



	Recepción	Almacén	Transporte Secado	Secado	Transporte Molienda	Molienda	Transporte empacado	Empacado	Transporte bodega	Despacho
Orden de Producción	ECUAM/RE/0009	ECUAM/AL/0009	ECUAM/TS/0009	ECUAM/SE/0009	ECUAM/TM/0009	ECUAM/MO/0009	ECUAM/TE/0009	ECUAM/EM/0009	ECUAM/TB/0009	ECUAM/DE/0009
Lote	0001	0001	0001	0001	0001	0001	0001	0001	0001	0001
Código QR								E0003	P0002	0001
Medida 1	Peso:9030Kg			Humedad:18%		Humedad:15%				
Medida 2				Temperatura:23°C		Temperatura:25°C				
Estado	Preparado	Espera	Preparado	Finalizado	Espera	Progreso	Espera	Espera	Espera	Espera
Confirmar	Orden realizada	Orden realizada	Orden realizada		Orden realizada		Orden realizada	Orden realizada	Orden realizada	Orden realizada
Foto	Foto							Leer_Código	Leer_Código	Leer_Código

Figura 14. Vista menú principal estaciones desde PC

4.3.2.5 PLC Secado y Molino

Para simular los dos autómatas de las estaciones de secado y molienda se han implementado 2 clientes MQTT los cuales interactúan con el sistema MES y el agente base de datos a través del servidor bróker. Ambos clientes esperan una orden de producción del sistema MES. Una vez recibida la orden de producción cambian su estado de máquina de “OFF” a “ON” y publican su estado (“Progress”). En este estado simulan la variación de la temperatura y el descenso de la humedad periódicamente mediante una sencilla rutina basada en la generación de números aleatorios. La simulación del proceso de producción finaliza una vez la humedad desciende por debajo de un determinado umbral en el valor de la humedad. Una vez alcanzado este umbral la máquina pasa al estado “OFF” y publica un mensaje para el sistema MES con la orden de producción y el estado (“Done”) (ver Anexo IV y V).

4.3.2.6 Sensor medidas ambientales

Para simular el sensor de medidas ambientales se ha implementado un cliente MQTT que genera medidas aleatorias de temperatura y humedad. Estas medidas son publicadas en un mensaje que es recibido por el agente base de datos que se encarga de registrarlas en la base de datos (ver Anexo VI).

4.3.2.7 Conexión Power BI

Los datos recopilados por la plataforma en la base de datos PostgreSQL podrán visualizarse en Microsoft Power BI (ver Figura 15).

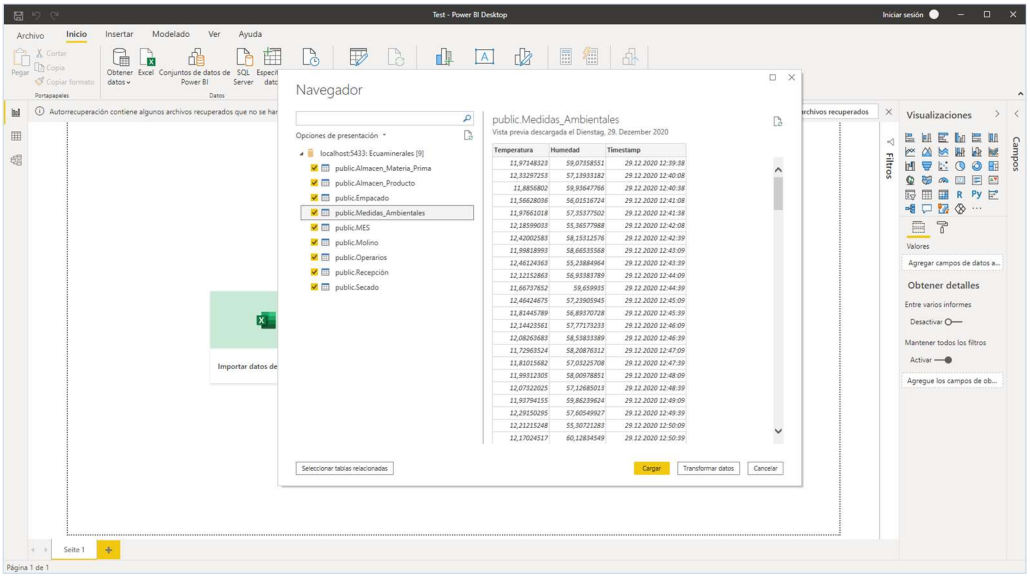


Figura 15. Conexión de la base de datos en Power BI

En esta prueba de concepto se ha comprobado que los datos de la simulación realizada durante la prueba de concepto pueden visualizarse de manera correcta. Las herramientas de visualización son un elemento fundamental en la industria 4.0. Esta conectividad permitirá integrar analizar y presentar los datos recopilados por la plataforma para detectar problemas, aumentar la eficiencia y, en definitiva, obtener con mayor facilidad conocimiento del proceso (ver Figura 16).

Order_Number	Quantity	DateStart	DateEnd	State	Lot	Timestamp
ECUAM/MO/00025	1	28.12.2020 12:18:48	28.12.2020 13:18:48	done	0000007	28.12.2020 18:33:03
ECUAM/MO/00025	1	28.12.2020 12:18:48	28.12.2020 13:18:48	progress	0000007	28.12.2020 18:32:03
ECUAM/MO/00025	1	28.12.2020 12:18:48	28.12.2020 13:18:48	confirmed	0000007	28.12.2020 18:31:00

Figura 16. Visualización de datos en Power BI de los datos registrados en la base de datos de la tabla MES

5. Conclusiones y trabajo futuro

5.1. Conclusiones

Ecuaminerales GB es una PYME con recursos limitados y un bajo nivel de automatización. La aplicación de las tecnologías de la industria 4.0 pueden suponer una gran oportunidad para iniciar la digitalización y automatización de sus procesos. Este trabajo fin de máster propone una solución basada en una plataforma IIoT para realizar la trazabilidad y calidad de su línea de fabricación de minerales no metálicos. Los resultados a destacar una vez realizado este proyecto fin de máster son los siguientes:

- Se ha realizado una revisión del estado del arte de la Industria 4.0 en el que se revela el potencial de estas tecnologías para provocar un cambio de paradigma en el desarrollo de sistemas de ejecución de la fabricación.
- Se ha analizado el proceso de fabricación de minerales no metálicos de la empresa Ecuaminerales GB.
- Se ha aplicado la metodología IIRA para de elaborar una arquitectura que, mediante un sistema CPS e IIoT, permita realizar la trazabilidad del proceso y capturar medidas para el control de la calidad del proceso estudiado.
- Se han seleccionado los componentes tecnológicos principales para la implementación de la solución propuesta.
- Se ha comprobado la viabilidad de la solución propuesta mediante una implementación open-source basada en Python y Eclipse Mosquitto.

Considerando los resultados obtenidos, se puede afirmar que se han los alcanzado los objetivos generales y específicos planteados por este proyecto fin de máster:

- Se han conseguido identificar las necesidades del proceso mediante el análisis de éste y gracias a la aplicación de la metodología IIRA.
- Se ha diseñado la arquitectura de una plataforma que permite realizar la trazabilidad del proceso y recopilar mediciones del proceso para el control de la calidad de los productos.
- Han sido seleccionados los componentes tecnológicos para la implementación real de la arquitectura propuesta.
- Se ha realizado una prueba de concepto en la cual se han implementado y probado los componentes claves de la plataforma.

El estudio del estado del arte revela como las tecnologías de la industria 4.0 aún no se utilizan de manera predominante en la trazabilidad y control de calidad de los procesos. La mayoría de las aplicaciones comerciales son productos de grandes multinacionales que ofrecen la integración de tecnologías propias basadas en licencias, aplicaciones, que no están diseñadas para pequeñas empresas. Sin embargo, existen numerosas experiencias en la literatura científica que integran las tecnologías habilitadoras de la industria 4.0 de manera sencilla y exitosa.

A lo largo de la realización del proyecto se ha comprobado la gran utilidad de aplicar un marco metodológico como el de IIRA. Con la aplicación de esta metodología se ha diseñado una arquitectura que no solo satisface las necesidades de la empresa, sino que también permite su ampliación o modificación gracias a su visión de alto nivel.

La revolución de las tecnologías habilitadoras de la industria 4.0 permite hoy en día desplegar soluciones baratas, seguras y fiables que son accesibles para pequeñas y grandes empresas. Esto es posible debido a la disminución de precios de los componentes y a la presencia de plataformas y comunidades de software open-source que permiten implementar soluciones flexibles y adaptadas a cada entorno. El uso de plataformas libres de licencia al que dan soporte comunidades colaborativas son una herramienta que aumenta el potencial de la ciencia y la tecnología. Estas plataformas han sido clave para la implementación de la plataforma IIoT desarrollada en este trabajo fin de Máster, en un corto espacio de tiempo, con pocos recursos y conocimientos limitados.

Finalmente, este proyecto muestra como impulsar la transformación digital de Ecuaminerales GB mediante la aplicación de tecnologías habilitadoras de la industria 4.0, lo que le permitirá mejorar sus procesos productivos, aumentando el conocimiento del proceso y disminuyendo los costes de producción.

5.2. Líneas de trabajo futuro

Los procesos de transformación digital en la industria deben integrarse en las empresas como un proceso continuo impulsado por la alta dirección e incorporarse a la cultura empresarial en todos los niveles de la empresa. Este trabajo propone integrar un CPS por medio de una plataforma IIoT para realizar la trazabilidad del proceso y la calidad de los productos. Las líneas de trabajo futuro podrían alinearse con este propósito incorporando nuevas funcionalidades al sistema de ejecución de la fabricación (agente MES) para comprobar la flexibilidad y la capacidad de la plataforma y del agente MES. Adicionalmente, una vez dispuesta la plataforma que permite recoger la información del proceso y su automatización, sería de gran utilidad realizar el análisis de los datos recogidos. Estos datos tienen el potencial de convertirse en conocimiento y en un activo para la empresa. Para llevarlo a cabo, es necesario analizar los datos utilizando las herramientas actuales de machine learning, mantenimiento predictivo o Big data, lo que sería una interesante línea de trabajo futuro de este proyecto fin de Máster. El estudio de la relación entre las medidas de temperatura y humedad en línea junto con las ambientales para conocer si tienen algún efecto en la productividad, sería un ejemplo de ello.

Finalmente, y de nuevo utilizando la plataforma como base, se propone la integración de un sistema de transporte entre estaciones que permita realizar una producción continua sin la intervención de mano de obra en el proceso productivo.

Las plataformas IIoT tienen un gran potencial para habilitar la integración de CPS. Estas líneas de investigación permitirían avanzar más en el estudio de los sistemas ciber físicos en ecosistemas IIoT para PYMES.

6. Bibliografía

ABB ACS380. (2020). *ACS380 - Convertidores de frecuencia para maquinaria - Construya sus beneficios con control y flexibilidad (Convertidores de frecuencia de baja tensión de CA) | ABB*. https://new.abb.com/drives/es/convertidores-baja-tension-ca/maquinaria/acs380

Almada-Lobo, F. (2016). The Industry 4.0 revolution and the future of Manufacturing Execution Systems (MES). *Journal of Innovation Management*, *3*(4), 16–21. https://doi.org/10.24840/2183-0606_003.004_0003

Beckhoff Cx9020. (2020). *Beckhoff Cx9020*. https://www.beckhoff.com/es-es/products/ipc/embedded-pcs/cx9020-arm-cortex-a8/cx9020.html

Choi, H., Song, J., & Yi, K. (2018). Brightics-IoT: Towards Effective Industrial IoT Platforms for Connected Smart Factories. *Proceedings - 2018 IEEE International Conference on Industrial Internet, ICII 2018*, 146–152. https://doi.org/10.1109/ICII.2018.00024

De Ugarte, B. S., Artiba, A., & Pellerin, R. (2009). Manufacturing execution system - A literature review. *Production Planning and Control*, *20*(6), 525–539. https://doi.org/10.1080/09537280902938613

Edge IIoT Gateway. (2020). *Edge IIoT Gateway*. https://www.amplified.com.au/modbus-iot-connectivity?gclid=EAIaIQobChMIm__PpPTc7QIVh-3tCh0A9AKGEAAYASAAEgLLNvD_BwE

Gartner. (2019). *Magic Quadrant for Industrial IoT Platforms*. https://www.gartner.com/doc/reprints?id=1-6NRJDE7&ct=190510&st=sb

Graube, M., Hensel, S., Iatrou, C., & Urbas, L. (2017). Information models in OPC UA and their advantages and disadvantages. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 1–8. https://doi.org/10.1109/ETFA.2017.8247691

Gregorio, F. Di. (2020). *psycopg2 · PyPI*. https://pypi.org/project/psycopg2/

Grgić, K., Špeh, I., & Hedi, I. (2016). A web-based IoT solution for monitoring data using MQTT protocol. *Proceedings of 2016 International Conference on Smart Systems and Technologies, SST 2016*, 249–253. https://doi.org/10.1109/SST.2016.7765668

Guth, J., Breitenbucher, U., Falkenthal, M., Leymann, F., & Reinfurt, L. (2017, March 6). Comparison of IoT platform architectures: A field study based on a reference architecture. *2016 Cloudification of the Internet of Things, CIoT 2016*. https://doi.org/10.1109/CIOT.2016.7872918

Hejazi, H., Rajab, H., Cinkler, T., & Lengyel, L. (2018). Survey of platforms for massive IoT. *2018 IEEE International Conference on Future IoT Technologies, Future IoT 2018*, *2018*-

January, 1–8. https://doi.org/10.1109/FIOT.2018.8325598

Iarovyi, S., Mohammed, W. M., Lobov, A., Ferrer, B. R., & Lastra, J. L. M. (2016). Cyber-Physical Systems for Open-Knowledge-Driven Manufacturing Execution Systems. *Proceedings of the IEEE*, *104*(5), 1142–1154. https://doi.org/10.1109/JPROC.2015.2509498

Industrial Internet Consortium (IIC). (2015). *The Industrial Internet Reference Architecture v 1.9*. https://www.iiconsortium.org/IIRA.htm

Jumo Temperature sensor. (2020). *Jumo Temperature sensor Sensor type Pt100 Temperature reading range-50 up to 400 °C Sensor diameter 6 mm | Conrad.com*. https://www.conrad.com/p/jumo-temperature-sensor-sensor-type-pt100-temperature-reading-range-50-up-to-400-c-sensor-diameter-6-mm-120387

Lee, J., Bagheri, B., & Kao, H. A. (2015). A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, *3*, 18–23. https://doi.org/10.1016/j.mfglet.2014.12.001

Lin et al. (2017). *The Industrial Internet of Things Volume G1: Reference Architecture*.

Liu, B., Zhang, Y., Zhang, G., & Zheng, P. (2019). Edge-cloud orchestration driven industrial smart product-service systems solution design based on CPS and IIoT. *Advanced Engineering Informatics*, *42*, 100984. https://doi.org/10.1016/j.aei.2019.100984

Luo, Z., Hong, S., Lu, R., Li, Y., Zhang, X., Kim, J., Park, T., Zheng, M., & Liang, W. (2017). OPC UA-Based Smart Manufacturing: System Architecture, Implementation, and Execution. *Proceedings - 2017 5th International Conference on Enterprise Systems: Industrial Digitalization by Enterprise Systems, ES 2017*, 281–286. https://doi.org/10.1109/ES.2017.53

Marjani, M., Nasaruddin, F., Gani, A., Karim, A., Hashem, I. A. T., Siddiqa, A., & Yaqoob, I. (2017). Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges. *IEEE Access*, *5*, 5247–5261. https://doi.org/10.1109/ACCESS.2017.2689040

Millman, K. J., & Aivazis, M. (2011). Python for scientists and engineers. In *Computing in Science and Engineering* (Vol. 13, Issue 2, pp. 9–12). https://doi.org/10.1109/MCSE.2011.36

Mosquitto.org. (2020). *Eclipse Mosquitto*. https://mosquitto.org/

Mourtzis, D., Vlachou, E., & Milas, N. (2016). Industrial Big Data as a Result of IoT Adoption in Manufacturing. *Procedia CIRP*, *55*, 290–295. https://doi.org/10.1016/j.procir.2016.07.038

Na, D. (2020). *XML-RPC client access. Python 3.9.1*. https://docs.python.org/3/library/xmlrpc.client.html#module-xmlrpc.client

Naik, N. (2017, October 26). Choice of effective messaging protocols for IoT systems: MQTT,

CoAP, AMQP and HTTP. *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*. https://doi.org/10.1109/SysEng.2017.8088251

NDC Technologies. (2020). *Serie 9 sensor humedad en línea NDC Technologies*. https://scl.es/blog/nueva-serie-9-de-sensores-en-linea-ndc-technologies/

Park, J. (2015). Evaluating a mobile data-collection system for production information in SMEs. *Computers in Industry*, *68*, 53–64. https://doi.org/10.1016/j.compind.2014.12.006

Pelino, M., & Miller, P. (2019). *The Forrester Wave™: Industrial IoT Software Platforms, Q4 2019*.

PostgreSQL.org. (2020). *PostgreSQL: The world's most advanced open source database*. https://www.postgresql.org/

Python.org. (2020). *Python.org*. https://www.python.org/

Rick Franzosa. (2019). *Magic Quadrant for Manufacturing Execution Systems*. https://www.gartner.com/doc/reprints?id=1-1XQ9L4I4&ct=191105&st=sb&elqTrackId=98c05c84a2cf4a3e8e12cbd232a4a5a1&elqaid=87365&elqat=2&source=:ow:lp:cpo::

Roger Light. (2020). *paho-mqtt · PyPI*. https://pypi.org/project/paho-mqtt/

Ronacher, A. (2020). *Flask*. https://pypi.org/project/Flask/

S.Lehmann. (2020). *Flask-MQTT*. https://pypi.org/project/Flask-MQTT/

Thangavel, D., Ma, X., Valera, A., Tan, H. X., & Tan, C. K. Y. (2014). Performance evaluation of MQTT and CoAP via a common middleware. *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*. https://doi.org/10.1109/ISSNIP.2014.6827678

Urbina Coronado, P. D., Lynn, R., Louhichi, W., Parto, M., Wescoat, E., & Kurfess, T. (2018). Part data integration in the Shop Floor Digital Twin: Mobile and cloud technologies to enable a manufacturing execution system. *Journal of Manufacturing Systems*, *48*, 25–33. https://doi.org/10.1016/j.jmsy.2018.02.002

Wan, J., Chen, B., Imran, M., Tao, F., Li, D., Liu, C., & Ahmad, S. (2018). Toward dynamic resources management for IoT-based manufacturing. *IEEE Communications Magazine*, *56*(2), 52–59. https://doi.org/10.1109/MCOM.2018.1700629

Zhang, Y., Guo, Z., Lv, J., & Liu, Y. (2018). A Framework for Smart Production-Logistics Systems Based on CPS and Industrial IoT. *IEEE Transactions on Industrial Informatics*, *14*(9), 4019–4032. https://doi.org/10.1109/TII.2018.2845683

Anexo I. Agente base de datos – Main

El agente base de datos es el encargado de realizar la ingesta de los datos de todos los elementos de la plataforma, ordenarlos y guardarlos en la base de datos. Esta es la sección de código principal donde se crean las suscripciones al servidor Bróker y se gestionan los mensajes recibidos para integrarlos en la base de datos.

```
# importación de librerías
import datetime
import time
import paho.mqtt.client as paho
import PostgresSQL_Connector
import queue

# Ubicación bróker
broker = "localhost"

# Conector Base de datos
DB = PostgresSQL_Connector.Insert()

# Cola en FIFO mensajes
messages = queue.Queue()

# Función que descodifica el mensaje
def decode_message(messages):
    while not messages.empty():
        mensaje = messages.get()
        if mensaje.topic == 'MES':
            DB.mes(mensaje.payload.decode("utf-8"))
        if mensaje.topic == 'PLC/Secado/Estado':
            DB.secado(mensaje.payload.decode("utf-8"))
        if mensaje.topic == 'PLC/Molino/Estado':
            DB.molino(mensaje.payload.decode("utf-8"))
        if mensaje.topic == 'MedidasAmbientales':
            DB.medidas_ambientales(mensaje.payload.decode("utf-8"))
        if mensaje.topic == 'Operario':
            DB.operario(mensaje.payload)

# Función que inserta los mensajes en una FIFO
def on_message(client, userdata, message):
    messages.put(message)
    time.sleep(1)

# Crea un cliente MQTT
client = paho.Client("AgentDB")

# Asocia el cliente a la función on_message
client.on_message = on_message

# Se crea conexión con bróker
print("connecting to bróker ", broker)
client.connect(broker)

client.loop_start() # # Comienza el bucle del proceso para recibir mensajes
print("Suscripción ")
client.subscribe("#")  # Suscripción a todos los topics
```

```
while True: # Rutina periódica para descodificar los mensajes e insertarlos
en la base de datos
    Timestamp = datetime.datetime.now().isoformat(' ', 'seconds')
    decode_message(messages)
    print('esperando')
    time.sleep(5)


client.disconnect()  # desconecta el cliente
client.loop_stop()  # detiene el hilo MQTT

# Cierra conexión a la base de datos
PostgresSQL_Connector.cursor.close()
PostgresSQL_Connector.conexion.close()
```

Anexo I.I Agente Base de datos – PostgreSQL

En esta sección de código se realiza la conexión a la base de datos y se realizan las consultas de escritura en la PostgreSQL.

```
# importación de librerías
import psycopg2
import datetime
from PIL import Image

# Configuración conexión ERP
conexion = psycopg2.connect(
    host="localhost",
    database="Ecuaminerales",
    user="postgres",
    password="******_01",
    port=5433)

# Se consolidan siempre los datos en la base de datos tras escribir
conexion.autocommit = True

# Se crea el cursor con el objeto conexión
cursor = conexion.cursor()


class Insert():
    def mes(self, Payload): # Escritura en tabla MES
        timestamp = datetime.datetime.now().isoformat(' ', 'seconds')
        # query al MES para guardar los datos en la base de datos
        insert_query = '''INSERT INTO public."MES"("Order_Number",
"Quantity", "DateStart", "DateEnd", "State", "Lot", "Timestamp")
                VALUES(%s,%s,%s,%s,%s,%s,%s);'''

        Payload_Array = Payload.split(sep=',', maxsplit=5)
        Order_Number = Payload_Array[0]
        Quantity = Payload_Array[1]
        DateStart = Payload_Array[2]
        DateEnd = Payload_Array[3]
        State = Payload_Array[4]
        Lot = Payload_Array[5]
```

```
        cursor.execute(insert_query, (Order_Number, Quantity, DateStart,
DateEnd, State, Lot, timestamp))
        print('MES escribe en DB ', Order_Number, Quantity, DateStart,
DateEnd, State, Lot, timestamp)


    def secado(self,Payload): # Escritura en tabla Secado
        timestamp = datetime.datetime.now().isoformat(' ', 'seconds')
        Payload_Array = Payload.split(sep=',', maxsplit=4)
        Order_Number = Payload_Array[0]
        Estado = Payload_Array[1]
        Lote = Payload_Array[2]
        Temperatura = Payload_Array[3]
        Humedad = Payload_Array[4]

        # Ejecutamos una consulta
        insert_query = '''INSERT INTO public."Secado"("Order_Number",
"Estado", "Lote", "Temperatura", "Humedad", "Timestamp")
                VALUES(%s,%s,%s,%s,%s,%s);'''

        cursor.execute(insert_query, (Order_Number, Estado, Lote,
Temperatura, Humedad, timestamp))
        print('MES escribe en DB ', Order_Number, Estado, Lote,
Temperatura, Humedad, timestamp)

    def molino(self,Payload): # Escritura en tabla Molino
        timestamp = datetime.datetime.now().isoformat(' ', 'seconds')
        Payload_Array = Payload.split(sep=',', maxsplit=4)
        Order_Number = Payload_Array[0]
        Estado = Payload_Array[1]
        Lote = Payload_Array[2]
        Temperatura = Payload_Array[3]
        Humedad = Payload_Array[4]

        # Ejecutamos una consulta
        insert_query = '''INSERT INTO public."Molino"("Order_Number",
"Estado", "Lote", "Temperatura", "Humedad", "Timestamp")
                VALUES(%s,%s,%s,%s,%s,%s);'''

        cursor.execute(insert_query, (Order_Number, Estado, Lote,
Temperatura, Humedad, timestamp))
        print('MES escribe en DB ', Order_Number, Estado, Lote,
Temperatura, Humedad, timestamp)

    def medidas_ambientales(self,Payload): # Escritura en tabla
medidas_ambientales
        timestamp = datetime.datetime.now().isoformat(' ', 'seconds')
        Payload_Array = Payload.split(sep=',', maxsplit=1)
        Temperatura = Payload_Array[0]
        Humedad = Payload_Array[1]

        # Ejecutamos una consulta
        insert_query = '''INSERT INTO
public."Medidas_Ambientales"("Temperatura", "Humedad", "Timestamp")
                VALUES(%s,%s,%s);'''

        cursor.execute(insert_query, (Temperatura, Humedad, timestamp))
        print('MES escribe en  Medidas ambientales DB ', Temperatura,
Humedad, timestamp)
```

Anexo II. Agente MES

Este cliente MQTT se encarga de realizar la conexión con el ERP de Odoo. El cliente consulta de manera periódica si existe una orden preparada para producir (Confirmed). Esta orden es enviada a la estación y una vez comenzada la orden de producción la estación devuelve al MES la confirmación de que está produciendo. Posteriormente, el MES actualiza en el ERP el estado de la estación (Progress). Una vez finalizada la orden de trabajo la estación envía la confirmación de que ha sido finalizada al MES y este a su vez actualiza el ERP con el estado de orden finalizada (Done).

```
# Importación de librerías
import time
import paho.mqtt.client as paho
import Odoo_Conexion
import queue
import os

# Comienza ejecución del broker
broker = "localhost"  # C:\Program Files\mosquitto
'''Start BRÓKER'''
os.startfile("C:\Program Files\mosquitto\mosquitto.exe")
time.sleep(5)

TypeBool = False

# Cola de FIFO de mensajes
messages = queue.Queue()

# Función para descodificar los mensajes MQTT
def decode_message(messages):
    mensaje_on = False
    order_number = ''
    state = ''
    if not messages.empty():
        mensaje_on = True
        mensaje = messages.get()
        message_array = mensaje.payload.decode("utf-8").split(sep=',',
maxsplit=4)
        order_number = message_array[0]
        state = message_array[1]
        #Lote = payload_array[2]
        #Temperatura = payload_array[3]
        #Humedad = payload_array[4]
    return mensaje_on, order_number, state

# Función que guarda en una FIFO los mensajes
def on_message(client, userdata, message):
    messages.put(message)
    print("received message =", str(message.payload.decode("utf-8")))
    time.sleep(1)

# Creación del cliente MQTT
client = paho.Client("MES")
# Asociación del cliente con la función on_message
client.on_message = on_message
```

```
# Conexión al servidor bróker
print("connecting to bróker ", broker)
client.connect(broker)  #
client.loop_start()  # Comienza el bucle del proceso para recibir mensajes

# Suscripción a los topics
print("subscribing to PLC")
client.subscribe("PLC/+/Estado")  # subscribe

# Creación clase ERP Odoo
Odoo = Odoo_Conexion.Production()

while True: # Rutina cíclica principal
    Timestamp = Odoo_Conexion.datetime.now().isoformat(' ', 'seconds')
    mensaje_nuevo = decode_message(messages)
    Orderconfirmed = Odoo.search_read('state', 'confirmed')

    if mensaje_nuevo[0]: # Si existe si se recibe un cambio de estado en
una estación
        Orders = Orderconfirmed + Odoo.search_read('state', 'progress')
        if len(Orders) > 0:  # Consulta si es necesario actualizar una
orden en Odoo
            for Index in range(0, (len(Orders))):
                if (mensaje_nuevo[1] == Orders[Index]['name']) and
(mensaje_nuevo[2] != Orders[Index]['state']):
                    Odoo.write(Orders[Index]['id'], mensaje_nuevo[2])
                    if type(Orders[Index]['lot_producing_id']) !=
type(TypeBool):
                        Lote = (Orders[Index]['lot_producing_id'][1])
                    else:
                        Lote = ''
                    Mensaje = (Orders[Index]['name']) + ',' +
str(Orders[Index]['product_qty']) + ',' + \
                              (Orders[Index]['date_planned_start']) + ',' +
\
                              (Orders[Index]['date_planned_finished']) +
',' + mensaje_nuevo[2] + ',' + Lote
                    client.publish("MES", Mensaje)  # publish
                    print('Actualiza orden de producción en Odoo', Mensaje)
    else:
        if len(Orderconfirmed) > 0:  # Consulta si existe una orden
preparada para producir
            for Index in range(0, (len(Orderconfirmed))):
                if (Orderconfirmed[Index]['state'] == 'confirmed') and \
                        (Timestamp >
(Orderconfirmed[Index]['date_planned_start'])):
                    if type(Orderconfirmed[Index]['lot_producing_id']) !=
type(TypeBool):
                        Lote =
(Orderconfirmed[Index]['lot_producing_id'][1])
                    else:
                        Lote = ''
                    Mensaje = (Orderconfirmed[Index]['name']) + ',' +
str(Orderconfirmed[Index]['product_qty']) + ',' + \
                              (Orderconfirmed[Index]['date_planned_start'])
+ ',' + \

(Orderconfirmed[Index]['date_planned_finished']) + ',' + 'confirmed' + ','
+ Lote
                    client.publish("MES", Mensaje)  # Publica el mensaje
```

```
                    print('Start order:', Mensaje)

    print('Esperando orden de producción')
    time.sleep(10)

client.disconnect()  # desconecta el cliente
client.loop_stop()  # detiene el bucle del proceso cliente
```

Anexo II.I Agente MES – Conexión con ERP

Esta sección de código implementa las funciones XLM-RPC para leer y escribir en el ERP de Odoo.

```
# importación de librerías
from datetime import datetime
import xmlrpc.client

# Configuración conexión servidor ERP Odoo
url = 'http://localhost:8069'
db = 'quique1883'
username = 'quique1883@hotmail.com'
password = '******_01'

# Creación conexión Odoo
common = xmlrpc.client.ServerProxy('{}/xmlrpc/2/common'.format(url))
uid = common.authenticate(db, username, password, {})
models = xmlrpc.client.ServerProxy('{}/xmlrpc/2/object'.format(url))

# Clase para realizar lecturas y escrituras en el módulo de fabricación
class Production():
    def search(self):# busca ids disponibles y devuelve todos los
parámetros
        model_name = 'mrp.production'
        function_name = 'search'
        orders_ids = models.execute_kw(db, uid, password, model_name,
function_name, [[]])
        return orders_ids

    def search_parameter(self, parameter, value):# busca ids disponibles y
devuelve un parámetro específico
        model_name = 'mrp.production'
        function_name = 'search'
        orders_ids = models.execute_kw(db, uid, password, model_name,
function_name, [[[parameter, '=', value]]])
        return orders_ids

    def search_read(self, parameter, value): # lee un parámetro con un
valor específico
        model_name = 'mrp.production'
        function_name = 'search_read'
        orders_ids = models.execute_kw(db, uid, password, model_name,
function_name, [[[parameter, '=', value]]])
        return orders_ids

    def read(self, order_id): # lee un id específico
        model_name = 'mrp.production'
        function_name = 'read'
```

```
        order = models.execute_kw(db, uid, password, model_name,
function_name, [[order_id]])
        return order

    def write(self, order_id, state): # actualiza el estado de una orden
        model_name = 'mrp.production'
        function_name = 'write'
        models.execute_kw(db, uid, password, model_name, function_name,
[[order_id], {'state': state}])
```

Anexo III. Interfaz operario

Este cliente MQTT utiliza Flask para desarrollar una interfaz web en HTML. Este cliente recoge la información de los distintos elementos para ser accesible desde un navegador desde cualquier dispositivo a través de un servidor web.

```
# importación librerías
import eventlet
import json
from flask import Flask, render_template, redirect, url_for, request,
Response
from flask_mqtt import Mqtt
from flask_table import Table, Col
from flask_socketio import SocketIO
from flask_bootstrap import Bootstrap
import random
import Scanner
import Payload_Decode
import E_mail

# Configuración Flask
eventlet.monkey_patch()
app = Flask(__name__, template_folder='templates')
app.config['SECRET'] = 'my secret key'
app.config['TEMPLATES_AUTO_RELOAD'] = True
app.config['MQTT_BROKER_URL'] = '127.0.0.1'
app.config['MQTT_BROKER_PORT'] = 1883
app.config['MQTT_USERNAME'] = ''
app.config['MQTT_PASSWORD'] = ''
app.config['MQTT_KEEPALIVE'] = 5
app.config['MQTT_TLS_ENABLED'] = False
app.config['MQTT_CLEAN_SESSION'] = True
username = 'admin'
password = 'password'

#Se crea conexión con bróker #MQTT
mqtt = Mqtt(app)
socketio = SocketIO(app)
bootstrap = Bootstrap(app)

# Se crea clase Scanner
video_stream = Scanner.VideoCamera()

# Variables gloabales
mensaje_Recepción = ''
mensaje_Almacen = ''
mensaje_Transporte_Secado = ''
mensaje_Secado = ''
mensaje_Transporte_Molienda = ''
mensaje_Molienda = ''
mensaje_Transporte_empacado = ''
mensaje_Empacado = ''
mensaje_Transporte_bodega = ''
mensaje_Despacho = ''
Frame = ""
Video = ""
Mensaje_ON =""
Template = "index.html"
```

```
# Creación de rutas a archivos HTML
@app.route('/')# Ruta inicial
def route():
    return render_template("login.html")

# Pantalla principal
@app.route('/index/', methods=['POST'])
def index():
    global mensaje, Mensaje_ON, FOTO
    global Template

    if Mensaje_ON == 'ON':
        Mensaje_ON = 'OFF'

    if request.method == "POST":
        print('post')
        attempted_capture = request.form.get('Captura_Albaran')
        attempted_QR = request.form.get('Leer_Código')
        print (attempted_QR)
        if attempted_capture:
            FOTO = 'NOK'
            return render_template("video.html")
        else:
            if attempted_QR:
                QR = video_stream.qrcode()
                Peso = random.uniform(49.8, 50.2)
                if QR != 'QR NO ENCONTRADO':
                    print('QR: ', QR)
                    return render_template(Template, QR=QR + ' , ' +
str(Peso) + 'Kg')
                    mqtt.publish("Operario", QR + ',' + str(Peso))
                else:
                    print('QR: ', QR)
                    return render_template(Template, QR=QR)
            else:
                if request.form.get('Volver') == 'Volver':
                    Template = "Menu.html"
                elif request.form.get('login') == 'Principal':
                    Template = "index.html"
                elif request.form.get('login') == 'Recepción':
                    Template = "index Recepción.html"
                elif request.form.get('login') == 'Almacén':
                    Template = "index Almacén.html"
                elif request.form.get('login') == 'Transporte Secado':
                    Template = "index Transporte Secado.html"
                elif request.form.get('login') == 'Secado':
                    Template = "index Secado.html"
                elif request.form.get('login') == 'Transporte Molienda':
                    Template = "index Transporte Molienda.html"
                elif request.form.get('login') == 'Molienda':
                    Template = "index Molienda.html"
                elif request.form.get('login') == 'Transporte empacado':
                    Template = "index Transporte empacado.html"
                elif request.form.get('login') == 'Empacado':
                    Template = "index Empacado.html"
                elif request.form.get('login') == 'Transporte bodega':
                    Template = "index Transporte bodega.html"
                elif request.form.get('login') == 'Despacho':
                    Template = "index Despacho.html"
                return render_template(Template, image_name="logo.png")
```

```
# Pantalla acceso ausuarios
@app.route('/login/', methods=['POST'])
def login():
    global username, password
    error = ''
    try:
        if request.method == "POST":
            attempted_username = request.form.get('username')
            attempted_password = request.form.get('password')
            attempted_login = request.form.get('login')

            if (attempted_login == 'Iniciar sesion') and
(attempted_username == username) and (attempted_password == password):
                print('welcome')
                return render_template("Menu.html")
            else:
                error = "Error de usuario o contraseña. Inténtelo de
nuevo."
                print(error)
                return render_template("login.html", error=error)
        else:
            return render_template("login.html")
    except Exception as e:
        print('Exception: ', e)
        return render_template("login.html", error=e)

# Pantalla captura Scanner
@app.route('/video/', methods=['GET', 'POST'])
def video():
        global Video, Frame
        global Template
        Video = Scanner.genFrame(video_stream)
        if request.method == "POST":
            attempted_capture = request.form.get('Captura')
            attempted_enviar = request.form.get('Enviar')
            attempted_volver = request.form.get('Volver')
            if attempted_volver == 'Volver':
                print('Volver')
                return render_template(Template)
            if attempted_enviar == 'Enviar':
                print('enviar')
                return render_template("video.html")
            if attempted_capture == 'Captura':
                Frame = Video
                print('Captura')
            return render_template("video.html")
        #else:
        #    return render_template("video.html")
        return render_template("video.html")


@app.route('/video_feed')
def video_feed():
    global Video
    return Response(Frame, mimetype='multipart/x-mixed-replace;
boundary=frame')


@app.route('/frame_feed')
def frame_feed():
    global Frame
```

```
    return Response(Frame, mimetype='multipart/x-mixed-replace;
boundary=frame')

# Socket publicaciones Cliente MQTT
@socketio.on('publish')
def handle_publish(json_str):
    data = json.loads(json_str)
    mqtt.publish(data['Operario'], data['message'])

# Socket suscripción Cliente MQTT
@socketio.on('subscribe')
def handle_subscribe(json_str):
    data = json.loads(json_str)
    print('Received')
    mqtt.subscribe('#')

# Cerrar socket suscripción Cliente MQTT
@socketio.on('unsubscribe_all')
def handle_unsubscribe_all():
    mqtt.unsubscribe_all()

# Función que evalua los mensajes recibidos
@mqtt.on_message()
def handle_mqtt_message(client, userdata, message):
    data = dict(
        topic=message.topic,
        payload=message.payload.decode()
    )
    global mensaje_Recepción, mensaje_Almacen, mensaje_Transporte_Secado
    global mensaje_Secado, mensaje_Transporte_Molienda, mensaje_Molino,
mensaje_Empacado
    global mensaje_Transporte_Bodega, mensaje_Despacho

    if message.topic == 'Operario/Recepción':
        mensaje_Recepción =
Payload_Decode.recepción(message.payload.decode())

    if message.topic == 'Operario/Almacen':
        mensaje_Almacen = Payload_Decode.almacen(message.payload.decode())

    if message.topic == 'Operario/Transporte_Secado':
        mensaje_Transporte_Secado =
Payload_Decode.transporte_secado(message.payload.decode())

    if message.topic == 'Secado':
        mensaje_Secado = Payload_Decode.secado(message.payload.decode())

    if message.topic == 'Operario/Transporte_Molienda':
        mensaje_Transporte_Molienda=
Payload_Decode.transporte_molienda(message.payload.decode())

    if message.topic == 'Molino':
        mensaje_Molino = Payload_Decode.molino(message.payload.decode())

    if message.topic == 'Operario/Transporte_Empacado':
        mensaje_Empacado =
Payload_Decode.empacado(message.payload.decode())

    if message.topic == 'Operario/Transporte_bodega':
        mensaje_Transporte_Bodega =
Payload_Decode.transporte_bodega(message.payload.decode())
```

```
    if message.topic == 'Operario/Despacho':
        mensaje_Despacho =
Payload_Decode.despacho(message.payload.decode())



    print(message.payload.decode())
    socketio.emit('mqtt_message', data=data)


@mqtt.on_log()
def handle_logging(client, userdata, level, buf):
    print(level, buf)


@mqtt.on_connect()
def handle_connect(client, userdata, flags, rc):
    print("Cliente conectado")


@mqtt.on_disconnect()
def handle_disconnect():
    print("Cliente desconectado")


if __name__ == '__main__':# Rutina principal

    # creación de cliente
    client = mqtt.client
    client.on_message = handle_mqtt_message

    #Suscripción al MES
    mqtt.subscribe('MES')
    # Arranca servidor HTML
    socketio.run(app, host='192.168.2.104', port=5000, use_reloader=False,
debug=False)
    #.run(app, host='192.168.2.102', port=5000, use_reloader=False,
debug=False)
```

Anexo III.I Interfaz operario - Scanner

En esta sección de código se implementa utiliza la librería cv2 de para la realización de fotografías y lectura de códigos QR.

```
# importación librerías
import cv2
from pyzbar.pyzbar import decode

# Clase cámara para escanear código QR
class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)
        print('Video')
        # self.video =
cv2.VideoCapture("http://admin:975@192.168.2.100:8081")
        print('Video2')

    def __del__(self):
        self.video.release()

    def qrcode(self):
        print('frame qr')
        ret, frame = self.video.read()

        self.info = decode(frame)
        if self.info != []:
            for i in self.info:
                code = (i[0].decode('utf-8') + '\n')
                print(code + '\n')
        else:
            code = "QR NO ENCONTRADO"
            print("QR NO ENCONTRADO")

        return code

    def get_frame(self):
        print('get_frame')
        ret, frame = self.video.read()

        ret, jpeg = cv2.imencode('.jpg', frame)
        im = cv2.rotate(jpeg, 2)

        return im.tobytes()

# Función que muestra imagen en archivo HTML
def genFrame(camera):
    frame = camera.get_frame()
    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

# Función que muestra video en archivo HTML
def genVideo(camera):
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
```

Anexo III.II Interfaz operario - Payload_Decode

Esta sección se encarga de decodificar los mensajes recibidos por MQTT.

```
# Función para descodificar una mensaje del MES
def mes(self, Payload):

    Payload_Array = Payload.split(sep=',', maxsplit=5)
    Order_Number = Payload_Array[0]
    Quantity = Payload_Array[1]
    DateStart = Payload_Array[2]
    DateEnd = Payload_Array[3]
    State = Payload_Array[4]
    Lot = Payload_Array[5]

    return Payload_Array

# Función para descodificar una mensaje del PLC de secado
def secado(self,Payload):
        Payload_Array = Payload.split(sep=',', maxsplit=4)
        Order_Number = Payload_Array[0]
        Estado = Payload_Array[1]
        Lote = Payload_Array[2]
        Temperatura = Payload_Array[3]
        Humedad = Payload_Array[4]

        print('Write Secado DB ', Order_Number, Estado, Lote, Temperatura,
Humedad)
        return Payload_Array


# Función para descodificar una mensaje del PLC de molino
def molino(self,Payload):
        Payload_Array = Payload.split(sep=',', maxsplit=4)
        Order_Number = Payload_Array[0]
        Estado = Payload_Array[1]
        Lote = Payload_Array[2]
        Temperatura = Payload_Array[3]
        Humedad = Payload_Array[4]

        print('Write Molino DB ', Order_Number, Estado, Lote, Temperatura,
Humedad)
        return Payload_Array


# Función para descodificar una mensaje del sensor de Medidas ambientales
def medidas_ambientales(self,Payload):
        Payload_Array = Payload.split(sep=',', maxsplit=1)
        Temperatura = Payload_Array[0]
        Humedad = Payload_Array[1]

        print('Write Medidas ambientales DB ', Temperatura, Humedad)
        return Payload_Array


# Función para descodificar una mensaje del sensor de operario
def operario(self,Payload):
    Payload_Array = Payload.split(sep=',', maxsplit=1)
    Temperatura = Payload_Array[0]
```

```
    Humedad = Payload_Array[1]

    return Payload_Array
```

Anexo III.III Interfaz operario - E_mail

Esta sección realiza el envió de e-mails a un servidor en internet.

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders

def send_email():
    # Iniciamos los parámetros del script
    remitente = 'quique1883@hotmail.com'
    destinatarios = ['enrique.garcia@evopro-ag.de']
#['destinatario_uno@correo.com', 'destinatario_dos@correo.com',
'destinatario_tres@correo.com']
    asunto = '[RPI] Correo de prueba'
    cuerpo = 'Este es el contenido del mensaje'
    ruta_adjunto = 'C:/Users/enrique.garcia/Desktop/test.png'
    nombre_adjunto = 'test.png'

    # Creamos el objeto mensaje
    mensaje = MIMEMultipart()

    # Establecemos los atributos del mensaje
    mensaje['From'] = remitente
    mensaje['To'] = ", ".join(destinatarios)
    mensaje['Subject'] = asunto

    # Agregamos el cuerpo del mensaje como objeto MIME de tipo texto
    mensaje.attach(MIMEText(cuerpo, 'plain'))

    # Abrimos el archivo que vamos a adjuntar
    archivo_adjunto = open(ruta_adjunto, 'rb')

    # Creamos un objeto MIME base
    adjunto_MIME = MIMEBase('application', 'octet-stream')
    # Y le cargamos el archivo adjunto
    adjunto_MIME.set_payload((archivo_adjunto).read())
    # Codificamos el objeto en BASE64
    encoders.encode_base64(adjunto_MIME)
    # Agregamos una cabecera al objeto
    adjunto_MIME.add_header('Content-Disposition', "attachment; filename=
%s" % nombre_adjunto)
    # Y finalmente lo agregamos al mensaje
    mensaje.attach(adjunto_MIME)

    # Creamos la conexión con el servidor
    sesion_smtp = smtplib.SMTP('pop3.live.com', 587) #('smtp.hotmail.com',
587)

    # Ciframos la conexión
    sesion_smtp.starttls()
```

```
    # Iniciamos sesión en el servidor
    sesion_smtp.login('quique1883@hotmail.com', '******')

    # Convertimos el objeto mensaje a texto
    texto = mensaje.as_string()

    # Enviamos el mensaje
    sesion_smtp.sendmail(remitente, destinatarios, texto)

    # Cerramos la conexión
    sesion_smtp.quit()
```

Anexo III.IV Interfaz operario – Menú

La pantalla principal Menú permite acceder individualmente a las distintas estaciones del proceso.

```
<html>

   <head>
   </head>
        <style>
                body {
                    background: #F5F5F5;

                }

                label{
                        color: #2D7B93;
                        font-weight: bold;
                }

                h1 {
                    color: #a6a6a6;
                }
                table {
                        font-family: arial, sans-serif;
                        width: 100%;
                        }

                        td, th {
                        border: 1px solid #dddddd;
                        text-align: center;
                        padding: 6px;
                        }

                        tr:nth-child(even) {
                        background-color: #dddddd;
                }
                .estado-warning {
                        border-color: darkred;
                        background-color:rgb(24, 122, 144);
                        color: white;
                        }

                .button {
```

```
                        border: none;
                        color: white;
                        padding: 15px 32px;
                        text-align: center;
                        text-decoration: none;
                        display: inline-block;
                        font-size: 16px;
                        margin: 4px 2px;
                        cursor: pointer;
                        }
                .hspacer {
                    width: 40px; /* define margin as you see fit */
                }


      /* Create three equal columns that floats next to each other */
                .column {
              float: left;
              width: 30.33%;
              padding: 10px;
              height: 300px; /* Should be removed. Only for demonstration
*/
            }

      /* Clear floats after the columns */
                .row:after {
              content: "";
              display: table;
              clear: both;
            }

        </style>


   <body>
      <form action = "http://192.168.2.104:5000/index" method = "POST">

                <div class="content">
                        <img src="{{ url_for('static',
filename='images/logo.png')}}">
                </div>
                <table class="default">
                        <tr>
                                <th><input type = "submit" value =
"Principal" name="login" style='width:350px; height:60px; font-size:30px;
font-weight: bold'/></th>
                        <th></th>
                        </tr>
                        <tr>
                                <th><input type = "submit" value =
"Recepción" name="login" style='width:350px; height:60px; font-size:30px;
font-weight: bold'/></th>
                                <th><input type = "submit" value =
"Almacén" name="login" style='width:350px; height:60px; font-size:30px;
font-weight: bold'/></th>
                        </tr>
                        <tr>
                                <th><input type = "submit" value =
"Transporte Secado" name="login" style='width:350px; height:60px; font-
size:30px; font-weight: bold'/></th>
                                <th><input type = "submit" value = "Secado"
```

```
name="login" style='width:350px; height:60px; font-size:30px; font-weight:
bold'/></th>
                        </tr>
                        <tr>
                                <th><input type = "submit" value =
"Transporte Molienda" name="login" style='width:350px; height:60px; font-
size:30px; font-weight: bold'/></th>
                                <th><input type = "submit" value =
"Molienda" name="login" style='width:350px; height:60px; font-size:30px;
font-weight: bold'/></th>
                        </tr>
                        <tr>
                                <th><input type = "submit" value =
"Transporte empacado" name="login" style='width:350px; height:60px; font-
size:30px; font-weight: bold'/></th>
                                <th><input type = "submit" value =
"Empacado" name="login" style='width:350px; height:60px; font-size:30px;
font-weight: bold'/></th>
                        </tr>
                        <tr>
                                <th><input type = "submit" value =
"Transporte bodega" name="login" style='width:350px; height:60px; font-
size:30px; font-weight: bold'/></th>
                                <th><input type = "submit" value =
"Despacho" name="login" style='width:350px; height:60px; font-size:30px;
font-weight: bold'/></th>
                        </tr>
                </table>
      </form>
   </body>
</html>
```

Anexo III.V Interfaz operario – Index

Código HTML de la pantalla principal del interfaz de comunicación

```
<html>

   <head>
   </head>
        <style>
                body {
                    background: #F5F5F5;

                }

                label{
                        color: #2D7B93;
                        font-weight: bold;
                }

                h1 {
                    color: #a6a6a6;
                }
                table {
                        font-family: arial, sans-serif;
                        width: 100%;
                        }
```

```
                        td, th {
                        border: 1px solid #dddddd;
                        text-align: center;
                        padding: 6px;
                        }

                        tr:nth-child(even) {
                        background-color: #dddddd;
                }
                .estado-warning {
                        border-color: darkred;
                        background-color:rgb(24, 122, 144);
                        color: white;
                        }

                .button {
                        border: none;
                        color: white;
                        padding: 15px 32px;
                        text-align: center;
                        text-decoration: none;
                        display: inline-block;
                        font-size: 16px;
                        margin: 4px 2px;
                        cursor: pointer;
                        }
                .hspacer {
                    width: 40px; /* define margin as you see fit */
                }


      /* Create three equal columns that floats next to each other */
                .column {
              float: left;
              width: 30.33%;
              padding: 10px;
              height: 300px; /* Should be removed. Only for demonstration
*/
            }

      /* Clear floats after the columns */
                .row:after {
              content: "";
              display: table;
              clear: both;
            }

        </style>


   <body>
      <form action = "http://192.168.2.104:5000/index" method = "POST">

                <div class="content">
                        <img src="{{ url_for('static',
filename='images/logo.png')}}">
                </div>
                <table class="default">

                        <tr>
```

```
                                <th></th>
                                <th>Recepción</th>
                                <th>Almacén</th>
                                <th>Transporte Secado</th>
                                <th>Secado</th>
                                <th>Transporte Molienda</th>
                                <th>Molienda</th>
                                <th>Transporte empacado</th>
                                <th>Empacado</th>
                                <th>Transporte bodega</th>
                                <th>Despacho</th>
                        </tr>

                        <tr>
                                <th> Orden de Producción </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Almacén }} </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Transporte_Secado}} </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Secado }} </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Transporte_Molino}} </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Molino }} </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Transporte_Empacado }} </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Empacado }} </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Transporte_Bodega }} </th>
                                <th> <div class="estado estado-warning"
div> {{ OP_Despacho }} </th>
                        </tr>

                </div>
                        <tr>
                                <th>Lote</th>
                                <th> {{ Lote_Recepción }} </th>
                                <th> {{ Lote_Almacén }} </th>
                                <th> {{ Lote_Transporte_Secado}} </th>
                                <th> {{ Lote_Secado }} </th>
                                <th> {{ Lote_Transporte_Molino}} </th>
                                <th> {{ Lote_Molino }} </th>
                                <th> {{ Lote_Transporte_Empacado }} </th>
                                <th> {{ Lote_Empacado }} </th>
                                <th> {{ Lote_Transporte_Bodega }} </th>
                                <th> {{ Lote_Despacho }} </th>
                        </tr>

                        <tr>
                                <th>Código QR</th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th> {{ QR_Empacado }} </th>
                                <th> {{ QR_Pallet }} </th>
```

```
                                <th> {{ Lote_Despacho }} </th>

                        </tr>
                        <tr>
                                <th>Medida 1</th>
                                <th>Peso:{{ Peso_Recepción }}  </th>
                                <th></th>
                                <th></th>
                                <th>Humedad:{{ Humedad_Secado }} </th>
                                <th></th>
                                <th>Humedad:{{ Humedad_Molino }} </th>
                                <th></th>
                                <th> {{ QR_Empacado }} </th>
                                <th> {{ QR_Pallet }} </th>
                                <th> {{ Lote_Despacho }} </th>

                        </tr>
                        <tr>
                                <th>Medida 2</th>
                                <th>Peso:{{ Peso_Recepción }} </th>
                                <th></th>
                                <th></th>
                                <th>Temperatura:{{ Temperatura_Secado }}
</th>
                                <th></th>
                                <th>Temperatura:{{ Temperatura_Molino }}
</th>
                                <th></th>
                                <th> {{ QR_Empacado }} </th>
                                <th> {{ QR_Pallet }} </th>
                                <th> {{ Lote_Despacho }} </th>
                        </tr>
                        <tr>
                                <th>Estado</th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th> {{ QR_Empacado }} </th>
                                <th> {{ QR_Pallet }} </th>
                                <th> {{ Lote_Despacho }} </th>
                                <th></th>
                        </tr>
                        <tr>
                                <th>Confirmar</th>
                                <th><input type = "submit" value = "Orden
realizada" name="Confirmación_Almacen"/>  </th>
                                <th><input type = "submit" value = "Orden
realizada" name="Confirmación_Almacen"/>  </th>
                                <th><input type = "submit" value = "Orden
realizada" name="Confirmación_Almacen"/>  </th>
                                <th>  </th>
                                <th><input type = "submit" value = "Orden
realizada" name="Confirmación_Almacen"/>  </th>
                                <th> </th>
                                <th><input type = "submit" value = "Orden
realizada" name="Confirmación_Almacen"/>  </th>
                                <th><input type = "submit" value = "Orden
realizada" name="Confirmación_Almacen"/>  </th>
```

```
                                <th><input type = "submit" value = "Orden
realizada" name="Confirmación_Almacen"/>  </th>
                                <th><input type = "submit" value = "Orden
realizada" name="Confirmación_Almacen"/>  </th>
                        </tr>
                        <tr>
                                <th>Foto</th>
                                <td> <input type = "submit" value = "Foto"
name="Captura_Albaran"/> </td>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                                <th></th>
                        </tr>

                </table>


      </form>
   </body>
</html>
```

Anexo III.VI Interfaz operario – Login

Código HTML pantalla registro de usuario del interfaz

```
<html>
  </head>
        <style>
                body {
                    background: #F5F5F5;

                }

                label{
                        color: #2D7B93;
                        font-weight: bold;
                }

                h1 {
                    color: #a6a6a6;
                }
                p {
                    font-size: 30px;
                }
                table {
                        font-family: arial, sans-serif;
                        width: 100%;
                        }
```

```
                        td, th {
                        border: 1px solid #dddddd;
                        text-align: center;
                        padding: 6px;
                        }

                        tr:nth-child(even) {
                        background-color: #dddddd;
                }
                .estado-warning {
                        border-color: darkred;
                        background-color:rgb(24, 122, 144);
                        color: white;
                        }

                .button {
                        border: none;
                        color: white;
                        padding: 15px 32px;
                        text-align: center;
                        text-decoration: none;
                        display: inline-block;
                        font-size: 30px;
                        margin: 4px 2px;
                        cursor: pointer;
                        }
                .hspacer {
                    width: 40px; /* define margin as you see fit */
                }


      /* Create three equal columns that floats next to each other */
                .column {
              float: left;
              width: 30.33%;
              padding: 10px;
              height: 300px; /* Should be removed. Only for demonstration
*/
            }

      /* Clear floats after the columns */
                .row:after {
              content: "";
              display: table;
              clear: both;
            }

        </style>
   <body>
      <form action = "http://192.168.2.104:5000/login" method = "POST">
       <img src="{{ url_for('static', filename='images/logo.png')}}">
         <p>Introduce tu usuario y contraseña:</p>
         <p><input type = "text" name = "username" style='width:350px;
height:40px; font-size:30px';/></p>
         <p><input type = "text" name = "password" style='width:350px;
height:40px; font-size:30px'/></p>
         <p><input type = "submit" value = "Iniciar sesion" name="login"
style='width:350px; height:40px; font-size:30px'/></p>
       <p>{{ error }}<p>
      </form>
```

```
    </body>
</html>
```

Anexo III.VII Interfaz operario – Scanner

Código HTML donde se muestra el video que permite hacer un scanner de un código QR.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Video Stream</title>
  </head>
        <style>
                body {
                    background: #F5F5F5;

                }

                label{
                        color: #2D7B93;
                        font-weight: bold;
                }

                h1 {
                    color: #a6a6a6;
                }
                table {
                        font-family: arial, sans-serif;
                        width: 100%;
                        }

                        td, th {
                        border: 1px solid #dddddd;
                        text-align: center;
                        padding: 6px;
                        }

                        tr:nth-child(even) {
                        background-color: #dddddd;
                }
                .estado-warning {
                        border-color: darkred;
                        background-color:rgb(24, 122, 144);
                        color: white;
                        }

                .button {
                        border: none;
                        color: white;
                        padding: 15px 32px;
                        text-align: center;
                        text-decoration: none;
                        display: inline-block;
                        font-size: 16px;
                        margin: 4px 2px;
                        cursor: pointer;
                        }
```

```
                .hspacer {
                    width: 40px; /* define margin as you see fit */
                }


      /* Create three equal columns that floats next to each other */
                .column {
              float: left;
              width: 30.33%;
              padding: 10px;
              height: 300px; /* Should be removed. Only for demonstration
*/
            }

      /* Clear floats after the columns */
                .row:after {
              content: "";
              display: table;
              clear: both;
            }

        </style>
  <body>
          <form action = "http://192.168.2.104:5000/video" method = "POST">
                <table class="default">
               <tr>
                  <th></th>
                  <th></th>
                  <th></th>
               <tr>
               <tr>
                  <th><input type = "submit" value = "Captura"
name="Captura"/></th>
                  <th><input type = "submit" value = "Enviar"
name="Enviar"/></th>
                  <th><input type = "submit" value = "Volver"
name="Volver"/></th>
               <tr>
               <tr>
                  <th></th>
                  <th></th>
                  <th></th>
               <tr>
                </table>
      </form>
         <form action = "imagenes" method = "GET">
         <img src="{{ url_for('frame_feed') }}" />
         <img src="{{ url_for('video_feed') }}" />
      </form>

  </body>
</html>
```

Anexo IV. PLC Molino

Este cliente MQTT simula las medidas de temperatura y humedad del molino recibe las órdenes de producción y una vez la humedad baja por debajo de un umbral envía el estado de terminado (Done) al sistema.

```
# importación de librerías
import datetime
import time
import paho.mqtt.client as paho
import random

# Ubicación Servidor Bróker
broker = "localhost"  # C:\Program Files\mosquitto

# Variables para la simulación
Temperatur_Min = random.uniform(19.0, 24.0)
Temperatur_Max = Temperatur_Min + 1
Humedad = random.uniform(19.0, 24.0)
Humedad_Dif = 0.0

# Variable Globales
sim_estado_maquina = 'OFF'
Order_Number = ''
Quantity = 0
DateStart = ''
DateEnd = ''
Status = ''
Lote = '123'

# Función que decodifica la información recibida en el mensaje
def decode_message(payload):
    payload_array = payload.split(sep=',', maxsplit=5)
    order_number = payload_array[0]
    quantity = payload_array[1]
    date_start = payload_array[2]
    date_end = payload_array[3]
    state = payload_array[4]
    lote = payload_array[5]
    estacion = order_number.split(sep='/', maxsplit=3)

    return order_number, quantity, date_start, date_end, state, lote,
estacion[1]

# Función que se ejecuta cuando se recibe un mensaje MQTT
def on_message(client, userdata, message):
    time.sleep(1)
    print("received message =", str(message.payload.decode("utf-8")))
    payload = decode_message(str(message.payload.decode("utf-8")))

    if (payload[6] == 'MO') and (payload[4] == 'confirmed'): # Orden de
producción
        global sim_estado_maquina
        global Order_Number
        global Lote
        global Temperatur_Min
        global Temperatur_Max
        global Humedad
        global Humedad_Dif
```

```
        Order_Number = payload[0]
        Lote = payload[5]
        sim_estado_maquina = 'ON'
        print('Molino ON')
        Temperatur_Min = random.uniform(19.0, 24.0)
        Temperatur_Max = Temperatur_Min + 1
        Humedad = random.uniform(19.0, 21.0)
        Humedad_Dif = 0.0

# Se crea la conexión como cliente para el PLC de Molino
client = paho.Client("PLC_Molino")

# Se asocia el cliente a la función on_message
client.on_message = on_message


print("connecting to bróker ", broker) # Se crea la conexión con el broker
client.connect(broker)  # connect
client.loop_start() # Comienza el bucle del proceso para recibir mensajes

print("subscribing to MES")
client.subscribe("MES")  # # Suscripción al topic MES

while True:
    Timestamp = datetime.datetime.now().isoformat(' ', 'seconds') # Marca
de tiempo

    if sim_estado_maquina == 'ON':
        # Temperatura
        Temeratura = random.uniform(Temperatur_Min, Temperatur_Max)
        client.publish("PLC/Molino/Temperatura", Temeratura)  # Se publica
la Temperatura actual
        print("Temeratura: ", Temeratura)

        # Humedad
        Humedad_Dif = random.uniform(0.0, 1.0)

        if Humedad_Dif > 0.5:
            Humedad = Humedad - (Humedad_Dif/5.0)

        client.publish("PLC/Molino/Humedad", Humedad)  # Se publica la
Humedad actual
        print("Humedad: ", Humedad)

        # Estado
        if Humedad > 15.0: #Si la Humedad disminuye de un 15% la máquina se
termina el proceso
            Mensaje = Order_Number + ',' + 'progress' + ',' + Lote + ',' +
str(Temeratura) + ',' + str(Humedad)
            client.publish("PLC/Molino/Estado", Mensaje)  # Se pública el
nuevo estado de máquina
            print("Estado: ", Mensaje)
        else:
            Mensaje = Order_Number + ',' + 'done' + ',' + Lote + ',' +
str(Temeratura) + ',' + str(Humedad)
            client.publish("PLC/Molino/Estado", Mensaje)  # Se pública el
nuevo estado de máquina
            print("Estado: ", Mensaje)
            sim_estado_maquina = 'OFF'
    else:
        print("Ready")
```

```
    time.sleep(5)  # espera 5 segundos para ejecutar de nuevo la rutina


client.disconnect()  # desconecta el cliente
client.loop_stop()  # detiene el bucle del proceso cliente
```

Anexo V. PLC Secado

Este cliente MQTT simula las medidas de temperatura y humedad de la estación de secado recibe las órdenes de producción y una vez la humedad baja por debajo de un umbral envía el estado de terminado (Done) al sistema.

```
# importación de librerías
import datetime
import time
import paho.mqtt.client as paho
import random

# Ubicación Servidor Bróker
broker = "localhost"  # C:\Program Files\mosquitto

# Variables para la simulación
Temperatur_Min = random.uniform(19.0, 24.0)
Temperatur_Max = Temperatur_Min + 1
Humedad = random.uniform(19.0, 24.0)
Humedad_Dif = 0.0

# Variable Globales
sim_estado_maquina = 'OFF'
Order_Number = ''
Quantity = 0
DateStart = ''
DateEnd = ''
Status = ''
Lote = '123'

# Función que decodifica la información recibida en el mensaje
def decode_message(payload):
    payload_array = payload.split(sep=',', maxsplit=5)
    order_number = payload_array[0]
    quantity = payload_array[1]
    date_start = payload_array[2]
    date_end = payload_array[3]
    state = payload_array[4]
    lote = payload_array[5]
    estacion = order_number.split(sep='/', maxsplit=3)

    return order_number, quantity, date_start, date_end, state, lote,
estacion[1]

# Función que se ejecuta cuando se recibe un mensaje MQTT
def on_message(client, userdata, message):
    time.sleep(1)
    print("received message =", str(message.payload.decode("utf-8")))
    payload = decode_message(str(message.payload.decode("utf-8")))
    print(payload[6])

    if (payload[6] == 'SE') and (payload[4] == 'confirmed'): # Orden de
producción
        global sim_estado_maquina
        global Order_Number
        global Lote
        global Temperatur_Min
        global Temperatur_Max
        global Humedad
```

```
        global Humedad_Dif
        Order_Number = payload[0]
        Lote = payload[5]
        sim_estado_maquina = 'ON'
        print('Secado ON')
        Temperatur_Min = random.uniform(19.0, 24.0)
        Temperatur_Max = Temperatur_Min + 1
        Humedad = random.uniform(19.0, 24.0)
        Humedad_Dif = 0.0


# Se crea la conexión como cliente para el PLC de Secado
client = paho.Client("PLC_Secado")

# Se asocia el cliente a la función on_message
client.on_message = on_message

client.connect(broker)  # Se crea la conexión con el broker
print("connecting to bróker ", broker)
client.loop_start()  # Comienza el bucle del proceso para recibir mensajes

print("subscribing to MES")
client.subscribe("MES")  # Suscripción al topic MES

while True:
    Timestamp = datetime.datetime.now().isoformat(' ', 'seconds') # Marca
de tiempo

    if sim_estado_maquina == 'ON': # Máquina en marcha
        # Temperatura
        Temeratura = random.uniform(Temperatur_Min, Temperatur_Max)
        client.publish("PLC/Secado/Temperatura", Temeratura)  # publish
        print("Temeratura: ", Temeratura)

        # Humedad
        Humedad_Dif = random.uniform(0.0, 1.0)

        if Humedad_Dif > 0.5: # Cada ciclo simula que la humedad disminuye
            Humedad = Humedad - (Humedad_Dif/5.0)

        client.publish("PLC/Secado/Humedad", Humedad)  # Se publica la
Humedad actual
        print("Humedad: ", Humedad)

        # Estado
        if Humedad > 20.0: #Si la Humedad disminuye de un 20% la máquina se
termina el proceso
            Mensaje = Order_Number + ',' + 'progress' + ',' + Lote + ',' +
str(Temeratura) + ',' + str(Humedad)
            client.publish("PLC/Secado/Estado", Mensaje)  # Se pública el
nuevo estado de máquina
            print("Estado: ", Mensaje)
        else:
            Mensaje = Order_Number + ',' + 'done' + ',' + Lote + ',' +
str(Temeratura) + ',' + str(Humedad)
            client.publish("PLC/Secado/Estado", Mensaje)   # Se pública el
nuevo estado de máquina
            print("Estado: ", Mensaje)
            sim_estado_maquina = 'OFF'
    else:
        print("Ready")
```

```
    time.sleep(5)  # espera 5 segundos para ejecutar de nuevo la rutina


client.disconnect()  # desconecta el cliente
client.loop_stop()  # detiene el bucle del proceso cliente
```

Anexo VI. Sensor medidas ambientales

Este cliente MQTT simula las medidas ambientales de la fábrica de temperatura y humedad.

```
# importación de librerías
import datetime # librería para obtener la fecha y la hora
import time # librería de tiempo
import paho.mqtt.client as paho # librería Cliente paho mqtt
import random # librería generación de números aleatorios


# Ubicación Servidor Bróker
broker = "localhost"  # C:\Program Files\mosquitto

# Variables para la simulación
Temperatura_Min = random.uniform(10.0, 24.0)
Temperatura_Max = Temperatura_Min + 1

Humedad_Min = random.uniform(40.0, 80.0)
Humedad_Max = Humedad_Min + 5

# Se crea la conexión con cliente MedidasAmbientales
client = paho.Client("MedidasAmbientales") # Cliente
print("connecting to bróker ", broker)
client.connect(broker)  # Conexión
client.loop_start()  # Comienza el bucle del proceso para recibir mensajes

while True: # Blucle de programa
    Timestamp = datetime.datetime.now().isoformat(' ', 'seconds')# Marca de
tiempo
    # Temperatura actual
    Temperatura = random.uniform(Temperatura_Min, Temperatura_Max)
    print("Temeratura: ", Temperatura)

    # Humedad actual
    Humedad = random.uniform(Humedad_Min, Humedad_Max)
    print("Humedad: ", Humedad)

    # Publica mensaje con temperatura y humedad
    client.publish("MedidasAmbientales", (str(Temperatura) + ',' +
str(Humedad)))  # publish

    time.sleep(30) # espera 30 segundos para ejecutar de nuevo la rutina


client.disconnect()  # desconecta el cliente
client.loop_stop()  # detiene el bucle del proceso cliente
```