CONTROL INTERACTIVO CON CALIBRACIÓN DE SISTEMAS DE TRACKING DE UNA PLANTA TERMOSOLAR CON UNA RASPBERRY

Autor

GUILLERMO MÁRQUEZ RUIZ

guimarr4@etsid.upv.es

Directores

JOSÉ ENRIQUE SIMÓ TEN, JUAN FRANCISCO BLANES NOGUERA y CARLO TERRUZZI

jsimo@disca.upv.es, pblanes@upvnet.upv.es y carlo.terruzzi@solatom.com

TRABAJO FIN DE MASTER
MASTER DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Valencia, 18 de septiembre de 2020

Control interactivo con calibración de sistemas de tracking de una planta termosolar con una raspberry

AGRADECIMIENTOS

A mis tutores del máster, por la formación y el apoyo ofrecido durante este año tan complicado para todos.

A mi cotutor, Carlo, y a todo el equipo de SOLATOM por la confianza depositada en mí y el hacerme sentir como uno más del equipo desde el primer día.

Por último, a mi familia y a mi novia, por tener la paciencia de aguantarme durante los largos días de cara al ordenador y por siempre ofrecerme unas palabras de ánimo cuando las necesitaba.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN EXTENDIDO

El objetivo de este trabajo es el desarrollo de un entorno interactivo en una Raspberry para el control de los sistemas de tracking de una planta termosolar. El entorno es accesible de forma remota para facilitar el seguimiento de los módulos solares y reducir los costes de mantenimiento. Estos protocolos comprenden ámbitos como la calibración automática de diversas partes de la planta, así como tareas de control de calidad y supervisión del estado de operación y mantenimiento.

El control de la planta se basa en comunicación Modbus y se incluyen diversas funcionalidades de calibración automática que utilizan esta comunicación para realizar las tareas de mantenimiento.

Además de las calibraciones automáticas, la aplicación sirve como una ayuda en tareas de instalación y *comissioning*, con la posibilidad de escribir y leer los registros Modbus con una interfaz sencilla e intuitiva.

Para realizar un mantenimiento preventivo se incluye la opción de descargar datos de variables internas de la planta para un seguimiento más exhaustivo, con la posibilidad de actualizar el firmware de los módulos de forma remota para el control de las versiones instaladas.

Palabras clave: Raspberry, calibración automática, planta termosolar, Modbus, mantenimiento

ABSTRACT

This project's objective is the develop of an interactive application installed on a Raspberry to control the thermo-solar plant tracking systems. The application will be accessible remotely to ease the solar modules tracing and reduce the maintenance costs. These protocols comprise areas like the automatic calibration of several systems, as well as quality control tasks and the operation and maintenance supervision.

The plant's control is based on Modbus communication and includes several automatic calibration functions that use this communication protocol to do the maintenance tasks.

In addition to automatic calibrations, the application is a tool in installing and comissioning tasks, with the posibility of reading and writing Modbus registers in an intuitive and easy-to-use interface.

It includes the function of downloading plant's internal variable data to perform a deep trace of the plants, with the possibility of updating the module firmware remotely for the installed version control.

Key words: Raspberry, automatic calibration, thermosolar plant, Modbus, maintenance

1. INTRODUCCIÓN

Este trabajo surge al realizar las prácticas del máster en la empresa Solatom CSP SL, situada en la Universidad Politécnica de Valencia. Las tareas a realizar durante estas prácticas comprenden aspectos de control de las plantas solares, desarrollo de electrónica, diseño CAD, comunicación y programación de herramientas de supervisión.

Con este último punto se plantea la idea de realizar una aplicación de supervisión de las plantas de forma remota con el añadido de la realización de tareas de calibración de forma automática. Estas tareas se realizan durante la instalación de las plantas solares y de forma continuada como tareas de mantenimiento.

Para ello se planteó la opción de crear una página web integrada en la pantalla HMI, que funciona con una Raspberry, y que junto con el PLC realizan el control industrial de las plantas. El objetivo principal era utilizar esta página de forma remota para controlar las tareas de calibración, puesto que el PLC se encarga del control.

Con este trabajo se pretende demostrar conocimientos en control y comunicación industrial, en concreto comunicación Modbus; manejo de estructuras de comunicación web (VPN, request 'GET', 'POST'…); conocimientos de electrónica y circuitos; y capacidad de manejar varios lenguajes de programación con sinergia y comunicación entre sí.

A lo largo de este documento se puede ver de forma detallada el desarrollo que ha seguido este proyecto, desde el planteamiento de objetivos, la búsqueda de soluciones, el desarrollo en sí y el análisis de los resultados.

2. JUSTIFICACIÓN

La motivación principal para la realización de este trabajo es la necesidad de un sistema de calibración y ajuste de los diferentes sistemas de los módulos solares de la empresa Solatom S.L. Esta aplicación es un complemento al control industrial de las plantas solares, la cuales ya tienen un HMI, pero sin rutinas de calibración.

Este proyecto es un paso inicial para el desarrollo de una aplicación de control industrial sencilla y con tareas de calibración para las plantas solares. A día de hoy, estas plantas están controladas por un PLC de una empresa subcontratada que se encarga del desarrollo de la parte industrial, sin embargo, en un futuro esta tarea debe ser realizada por la propia empresa.

Los métodos elegidos se escogen para amoldarse a la arquitectura que realiza esta empresa con el control industrial. Debido al uso de una pantalla industrial basada en una Raspberry se ha decidido usar esta Raspberry como servidor web para facilitar el acceso a la aplicación desarrollada.

La comunicación de forma remota también se incluye en este proyecto, la cual proporciona la mayor comodidad para el control y seguimiento de las plantas solares. Esta parte se recoge en uno de los puntos del desarrollo del proyecto.

3. OBJETIVOS

3.1. OBJETIVO GENERAL

El objetivo de este trabajo es el desarrollo de un entorno interactivo en una Raspberry para el control de los sistemas de tracking de una planta termosolar. Se pretende realizar el control de la planta con varios protocolos ejecutándose en la misma placa de desarrollo. Estos protocolos comprenden ámbitos como la calibración automática de diversas partes de la planta, así como tareas de control de calidad y supervisión del estado de operación y mantenimiento.

3.2. OBJETIVOS ESPECÍFICOS

Una vez planteado el objetivo del trabajo, se debe dividir en una serie de objetivos específicos que la solución adoptada debe cumplir para cumplir el objetivo. Estas necesidades han sido propuestas en gran parte por la empresa Solatom.

- ***Comunicación Modbus:*** debido a la arquitectura del proyecto, la comunicación de la aplicación con los dispositivos electrónicos de los módulos solares se deberá realizar mediante el protocolo Modbus RTU sobre RS485. Se dispone de una tabla de registros con sus direcciones para poder controlar el comportamiento de los módulos.

- ***Implementación:*** para implementar la aplicación, se deberá usar una Raspberry Pi como servidor para reciba las peticiones de acceso, ya que las plantas ya cuentan con una placa de este tipo para las comunicaciones de forma remota.

- ***Calibración del offset de velocidad de los servos:*** los servos que controlan el giro de los espejos deben ser calibrados por la aplicación. Para realizar la calibración se deben seguir una serie de pasos de escritura y lectura de registros.

- ***Calibración de los acelerómetros:*** los acelerómetros miden el ángulo de giro de los espejos y también deben ser calibrados de forma automática.

- ***Visualización de datos de calibración:*** una vez realizadas las calibraciones debe ser posible visualizar los valores para poder detectar posibles ejes con errores.

- ***Supervisión:*** se debe añadir una pantalla para poder observar el estado de los módulos, principalmente ángulos de giro, estado de alarmas y errores.

- ***Actualización del firmware de los módulos:*** se debe poder actualizar el programa de los módulos de forma remota para añadir actualizaciones y tener un control sobre las versiones de software de cada planta.

- ***Descarga de datos de los módulos:*** una de las funcionalidades debe incluir la posibilidad de descargar datos de las variables de control y sensorización del módulo de forma remota y que se organice en un formato sencillo de leer por programas de representación de datos.

- ***Accesibilidad en remoto:*** uno de los factores más importantes del proyecto es crear una estructura que permita el acceso a estas funcionalidades de forma remota (desde cualquier lugar con acceso a internet). Junto con esto se necesita que exista cierta seguridad para proteger el control de la planta.

- ***Interfaz simple y visual:*** estas funcionalidades deben poder ser manejadas con facilidad por cualquier persona, por lo que la interfaz debe ser intuitiva, simple y fácil de utilizar.

4. ESTADO ACTUAL

4.1. Proyecto Solatom

El objetivo de Solatom es conseguir una solución flexible para que las empresas con procesos de calor puedan ver reducida su factura energética. Esto es posible gracias al desarrollo de un concentrador solar a partir de módulos intercambiables, lo que facilita el transporte y el despliegue de estos.

Los módulos mencionados utilizan la tecnología de un colector solar, en concreto un reflector lineal de Fresnel (European Solar Thermal Electricity Association, n.d.). Este sistema se basa en una serie de espejos dispuestos en paralelo que realizan un seguimiento del sol y lo reflejan en una pequeña línea sobre el reflector, el cual está situado sobre los espejos y por el que pasa el tubo absorbedor con el agua, vapor o aceite térmico.

La energía termosolar es el aprovechamiento térmico de la energía del Sol, lo que la diferencia de la tecnología fotoeléctrica (paneles fotovoltaicos), la cual aprovecha la luz que incide sobre estos paneles y producen electricidad mediante el efecto fotoeléctrico. En cambio, para aprovechar la energía termosolar se utilizan los mencionados colectores solares

4.2. Estado del arte

En este punto se comentan aspectos del estado de la tecnología y arquitectura a utilizar en el proyecto.

I. Microcontroladores y Raspberry Pi

Un microcontrolador es un circuito integrado que contiene como mínimo los siguientes componentes: un microprocesador o CPU, una unidad RAM, una unidad ROM y puertos de entrada y salida. Además de estos componentes, los microcontroladores pueden presentar periféricos que mejoran el funcionamiento de la placa, los cuales pueden ser sensores, microprocesadores u otro tipo de chips de procesamiento de datos dependiendo de la aplicación.

Todos los microcontroladores necesitan un programa para realizar una función específica, sin el cual carecen de utilidad. Estos programas se escriben en el lenguaje de programación elegido por el usuario y una vez compilado, se escriben en el microcontrolador.

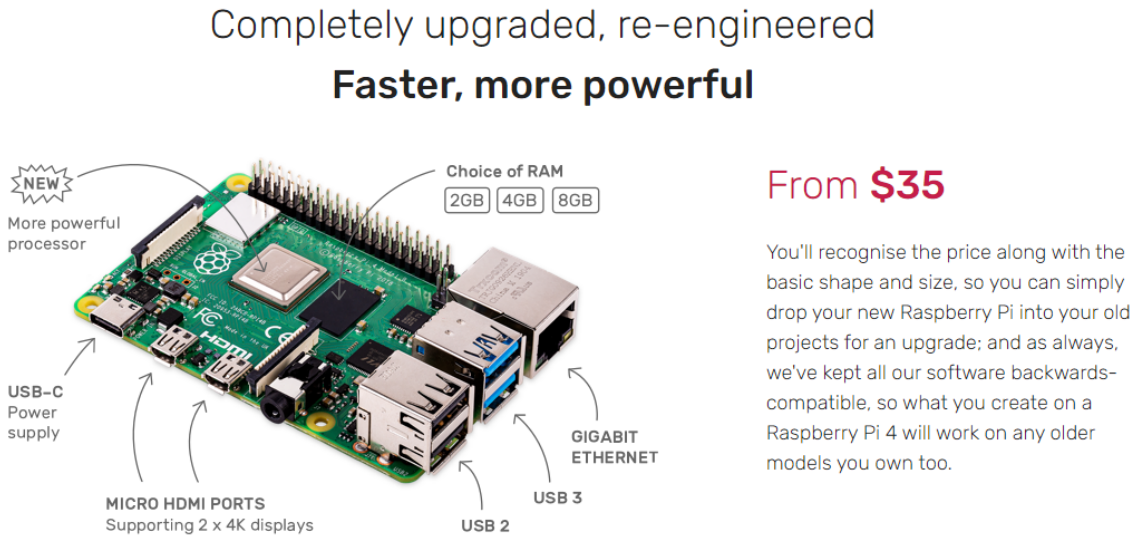
Algunos de estos microcontroladores se han desarrollado de tal manera y presentan una serie de periféricos que los permite igualar la potencia de computación de un ordenador básico; este es el caso de la Raspberry Pi.

Este pequeño ordenador fue desarrollado en 2006 por un grupo de la Universidad de Cambridge con la misión de fomentar la enseñanza de las ciencias de computación para los niños. Las primeras versiones se basaron en el microcontrolador de Atmel ATmega644 con una arquitectura ARM y sistema operativo Raspbian (derivada de Debian).

Esta placa se popularizó por el precio tan bajo del primer modelo y por la capacidad de computación, además de una serie de periféricos que daban la posibilidad de desarrollar proyectos para infinidad de aplicaciones.

El último modelo a día de hoy (Raspberry Pi 4 model B (raspberrypi.org, 2020)), tiene 2 puertos USB 3.0, 2 puertos HDMI que soportan resolución 4K, conexión a internet mediante wifi y Gigabit Ethernet, posibilidad de conectar una cámara y multitud de pines de entrada y salida para el control de otros periféricos.

Ilustración 1- Raspberry Pi 4 model B



Esta gran potencia y el bajo precio ha creado toda una comunidad de makers que aportan mejoras al código abierto y resuelven problemas y dudas para los que se inician en un proyecto con Raspberry Pi; lo que la ha llevado a ser una de las mejores opciones a la hora de realizar un prototipo con un microcontrolador.

II. Protocolos de comunicación industrial

En la industria existen diversos protocolos de comunicación para la transferencia de datos entre procesos. El uso de un protocolo u otro depende de la aplicación para la que se usará y otros aspectos como la rapidez o la cantidad de datos. Los protocolos industriales más utilizados (“Información Detallada sobre el Protocolo Modbus,” 2019) (Alonso, 2013) (LogicBus, n.d.) son los siguientes:

- ***DeviceNET***: este protocolo de comunicación es usado en la industria de automatización para conectar dispositivos de control y que intercambien datos. Permite que dispositivos como arrancadores, sensores, escáners… comuniquen con

el controlador de red. Se usa un único cable, por lo que es más fácil de conectar y operar, siendo útil para los dispositivos más simples.

- ***Profinet:*** protocolo basado en Ethernet industrial, TCP/IP y estándares de comunicación de informática. Uno de los puntos a destacar es el Ethernet a tiempo real, porque los dispositivos usan el bus de campo para cooperar en el procesamiento de solicitudes dentro del bus. Dentro de PROFINET existen diversas variantes: PROFINET/CBA, PROFINET/DCP, PROFINET/IO, PROFINET/MRP, PROFINET/MRRT, PROFINET/PTCP y PROFINET/RT.

- ***Modbus:*** este protocolo fue concebido para el control centralizado desde un SCADA o PLC de los procesos, sensores, drivers y otros dispositivos de entrada/salida físicos mediante una arquitectura maestro-esclavo, donde el maestro pregunta a una dirección y el esclavo con esa dirección responde. Entre los tipos de Modbus están Modbus TCP (modelo cliente/servidor), Modbus RTU (transmisión a través de RS-232 o RS-485) y Modbus ASCII (poco usado).

- ***ProfIbus****:* se trata de un estándar de red de campo abierto con la posibilidad de implementarlo en gran cantidad de aplicaciones. Las tres versiones de Profibus son Profibus-DP (alta velocidad de transmisión), Profibus-PA (comunicación fiable en ambientes peligrosos) y Profibus-FMS (gran volumen de información).

- ***Ethercat:*** es un protocolo de código abierto con un alto rendimiento basado en Ethernet. Es uno de los protocolos más rápidos, por lo que sus principales aplicaciones son procesos rápidos como máquinas de embalar, procesos de mecanizado o aplicaciones de movimiento en general. También cabe destacar el ahorro a la hora de la instalación al no necesitar conmutadores, enrutadores ni concentradores.

- ***Bacnet:*** es uno de los protocolos de comunicación más completos puesto que admite varias capas físicas y de enlace diferentes: PTP (punto a punto), MS/TP (master slave / token passing), ARCNET (bus token) y LONtalk (requiere herramientas especiales). Entre sus principales aplicaciones se encuentra la automatización de los distintos dispositivos de los edificios como hospitales, centros comerciales y grandes superfícies.

III. Desarrollo web y Python

En los inicios de internet, las páginas web se basaban en mostrar contenido de forma ordenada gracias a uno de los primeros lenguajes de programación creados para ello, el HTML. Pero a medida que avanzó el uso de internet, las páginas web se fueron haciendo más visuales gracias a lenguajes como el CSS y llegando a tener pequeñas animaciones y efectos por el uso del lenguaje Javascript.

Hasta aquí, los tres lenguajes básicos para realizar cualquier página web desde cero, pero en la actualidad existen infinidad de opciones para desarrollar una página web debido a la cantidad de librerías de código abierto y a los frameworks, que proporcionan un entorno de trabajo para facilitar el desarrollo.

El HTML, CSS, Javascript con sus respectivas librerías sirven principalmente para desarrollar el frontend, es decir, la parte visual de la página web. Lo que se ve cada vez que alguien hace una petición a la dirección donde se encuentra alojada.

Por otro lado, para manejar estas peticiones y gestionar el comportamiento de la web, se necesita un desarrollo backend, que se encarga de recibir datos de la interacción del cliente y enviar una respuesta que cambie el comportamiento de la página o permita enviar más datos.

En esta parte es donde se encuentra el servidor, que gestiona estas peticiones de datos. Para el backend también existen diversos frameworks y entornos de desarrollo para diferentes lenguajes de programación. Los lenguajes de programación más populares para el desarrollo de backend son Python, Ruby, Javascript (Nodejs) y PHP (“Lenguajes de programación backend,” n.d.).

En concreto Python, presenta Django(“Django Framework,” n.d.), el cual es un framework de alto nivel que ayuda al rápido desarrollo de páginas web de forma limpia y directa. Además de la rapidez es fácilmente escalable y presenta útiles herramientas que mejoran la seguridad de la página web.

5. DESARROLLO DEL PROYECTO

En este punto se pretende explicar cómo se ha desarrollado el proyecto, desde el inicio hasta el final; teniendo en cuenta cómo se ha planteado, que problemas se han encontrado, cómo se han solventado y qué resultado final se ha obtenido.

5.1. Metodología de trabajo

La metodología de trabajo a seguir será la siguiente:

1. *Planteamiento del problema*
2. *Establecer objetivos*
3. *Plantear especificaciones y necesidades*
4. *Búsqueda de soluciones posibles*
5. *Establecer estructura de la solución elegida*
6. *Desarrollo de la solución elegida*
7. *Pruebas en prototipos*
8. *Pruebas en campo*
9. *Iteración de la solución adoptada*
10. *Implementación de la solución definitiva*

Los pasos anteriores serán utilizados para el desarrollo del trabajo, los cuales se irán detallando a medida que avance el siguiente documento.

5.2. Planteamiento de soluciones alternativas y justificación de la solución adoptada

5.2.1. Tipo de aplicación

1) Soluciones alternativas y criterios de selección

A continuación, se presentan las distintas opciones planteadas para el desarrollo de la aplicación

Tabla 1 - Comparativa tipo de aplicación

	Aplicación ejecutable (Java, C++, Python…)	**Aplicación móvil (Android Studio, Flutter…)**	**Aplicación Web**
Experiencia	Poca	Ninguna	Poca
Escalabilidad	Posibilidad de escalar	Poca	Muy escalable
Implementación	Difícil de crear el acceso de forma remota	Incompatibilidad con ciertos dispositivos	Fácil en servidor web
Compatibilidad con Debian (Raspberry)	Compatible para algunos lenguajes	Incompatible	Compatible como servidor web
Dificultad de programación	Complejidad de desarrollo de una interfaz gráfica	Ninguna experiencia, mayor curva de aprendizaje	Mayor facilidad de crear interfaz, gran cantidad de tutoriales online

2) Justificación de la solución adoptada

La alternativa de crear una aplicación móvil fue descartada desde el inicio ya que era más complicado implementarla y la curva de aprendizaje mayor al no conocer nada de este tipo de desarrollo.

En cuanto a la aplicación ejecutable el problema que había era el buscar una librería con la que realizar un diseño de la interfaz gráfica atractiva, puesto que la implementación resultaría parecida a la aplicación web.

La solución adoptada finalmente es la aplicación web, ya que se puede usar la raspberry de servidor y crear un puerto a través del cual acceder desde una conexión remota. Además, es más sencillo crear una interfaz atractiva gracias a los lenguajes HTML, CSS y Javascript. Otra de las ventajas es que si se diseña bien se puede crear una interfaz adaptable a móvil, por lo que cubre la aplicación móvil.

5.2.2. Lenguaje de programación

1) Soluciones alternativas y criterios de selección

Una vez definida la estructura de la aplicación, se debe elegir el lenguaje de desarrollo o el framework de trabajo para la realización de la web.

Tabla 2 - Comparativa lenguaje programación

	Python (Django)	**PHP**	**Javascript**	**Ruby**
Experiencia	Poca	Ninguna	Poca	Ninguna
Dificultad	Sencillo de interpretar	Sencillo para principiantes	Sencillo, basado en C	Sencillo de aprender
Tipo de lenguaje	Multipropósito	Scripting	Orientado a objetos	Orientado a objetos
Librerías Modbus	Sí	No	Sí	Sí, pero no RTU

2) Justificación de la solución adoptada

Ante la gran similitud entre los lenguajes para programar una página web, se ha optado por Python por diversos motivos:

- **Experiencia**: es un lenguaje en el que se tiene cierta experiencia porque ya se han desarrollado algunas aplicaciones en este lenguaje para la empresa.
- **Reciclaje de funcionalidades**: puesto que ya se tienen programas en este lenguaje es más sencillo de implementar en esta aplicación.
- **Framework Django**: este entorno de trabajo está específicamente diseñado para el desarrollo web, por lo que ofrece herramientas específicas como características de seguridad, acceso a bases de datos, sesiones, procesamiento de plantillas, ruteo de URLS, internacionalización, ubicación y comandos sencillos para el desarrollo de la parte backend.

5.3. Estructura del proyecto

En primer lugar, se va a explicar la estructura del proyecto, principalmente el uso de los distintos lenguajes de programación y cómo debe comunicarse entre sí el servidor junto con la interfaz gráfica.

5.3.1. Instalación y estructura de archivos

Django es una librería de Python, por lo que se debe instalar un entorno de Python para poder ejecutar este framework. Para ello es recomendable también crear un entorno virtual donde instalar las librerías necesarias y no provocar conflictos con el sistema. En este caso se instala Python 3.6 con la versión de Django 3.0 en un entorno virtual con la librería *virtualenv* ("Virtualenv," n.d.).

Una vez instalado, se debe iniciar un proyecto de Django con el comando "django-admin startproject hmi" (en este caso el proyecto se llama hmi), el cual crea el directorio de archivos de la siguiente imagen:

```
TFM_GUI/
 |-- hmi/
 |     |-- hmi/
 |     |    |-- __init__.py
 |     |    |-- settings.py
 |     |    |-- urls.py
 |     |    |-- wsgi.py
 |     +-- manage.py
```

- **manage.py:** este archivo se usa para ejecutar comandos de administración como desarrollo en el servidor, pruebas, migraciones y tests.
- **init.py**: Este archivo vacío le dice a Python que esta carpeta es un paquete.
- **settings.py:** Este archivo contiene la configuración de todo el proyecto.
- **urls.py:** Este archivo es el responsable de mapear las rutas y caminos (paths) en nuestro proyecto.
- **wsgi.py:** Este archivo es simplemente una interfaz de puerta de enlace usada para despliegues.

Después de crear el proyecto, se debe crear una aplicación para que contenga las vistas y las funcionalidades de la página web.

```
TFM_GUI/
 |-- hmi/
 |     |-- rw_arduino/            <-- aplicación Django
 |     |    |-- migrations/
 |     |    |    +-- __init__.py
 |     |    |-- __init__.py
 |     |    |-- admin.py
 |     |    |-- apps.py
 |     |    |-- models.py
 |     |    |-- tests.py
 |     |    +-- views.py
 |     |-- hmi/
```

```
|    |    |-- __init__.py
|    |    |-- settings.py
|    |    |-- urls.py
|    |    |-- wsgi.py
|    +-- manage.py
```

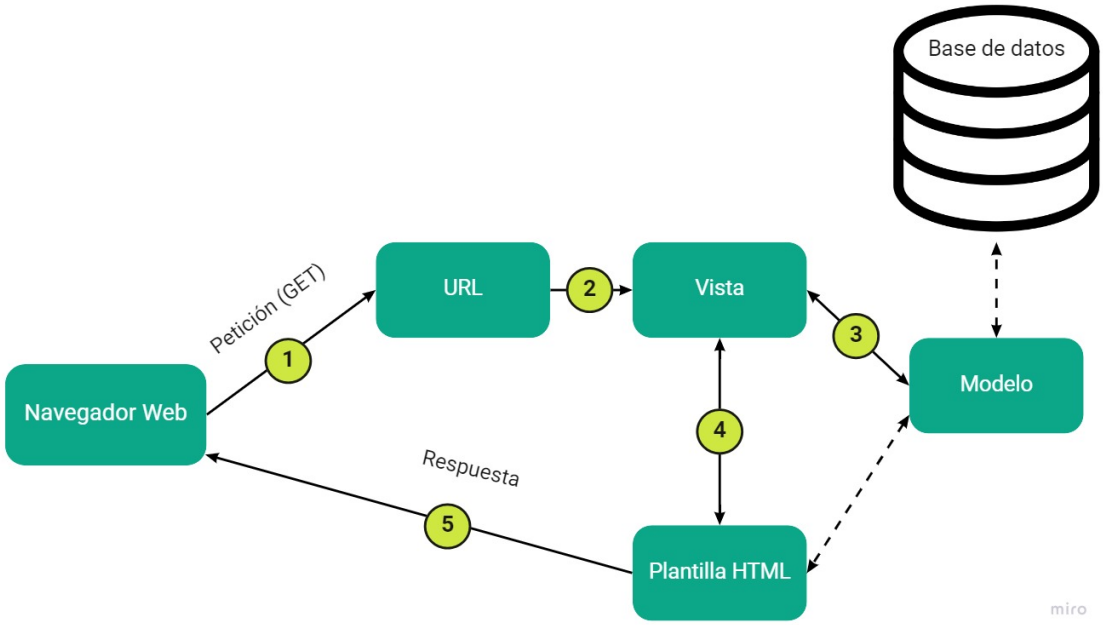
Al crear la aplicación se crean los archivos siguientes:

- **migrations:** carpeta donde se almacenan algunos archivos para mantener el registro de los cambios que se crean en el archivo models.py, con las entradas de la base de datos.
- **admin.py:** este es un archivo de configuración para la aplicación instalada de serie que gestiona algunos aspectos de la página web.
- **apps.py:** archivo de configuración de la aplicación.
- **models.py:** en este archivo se definen las entidades de la aplicación web. Los modelos son traducidos automáticamente por Django a tablas de base de datos.
- **tests.py:** archivo para escribir pruebas y testear el comportamiento de la aplicación.
- **views.py:** en este archivo se manejan las peticiones del navegador, manejando los datos recogidos, escribiendo o leyendo en la base de datos y ejecutando los comandos necesarios para finalmente devolver una respuesta a la petición.

Con estos archivos básicos, el funcionamiento de django para gestionar las vistas de la página es la siguiente:

1. El navegador web realiza una petición a una url determinada.
2. El servidor recibe la petición y carga la vista enlazada a esta url.
3. La vista es un script que se comunica con la base de datos mediante los modelos definidos y ejecuta los comandos necesarios.
4. Una vez finalizado el script, carga una plantilla HTML con la información necesaria.
5. Envía la respuesta al navegador.

Ilustración 2 - Esquema respuesta servidor



Este esquemático es un ejemplo sencillo del funcionamiento de django, el cual presenta muchas más funcionalidades que se irán explicando a medida que se vayan usando en el desarrollo del proyecto.

5.4. Vista “Inicio”

5.4.1. Objetivo y posibles soluciones

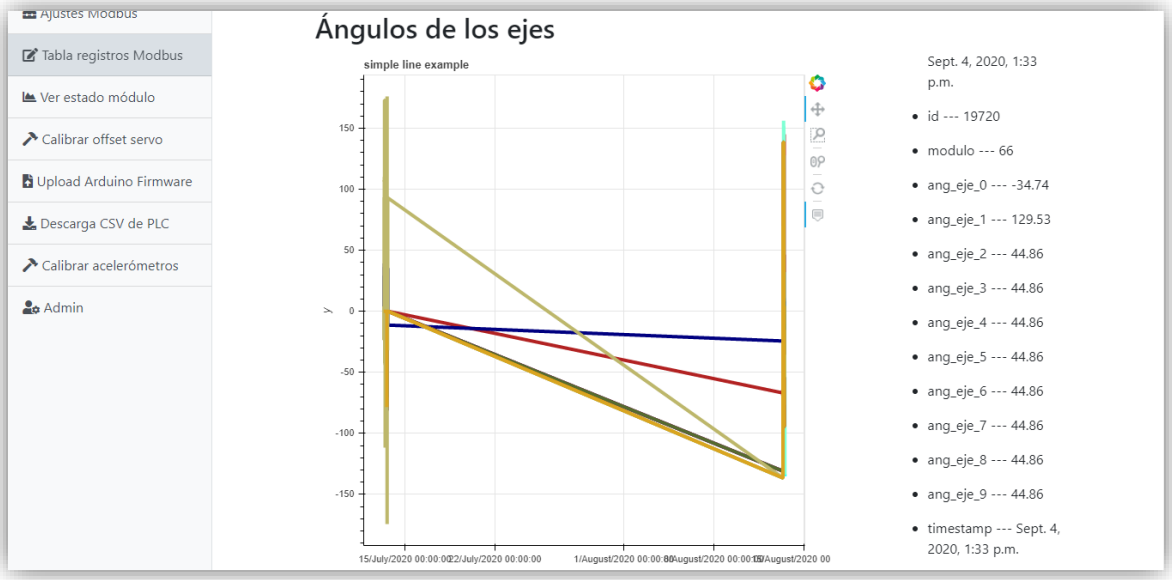
El objetivo de esta vista de inicio es presentar de forma sencilla y visual el estado de los módulos de la planta. Para ello se debe poder ver por pantalla el estado de los espejos (si se encuentra alguno en estado de error y el ángulo actual) y variables como temperatura de entrada y de salida.

5.4.1.1. Solución 1: Bokeh de Python

La primera solución que se pensó fue el uso de la librería Bokeh (Bokeh Org, n.d.) de Pyhton, la cual permite realizar gráficos de forma visual y se puede complementar con Django, ya que los gráficos se exportan en HTML e integrar en la página web.

Se desarrolla un gráfico que permite ver el estado de los ángulos de los espejos durante todo el día. En la imagen se puede ver el gráfico con pruebas de unos acelerómetros en un prototipo (por lo que los cambios de ángulos son bruscos) y a la derecha el estado actual de los ángulos en forma de lista.

Ilustración 3 - Gráfico ángulos ejes con Bokeh



Sin embargo, el **problema** de esta solución es que al instalar la librería Bokeh en la Raspberry, se descubre que existen problemas de incompatibilidad con la versión de Python y por tanto no se puede instalar esta librería.

5.4.1.2. Solución 2: Canvas de Javascript

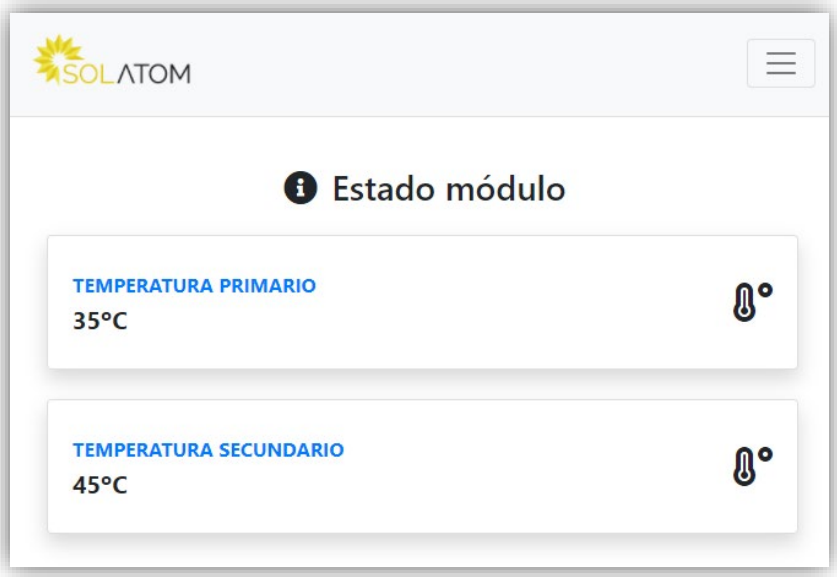
La segunda solución por la que se opta es la de realizar un desarrollo de unos gráficos en Javascript con la librería de Canvas, la cual permite realizar dibujos con figuras geométricas,

por lo que se pretende hacer un esquema de los módulos y mostrarlo en la página de inicio. **Esta es la solución que se implementa para poder observar el estado de la planta solar.**

5.4.2. Vista

La vista se basa en una serie de tarjetas que muestran variables cuya visualización es importante para el control del estado de la planta solar como son: temperatura primario, temperatura secundario…

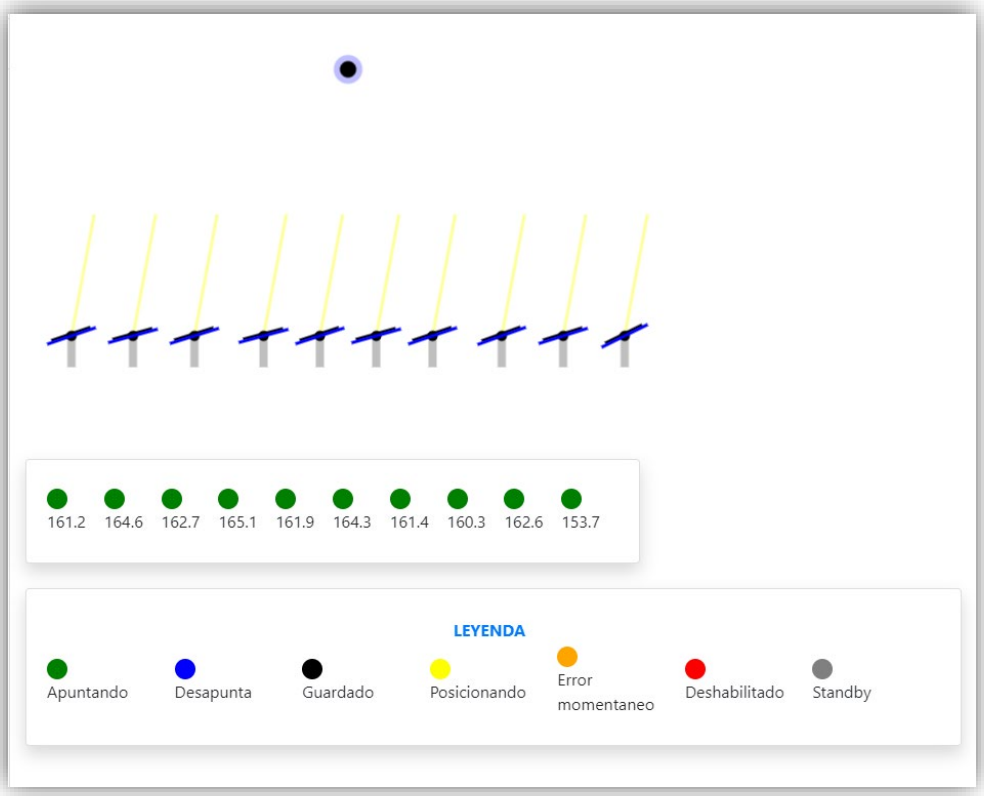
Ilustración 4 - Vista Inicio



A continuación, se puede ver el dibujo del estado de los espejos, realizado con Canvas de Javascript. El círculo que se encuentra encima de los espejos representa el tubo colector por donde pasa el fluido a calentar. De los espejos salen dos rayos: el rayo amarillo representa el rayo del sol (indicando la posición de este en el cielo) y el rayo naranja representa el reflejo del sol sobre el espejo.

Bajo el dibujo del módulo, se encuentra una etiqueta con la lista de módulos, sus ángulos y un círculo de color para mostrar el estado de ese eje. Más abajo se encuentra la leyenda para conocer el significado de los colores del círculo.

Ilustración 5 - Vista gráfico Canvas página Inicio



Estos datos de estado de los ángulos y variables de temperatura se obtienen de una llamada de tipo Ajax al servidor, el cual gestiona la llamada como se explica en el siguiente punto. La petición de Ajax, envía el módulo del cual se quieren obtener los datos y una vez recibida la respuesta, actualiza el Canvas y los valores de ángulos y temperatura.

Además, se ha añadido una función que se ejecuta y actualiza los valores y el dibujo al cambiar de módulo en el menú superior.

5.4.3. Script Pyhton

Este script se encarga de recibir las peticiones de Ajax y de cargar la página inicial.

5.4.3.1. Página principal

Para cargar la página principal, primero llama a la función *get_header_address,* que carga un formulario con los módulos añadidos a la base de datos y lo pasa a la plantilla HTML. Esta función se ejecuta en todas las páginas para poder interactuar con los módulos de forma individual.

5.4.3.2. Peticiones POST Ajax

Este script también se encarga de gestionar las peticiones Ajax de Javascript para devolver las variables de estado de los módulos. Al tratarse de la página de inicio y debido a que la

comunicación Modbus no se ha iniciado, los datos se recogen de la API integrada del webcontroller del PLC, la cual permite realizar peticiones de tipo GET y POST para consultar variables o cambiar algunos parámetros.

Por lo tanto, se ha escrito una variable de Pyhton de tipo diccionario con todas las variables, y su correspondiente registro, necesarias para la página principal (en el caso de necesitar cambiar esta lista sería posible sin necesidad de cambiar el código del script).

A continuación, se realiza un bucle *for* sobre el diccionario realizando una petición de tipo GET apuntando a la ip del PLC. Esta conexión ip se realiza de forma remota a través del servidor central de Solatom (cuya conexión se detalla más adelante en el punto 5.12) durante el desarrollo de la aplicación. Cuando se instale el programa, se usará la ip local del PLC para reducir el consumo de datos de la tarjeta 3G.

La estructura de la url para realizar la petición viene determinada según la documentación del PLC Eclypse. En este caso se obtienen los valores actuales de las variables de tipo analógico con la siguiente forma:

Ilustración 6 - URL de petición API PLC

```
ip = "51.75.121.235:33802"

#ip = "192.168.0.30" #local Raspberry
url = "http://" + str(ip) + "/api/rest/v1/protocols/bacnet/local/objects/analog-value/"

url += str(value) + "/properties/present-value"
```

La respuesta de la petición tiene formato JSON, por lo que se debe extraer el valor analógico para añadir a otra variable de tipo JSON que será la respuesta que reciba la petición Ajax al servidor. En el apartado de *Análisis de resultados* se comenta cómo ha funcionado esta estructura de peticiones al PLC.

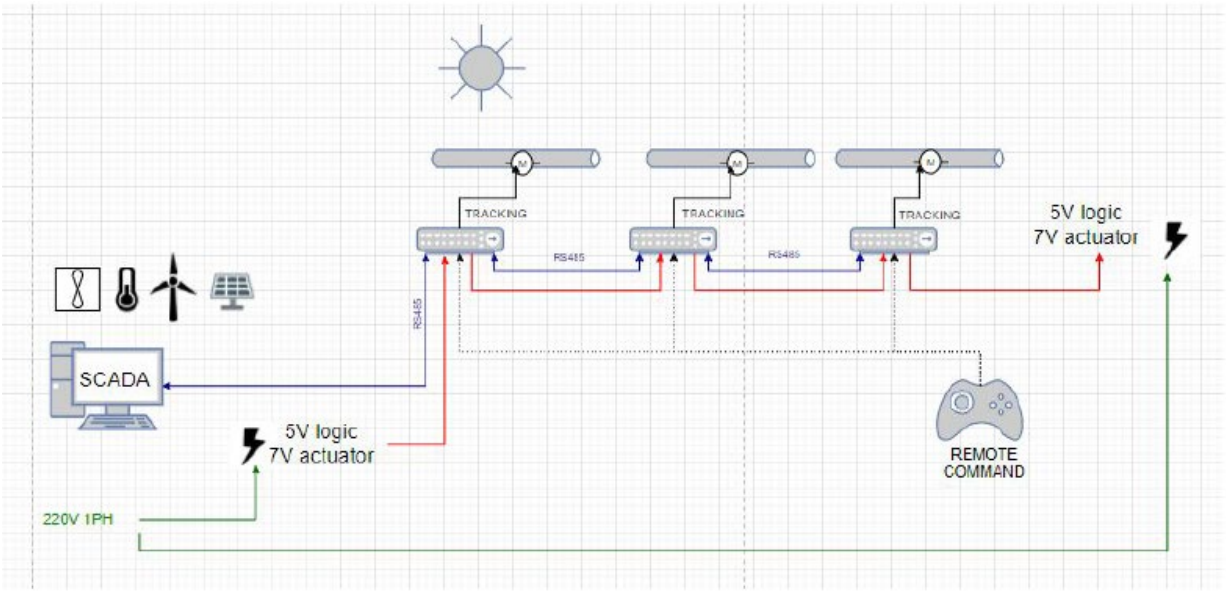
5.5. Comunicación Modbus

En este apartado se va a detallar el desarrollo de la comunicación Modbus con los módulos y su implementación en el proyecto.

5.5.1. Arquitectura del sistema

La arquitectura del sistema se basa en varios módulos interconectados en serie entre sí mediante un cable UTP con RS485 (“Modbus vs RS485,” n.d.). Cada módulo tiene una línea de comunicación con el SCADA o PLC de control y otra línea de alimentación. En la siguiente imagen se puede ver el esquemático con esta arquitectura:

Ilustración 7 - Esquema arquitectura planta solar



Con la comunicación Modbus (Witte Software, 2020), cada módulo es un esclavo controlado por el maestro, que en este caso es el SCADA o PLC. Para diferenciar los módulos, se nombran alfabéticamente desde la A, por lo que en formato ASCII, el primer módulo tendrá una dirección o ID de valor 65 (A), el segundo 66 (B) y así hasta el último. Con esta arquitectura para leer o escribir en un módulo, se usa su dirección y de esta forma el módulo “nombrado” responde con el valor pedido. Para realizar una petición a todos los módulos, se usa el valor 0.

5.5.2. Objetivo y planteamiento de soluciones

El objetivo principal de este proyecto era conseguir una comunicación basada en Modbus que fuera fiable y estable, puesto que es la única forma de comunicarse con los módulos.

Como el proyecto va a tener diversas rutinas de calibraciones que utilizan este tipo de comunicación para realizar las operaciones se plantea el realizar una rutina que funciona de forma continua que se encarga de escribir y leer registros Modbus.

5.5.2.1. Problema: rutina continua y sincronización

Al realizar pruebas con Django se comprueba que los scripts que manejan las vistas se ejecutan y al acabar devuelven el render de la página, por lo que no es viable que una de las páginas sea la rutina ya que se quedaría la página cargando y bloquearía el resto de las aplicaciones.

La solución a este problema es que la rutina funcione en *background* mediante el **uso de hilos de ejecución**. Este hilo se lanza y no necesita que ninguna vista se mantenga abierta para su ejecución. Por lo tanto, se pueden acceder a todos los apartados de la web y la rutina seguirá corriendo hasta que se le mande pararse.

Al utilizar hilos de ejecución se debe buscar alguna forma de sincronizarlos para que no se produzcan problemas de condiciones de carrera. La solución a esta parte es **el uso de semáforos y eventos de condición**, con los cuales se puede parar la ejecución de un hilo si hay otro realizando una tarea crítica y poder sincronizar la lectura y escritura correctamente.

En el siguiente apartado se explica con detalle el desarrollo de estas soluciones planteadas.

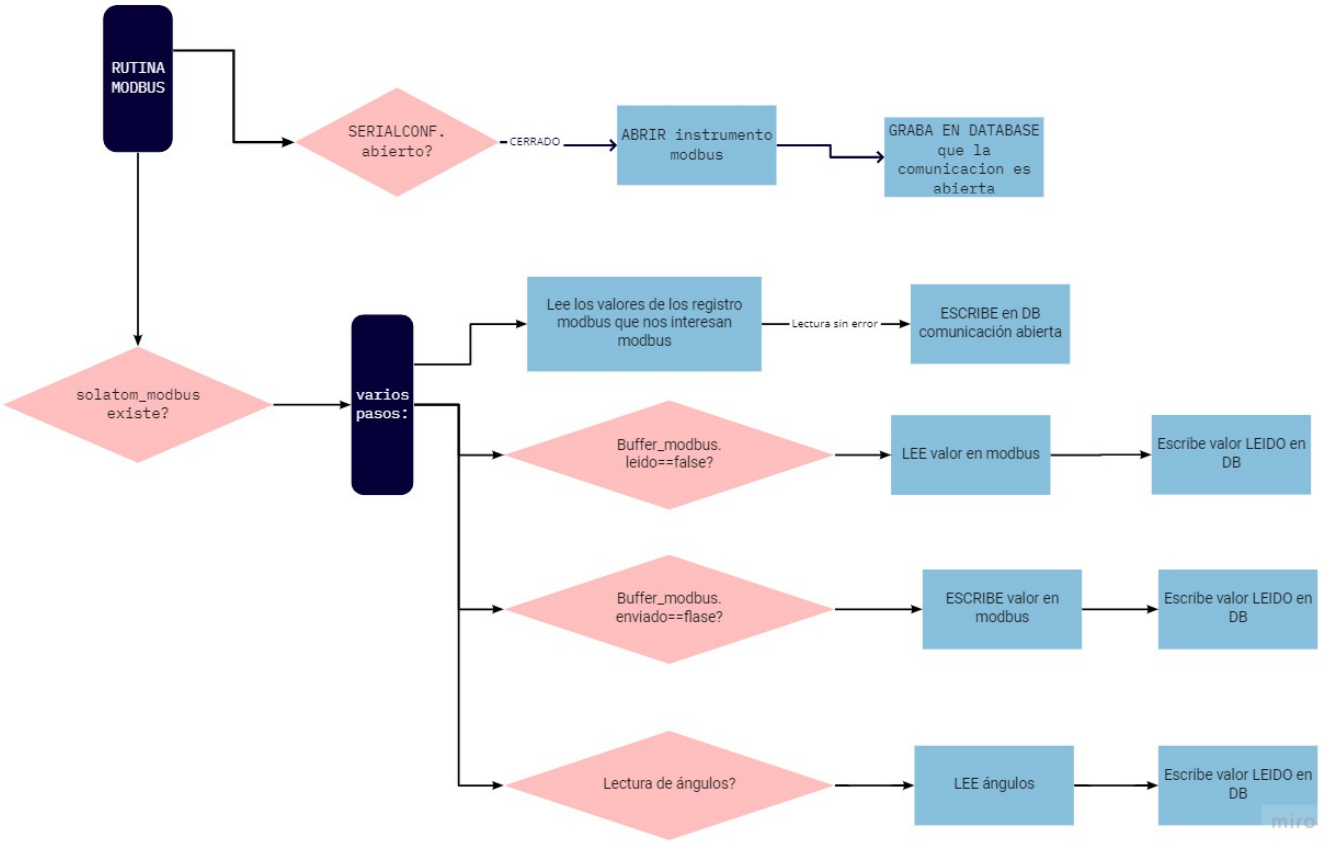
5.5.3. Rutina lectura y escritura Modbus

5.5.3.1. Lógica

Se desarrolla un hilo principal que se llamará “rutina_modbus”, que se encarga de mantener la comunicación con los módulos abierta y realizar las lecturas y escrituras de los registros demandados por las diferentes rutinas. La rutina recoge la cantidad de módulos introducidos por el usuario en la vista y realiza las conexiones de comunicación modbus.

En la siguiente imagen se puede ver la lógica de esta rutina modbus que se ejecutará periódicamente y para cada esclavo en la base de datos desde que se lanza hasta que se le manda terminar desde alguna de las vistas de la página web. Mientras está en funcionamiento se pueden cambiar los ajustes de conexión modbus para cada esclavo y una vez acabado el hilo se puede volver a lanzar.

Ilustración 8 - Lógica rutina Modbus

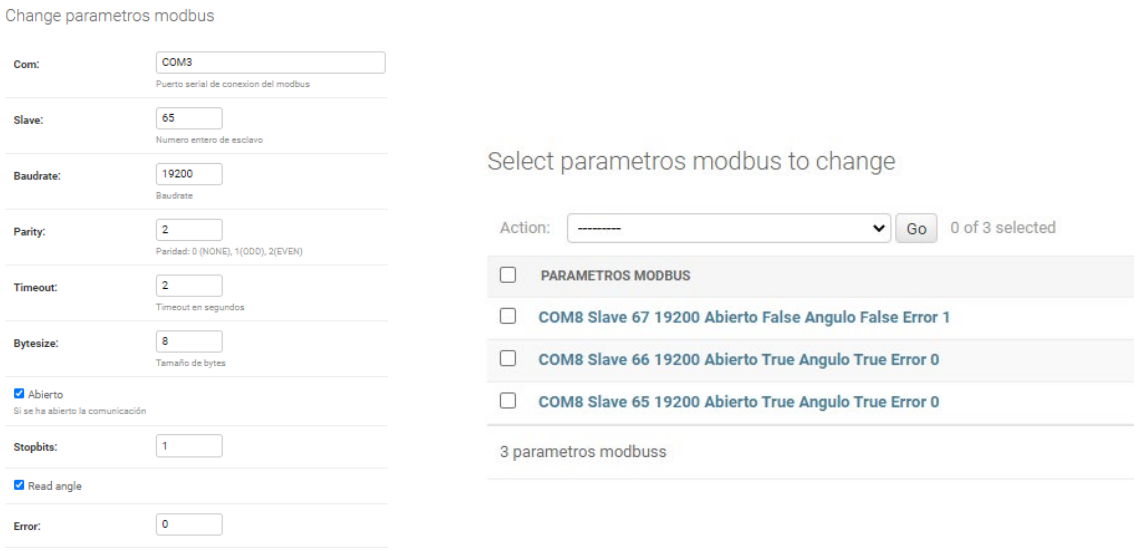


5.5.3.2. Modelos base de datos

Este hilo se apoya en varios modelos de la base de datos para realizar las lecturas y escrituras de los registros pedidos. La base de datos hace la función de buffer en la que se escriben las peticiones con una variable en *False* para que la rutina filtre por esta variable y realice la lectura o escritura. Los modelos de la base de datos utilizados en esta rutina son los siguientes:

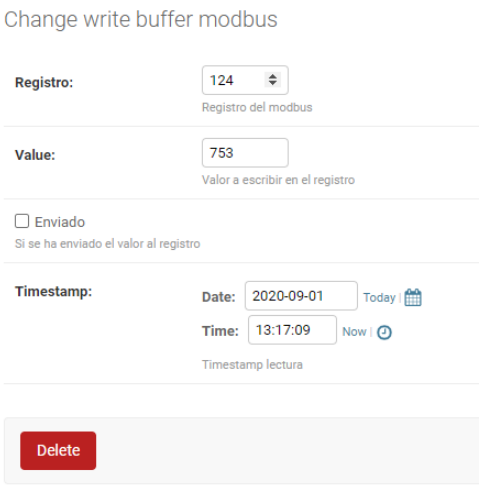
- *Parámetros Modbus:* en este modelo se guardan los parámetros de comunicación modbus con los que la rutina crea la conexión. La gestión de las vistas y la rutina modbus está hecho de manera que por cada esclavo registrado hay un elemento en la base de datos.

Ilustración 9 - Modelo Parámetros Modbus



- *Write Buffer Modbus:* en este modelo se tiene como parámetros el registro al que escribir, el valor a escribir, si se ha escrito o no y la hora en que se ha hecho.

Ilustración 10 - Modelo Write Buffer Modbus



- *Read Buffer Modbus:* este modelo es igual que el anterior sólo que sirve para leer registros.

5.5.3.3. Librería modbus

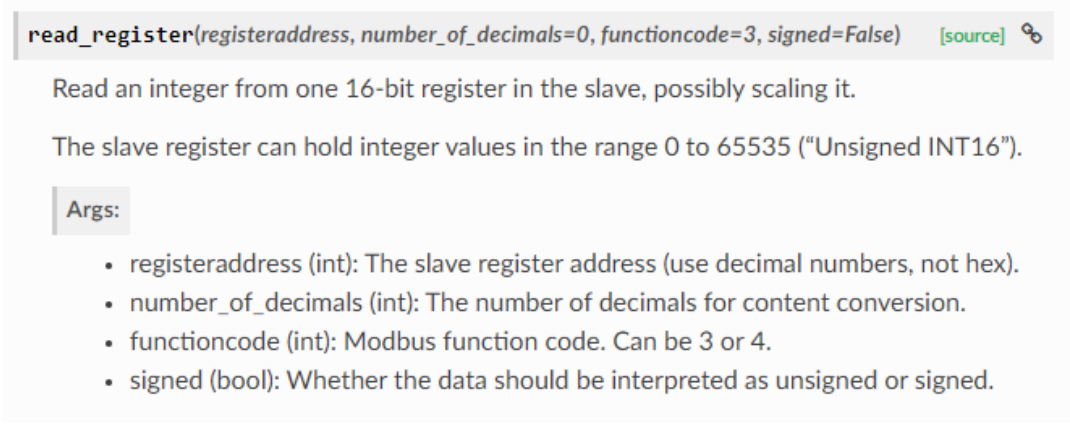
Para la lectura y escritura de los registros se ha buscado una librería modbus en Python, cuyo nombre es *minimalmodbus* (Revision, 2019). Esta librería se crea un instrumento asociado a un puerto y a una dirección de esclavo o id. Este instrumento tiene una serie de funciones para leer o escribir registros como marca el protocolo modbus. Hay que tener en cuenta que existen cuatro tipos de variables en modbus:

- *Bits (Coils)*
- *Discrete inputs*
- *Holding registers*
- *Input registers*

Para cada tipo de variable existe un código de función determinado para la lectura o escritura de uno o múltiples registros. En este proyecto, todas las variables son holding registers por lo que se usan las siguientes funciones:

- Para la lectura de un único registro:

Ilustración 11 - Función read_register de minimalmodbus



read_register(*registeraddress, number_of_decimals=0, functioncode=3, signed=False*) [source]

Read an integer from one 16-bit register in the slave, possibly scaling it.

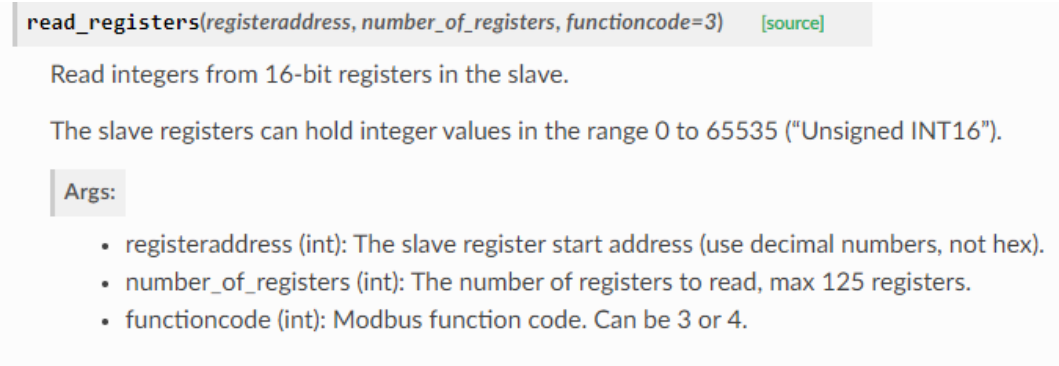
The slave register can hold integer values in the range 0 to 65535 ("Unsigned INT16").

Args:

- registeraddress (int): The slave register address (use decimal numbers, not hex).
- number_of_decimals (int): The number of decimals for content conversion.
- functioncode (int): Modbus function code. Can be 3 or 4.
- signed (bool): Whether the data should be interpreted as unsigned or signed.

- Para la lectura de varios registros (usado en la lectura de ángulos):

Ilustración 12 - Función read_registers de minimalmodbus



read_registers(*registeraddress, number_of_registers, functioncode=3*) [source]

Read integers from 16-bit registers in the slave.

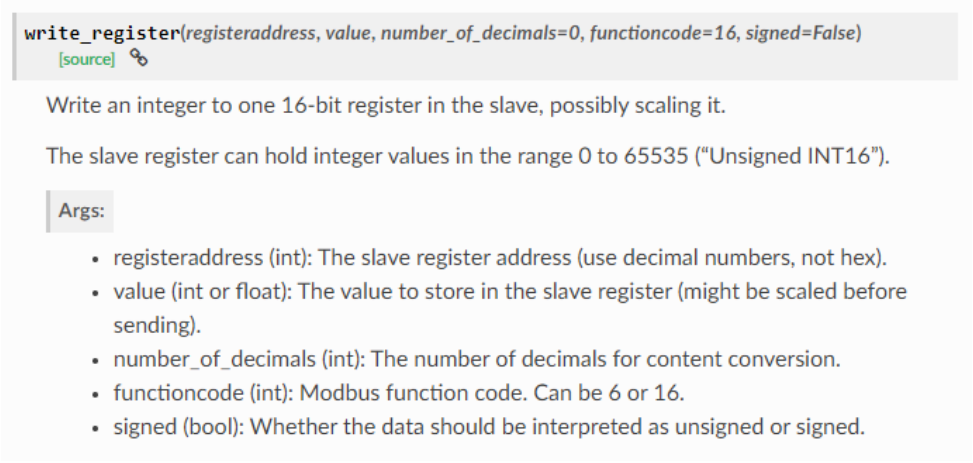
The slave registers can hold integer values in the range 0 to 65535 ("Unsigned INT16").

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- number_of_registers (int): The number of registers to read, max 125 registers.
- functioncode (int): Modbus function code. Can be 3 or 4.

- Escritura de un registro:

Ilustración 13 - Función write_register de minimalmodbus



`write_register`(*registeraddress, value, number_of_decimals=0, functioncode=16, signed=False*)
[source]

Write an integer to one 16-bit register in the slave, possibly scaling it.

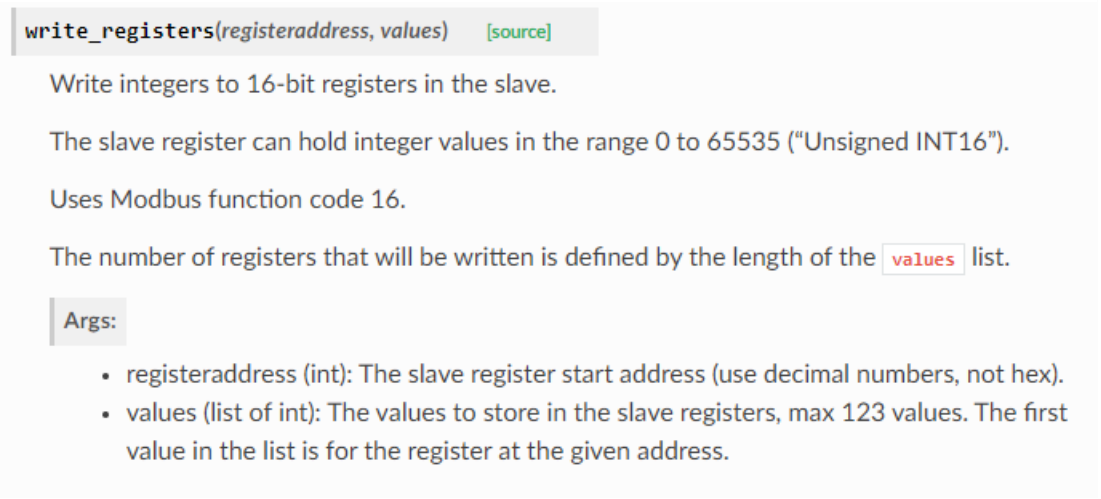
The slave register can hold integer values in the range 0 to 65535 ("Unsigned INT16").

Args:

- registeraddress (int): The slave register address (use decimal numbers, not hex).
- value (int or float): The value to store in the slave register (might be scaled before sending).
- number_of_decimals (int): The number of decimals for content conversion.
- functioncode (int): Modbus function code. Can be 6 or 16.
- signed (bool): Whether the data should be interpreted as unsigned or signed.

- Escritura de varios registros:

Ilustración 14 - Función write_registers de minimalmodbus



`write_registers`(*registeraddress, values*) [source]

Write integers to 16-bit registers in the slave.

The slave register can hold integer values in the range 0 to 65535 ("Unsigned INT16").

Uses Modbus function code 16.

The number of registers that will be written is defined by the length of the `values` list.

Args:

- registeraddress (int): The slave register start address (use decimal numbers, not hex).
- values (list of int): The values to store in the slave registers, max 123 values. The first value in the list is for the register at the given address.

Con la lectura y escritura de varios registros en una misma petición, se optimiza la velocidad de respuesta de los esclavos porque la carga de bytes debidos al header y el checksum es menor.

5.5.3.4. Concurrencia

Para facilitar las peticiones, se programan unas funciones que escriben o leen en la base de datos y devuelven el valor del registro Modbus que han pedido. Dentro de estas funciones se ha configurado una condición de semáforo de lectura y otra de escritura para sincronizar la rutina modbus con estas funciones. En el siguiente trozo de código se puede ver la lógica del semáforo en la función de lectura de registros:

Ilustración 15 - Función read_modbus

```
def read_modbus(register, address):
    #Adquiere la condición de semáforo
    global_config.read_buffer_cond.acquire()

    #Escribe en la base de datos
    read_register = ReadBufferModbus(address=address, registro=register, value=0, leido=False)
    read_register.save()
    idr = read_register.id

    #Condicion de espera para leer en rutina modbus
    global_config.read_buffer_cond.wait(timeout=3)
    read_obj = ReadBufferModbus.objects.filter(id=idr)
    value = read_obj[0].value

    #Devuelve la condición de semáforo
    global_config.read_buffer_cond.release()
    return int(value)
```

En la parte de la rutina modbus también se implementa un semáforo para lectura y otro para escritura de registros de la siguiente forma:

Se adquiere la condición del semáforo al inicio de la parte de escritura de registros:

Ilustración 16 - Condición de semáforo

```
if(write_buffer):
    global_config.write_buffer_cond.acquire()
    num_data = len(write_buffer)
    print(str(num_data) + " registros por escribir...")
```

Se notifica a los hilos en espera cuando se ha escrito el registro sin errores y se libera la condición del semáforo.

Ilustración 17 - Condición de semáforo release

```
if(wr_error == 0):
        global_config.write_buffer_cond.notify()

global_config.write_buffer_cond.release()
```

Estos objetos de condición de semáforo se inicializan en un script externo (*global_config.py*), de esta manera se pueden utilizar como variables globales sobre varios scripts de forma sincronizada. Se inicializan los objetos de condición junto con las variables de los hilos de la siguiente forma:

Ilustración 18 - Fichero global_config.py

```
1
2   import threading
3
4   #GLOBAL VARS
5   rutina_modbus = ''
6   rutina_offset = ''
7   rutina_tunning = ''
8   output_line = 'hola'
9
10  read_buffer_cond = threading.Condition()
11  write_buffer_cond = threading.Condition()
12  read_angle_ok = threading.Event()
13  solatom_modbus = []
14  modbus_cycle = 0.5 #Tiempo entre esclavos de modbus
15  modbus_wait_instrument = 3 #Tiempo de espera cuando se crea el instrumento modbus
16  modbus_read_wait = 0.1 #Tiempo de espera entre peticiones de lectura
17  modbus_write_wait = 0.1 #Tiempo de espera entre peticiones de escritura
18
```

5.5.3.5. Gestión de errores

Según la documentación de la librería minimalmodbus determina los posibles errores que pueden surgir durante la comunicación modbus. Por lo tanto, para evitar la finalización del hilo, se recoge esta serie de errores con funciones *try except.*

- *Minimalmodbus.NoResponseError:* este error salta al hacer una petición al instrumento minimalmodbus y no recibir respuesta durante el tiempo determinado en el timeout de los parámetros modbus. En este caso se ha establecido un timeout de 2 segundos.

Por lo tanto, para recoger este error se usa un try except siempre que se utilicen las funciones de lectura o escritura, como se puede ver en el siguiente fragmento de código:

Ilustración 19 - Gestión de errores write buffer

```
for data in write_buffer:
    #time.sleep(0.5)
    if(data.enviado == False):
        if(data.value < 0):
            data.value = 65536 + data.value
        try:
            global_config.solatom_modbus.write_register(data.registro, data.value, 0)
            data.enviado = True
            data.save()

            #No error
            modbus_conf = ParametrosModbus.objects.last()
            modbus_conf.error = 0
            modbus_conf.save()
        except minimalmodbus.NoResponseError as e:
            print(e)
            wr_error = 1

            if(tries_w < 3):
                tries_w +=1
            else:
                tries_w = 1
                global_config.solatom_modbus.serial.close()
                modbus_conf = ParametrosModbus.objects.last()
                modbus_conf.error = 2
                modbus_conf.abierto = False
                modbus_conf.save()
                break
```

Una vez se da este error, se escribe una variable para que a la tercera vez que suceda el error se cierre el puerto serial y se reinicie la comunicación.

- *Minimalmodbus.InvalidResponseError:* este error se da cuando la trama de datos de la comunicación modbus no tiene el formato correcto o el checksum no es correcto. Para capturar este error, se recoge toda la función de rutina modbus en un try except y se gestiona de la siguiente forma:

Ilustración 20 - Gestión error Invalid Response

```
except minimalmodbus.InvalidResponseError as e:
    global_config.solatom_modbus.serial.close()
    modbus_conf = ParametrosModbus.objects.last()
    modbus_conf.error = 1
    modbus_conf.abierto = False
    modbus_conf.save()
    try:
        global_config.write_buffer_cond.release()

    except RuntimeError as e:
        print("RUNTIME ERROR WRITE 1")
        print(e)

    try:
        global_config.read_buffer_cond.release()

    except RuntimeError as e:
        print("RUNTIME ERROR WRITE 2")
        print(e)
```

Se cierra el puerto serial y se cierra la comunicación para volver a establecerla, además de liberar las variables de condición de semáforo.

- *serial.serialutil.SerialException:* este error se da cuando el instrumento creado por minimalmodbus no puede acceder al puerto serial que se le ha especificado, por lo que se usa un try except al crear el instrumento. Este error se gestiona, cerrando la comunicación serial e impidiendo que se ejecute el resto del hilo hasta que se reinicie.

Para la gestión de las diferentes tareas y el feedback para la interfaz gráfica del estado de la comunicación, existen una serie de variables de error que cambian su valor para que en el caso de que salte un try except, la tarea del hilo no se ejecute hasta que vuelva a haber comunicación. Este código de error está definido de la siguiente forma:

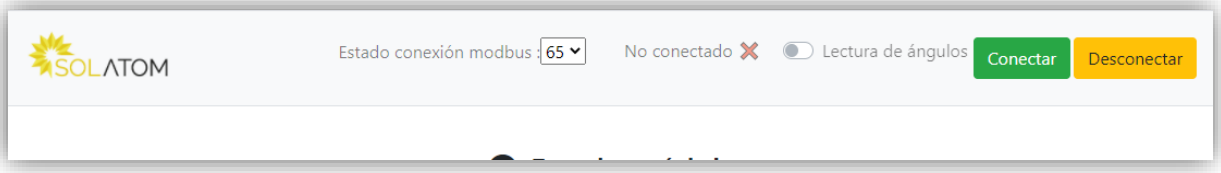
Tabla 3 - Siginificado error de comunicación Modbus

Código de error	Significado del error	Tipo de error
0	No hay error de ningún tipo	-
1	No se pudo abrir el puerto especificado	*serial.serialutil.SerialException*
2	Error de escritura (No communication with the instrument: No answer)	*minimalmodbus.NoResponseError*
3	Error de lectura (No communication with the instrument: No answer)	*minimalmodbus.NoResponseError*
4	Error de lectura y escritura	*minimalmodbus.NoResponseError*

5.5.3.6. Lanzamiento y fin del hilo

Para lanzar y acabar el hilo se utilizan los botones situados en la parte superior de la página web con el nombre de "*Conectar"* y *"Desconectar"*.

Ilustración 21 - Menú superior Conectar y Desconectar



- *Conectar:*

Al presionar el botón de "*Conectar"* se hace una petición AJAX al script *header_connect_modbus,* la cual ejecuta la función *check_rutina_modbus().* Esta función comprueba si el hilo se ha lanzado y lo lanza en el caso de que no esté ejecutándose.

- *Desconectar:*

El botón de *"Desconectar"* también realiza una petición AJAX pero al script *end_rutina_modbus.* Esta función hace un *kill* al hilo de rutina Modbus y espera a que se cierre con un *join.* También pone todos los objetos del modelo *ParametrosModbus* de comunicación a cerrado.

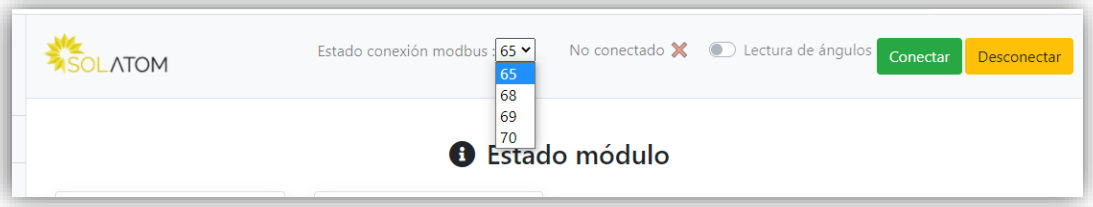
5.5.4. Vista de "Ajustes Modbus"

En este apartado, se va a explicar el funcionamiento de la vista de "Ajustes Modbus" y cómo se lanza y se detiene el hilo de rutina modbus. Como se ha explicado antes en el apartado 5.3.1 Instalación y estructura de archivos, el navegador hace una petición a una url, la cual ejecuta un script de Python (una vista) y devuelve una plantilla renderizada.

Al script de la vista se le pasa una variable *request* (de tipo 'GET' o 'POST') con la información que pide el navegador web. Por lo tanto, según el tipo de *request* se puede cambiar el funcionamiento de la vista y lo que se renderiza en la página web. Los navegadores web, cuando hacen una petición para abrir una URL, normalmente es de tipo *GET*.

Pero antes de comprobar el tipo de request realizado a la página, se obtienen los puertos seriales abiertos del dispositivo y se añaden a las opciones del formulario. También se recogen las instancias del modelo ParametrosModbus y se obtiene una lista de direcciones de esclavos para pasarlo a la cabecera de la página web. De esta forma, se puede controlar qué módulo se está controlando para realizar las calibraciones o leer y escribir registros.

Ilustración 22 - Selección de módulo menú superior



5.5.4.1. Request 'GET':

Quiere decir que ha sido el navegador el que ha hecho la petición y por lo tanto se crea un formulario con las opciones de parámetros de conexión Modbus. Estos parámetros se auto completan con el último valor escrito en la base de datos de ParametrosModbus y se añaden al formulario para ser renderizado.

Django tiene la opción de crear una clase de tipo formulario en el script *forms.py* y cargarlo como una variable de Python a la plantilla HTML para que se renderice como HTML simplificando la cantidad de código HTML a escribir. A este formulario se le pasan los puertos seriales abiertos para añadirlos a las opciones del parámetro 'COM'. La clase de tipo form se crea de la siguiente forma:

Ilustración 23 - Formulario Ajustes Modbus

```
class ModbusConnectForm(forms.Form):
    baudrate_choice = [('9600', '9600'), ('19200', '19200')]
    parity_choice = [('0', 'None'), ('1', 'Odd parity'), ('2','Even parity')]
    ini_com = ''
    ini_slave = 0
    def __init__(self,*args,**kwargs):
        self.ports = kwargs.pop('ports')
        self.modbus = kwargs.pop('modbus')
        super(ModbusConnectForm,self).__init__(*args,**kwargs)
        print(self.modbus)

        self.fields['coms'].choices=self.ports
        self.fields['coms'].initial=self.modbus.com
        self.fields['slaveid'].initial=self.modbus.slave

    coms = forms.ChoiceField(label="Elige un puerto ")
    slaveid = forms.CharField(label="Slave ID ", max_length=5)
    baudrate = forms.ChoiceField(label="Baudrate ", choices=baudrate_choice, initial='19200')
    time_out = forms.DecimalField(label="Timeout (sec) ", initial=2)
    parity = forms.ChoiceField(label="Parity ", choices=parity_choice, initial='2')
```

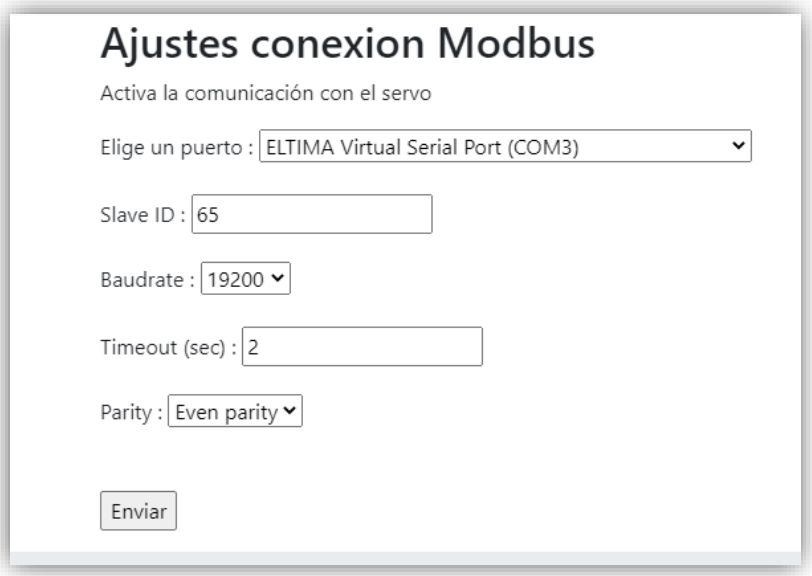
Por lo tanto, una vez se ejecuta el script, se crea una instancia de la clase *ModbusConnectForm* y se pasa a la plantilla HTML a renderizar como la variable *form*.

Ilustración 24 - HTML del formulario Ajustes Modbus

```
<form action="{% url 'ajustes_modbus' %}" method="post">
  {% csrf_token %}
  <p>Activa la comunicación con el servo</p>


  {{ form }}

  <br>

  <input type="submit" id="id_btn_enviar" value="Enviar">


</form>
```

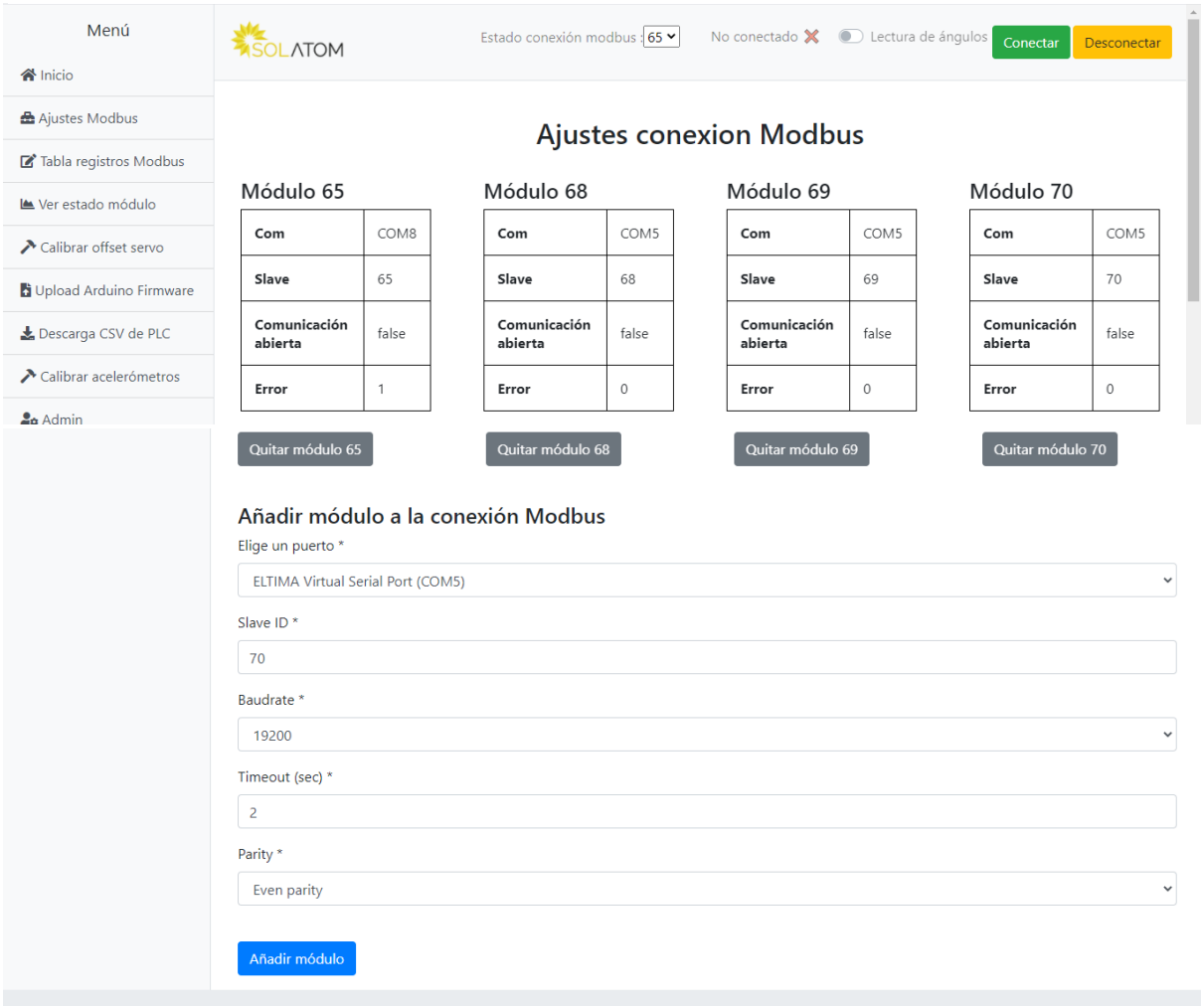
En la siguiente imagen se puede ver el HTML renderizado con el formulario en la web:

Ilustración 25- Formulario HTML renderizado



Además del formulario, se puede observar el estado de los módulos añadidos para realizar la comunicación Modbus. Estos se pasan del modelo de la base de datos ParametrosModbus como una lista y se pueden ver en la página web de la siguiente forma:

Ilustración 26 - Vista Ajustes Modbus



Desde esta página se puede gestionar los módulos a los que se conecta el algoritmo. Al final de la página se encuentra el formulario para añadir módulos y debajo de cada módulo un botón para quitarlo.

5.5.4.2. Javascript

- *Quitar módulo:*

Al pulsar el botón de "Quitar módulo", este módulo se elimina de la página y de la base de datos. Esto se realiza con Javascript para la parte gráfica y una petición AJAX (librería de jQuery) hacia una url del servidor.

El código de Javascript está configurado por las variables de Python, por lo tanto, dependiendo del número de módulos en la base de datos, se crea un EventListener para cada uno. En la siguiente imagen se puede ver la forma en que se escribe el código Javascript, dependiendo de si hay módulos en module_list y por cada módulo en esta lista:

Ilustración 27 - Variables Python en el HTML

```
{% if module_list %}

{% for module in module_list %}
var quit_module_{{ module.slave}} = document.getElementById("id_quitar_modulo_{{module.slave}}");
console.log(quit_module_{{ module.slave}});
quit_module_{{ module.slave}}.addEventListener('click', function() {
  document.getElementById("id_div_modulo_{{ module.slave }}").remove()
  //Petición al servidor para borrar el módulo
  $.ajax({
      url: '/rw/ajax/delete_module/',
      method: 'POST',
      data: {
        'module':{{ module.slave }},
      },
      dataType: 'json',
      success: function (data) {
        console.log(data)
      }});
});

{% endfor %}
```

El método de AJAX realiza una petición 'POST' a la url '/rw/ajax/delete_module/' y le pasa la dirección del módulo a eliminar. Para esta url, se ha creado la vista en Django *ajax_delete_module* que recibe esta petición y elimina el módulo de la base de datos del modelo *ParametrosModbus*.

- *Feedback de conexión Modbus:*

Una vez añadidos los módulos necesarios y con la conexión Modbus establecida se puede observar el estado de conexión de cada módulo gracias a las peticiones que hace el código Javascript con AJAX al servidor. Este código también está controlado por el script de Python, por lo que se escribe de forma automática al renderizar la página.

Ilustración 28 - Función Ajax Javascript

```
window.setInterval(function () {
  //call your function here
  $.ajax({
    url: '/rw/ajax/ajustes_modbus/',
    data: {
      'state': 0
    },
    dataType: 'json',
    success: function (data) {
      //console.log(data)
      module_len = Object.keys(data.modules).length;
      //console.log(module_len);
      //Variables de control de Python
      {% for module in module_list %}

      for(var i = 0; i<module_len;i++){
        if(data.modules[i].slave == {{ module.slave }}){
          // console.log("OKEY {{ module.slave }}");
      //Cambio del texto del HTML de cada módulo
        document.getElementById("id_com_{{ module.slave}}").innerHTML = data.modules[i].com;
        document.getElementById("id_slave_{{ module.slave}}").innerHTML = data.modules[i].slave;
        document.getElementById("id_abierto_{{ module.slave}}").innerHTML = data.modules[i].abierto;
        document.getElementById("id_error_{{ module.slave}}").innerHTML = data.modules[i].error;
        }
      }
      {% endfor %}
      }
    });
}, 2000);
{% endif %}
```

Esta petición de AJAX se envía a la vista *ajax_ajustes_modbus* que devuelve en formato JSON una lista con los valores del modelo *ParametrosModbus* (que se actualiza durante la ejecución de la rutina Modbus). El código de Javascript lo recibe y actualiza los valores en la página web sin necesidad de refrescar la página. Los datos recibidos tienen el siguiente formato:

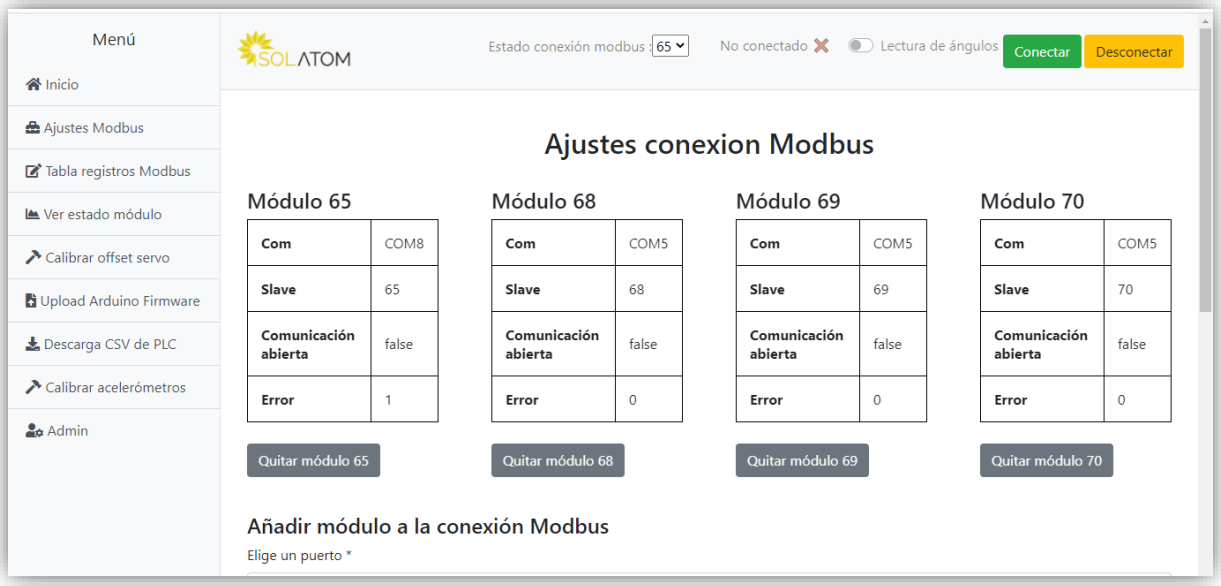
{'mod_65': {'com': 'COM8', 'slave': 65, 'abierto': True, 'error': 0, 'read_angle': True, 'running': False},

'mod_66': {'com': 'COM8', 'slave': 66, 'abierto': False, 'error': 0, 'read_angle': True, 'running': False}}

5.5.4.3. Request 'POST'

Esta petición se realiza completando el formulario y presionando el botón *"Añadir módulo"*. De esta forma se recarga la página y se pasan los datos del formulario al script del servidor. Este los valida y lo añade a la base de datos, después recoge todos los módulos que existan en la base de datos y recarga la página con estos, por lo que aparece un módulo más con los datos introducidos en el formulario.

Ilustración 29 - Vista Ajustes Modbus



5.6. Escritura y lectura de registros

En este apartado se comenta cómo se ha desarrollado la parte de lectura y escritura de registros.

5.6.1. Objetivo y planteamiento de soluciones

El objetivo de este apartado era desarrollar una interfaz atractiva y fácil para leer y escribir registros de los diferentes módulos mediante comunicación Modbus. Una vez desarrollada la rutina Modbus, este apartado se debe encargar de escribir en el buffer de datos (de lectura o escritura) la petición de registros y mostrar el resultado.

5.6.1.1. Primera solución: dos formularios

La primera iteración utiliza dos formularios renderizados en la misma página donde cada

Ilustración 30 - Primera versión de escritura y lectura de registros

uno escribe en un buffer determinado. El problema de esta solución es que, al enviar un registro, se inhabilita el otro formulario, por lo que se debe recargar la página para poder habilitarlo, perdiendo el resultado del anterior. Además, resulta poco atractivo y difícil de usar.

5.6.1.2. Segunda solución: mejora de Modbus Poll

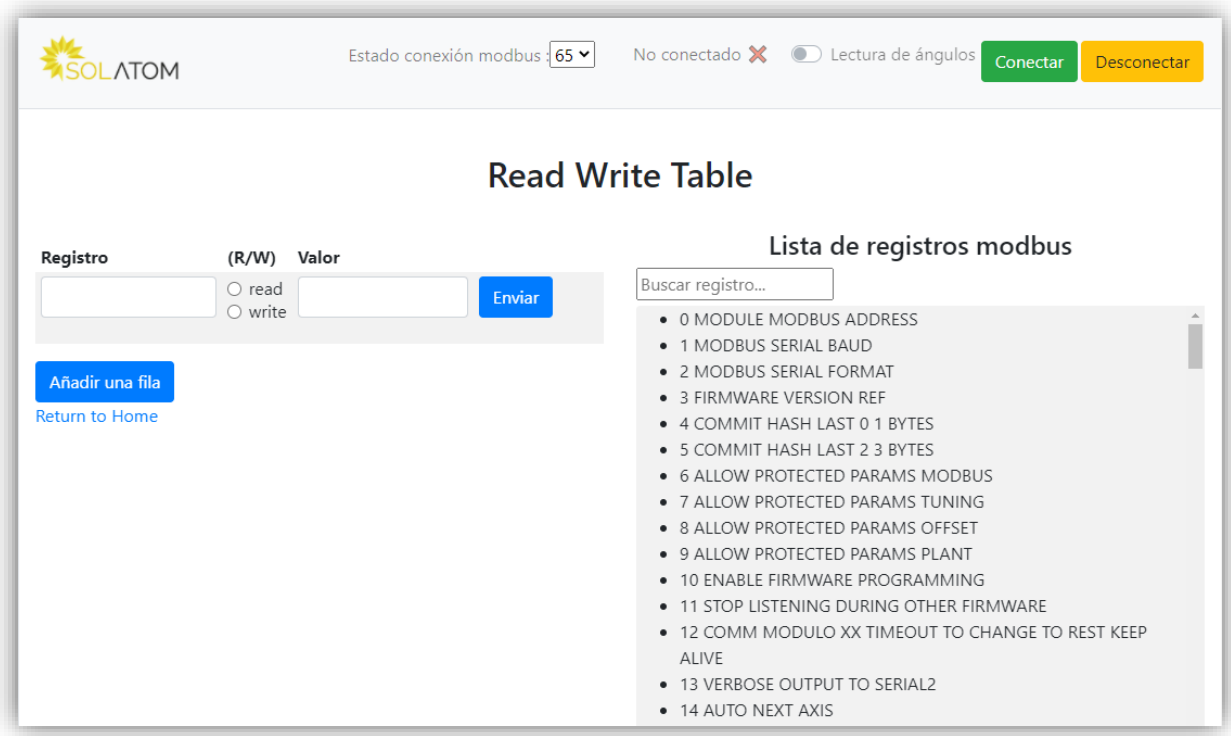
La aproximación final a la que se ha llegado nace inspirada al comportamiento de la aplicación de testeo de conexión Modbus conocida como Modbus Poll (Modbus Tools, n.d.) pero teniendo ventajas respecto la versión comercial como son el acceso remoto común a todo el proyecto y la posibilidad de buscar y seleccionar por nombre los registro Modbus y ver sus características (unidad de medida y rangos aceptables).

Con esta solución se pueden añadir los registros que sean necesarios y escribir o leer al registro necesario sin la necesidad de recargar la página en ningún momento, por lo que se pueden preparar varios registros en filas distintas y escribirlos con un simple click. Esta solución de describe en los siguientes apartados.

5.6.2. Interfaz

La interfaz se ha realizado con HTML y Javascript, dando el siguiente resultado:

Ilustración 31 - Segunda versión tabla lectura y escritura



Se trata de una interfaz limpia y visual con una tabla a la izquierda, donde se pueden enviar registros para leer o escribir y añadir las filas que sean necesarias. A la derecha está la lista de registros ordenados para poder consultar el nombre de cada uno. También se ha añadido un buscador para facilitar la búsqueda de los registros.

5.6.2.1. HTML

En cuanto a la parte de HTML, se usan las clases de Bootstrap para organizar el layout de la página y para añadir un toque visual.

Se trata de una tabla a la izquierda que se rellena con los apartados de una clase de tipo formulario que se crea en el script de Python y se pasa a la plantilla HTML. Se realiza de la siguiente forma:

Ilustración 32 - Tabla HTML

```
<tr>

  <td>{{ form.register | as_crispy_field}} </td>
  <td><form action="">{{ form.read_write | as_crispy_field}} </form> </td>
  <td>{{ form.value | as_crispy_field}} </td>
  <td><button type="submit" class="btn btn-primary" id="id_enviar">Enviar</button> </td>
  <td></td>
</tr>
```

De esta forma se puede seleccionar dónde se renderizan los elementos del formulario de forma individual y la parte de "*as_crispy_field*" es un filtro para mejorar el aspecto de los formularios.

En la parte derecha, se añade la lista de todos los registros existentes en los módulos solares mediante un "*container"* restringido en tamaño, lo que provoca que tenga su propio *scroll.* Gracias a las variables de Python se puede escribir toda la tabla con un bucle for dependiendo del tamaño de la lista que cree el script.

Ilustración 33 - Lista de registros modbus

```
<div class="container col-sm-4 ml-sm-10" >
  <!-- <div class=" container "> -->
    <h2>Lista de registros modbus</h2>
    <input type="text" id="id_search" onkeyup="searchRegFunction()" placeholder="Search.." title="Type in a category">
    {% if lines %}
    <div class="overflow-auto" style="height: 500px; width: 100%;">
      <ul id="id_regList">
    {% for line in lines %}
      <li>
        <a>{{ line }}</a>
      </li>

      {% endfor %}
      </ul>
    </div>
    {% endif %}
  <!-- </div> -->
</div>
```

También se añade un elemento de tipo *input* para que con Javascript se ejecute la función para buscar el registro necesario rápidamente.

5.6.2.2. Javascript

A continuación, se explican las funciones que hace el Javascript y cómo se han desarrollado:

- *Cuando se quiere hacer una lectura y se pulsa read, desaparece el campo de valor hasta que se recibe el valor. Cuando se pulsa write, vuelve a aparecer el campo de valor.*

Lo primero que se realiza es identificar los botones de *read* y *write* mediante el DOM del documento. Se pueden identificar por tipo de elemento, por clase o por ID, en este caso se identifican por ID.

Para cada botón se le añade un EventListener que detecta si se realiza la acción que se le indica y ejecuta una función determinada. En este caso las funciones son *hideValue* y *showValue*, cuyo funcionamiento es el de buscar el elemento input de propia la fila y cambiarle la propiedad de visibilidad. Las funciones son las siguientes:

Ilustración 34 - Funciones Javascript

```
function showValue(){

    //Muestra el campo de valor
    var td = this.parentNode.parentNode.parentNode.parentNode.parentNode;
    console.log(td);
    console.log( td.nextElementSibling);
    td.nextElementSibling.getElementsByTagName('input')[0].style.setProperty('visibility', "visible");

}

function hideValue(){

  //Esconde el campo de Valor
  var td = this.parentNode.parentNode.parentNode.parentNode.parentNode;
  console.log(td);
  console.log( td.nextElementSibling);
  td.nextElementSibling.getElementsByTagName('input')[0].style.setProperty('visibility', "hidden");

}
```

- *Al pulsar el botón "Añadir nueva fila" se añade una fila a la tabla con las mismas funcionalidades que las anteriores.*

Para ello, se añade un *EventListener* al botón de "Añadir una fila" con el evento de 'click' y la función de *appendRow*.

Esta función busca el id de la tabla y añade una fila a las que ya tiene. Después para cada celda de la nueva fila inserta el elemento correspondiente para imitar la primera fila. Esto se ha realizado de forma manual con Javascript puro, puesto que para poder añadir el *EventListener,* el nuevo elemento se debe crear de una determinada forma y con variables de Python no lo reconocía.

Por lo tanto, se crea una función para cada elemento, la cual crea el elemento nuevo, le añade el estilo y el id necesario y también las funcionalidades del resto de filas. Resulta relativamente sencillo con las clases de Javascript, aunque un poco tedioso. En la siguiente imagen se puede ver la función que crea el botón de enviar a modo de ejemplo.

Ilustración 35 - Función createButton Javascript

```
function createButton(cell) {
    var button = document.createElement('button');
    button.type = "submit";
    button.className = " btn btn-primary";
    button.id = "id_enviar";
    button.innerText = "Enviar";
    button.addEventListener('click', sendReg);
    cell.append(button);
}
```

- *Al pulsar "Enviar" se envía la petición de lectura o escritura de la fila y se devuelve el valor leído en ese registro.*

A cada botón "Enviar" se le añade el *EventListener* con el evento 'click' y la función *sendReg.* Esta función busca los elementos de la misma fila según la estructura de padres e hijos del DOM y recoge sus valores.

Después inicia el *spinner* de pensando y envía una petición AJAX con estos valores a la url '/rw/ajax/table_registers/' que recoje la vista *ajax_read_write_table_registers* del servidor, la cual se explica en el siguiente apartado.

- *En la lista de registros se muestran los registros que coincidan con los caracteres de la búsqueda.*

Se crea la función *searchRegFunction,* que busca de entre la lista de variables de registros las que coincidan con los términos de búsqueda y deshabilita el resto para que sólo se vean los registros con coincidencias. Esta función se añade al evento '*onKeyUp'* que quiere decir al soltar una tecla.

Ilustración 36 - Función de buscar en lista de registros Javascript

```
function searchRegFunction() {
// Declare variables
var input, filter, ul, li, a, i;
input = document.getElementById("id_search");
filter = input.value.toUpperCase();
ul = document.getElementById("id_regList");
li = ul.getElementsByTagName("li");

// Loop through all list items, and hide those who don't match the
for (i = 0; i < li.length; i++) {
  a = li[i].getElementsByTagName("a")[0];
  if (a.innerHTML.toUpperCase().indexOf(filter) > -1) {
    li[i].style.display = "";
  } else {
    li[i].style.display = "none";
  }
}
}
```

5.6.3. Script de Python

5.6.3.1. Vista

La parte de la vista de la página web es manejada por el script *table_registers_view*. Este script realiza una instancia del formulario de la clase *ReadWriteTableForm* para renderizar los campos de cada fila.

Ilustración 37 - Formulario ReadWriteTable

```
class ReadWriteTableForm(forms.Form):
    rw_choices = [(0,'read'), (1, 'write')]
    register = forms.DecimalField(label=False)
    read_write = forms.ChoiceField(choices=rw_choices, widget=forms.RadioSelect(), label=False)
    value = forms.DecimalField(required=False, label=False)
```

Para la parte de la lista de registros, abre el archivo modbus_reg.py donde están declaradas las variables de los registros Modbus, formatea cada línea para que sea más visual y devuelve la plantilla HTML con el formulario, la lista de registros y los esclavos Modbus de la cabecera.

Ilustración 38 - Script Python table_registers_view

```
def table_registers_view(request):
    header = get_header_address()
    check_rutina_modbus(1)
    form = ReadWriteTableForm()
    f_modbus_path = sys.path[0] + "\\rw_arduino\\modbus_reg.py"

    f=open(f_modbus_path, 'r')
    lines_trans = []
    with f:
        print("Abriendo archivo modbus_reg")
        lines = f.readlines()
        #print(lines)
        for line in lines:
            line = line.replace("_"," ")
            for word in line.split():
                try:
                    num = int(word)
                except ValueError:
                    pass
            line = line.replace("=","")
            line = line.replace(str(num), "")
            line = str(num) + " " + line
            #print(line)
            lines_trans.append(line)

    return render(request, 'table_read_write.html', {'form':form, 'lines':lines_trans, 'header_address':header})
```

5.6.3.2. AJAX

Este script recibe la petición del Javascript al pulsar el botón de enviar con los valores de registro y lectura o escritura. Según se trate de lectura o escritura, ejecuta una función que escribe en uno de los modelos de la base de datos (*ReadBufferModbus* o *WriteBufferModbus*) y devuelve una respuesta en formato JSON con el valor de lectura del registro aunque se haya escrito para verificar la escritura.

Ilustración 39 - Función respuesta de Ajax

```
def ajax_read_write_table_registers(request):
    if request.method == 'POST':
        #username = request.GET.get('username', None)
        register = request.POST.get('register')
        read_write = request.POST.get('read_write') # 0 = read, 1 = write
        address = request.POST.get('address')
        print(str(read_write)+ " address "+ address)
        try:
            value = request.POST.get('value')
        except ValueError as e:
            print(e)

        if(read_write == '0'): #Read
            print(str(read_write)+ ' read')
            rw = 'R'
            value_rw = read_modbus(register, address)
        else: #Write
            print(str(read_write)+ ' write')
            rw = 'W'
            value_rw = write_modbus_fb(register, value, address)

        data = {
        'register' : register,
        'read_write': rw,
        'value' : value_rw,
        'address': address,

        }
        print(str(data))
        return JsonResponse(data)
```

5.7. Rutina automática de calibración del offset de los servos

En este apartado se comenta el proceso de calibración del offset de los servos que controlan los ejes de los espejos solares. También se comenta cómo se ha automatizado este proceso y añadido al proyecto.

5.7.1. Objetivo y posibles soluciones

Los servos son controlados en velocidad por pulsos PWM, de manera que pulsos de 1.00 ms dan la velocidad máxima hacia un lado, y con 2.00 ms todo hacia el otro lado. A 1.50 ms deberían estar parados, pero cada motor tiene un pequeño offset (error) de fábrica. El control interno de movimiento de los espejos tiene un offset software para contrarrestar este error, ajustado durante la construcción del módulo solar. El proceso manual de calibrado de este offset es largo y necesita un operario atento.

La solución consiste en una rutina que automatice este proceso, liberando el operario que realiza la rutina de forma manual y pudiendo dedicarse a otras tareas.

5.7.2. Funcionamiento manual

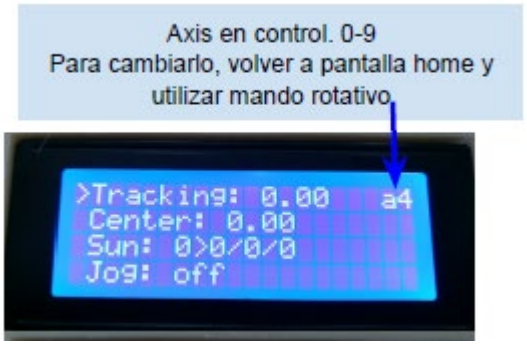
Cada eje de los módulos solares está controlado por un servo y un acelerómetro calcula el ángulo de inclinación del eje. Estos dos elementos deben ser calibrados para que la precisión de apuntado de los espejos sea correcta.

La calibración del servo se basa en ajustar el valor de offset de la velocidad de giro, ya que al estar controlados por velocidad, a velocidad cero no deben moverse. Debido a la zona muerta de los motores, debe haber un rango de valores de velocidad en los que el servo no se mueve, normalmente suele estar entre -3 y 3 de jog speed(unidad usada para la calibración en que cada step de jog speed representa 10 microsegundos).

Normalmente, la calibración se realiza durante la instalación y comissioning de los módulos y se debe hacer por cada módulo de forma manual, lo que conlleva un gasto importante de tiempo, ya que cada módulo son 10 espejos y las plantas solares suelen tener de 5 a 20 módulos.

La forma de ajustar los ejes es utilizar el mando de control, con el que se determina la velocidad a 0 y se observa si cambia el valor de ángulo o visualmente si el eje se mueve. Entonces se ajusta dándole un valor positivo o negativo de jog hasta que el eje deja de moverse y se guarda este valor como offset del eje.

Ilustración 40 - Pantalla mando de control LCD



En la imagen anterior se puede ver la pantalla del mando, con el que se ajusta el valor de offset en el apartado de "Jog: off". Una vez seleccionado, se usa la rueda del mando para controlar el valor de jog y observar el giro del eje. En la parte superior derecha se puede ver el eje para el que se está ajustando el offset.

5.7.3. Rutina automática

Para el desarrollo de la rutina automática se necesita conocer el funcionamiento interno de la electrónica del módulo y de los registros, puesto que la calibración se realizará leyendo y escribiendo diversos registros Modbus de forma continuada y tomando decisiones según las lecturas.

En primer lugar, en la siguiente imagen están los registros de protección de escritura de los registros de ajuste de offset como son el 8 ("allow protected params offset"), 9 ("allow protected params plant") y 10 "allow protected params plant". A estos registros se les escribe un valor de *timeout* y mientras no acabe este tiempo se pueden escribir los registros que protegen. También se escribe el registro 12 con un valor de *timeout* muy alto para evitar que los ejes entren en modo reposo y el registro 14 a OFF para evitar que cambie de eje sin que lo controle el programa de calibración.

Tabla 4 - Registros Modbus de protección

0	Module Modbus Address	R/W	modbus	-	-	0x41 (ascii	0xEF
1	Modbus Serial Baud	R/W	modbus	see table			
2	Modbus Serial format	R/W	modbus	see table			
3	Firmware version ref.	R		progressive			
4	commit hash last -0 -1 bytes	R					
5	commit hash last -2 -3 bytes	R					
6	allow protected params modbus	R/W		timeout s	10	0	32767
7	allow protected params tuning	R/W		timeout s	10	0	32767
8	allow protected params offset	R/W		timeout s	10	0	32767
9	allow protected params plant	R/W		timeout s	10	0	32767
10	enable firmware programming	R/W	modbus	timeout s	10	0	32767
11	stop listening during other firmware	R/W	modbus	timeout s	10	0	1800
12	comm MODULO.xx timeout to change to rest (keep alive)	R/W	modbus	s		1	32767
13	verbose output to Serial2	R/W	modbus	-		0	1
14	auto_next_axis	R/W	modbus	ON/OFF		0	1

En la siguiente tabla se encuentra la lista de registros que guardan el valor de offset de los servos:

Tabla 5 – Registros Modbus valor de offset de velocidad

120	servo offset de velocidad (axis 0)	R/W	tuning	us	1
121	servo offset de velocidad (axis 1)	R/W	tuning	us	1
122	servo offset de velocidad (axis 2)	R/W	tuning	us	1
123	servo offset de velocidad (axis 3)	R/W	tuning	us	1
124	servo offset de velocidad (axis 4)	R/W	tuning	us	1
125	servo offset de velocidad (axis 5)	R/W	tuning	us	1
126	servo offset de velocidad (axis 6)	R/W	tuning	us	1
127	servo offset de velocidad (axis 7)	R/W	tuning	us	1
128	servo offset de velocidad (axis 8)	R/W	tuning	us	1
129	servo offset de velocidad (axis 9)	R/W	tuning	us	1

También se necesitan los valores de lectura de ángulos de los ejes:

Tabla 6 - Registros Modbus de ángulo actual

210	angle actual (axis 0)	R	deg.	100	-18000	18000
211	angle actual (axis 1)	R	deg.	100	-18000	18000
212	angle actual (axis 2)	R	deg.	100	-18000	18000
213	angle actual (axis 3)	R	deg.	100	-18000	18000
214	angle actual (axis 4)	R	deg.	100	-18000	18000
215	angle actual (axis 5)	R	deg.	100	-18000	18000
216	angle actual (axis 6)	R	deg.	100	-18000	18000
217	angle actual (axis 7)	R	deg.	100	-18000	18000
218	angle actual (axis 8)	R	deg.	100	-18000	18000
219	angle actual (axis 9)	R	deg.	100	-18000	18000

Para seleccionar el eje a calibrar se utiliza el registro 164 (axis in control):

Tabla 7 - Registro Modbus eje de control

164	axis in control	R/W (actua	plant	axis	0	9

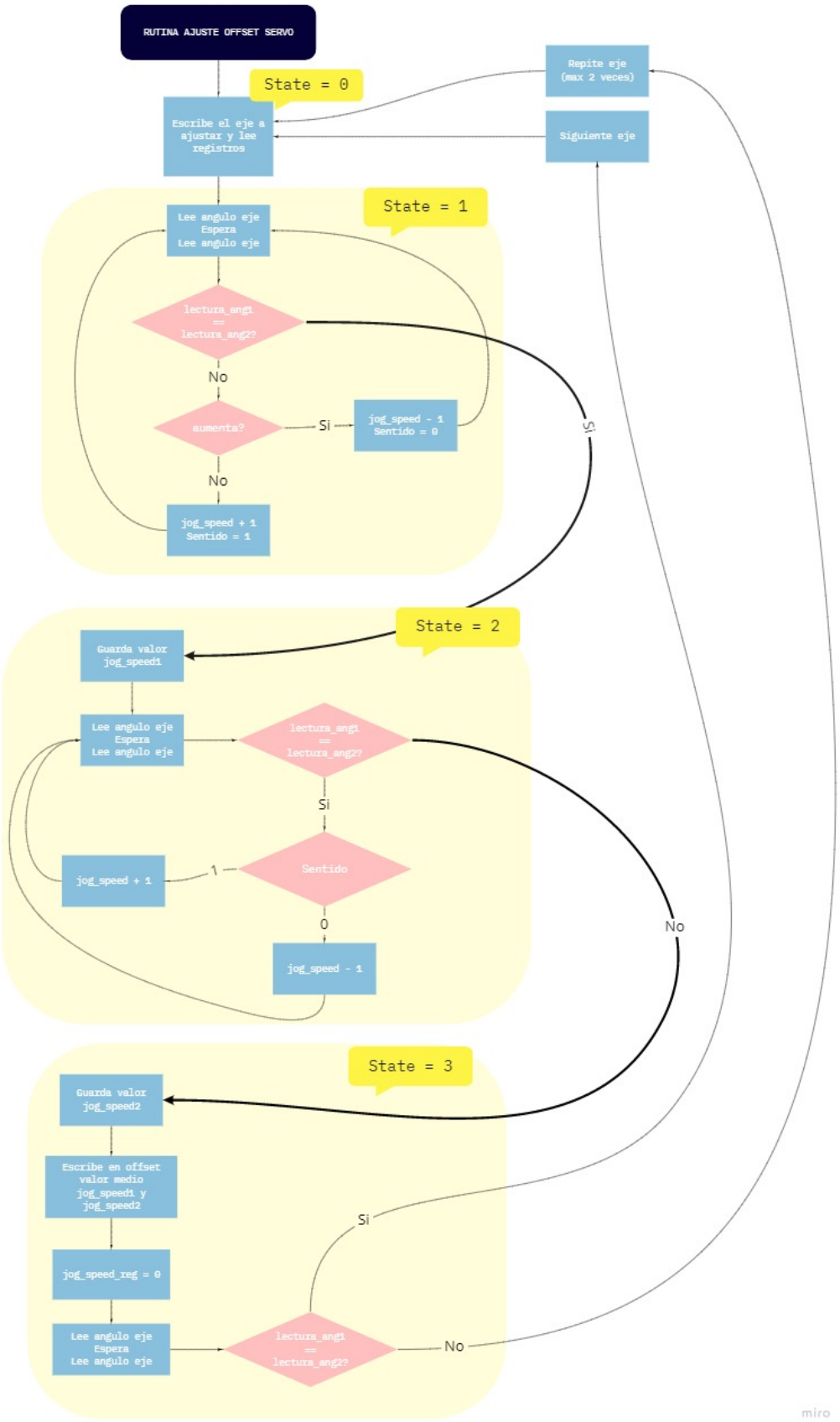
Por último, se necesitan estos dos registros para controlar la velocidad de giro de los servos (242) y el tiempo de giro (243) del eje que esté seleccionado mediante el registro 164 (axis in control).

Tabla 8 - Registros Modbus jog speed y start jog

242	jog speed	R/W		servo speed	10	-50	50
243	START jog of current axis	R/W	plant	timeout s	10	0	32767

La lógica del funcionamiento se describe en el siguiente esquema de la siguiente imagen y se divide en 4 estados:

Ilustración 41 - Lógica Rutina ajuste de offset



- *Estado 0:* en este estado se escriben los registros de protección y se deshabilita el "auto next axis" y el "rest position". También se pone el offset a cero y se inicia el giro para así observar el sentido de giro en jog 0.
- *Estado 1:* gira en un sentido mientras que dos lecturas del ángulo en un intervalo de tiempo sean diferentes, es decir, que el eje esté girando. En cada iteración resta o suma 1 al valor de jog speed hasta que se para (depende del sentido de giro con jog 0). Guarda el valor de jog en el que se para.
- *Estado 2:* una vez se para, se sigue sumando o restando valores al jog hasta que comienza a girar otra vez. Guarda el valor de jog en el que comienza a moverse otra vez.
- *Estado 3:* con los dos valores de estado 1 y estado 2, se hace la media para tener el valor de jog de offset. Se comprueba escribiendo el offset y con velocidad 0. Si no se mueve el eje, se da por válido y si no repite el proceso para el mismo eje un máximo de 3 veces hasta declararlo en estado de fallo.

Este proceso se repite para todos los ejes del módulo con un bucle for y se va guardando el proceso y los estados por los que pasa en la base de datos para tener registro de cuanto tarda en hacer cada eje y si hay algún fallo.

Al final la calibración, se vuelven a escribir los valores normales de *"auto next axis"* y *"rest position"*.

5.7.4. Interfaz

Para la vista de "Rutina ajuste servo", se programa un formulario con una lista de opciones como *CheckboxSelectMultiple* para los ejes del módulo (de 0 a 9) que al cargar la página salen todos seleccionados y la opción del módulo a calibrar.

Ilustración 42 - Vista lanzamiento rutina ajuste de offset

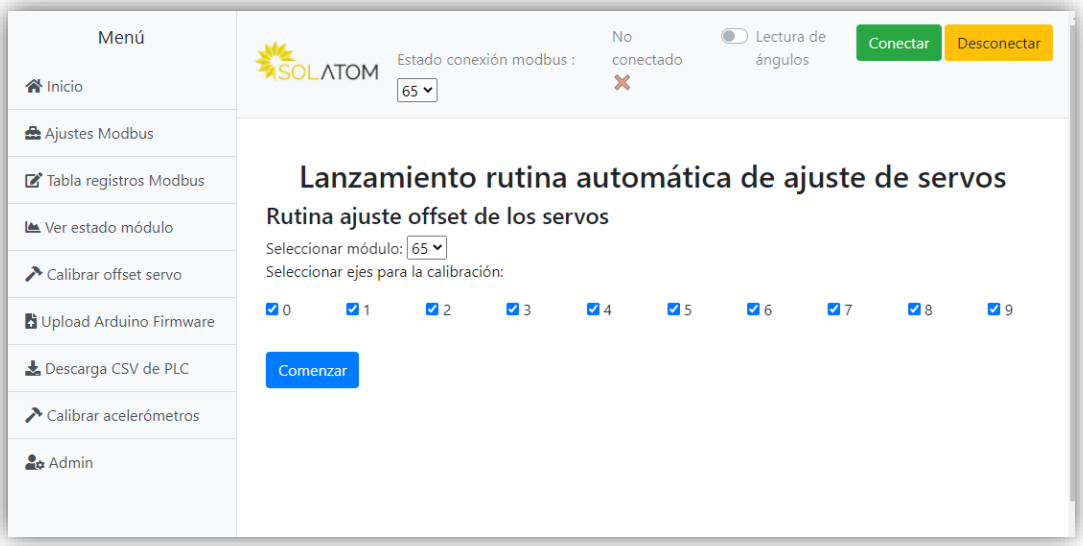


Ilustración 43 - Formulario OffsetServoForm

```
class OffsetServoForm(forms.Form):

    def __init__(self,*args,**kwargs):
        super(OffsetServoForm,self).__init__(*args,**kwargs)
        print(self.fields['axis'])
        axis_choice = [(0,0), (1, 1), (2, 2), (3, 3),(4, 4),(5, 5),(6, 6),(7, 7),(8, 8),(9, 9)]
        self.fields['axis'].choices = axis_choice
        self.fields['axis'].initial = [choice[0] for choice in axis_choice]
```

Al pulsar el botón de "Comenzar", se realiza una petición de tipo 'POST' que gestiona el mismo script que genera el formulario inicial. Lo que hace es lanzar un hilo con la función que se ha explicado en el apartado anterior y redirigir a una página para poder ver el estado de calibración de los ejes.

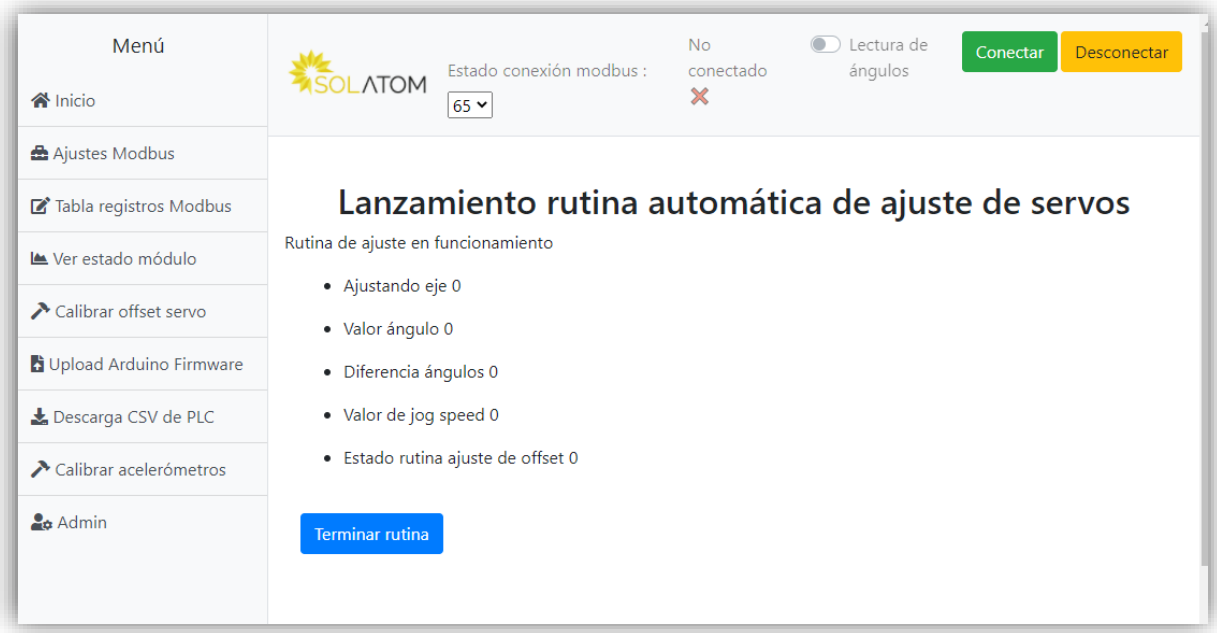
El estado de calibración se realiza con javascript y Ajax, de la misma forma que el resto de las páginas. El Ajax realiza una petición al servidor a la url "/rw/ajax/rutina_offset/" y que gestiona la vista *ajax_rutina_offset.* Esta vista lo que hace es recoger la última fila de la base de datos de estado de calibración y la devuelve en formato JSON.

Ilustración 44 - Función de respuesta Ajax Rutina offset

```
def ajax_rutina_offset(request):
    #username = request.GET.get('username', None)
    data = {
        'eje' : AjusteOffsetVelocidad.objects.last().eje,
        'angulo' : AjusteOffsetVelocidad.objects.last().angulo,
        'deriva' : AjusteOffsetVelocidad.objects.last().deriva,
        'jog_speed' : AjusteOffsetVelocidad.objects.last().jog_speed,
        'estado' : AjusteOffsetVelocidad.objects.last().estado,
        'running' : AjusteOffsetVelocidad.objects.last().running
    }
    return JsonResponse(data)
```

La función de Ajax, recoge la respuesta y actualiza la vista para refrescar los valores y tener seguimiento del estado de la función de calibración de los servos.

Ilustración 45 - Vista de feedback de rutina offset servo



5.8. Actualización del firmware de la planta

5.8.1. Objetivo y posibles soluciones

Uno de los aspectos más importantes es tener la posibilidad de actualizar el firmware de la electrónica de los módulos de forma remota. De momento esto se realiza mediante una placa de desarrollo propia con un relé que activa la funcionalidad de reseteo de la electrónica del módulo para poder actualizar el firmware.

El **problema** es que, actualmente, esta placa debe estar conectada al ordenador vía USB y después con cable UTP a los módulos. El ordenador tenía que correr la IDE de Arduino así que aun queriendo hacerlo de forma remota con una raspberry, hay que gastar una pesada conexión grafica tipo *remote desktop*.

Además, para cargar el firmware, se debe habilitar la comunicación en modo full-duplex, mientras que de normal los módulos funcionan en half-duplex, enviando previamente un comando serial a la “placa roja”.

Hay que habilitar manualmente los registro Modbus para que el Arduino del módulo solar que se quiere actualizar, acepte un nuevo firmware (todos los módulos están en el mismo bus Modbus)

Ilustración 46 - PCB roja de activación de actualización de firmware

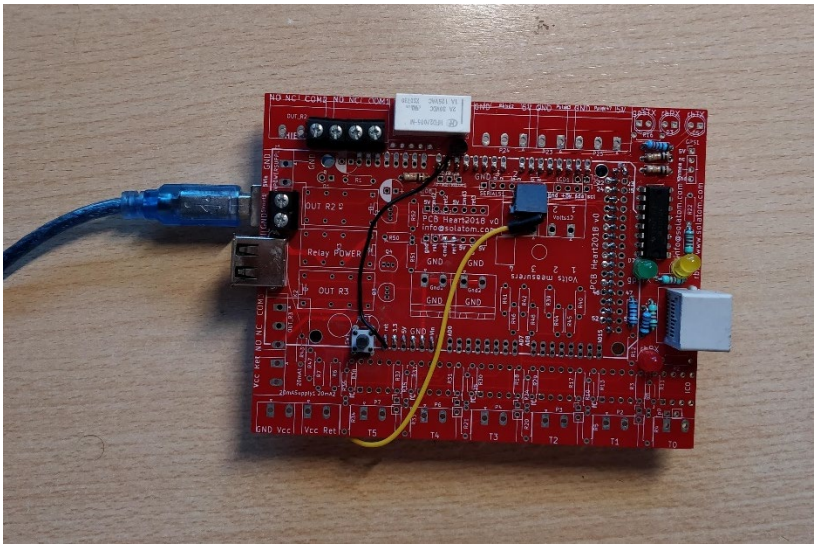
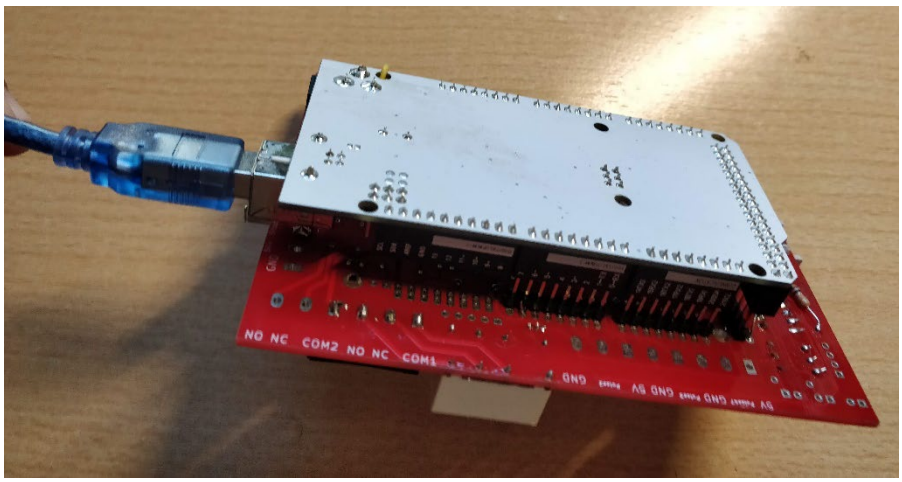
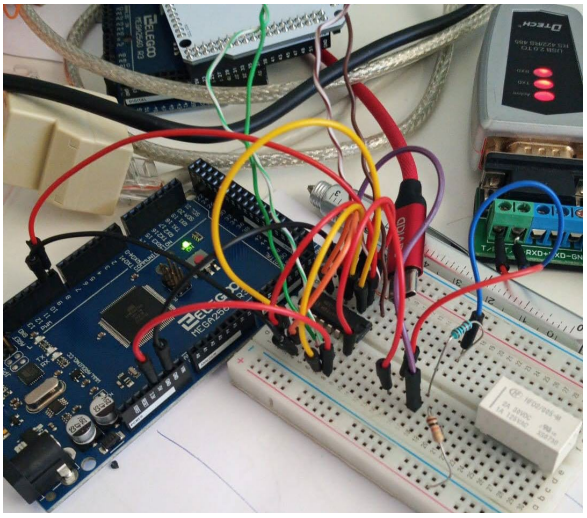


Ilustración 47 - Arduino de la PCB roja de activación de actualización de firmware



Sin embargo, ya está en desarrollo una forma de activar el reseteo de los módulos de forma remota con una placa igual acoplada al cuadro eléctrico y conectada por USB a la Raspberry que hace de pantalla y servidor web.

Ilustración 48 - Prototipo de nueva PCB de actualización de firmware



El comando avrdude (Nongnu Org, n.d.) es el encargado de escribir el programa en el Arduino, por lo que si no se utiliza un terminal directamente para el upload con avrdude, se pierde el feedback de cómo iba el upload al arduino, que era una necesidad de los operadores. Utilizando la librería de subprocess de Python se ha conseguido renderizar el mismo output de avrdude en la visualización web.

La posibilidad de actualizar el firmware fácilmente por operadores no especializados conlleva la responsabilidad de *trackear* con mucha fiabilidad la versión de firmware. A ese objecto, se ha añadido un automatismo (*hook*) en arduino ide para que escriba una variable en el mismo firmware con el valor del *commit* GIT (control de versión), para que sea disponible

en lectura de registro Modbus 4 y 5. La compilación del *hex* prevee que el nombre del fichero indique el mismo valor de *commit.*

5.8.2. Rutina actualización del firmware

La rutina de actualización de firmware automatiza todos estos pasos: manda el comando de actuación a la placa roja de desarrollo para que active el relé y habilita la comunicación a los módulos, escribir los registros del módulo seleccionado y ejecutar el comando para el envío del programa.

El firmware se puede cargar directamente como fichero hex precompilado con Arduino IDE anteriormente.

Esta función recibe de la vista los siguientes parámetros:

- *Port:* puerto en el que está conectada la placa roja.
- *Module_address:* id del módulo al que se le va a actualizar el firmware.
- *Operation:* existe la opción de escribir o leer (en este caso comprueba si el programa subido y el que tiene es el mismo).
- *Path:* esta es la ruta interna donde está ubicado el archivo de firmware de tipo .hex.
- *Name:* nombre del archivo.

5.8.2.1. Funcionamiento

Lo primero que hace es abrir el puerto serial que se la pasado y conectarse al Arduino rojo para activar el relé con el comando determinado. Al acabar cierra el puerto serial.

Después, crea un instrumento de comunicación Modbus de la misma forma que la rutina modbus (aunque está se acaba al iniciar la rutina de upload). Con este instrumento activa los registros necesarios para la subida del programa y comprueba que se han escrito correctamente. Estos registros son:

Tabla 9 - Registros de activación de actualización firmware

5	commit hash last -2 -3 bytes	R					
6	allow protected params modbus	R/W		timeout s	10	0	32767
7	allow protected params tuning	R/W		timeout s	10	0	32767
8	allow protected params offset	R/W		timeout s	10	0	32767
9	allow protected params plant	R/W		timeout s	10	0	32767
10	enable firmware programming	R/W	modbus	timeout s	10	0	32767
11	stop listening during other firmware	R/W	modbus	timeout s	10	0	1800

Una vez está preparado el Arduino para recibir el programa, se lanza el comando avr con la configuración del Arduino de los módulos y la opción de leer o escribir. Para ello se usa la librería *subprocess* de Python, la cual nos permite lanzar estos procesos y leer la salida estándar (stdout) y de error (stderr) mientras se ejecutan. La salida se pasa a una variable global que se usa para actualizar el estado de la interfaz.

Una vez subido, se cierra el puerto serial y se acaba el hilo.

5.8.3. Interfaz

La interfaz es un formulario de tipo *DocumentForm* y tiene los siguientes campos:

Ilustración 49 - Vista Upload Arduino Firmware

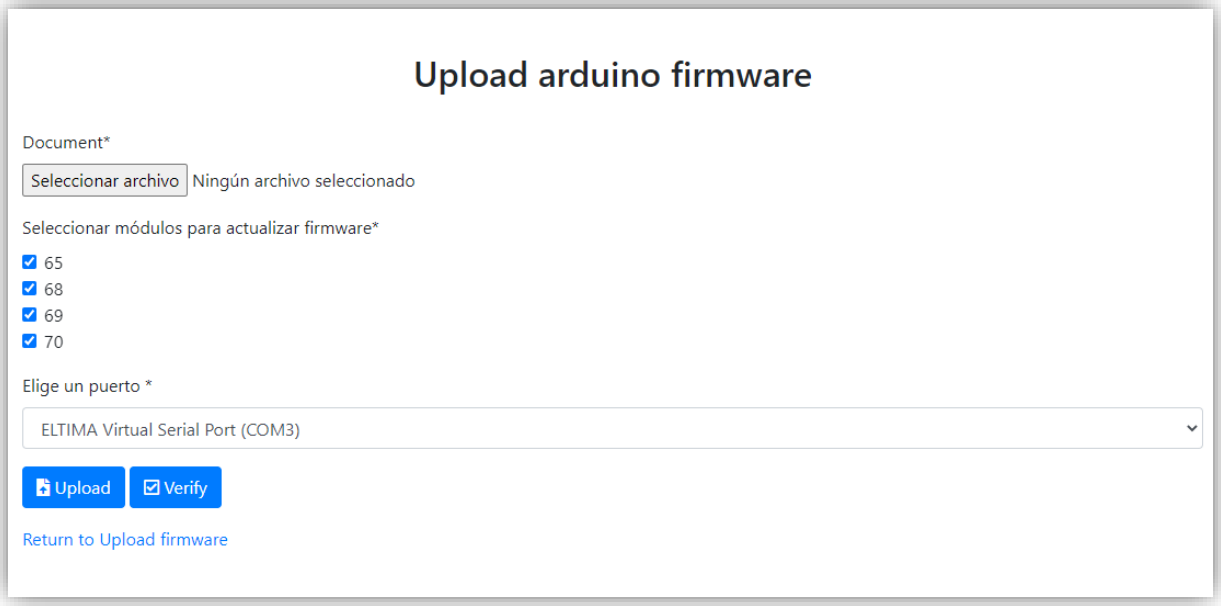


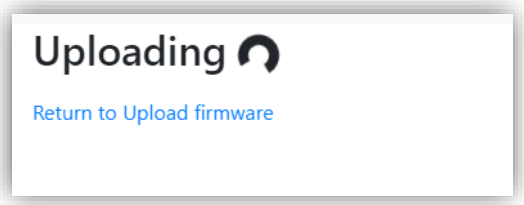
Ilustración 50 - Formulario de Upload Firmware

```
class DocumentForm(forms.ModelForm):
    def __init__(self,*args,**kwargs):
        self.ports = kwargs.pop('ports')
        super(DocumentForm,self).__init__(*args,**kwargs)

        self.fields['coms'].choices=self.ports

    module_address = forms.DecimalField(label="Module address ", initial='65')
    coms = forms.ChoiceField(label="Elige un puerto ", initial='COM25')
    #operation = forms.CharField(label="Acción a ejecutar (v = verificar, w = escribir) ", initial='w')
    class Meta:
        model = ArduinoFirmware
        fields = ['document', ]
```

A este formulario también se le pasan los puertos seriales abiertos como al formulario de "Ajustes Modbus". Y además de puerto y dirección tiene un campo para subir el archivo hexadecimal de la compilación del firmware de Arduino.

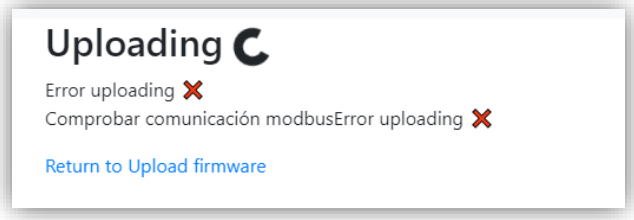
Al rellenar los campos y subir el programa, se pulsa el botón de "Upload" si se quiere actualizar el programa o "Verify" para comprobar si el programa subido es correcto.

Ilustración 51 - Feedback de Upload Firmware



En el caso de que haya algún problema de comunicación con el instrumento, aparece este mensaje:

Ilustración 52 - Error de Upload Firmware



Si la comunicación Modbus funciona y el programa se puede subir correctamente irá apareciendo la salida del comando avr en la página de la siguiente forma:

Ilustración 53 - Feedback del terminal avr

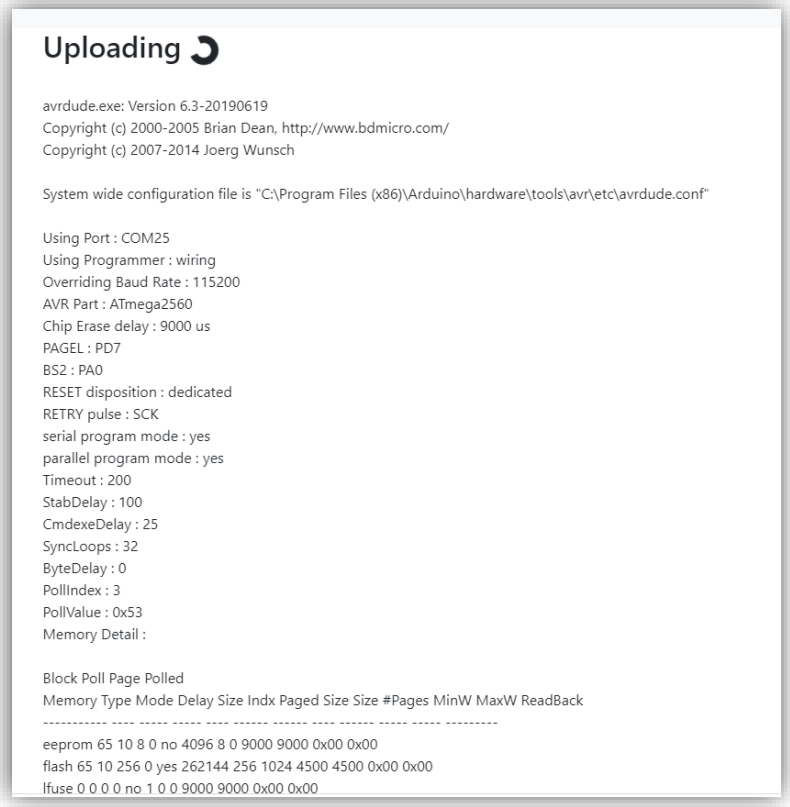
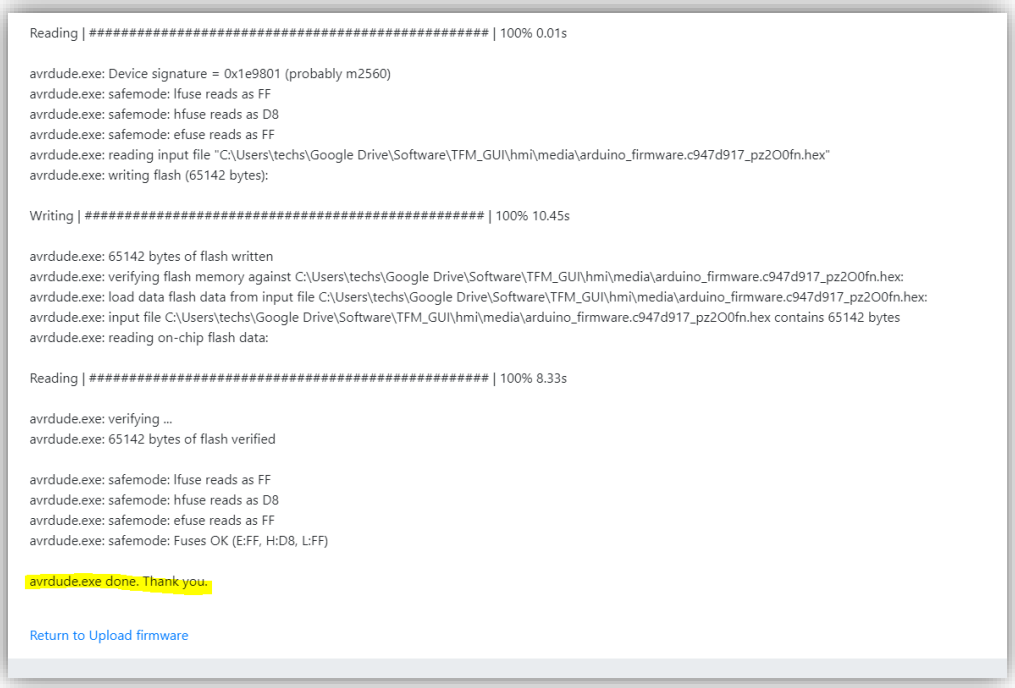


Ilustración 54 - Final de feedback del terminal avr



Con el último mensaje de “avrdude.exe done. Thank you.” se puede verificar que se ha subido el programa satisfactoriamente.

Para conseguir que la salida del proceso se muestre por pantalla se usa Javascript con Ajax sobre la variable global de Python que se actualiza con cada línea de salida del proceso avr. El script de Ajax es el siguiente:

Ilustración 55 - Script de Ajax de Javascript

```
<script>
  if(document.getElementById("id_upload")){


    window.setInterval(function () {
      /// call your function here
      $.ajax({
          url: '/rw/ajax/terminal_feedback/',
          data: {
            'line': 0
          },
          method: 'GET',
          dataType: 'json',
          success: function (data) {


            document.getElementById("id_upload").innerHTML= data.line;
            //document.getElementById( "id_upload" ).scrollIntoView();
            if(document.getElementById("id_upload").innerHTML.includes("avrdude.exe done.")){
              document.getElementById("id_h2_uploading").innerHTML= "Uploaded correct ✓";

            }else if(document.getElementById("id_upload").innerHTML.includes("timeout communicating with programmer")){
              document.getElementById("id_h2_uploading").innerHTML= "Uploaded fail ✗";
            }
          }
        });

    }, 1000);
  }
</script>
```

Y esta la respuesta del servidor, que devuelve la variable global en formato JSON:

Ilustración 56 - Respuesta del servidor de Ajax

```
def ajax_terminal_feedback(request):
    # line = ''
    # if(str(global_config.output_line)=="b''"):
    #     pass
    # else:
    #     line = str(global_config.output_line)

    line = global_config.output_line
    data = {
        "line" : line,
    }
    return JsonResponse(data)
```

5.9. Descarga CSV de datos

5.9.1. Objetivo y posibles soluciones

5.9.1.1. Problema actual

El *webcontroller* del PLC que actualmente está montado en varias plantas no permite una descarga inteligente de csv, hay que abrir un entorno gráfico, seleccionar los datos, elegir el tiempo (que no puede ser un día ad hoc o un intevalo de tiempo concreto, sino hoy, ayer, esta semana, semana pasada, este mes, mes pasado, todo), y luego descargar el csv. El entrono es lento y consume muchos datos (en las plantas hay contratos 3G a datos limitados).

5.9.1.2. Solución 1

Afortunadamente el *webcontroller* permite el acceso REST a sus variables, pero la descarga es como objectos *bacnet* en formato *json*, así que por un valor *float* de 4 byte por ejemplo, hay que descargar un *json* de 500 caracteres (500 bytes inútiles). siendo por cada variable 1440 *samples* al día (1/min), y alrededor de 30 variables, los datos a descargar intercambiar son 20 Mb, que tiene que preparar y enviar el *webcontroller* por cada petición.

5.9.1.3. Solución 2

Corriendo este proyecto en el servidor de la raspberry de cada planta, esta petición REST se puede realizar en local, siendo más rápida y sin consumo de datos, y el algoritmo inteligente confecciona un CSV de pocos kb con todos los datos necesarios.

5.9.1.4. Solución 3

Aunque se realiza en conexión local, las peticiones siguen siendo lentas, así que se cambia de una descarga directa del fichero CSV mediante el browser, a un envío de este por email en que simplemente, mediante un formulario se registra el email y los parámetros de tiempo.

5.9.1.5. Solución 4

Habilitando en el mismo servidor de la raspberry unas vistas para llamadas POST, el servidor central de solatom puede disparar el envío automático de CSV por email a destinatarios predefinidos para tener un seguimiento diario de los datos de las plantas.

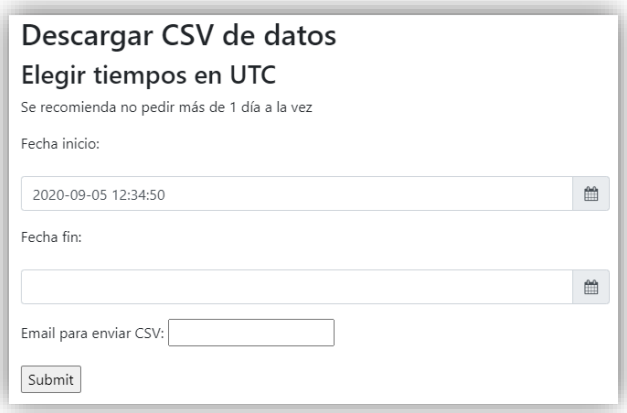
Después de varias iteraciones y alternativas, se pasa a explicar cómo se ha solucionado el objetivo de descargar datos de funcionamiento de los módulos solares.

5.9.2. Interfaz

Para la interfaz se usa un formulario, declarado en el script forms.py como la clase *LogDataAPIPLCForm,* con los siguientes campos:

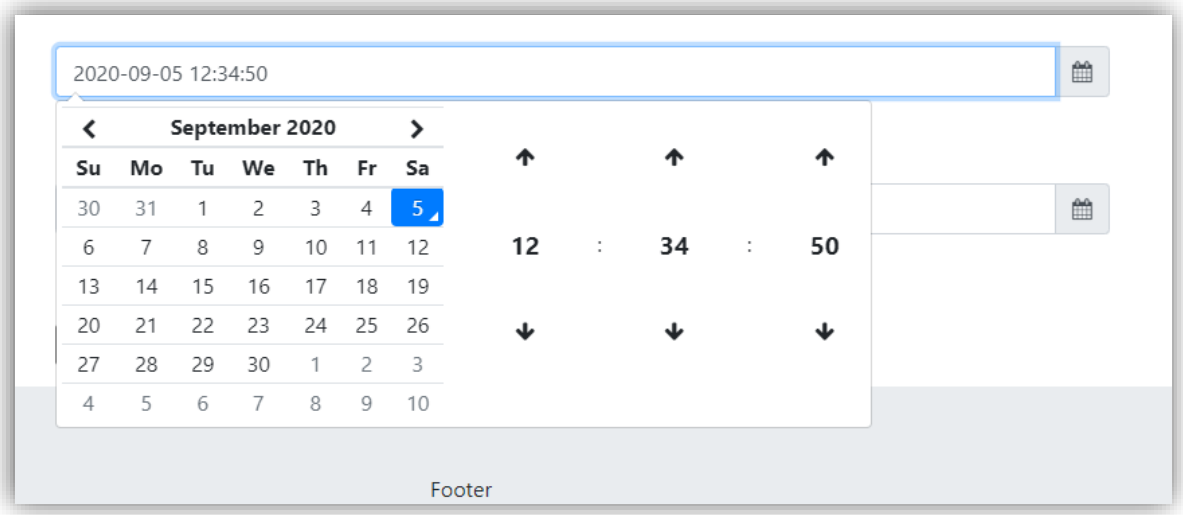
Ilustración 57 - Formulario de Descargar CSV de datos

```
class DjangoTempusDominusForm(forms.Form):

    fecha_inicio = forms.DateTimeField(
        widget=DateTimePicker(
            options={
                'useCurrent': True,
                'collapse': False,
                'sideBySide': True,
            },
            attrs={
                'append': 'fa fa-calendar',
                'icon_toggle': True,
            }
        ),
    )
    fecha_fin = forms.DateTimeField(
        widget=DateTimePicker(
            options={
                'useCurrent': True,
                'collapse': False,
                'sideBySide': True,
                #'timeZone': 'Europe/Madrid'
            },
            attrs={
                'append': 'fa fa-calendar',
                'icon_toggle': True,
            }
        ),
    )
    mail_to = forms.CharField(label="Email para enviar CSV")
```



Al hacer click en la fecha de inicio o fecha fin aparece un calendario para seleccionar el día y la hora.

Ilustración 58 - Widget de DjangoTempusDominus



Para mostrar el calendario se ha usado un widget de Django llamado *DjangoTempusDominus* y se programa en la declaración de la clase en *forms.py*.

También existe un campo para introducir el correo al que se enviará el archivo CSV una vez se hayan procesado los datos.

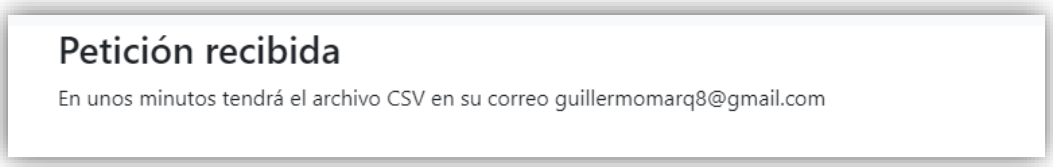
Al lanzar el formulario con los campos seleccionados, se lanza un hilo que gestiona la descarga de datos, el procesamiento y el envío del correo.

Ilustración 59 - Lanzamiento de Hilo de API

```
#Lanzamiento de hilo de API data
rutina_get_api = get_api_data_thread('Hilo API data',start_date, stop_date, mail_to)
rutina_get_api.start()
```

Al hilo se le pasan los datos del formulario y envía a una página de respuesta como la siguiente:

Ilustración 60 - Página de respuesta de Descarga de CSV



5.9.3. Método de descarga

Se ha usado la API (*Application Programming Interfaces*) del propio PLC de la marca Eclypse (Eclypse, n.d.) que realiza el control industrial de las plantas. Al consultar la documentación del PLC, se observa que se puede recoger la información de las variables que controla el PLC mediante peticiones tipo *GET* y *POST*. La estructura de la petición debe contener la dirección IP del PLC, la variable de petición de datos y el intervalo de tiempo que se desee.

Ilustración 61 - URL de la API del PLC

```
GET /protocols/bacnet/local/objects/{objectType}/{objectInstance}/trend
```

Por lo tanto, se procede a realizar un script que realice esta petición y descarga de los datos de forma automática. Para simplificar el proceso, se hace una petición de todas las variables del PLC para así tener una visión general del estado de la planta solar.

Se hace uso de un Excel en el que están descritos los registros de las variables del PLC para crear una lista en Python sobre la que iterar y realizar peticiones de todas las variables. De esta forma si en un futuro se cambian los registros, esta aplicación seguirá funcionando sin necesidad de cambiar nada. La librería usada para leer el archivo Excel es *csv* de Python.

Con la lista de variables realizada, se realiza un bucle que hace la petición por cada variable creando la URL con el tiempo solicitado del formulario de la página. Para la ip del PLC existen varias opciones según la planta que se trate. Esto se debe a que, si la raspberry está en la misma red local que el PLC, la ip del PLC es fija de valor "192.168.0.30". En el caso de que sea necesario hacer la petición de forma remota se usan las ips de la imagen ya que las plantas están redirigidas a un servidor de la empresa que hace de túnel para realizar estas operaciones (esta parte se desarrolla en el punto 5.12 Conexión remota).

Sin embargo, cuando las raspberrys de las plantas dispongan de esta aplicación todas tendrán la ip local para minimizar el consumo de datos.

Ilustración 62 - IP del PLC

```
#IPs de las plantas
#ip_planta = "192.168.0.30" #local
ip_planta = "51.75.121.235:33800" #unex
#ip_planta = "51.75.121.235:33801" #itc canarias
#ip_planta = "51.75.121.235:33802" #solpinvap
url = "http://"+ ip_planta +"/api/rest/v1/protocols/bacnet/local/objects/trend-log/"+str(trend_
```

5.9.4. Gestión de la respuesta

Una vez realizada la petición, se recibe la respuesta del PLC en formato JSON y debe ser tratada para recoger los datos necesarios y exportarlos a un formato con el que se pueda trabajar, en concreto en formato CSV.

El problema que se encuentra es que hay diferencias entre algunas variables en cuanto al tiempo de muestreo por lo que para el mismo intervalo de tiempos hay más datos de unas variables que de otras. Es un problema porque el resultado debería ser un CSV con una columna de timestamp para todas las variables, por lo que se gestionará de la siguiente forma:

- Se crea una lista de Python con intervalos de 1 segundo entre los tiempos introducidos por el ususario.
- Se hace un bucle para esta lista de timestamp global y se comprueban los timestamps de las variables.
- Para cada variable y cada instante de timestamp global, se busca **el mínimo de los mayores al timestamp global**. De esta forma se mantiene el valor de la variable mientras que no haya uno nuevo.
- Si no existe ningún valor que cumpla la condición anterior, se añade el anterior valor. Y si no hay valor anterior se añade un cero, que quiere decir que no hay valores anteriores a ese tiempo.

Con la lista de valores manejados según el timestamp global se crea un CSV con estos datos y se envía por correo al proporcionado por el usuario.

5.9.5. Envío diario automático de CSV

Para realizar un seguimiento y recopilación de los datos de las plantas se ha diseñado una vista que tramita peticiones POST y lanza el algoritmo que envía el CSV por email. Esta vista se diseña para que desde el servidor central de Solatom se puedan realizar peticiones diarias y registrar los datos en formato CSV de ese día.

El funcionamiento es el siguiente:

- *Recoge los datos de la petición POST*
- *Lanza el hilo con la función de recogida de datos de la API*
- *Devuelve como respuesta un JSON con los datos de la petición POST*

Esta vista presenta el filtro @csrf_exempt (Ionos, 2020) para que no se necesite autenticación token y se pueda pedir desde el servidor central.

Ilustración 63 - Filtro csrf_exempt

```
@csrf_exempt
def trigger_log_data_api_plc(request):
    if request.method == 'POST':
            start_date = datetime.fromtimestamp(float(request.POST['fecha_inicio'])/1000)
            stop_date = datetime.fromtimestamp(float(request.POST['fecha_fin'])/1000)
            mail_to = request.POST['mail_to']
            print(start_date, stop_date)

            rutina_get_api = get_api_data_thread('Hilo API data',start_date, stop_date, mail_to)
            rutina_get_api.start()

            content = {'result': "pending csv to email" , 'email': mail_to, 'start_date': start_date, 'stop_date': stop_date}
            return JsonResponse(content)
```

5.9.5.1. Script de envío de trigger diario

Debido a la necesidad de tener seguimiento diario de los datos de las plantas, se programa un script en Python que comprueba el estado de todas las plantas en funcionamiento y lanza un email diario con el seguimiento de los ejes y producción diaria.

A este script se le ha añadido la funcionalidad de activar el envío del CSV de datos de cada planta mediante un *request* de tipo POST con los parámetros del día y los emails de destino.

Este script se ejecuta diariamente en el servidor central de Solatom con la funcionalidad de *crontab*.

5.10. Calibración acelerómetros

5.10.1. Objetivo y posibles soluciones

La calibración de los acelerómetros que miden la posición de los ejes es otro de los aspectos críticos (junto con la calibración de offset de los servos) para el correcto funcionamiento de los módulos solares. Además, se podrá realizar el mantenimiento de estos sensores de forma remota, **reduciendo costes en operarios de mantenimiento.**

Las placas electrónicas Arduino que controlan cada módulo, tienen un algoritmo que se encarga de realizar la calibración de los acelerómetros, sin embargo, se debe lanzar de forma manual mediante el mando de control.

Es por eso que, para facilitar la instalación y minimizar los tiempos de *comissioning*, **es necesario automatizar este proceso de calibración** mediante la aplicación desarrollada, mostrando por pantalla el estado de la calibración.

5.10.2. Funcionamiento automático calibración Arduino

Este algoritmo de Arduino, realiza una serie de procedimientos de giro y ajuste para remapear el cálculo de la inclinación del espejo utilizando un algoritmo cubierto por secreto industrial.

Para automatizar el proceso de lanzamiento de la calibración, se necesita leer y escribir los siguientes registros:

Tabla 10 - Registros de protección de calibración de acelerómetros

9	allow protected params plant	R/W		timeout s	10	0	32767
251	Tuning procedure, all axis, starting at axis	R/W	plant	axis 10=click; 99=off;		0	9; skip:99
252	Tuning procedure, only axis n	R/W	plant	axis		0	9; skip:99
253	Tuning percent complete	R		por mil		0	1000
254	Tuning time remain	R		min	10	0	600

Primero se desactiva la protección escribiendo el registro 9 y seguidamente se escribe un 0 en el registro 251 para comenzar la calibración de todos los ejes.

A continuación, se realiza una lectura continua de los registros del 251 al 254 para tener feedback del estado de la calibración y mostrarlo por pantalla. Esta lectura se guarda en la base de datos, del modelo TunningAcc, el cual tiene los siguientes datos:

Ilustración 64 - Crear elemento de la base de datos de TunningAcc

```
tunning_state = TunningAcc(eje=read_list[0], porcentaje=read_list[2]/10, tiempo_restante=read_list[3]/10, running=running)
```

Para salir del bucle de lectura continua se usa la condición de lectura del registro 251 con valor de 99, que significa que el algoritmo de calibración de acelerómetros ha acabado.

Otra opción para salir del bucle es pulsar el botón de "Terminar rutina de tuning", que lanza la función *end_tunning_acelerometros_view*. Esta función escribe en el registro 251 el valor de 99, forzando la parada de calibración y matando el hilo que estaba leyendo el feedback del estado de los ejes.

5.10.3. Vista "Tuning acelerómetros"

Esta vista se encarga de lanzar la rutina automática de calibración de los acelerómetros por lo que no existen parámetros para modificar el funcionamiento de esta, sino que se lanza pulsando el botón de "Lanzar rutina tuning".

Ilustración 65 - Formulario de calibración de acelerómetros

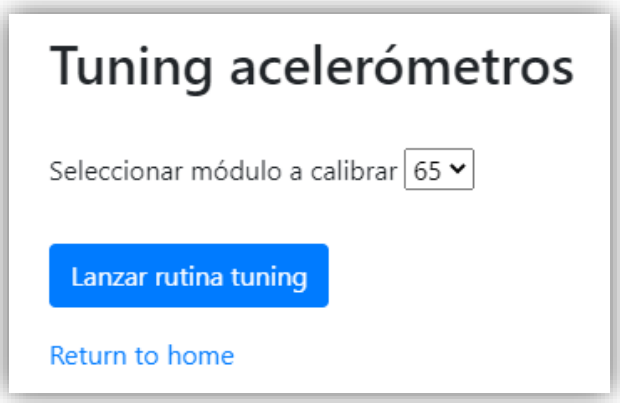
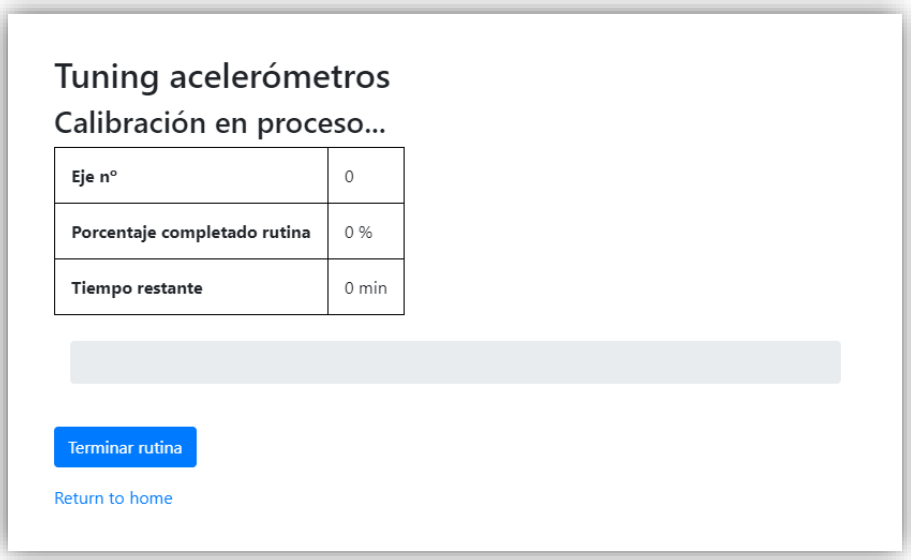


Ilustración 66 - Feedback del estado de calibración de los acelerómetros



Al pulsarlo, se carga la página que se ve en la siguiente imagen, la cual muestra el eje que se está calibrando, el porcentaje completado y el tiempo restante en minutos. También se añade una barra de progreso para el porcentaje completado.

Esta vista recibe los datos de una petición Ajax de javascript que gestiona la vista *ajax_tunning_feedback* y devuelve el último modelo de la base de datos que ha escrito el hilo de tuning durante la lectura.

5.11. Front-end

Uno de los objetivos principales es conseguir una interfaz visual, simple y fácil de manejar, por lo que al tratarse de una página web, se necesita realizar un desarrollo *front-end.* Las herramientas para el desarrollo han sido: la librería Bootstrap v.4.0, CSS, Javascript y el sistema de plantillas HTML de Django.

5.11.1. Bootstrap

Esta librería código abierto de CSS y Javascript utiliza una serie de clases predefinidas que se añaden dentro del código HTML para modificar el aspecto y comportamiento de la página. Se caracteriza por facilitar el diseño rápido y totalmente customizado para crear páginas web *responsive* gracias al sistema de diseño de cuadrículas.

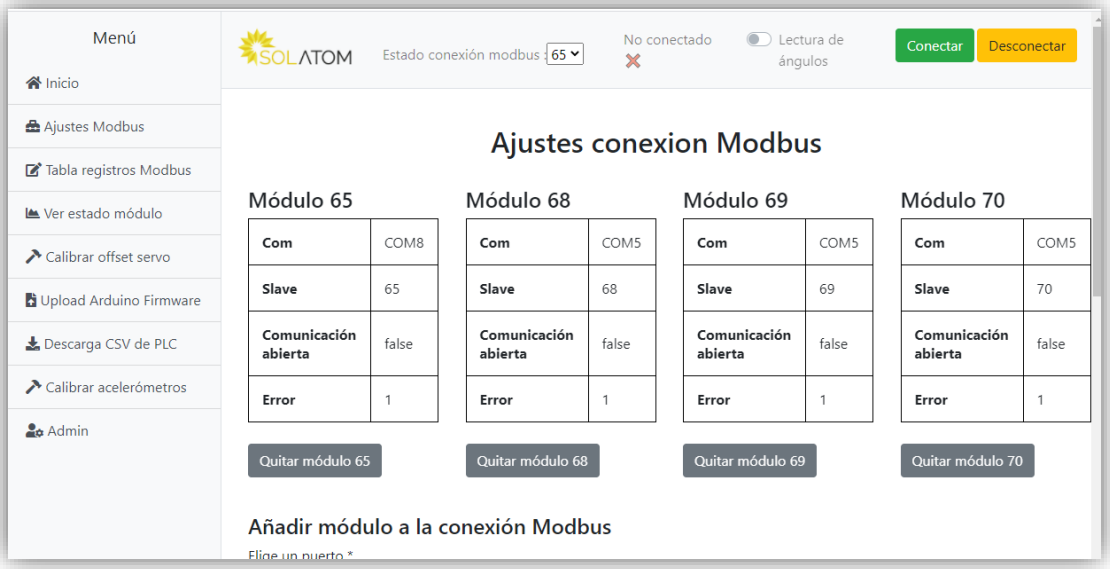
Para instalarla, se puede hacer uso de su CDN (Content Delivery Network), que consta de unas líneas de código que se ponen en la plantilla base HTML y permite utilizar los recursos online sin la necesidad de descargar la librería de forma local.

5.11.1.1. Comportamiento responsive

Con esta librería se ha desarrollado el menú *responsive* de la izquierda de la página web, el cual se puede mostrar y esconder al pulsar el botón del logo o cuando la pantalla es pequeña, lo que quiere decir que se está mostrando en un dispositivo móvil y se adapta automáticamente a su tamaño.

La página de "Ajustes Modbus" está programada con una serie de clases para que los diferentes dispositivos que se añadan se ordenen de manera que no se choquen y se ajusten al tamaño de la página.

Ilustración 67 - Módulos añadidos a Ajustes Modbus



Y cuando la página se ajusta, se esconde el menú del lado izquierdo y el menú superior. Además, en esta página, los módulos se ordenan uno debajo de otro para que se vean correctamente. Para mostrar los menús se puede volver a pulsar en el botón del logo para mostrar el menú lateral y el botón derecho con las tres rayas para mostrar el menú superior.

Ilustración 68 - Vista de móvil en Ajustes Modbus

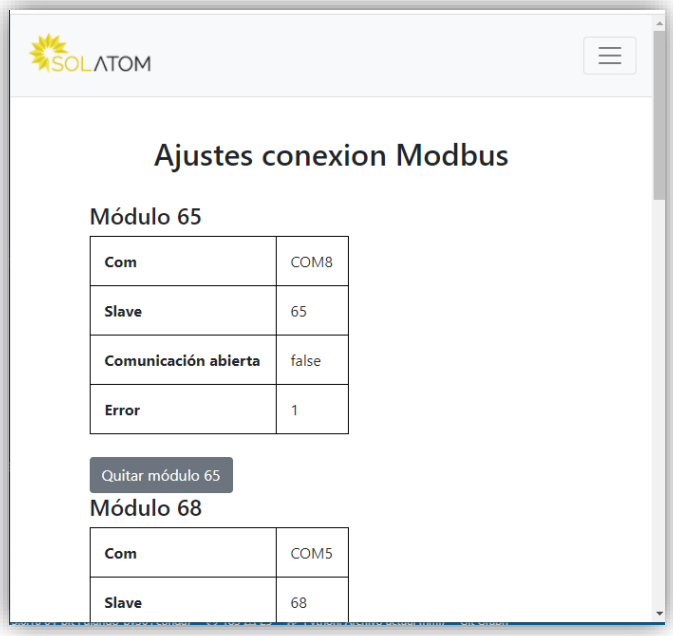
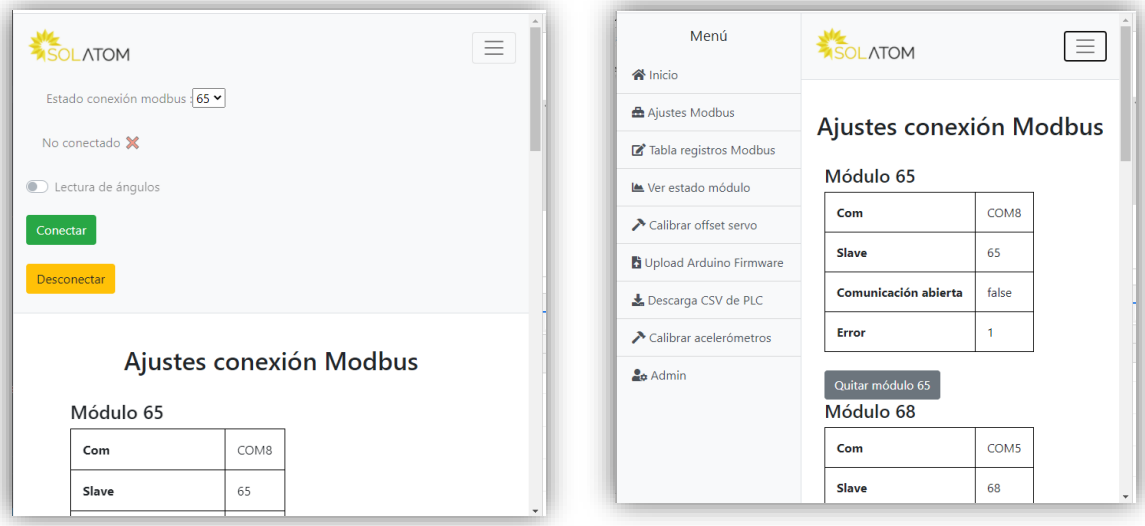
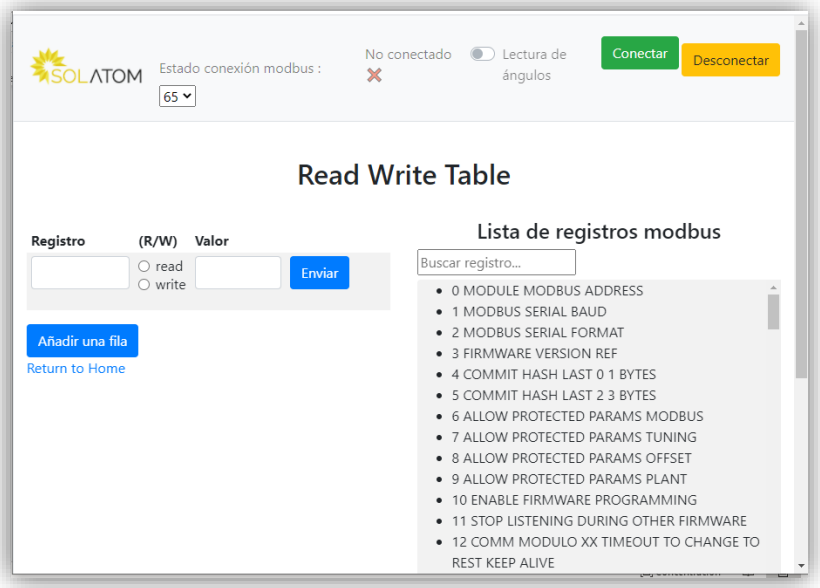


Ilustración 69 - Menús superior y lateral



Con el uso de esta librería se puede hacer que todas las páginas se ajusten al tamaño de la pantalla según el dispositivo para mejorar el uso. Además, mejora el aspecto añadiendo colores y formas atractivas a la vista.

Ilustración 70 - Vista de la página Read Write Table



5.11.2. Javascript y CSS

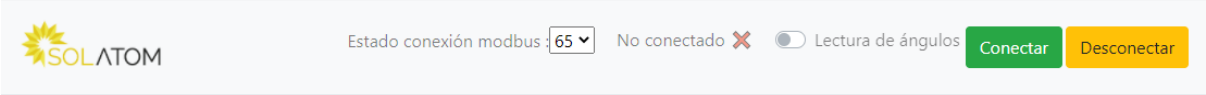
5.11.2.1. Menú superior

El uso de Javascript se ha explicado en cada punto del desarrollo de las páginas de forma detallada, y su finalidad principal es añadir comportamientos a la página cuando se interactúa con ella y permitir la comunicación con el servidor mediante la librería de jQuery.

Para el comportamiento del encabezado se ha utilizado Javascript, el cual interactúa con el servidor para recibir y enviar datos mediante Ajax. Las opciones que ofrece este menú son las siguientes:

- *Estado conexión modbus:* esta opción permite elegir el módulo sobre el que trabajar. Al seleccionar un módulo, se activa o no la lectura de ángulos para ese módulo según la base de datos y se ve si está conectado o no.
- *No conectado* ✗*:* en esta opción se puede ver si la comunicación Modbus se ha establecido correctamente o no.
- *Lectura de ángulos:* se trata de un *switch* con el que se puede tener feedback del estado de la lectura de ángulos en la base de datos y también se puede activar o desactivar según se necesite.
- *Conectar:* este botón envía una petición al servidor para iniciar la comunicación Modbus. Al pulsarlo se puede ver un *spinner* como indicativo de que la conexión está funcionando.
- *Desconectar:* este botón envía la petición de desconectar del servidor.

Ilustración 71 - Menú superior



Estas funcionalidades se han añadido a un fichero Javascript que se ejecuta en todas las páginas. A parte de este fichero, en cada página se pueden añadir scripts específicos que se ejecutan únicamente en la página determinada.

Por otro lado, el CSS se usa para retocar pequeños detalles de aspectos como márgenes, espaciadores, colores, líneas…

5.11.3. Plantillas

El sistema de plantillas de Django permite trabajar de forma más cómoda, al poder separar las páginas por pequeños bloques HTML y poder modificar el comportamiento renderizando un bloque u otro dependiendo de ciertas condiciones.

En primer lugar, se crea una plantilla de base que tiene los encabezados y la estructura normal de un archivo HTML, sin embargo, se añaden los bloques marcados en amarillo en la imagen para, una vez renderizado, poder añadir más código HTML dependiendo de la página que se quiera mostrar.

Ilustración 72 - Variables de Python de gestión de plantillas

```
{% load static %}

<html>
<head>

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity="sha384-9aIt2n

    {% block stylesheet %}{% endblock %}
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/tempusdominus-bootstrap-4/5.0.0-alpha14/css/tempusdominus-b


    <script src="https://kit.fontawesome.com/119b2dcfa5.js" crossorigin="anonymous"></script>

    {{ script | safe }}
    {{ form.media }}
    <link href="{% static 'css/styles.css' %} " rel="stylesheet">


</head>
```

Se pueden añadir bloques de varios tipos, según el tipo de código que sea: “block content” para HTML, “block stylesheet” para CSS y “block javascript” para Javascript.

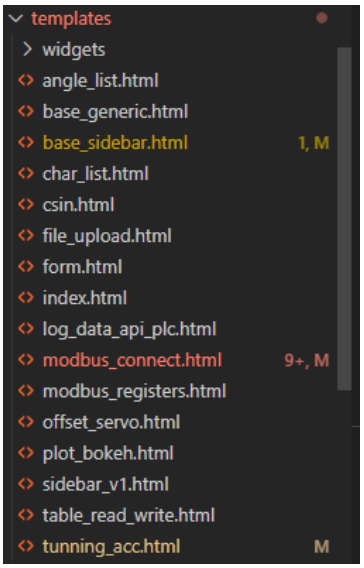
Con la plantilla principal, se pueden crear otros archivos HTML para cada página en los que se debe indicar que pertenecen a una plantilla superior para que se rendericen el bloque HTML con la plantilla. Con estos modificadores de Django se pueden añadir librerías y bloques de código de varios lenguajes en un mismo archivo.

Ilustración 73 - Variables Python gestión de plantillas HTML

```
{% extends "base_sidebar.html" %}

{% load crispy_forms_tags %}

{% block javascript %}
<script>


  var spinner = document.getElementById("id_spinner");




  {% if module_list %}

  {% for module in module_list %}
  var quit_module_{{ module.slave }} = document.getElementById("id_quitar_modulo_{{module.slave}}");
  console.log(quit_module_{{module.slave}});
```

Por último, para renderizar cada página se divide el código en archivos separados y todos hacen referencia a la plantilla principal. Para ello se hace uso de una carpeta con todas las plantillas:

Ilustración 74 - Plantillas HTML



Y al renderizar estas plantillas, se carga el código de la plantilla principal y se incrustan los bloques de las otras plantillas en los lugares del código que corresponda.

5.12. Conexión remota

5.12.1. Objetivo y posibles soluciones

La conexión de forma remota a las funcionalidades descritas en este trabajo es un punto crítico, ya que facilita el control de las plantas y ahorra muchos costes de desplazamiento y personal de mantenimiento. Por tanto, existe la necesidad de una conexión estable y segura a las plantas, además de que sea sencillo de mantener y recuperar en el caso de que se pierda la comunicación, con la posibilidad de recuperación de forma automática.

5.12.1.1. Solución 1: túnel VPN

Para que la comunicación sea estable y segura se hace uso de un túnel VPN mediante el protocolo SSH ("SSH Tunnel - Local and Remote Port Forwarding Explained With Examples," n.d.) ("SSH Protocol," n.d.). A esto se le llama *local port forwarding* y permite acceder de forma segura al *webcontroller* del PLC y a las funcionalidades de la raspberry conectada al PLC. Para ello se usa la raspberry como jumpserver a la ip del webcontroller del PLC, sin embargo, se necesita que la ip de la raspberry sea accesible de forma externa.

5.12.1.2. Solución 2: túnel VPN con peer2peer de Remote.it

Para conseguir acceder a la raspberry desde cualquier conexión externa se hace uso de una solución comercial llamada Remote.it. Esta aplicación permite crear conexiones proxy o peer2peer desde el dispositivo que hace la petición de conexión a la raspberry. El uso de una conexión peer2peer presenta unas **ventajas respecto a la conexión proxy** (remote.it, n.d.-a), y son las siguientes:

- En conexión peer2peer, los datos no pasan a través de los servidores de remote.it, mientras que el proxy pasa al servidor de la empresa y luego a la raspberry.
- El flujo de datos es significativamente más rápido al no ser necesario el uso de servidores de terceros.
- La conexión es estable mientras que exista conexión a internet en los dos dispositivos, mientras que en proxy puede fallar el servidor de remote.it.
- La URL es siempre la misma y se puede guardar para reestablecer la conexión de forma automática. En proxy, la url cambia cada cierto tiempo.

Ilustración 75 - Peer2peer vs Proxy

Por lo tanto, se usa conexión peer2peer con el servicio de remote.it para crear una ip externa accesible para la raspberry.

Prueba de conexión desde Windows:

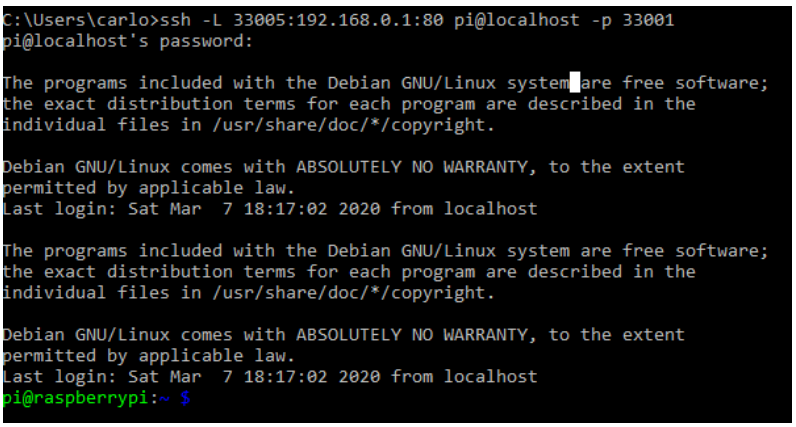
Para comprobar el comportamiento de la conexión peer2peer y el local port forwarding se prueba los siguiente:

Directamente desde Windows se crea *port fowarding* de un puerto local libre (por ejemplo 33005) hacia el puerto 80 de la direccion ip del ruoter de la planta, utilizando como jump server la raspberry con conexión ssh creada en remot3.it (con la app en localhost:33001).

El comando del terminal es el siguiente: *ssh -L 33005:192.168.0.1:80 pi@localhost:33001*

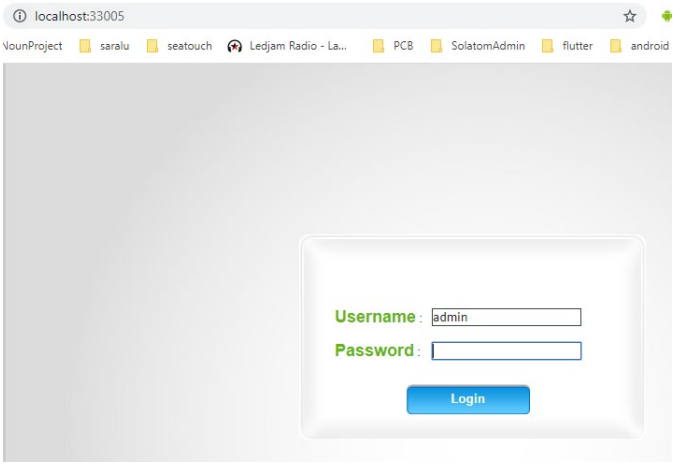
Esto permite conectarse vía ssh a la raspberry:

Ilustración 76 - Acceso a terminal Raspberry



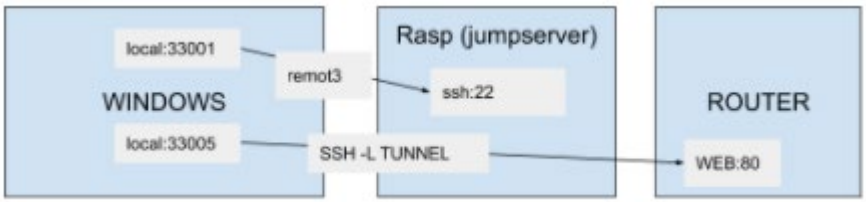
Y se puede abrir localhost:33005 en el navegador y acceder a la página del router Por lo tanto estamos accediendo a una ip (192.168.0.1:80 página de ajustes del router) de la misma red que la raspberry directamente desde nuestro ordenador usando la raspberry como jumpserver.

Ilustración 77 - Acceso a router vía redirección SSH



El esquema de la conexión que se ha realizado de prueba es el siguiente:

Ilustración 78 - Esquema de conexiones VPN y redirección SSH



5.12.1.3. Solución 3: Automatizar conexión y recuperación de conexión

Una vez probada la conexión remota se realiza la automatización de estos procesos mediante el servidor de la empresa para que las conexiones se mantengan activas y se puedan recuperar en el caso de que se pierda la conexión.

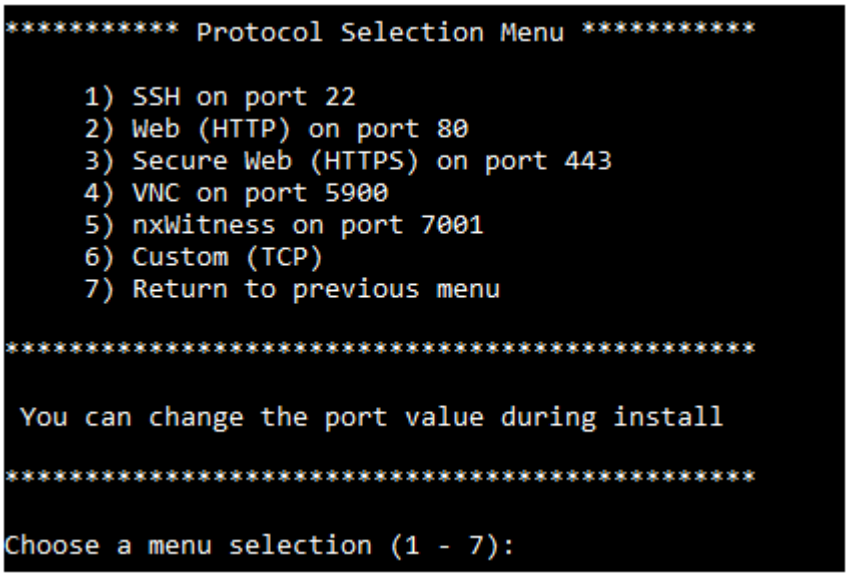
5.12.2. Configuración e instalación en Raspberry

5.12.2.1. Instalación remote.it

Como ya se ha comentado antes, la pantalla HMI conectada al PLC de las plantas solares se basa en una raspberry que es accesible desde el exterior mediante ssh o escritorio remoto cuando se configura la aplicación de remote.it. Por lo tanto, se va a usar como jumpserver para acceder al HMI y a esta aplicación de forma remota.

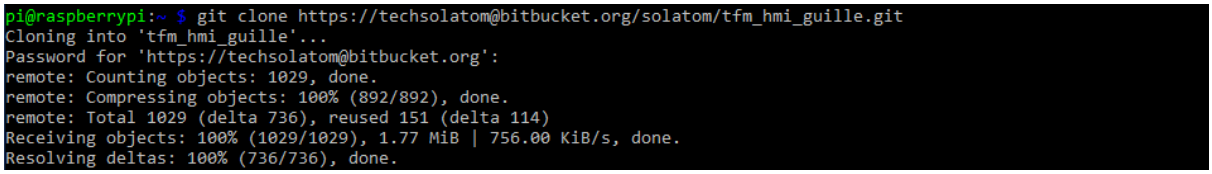
La instalación se hace siguiendo los pasos de la documentación de Remote.it (remote.it, n.d.-b), la cual permite instalar varios servicios:

Ilustración 79 - Instalación Remote.it



Una vez instalado Remote.it, se puede instalar la aplicación del TFM y las librerías necesarias vía ssh o escritorio remoto. Para ello se hace uso del software git, que a parte de hacer un seguimiento y control de versiones permite clonar un proyecto entero de forma remota.

Ilustración 80 - Git clone TFM



A continuación, se deben instalar las librerías necesarias para el funcionamiento de la página web. Para ello se crea un entorno virtual (mediante la librería *virtualenv*) para que no afecte a las dependencias de la versión de la raspberry. El comando para crear un entorno con el nombre "django_env36" y versión de Python 3.6 es: *virtualenv django_env36 -p python3.6.*

Una vez creado, se debe activar e instalar las librerías necesarias mediante *pip.* Para ello se puede crear un archivo de dependencias e instalar todas a la vez. En la siguiente imagen se puede ver el archivo de dependencias *requirements.txt* con las librerías necesarias para este proyecto:

Ilustración 81 - Librerías de Python para el proyecto

```
(django_env36) pi@raspberrypi:~/tfm_hmi_guille $ cat requirements.txt
certifi==2020.6.20
chardet==3.0.4
Django==2.2.14
django-crispy-forms==1.9.2
django-tempus-dominus==5.1.2.13
future==0.18.2
idna==2.10
iso8601==0.1.12
minimalmodbus==1.0.2
pyserial==3.4
python-dateutil==2.8.1
pytz==2020.1
PyYAML==5.3.1
requests==2.24.0
serial==0.0.97
six==1.15.0
sqlparse==0.3.1
urllib3==1.25.10
```

Con las dependencias instaladas, se crea un archivo .sh para ejecutar la activación del entorno y lanzar el servidor Django con la aplicación accesible para cualquier ip en el puerto 8000 (por eso es 0.0.0.0:8000) porque el puerto 80 ya está ocupado por la pantalla HMI del PLC. De esta forma se puede lanzar el ejecutable desde el crontab con el reinicio de la raspberry para que el servidor esté siempre activo.

Ilustración 82 - Activar virtualenv

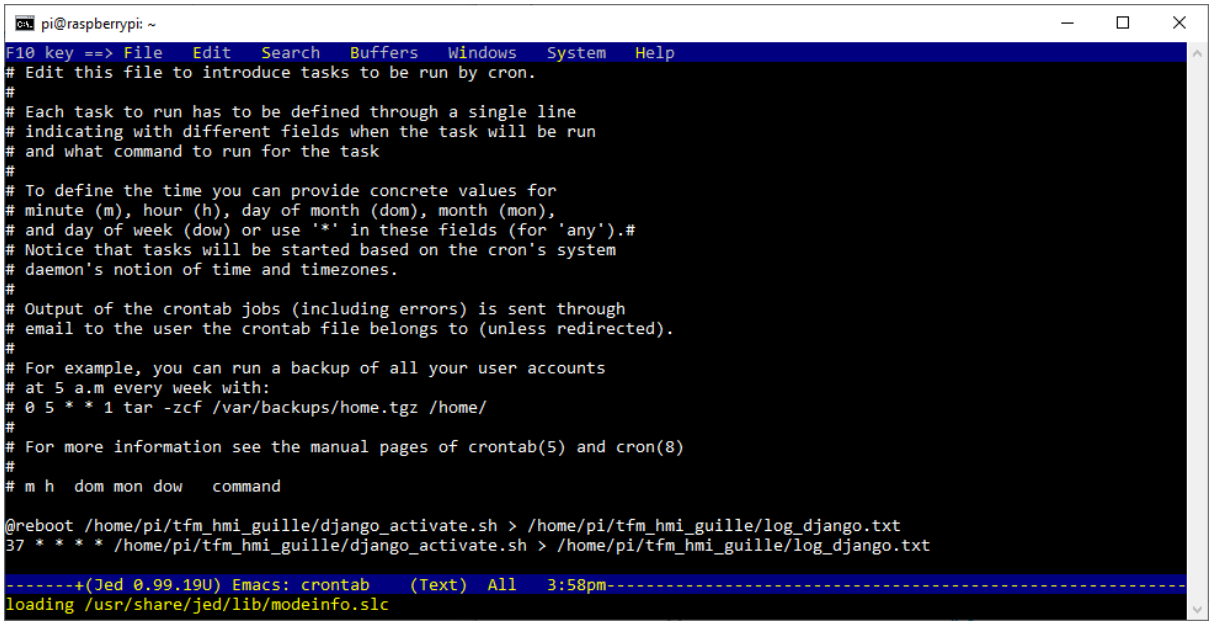
```
(django_env36) pi@raspberrypi:~/tfm_hmi_guille $ cat django_activate.sh
#!/bin/bash

source ~/django_env36/bin/activate
python3.6 ~/tfm_hmi_guille/manage.py runserver 0.0.0.0:8000
```

Ilustración 83 - Servidor Django iniciado

```
(django_env36) pi@raspberrypi:~/tfm_hmi_guille $ ./django_activate.sh
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
September 08, 2020 - 13:35:35
Django version 2.2.14, using settings 'hmi.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
```

En la siguiente imagen, se puede ver el archivo crontab de la raspberry para que lance el ejecutable django_activate.sh con cada reinicio y cada hora por si hay algún fallo. Se guarda la salida de este proceso en el archivo *log_django.txt* para hacer *debug* de la aplicación.

Ilustración 84 - Crontab de la Raspberry



5.12.3. Configuración servidor central Solatom

Ahora es necesario activar la redirección desde el servidor central para poder acceder a la página web y al HMI de la planta solar.

Para ello se deben instalar remote.it para crear la conexión peer2peer como indica en la documentación de Remote.it.

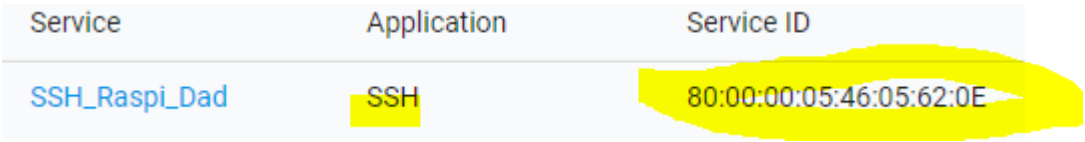
A continuación, se crea un archivo *p2p.sh* al que se le activa la modalidad de ejecución (con el comando *chmod +x ./p2p.sh*). Este archivo se encarga de abrir la comunicación peer2peer con un dispositivo configurado en Remote.it y mostrarlo en un puerto determinado. Se ejecuta como *./p2p.sh username@gmail.com password.*

En este archivo se deben modificar las líneas subrayadas para determinar el dispositivo a conectar (*uid*), la dirección y el puerto al que redirigir esta conexión (*address* y *port*).

```
#!/bin/sh
# p2p.sh - connectd P2P initiator example script shows the use of command line arguments for target and user
# account details

uid="80:00:00:05:46:05:62:0E"
address=127.0.0.1
port=33100
usage()
{
    echo "usage: $0 username password"
}

if [ "$1" = "" ]; then
    usage
```

```
    exit 1
elif [ "$2" = "" ]; then
    usage
    exit 1
fi
username=$1
password=$2

if [ "$(which connectd)" = "" ]; then
    echo "connectd is not properly installed."
    exit 1
fi

echo "Setting up a P2P connection to $uid on 127.0.0.1 port $port."
b64username="$(echo "$username" | tr -d '\n' | base64)"

b64password="$(echo "$password" | tr -d '\n' | base64)"

connectd -c $b64username $b64password "$uid" T"$port" 2 "$address" 12 &

# get the process ID if the command is still running
pid=$(ps ax | grep $b64username | grep $uid | awk '{ print $1 }')
if [ "$pid" != "" ]; then
    sleep 10
    echo
    echo "Your connection is now active on $address port $port."
    echo
    echo "To terminate this connection, type in:"
    echo "kill $pid"
    echo "To terminate this connection, type in:" >> log.txt
    echo "kill $pid" >> log.txt

fi
```

Ilustración 85 - UID de la configuración SSH de la Raspberry



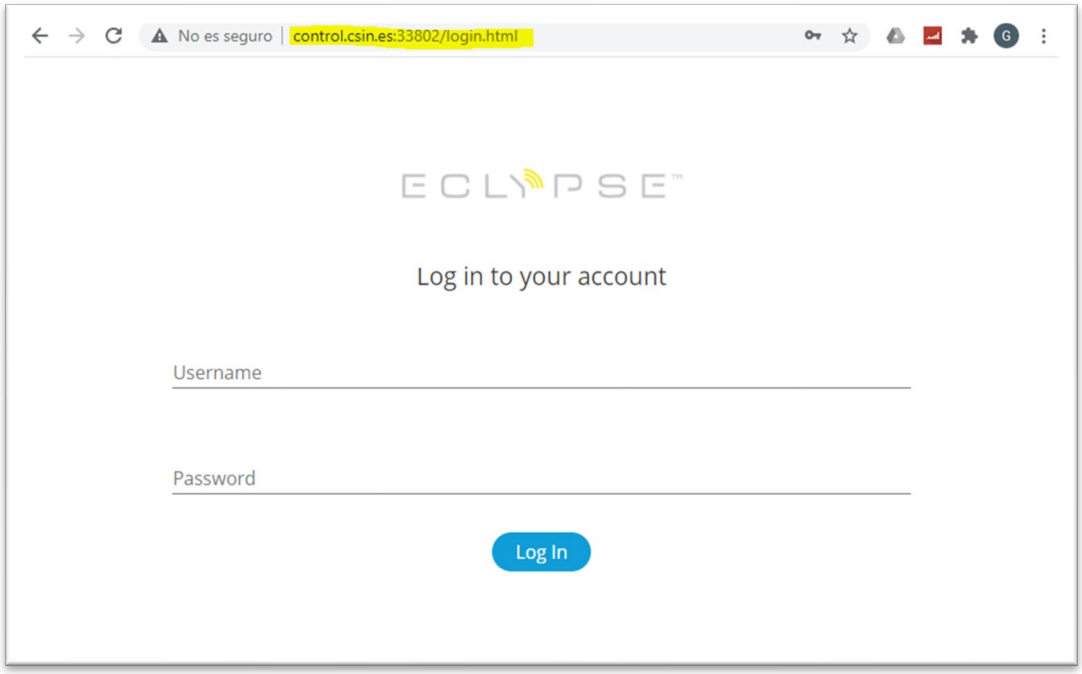
5.12.3.1. Redirección del puerto

Cuando está disponible en el VPS (Virtual Private Server, el servidor central) el puerto ssh a la raspberry en el puerto 33100, desde el VPS se hace un túnel en el puerto público *:33802 que redirige al equipo local (el HMI de la raspberry). El comando es el siguiente:

*ssh -L *:33802:192.168.0.30:80 pi@localhost -p 33100*

De esta forma ya se tiene acceso al HMI de la planta, el cual muestra el webcontroller en la ip 192.168.0.30 (ip del PLC) en el puerto 80. En la imagen se puede ver la página de acceso al HMI del PLC Eclypse en la dirección *control.csin.es/33802* (control.csin.es es la dirección del servidor central).

Ilustración 86 - Acceso al HMI del PLC desde la redirección SSH



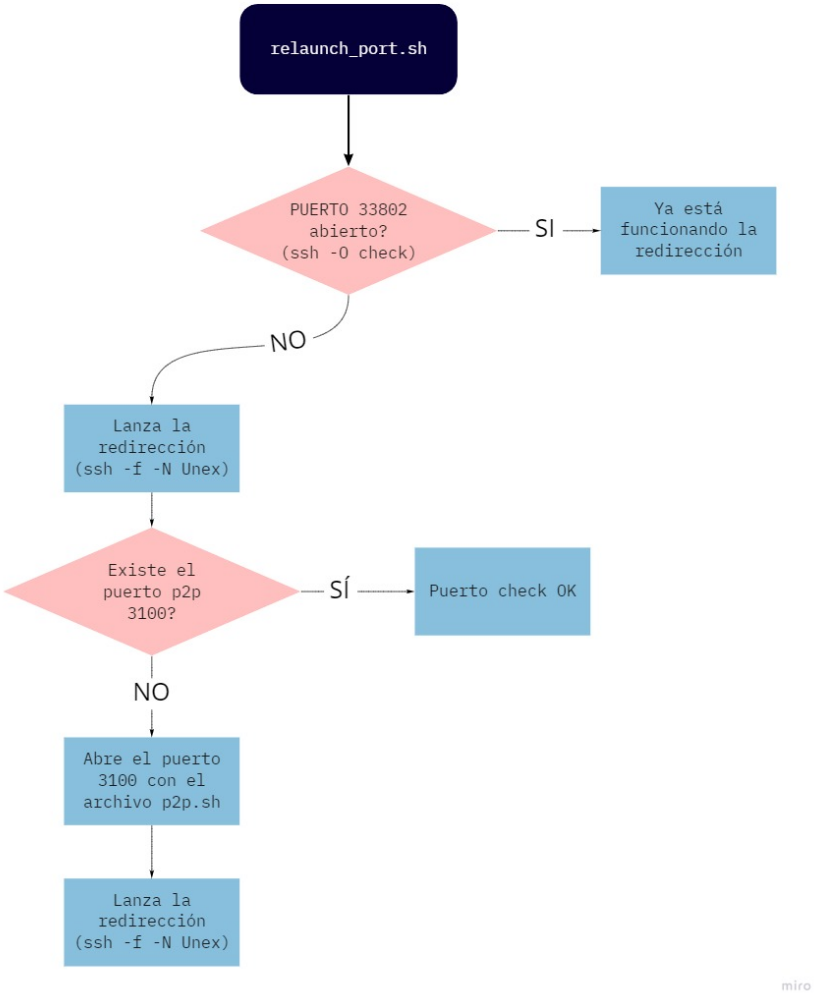
5.12.3.2. Automatización de la redirección

Para automatizar la redirección del puerto y mantener la conexión abierta aunque se pierda la conexión se crea un host en el archivo *~/.ssh/config* con la siguiente estructura:

```
Host Unex
   HostName 127.0.0.1
   Port 33100
   User pi
   LocalForward *:33800 192.168.0.30:80
   RequestTTY no
   ExitOnForwardFailure yes
   ControlMaster auto
   ControlPath ~/.ssh/cm_sockets/%r@%h:%p
```

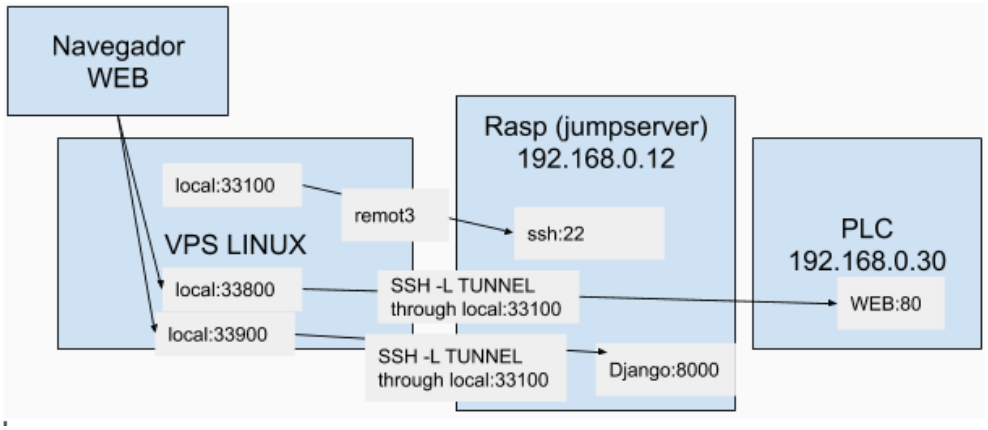
Por tanto, para cada planta se crea un host como el anterior cambiando los puertos para mostrar el HMI y el TFM. Además para activar el lanzamiento automático ante la pérdida de conexión se crea el fichero *relaunch_port.sh* por cada planta, el cual tiene la siguiente lógica:

Ilustración 87 - Esquema lógica relaunch_port.sh



Por lo tanto, la estructura final es la siguiente:

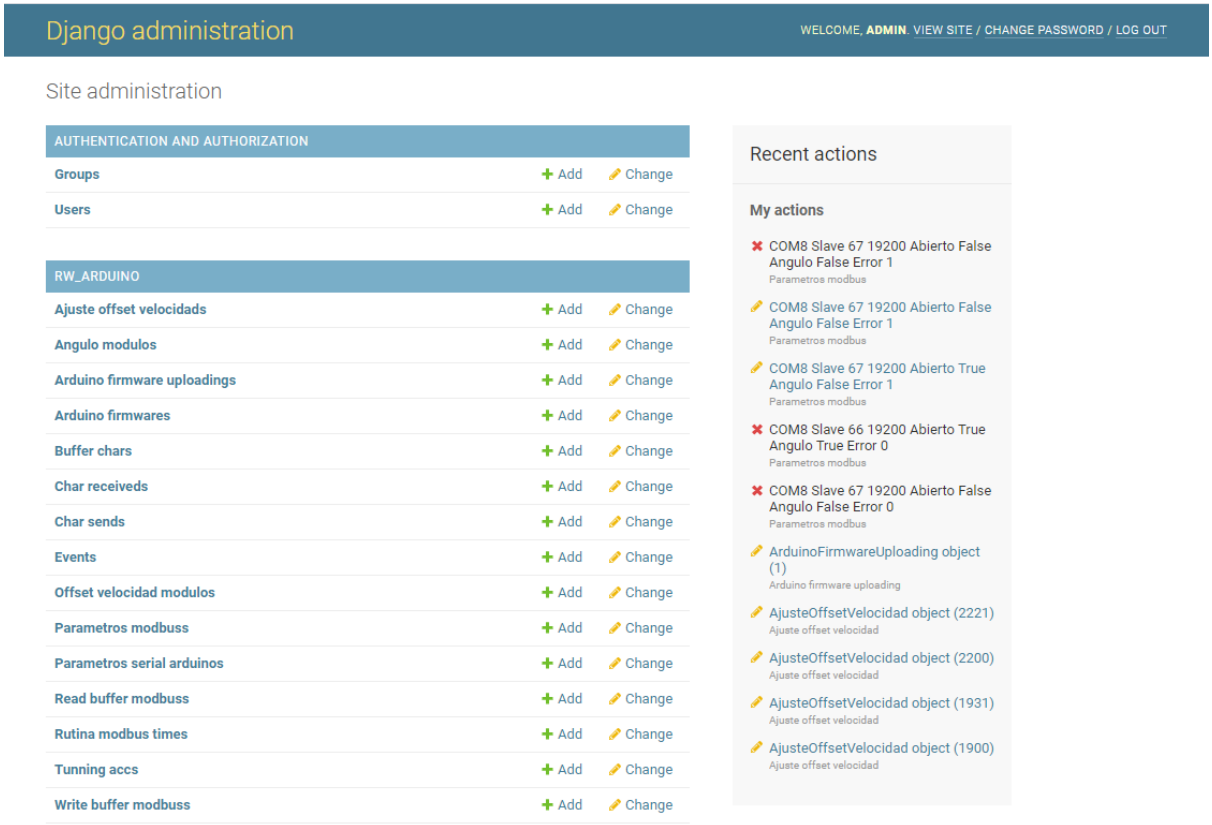
Ilustración 88 - Esquema redirección SSH del HMI y del TFM



5.13. Admin

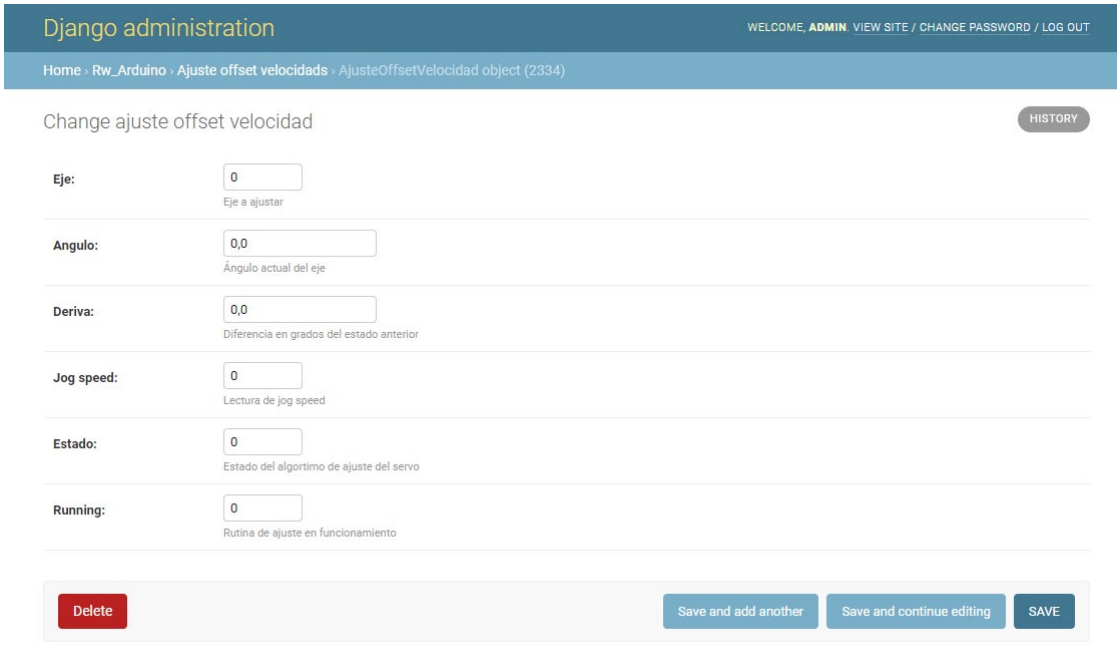
La vista de Admin viene preinstalada en el framework de Django y en este caso se utiliza para manejar y observar los modelos de la base de datos. También existe la opción de crear grupos de usuarios con permisos diferentes y añadir usuarios para el acceso a Admin.

Ilustración 89 - Vista de Admin



Al entrar en un modelo, se pueden observar todas las instancias de este modelo y modificar, borrar o añadir nuevas instancias.

Ilustración 90 - Vista de una instancia de modelo



6. Análisis de resultados

En este apartado se va a comprobar el correcto funcionamiento de las vistas. Para ello, se obtendrán los resultados necesarios y se comentará respecto a los resultados si se ha cumplido el objetivo o si, en caso contrario, es necesario realizar otro planteamiento del problema.

6.1. Vista "Inicio"

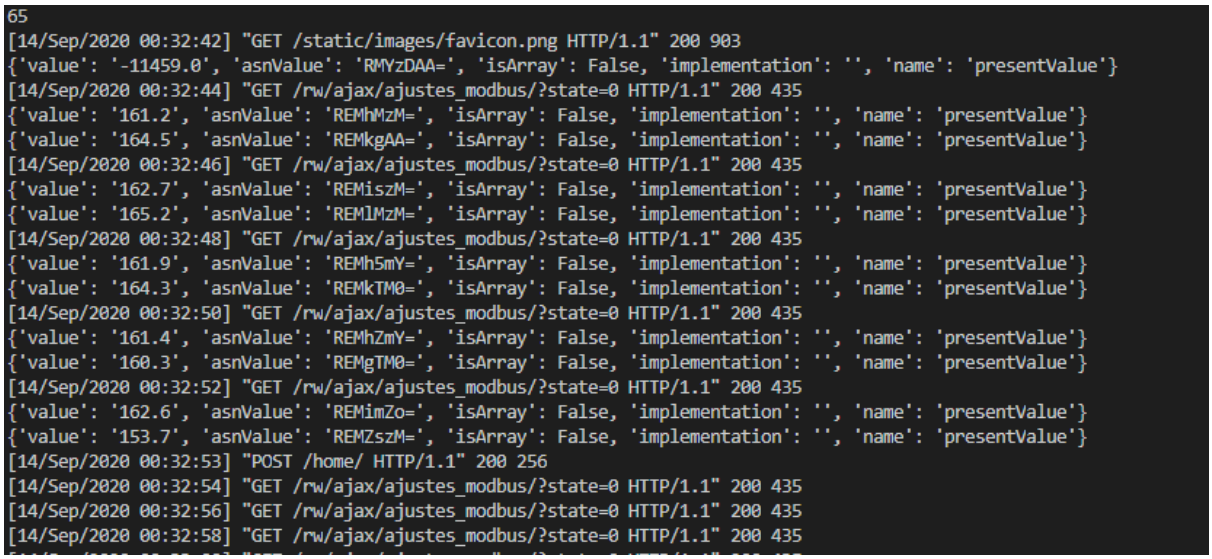
En esta vista se debe comprobar que las peticiones, tanto de Ajax como las del PLC, se realizan de forma correcta y que la información pasa desde el PLC a la vista de la página web.

6.1.1. Petición al PLC

Por lo tanto, se realiza una tarea de *debug*, para que el resultado de la respuesta de la petición al PLC se muestre por la consola del terminal del servidor. En la siguiente imagen, se puede observar que se imprimen las respuestas de las diferentes variables analógicas. Dicha respuesta está en formato JSON con más información sobre la variable, en este caso lo interesante es el valor actual, por lo que se extrae y se añade a otra variable JSON para ser enviada como respuesta al Javascript.

Se agrupan las respuestas en formato JSON, ya que así se puede enviar una respuesta serializada, lo cual añade seguridad a la transferencia de datos.

Ilustración 91 - Respuesta JSON de la API del PLC



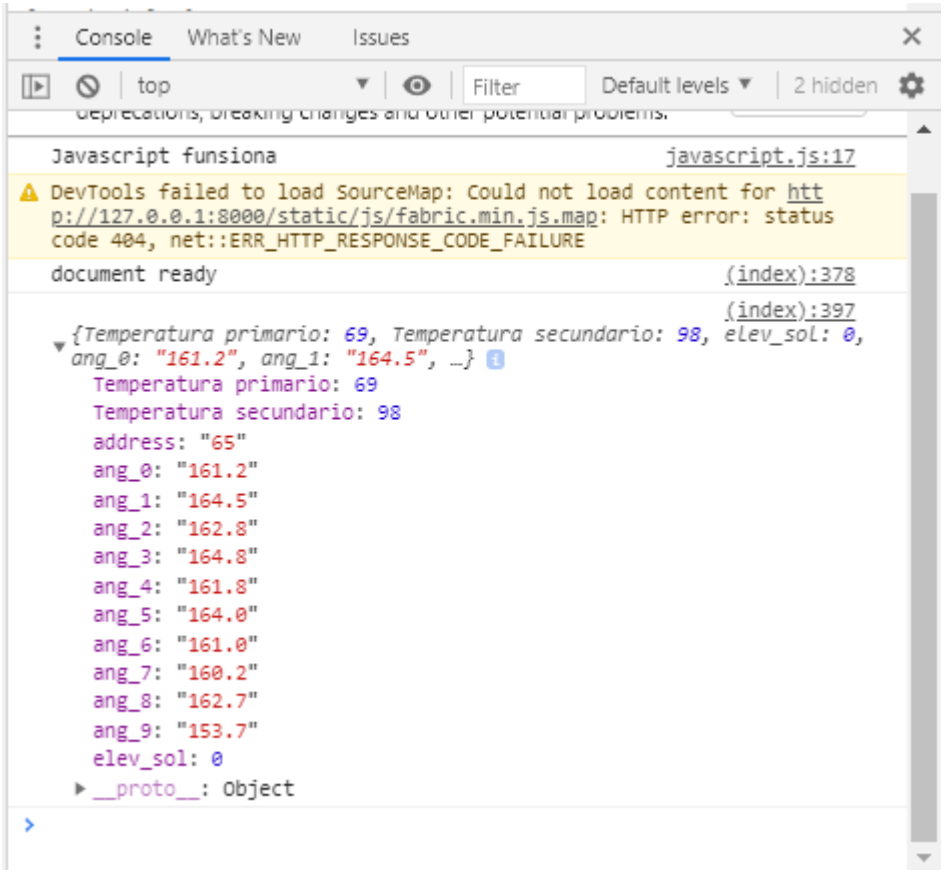
Esta respuesta del PLC se recibe de forma remota durante el desarrollo del proyecto, debido a que el programa todavía no está implementado en la raspberry. Al tratarse de una respuesta a través del servidor central, el tiempo de respuesta aumenta un poco debido al tiempo de respuesta de la señal 3G. Una vez se realice la implementación del programa, la ip pasará a ser local (ya que la raspberry y el PLC están en la misma red) y así disminuirá notablemente el tiempo de respuesta.

6.1.2. Petición Ajax de Javascript

El script de Javascript de esta página realiza la petición al servidor de Django para obtener los datos de las variables del PLC del módulo especificado en el menú superior de la página. Esta función se debe realizar al iniciar la página y al cambiar de módulo desde el menú superior. Para comprobar que se cambien los datos y reciba la respuesta del servidor de Django, se abren las herramientas para desarrolladores de Chrome y así se puede comprobar en la consola los datos recibidos.

Al recargar la página de Inicio y al cabo de unos segundos, se recibe la respuesta desde Django y se imprime en la consola de Chrome la respuesta en formato JSON como se puede ver en la siguiente imagen:

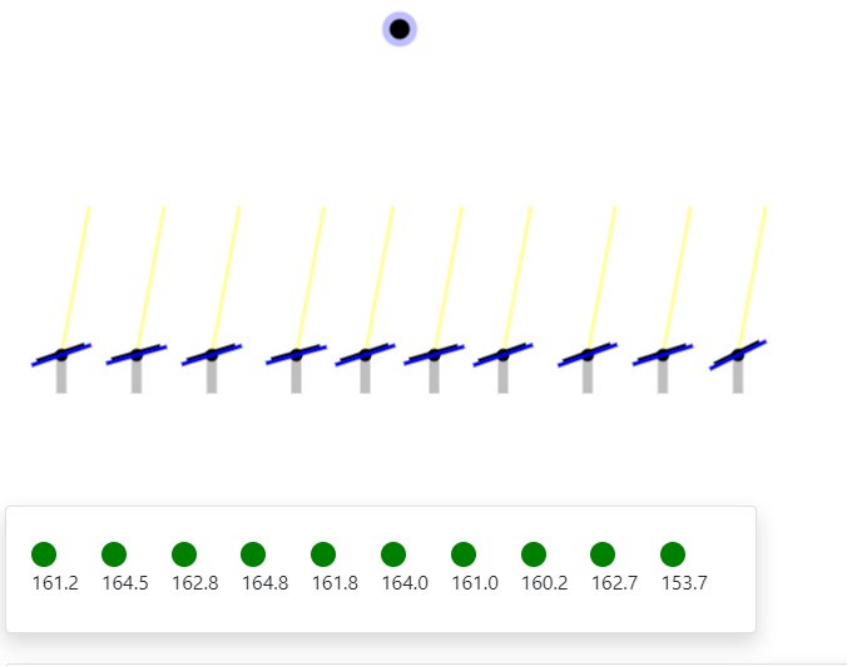
Ilustración 92 - Consola de Herramientas para desarrolladores



En la imagen se puede ver la respuesta de tipo JSON desglosada en variables con el valor actual y también el módulo del cual provienen los datos.

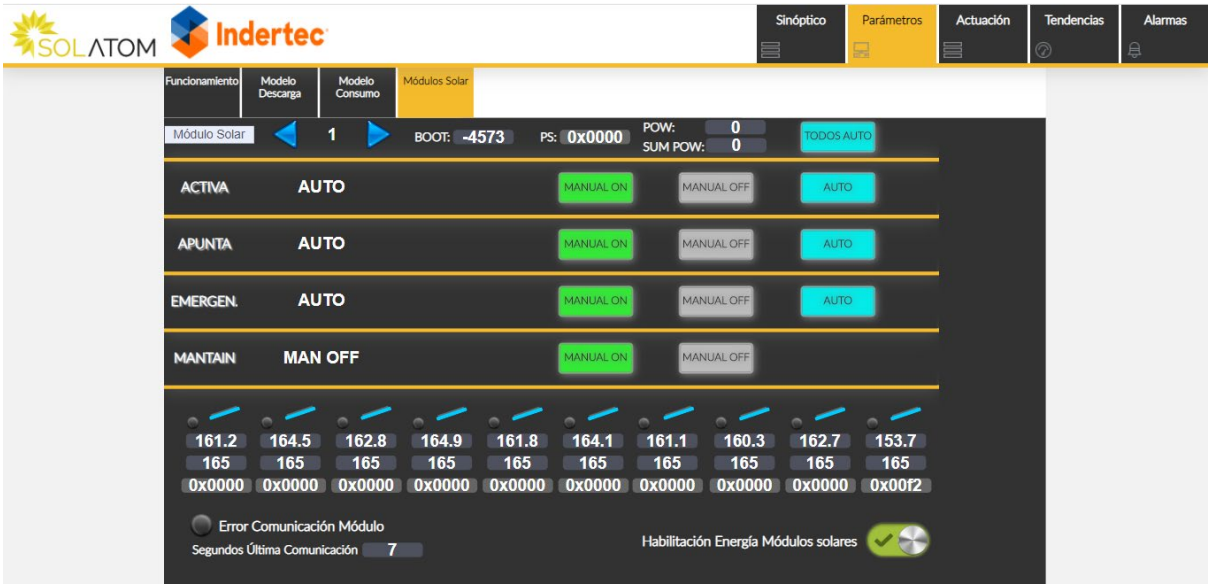
Dentro de la petición Ajax también se gestiona la actualización de la imagen Canvas cuando se recibe la respuesta del servidor Django, como se ve en la siguiente imagen:

Ilustración 93 - Dibujo Canvas de los ejes de un módulo



Si se accede al HMI del PLC de forma remota, se comprueba que los ángulos del módulo 1 (dirección 65) son correctos, ya que el sistema se encuentra actualmente en paro y, por tanto, los módulos están en reposo, es decir, apuntando hacia abajo.

Ilustración 94 - HMI del PLC



6.1.3. Análisis de resultados

A la vista de los resultados obtenidos, se ha conseguido una forma de visualizar el estado de los módulos solares de forma rápida y de bajo coste de datos 3G. Además, se trata de una solución que no interfiere en la comunicación Modbus, sino que está hecha de manera que se

pueda ver el estado de los módulos de forma externa mediante la API del PLC. Por lo que al iniciar la aplicación se puede analizar rápidamente los ejes en estado de error y determinar el error que presentan.

Por lo tanto, se ha cumplido el objetivo de tener una forma de visualizar el estado de los módulos de la planta solar y además se trata de una solución que no interfiere en la comunicación Modbus.

6.2. Comunicación modbus

Este punto es uno de los objetivos clave para el correcto funcionamiento del programa, puesto que, si no se consigue una comunicación rápida y fiable, podría resultar en problemas de lectura y escritura a la hora de realizar las calibraciones de los servos y los acelerómetros.

6.2.1. Obtención de resultados: Pruebas en prototipos

Para comprobar el correcto funcionamiento se realiza la conexión de dos placas de electrónica de los módulos para simular el comportamiento de una planta solar con dos módulos. El esquema de conexión es el siguiente:

Ilustración 95 - Esquema de conexión de pruebas de comunicación Modbus

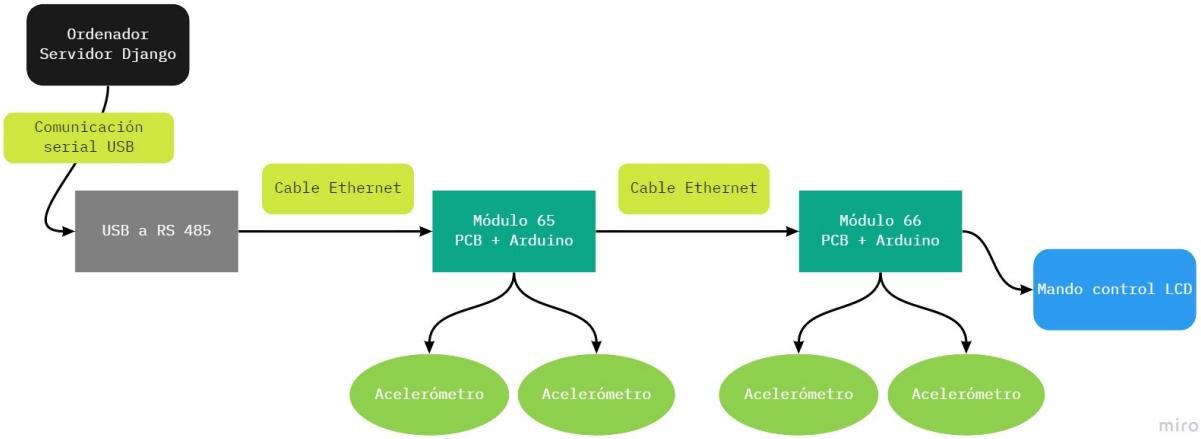


Ilustración 96 - USB serial a RS 485

Ilustración 97 - PCB módulo 65

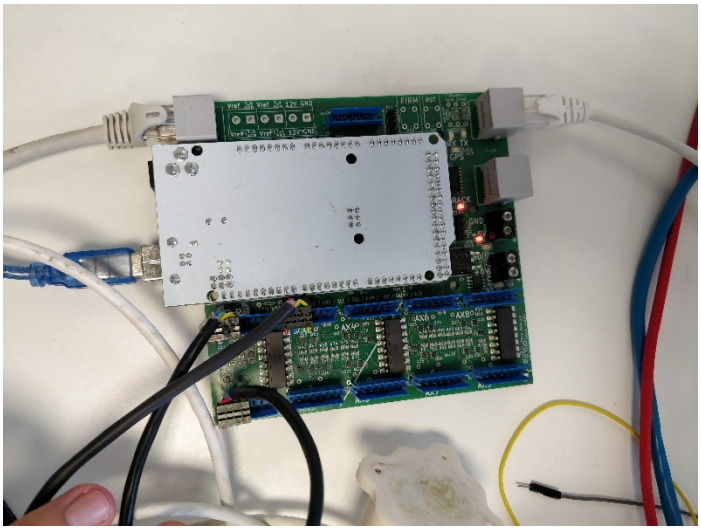
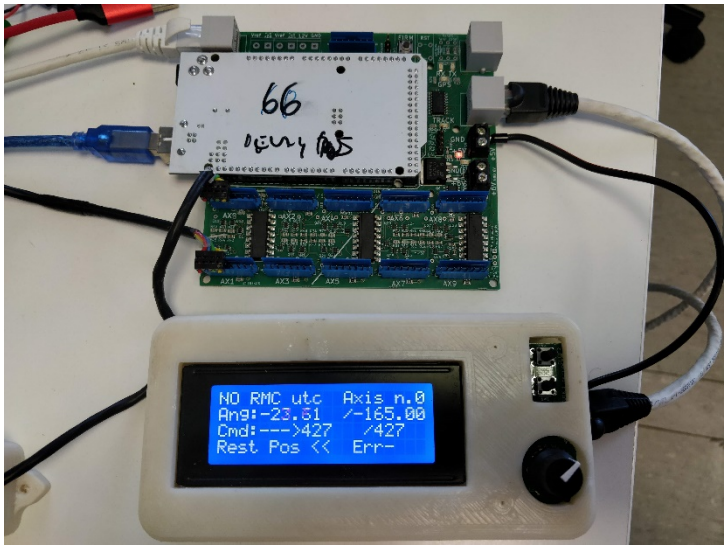


Ilustración 98 - PCB módulo 66 con mando LCD



En el ordenador es donde está funcionando el servidor de Django y realiza la tarea de maestro Modbus, enviando comandos de lectura y escritura de registros. Se usa el conector de USB a RS 485 con *half duplex* para conectarse vía cable de Ethernet a los módulos, los cuales se encuentran en serie uno con otro, de manera que solo contesta el módulo al que se le envía el comando.

Cada placa tiene conectada un par de acelerómetros para poder medir cambios en los ángulos y comprobar el correcto funcionamiento de la comunicación. La primera tiene la dirección 65 y la segunda la dirección 66, esta última también tiene el mando LCD conectado para comprobar el valor de los ángulos con el valor recibido por Modbus.

A continuación, en la página de “Ajustes Modbus”, se añaden los dos módulos (65 y 66) a la aplicación con el puerto USB Serial. Y una vez añadidos, se pulsa el botón de conectar.

Ilustración 99 - Formulario Añadir módulo de conexión Modbus

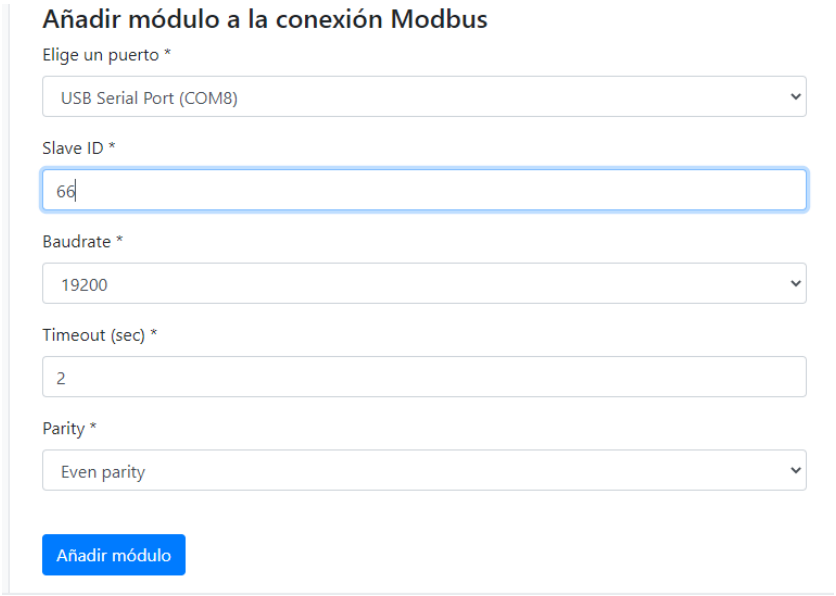
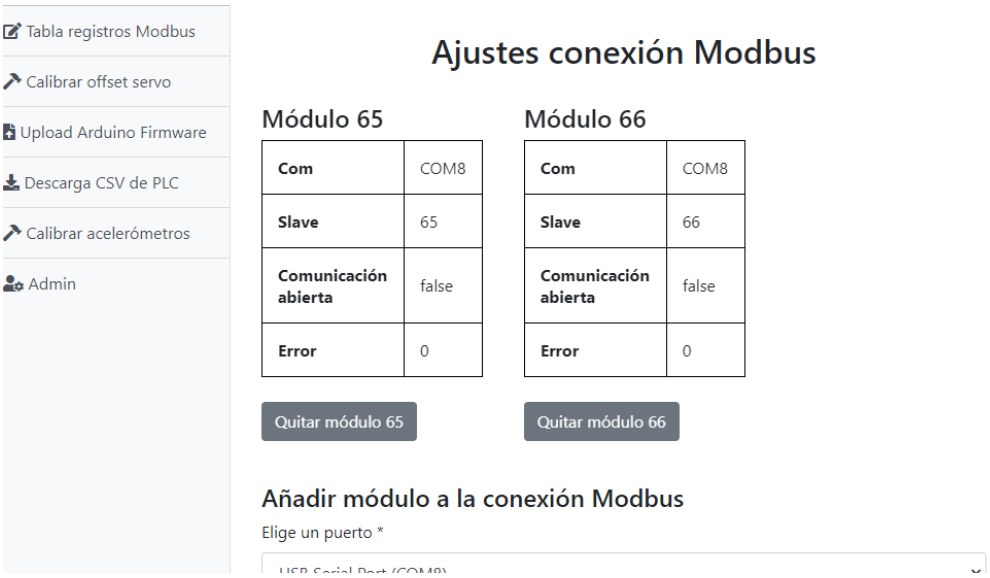


Ilustración 100 - Módulos añadidos

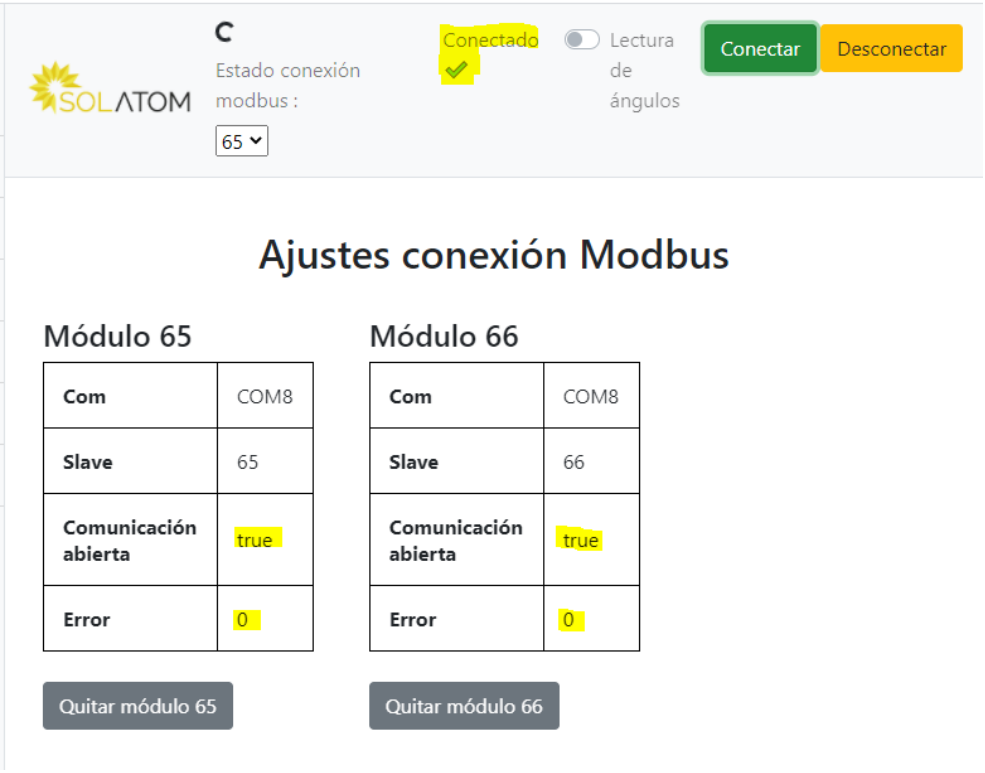


6.2.2. Resultados

Una vez están los módulos añadidos, se pulsa en "Conectar" y comienza la comunicación Modbus. Se observa el *spinner* en movimiento para indicar que la rutina está en funcionamiento y cuando se establece la comunicación se puede observar en el menú superior el texto de "Conectado ✔".

Además, en la página de "Ajustes Modbus" se puede ver por cada módulo si la comunicación se ha abierto o si presenta algún error.

Ilustración 101 - Feedback de la conexión Modbus



En la siguiente imagen se puede observar la consola del servidor, donde en cada comprobación de un módulo se puede ver por pantalla una línea con los ajustes del módulo y si tiene errores. En la siguiente línea se ve si se ha activado la lectura de ángulos o no.

El tiempo de comprobación de la comunicación está puesto a 0.5 segundos por cada módulo. Y como se puede ver en la siguiente imagen, cada 2 segundos hace 2 comprobaciones de cada módulo, por lo que la comunicación se realiza de forma correcta y con posibilidad de reducir el tiempo de comprobación.

Ilustración 102 - Peticiones de respuesta de los módulos 65 y 66

```
[14/Sep/2020 12:00:37] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 214
[14/Sep/2020 12:00:37] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 214
COM8 Slave 65 19200 Abierto True Angulo False Error 0
ANGLE READ = FALSE
COM8 Slave 66 19200 Abierto True Angulo False Error 0
ANGLE READ = FALSE
COM8 Slave 65 19200 Abierto True Angulo False Error 0
ANGLE READ = FALSE
COM8 Slave 66 19200 Abierto True Angulo False Error 0
ANGLE READ = FALSE
[14/Sep/2020 12:00:39] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 214
[14/Sep/2020 12:00:39] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 214
```

Por otro lado, se comprueba que al activar la lectura de ángulos en el módulo 65, comienza la lectura de los 10 ángulos de cada módulo solar. En la consola, se muestra por pantalla un *array* con los 10 ángulos leídos del módulo.

Ilustración 103 - Menú superior con comunicación iniciada

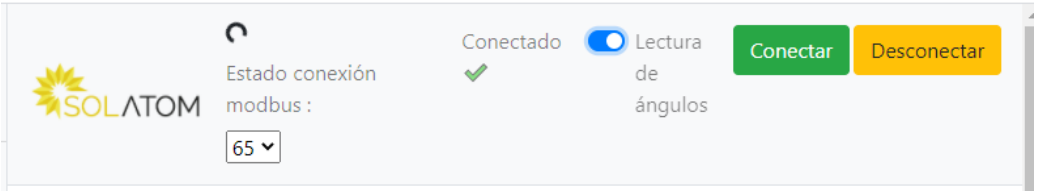


Ilustración 104 - Módulo 65 leyendo ángulos

```
[14/Sep/2020 12:01:23] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 213
[14/Sep/2020 12:01:23] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 213
COM8 Slave 65 19200 Abierto True Angulo True Error 0
Leyendo ángulos de los ejes
[-158.29, 2.05, -145.6, -152.09, -152.09, -152.09, -116.02, -152.09, -152.09, -152.09]
COM8 Slave 66 19200 Abierto True Angulo False Error 0
ANGLE READ = FALSE
COM8 Slave 65 19200 Abierto True Angulo True Error 0
Leyendo ángulos de los ejes
[-158.32, 2.05, -145.6, -152.09, -152.09, -152.09, -116.02, -152.09, -152.09, -152.09]
[14/Sep/2020 12:01:25] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 213
[14/Sep/2020 12:01:25] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 213
```

Si se activa la lectura de ángulos en el segundo módulo, la lectura se realiza correctamente y dentro de los tiempos de comprobación que se han marcado de 0.5 segundos por módulo.

Ilustración 105 - Menú superior con lectura de ángulos

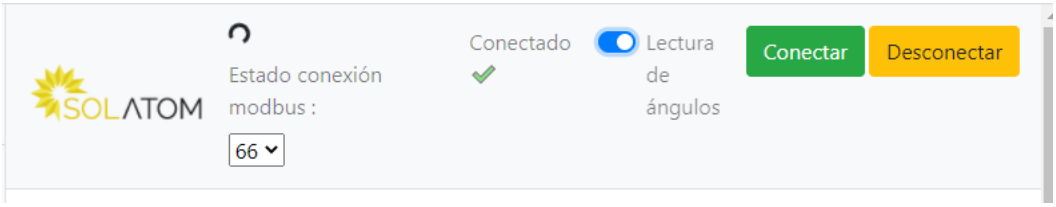
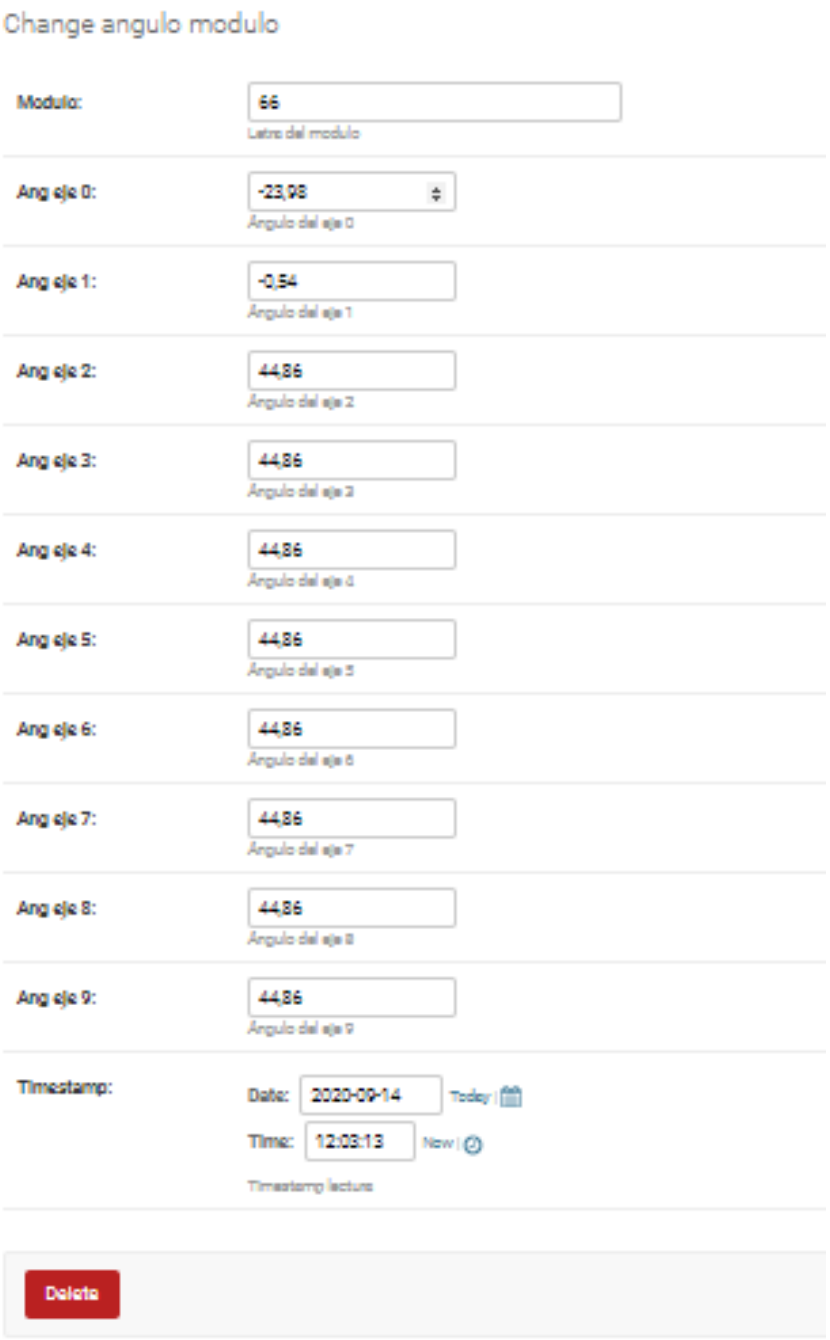


Ilustración 106 - Vista de dos módulos leyendo los ángulos

```
[14/Sep/2020 12:02:31] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 212
[14/Sep/2020 12:02:31] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 212
COM8 Slave 66 19200 Abierto True Angulo True Error 0
Leyendo ángulos de los ejes
[-26.17, -0.54, 44.86, 44.86, 44.86, 44.86, 44.86, 44.86, 44.86, 44.86]
COM8 Slave 65 19200 Abierto True Angulo True Error 0
Leyendo ángulos de los ejes
[-158.35, 2.05, -145.6, -152.09, -152.09, -152.09, -116.02, -152.09, -152.09, -152.09]
COM8 Slave 66 19200 Abierto True Angulo True Error 0
Leyendo ángulos de los ejes
[-25.35, -0.54, 44.86, 44.86, 44.86, 44.86, 44.86, 44.86, 44.86, 44.86]
[14/Sep/2020 12:02:33] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 212
[14/Sep/2020 12:02:33] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 212
```

Al activarse la lectura de ángulos, en la base de datos se van añadiendo las lecturas de estos ángulos, el módulo del que se ha leído y el instante en que se ha leído.

Ilustración 107 - Modelo base de datos de lectura de ángulos



Por último, se comprueba que al pulsar el botón de “Desconectar” la rutina de comunicación Modbus termina como se puede observar en la siguiente imagen:

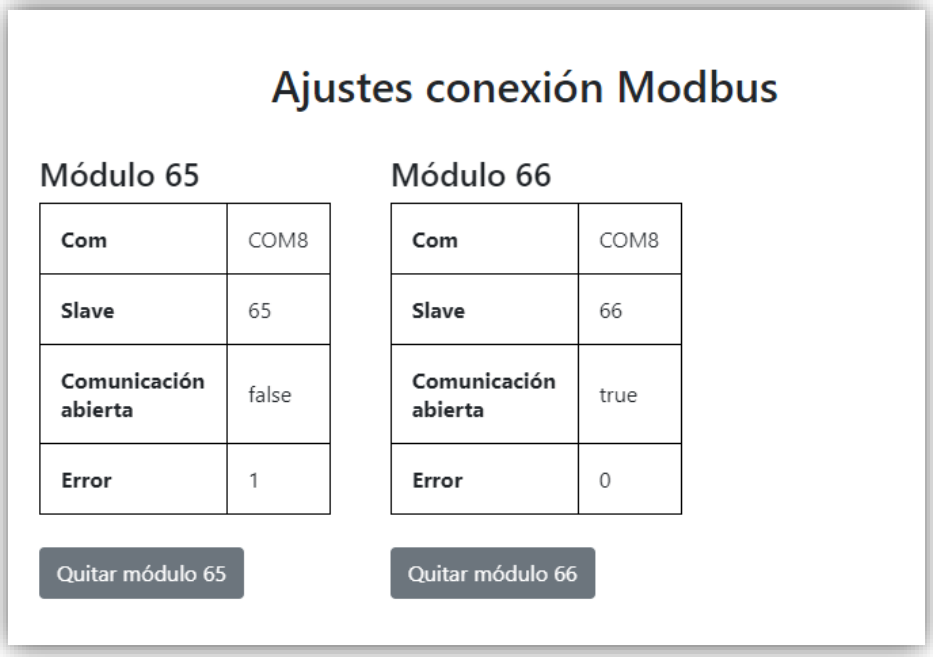
Ilustración 108 - Post de fin de rutina Modbus

```
Leyendo ángulos de los ejes
[-158.3, 1.96, -145.6, -152.09, -152.09, -152.09, -116.02, -152.09, -152.09, -152.09]
END RUTINA MODBUS
Hilo rutina modbus 3480
[14/Sep/2020 12:04:21] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 214
[14/Sep/2020 12:04:21] "GET /rw/ajax/ajustes_modbus/?state=0 HTTP/1.1" 200 214
Rutina modbus no tiene serial
END RUTINA Hilo rutina modbus 3480
[14/Sep/2020 12:04:22] "POST /rw/fin_rutina_modbus/ HTTP/1.1" 302 0
[14/Sep/2020 12:04:22] "GET /rw/settings_modbus/ HTTP/1.1" 200 15489
```

En la imagen anterior, se puede ver la respuesta a la petición POST del Ajax resaltada y en la línea anterior un debug para comprobar que la rutina realmente ha finalizado con su id de hilo de ejecución.

Por último, se comprueba que, si ha habido algún error en un módulo, se puede ver el feedback en la página de "Ajustes Modbus". En la siguiente imagen, se observa que al haber un error en el módulo 65, se muestra también que la comunicación se ha cerrado.

Ilustración 109 - Módulos añadidos



6.2.3. Análisis de resultados

El objetivo era desarrollar una forma de utilizar la comunicación Modbus para acceder a los registros de los módulos y realizar las calibraciones, por lo que la comunicación debía ser estable.

Después de realizar pruebas de comunicación se ha concluido que el sistema de hilos ejecutando en background proporciona la posibilidad de realizar otras tareas sin interferir en la ejecución de esta rutina Modbus. Además, la forma de controlar el número de módulos conectados a la comunicación Modbus es sencilla y fácil de usar.

Por lo que se ha conseguido desarrollar un algoritmo de comunicación Modbus a los módulos de una planta solar que funciona de forma estable, con la posibilidad de leer una serie de registros de varios módulos de forma ordenada y en poco tiempo. Se han realizado pruebas de lectura de 10 registros en una sola petición Modbus cada 0.5 segundos a varios módulos sin perder la conexión o sobrecargar el bus de datos, por lo que se podría llegar a ajustar más el tiempo de lectura, aunque no sería necesario ya que los ejes no giran tan rápido como para necesitar tiempos de muestreo menores de 1 segundo.

Además, con la gestión de todos los errores posibles ante un fallo de comunicación, el hilo se puede reestablecer automáticamente sin terminar el hilo, lo que añade seguridad y estabilidad a la comunicación.

La interfaz resulta sencilla de utilizar ya que desde el menú superior se puede gestionar la conexión o desconexión de la comunicación, tener *feedback* del estado de la conexión, controlar la lectura de ángulos y elegir el módulo sobre el que trabajar. Por lo que desde cualquier página se puede usar estos controles.

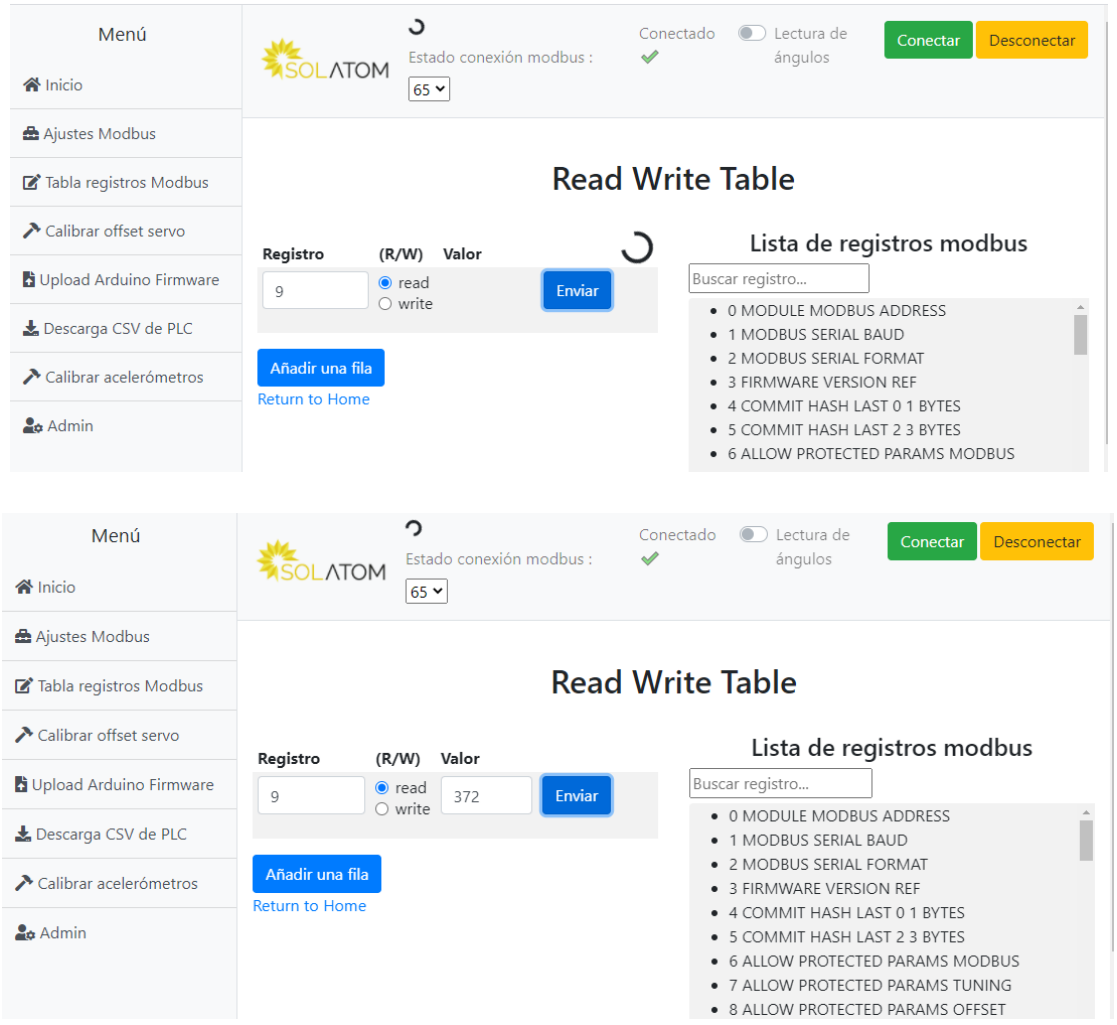
6.3. Lectura y escritura de registros

En este apartado se va a comentar los resultados de la página “Tabla registros Modbus” y si se ha cumplido el objetivo de leer y escribir registros.

6.3.1. Lectura de registros

Una vez iniciada la comunicación Modbus, se utiliza esta página para añadir registros al buffer de lectura o de escritura. Al añadir un registro y pulsar la opción de “read” se muestra un pequeño *spinner* en la primera fila de la tabla como indicativo de que se está procesando la petición. Al cabo de algo menos de 1 segundo, el *spinner* se ha vuelto invisible y se muestra el campo de “Valor” con el valor del registro que se ha pedido.

Ilustración 110 - Secuencia de envío de petición de lectura de registro

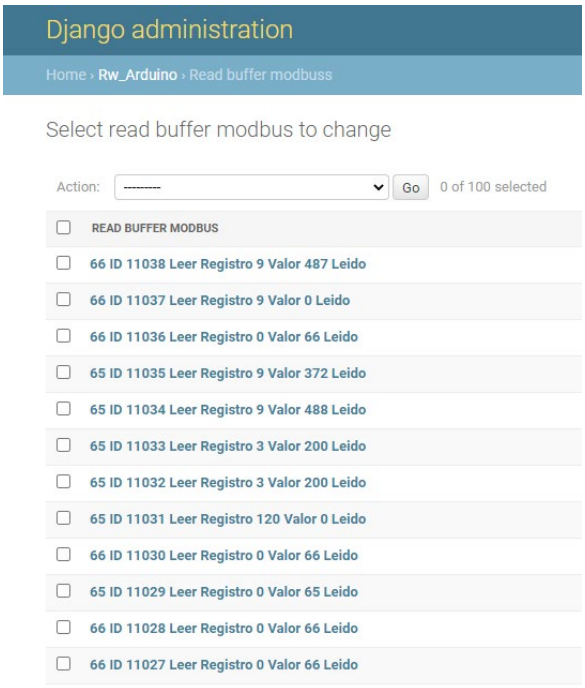


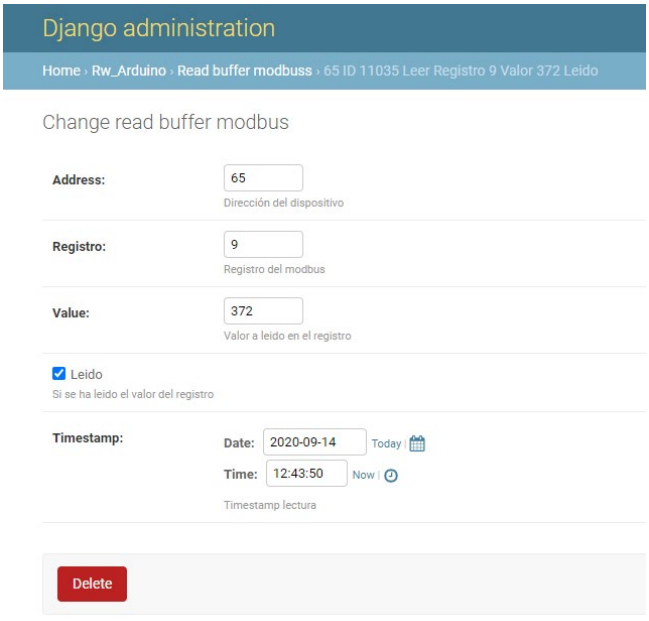
El proceso se realiza de forma muy rápida gracias a la rutina de comunicación Modbus que se está ejecutando en el *background.*

Además, si se entra en la página de “Admin” y se selecciona el modelo de *ReadBufferModbus* se puede ver una lista con todas las peticiones de lectura realizadas, y

entrando en una de las peticiones se puede obtener más información (dirección del módulo, el registro, el valor leído, si se ha completado la lectura y el instante en que se ha realizado la petición).

Ilustración 111 - Vista admin de Read buffer





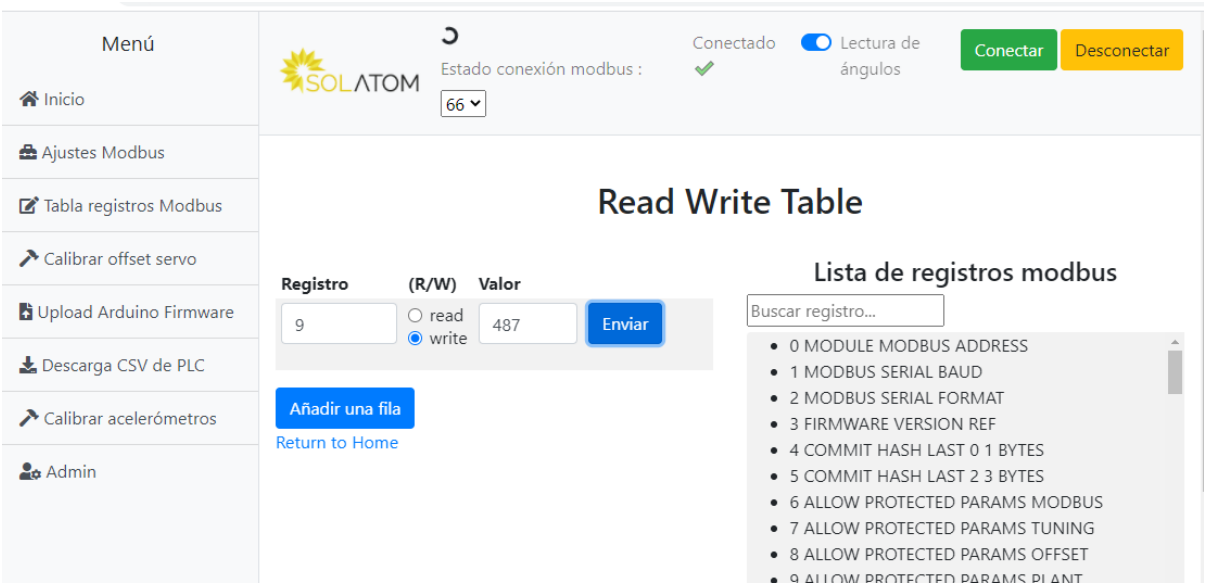
6.3.2. Escritura de registros

La escritura de registros se realiza de forma análoga, por lo que no necesita mayor explicación. La única diferencia es que para comprobar la escritura correcta del registro (debido

a que hay registros protegidos contra escritura), también se realiza una lectura después de la escritura.

En el caso de la siguiente imagen, se ha escrito el registro 9 (que es uno de los registros de protección de tipo cuenta atrás) con un valor de 500, y al realizarse la escritura, se observa en la línea que el valor cambia de 500 a 487, lo que quiere decir que desde que se ha lanzado la petición hasta que se ha escrito y leído han pasado 1.3 segundos.

Ilustración 112 - Vista de escritura de registro

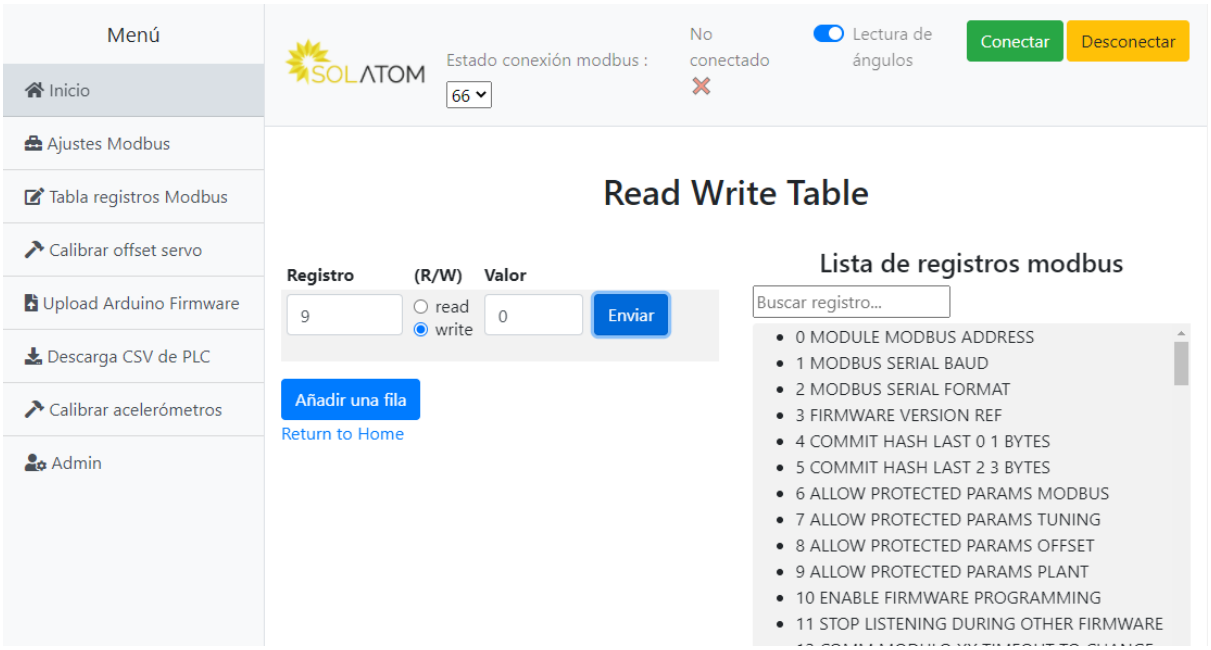


6.3.3. Gestión de errores

En el caso de que haya un error de comunicación y que la petición no se pueda realizar el valor devuelto será de 0 debido al *timeout* de la condición de concurrencia de ejecución de la rutina Modbus.

En la siguiente imagen, se puede observar que el resultado de la petición de escritura ha sido de 0, lo que puede llevar a alguna confusión. Sin embargo, en el menú superior se observa el texto que pone “No conectado”, por lo que da información al operario de que debe comprobar la comunicación.

Ilustración 113 - Error de escritura



6.3.4. Análisis de resultados

Ante los resultados obtenidos con este sistema de *buffers* de lectura y escritura, cabe destacar que funciona mejor de lo esperado, mostrando los valores de los registros en poco más de 1 segundo, con lo que es suficiente para los módulos solares, ya que no es necesario un control con bajo tiempo de muestreo.

Además, gracias a la interfaz sencilla y con la posibilidad de añadir varios registros y cambiar de módulo fácilmente, resulta una herramienta muy útil a la hora de consultar algunos registros para comprobar el estado de las variables internas. Y en la lista de registros se puede buscar rápidamente el registro que se necesite con una palabra clave.

Por último, al tratarse de *buffers* de datos, se guarda el registro de las peticiones realizadas en la base de datos para posibles consultas, mejoras, mantenimiento preventivo…

6.4. Rutina automática de calibración del offset de los servos

En cuanto a la calibración automática de los servos, es uno de los objetivos que más tiempo ha consumido del trabajo por lo diferentes retos que ha supuesto a lo largo del desarrollo además de por realizar comprobaciones en el banco de ensayos y en una planta solar real.

6.4.1. Resultados del banco de ensayos

El banco de ensayos es un pequeño prototipo que simula el comportamiento de dos ejes de espejos de un módulo solar. Consta de una caja de electrónica y dos estructuras de engranajes con el servo que controla el giro y el acelerómetro que mide el ángulo. La caja de electrónica tiene dos cables de Ethernet: uno para conectarlo al ordenador y otro para conectarle un mando LCD.

Ilustración 114 - Banco de ensayos

Por lo tanto, para probar el algoritmo de calibración de los servos se ha usado este banco de ensayos con el firmware de la placa de electrónica actualizada y conectado al ordenador con el conversor USB serial a RS 485 vía cable de Ethernet.

- ***Primer reto: ruido de los acelerómetros***

El primer obstáculo que se encontró fue el hecho de que los acelerómetros presentan ruido en la lectura del ángulo, por lo que para hacer comparaciones de ángulos y realizar varias lecturas se debe usar un rango de error. Además, este problema es mayor en instalaciones reales ya que pequeñas ráfagas de viento pueden mover ligeramente los espejos provocando errores de incluso varios grados, por lo que el rango de error de comparación debía ser algo mayor.

Después de muchas pruebas con el banco de ensayos e iteraciones en el algoritmo se consiguió automatizar el proceso de calibración según se explica en el punto de Calibración de

Servos en el desarrollo del proyecto. Sin embargo, surgió el segundo problema: la rutina tardaba demasiado por cada espejo.

- ***Segundo reto: tiempo de calibración***

Por lo tanto, el segundo reto fue conseguir reducir este tiempo de calibración deshabilitando algunas funciones del módulo, como el *auto next axis,* el cual cambiaba de eje cuando pasaba un determinado tiempo. Pero la mejora que mayor resultado dio fue la de ajustar tiempos de espera entre lecturas del ángulo para ver si había movimiento o no. Con esto y la mejora de la comunicación Modbus, se consiguió calibrar un espejo en no más de 2 minutos.

- ***Tercer reto: evitar la rotura del cable del sensor***

Otro de los problemas que pueden surgir al realizar la calibración automática de los servos, que se comprueba durante las pruebas, es el hecho de que el eje gire de forma descontrolada varias vueltas seguidas en el mismo sentido, llegando a doblar e incluso romper el cable del acelerómetro, lo que provocaría la necesidad de reemplazar este sensor completamente.

Por lo tanto, existe la necesidad de controlar el giro del eje y evitar que supere los ángulos de 90 y -90 grados. Para ello, se desarrolla una función que comprueba el ángulo y devuelve el eje a la posición inicial para evitar este problema.

6.4.2. Resultados de una instalación real

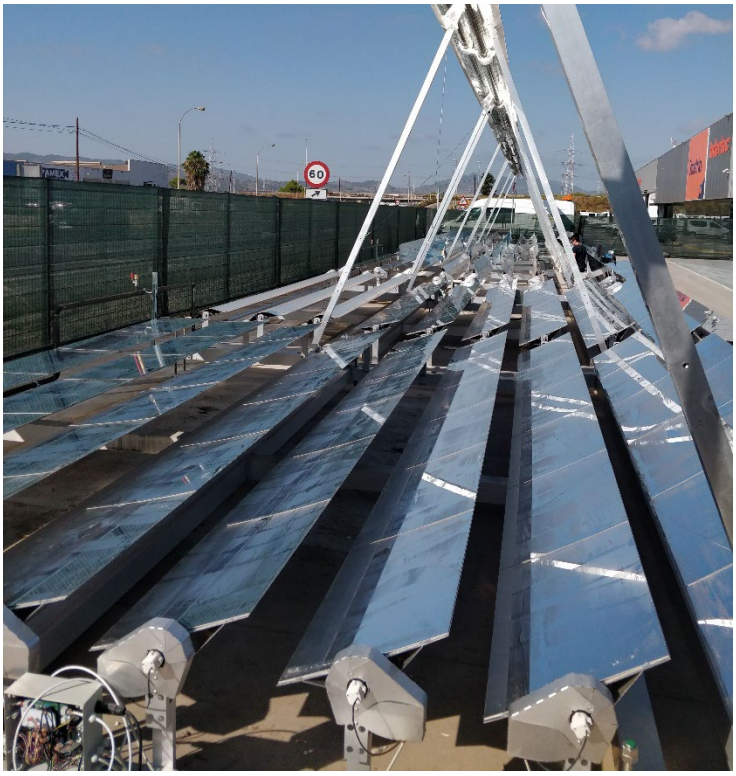
Una vez probado y validado el algoritmo con el banco de ensayos, se pudo realizar una prueba en una instalación real en Castellón. Para realizar las pruebas se procedió desconectando el módulo a calibrar de la red de comunicación Modbus de la planta para realizar el seguimiento del algoritmo en modo *debug* lanzado directamente desde el ordenador.

- ***Cuarto reto: montaje de los servos***

Tras varias pruebas se pudo ver que algunos ejes los podía calibrar sin problema, pero en otros siempre fallaba al comprobar el valor de offset final. Probando a realizar la calibración de forma manual se llegó a la conclusión de que debido a que los ejes no están montados de forma simétrica hay unos motores que giran en un sentido y otros en el sentido contrario, por lo que la calibración debía ser diferente según como estuviera montado el eje.

Una vez solucionado este problema cambiando el sentido de calibración según el montaje, se pudo realizar una pequeña prueba que calibraba los ejes da igual cual fuera el sentido de montaje

Ilustración 115 - Instalación planta solar



6.4.3. Análisis de resultados

Como análisis de los resultados, se observa que los ejes se pueden calibrar con el algoritmo desarrollado y de forma autónoma, por lo que se puede afirmar que se ha cumplido el objetivo con esta primera versión funcional del algoritmo, aunque tiene posibilidades de mejora.

Después de superar los retos durante el desarrollo se ha conseguido calibrar varios ejes de forma satisfactoria, aunque de momento no se ha podido testear a gran escala en una instalación real para comprobar la eficacia y el número total de ejes calibrados automáticamente.

Sin embargo, aunque no pueda calibrar el 100% de los ejes automáticamente, el algoritmo guarda en la base de datos los ejes que se han calibrado correctamente y los que han dado algún fallo. Por lo que se puede comprobar manualmente por qué esos ejes habrían fallado.

6.5. Actualización del firmware de la planta

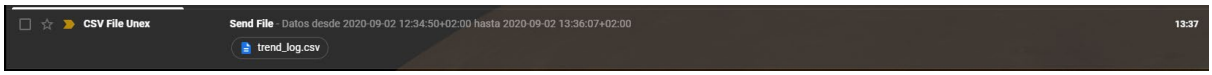
Durante la visita a la instalación de Castellón también se pudo probar el software de actualización del firmware de la planta. Los resultados fueron positivos ya que se cumplió el objetivo de actualizar el firmware de los módulos, pero con el ordenador como servidor y conectado a la placa de desarrollo roja vía USB. Al actualizar el firmware se pudo ver por la página web el estado del comando de actualización de firmware *avrdude*.

La opción de actualizar el firmware de forma remota todavía no se ha podido probar debido a algunos problemas disponibilidad de la instalación solar, pero próximamente se realizará una prueba, ya que es una funcionalidad muy útil para la empresa.

6.6. Descarga CSV de datos

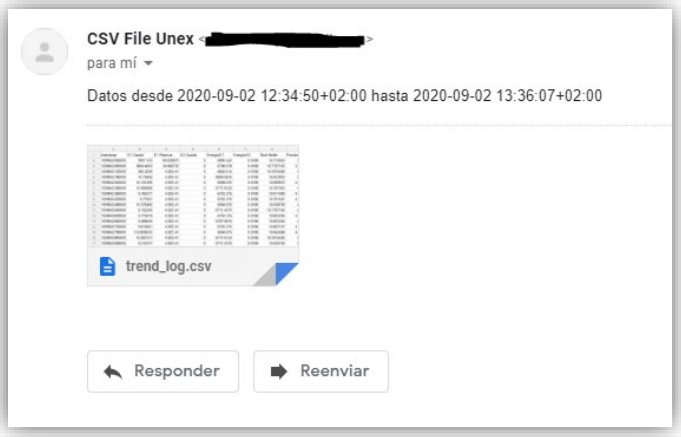
Al realizar la petición del formulario se envía un correo con el archivo de datos CSV, cuyo formato sería el siguiente:

Ilustración 116 - Email de descarga de CSV



En el correo se indica el intervalo de tiempo solicitado y se adjunta el CSV con los datos solicitados.

Ilustración 117 - Contenido de email de descarga de CSV



En el correo que recibe el usuario no aparece nadie más en copia, pero se ha añadido el correo de la empresa en copia oculta para tener un seguimiento de las peticiones de correo.

El archivo de CSV quedaría de la siguiente forma:

Ilustración 118 - CSV

timestamp	E1 Caudal	E1 Potencia	E2 Caudal	Energía E1	Energía E2	Nivel Kettle	Presión de Vapo	Presión P1	Presión P2	Presión P3	Presión P5	Radiación Solar	Temperatura Am	Temperatura T1	Temperatura T2	Temperatura T3	Temperatura
1599042000000	5.957.223	60.626.873	0.0	6.695.442	188	10.713.043	3.187.057	23.282.514	28.361.425	129.237.175	0.0	7.508.346	2.791.465	6.383.721	7.276.196	25.2	80.32
1599042060000	58.944.063	24.066.755	0.0	6.708.078	188	107.797.165	39.087.074	23.566.856	28.302.596	129.237.175	0.0	75.436.426	27.475.391	61.505.093	65.461.006	25.21	80.44
1599042120000	2.802.094	4,92E-37	0.0	6.692.615	188	107.875.595	4.247.403	27.733.915	27.856.476	13.237.476	0.0	75.806.555	2.737.342	674.854	6.252.165	25.24	80.53
1599042180000	1.070.484	4,92E-37	0.0	66.895.625	188	10.822.853	4.131.766	27.601.547	27.719.207	129.237.175	0.0	75.801.654	27.383.224	6.722.817	6.349.858	25.27	80.54
1599042240000	10.101.499	4,92E-37	0.0	6.698.625	188	10.689.507	41.321.497	2.760.645	27.704.499	12.609.959	0.0	7.580.411	27.542.065	6.708.079	63.495.293	25.29	80.45
1599042300000	10.980.958	4,92E-37	0.0	67.178.125	188	10.767.952	4.164.665	2.760.645	27.694.695	131.464	0.0	75.799.207	27.440.094	6.706.667	63.217.995	25.33	80.48
1599042360000	9.785.377	4,92E-37	0.0	6.702.375	188	10.811.089	41.692.533	27.581.937	27.699.597	9.158.611	0.0	76.152.185	27.304.787	6.742.511	63.179.035	25.33	80.48
1599042420000	977.941	4,92E-37	0.0	6.702.375	188	10.791.481	41.768.675	2.758.684	27.699.597	7.903.576	0.0	76.517.413	2.751.461	6.725.667	63.114.475	25.36	80.34
1599042480000	10.376.466	4,92E-37	0.0	6.694.875	188	10.528.709	4.149.974	27.591.743	27.699.597	68.712.234	0.0	76.512.506	27.649.921	6.743.038	6.334.303	25.36	80.34
1599042540000	9.192.245	4,92E-37	0.0	67.114.375	188	107.797.165	4.153.612	27.601.547	27.689.793	68.712.234	0.0	7.651.006	27.804.832	6.711.257	63.312.023	25.4	80.27
1599042600000	9.775.419	4,92E-37	0.0	6.702.375	188	10.803.246	41.277.037	2.760.645	27.689.793	16.283.989	0.0	7.651.496	27.773.457	67.263.725	63.533.375	25.42	80.19
1599042660000	9.486.659	4,92E-37	0.0	67.076.875	188	10.803.246	4.129.508	27.557.425	27.714.305	0.0	0.0	77.228.265	27.889.156	674.523	63.517.918	25.45	80.11
1599042720000	9.815.461	4,92E-37	0.0	6.702.375	188	10.807.167	41.040.635	27.513.304	27.694.695	0.0	0.0	7.722.581	27.846.012	6.676.779	63.736.523	25.45	79.99
1599042780000	103.856.535	4,92E-37	0.0	6.694.875	188	10.842.468	40.780.706	27.577.035	27.704.499	0.0	0.0	77.228.265	27.844.051	66.942.535	63.961.212	25.49	79.88
1599042840000	10.961.917	4,92E-37	0.0	67.178.125	188	107.875.595	4.080.965	2.760.645	27.694.695	0.0	0.0	77.404.755	27.912.685	6.659.416	63.936.123	25.5	79.79
1599042900000	9.214.751	4,92E-37	0.0	67.114.375	188	10.830.704	4.083.466	27.630.963	27.714.305	0.0	0.0	7.775.772	27.877.388	6.662.461	6.391.446	25.55	79.66
1599042960000	9.169.005	4,92E-37	0.0	67.114.375	188	10.791.481	40.422.816	27.630.963	27.719.207	0.0	0.0	7.811.805	27.791.107	66.626.884	64.272.896	25.58	79.57
1599043020000	10.101.237	4,92E-37	0.0	6.698.625	188	10.744.415	40.445.213	27.630.963	27.729.013	0.0	0.0	78.115.594	2.795.779	66.613.815	6.425.331	25.62	79.42
1599043080000	10.121.546	4,92E-37	0.0	6.698.625	188	10.764.023	40.487.385	27.630.963	27.714.305	0.0	0.0	7.811.805	28.102.905	66.615.456	6.421.646	25.65	79.31
1599043140000	9.775.589	4,92E-37	0.0	6.702.375	188	10.760.101	39.939.167	27.581.937	27.729.013	0.0	0.0	78.471.027	28.126.434	6.677.969	6.469.847	25.68	79.19
1599043200000	9.787.435	4,92E-37	0.0	6.702.375	188	107.875.595	39.989.038	2.768.489	27.719.207	0.0	0.0	78.649.963	27.995.049	667.816	6.465.435	25.7	79.08
1599043260000	9.788.213	4,92E-37	0.0	6.702.375	188	1.069.735	40.033.956	2.766.528	27.743.719	0.0	0.0	78.836.255	28.018.581	6.663.332	6.461.466	25.7	79.08
1599043320000	9.800.272	4,92E-37	0.0	6.702.375	188	10.850.311	40.072.317	27.640.767	27.729.013	0.0	0.0	788.338	28.193.108	66.784.775	64.580.795	25.74	78.87
1599043380000	8.903.879	4,92E-37	0.0	6716.75	188	10.838.547	4.011.018	2.762.606	27.733.915	0.0	0.0	7.917.943	2.850.098	66.822.655	6.454.741	25.78	78.72
1599043440000	9.775.786	4,92E-37	0.0	6.702.375	188	10.834.625	39.564.068	27.650.573	27.733.915	0.0	0.0	7.918.188	28.375.477	6.678.688	650.321	25.78	78.72
1599043500000	9.477.578	4,92E-37	0.0	67.076.875	188	10.701.279	39.595.966	27.699.597	27.758.427	0.0	0.0	79.542.206	28.349.983	66.974.655	650.036	25.8	78.61
1599043560000	887.976	4,92E-37	0.0	6716.75	188	1.075.618	3.909.064	27.655.475	27.733.915	0.0	0.0	79.547.107	28.193.108	6.632.617	6.545.778	25.83	78.48

En líneas generales, todo el proceso hasta que se recibe el correo no suele tardar más de 5-10 minutos máximo. Sin embargo, se ha añadido un mensaje de advertencia de que no se pida un intervalo de más de un día puesto que el hilo tarda demasiado en procesar la respuesta.

6.6.1. Análisis de resultados

Esta funcionalidad del proyecto se ha comprobado que se ejecuta correctamente y mejora el actual método de descarga de datos en que añade más libertad a la hora de escoger un intervalo de tiempos y reduce considerablemente la cantidad de datos 3G que se usan para enviar los datos.

Por otro lado, el algoritmo de gestión de datos se puede mejorar notablemente para reducir el tiempo de ejecución y poder pedir intervalos de tiempo mayores de 1 día sin problemas.

6.7. Calibración acelerómetros

Para esta calibración se ha usado la misma configuración que en la comprobación de la comunicación Modbus para poder mover los acelerómetros de forma manual y comprobar que cambian los valores en la página web. Sin embargo, no se ha podido realizar una prueba con una planta solar real, por lo que los acelerómetros no llegan a calibrarse y termina el algoritmo de calibración del Arduino porque detecta un fallo de giro de los sensores.

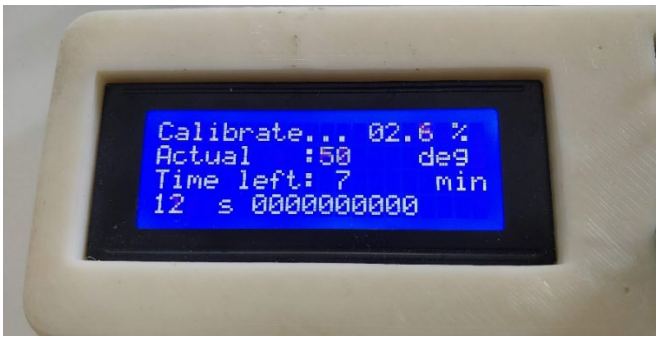
Durante las pruebas con las placas de desarrollo se ha podido probar que la transferencia de datos es correcta entre la rutina de calibración de los acelerómetros; la rutina modbus para lectura y escritura de registros; y las peticiones de Ajax con Javascript.

En la siguiente imagen, se observa la pantalla LCD del mando de control al lanzar la rutina de calibración de los acelerómetros desde la página web. El algoritmo del Arduino está esperando a que los ejes se posicionen de una determinada forma para favorecer la calibración de los acelerómetros.

Ilustración 119 - Mando LCD calibrando acelerómetro

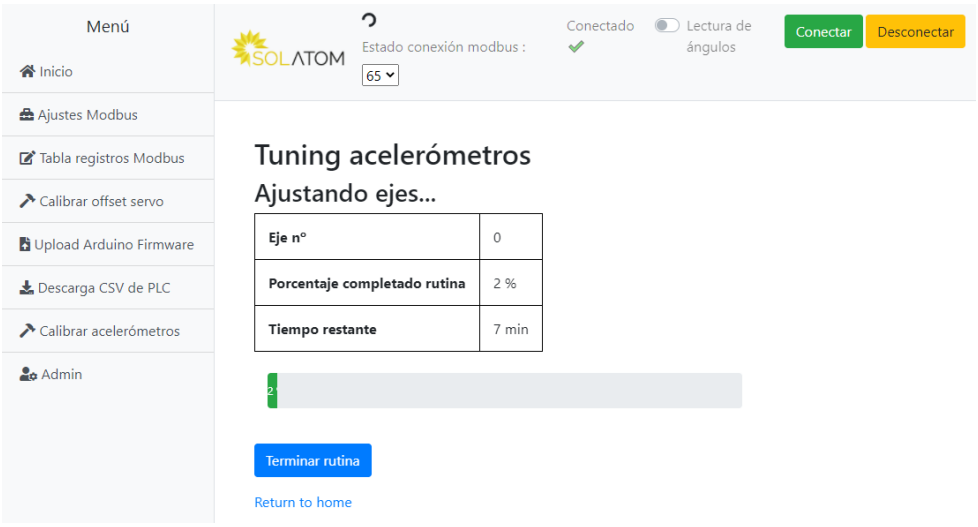
A continuación, comienza la calibración y muestra por el LCD el porcentaje de calibración completado, el ángulo del eje actual y el tiempo restante de calibración.

Ilustración 120 - Mando LCD calibrando acelerómetro



Para comprobar que el funcionamiento es correcto, se puede verificar que el valor de porcentaje y tiempo restante en la pantalla LCD es el mismo que en la página web, por lo que la comunicación es correcta.

Ilustración 121 - Porcetaje de calibración de acelerómetro



Desde esta página también se comprueba que se puede terminar la rutina de calibración pulsando el botón de "Terminar rutina" y que funciona correctamente, terminando la rutina de calibración también en el Arduino.

6.7.1. Análisis de resultados

Como se ha podido comprobar, se ha cumplido el objetivo de realizar el seguimiento de la calibración automática de los acelerómetros. Además el añadido es que se puede lanzar el algoritmo de forma remota y tener seguimiento del estado de la calibración en una interfaz visual sencilla.

6.8. Conexión remota

El hecho de poder realizar las tareas presentadas en este trabajo de forma remota facilita de forma significativa el seguimiento y control de las instalaciones, por lo que ha sido un gran avance para la parte técnica de la empresa. Además, la redirección web permite tener un acceso fácil desde cualquier dispositivo al estado de las plantas solares.

Desde el servidor central se puede tener seguimiento del estado de las conexiones abiertas, aunque gracias al relanzamiento automático no es necesario estar pendiente de si alguna conexión ha fallado puesto que a los pocos minutos se vuelve a reestablecer.

Por otro lado, si en algún momento fuera necesario añadir o cambiar las conexiones y redirecciones sería tan sencillo como editar el fichero que aparece a continuación y usar la misma estructura que con los proyectos que ya están redireccionados. De esta forma en un mismo fichero se pueden tener controlados todos los puertos e ips de los proyectos.

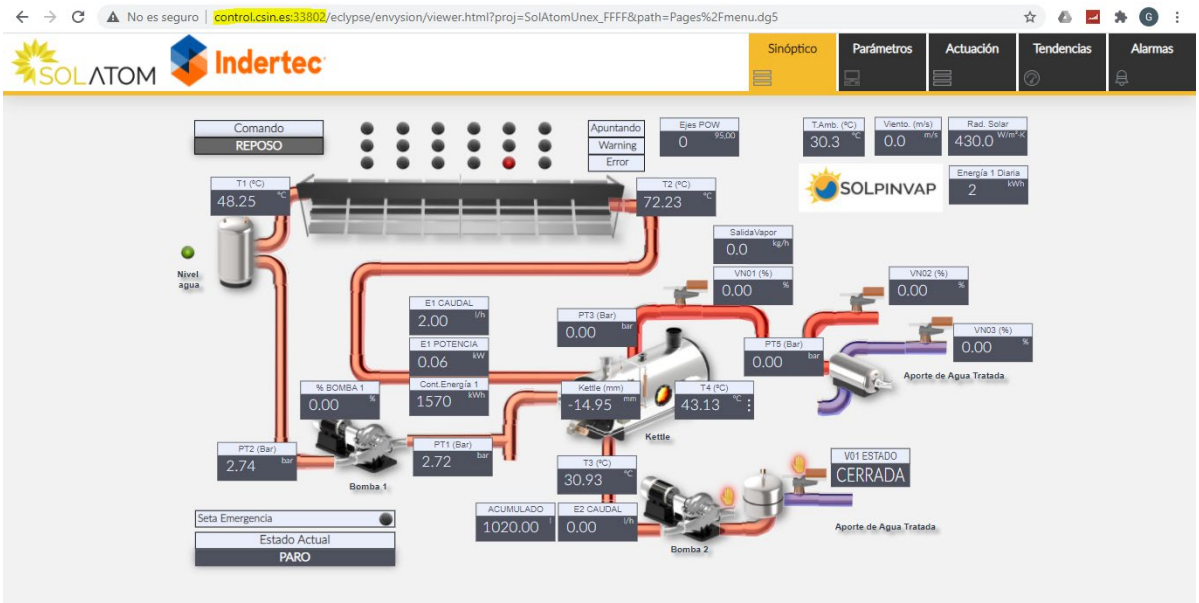
Ilustración 122 - Fichero ssh config

```
root@vps798550:~# cat ~/.ssh/config
Host Unex
    HostName 127.0.0.1
    Port 33100
    User pi
    LocalForward *:33800 192.168.0.30:80
    RequestTTY no
    ExitOnForwardFailure yes
    ControlMaster auto
    ControlPath ~/.ssh/cm_sockets/%r@%h:%p

Host UnexAPI
    HostName 127.0.0.1
    Port 33100
    User pi
    LocalForward *:33900 192.168.0.12:8000
    RequestTTY no
    ExitOnForwardFailure yes
    ControlMaster auto
    ControlPath ~/.ssh/cm_sockets/api%r@%h:%p

Host Itc
    HostName 127.0.0.1
    Port 33101
    User pi
    LocalForward *:33801 10.141.188.197:80
    RequestTTY no
    ExitOnForwardFailure yes
    ControlMaster auto
    ControlPath ~/.ssh/cm_sockets/%r@%h:%p

Host Solpinvap
    HostName 127.0.0.1
    Port 33202
    User pi
    LocalForward *:33802 192.168.0.30:80
    RequestTTY no
    ExitOnForwardFailure yes
    ControlMaster auto
```

La implementación de esta solución también ha servido para crear un acceso directo al HMI del PLC de las plantas solares, lo que facilita el control de las plantas de forma totalmente remota. En la siguiente imagen, se puede ver el acceso al PLC de la planta Solpinvap mediante

la redirección del puerto 33802 del servidor central (como se indica en la imagen anterior del fichero *config*).

Ilustración 123 - Página HMI del PLC por redirección SSH



6.8.1. Análisis de resultados

Los resultados de esta parte son realmente satisfactorios y cumplen totalmente el objetivo de poder realizar tareas de control y supervisión de las plantas de forma remota. Se trata de un gran avance por las posibilidades que ofrece y la estabilidad del sistema, ya que se relanza de forma automática en caso de fallo, por lo que el mantenimiento es mínimo.

7. Conclusiones

El presente trabajo ha supuesto un avance en la mejora del mantenimiento y supervisión de las plantas solares después de la instalación, con lo cual es un añadido importante al producto que ofrece la empresa a sus clientes. El sistema es un complemento al control que realiza el PLC, mejorando algunos aspectos como la descarga de datos en formato CSV y añadiendo otras funcionalidades que resultan útiles durante la instalación, *comissioning* y posterior mantenimiento.

La aplicación se encuentra disponible de forma local en la red wifi de la planta y de forma remota desde cualquier lugar con conexión a internet gracias a la redirección mediante el servidor central de la empresa. Este aspecto supone una mejora no sólo para la aplicación desarrollada en este TFM, sino también para la supervisión y control de las plantas que ya están en funcionamiento. De esta forma, se puede acceder de forma remota con un navegador web al HMI del PLC que realiza el control de la planta para así ajustar variables de control y detectar los fallos en menor tiempo y con más detalle para mejorar el producto de cara a futuros proyectos reduciendo costes económicos y de tiempo.

En general, se cumple el objetivo de desarrollar un sistema de calibración y mantenimiento de los distintos sistemas de las plantas solares mediante comunicación Modbus. De forma específica, todas las funcionalidades que se han planteado al inicio de este trabajo han sido desarrolladas satisfactoriamente, algunas de forma más óptima que otras pero todas han cumplido el objetivo que se proponía.

La calibración automática del offset de los servos ha sido uno de los aspectos que más tiempo ha requerido del proyecto, no por la complejidad del algoritmo, sino por el hecho de realizar pruebas tanto en el banco de ensayos como en una instalación real para conseguir que el algoritmo funcione correctamente pese a los retos que ha supuesto durante el desarrollo. Una vez se comience a usar durante el *comissioning* de los proyectos, se podrá observar qué aspectos se pueden mejorar y optimizar el algoritmo para que el tiempo de la instalación se reduzca notablemente.

Otro de los aspectos que mejora este proyecto respecto al anterior sistema es la capacidad de actualizar el firmware de los módulos de forma totalmente remota. Actualmente, la parte de software está implementada en este trabajo y requiere implementar una PCB electrónica para controlar el cambio de comunicación y habilitar el reseteo de los módulos. Sin embargo, las pruebas que se han realizado son satisfactorias, por lo que se podrá utilizar la funcionalidad desarrollada en este proyecto.

Por lo tanto, este proyecto ofrece una base sobre la que trabajar para mejorar diversos aspectos y conseguir desarrollar una aplicación que realice el mantenimiento de las plantas solares de forma totalmente autónoma en un futuro cercano.

8. Recomendaciones y trabajos futuros

En este apartado se va a comentar las mejoras que se irán implementando en las distintas funcionalidades a medida que se prueben a fondo en instalaciones reales para llegar a conseguir el control de mantenimiento automático.

- ***Comunicación Modbus**:* la comunicación Modbus se puede mejorar en que llegue un momento en que no sea necesario la actuación de un PLC de control de los módulos. Para ello se debe conseguir una comunicación sin fallos, para mantener los módulos controlados con total seguridad. En un futuro cercano se puede implementar la lectura de variables mediante la API del PLC para tener la opción de leer registros específicos sin necesidad de establecer ningún tipo de comunicación cableada con el sistema.

- ***Calibración de offset de los servos**:* las posibles mejoras que se pueden realizar de este apartado serían realizar una mejora en el tiempo de ejecución del algoritmo ejecutando varios módulos de forma simultánea o reduciendo los intervalos de tiempos de espera entre lecturas. Otra opción sería el hecho de leer otras variables disponibles como el consumo de corriente de los servos para comprobar si está en funcionamiento o no. Por último, se podría añadir el método de la bisección para el cálculo del valor óptimo de jog speed, aunque no sería prioritario puesto que los valores de jog speed son limitados.

- ***Upload firmware**:* en un futuro cercano la mejora que se va a realizar es la implementación de una placa de electrónica en cada planta para poder controlar de forma remota, y sin necesidad de un operario en la planta, la actualización del firmware de los módulos solares. Otra de las mejoras que se podría implementar el hecho de tener un control sobre la versión instalada en cada fábrica mediante el hash del *commit* de la versión de *git*.

- ***Descarga de datos en formato CSV**:* en este apartado sería interesante gestionar la gran cantidad de datos que se pueden recibir de forma más eficiente para reducir el tiempo de respuesta y que en pocos segundos se pueda descargar un archivo CSV.

- ***Mantenimiento inteligente**:* este aspecto resulta interesante ya que la propia planta realizaría las rutinas de calibración que se han desarrollado en este proyecto de forma periódica cada cierto tiempo, lo que conllevaría una reducción del gasto en mantenimiento. Además, se podría añadir la recopilación de alarmas para crear estadísticas de frecuencia de alarmas para crear un mantenimiento inteligente.

- ***Supervisión:*** otro aspecto que sería interesante implementar es la mejora de la parte de supervisión con la visualización de los datos de operación y la posibilidad de enviar comandos de operación además de monitorizar el estado de alarmas en una misma página como un HMI.

9. Referencias bibliográficas

Alonso, N. O. (2013). *Redes de comunicaciones industriales*. UNED. Retrieved from https://books.google.es/books?hl=es&lr=&id=4TKJ9IpMSJEC&oi=fnd&pg=PP1&dq=protocolo+de+comunicación+industrial&ots=gTHyBsKsvp&sig=wVHvzJbNJIDuoiSsJXwJ4dshxTo#v=onepage&q=protocolo de comunicación industrial&f=false

Bokeh Org. (n.d.). Bokeh. Retrieved from https://bokeh.org/

Django Framework. (n.d.). Retrieved from https://www.djangoproject.com/

Eclypse. (n.d.). Eclypse API reference. Retrieved from http://eclypseapi.distech-controls.com/restApi/bacnet/resources/1081/GET

European Solar Thermal Electricity Association. (n.d.). LINEAR FRESNEL REFLECTOR. Retrieved from https://www.estelasolar.org/techologies-plants/the-4-types-of-csp-electricity-technologies/linear-fresnel-reflector/

Información Detallada sobre el Protocolo Modbus. (2019). *Ni.Com*. Retrieved from https://www.ni.com/es-es/innovations/white-papers/14/the-modbus-protocol-in-depth.html

Ionos. (2020). CSRF: explicación del ataque Cross Site Request Forgery. *Ionos.Es*. Retrieved from https://www.ionos.es/digitalguide/servidores/seguridad/cross-site-request-forgery/

Lenguajes de programación backend. (n.d.). *Index Desarrollo*. Retrieved from https://indexdesarrollo.com/lenguajes-de-programacion-web-mas-recomendables/

LogicBus. (n.d.). Protocolos de comunicación industriales. *Logicbus.Com.Mx*. Retrieved from https://www.logicbus.com.mx/blog/protocolos-de-comunicacion-industriales/

Modbus Tools. (n.d.). Modbus Poll. Retrieved from https://modbustools.com/modbus_poll.html

Modbus vs RS485. (n.d.). *Virtual Serial Port Org*. Retrieved from https://www.virtual-serial-port.org/es/articles/modbus-vs-rs485/

Nongnu Org. (n.d.). AVRDUDE. Retrieved from https://www.nongnu.org/avrdude/

raspberrypi.org. (2020). Raspberry Pi 4. Retrieved from https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

remote.it. (n.d.-a). Peer to Peer vs Proxy Connections. Retrieved from https://docs.remote.it/peer-to-peer-p2p-vs.-proxy-connections

remote.it. (n.d.-b). Using connectd in P2P initiator mode on macOS and Linux. Retrieved from https://docs.remote.it/advanced-users/using-connectd-in-p2p-initiator-mode/connectd

Revision, J. B. (2019). MinimalModbus. Retrieved from https://minimalmodbus.readthedocs.io/en/stable/apiminimalmodbus.html#minimalmodbus.Instrument.read_registers

SSH Protocol. (n.d.). *Ssh.Com*. Retrieved from https://www.ssh.com/ssh/protocol/

SSH Tunnel - Local and Remote Port Forwarding Explained With Examples. (n.d.). *Trackets Blog*. Retrieved from https://blog.trackets.com/2014/05/17/ssh-tunnel-local-and-remote-port-forwarding-explained-with-examples.html

Virtualenv. (n.d.). Retrieved from https://virtualenv.pypa.io/en/latest/

Witte Software. (2020). Modbus Protocol. Retrieved from https://www.modbustools.com/modbus.html