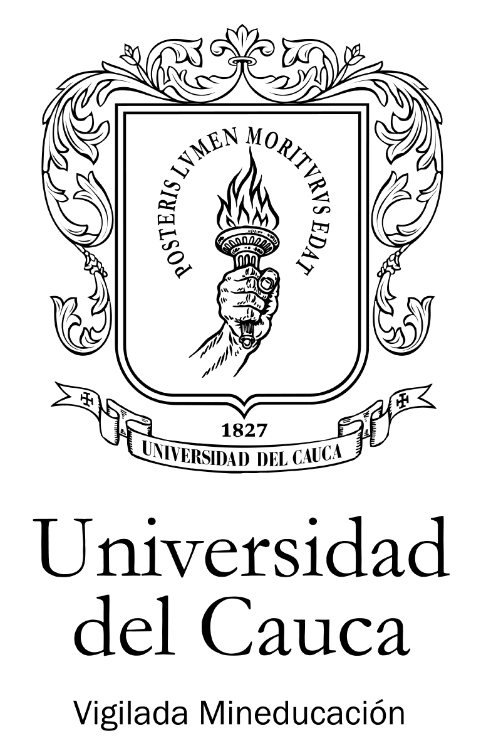
**SmartPlant: Sistema IoT Semántico para Monitoreo de Plantas de Interior**



Web Semántica

Presentado por:

Santiago Alexander Dorado Gomez

Profesor:

Miguel Angel Niño

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Ingeniería de Sistemas

Popayán, diciembre 2025

Introducción

El cuidado adecuado de plantas de interior requiere conocer y controlar variables ambientales como la temperatura, la humedad y la iluminación. Sin embargo, la mayoría de los usuarios no cuenta con instrumentos especializados ni con conocimientos técnicos para interpretar estos datos. Las tecnologías de Internet de las Cosas (IoT) y la Web Semántica permiten construir sistemas que no solo midan estas variables, sino que también sean capaces de interpretarlas y generar recomendaciones automáticas de cuidado.

En este proyecto se desarrolla SmartPlant, un sistema IoT semántico basado en un nodo sensor ESP32 equipado con un sensor DHT11 y un fotoresistor (LDR), un backend en Flask y un dashboard web moderno. El sistema captura datos ambientales cerca de una planta de interior, los envía a un servidor local mediante HTTP y MQTT, los estructura siguiendo la ontología SSN/SOSA y la SensorThings API, y presenta en una interfaz web lecturas en tiempo real, estados semánticos (bajo/ideal/alto) y recomendaciones de cuidado específicas para distintos tipos de planta.

Además, se implementan funciones de gestión de plantas (registro de múltiples plantas, selección de planta activa, histórico por planta) y un mecanismo de configuración dinámica que permite ajustar el intervalo de muestreo del ESP32 desde el frontend sin necesidad de reprogramar el firmware.

**Justificación**

El proyecto SmartPlant responde a varias necesidades:

* **Accesibilidad al monitoreo ambiental**: muchos usuarios tienen plantas de interior, pero no conocen sus requerimientos de luz, temperatura y humedad. Un sistema sencillo que indique si las condiciones son adecuadas y que sugiera acciones ayuda a reducir la mortalidad de plantas por riego excesivo, falta de luz, etc.
* **Aprendizaje práctico en IoT y Web Semántica**: el proyecto integra sensores físicos, comunicación WiFi, protocolos HTTP y MQTT, modelado semántico con SSN/SOSA y visualización web. Esto permite aplicar de forma concreta contenidos de IoT, arquitecturas de software y ontologías.
* **Extensibilidad**: aunque el prototipo se centra en una planta de interior, la arquitectura es extensible a invernaderos, jardines inteligentes o sistemas domésticos más complejos. La capa semántica facilita en un futuro interoperar con otros servicios, bases de conocimiento o asistentes inteligentes.
* **Bajo costo y reuso**: se usan componentes de bajo costo y software libre (ESP32, DHT11, LDR, Mosquitto, Flask, Chart.js), lo que permite replicar y extender el sistema fácilmente.

En resumen, SmartPlant no solo resuelve un problema concreto de monitoreo de una planta, sino que sirve como demostrador de la convergencia entre IoT, semántica web y visualización interactiva.

**Objetivo General**

Desarrollar un sistema IoT semántico llamado **SmartPlant** que permita monitorear en tiempo real las condiciones ambientales de una planta de interior (temperatura, humedad y nivel de luz), estructurar las observaciones usando la ontología SSN/SOSA y generar recomendaciones automáticas de cuidado mediante reglas semánticas, integrando un nodo sensor ESP32, un backend web y un dashboard interactivo.

**Objetivos Específicos**

1. Diseñar y construir un nodo sensor basado en ESP32 con sensores DHT11 y LDR, así como indicadores LED, para la captura y visualización local de estados ambientales.
2. Implementar un servicio web con Flask que reciba observaciones desde el ESP32 vía HTTP y MQTT, las almacene en formato JSON y RDF, y exponga endpoints REST para consulta.
3. Modelar las observaciones usando la ontología SSN/SOSA y la SensorThings API, generando un grafo RDF de lecturas, sensores, entidades observadas y fenómenos.
4. Desarrollar una interfaz web que visualice en tiempo real las lecturas de temperatura, humedad y luz, las clasifique en estados semánticos (bajo/ideal/alto) según el tipo de planta, y muestre recomendaciones de cuidado.
5. Implementar reglas simples de interpretación semántica para recomendar acciones (regar, mover a un lugar más iluminado, reducir riego, etc.) basadas en los umbrales de cada planta.
6. Permitir el registro y selección de múltiples plantas, manteniendo un histórico separado por planta activa.
7. Validar el sistema mediante pruebas locales con una o varias plantas de interior.

**Requerimientos de Usuario**

* **RF01**: El sistema debe permitir registrar una planta especificando nombre, ubicación en la casa, tipo de planta y frecuencia de muestreo.
* **RF02**: El usuario debe poder seleccionar, desde el dashboard, una planta previamente registrada y ver únicamente el histórico asociado a esa planta.
* **RF03**: El dashboard debe mostrar en tiempo real las lecturas actuales de temperatura, humedad y nivel de luz, así como un estado semántico (“Bajo”, “Ideal”, “Alto”) para cada variable.
* **RF04**: El sistema debe mostrar recomendaciones de cuidado (por ejemplo: “Riega menos”, “Mueve la planta a un lugar más iluminado”) según el tipo de planta configurado.
* **RF05**: La interfaz debe ser responsive, visualmente agradable y fácil de usar, con un flujo claro: configuración inicial, vista de inicio (estado), vista de planta (perfil) y vista de configuración.
* **RF06**: El usuario debe poder cambiar la frecuencia de muestreo desde el frontend y el ESP32 debe actualizar su intervalo sin reprogramación.
* **RF07**: El sistema debe mostrar si la ESP32 está conectada **o** desconectada con una indicación visual clara.
* **RF08**: El histórico y las gráficas deben mostrar únicamente las lecturas de la planta actualmente activa.

**Requerimientos de Hardware**

* **RH01**: Un **ESP32** como microcontrolador principal, con WiFi integrado.
* **RH02**: Un sensor **DHT11** para lecturas de temperatura y humedad ambiente.
* **RH03**: Un **fotoresistor (LDR)** más una resistencia fija para conformar un divisor de tensión medido por el ADC del ESP32.
* **RH04**: Tres **LEDs (verde, amarillo y rojo)** con sus resistencias limitadoras, conectados a pines digitales del ESP32 para indicar estados locales de la planta.
* **RH05**: Protoboard, cables Dupont y fuente de alimentación (USB) para el ESP32.
* **RH06**: Un PC o laptop en la misma red WiFi que la ESP32, para ejecutar el backend Flask, el broker MQTT Mosquitto y el frontend.
* **RH07**: Opcionalmente una maceta con una planta de interior (por ejemplo Pothos, Zamioculca, Sansevieria, etc.) para validación real.

**Requerimientos de Software**

* **RS01**: Sistema operativo de escritorio (Windows) en el PC para ejecutar Python, Mosquitto y servir el frontend.
* **RS02**: Python 3.11+ y dependencias del backend instaladas mediante requirements.txt.
* **RS03**: Framework Flaskpara implementar la API REST del backend.
* **RS04**: Librería paho-mqtt en el backend para suscripción a tópicos MQTT y procesamiento de mensajes del ESP32.
* **RS05**: Librería rdflib para la generación de triples RDF en formato Turtle y JSON-LD usando SSN/SOSA.
* **RS06**: Broker MQTT Mosquitto configurado para aceptar conexiones desde la red local.
* **RS07**: Entorno de desarrollo PlatformIO  para programar y cargar el firmware del ESP32.
* **RS08**: Librería PubSubClient en el firmware para comunicación MQTT, junto con las librerías WiFi.h, HTTPClient.h y DHT.h.
* **RS09**: Frontend en HTML, CSS y JavaScript, usando Chart.js para gráficas y fetch para consumir la API REST.
* **RS10**: Navegador web moderno (Chrome/Edge/Firefox) para visualizar el dashboard

**Arquitectura del Sistema**

La arquitectura de SmartPlant es un sistema distribuido con tres componentes principales:

1. **Nodo Sensor (ESP32)**

* Lee temperatura y humedad del DHT11 y nivel de luz del LDR.
* Calcula un nivel de luz relativo en porcentaje (0–100 %) a partir de la lectura analógica.
* Envía periódicamente las observaciones al backend vía HTTP (/api/observations) y opcionalmente vía MQTT (smartplant/observations).
* Consulta el endpoint /api/config para obtener la frecuencia de muestreo configurada en el frontend.
* Controla 3 LEDs (verde, amarillo, rojo) para mostrar estados locales basados en umbrales de temperatura, humedad y luz.

1. **Backend / Servidor Semántico (Flask)**

Expone endpoints REST como:

* /api/observations (POST): recibe lecturas del ESP32.
* /api/observations/latest (GET): devuelve la última observación por planta.
* /api/observations/rdf (GET): devuelve el grafo RDF en Turtle/JSON-LD.
* /api/config (GET/POST): guarda y devuelve parámetros de configuración (nombre de planta, ubicación, frecuencia, tipo de planta).
* /api/plants (GET): lista de tipos de plantas con sus rangos semánticos y recomendaciones.
* /api/plants/configs (GET/POST): gestión de configuraciones de plantas guardadas.
* /api/device (GET): metadatos descriptivos del nodo IoT (identificador, sensores, ubicaciones, capacidades).
* Se suscribe al tópico MQTT smartplant/observations y trata los mensajes de la misma forma que las peticiones HTTP.
* Almacena observaciones y configuraciones en archivos JSON y genera un archivo TTL con las observaciones semánticas.

1. **Interfaz Web (Frontend)**

Al cargar, muestra un modal obligatorio de configuración donde el usuario registra:

* Nombre de la planta.
* Ubicación (selector con lugares de la casa).
* Tipo de planta (Pothos, Zamioculca, Sansevieria, Spathiphyllum, Suculenta).
* Frecuencia de muestreo en segundos.
* Una vez guardada la configuración:
* Muestra en la vista de inicio las lecturas actuales, el estado de la planta y recomendaciones.
* Actualiza periódicamente las lecturas (polling) desde el backend.
* Permite ver un perfil de planta con rangos semánticos y consejos.
* Permite cambiar la planta activa y la frecuencia de muestreo desde la vista de configuración.
* Aplica estilos modernos (glassmorphism, tarjetas, grid responsivo) e indica el estado de conexión de la ESP32.

**Metadatos y Modelado Semántico**

SmartPlant representa las observaciones siguiendo la ontología SSN/SOSA y conceptos de la SensorThings API:

Cada observación se modela como instancia de sosa:Observation:

* sosa:observedProperty: temperatura, humedad, iluminancia.
* sosa:hasFeatureOfInterest: la planta específica.
* sosa:madeBySensor: el DHT11 o el LDR.
* sosa:resultTime: marca de tiempo de la lectura.
* sosa:hasSimpleResult: valor numérico (°C, %, % luz).

Se generan triples RDF en Turtle almacenados en observations.ttl.

El endpoint /api/observations/rdf permite descargar el grafo completo en TTL o JSON-LD, demostrando el uso de la ontología y facilitando posibles consultas SPARQL.

**Tecnologías y herramientas usadas en el desarrollo del aplicativo**

1. **Python**: Lenguaje principal para el desarrollo del backend, lógica semántica y generación de RDF.
2. **Flask**: Framework web en Python para la implementación de la API REST (/api/observations, /api/config, /api/plants, /api/device, etc.).
3. **Mosquitto**: Broker MQTT usado para la comunicación ligera entre el ESP32 y el backend mediante el tópico smartplant/observations.
4. **PlatformIO**: Entorno de desarrollo y construcción del firmware embebido para el ESP32 (lectura de sensores, envío HTTP/MQTT, control de LEDs).
5. **ESP32 + librerías Arduino**: Microcontrolador principal con WiFi; uso de WiFi.h, HTTPClient.h, PubSubClient, DHT.h y la API de ADC para el LDR.
6. **rdflib**: Librería Python para la creación y serialización de grafos RDF en formatos Turtle y JSON-LD usando la ontología SSN/SOSA.
7. **paho-mqtt**: Cliente MQTT en el backend para suscribirse al tópico de observaciones publicadas por el ESP32.
8. **HTML, CSS y JavaScript**: Tecnologías base del dashboard web, con diseño responsivo y componentes tipo tarjetas.
9. **Chart.js**: Librería JavaScript para la visualización gráfica del histórico de temperatura, humedad y nivel de luz.
10. **Fetch API y JSON**: Mecanismo de consumo de la API REST desde el frontend, intercambio de datos en formato JSON.
11. **Ontologías SSN/SOSA y conceptos de SensorThings API**: Estándares semánticos usados para modelar sensores, observaciones, plantas (features of interest) y propiedades observadas.

**Protocolo de comunicación**

Se implementaron dos protocolos de comunicación principales en el sistema SmartPlant:

1. **HTTP**:

Es el protocolo usado en el modelo cliente–servidor entre el **ESP32/frontend** y el **backend Flask**.

* El ESP32 envía lecturas al endpoint /api/observations y consulta /api/config mediante peticiones HTTP.
* El frontend consume endpoints como /api/observations/latest, /api/recommendations/latest, /api/plants y /api/device usando fetch.
* HTTP es útil por su universalidad, facilidad de depuración y porque permite exponer servicios REST claros para otros posibles clientes.

1. **MQTT**:

Se utiliza para la comunicación ligera y en tiempo (casi) real entre el **ESP32**, el **broker Mosquitto** y el **backend**.

* El ESP32 publica observaciones en el tópico smartplant/observations.
* El backend (paho-mqtt) se suscribe a ese tópico y procesa los mensajes igual que las peticiones HTTP.
* MQTT es muy eficiente en redes IoT porque minimiza el ancho de banda y la latencia, lo que lo hace ideal para lecturas periódicas de sensores y posible actuación futura (control remoto de riego, etc.).

**Trabajos Futuros**

* Reemplazar el LDR por un sensor de luz calibrado (por ejemplo **BH1750**) para obtener valores de lux físicos más precisos.
* Integrar un **triple store** o base de datos semántica (por ejemplo GraphDB o Fuseki) y definir consultas **SPARQL** para análisis más complejos sobre las observaciones.
* Añadir **actuación remota**, por ejemplo control de una bomba de riego o ventilador, gobernada por las mismas reglas semánticas que hoy solo generan recomendaciones.
* Exponer la información semántica mediante JSON-LD y vocabularios estándar adicionales (como schema.org) para mejorar aún más la interoperabilidad.
* Crear una versión móvil o PWA del dashboard para monitorear las plantas desde cualquier dispositivo dentro de la red local.
* Realizar pruebas con varios usuarios y diferentes tipos de plantas para ajustar mejor los rangos semánticos y recomendaciones por especie.