UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERIA ELÉCTRICA Y ELECTRÓNICA

PROGRAMACION ORIENTADA A OBJETOS

BMA15

Proyecto final

INTEGRANTES:

- **Garcia Ferrer Flavio Cesar** **20221499B**
- **Jafet Benjamin Tongombol Huerta** **20221517K**
- **Garibay Cuba Angel Fabrizio** **20221398A**

SECCIÓN: Q

DOCENTE: Tello Canchapoma, Yury Oscar

2024 - 2

INDICE

Introduccion:

En el contexto actual, el uso eficiente de los recursos energéticos se ha convertido en una prioridad tanto para hogares como para industrias. La constante dependencia de dispositivos eléctricos, sumada al incremento de los costos energéticos, plantea la necesidad de implementar soluciones que permitan supervisar y optimizar el consumo de energía. Sin embargo, muchas veces estas soluciones son inaccesibles debido a su alto costo o complejidad técnica, dejando a una gran cantidad de usuarios sin herramientas que faciliten el control y la toma de decisiones informadas.

El monitoreo energético en tiempo real no solo permite identificar patrones de consumo, sino también detectar anomalías, reducir el desperdicio, y fomentar prácticas más responsables en el uso de la energía. Tecnologías como Arduino, combinadas con sensores de corriente y software de visualización, ofrecen una solución económica, flexible y personalizable que puede ser utilizada en una amplia gama de aplicaciones.

Este proyecto tiene como objetivo diseñar e implementar un **Sistema de Monitoreo de Consumo Energético en Tiempo Real**, capaz de medir, registrar y visualizar el consumo eléctrico de diversos aparatos. Utilizando un microcontrolador Arduino, sensores de corriente y una interfaz gráfica interactiva, se buscará proporcionar a los usuarios una herramienta práctica y accesible para el análisis de datos energéticos.

Entre las características más destacadas de este sistema se encuentran:

1. **Medición en tiempo real**: Permitiendo la actualización constante de datos sobre corriente, voltaje y potencia.
2. **Interfaz gráfica amigable**: Que facilita la comprensión y el análisis de los datos registrados, incluso para usuarios con poca experiencia técnica.
3. **Opciones de entrada manual y automática**: En caso de que el Arduino no esté conectado, el sistema permitirá el ingreso manual de datos para asegurar la continuidad del monitoreo.
4. **Portabilidad y escalabilidad**: El diseño modular del sistema hace posible adaptarlo a diferentes entornos y necesidades específicas.

Con este proyecto, se espera contribuir al aprovechamiento óptimo de los recursos energéticos, promoviendo la sostenibilidad y la conciencia ambiental, además de ofrecer una solución que pueda ser replicada y mejorada por otros entusiastas de la tecnología.

Objetivos:

Objetivos general:

• Diseñar y desarrollar un sistema integral de monitoreo de consumo energético en tiempo real que utilice un microcontrolador Arduino y sensores de corriente. Este sistema permitirá la recopilación, supervisión y análisis de datos sobre el consumo eléctrico, con el objetivo de optimizar el uso de la energía, reducir el desperdicio, identificar patrones de consumo y fomentar prácticas responsables y sostenibles en diversos entornos, tanto residenciales como industriales.

Objetivos Específicos:

• Implementar un sistema de medición precisa y en tiempo real que registre variables eléctricas fundamentales como la corriente, el voltaje y la potencia, brindando información detallada y actualizada del consumo energético de los dispositivos monitoreados.

• Diseñar e integrar una base de datos centralizada que almacene de forma organizada, segura y accesible los datos recopilados sobre el consumo energético, permitiendo su posterior análisis y consulta para generar reportes históricos.

• Incorporar modelos de machine learning en el sistema que permitan detectar patrones anómalos o inusuales en el consumo eléctrico, alertando al usuario sobre posibles fallas, sobrecargas o comportamientos ineficientes que puedan requerir atención inmediata.

• Desarrollar una interfaz gráfica amigable e intuitiva que permita al usuario visualizar, analizar y descargar los datos de consumo energético de manera sencilla. Esta herramienta facilitará la interpretación de los datos mediante gráficos, tablas y resúmenes, empoderando al usuario con la información necesaria para tomar decisiones informadas sobre el uso de energía.

Proyectos similares:

Smart Water Monitoring:

Desarrollado por IBM, con enfoque a soluciones de Internet de las Cosas (IoT) y análisis de datos avanzados para la gestión de agua y demás recursos hídricos. Con este sistema se puede monitorear y analizar en tiempo real las redes de agua, tanto en entornos urbanos como industriales, ayudando a mejorar la eficiencia del uso del agua, prevenir fugas y garantizar la calidad de este recurso.

Características:

Análisis Predictivo: Esta iniciativa puede predecir problemas antes de que ocurran. Usando técnicas de Machine Learning.

Integración con Datos Externos: El sistema también puede integrarse con otras fuentes de datos externas, como información meteorológica, para ayudar a gestionar mejor los recursos hídricos en función de las condiciones ambientales.

Tecnologías Utilizadas:

Sensores IoT: Instalados en redes de tuberías, tanques de agua, embalses y estaciones de bombeo.

IBM Watson IoT: Plataforma en la nube que gestiona y analiza los datos en tiempo real.

Análisis de Big Data: Para la detección de patrones y predicción de fallos en la infraestructura.

Machine Learning: Para prever posibles escenarios de fallos, fugas y optimización del agua

Smart Grids

Implementado por Schneider Electric donde usa el análisis y monitoreo de las redes eléctricas implementando por dispositivos de recepción de datos . Este sistema integrado a las redes existentes ayuda a detectar los fallos que puedan ocurrir e incluso se puede hacer una corregimiento preventivo .

Características

Automatización y Monitoreo en Tiempo Real : Los dispositivos IoT como medidores inteligentes, sensores de red y controladores recopilan datos en tiempo real sobre el consumo de energía, el estado de la red.

Almacenamiento masivo de datos: Las Smart Grids generan grandes cantidades de información que se almacenan en bases de datos escalables (a menudo en la nube) que permiten el acceso a los datos históricos y en tiempo real.

Cronograma de actividades:

Semana 1 **Planificación y comienzo del diseño del Proyecto**	**Semana 2** **Crear la clase para manejar los sensores de corriente y voltaje.**	**Semana 3** **Crear el reporte energético usando los sensores implementados.** **Crear el analizador energético**	**Semana 4** **Crear el servidor que almacenará y gestionará los datos energéticos**
Todos: Corrección de detalles en el diagrama de clases(atributos y métodos)	**Fabrizzio**: Desarrollar la case, simular lecturas de sensores y verificar que se almacenen correctamente.	**Flavio**: Desarrollar la clase, calcular la potencia a partir de las lecturas de los sensores.	**Alberto**: Implementar el servidor, enviar y recibir datos de reportes energéticos al servidor.
Flavio: Desarrollar la estructura básica, crear la clase usuario asignarles reportes energéticos, generar reporte.	**Alberto**: Probar la funcionalidad con lecturas simuladas.	**Benjamín**: Probar los cálculos de potencia, generar y validar el formato del reporte.	**Flavio**: Probar el almacenamiento y recuperación de datos.
Benjamín: Verificar que los usuarios puedan generar un reporte.		**Fabrizzio**: Implementar la clase.(Analizador energético), detectar anomalías en los datos de potencia.	**Alberto**: Analizar y reportar tendencias de consumo energético.

Semana 5 **Desarrollar la interfaz web para visualizar los datos energéticos.**	**Semana 6** **Pruebas Finales e Integración**	**Semana 7** **Pruebas Automatizadas**	**Semana 8** **Informe final**
Benjamín: Crear la interfaz, donde se mostrara los datos concluidos.	**Benjamín**: Realizar pruebas de integración de todo el sistema y corregir errores.	**Flavio**: Realizar pruebas de carga y rendimiento del sistema completo.	**Todos:** Presentacion del proyecto final.
Fabrizzio: Probar la visualización y actualización de gráficos.	**Fabrizzio**: Optimizar el código y asegurar que todas las clases funcionen correctamente.	**Alberto**: Preparar la documentación final y realizar una presentación de los resultados obtenidos.	

Sensor de corriente SCT-013-030

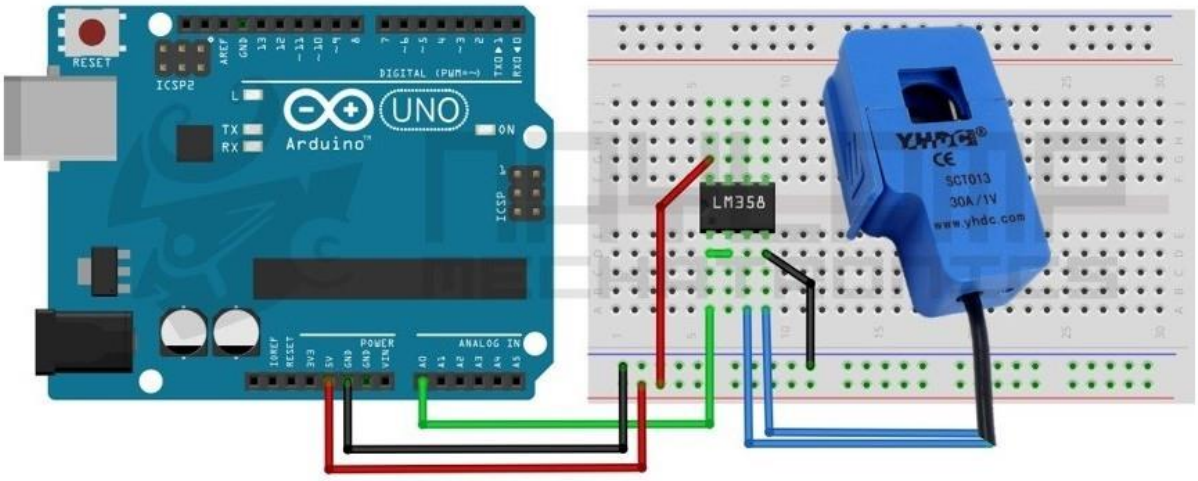
Nos permite realizar medidas en un rango de 3A. Este modelo tiene una resistencia de carga interna, entregándonos una salida voltaje. La relación es de 30A/1V. La cantidad de espiras representa la relación entre corriente que circula por el cable y la que el sensor nos entrega, esta relación o proporción es la que diferencia entre los diferentes modelos de sensores SCT-013, adicionalmente tiene una una resistencia de carga en la salida de esta forma en lugar de corriente se trabaja con una salida voltaje.

Materiales:

- Arduino Uno R3
- Sensor de corriente SCT-013-030
- Cable tipo A-B
- Cables para protoboard
- Protoboard 830
- LM358 Amplificador Operacional de

Conexión CST-013 y Arduino:

Para esto usamos un operacional, configurado en un seguidor de voltaje, usaremos el operacional LM358, que trabaja con polaridad positiva, de esta forma se eliminará la parte negativa de la señal, si bien no es un rectificador de onda completa, pero con una rectificación de media onda podemos trabajar.



Fuente:https://naylampmechatronics.com/blog/51_tutorial-sensor-de-corriente-ac-no-invasivo-sct-013.html

Codigo Arduino

Fuente:https://naylampmechatronics.com/blog/51_tutorial-sensor-de-corriente-ac-no-invasivo-sct-013.html

Diagrama UML:

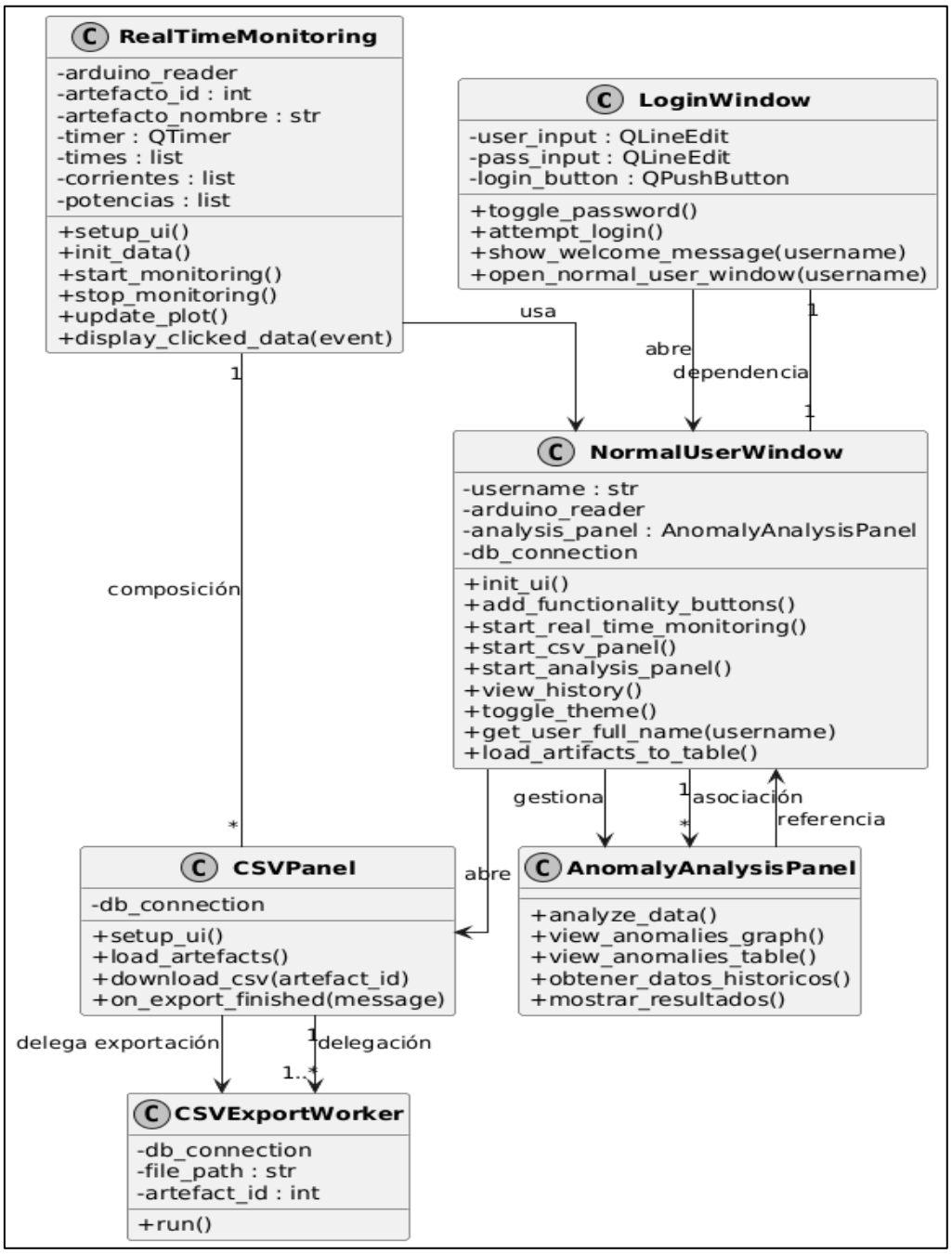


Diagrama UML

Relación UML

1. LoginWindow → NormalUserWindow(dependencia)

Explicación: **LoginWindow** crea y depende de **NormalUserWindow** después de que el usuario inicia sesión correctamente.

2. NormalUserWindow → CSVPanel: (asociación)

Explicación: **NormalUserWindow** tiene un panel **CSVPanel** para exportar datos a formato CSV.

3. NormalUserWindow → AnomalyAnalysisPanel: (**asociación)**

Explicación: **NormalUserWindow** proporciona acceso al **AnomalyAnalysisPanel** para realizar análisis de datos.

4. RealTimeMonitoring → NormalUserWindow(dependencia)

Explicación: **RealTimeMonitoring** depende de **NormalUserWindow** para mostrar los gráficos de monitoreo en tiempo real.

5. CSVPanel → CSVExportWorker: (delegación)

Explicación: **CSVPanel** delega la tarea de exportación de datos al **CSVExportWorker** para realizar la exportación en segundo plano.

6. AnomalyAnalysisPanel → NormalUserWindow (referencia)

Explicación: **AnomalyAnalysisPanel** hace referencia a **NormalUserWindow** para acceder a los datos necesarios para los análisis.

Clases y su funcionalidad

- QWidget:

Clase base para las interfaces gráficas. Tiene métodos para mostrar/ocultar ventanas y configurar el título de la ventana.

- AnomalyAnalysisPanel:

Panel principal para el análisis de anomalías.

Configura la interfaz de usuario, analiza los datos, obtiene datos históricos, muestra los resultados en una tabla y gráfico, y visualiza anomalías.

Contiene componentes como el monitoreo en tiempo real y el manejo de archivos CSV.

- CSVPanel:

Maneja la carga de artefactos desde una base de datos y la descarga de archivos CSV.

Permite interactuar con los artefactos almacenados en la base de datos y gestionar la exportación de datos en formato CSV.

- RealTimeMonitoring:

Monitorea en tiempo real los datos provenientes de un lector Arduino.

Inicia la monitorización, actualiza gráficos en vivo y detiene el monitoreo.

- LoginWindow:

Ventana para el inicio de sesión del usuario.

Permite ingresar las credenciales (usuario y contraseña), mostrar/ocultar la contraseña y realizar el intento de inicio de sesión.

- CSVExportWorker:

Gestiona la exportación de datos a un archivo CSV.

Toma la conexión a la base de datos, la ruta del archivo y el ID del artefacto, y ejecuta la exportación.

- Arduino:

Maneja la comunicación con un dispositivo Arduino.

Se encarga de actualizar los gráficos en tiempo real con los datos que recibe del Arduino.

Relaciones:

- Herencia:

AnomalyAnalysisPanel, CSVPanel, RealTimeMonitoring, LoginWindow heredan de QWidget, lo que significa que todas son ventanas de la interfaz gráfica.

Composición:

AnomalyAnalysisPanel contiene RealTimeMonitoring, CSVPanel y CSVExportWorker, indicando que el panel de análisis de anomalías gestiona estos subcomponentes.

AnomalyAnalysisPanel también interactúa con Arduino, lo que sugiere que maneja datos provenientes de dispositivos Arduino.

- Relaciones:

1 a muchos:

RealTimeMonitoring recibe datos de múltiples Arduino.

CSVPanel invoca múltiples CSVExportWorker.

1 a 1:

RealTimeMonitoring está gestionado por AnomalyAnalysisPanel.

Dependencias:

AnomalyAnalysisPanel consulta bases de datos a través de pyodbc y usa joblib para modelos de Machine Learning.

Código y Funciones Principales

class RealTimeMonitoring:

Monitorea en tiempo real la corriente y potencia de un dispositivo conectado a Arduino y mostrar esos datos a la interfaz gráfica.

Dependencias o librerias:

pyqtgraph: Para la creación de gráficos en tiempo real.

pyserial: Para la comunicación con el Arduino a través del puerto serial.

PyQt5: Para crear la interfaz gráfica.

Codigos Principales:

```
def init_data(self):
    """
    La función `init_data` inicializa los datos necesarios para el monitoreo, creando listas vacías para almacenar
    los tiempos, corrientes y potencias. También establece el tiempo de inicio para las lecturas.
    """
    """Inicializa los datos del monitoreo."""
    self.times = []
    self.corrientes = []
    self.potencias = []
    self.start_time = time.time()
```

La función **init_data** inicializara los datos del monitoreo.

```
def start_monitoring(self):
    """
    La función start_monitoring inicializa el monitoreo en tiempo real conectándose a un dispositivo Arduino y
    leyendo los datos cada 100 milisegundos.
    """
    #Inicia el monitoreo en tiempo real
    if not self.arduino_reader:
        try:
            self.arduino_reader = serial.Serial('COM7', 9600, timeout=1)
        except Exception as e:
            QtWidgets.QMessageBox.critical(self, "Error", f"No se pudo conectar al Arduino: {e}")
            return

    self.timer.start(100) # Leer datos cada 100 ms
```

La función **start_monitoring** establece una conexión en tiempo real con un dispositivo Arduino a través del puerto COM7, utilizando una velocidad de transmisión de 9600 baudios. Si la conexión es exitosa, comienza a leer los datos cada 100 milisegundos. En caso de que no se pueda establecer la conexión, se muestra un mensaje de error.

```python
def update_plot(self):
    """Actualiza las gráficas con los datos del Arduino y guarda las lecturas en la base de datos."""
    if not self.arduino_reader:
        return

    while self.arduino_reader.in_waiting > 0:
        try:
            line = self.arduino_reader.readline().decode('utf-8').strip()
            if "Irms" in line and "Potencia" in line:
                parts = line.split(",")
                corriente = float(parts[0].split(":")[1].strip().replace("A", ""))
                potencia = float(parts[1].split(":")[1].strip().replace("W", ""))

                # Eliminar o comentar los prints
                # print(f"Lectura recibida: Corriente={corriente}, Potencia={potencia}, Artefacto={self.artefacto_id}")

                elapsed_time = time.time() - self.start_time
                self.times.append(elapsed_time)
                self.corrientes.append(corriente)
                self.potencias.append(potencia)
                self.plot_corriente.plot(self.times, self.corrientes, clear=True, pen='r')
                self.plot_potencia.plot(self.times, self.potencias, clear=True, pen='b')

                # Guardar la lectura en la base de datos
                if self.artefacto_id:
                    guardar_lectura(corriente, potencia, self.artefacto_id)

        except Exception as e:
            print(f"Error al procesar datos del Arduino: {e}")
```

La función principal de update_plot es leer y procesar en tiempo real los datos de corriente y potencia provenientes de un dispositivo Arduino. Una vez que se reciben los datos, los valores de corriente y potencia se extraen y se almacenan en listas, mientras que las gráficas correspondientes se actualizan para reflejar los nuevos datos.

class LoginWindow

Esta clase define la ventana de inicio de sesión de una aplicación de monitoreo energético. Gestiona la autenticación de usuarios y administra la interfaz gráfica del inicio de sesión, incluyendo la validación de usuario, contraseña y clave dinámica para administradores.

Dependencias o librerías:

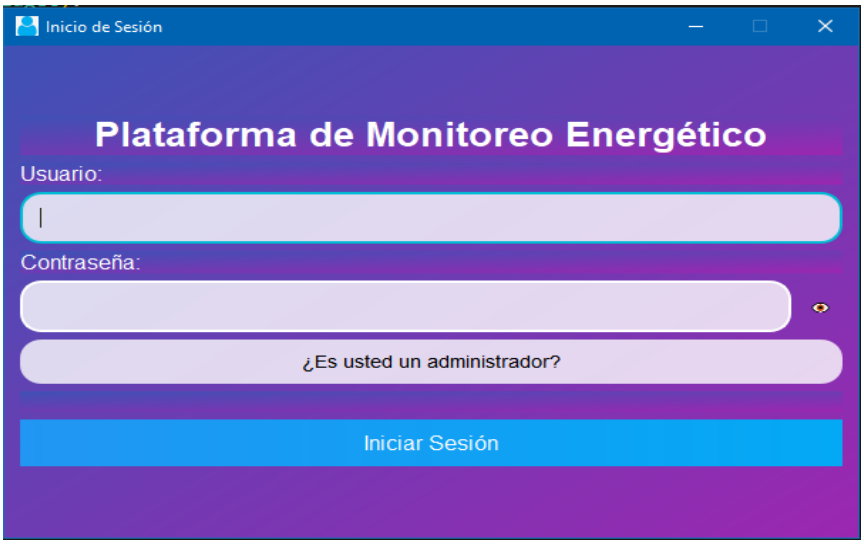
PyQt5: Para crear y gestionar la interfaz gráfica de la ventana.

QtGui, QtCore, QtWidgets: Para los componentes gráficos, como los botones, etiquetas y entrada de texto.

Códigos Principales:

```python
class LoginWindow(QtWidgets.QWidget):
    def __init__(self):
        """
        La función inicializa una ventana para la pantalla de inicio de sesión
          con un tamaño específico, título, icono de ventana y estilo de fondo con degradado.
        """
        super().__init__()
        self.setWindowTitle("Inicio de Sesión")
        self.setFixedSize(600, 400)
        self.setWindowIcon(QtGui.QIcon("C:/Users/Frabr/OneDrive/Desktop/Entornos/imagenes/icono.ico"))

        # Fondo con imagen o degradado
        self.setStyleSheet("""
            QWidget {
                background: qlineargradient(
                    spread:pad, x1:0, y1:0, x2:1, y2:1,
                    stop:0 rgba(63, 81, 181, 255),
                    stop:1 rgba(156, 39, 176, 255)
                );
            }
        """)
```

Configura y muestra la ventana principal de la pantalla de inicio de sesión, definiendo sus propiedades visuales como el tamaño, título, icono y estilo de fondo.



Se integraron elementos visuales como imágenes y colores personalizados para mejorar la estética de la interfaz de usuario. Esto incluye la adición de un icono para la ventana, un fondo con un degradado de colores. Además, se aplicaron estilos personalizados a los botones, campos de texto y otros elementos, como bordes redondeados, colores con el objetivo de ofrecer una experiencia visual más atractiva y coherente.

class NormalUserWindow

La clase NormalUserWindow es una de las clases principales y su propósito es servir como la ventana principal de la aplicación para el usuario después de que inicia sesión.

Dependencias o librerías:

PyQt5: Para los componentes GUI.

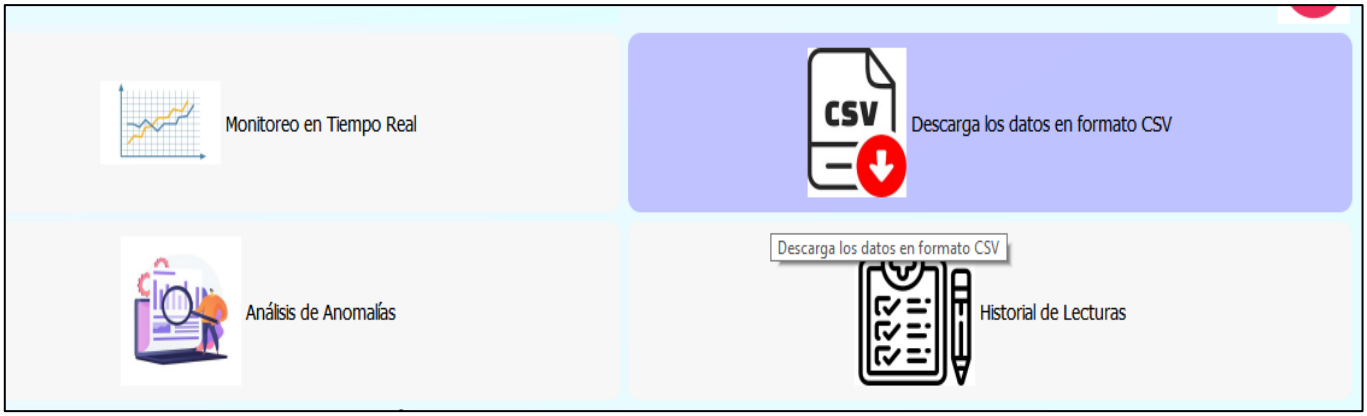
serial: Para manejar la conexión con el Arduino y la lectura de datos desde el puerto serial.

Pyodbc: conectar la aplicación con una base de datos SQL Server

Códigos Principales:

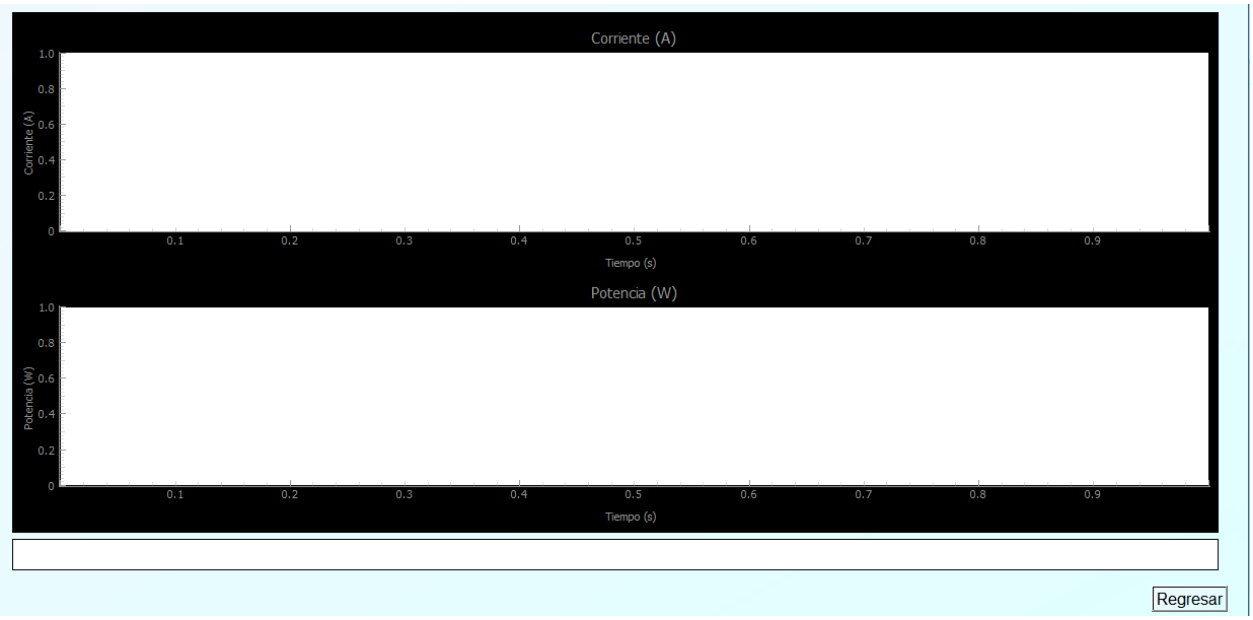
```
def add_functionality_buttons(self):
    """Añadir botones de funcionalidad como 'Monitoreo en Tiempo Real'."""
    func_layout = QtWidgets.QGridLayout()

    # Opciones de funcionalidad
    func_options = {
        "Monitoreo en Tiempo Real": (
            "C:/Users/BENJAMIN/OneDrive/Escritorio/graficotiempo.jpg",
            self.start_real_time_monitoring,
            "Visualiza datos en tiempo real"
        ),
        "Descargar Datos": (
            "C:/Users/BENJAMIN/OneDrive/Escritorio/descargacsv.png",
            self.start_csv_panel,
            "Descarga los datos en formato CSV"
        ),
        "Análisis de Anomalías": (
            "C:/Users/BENJAMIN/OneDrive/Escritorio/anomalias.jpg",
            self.start_analysis_panel,
            "Recibe alertas de anomalías"
        ),
        "Historial de Lecturas": (
            "C:/Users/BENJAMIN/OneDrive/Escritorio/historial.jpg",
            self.view_history,
            "Consulta lecturas históricas"
        ),
```

Agregan botones de funcionalidad a una interfaz gráfica




```python
def show_graph(self, artifact_id, artifact_name):
    """Muestra la gráfica para el artefacto seleccionado."""
    # Eliminar el contenido actual
    self.layout.removeWidget(self.main_widget)
    self.main_widget.deleteLater()

    # Crear el widget de monitoreo
    self.main_widget = QtWidgets.QWidget()
    layout = QtWidgets.QVBoxLayout(self.main_widget)

    # Título dinámico
    title = QtWidgets.QLabel(f"Monitoreo en Tiempo Real ({artifact_name})")
    title.setFont(QtGui.QFont("Arial", 16, QtGui.QFont.Bold))
    title.setAlignment(QtCore.Qt.AlignCenter)
    layout.addWidget(title)

    # Agregar el monitoreo
    self.monitoring_widget = RealTimeMonitoring(self.arduino_reader, artifact_id, artifact_name)
    self.monitoring_widget.start_monitoring()
    layout.addWidget(self.monitoring_widget)

    # Botón de regresar
    back_button = QtWidgets.QPushButton("Regresar")
    back_button.setFont(QtGui.QFont("Arial", 12))
    back_button.clicked.connect(self.start_real_time_monitoring)
    layout.addWidget(back_button, alignment=QtCore.Qt.AlignRight)

    self.layout.addWidget(self.main_widget)
```

El código muestra un gráfico en tiempo real para un artefacto seleccionado, donde se agregarían los valores de intensidad (corriente) y potencia.




```
try:
    # Crear la conexión a la base de datos (solo al hacer clic)
    db_connection = pyodbc.connect(
    'DRIVER={ODBC Driver 17 for SQL Server};'
    'SERVER=LAPTOP-QEI1RO77;'
    'DATABASE=DataBaseProject;'
    'Trusted_Connection=yes;'
    )
```

Establece la conexión entre tu aplicación Python y la base de datos SQL Server. Sin una conexión exitosa, no se podría consultar ni manipular los datos en la base de datos.

CLASS CSVEXPORTWORKER

Componente de interfaz gráfica en **PyQt** que permite a los usuarios descargar datos de artefactos en formato CSV desde una base de datos.

Dependencias o librerias:

PyQt5: Se usa para crear la interfaz gráfica, mostrando tablas y botones para interactuar con el usuario.

csv: Facilita la exportación de datos en formato CSV.

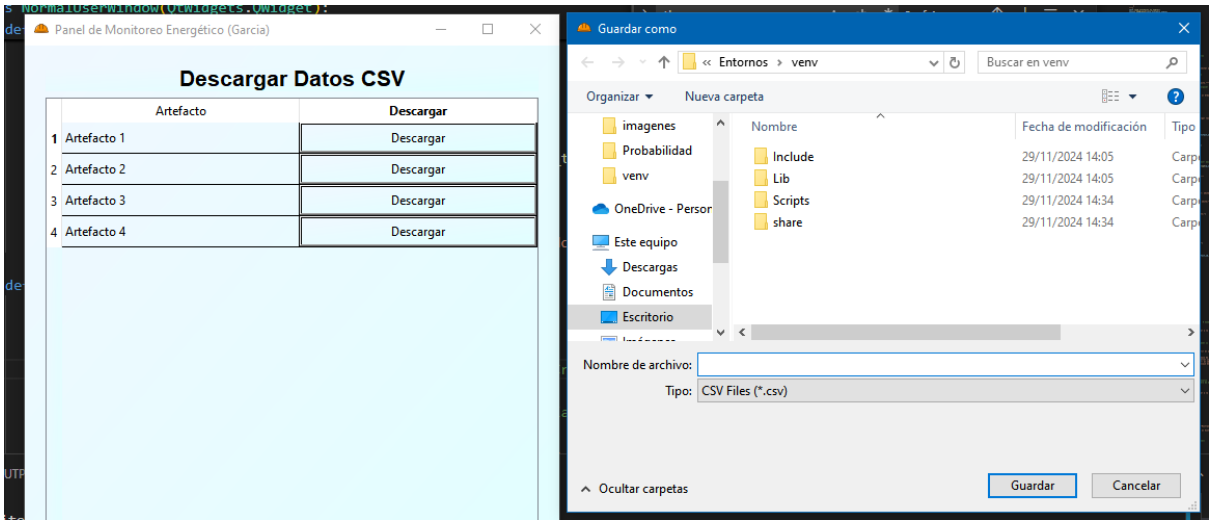
pyserial: Se utiliza para leer datos de dispositivos conectados por puerto serie, como un Arduino.

threading: Permite ejecutar la exportación de datos en un hilo separado, evitando que la interfaz gráfica se congele mientras se realiza la operación.

Códigos Principales:

```python
def run(self):
    """
    Esta función de Python exporta datos desde una consulta de base
    de datos a un archivo CSV en un hilo separado.
    """
    """Realiza la exportación en un hilo separado."""
    try:
        cursor = self.db_connection.cursor()
        cursor.execute("""
            SELECT fecha_hora, corriente, potencia
            FROM lecturas
            WHERE artefacto_id = ?
            ORDER BY fecha_hora
        """, (self.artefact_id,))
        rows = cursor.fetchall()

        with open(self.file_path, mode='w', newline='', encoding='utf-8') as file:
            writer = csv.writer(file)
            writer.writerow(["Fecha y Hora", "Corriente (A)", "Potencia (W)"])
            writer.writerows(rows)

        self.finished.emit("Exportación completada con éxito.")
    except Exception as e:
        self.finished.emit(f"Error durante la exportación: {e}")
```

La función run exporta datos desde una consulta a una base de datos a un archivo CSV en un hilo separado. Primero, ejecuta una consulta SQL para obtener las lecturas de un artefacto específico, ordenadas por fecha y hora. Luego, guarda los resultados en un archivo CSV, escribiendo las cabeceras correspondientes y las filas de datos.



La imagen muestra una interfaz con una tabla que lista los artefactos disponibles y un botón "Descargar" junto a cada uno. Al hacer clic en el botón, se inicia la exportación de los datos del artefacto a un archivo CSV.

class CSVExportWorker

Diseñado para exportar datos de un artefacto desde una base de datos a un archivo CSV de manera asíncrona.

Dependencias o librerias:

PyQt5: Para la interfaz gráfica.

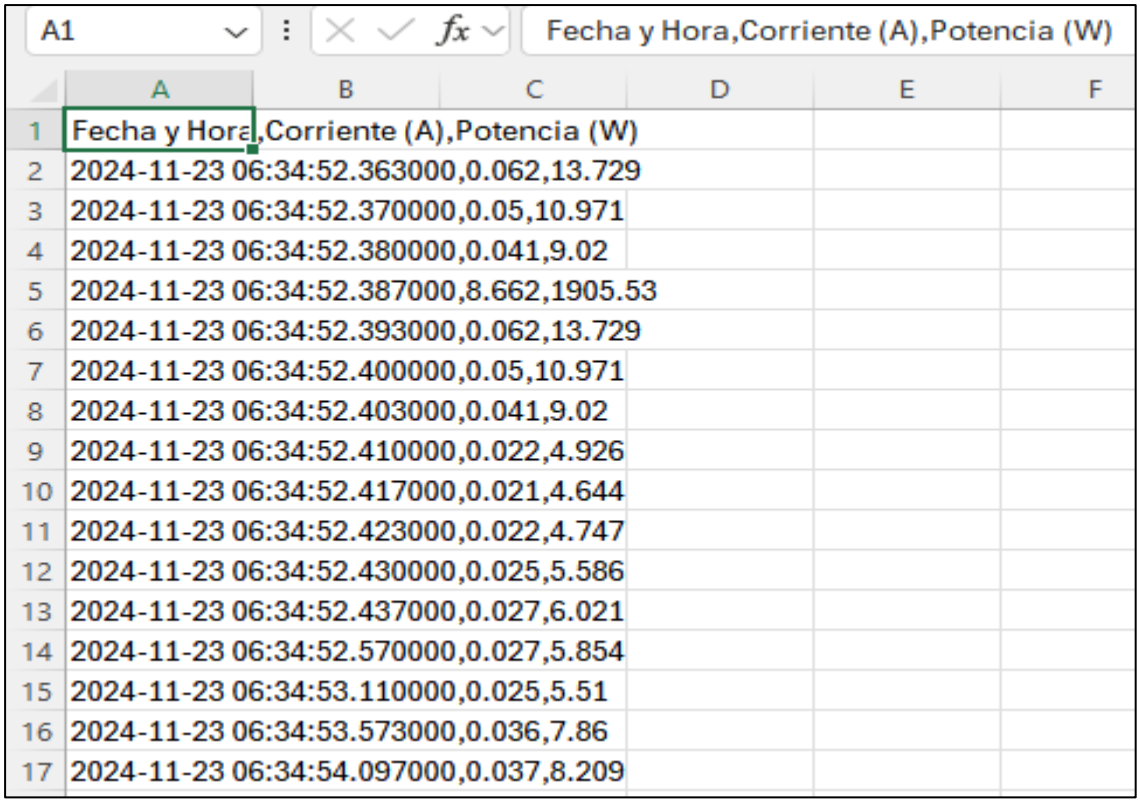
csv: Para la creación y exportación de archivos CSV.

pyodbc: Para la conexión y consulta a la base de datos.

Códigos Principales

```python
def run(self):
    """Realiza la exportación en un hilo separado."""
    try:
        cursor = self.db_connection.cursor()
        cursor.execute("""
            SELECT fecha_hora, corriente, potencia
            FROM lecturas
            WHERE artefacto_id = ?
            ORDER BY fecha_hora
        """, (self.artefact_id,))
        rows = cursor.fetchall()

        with open(self.file_path, mode='w', newline='', encoding='utf-8') as file:
            writer = csv.writer(file)
            writer.writerow(["Fecha y Hora", "Corriente (A)", "Potencia (W)"])
            writer.writerows(rows)

        self.finished.emit("Exportación completada con éxito.")
    except Exception as e:
        self.finished.emit(f"Error durante la exportación: {e}")
```

La función exporta datos de una base de datos a un archivo CSV en un hilo separado. Realiza una consulta SQL para obtener los datos, los escribe en el archivo CSV y emite una señal para indicar si la exportación fue exitosa o si ocurrió un error.



	A	B	C	D	E	F
1	Fecha y Hora,Corriente (A),Potencia (W)					
2	2024-11-23 06:34:52.363000,0.062,13.729					
3	2024-11-23 06:34:52.370000,0.05,10.971					
4	2024-11-23 06:34:52.380000,0.041,9.02					
5	2024-11-23 06:34:52.387000,8.662,1905.53					
6	2024-11-23 06:34:52.393000,0.062,13.729					
7	2024-11-23 06:34:52.400000,0.05,10.971					
8	2024-11-23 06:34:52.403000,0.041,9.02					
9	2024-11-23 06:34:52.410000,0.022,4.926					
10	2024-11-23 06:34:52.417000,0.021,4.644					
11	2024-11-23 06:34:52.423000,0.022,4.747					
12	2024-11-23 06:34:52.430000,0.025,5.586					
13	2024-11-23 06:34:52.437000,0.027,6.021					
14	2024-11-23 06:34:52.570000,0.027,5.854					
15	2024-11-23 06:34:53.110000,0.025,5.51					
16	2024-11-23 06:34:53.573000,0.036,7.86					
17	2024-11-23 06:34:54.097000,0.037,8.209					

Archivo en formato CSV donde se detalla el tiempo , corriente y potencia .

class AnomalyAnalysisPanel

Permite realizar un análisis de anomalías en datos históricos. Este panel incluye una serie de botones y una tabla para visualizar los resultados.

Este código se utiliza para analizar lecturas de corriente y potencia de artefactos, identificar posibles anomalías mediante un modelo de **machine learning** y mostrar los resultados tanto en la tabla.

Dependencias o librerías:

PyQt5: Para crear la interfaz gráfica (GUI), incluyendo botones, tablas y otras interacciones.

joblib: Para cargar y usar el modelo de Machine Learning entrenado.

pyodbc: Para conectarse a la base de datos y obtener los datos históricos.

matplotlib.pyplot: Para generar gráficos de dispersión que muestran las anomalías.

Modelo de Machine Learning

Isolation Forest

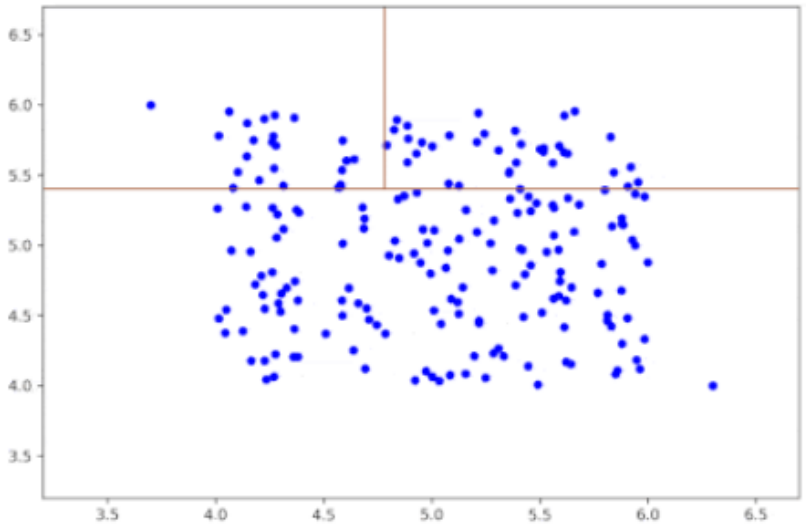
Según DataScientest(2024) :

‘Encontramos en Isolation porque es una técnica de detección de anomalías que identifica **directamente** las anomalías (comúnmente conocidas como «**outliers**«), a diferencia de las técnicas habituales que discriminan los puntos con respecto a un perfil general “normalizado”.’

En pocas palabras el algoritmo trata de identificar esos puntos de datos que son diferentes y no siguen la misma tendencia que el resto.

El principio de este algoritmo es muy sencillo:

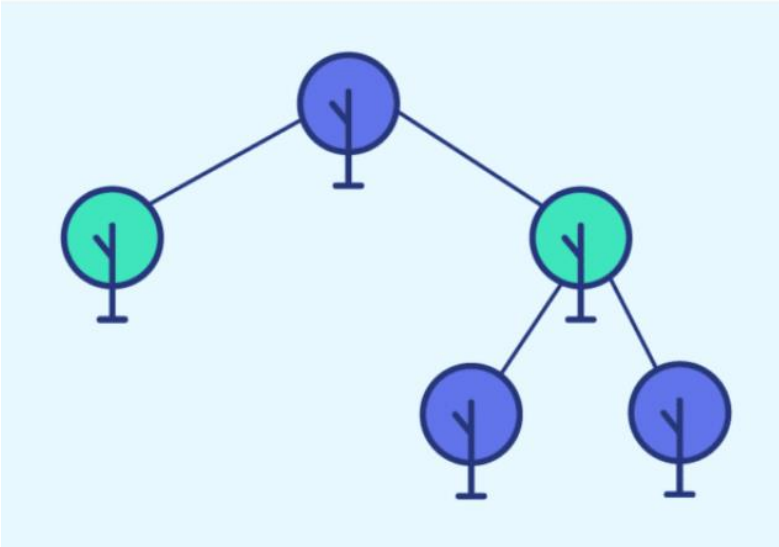
- Se selecciona una **variable** (feature) de forma **aleatoria**.
- A continuación, el conjunto de datos se divide aleatoriamente en función de esta variable para obtener **dos** subconjuntos de datos.
- Las dos etapas anteriores se repiten hasta **aislar** un dato.
- Los pasos anteriores se repiten **recursivamente**.



Fuente: https://datascientest.com/es/isolation-forest

También señalan que DataScientest(2024) :

'La construcción del algoritmo da lugar a una estructura de árbol, cuyos nudos son los conjuntos particionados durante las etapas del algoritmo y cuyas hojas son los puntos aislados. Intuitivamente, las anomalías tenderán a ser las hojas **más cercanas a la raíz del árbol**.



Fuente : https://datascientest.com/es/isolation-forest

En este contexto se utilizara los datos que se obtienen de haber guardado en la base da datos donde registran tanto la corriente como la potencia, esto con el fin de ayudar al modelo en su proceso de preentrenado .

Códigos Principales

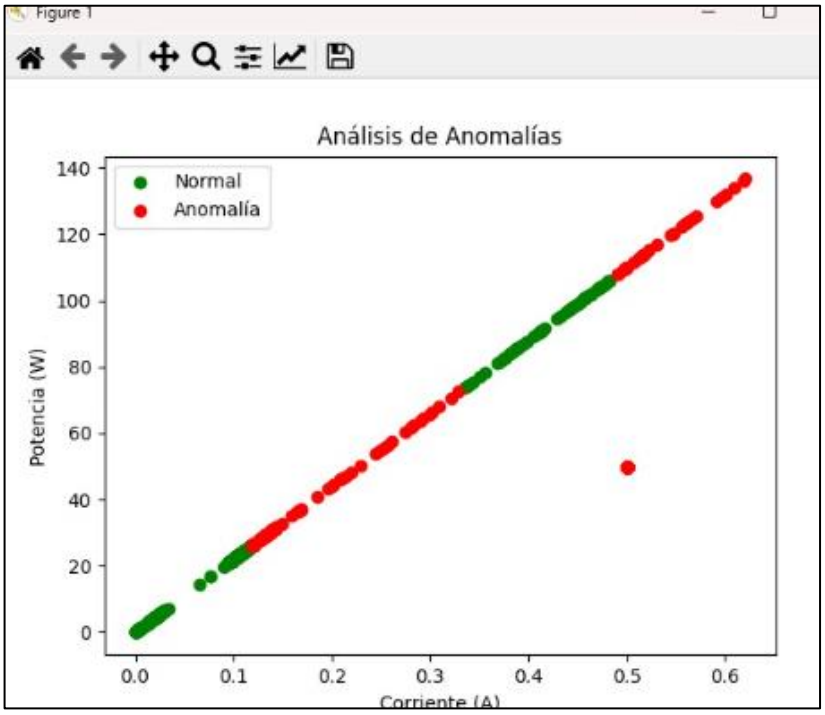
```
def obtener_datos_historicos(self):
    """Obtiene los datos históricos de la base de datos."""
    try:
        conn = self.parent.get_connection()  # Usa el método get_connection del panel principal
        if conn is None:
            return []  # Si no hay conexión, regresa una lista vacía
        cursor = conn.cursor()
        cursor.execute("SELECT corriente, potencia FROM Lecturas WHERE artefacto_id IS NOT NULL")
        rows = cursor.fetchall()
        conn.close()
        return [[row.corriente, row.potencia] for row in rows]
    except Exception as e:
        print(f"Error al obtener datos históricos: {e}")
        return []
```

Accede a los datos de la base de datos y obtener las lecturas de corriente y potencia necesarias para el análisis de anomalías, también que nos servirá para el alimentar el modelo del machine learning.

```
def view_anomalies_graph(self):
    """Muestra un gráfico de las anomalías detectadas."""
    try:
        data = self.obtener_datos_historicos()
        model = joblib.load("modelo.pkl")
        predictions = model.predict(data)

        # Preparar datos para graficar
        normales = [data[i] for i in range(len(data)) if predictions[i] == 1]
        anomalias = [data[i] for i in range(len(data)) if predictions[i] == -1]

        normales_x = [d[0] for d in normales]
        normales_y = [d[1] for d in normales]
        anomalias_x = [d[0] for d in anomalias]
        anomalias_y = [d[1] for d in anomalias]
```

La función **view_anomalies_graph** genera un gráfico de dispersión que visualiza las anomalías detectadas en los datos históricos. Utiliza un modelo entrenado para clasificar los datos

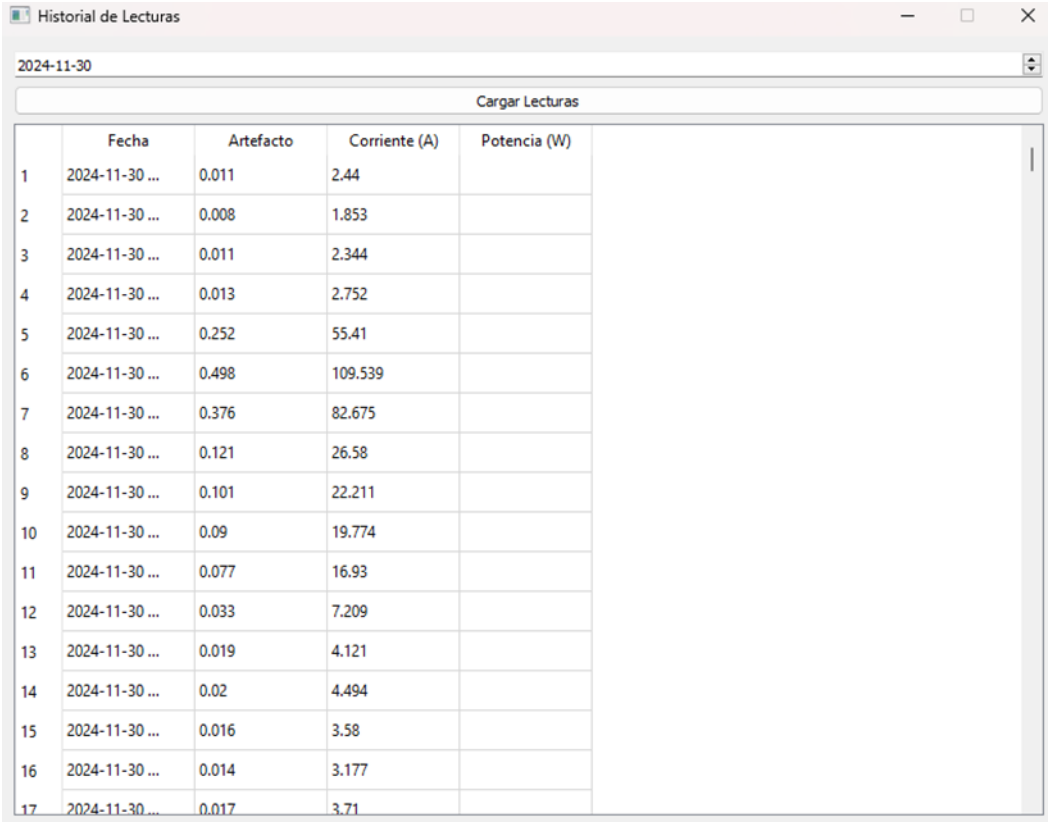
como normales o anómalos, y los muestra en un gráfico, donde los puntos normales son verdes y las anomalías son rojas.



Identificación de los patrones anómalos en el conjunto de datos, permitiendo distinguir rápidamente los puntos fuera de lo esperado (anomalías) de los que siguen el comportamiento normal.

class HistorialLecturasWindow

Esta clase es una interfaz gráfica de usuario (GUI) en PyQt5 que permite a los usuarios seleccionar una fecha y cargar las lecturas correspondientes desde una base de datos. Esta clase facilita la visualización de las lecturas de un artefacto en función de la fecha seleccionada y se conecta a una base de datos para obtener la información.



	Fecha	Artefacto	Corriente (A)	Potencia (W)
1	2024-11-30 ...	0.011	2.44	
2	2024-11-30 ...	0.008	1.853	
3	2024-11-30 ...	0.011	2.344	
4	2024-11-30 ...	0.013	2.752	
5	2024-11-30 ...	0.252	55.41	
6	2024-11-30 ...	0.498	109.539	
7	2024-11-30 ...	0.376	82.675	
8	2024-11-30 ...	0.121	26.58	
9	2024-11-30 ...	0.101	22.211	
10	2024-11-30 ...	0.09	19.774	
11	2024-11-30 ...	0.077	16.93	
12	2024-11-30 ...	0.033	7.209	
13	2024-11-30 ...	0.019	4.121	
14	2024-11-30 ...	0.02	4.494	
15	2024-11-30 ...	0.016	3.58	
16	2024-11-30 ...	0.014	3.177	
17	2024-11-30 ...	0.017	3.71	

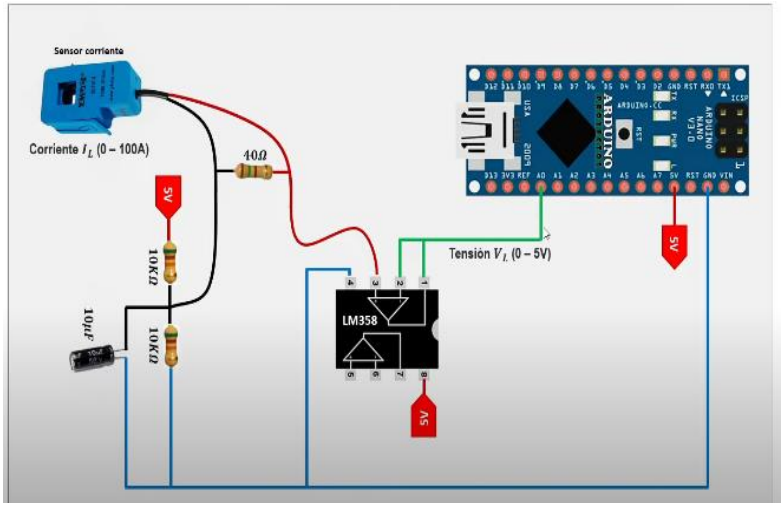
Muestra la tabla donde escogiendo la fecha nos muestra la tabla de los registros de la fecha escogida, en la tabla se ve el nombre del artefacto, la corriente y la potencia.

Evidencias de proyecto.

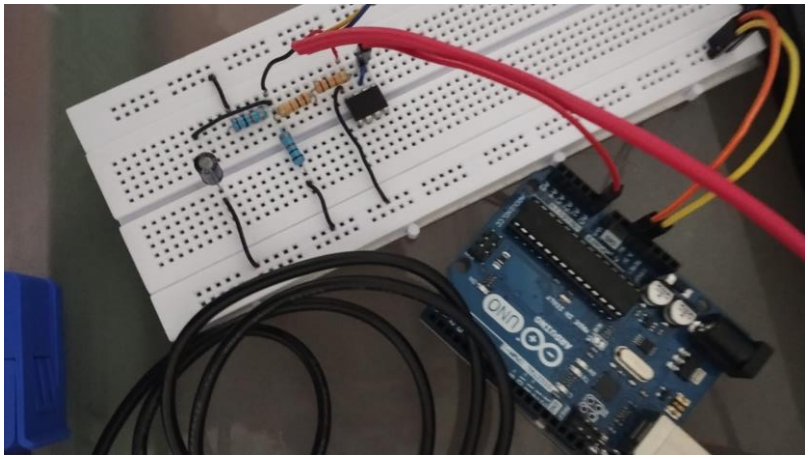
Durante la realización del proyecto se modificó el componente del Arduino con el fin de que registre las intensidades de corriente y potencia que se pudo para poder hacer la funcionalidad del monitoreo.

Prototipo 1

Se procedió a iniciar el primer esquema del circuito utilizando resistencias y capacitadores con tal que el Arduino pueda estar en funcionamiento según este diagrama:



Fuente: https://www.youtube.com/watch?v=RK75IHWeWb8

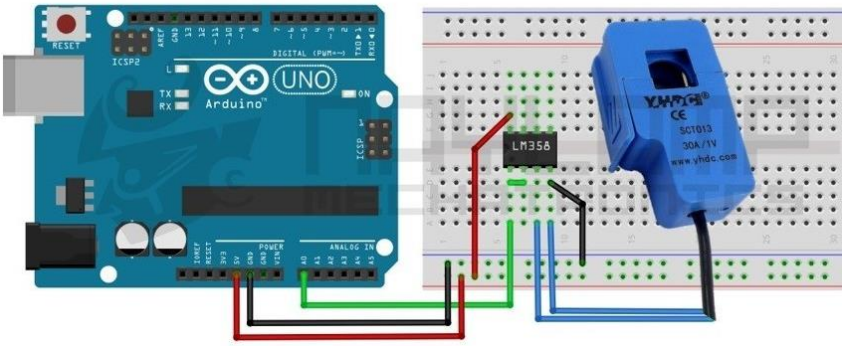


Circuito hecho del diagrama

Debido a problemas de los componente que adquirimos no se pudo seguir ante el procedimiento , esto debido a que los OPAMS (componente que mejora la precisión de las mediciones) dejaban de responder y no se podía registrar ningún valor del Arduino .

Prototipo 2

Para este ultimo prototipo buscamos algo más simplificado, para ello se descartó los capacitores y basarnos en el siguiente circuito .



Fuente: https://naylampmechatronics.com/blog/51_tutorial-sensor-de-corriente-ac-no-invasivo-sct-013.html

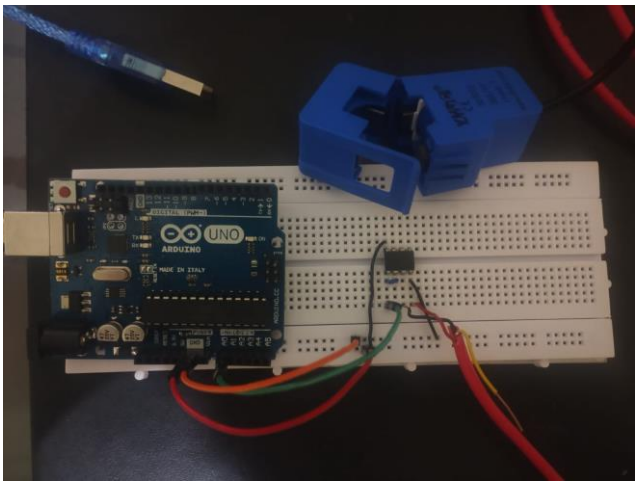


Diagrama del segundo prototipo

Conclusiones

- Fue un tema enriquecedor, ya que se utilizó componentes físicos para medir la intensidad y potencia de los artefactos, registrando los datos y mostrándolos en una interfaz que facilitó su visualización y análisis.

- Se logró simplificar lo que en proyectos de gran escala involucra mediciones y análisis de corrientes, lo que permitió aplicar estos conceptos de manera práctica y comprensible en un entorno más controlado.

- Este proyecto permitió entender cómo la integración de tecnologías como Arduino puede optimizar la medición de variables eléctricas en tiempo real, lo que resulta fundamental para el desarrollo de sistemas más eficientes y accesibles en diversos campos

Referencias

Daniel. (2024, 30 enero). *Isolation Forest: ¿cómo detectar anomalías en un conjunto de datos?* Formación En Ciencia de Datos | DataScientest.com. https://datascientest.com/es/isolation-forest

Jesus Correa - PLC. (2024b, enero 22). *Corriente alterna (consumo electrico) - Arduino / Sensor 100A no invasivo* [Vídeo]. YouTube. https://www.youtube.com/watch?v=RK75IHWeWb8

Tutorial sensor de corriente AC no invasivo SCT-013. (s. f.-b). Naylamp Mechatronics - Perú. https://naylampmechatronics.com/blog/51_tutorial-sensor-de-corriente-ac-no-invasivo-sct-013.html