Instituto Tecnológico y de Estudios Superiores de Occidente – ITESO

Reporte final

Materia: *Internet of Things*

Maestro: Héctor Edmundo Ramírez Gómez

Fecha: 20 de abril de 2020

Autor: Arpio Fernández, León. IE702086

Índice

Introducción

Planteamiento del problema

Conforme aumenta la población, cada día es más importante el manejo correcto de los recursos a nuestro alrededor, ya que la demanda de estos aumenta de manera directa e indirecta, viéndose la agricultura cada vez más afectada por este incremento, ya que no solo debe de cumplir con el mercado, sino que es más costoso acceder a los recursos necesarios para mantener la producción. A esto se suma el impacto ambiental de la agricultura, debido al consumo constante de agua, suelo y energía que requiere para poder proveer alimentos a la población.

Por eso, se existe la problemática del uso ineficiente de los recursos en la agricultura, ya que muchas veces se desperdician algunos recursos como agua, sustratos, abonos, y no se aprovechan de manera correcta otros, como la luz solar. Esto genera una amplia variedad de problemas, que van desde generar un producto de mala calidad, hasta perder cosechas por causa de enfermedades y plagas. A esto, se suma el hecho de que se invirtiera mucho dinero, tiempo, mano de obra y materias primas en el plantío.

También, aunque un plantío no tenga problemas en la calidad del producto, o con enfermedades en las plantas, este puede ser víctima de un uso muy ineficiente de recursos, ya sea que estos se utilicen de más o de menos, lo que se ve reflejado en un impacto monetario, que tal vez no sea tan obvio, pero que, manejando estas materias de manera correcta, podría aumentar el margen de utilidad.

Objetivo

El objetivo de este proyecto es el de desarrollar un dispositivo capaz de monitorear las variables más importantes en un cultivo de lechuga, y que pueda transmitir la información a un servidor remoto, de manera que esta sea desplegada en tiempo real utilizando un tablero para dar un formato visual y de fácil entendimiento, para de esta manera ayudar a los agricultores a reducir los desperdicios tanto de productos como de recursos dedicados al cuidado de los cultivos.

Validaciones

Se realizaron entrevistas a personas con conocimiento sobre el cuidado de cultivos, especialmente de la lechuga, para conocer cuales son los principales riesgos y desperdicios, así como las variables más importantes a monitorear para reducir o prevenir estos problemas.

Entrevistas

Laura Bucio – Proyecto de hidroponía para asignatura *Internet of Things*

¿Cuáles son los riesgos más comunes en el cultivo de la lechuga?

Muy sensible a la falta de higiene, ya que se pueden generar muchas bacterias. Lo más recomendable es tener el área lo más limpio posible, para evitar que la lechuga se empiece a pudrir.

¿Qué recursos y/o materias primas se desperdician más en un cultivo?

Los solutos, pues a veces no hay una medición precisa para saber cuánto se necesita para la hidroponía.

¿Qué es lo que más valor te agregaría a tu negocio?

Tener un sistema autónomo que permita tener el monitoreo del cultivo a través de cualquier dispositivo. Conectividad y medición de variables.

Si pudieras tener un dispositivo para el monitoreo de cultivos:

¿Cuáles son las variables más importantes por monitorear en un cultivo de lechugas?

Conductividad, luz, temperatura, pH.

¿Con qué frecuencia te gustaría actualizar la información de las variables?

Aproximadamente cada media hora, dependiendo de las hectáreas de lechuga que se tenga.

¿A qué alertas te gustaría tener acceso?

Alertas de todas las mediciones, aunque dependerá de la lechuga a la que se quiera enfocar; romana, italiana, entre otras. Cada una tiene un tipo de pH, aunque hay un estándar para las lechugas en general, hay ligeras variaciones para cada una.

¿Te gustaría ver alguna otra característica en el dispositivo?

Que sea visual, ya que algunos agrónomos prefieren que se les dé el dato y otros prefieren que al final del día se les proporcione una gráfica con la información recolectada.

Álvaro – Verde tu vida

¿Cuáles son los riesgos más comunes en el cultivo de la lechuga?

Plagas, las más comunes son la mosca blanca y el pulgón amarillo. También la refrigeración y almacenamiento del producto.

¿Qué recursos y/o materias primas se desperdician más en un cultivo?

Comúnmente, el agua, aunque depende mucho del diseño del sistema, también, los fertilizantes, ya que se desperdician si estos no son asimilables o útiles para la planta, e incluso se puede ver afectado el rendimiento del cultivo

¿Qué es lo que más valor te agregaría a tu negocio?

Incluir un envase, ya que estos aumentan mucho el precio del producto, y es necesario pues al consumidor le da mucha confianza ver el producto empacado. Además, esto tiene un impacto ecológico negativo.

Si pudieras tener un dispositivo para el monitoreo de cultivos:

¿Cuáles son las variables más importantes por monitorear en un cultivo de lechugas?

La luminosidad, el control de la humedad te puede evitar problemas con hongos y con plagas, así como la temperatura. Otras variables importantes son la conductividad eléctrica y el nivel de oxígeno en el agua. Sería útil un sistema de detección de plagas, para que la detección sea en una fase temprana y no cuando el problema sea evidente y mucho más difícil de erradicar o controlar.

¿Con qué frecuencia te gustaría actualizar la información de las variables?

Sería suficiente cada hora, ya que con este tiempo es suficiente reaccionar ante cualquier cambio o fluctuación de las variables. Menos de una hora no sería necesario, incluso en algunas variables por un periodo más prolongado, tal vez 3 horas.

¿A qué alertas te gustaría tener acceso?

Sería interesante tener alertas de bajas y altas temperaturas, de humedad baja y alta, niveles de pH. Una alerta para detección temprana de presencia de plagas también sería atractiva.

Incluso sería interesante un monitoreo del crecimiento de la planta, por si la producción está de algún modo escalonada, saber que lote es el más próximo a cosecha, para tenerlas en el momento más oportuno.

Víctor Mojarro – Inncampo

¿Cuáles son los riesgos más comunes en el cultivo de la lechuga?

Primero, están los climáticos, que son los que tienen que ver con la temperatura, la radiación solar, la luminosidad, entre otros. También puede haber daños físicos a las plantas, como falta de cuidado en la manipulación de las plantas, mal manejo de herbicidas, sustratos, abono, etc. Y están los daños biológicos, que son las plagas, hongos, patógenos y bacterias indeseadas en el ambiente de cultivo.

Además de estos riesgos, también puede haber una mala selección de la lechuga, ya que estas se dan por temporadas, una mala planificación del terreno, logística o los recursos humanos.

¿Qué recursos y/o materias primas se desperdician más en un cultivo?

El principal es el dinero, ya que, como no hay un estándar de “los pasos a seguir”, no se utilizan correctamente el agua, los fertilizantes y químicos, pues no se usan las cantidades necesarias en el momento indicado.

Muchas veces el suelo no se aprovecha de manera correcta, ya que este no se estudia lo suficiente antes de determinar cuántas plantas se cultivarán por metro cuadrado.

¿Qué es lo que más valor te agregaría a tu negocio?

Tener datos tanto climatológicos como eficiencias administrativas. El análisis de datos en todo momento ayuda para mejorar el uso de los recursos destinados al cultivo, ya que solo se utilizan las cantidades necesarias, y se disminuye el desperdicio. Además, el uso incorrecto de

bienes como el agua, o los sustratos, pueden influir en la presencia de hongos o enfermedades en la planta de lechuga.

Si pudieras tener un dispositivo para el monitoreo de cultivos:

¿Cuáles son las variables más importantes por monitorear en un cultivo de lechugas?

Humedad en el ambiente y suelo, luminosidad, radiación UVA y UVB, velocidad del viento, temperatura ambiente y en el suelo, PH en el suelo y en el agua, conductividad eléctrica y salinidad en el suelo.

También sería interesante tener mediciones del sistema de riego, para tener un control del volumen de agua que está siendo utilizada para el riego.

¿Con qué frecuencia te gustaría actualizar la información de las variables?

Cada 30 minutos, aproximadamente, a lo largo del día y la noche.

Aproximadamente, ¿Cuantos dispositivos por unidad de área son necesarios?

Dependerá de la planta, y del terreno, pero se necesita al menos 1 dispositivo por cada 100 m2.

¿A qué alertas te gustaría tener acceso?

A mediciones altas y bajas de todas las variables leídas por el dispositivo.

¿Hay alguna relación entre los valores de las variables y la existencia de plagas?

Hay enfermedades, como la cenicilla, que se presenta cuando tienes alta humedad, temperatura y un pH desfavorable. También, el exceso de ácido nítrico (Compuesto químico usado en abonos) puede generar putrefacción en la planta.

Hay que tomar en cuenta que estas condiciones favorecen las enfermedades en las plantas, mas no son indicadores de que el cultivo presenta estos males.

¿Te gustaría ver alguna otra característica en el dispositivo?

Algo que no he visto en ningún dispositivo, y que sería muy útil su monitoreo, es el de la vida microbiana en el suelo, aunque esto es extremadamente difícil de lograr.

Consideraciones

La lechuga es muy sensible, y necesita mucho cuidado, desde la selección de la semilla, ya que las diferentes lechugas son para diferentes temporadas; hay de frío a frío, frío a calor, calor a calor y calor a frío.

La lechuga puede crecer en invernadero, marcotúnel, hidroponía o en campo abierto, y siempre es necesario hacer un análisis de suelo adecuado, ya que, por ejemplo, un suelo arenoso no tiene mucha retención de agua, lo que significa que se necesitará un riego constate, o utilizar sustratos para cambiar esta propiedad. Además, es importante considerar que la planta requiere de diferentes cantidades de recursos durante el día, por ejemplo, en la mañana, consume más agua y nutrientes, ya que está en proceso de fotosíntesis.

También, es importante diseñar el sistema, y el monitoreo que sea de fácil uso, pues la mayoría de los agricultores prefieren no invertir mucho tiempo en configurar una aplicación, o en instalar un dispositivo, además de que, si se presentan problemas en el sistema, rápidamente desecharán la solución.

Investigación en fuentes secundarias

Se realizó una investigación sobre el cuidado de la lechuga en fuentes secundarias para validar y extender la información obtenida de las entrevistas. Específicamente de las variables a monitorear, sus rangos óptimos, para poder determinar también las alarmas.

Temperatura

La lechuga necesita diferentes rangos de temperatura durante el día y la noche, además de que estos rangos cambian dependiendo de la etapa de crecimiento de la lechuga.

Durante la germinación la temperatura requerida por la lechuga oscila entre los 18°C y los 20C. En la fase de crecimiento la lechuga requiere de 14°C a 18°C durante el día y 5°C a 8°C durante la noche. Por último, en la fase de acogollado, se requieren 12°C durante el día y 3°C a 5°C durante la noche.

Humedad relativa

Durante toda la vida de la lechuga, se requiere una humedad relativa de 60% a 80%.

Suelo

El pH óptimo para el crecimiento de la lechuga se encuentra entre 6.7 a 7.4

Luz

La lechuga se desarrolla mejor entre 10 klx y 40 klx, que corresponde a disminuir la luz de un día soleado con una protección, como plástico lechoso o malla sombra.

Análisis de alternativas

Tecnología de comunicación

Se realizó una comparación de diferentes tecnologías de comunicación inalámbrica, de manera tal que se elija la que mejor se adecúe a la necesidad de tener un dispositivo con bajo consumo de energía, posiblemente un gran tamaño de los mensajes, seguridad en la red, amplia cobertura y bajo costo.

LoRa

Descripción

Es una tecnología de Red de Área Amplia de Bajo Consumo (LPWAN, por sus siglas en inglés), creada en 2015 para cubrir necesidades de Internet de las Cosas, con el objetivo de permitir al usuario controlar su propia red. La banda de operación está entre 433 MHz y 915 MHz dependiendo de la ubicación (En América es de 915 MHz).

Ventajas

- Alta tolerancia a interferencias y jammers.
- Bajo consumo de potencia.
- Largo alcance (10 – 20 km).
- Bidireccional.
- Más capacidad de configuración.
- El uso de la red no requiere costo adicional.

Desventajas

- Más complejidad de uso.

- No ofrece encriptación.
- Se necesita instalar un servidor central para comunicar datos a internet.

ZigBee

Descripción

Tecnología de Redes Inalámbricas de Área Personal (WPAN, por sus siglas en inglés) diseñada en 2004 para usos industriales, científicos y médicos, ya que esta ofrece espacio para una gran cantidad de dispositivos conectados al mismo tiempo asegurando la integridad de la red y los datos. La banda de operación es de 868 MHz para Europa y 915 para América, aunque a nivel mundial se puede utilizar la banda de 2.4 GHz.

Ventajas

- No se ve afectada por la existencia de otras redes dentro de la misma banda (Bluetooth, Wifi y otras).
- Permite alta densidad de operación.
- Permite la existencia de muchas redes y nodos ZigBee sin afectar su comunicación (Hasta 16,000 diferentes redes, cada una con 65,000 dispositivos).
- Autorrecuperación en caso de fallas dentro de la red.
- Encriptación AES128.
- Enfocada al bajo consumo.

Desventajas

- Corto rango de cobertura (10 – 75 m).
- Requiere un servidor central (Coordinador) y varios intermediarios (Ruteador) para el funcionamiento de la red.
- Diseñada para paquetes de información pequeños.

Sigfox

Descripción

LPWAN desarrollada en 2010 por la empresa homónima y diseñada para ultra bajo consumo manteniendo una amplia cobertura. Además, Sigfox ofrece una API sencilla de utilizar, y servicio en la nube para almacenamiento y manejo de datos, incluida con la subscripción. La banda de operación es de 868 MHz para Europa y 902 MHz para América.

Ventajas

- Diseñada para bajo consumo.
- Bajo costo por paquete.
- Alta tolerancia a interferencias y jammers.
- Sigfox se encarga de la instalación de la red.
- Ofrece servicio de un servidor para almacenamiento de datos y ejecución de callbacks.
- Ofrece encriptación para mensajes pequeños (Propia de Sigfox).

Desventajas

- Límite de paquetes y de tamaño de cada paquete.
- Unidireccional.
- Tecnología patentada por Sigfox, por lo que reduce las opciones del servicio.
- Al ser provista por otra empresa, no se tiene control sobre la red como si fuera desarrollada por el usuario.
- Al ser patentada, no permite un desarrollo más eficiente de múltiples compañías como en tecnologías libres.
- Requiere una subscripción por dispositivo.

Wifi

Descripción

Estándar de la IEEE que define una tecnología para interconexión inalámbrica de dispositivos entre sí, permitiendo, además, la conexión con internet, lo que facilita el acceso de dispositivos a diversos servidores en la nube. Trabaja sobre las bandas de 2.4 GHz o 5 GHz, en todo el mundo. Wifi permite tanto redes locales o públicas.

Ventajas

- Soporta encriptación, aunque no está configurada por defecto.
- Soporta diferentes protocolos de comunicación (MQTT, TCP, SSH, HTTP, etc.).
- Soporta varios niveles de encriptación.
- Puede utilizarse para comunicación tanto local como pública.
- Alta velocidad de comunicación (Hasta 54 Mbps).

Desventajas

- Fácil de interferir.
- Requiere de mediadores entre los dispositivos y un servidor central (AP/Routers).
- Corto alcance (100 m).
- No está enfocado al bajo consumo.
- Alto costo para una red con acceso a internet.

Red celular (2G/3G/4G)

Descripción

Medio de comunicación inalámbrico, utilizado para dispositivos móviles, que permite la conexión de ellos a internet, además de la intercomunicación entre ellos por medio de protocolos como SMS. El acceso a la web es provisto por diversos operadores. Las bandas de operación van desde 800 MHz hasta 2.1 GHz.

Ventajas

- Gran cobertura en todo el mundo.
- Costo ajustable dependiendo del uso.
- Encriptación por defecto de A5/1 y A5/2.
- Posibilidad de incluir más capas de encriptación como SSL y TLS.
- Variedad de proveedores de red.
- Fácil cambio entre 2G/3G/4G y futuras.
- Futuro cambio de 2G a NB, red celular diseñada para IoT (Bajo consumo, bajo costo y pago por uso).

Desventajas

- “Huecos” de cobertura en ciertas áreas geográficas.
- Alto costo por mensaje.
- Sensible a interferencias o *jammers*.
- No se tiene control sobre la red.
- No diseñada para bajo consumo.
- Actualización de la red no dependiente del usuario.
- Obliga a trabajar con un operador y al pago por el uso del servicio.

Elección de la tecnología

Tomando en cuenta las redes investigadas, se llegó a la conclusión de que la red celular es la mejor opción, ya que permite la mayor libertad para métodos tanto de encriptación y protocolos de comunicación, además de que no se requiere instalar mediadores entre el dispositivo y un servidor. El mayor inconveniente es que la red no está diseñada para el bajo consumo, pero dado que la frecuencia de muestreo es baja (Al menos un mensaje cada minuto), es posible llevar al dispositivo a modo de bajo consumo de energía para el ahorro de esta. Por último, esta tecnología permite una gran variación en el tamaño del mensaje, y seguridad mucho más robusta que otras redes.

Sensores

Se realizó una tabla comparativa de diferentes sensores de temperatura en el ambiente, humedad relativa en la tierra, luminosidad y conductividad eléctrica en la tierra, que son las variables más importantes por monitorear, de acorde a las entrevistas realizadas.

Nombre	Precio (USD)	Protocolo de comunicación	Voltaje de alimentación	Variables que mide	Rango	Precisión
Catnip I2C Soil Moisture Sensor	$ 13.00	I2C	3.3V - 5V	Temperatura en el ambiente Humedad relativa en la tierra	0°C - 85°C 10% - 100%	± 2% ± 2%
SEN0114	$ 6.23	Analógico	3.3V - 5V	Humedad relativa en la tierra	0% - 100%	No especificado
SHT11	$ 51.87	I2C	3V - 5V	Temperatura en el ambiente Humedad relativa en la tierra	-40°C - 123°C 0% - 100%	± 0.4°C ± 3% RH

LM35	$ 7.31	Analógico	4V - 30V	Temperatura en el ambiente	-55°C - 150°C	± 0.1°C
MAX6627	$ 8.57	SPI	3V - 5.5V	Temperatura en el ambiente	-55°C - 125°C	± 2.4°C
VEML7700	$ 6.44	I2C	2.5V - 3.6V	Luz ambiental	0.0072lx - 120000 lx	± 0.0036 lx
APDS9007	$ 2.07	Analógico	2V - 3.6V	Luz ambiental	3lx - 75000lx	± 10 lx
GA1AS202WP	$ 1.16	Analógico	2.3V - 3.2V	Luz ambiental	3lx - 55000lx	± 0.5 lx
Liyuan conductivity, temperature and humidity	$ 51.00	RS232 (UART)	5V - 30V	Temperatura en el ambiente Humedad relativa en la tierra Conductividad eléctrica de la tierra	0 µS/m - 200000 µS/m 0% - 100% -40°C - 80°C	± 3% ± 3% ± 3%
Arduino Conductivity Sensor	$ 85.00	Analógico	5V - 12V	Conductividad eléctrica de la tierra	0 µS/m - 5000 µS/m	± 0.2%
Isolated EC probe	$ 60.00	I2C	2V - 5V	Conductividad eléctrica de la tierra Sólidos disueltos Temperatura en el ambiente	0.5 mS/cm - 20 mS/m 250 PPM - 10000 PPM -55°C - 125°C	No especificado No especificado No especificado

Tabla 1. Tabla comparativa de los posibles sensores a utilizar. Referencias en capítulo Hojas de datos consultadas*.*

De la tabla 1, se decidió elegir Catnip I2C Soil Moisture Sensor, ya que cubre 2 variables (Temperatura en el ambiente, y humedad en la tierra), cubriendo el rango necesario para ser medido, pues la temperatura mínima que soporta la lechuga es de 3°C, y este sensor es capaz de leer hasta los 0°C sin problema alguno. La planta de lechuga requiere una humedad relativa en la tierra entre el 60% y el 80%, entrando también en el rango de lectura del sensor. Por último, aunque otros sensores tienen una mejor precisión, no es indispensable que esta sea muy pequeña, ya que no se trata de un sistema extremadamente sensible.

Para la luminosidad, se decidió elegir el sensor GA1AS202WP, ya que cubre el rango necesario, pues la lechuga necesita entre 10 klx y 40 klx, y el sensor es capaz de medir entre 3 lx y 50 klx. Además, ofrece el mejor precio, y tiene una muy buena precisión de 0.5 lx.

Por último, se decidió no elegir ningún sensor de conductividad eléctrica en la tierra, ya que, además de que ya se están cubriendo las variables más esenciales, el menor precio por unidad es de $51 USD, lo cual implica un aumento en el costo total que dificultaría la compra del dispositivo por muchos usuarios.

Plataforma de desarrollo

Se realizó una tabla comparativa para elegir la plataforma de desarrollo y su respectivo microcontrolador que se utilizará para resolver la problemática, para esto, se deberá tomar en cuenta los sensores a utilizar, es decir, si la plataforma soporta el protocolo de comunicación del sensor y la tecnología de comunicación utilizada.

Tarjeta de evaluación	Microcontrolador	Precio (USD)	Memoria	Frecuencia de operación	Periféricos disponibles	Lenguaje de programación	Notas
Arduino Uno	ATmega328P	$58.37	2 KB RAM 32 KB FLASH 1 KB EEPROM	16 MHz	GPIO (14) PWM (6) ADC (6) SPI (1) UART (1) I2C (1)	C ++	
Arduino MKR FOX 1200	Cortex M0+	$92.47	32 KB RAM 256 KB FLASH	48 MHz	GPIO (8) PWM (13) UART (1) SPI (1) I2C (1) ADC 8/10/12 bits (7) DAC 10 bits (1)	C ++	Integrado con módulo Sigfox y conector para antena. SOLO FUNCIONAL EN EUROPA
Arduino MKR GSM 1400	Cortex M0+	$92.47	32 KB RAM 256 KB FLASH	48 MHz	GPIO (8) PWM (13) UART (1) SPI (1) I2C (1) ADC 8/10/12 bits (7) DAC 10 bits (1)	C ++	Integrado con módulo 3G y conector para antena.
	MK64FN1M0VLL12 (Cortex M4)	$56.88	256 KB RAM	21 MHz - 120 MHz	GPIO (32) PWM (4)	Embedded C	

NXP FRDM-K64F			1 MB FLASH		UART (4) SPI (2) I2C (2) ADC 8/10/12/16 bits (24) DAC 12 bits (2)		

Tabla 2. Tabla comparativa de las posibles plataformas de desarrollo a utilizar. Referencias en capítulo: Hojas de datos consultadas.

Tomando como base la tabla 2, y en conjunto con la elección de la red a utilizar (Red celular), se decidió elegir la plataforma Arduino MKR GSM 1400. Esta tarjeta viene integrada con un módulo GSM 3G, y con un Cortex-M0 con los periféricos necesarios para comunicarse casi con cualquier sensor (ADC, UART, SPI, I2C, etc), además de tener una alta frecuencia de operación, que permite operaciones en tiempo real. Dado que la frecuencia de muestreo del sistema es lenta (Mínimo 15 minutos entre una muestra y otra), esta frecuencia de operación es más que suficiente para ejecutar los cálculos necesarios antes de enviar la información.

La programación de este módulo es por medio de C++, en el IDE (Integrated Developmente Environment) de Arduino. Al ser abierto tanto el hardware como el software, se tiene acceso a los esquemáticos, en caso de que se desee diseñar una PCB para el proyecto. También, se tiene acceso a las librerías para el manejo del módulo GSM, de manera que facilita el desarrollo de la solución.

Por último, una ventaja de esta plataforma es el hecho de que la documentación de este dispositivo es extensa, y muy utilizada, por lo que también se pueden encontrar foros en donde se presentan soluciones a diversos problemas que se pueden presentar durante su uso.

Servidor

Se realizó una comparación entre diferentes servidores para recibir la información enviada por el dispositivo, procesarla y almacenarla, así como redirigirla a la aplicación en la que se presentarán los datos.

Amazon Web Services (AWS)

Servidor en la nube de Amazon, que ofrece servicios como bases de datos, procesamiento de información, respuestas a eventos, y comunicación por MQTT y HTTP de manera segura. AWS

permite el manejo de los datos por medio de la función Lambda, la cual, ante un mensaje u otro tipo de evento, se ejecuta, a manera de *callback*, permitiendo manejar la información de la manera deseada. Además, el servicio de *Dynamo DB*, permite el almacenamiento de datos de manera rápida y segura. Este servicio es gratuito hasta 25 GB de almacenamiento, y un millón de solicitudes de Lambda al mes.

Microsoft Azure

Este proveedor de servicios en la nube ofrece beneficios similares a los mencionados en AWS, pues ofrece comunicación por MQTT y HTTP, también con posibilidad de brindar una conexión segura. También, Azure ofrece *Function* como procesador de eventos, permitiendo hasta 1 millón de solicitudes cada mes, sin embargo, AWS no provee un servicio de bases de datos gratuito, de manera que, o habría que pagar este servicio, o se debe de implementar en un servidor aparte, una base de datos controlada por el usuario.

Mosquitto

Mosquitto es un proyecto open-source que ofrece comunicación MQTT de manera segura y siempre gratis. A diferencia de las otras dos opciones revisadas anteriormente, esta solución debe de ser instalada en un servidor por el usuario, además de que no ofrece un servicio de base de datos. La ventaja del uso de este broker es el de tener el control total del servidor, sin tener que pagar en ningún momento por su uso, aunque, si en algún momento el hardware ya no es capaz de procesar las solicitudes, se tendrá que mudar a otro dispositivo, o aumentar de alguna manera su capacidad.

Después de revisar las opciones para el servidor, se decidió utilizar AWS, ya que ofrece un servicio de base de datos, comunicación por MQTT y procesamiento de los mensajes recibidos de manera gratis. Aunque el servidor de Amazon tiene un límite para su uso gratuito, este permitirá, a una frecuencia de envío de 15 minutos, aproximadamente 320 dispositivos. A diferencia de Azure o Mosquitto, el servicio de Amazon también ofrece almacenamiento de la información recibida, lo cual es benéfico pues ya no se requiere instalar un servidor para manejar la información, o pagar extra por el espacio de almacenamiento.

Costos de desarrollo

Elemento	Costo (USD)
Arduino MKR 1400	$ 92.47
Antena GSM	$ 5.71
Batería Li-Po	$ 14.95
I2C Soil Moisture Sensor	$ 13.00
GA1AS202	$ 5.14
AWS	$ -
Ubidots	$ -
SIM	$ 1.00
Saldo	$ 2.50
Protoboard	$ 4.00
Componentes extra (Alambre, resistencias, capacitores, LED)	$ 2.00
Envios	$ 5.00
Total	$ 145.77

Tabla 3. Costos de desarrollo.

Diseños y desarrollos técnicos

Arquitectura general

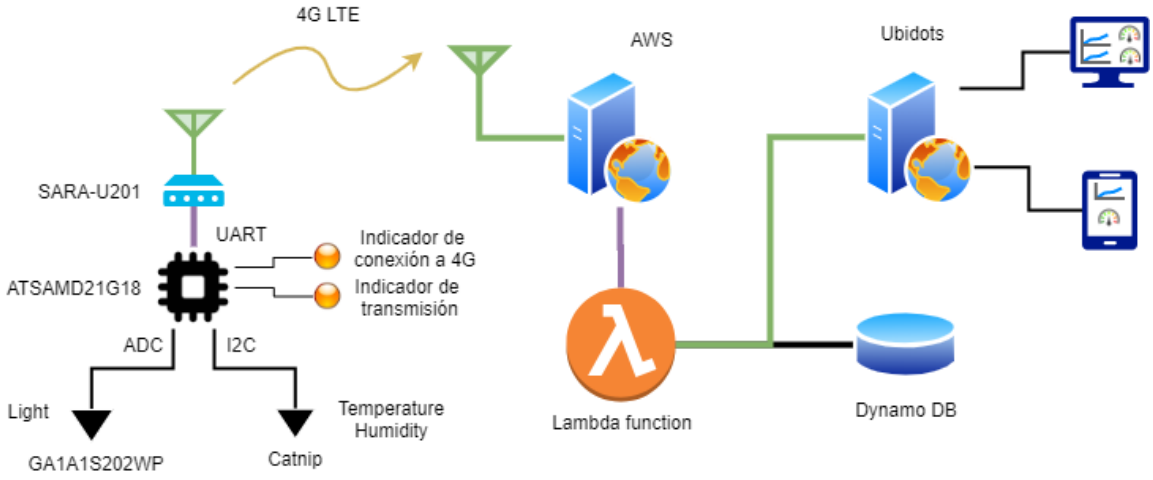

Figura 1. Arquitectura general del sistema.

La primera etapa del sistema consiste en la lectura de los sensores utilizando el microcontrolador ATSAMD21G18, donde este enviará la información encriptada al módulo de comunicación celular SARA-U201, el cual se encargará de transmitir la información recibida por medio del protocolo de Transporte de Colas de Mensajes por Telemetría (MQTT),

encriptada con Capa de Puertos Seguros (SSL) a la plataforma de Amazon Web Services (AWS).

Una vez recibido el mensaje por el servidor, la función Lambda desencriptará la información y la guardará en el servicio de bases de datos de Amazon llamado *Dynamo DB*, además, enviará la información a Ubidots, plataforma utilizada para presentar los datos, la cual puede ser accedida por medio de una computadora, tableta o un celular inteligente, en esta aplicación se presentará la información tanto en forma de gráfica, pudiendo el usuario elegir el periodo de tiempo que se mostrará en la gráfica. Además, Ubidots enviará una alerta en caso de que una variable esté en un modo indeseado durante cierto tiempo, permitiendo reaccionar rápidamente al evento sin necesidad de estar al pendiente de la información arrojada por la aplicación.

Diseño electrónico

Para el desarrollo con el Arduino, no fue necesaria ninguna etapa de adecuación, ya que el sensor de luminosidad ya viene integrado con una resistencia, necesaria para la interpretación del sensor, y un capacitor para filtrar ruido, de manera que solo se necesita conectar a voltaje y tierra para que la salida de un valor interpretable. La salida de este sensor está definida de la siguiente manera.

$$I_o = 10\log(E_v)$$

$$Donde:$$

$$E_v: Luminosidad\ en\ Lux.$$

$$I_o: Corriente\ de\ salida\ del\ sensor.$$

Dado que la resistencia que viene integrada con el transductor es de 68 kΩ, el cálculo de la corriente con respecto a la iluminación está dado de la siguiente manera.

$$I_{o[\mu A]} = \frac{V_o}{R} = 10(1\mu A)\log(E_v)$$

$$Donde:$$

$$R: Resistencia\ del\ sensor\ (68\ k\Omega).$$

$$V_o = Voltaje\ de\ salida\ del\ sensor.$$

Despejando para E_v.

$$\log(E_v) = \frac{V_o}{(10)(R)(1\mu A)}$$

$$E_v = 10^{\frac{V_o}{(10)(R)(1\mu A)}} = 10^{\frac{V_o}{(10)(68x10^3)(1x10^{-6})}}$$

De igual manera, el sensor de humedad y temperatura de I2C no necesita nada extra de hardware para se usado con el Arduino MKR 1400, ya que este cuenta con resistencias integradas de *pull-up* necesarias para el funcionamiento de I2C. Este sensor devuelve la temperatura en decimales de °C y la humedad en decimales de % RH, por lo que únicamente hay que dividir el valor obtenido de cada variable entre 10 para obtener su valor en unidades.

Se agregaron 3 LED indicadores para conocer el estatus del microcontrolador sin necesidad de conectarse al puerto serial, estos LED representan la conexión a red 4G, conexión al servidor MQTT y envío de mensaje exitoso; este último se encenderá durante 2 segundos, cada vez que se publique exitosamente a AWS. Para esto, se utilizaron resistencias de 1kΩ para proveer la corriente necesaria al LED.

Nota: Nunca se deberá de utilizar el dispositivo sin antena de red celular, ya que este se puede dañar parcial o totalmente.

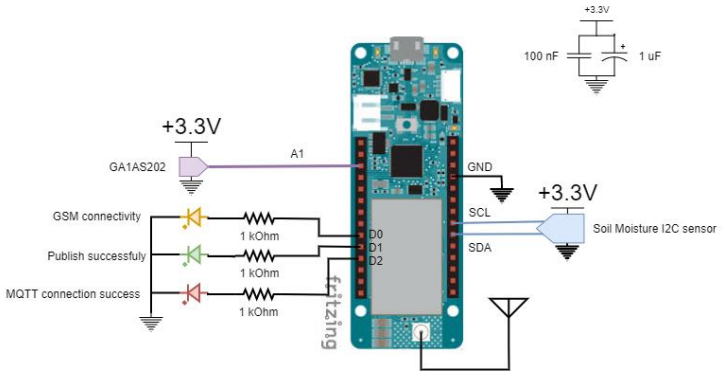

Figura 2. Esquemático del sistema con Arduino GSM 1400.

El mensaje enviado por el dispositivo a AWS es una cadena de caracteres, en formato JSON, con el siguiente formato:

```
{
        "time":timestamp,
        "IMEI":"imei",
        "temp":temperature,
        "hum":humidity,
        "lum":luminosity
}
```

Donde:

- Timpestamp: Cantidad de segundos que han pasado desde el 1 de enero de 1970 (Tiempo Unix).
- IMEI (Identidad Internacional de Equipo Móvil, por sus siglas en inglés): Identificador único del dispositivo, compuesto por una cadena de 16 dígitos.
- Temperatura: Temperatura en decimas de °C.
- Humidity: Humedad relativa en decimas de %.
- Luminosity: Luminosidad en lx.

Todos los valores enviados deberán estar en formato ASCII.

Seguidamente se procedió a diseñar el esquemático, en *Eagle*, para el diseño de la PCB, tomando como referencia el del Arduino MKR GSM 1400, al cual se eliminaron los elementos que no son requeridos para el producto final, como los *headers*, y hardware extra, y se agregaron los *pads* para soldar los sensores, así como los LED y sus respectivas resistencias. Para poder acceder a todos los componentes, tanto la figura del esquemático, como su *footprint* para la PCB, ir a *File>Import>EAGLE drawing* y seleccionar el archivo .sch deseado. Al ir al botón *Generate/switch to board*, aparecerán todos los *footprint* correspondientes del esquemático.

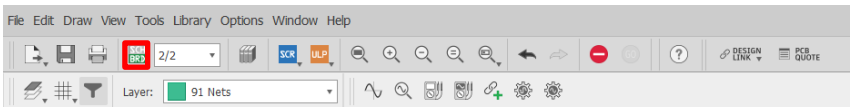

Figura 3. Botón para generar o cambiar al diseño de la PCB.

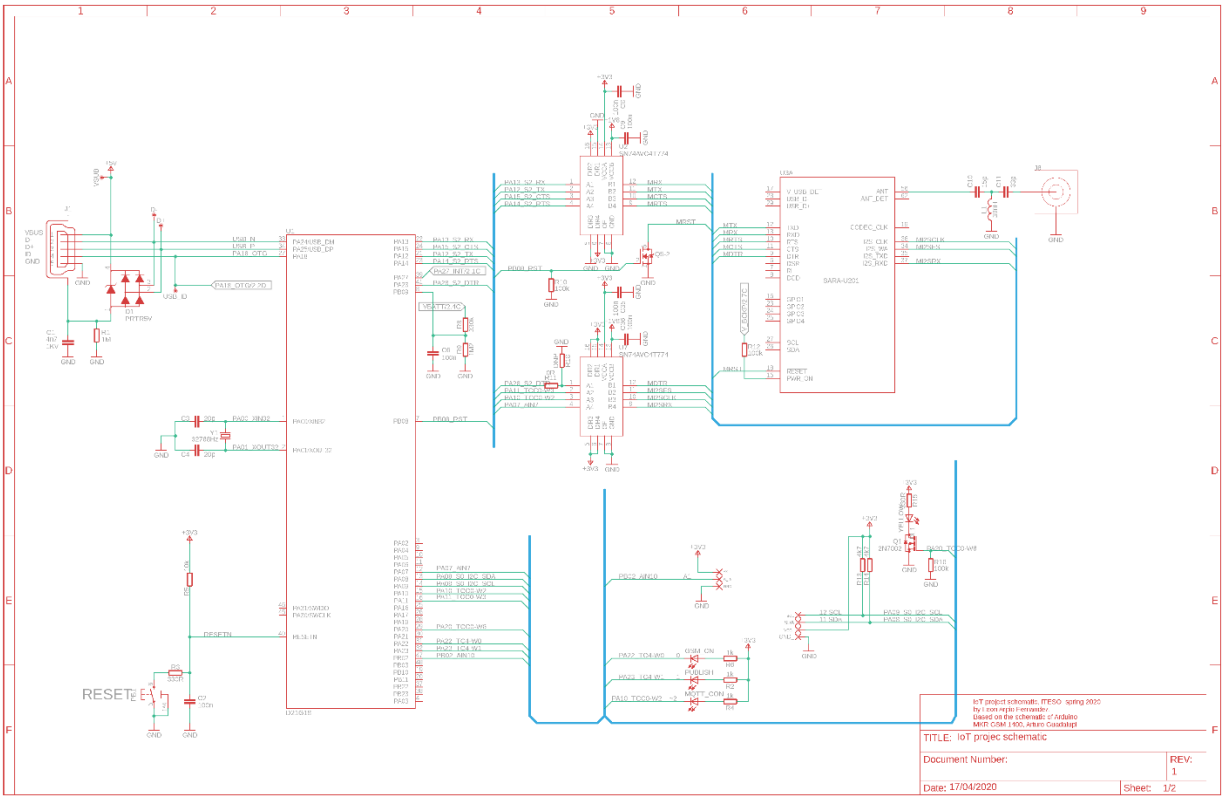

Figura 4. Esquemático del dispositivo, página 1 de 2.

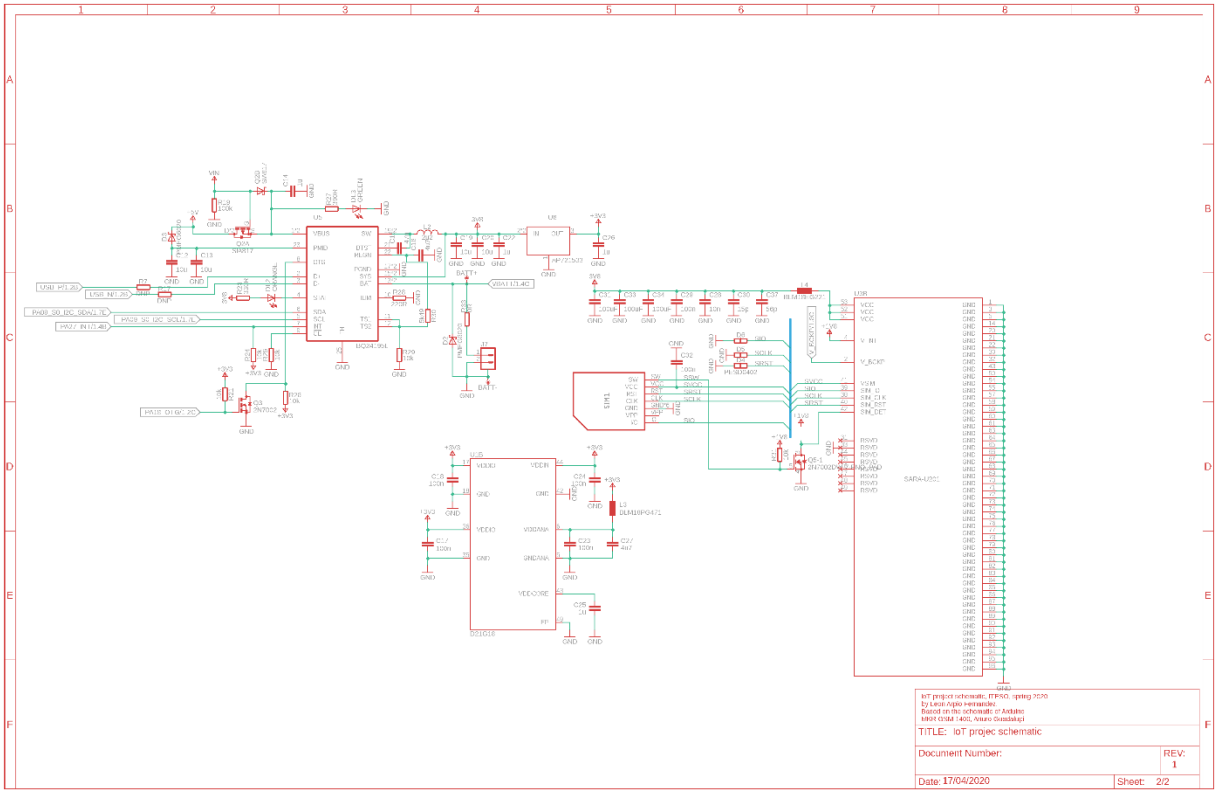

Figura 5. Esquemático del dispositivo, página 2 de 2.

Después de diseñar el esquemático, se procedió al diseño de la PCB, utilizando también *Eagle.*

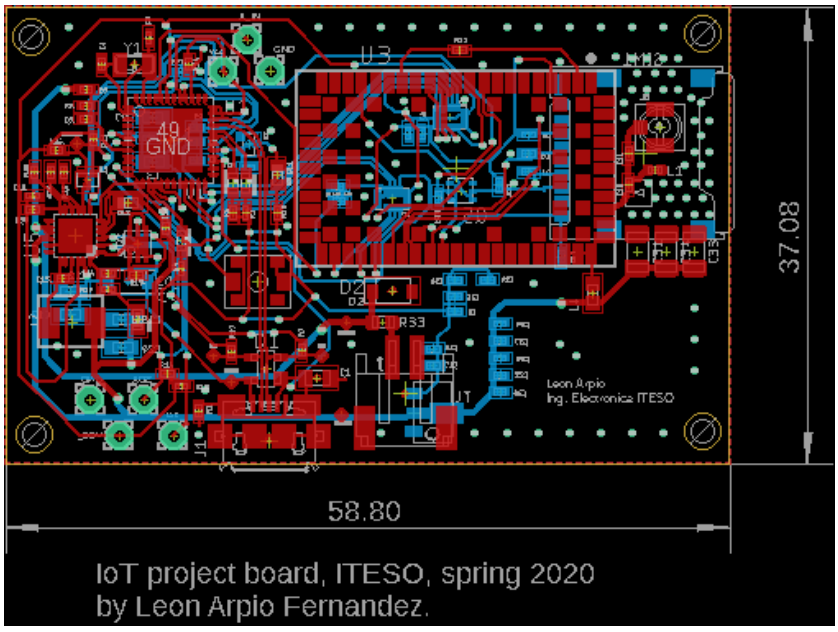

Figura 6. Diseño de la PCB.

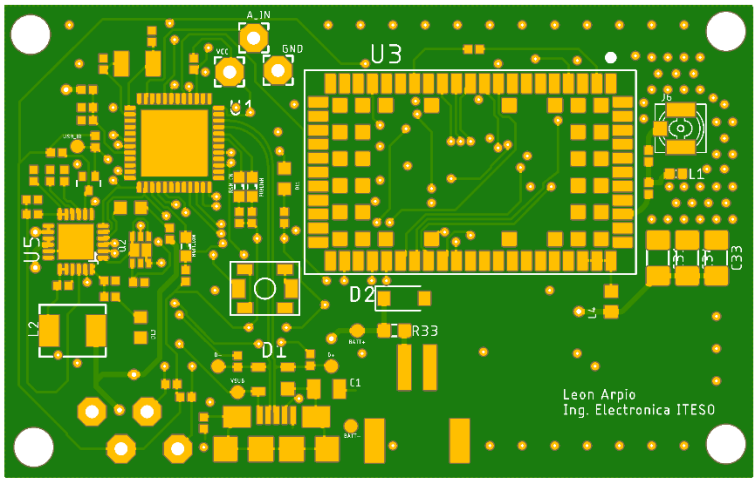

Figura 7. Vista de la PCB final (Top).

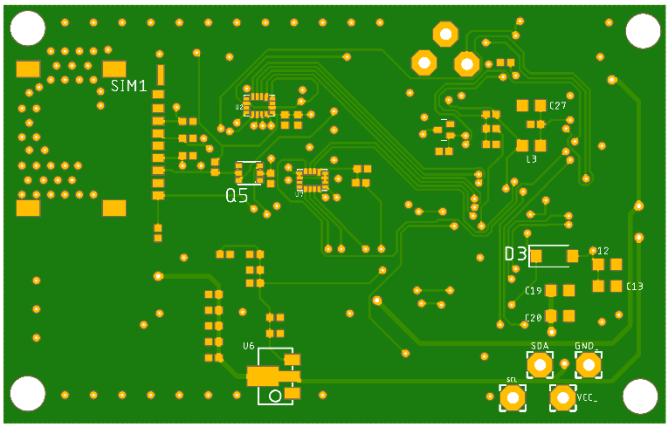

Figura 8. Vista final de la PCB (Bottom).

Diseño informático

Para el desarrollo informático, primero se procedió a configurar AWS de manera que el dispositivo se pueda conectar, publicar y recibir mensajes por medio del protocolo de Transporte de Colas de Mensajes por Telemetría (MQTT, por sus siglas en inglés). Una vez configurados los dispositivos que podrán publicar o recibir información de un tema por MQTT (Ver el tutorial de AWS, en el repositorio de este proyecto), se procedió a programar la función Lambda que se encargará de recibir y procesar los mensajes para mandarlos al servicio de bases de datos *Dynamo DB*, así como a la aplicación Ubidots, publicando la información por medio HTTPS.

Dado los datos por almacenar no requieren de una base de datos compleja, además de que el servicio *DynamoDB* provee de una base de datos NoSQL, solamente se requiere que el dispositivo envíe su IMEI, el cual será la clave principal, y el *timestamp*, el cual será la clave de ordenación, dando la flexibilidad de que se utilicen dispositivos que midan diferentes variables y envíen la información recolectada al mismo servidor. En otras palabras, cualquier otro campo puede ser enviado y será almacenado sin mayor problema en la base de datos.

La aplicación Ubidots, recibirá la información de cada dispositivo, reenviado a este por la función Lambda, y la graficará permitiendo un análisis rápido y sencillo de este. Además, esta plataforma podrá enviar una alarma, ya sea un correo, mensaje SMS u otro, en caso de que una alarma tenga un valor mayor o menor a cierto umbral. (Ver tutorial de Ubidots en el repositorio de este proyecto).

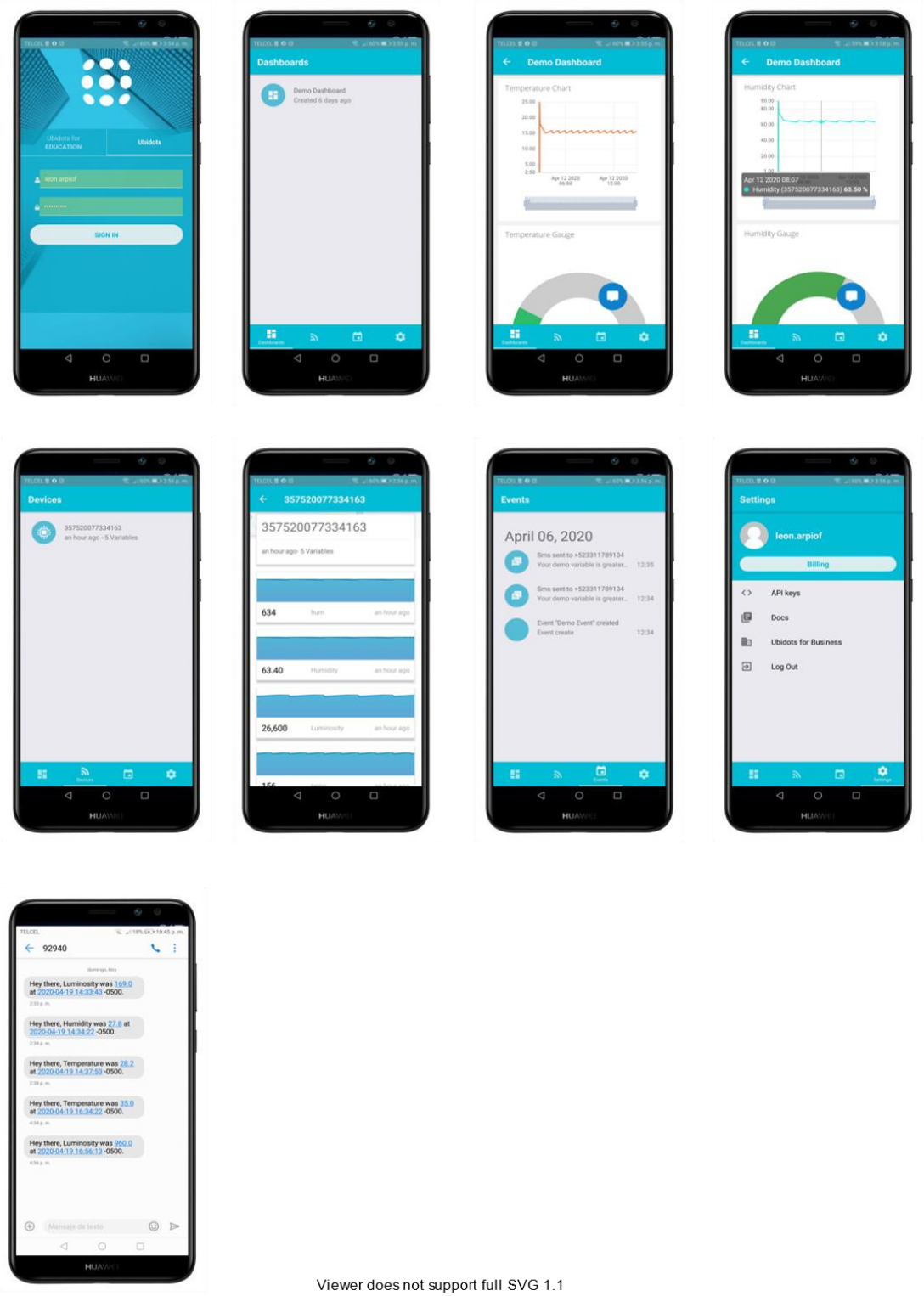

Figura 9. Interfaz de usuario en aplicación móvil de Ubidots.

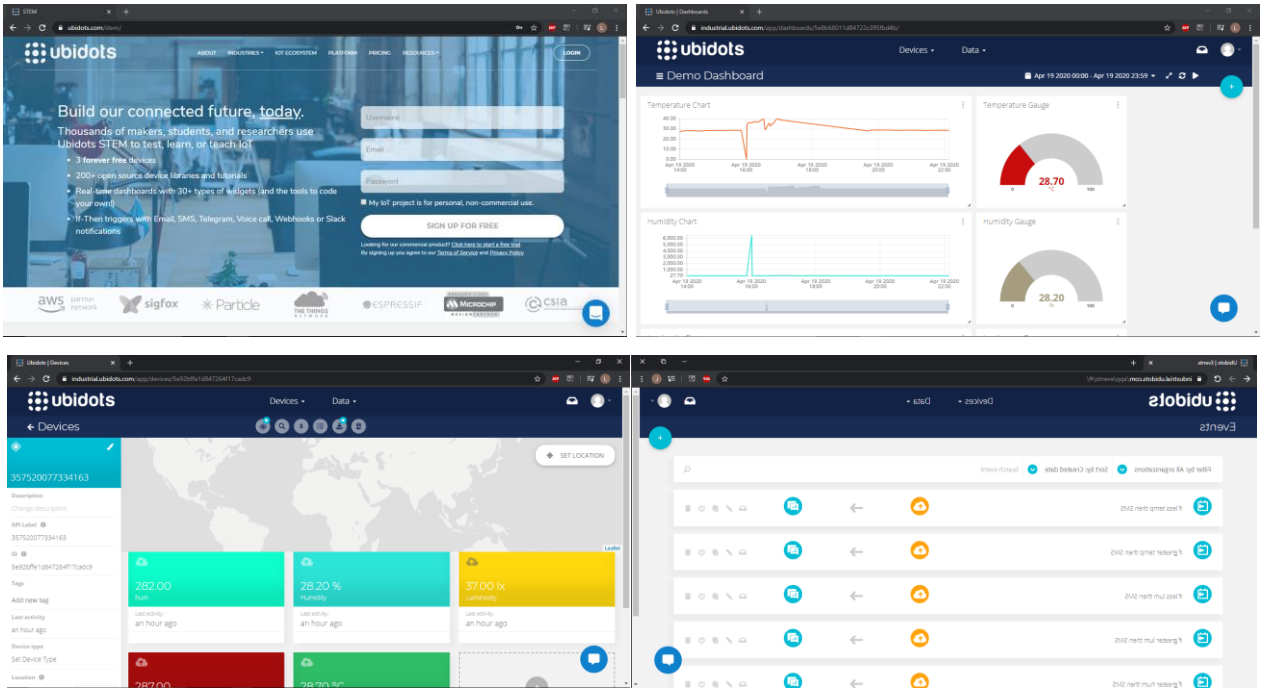

Figura 10. Interfaz de usuario en aplicación web de Ubidots.

Análisis de riesgos y seguridad de la información

Tras analizar la arquitectura del sistema, se lograron identificar los siguientes riesgos:

Comunicación entre microcontrolador y módulo celular

Dado que el microcontrolador y el módulo celular se encuentran conectados por UART, fácilmente alguien podría conectarse a estas líneas y recibir la información que se enviará por MQTT, ya que el dispositivo enviará la información recolectada para que esta pueda ser transmitida a internet.

Comunicación del dispositivo a AWS

Al enviar los datos por comunicación celular, la información se vuelve accesible para quien tenga una antena de 2.4 GHz, ya que, al ser comunicación inalámbrica, cualquiera puede escuchar los datos transmitidos. Además, esta información llega a internet, donde es susceptible a ataques.

Comunicación de AWS a Ubidots

Al igual que en el punto anterior, la información enviada a la aplicación en la que se desplegará la información es vulnerable, si esta no tiene la seguridad suficiente, aunque es menos propensa a ataques porque se ve protegida por la seguridad de los servidores que la emiten.

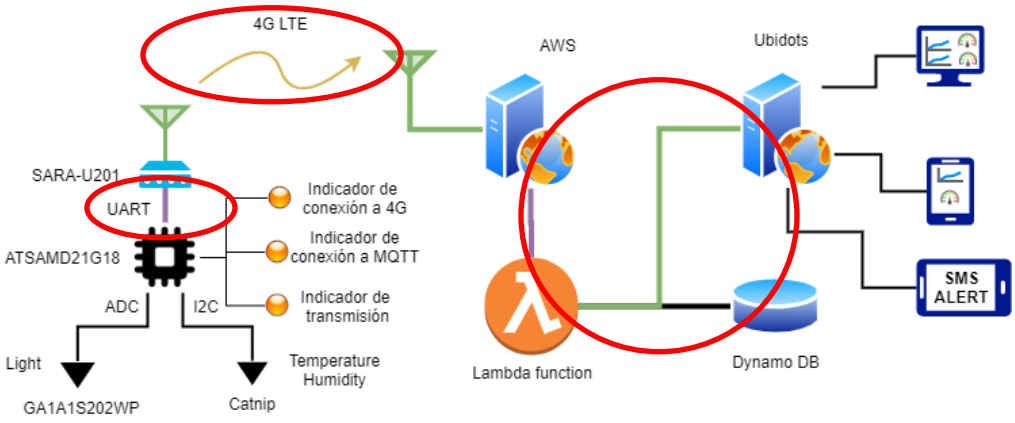

Figura 11. Riesgos de seguridad en la arquitectura.

Para resolver el primer riesgo, solamente se requiere una encriptación sencilla de los datos sensibles, por ejemplo, cifrado por XOR. Aunque este método es fácil de descifrar, para lograrlo se tiene que estar físicamente con el dispositivo, además de que, para encontrar la contraseña, se debe de alimentar con datos conocidos al sistema, lo cual se dificulta debido a que la manera de que el sistema recibe los datos es por medio de ADC o de I2C. Es importante hacer notar que sólo se puede cifrar la información medida, es decir, dado que los datos se envían en formato JSON, sólo podemos cifrar el valor de algunos datos, pues AWS solamente recibe mensajes en dicho formato. Además, una encriptación más compleja pudiera afectar el rendimiento del dispositivo.

La manera más efectiva de resolver el segundo riesgo es encriptando la conexión por medio de TLS, ya que con este protocolo se vuelve prácticamente imposible descifrar la información sin los certificados y llaves con los que fueron encriptados. Además, la red celular ya viene con encriptación A1/5 por defecto, aunque esta capa no es lo suficientemente segura. De igual manera, el tercer riesgo se debe de resolver con TLS.

Código desarrollado

Descripción de la arquitectura del software y firmware

Códigos en capítulo Anexos.

Arduino

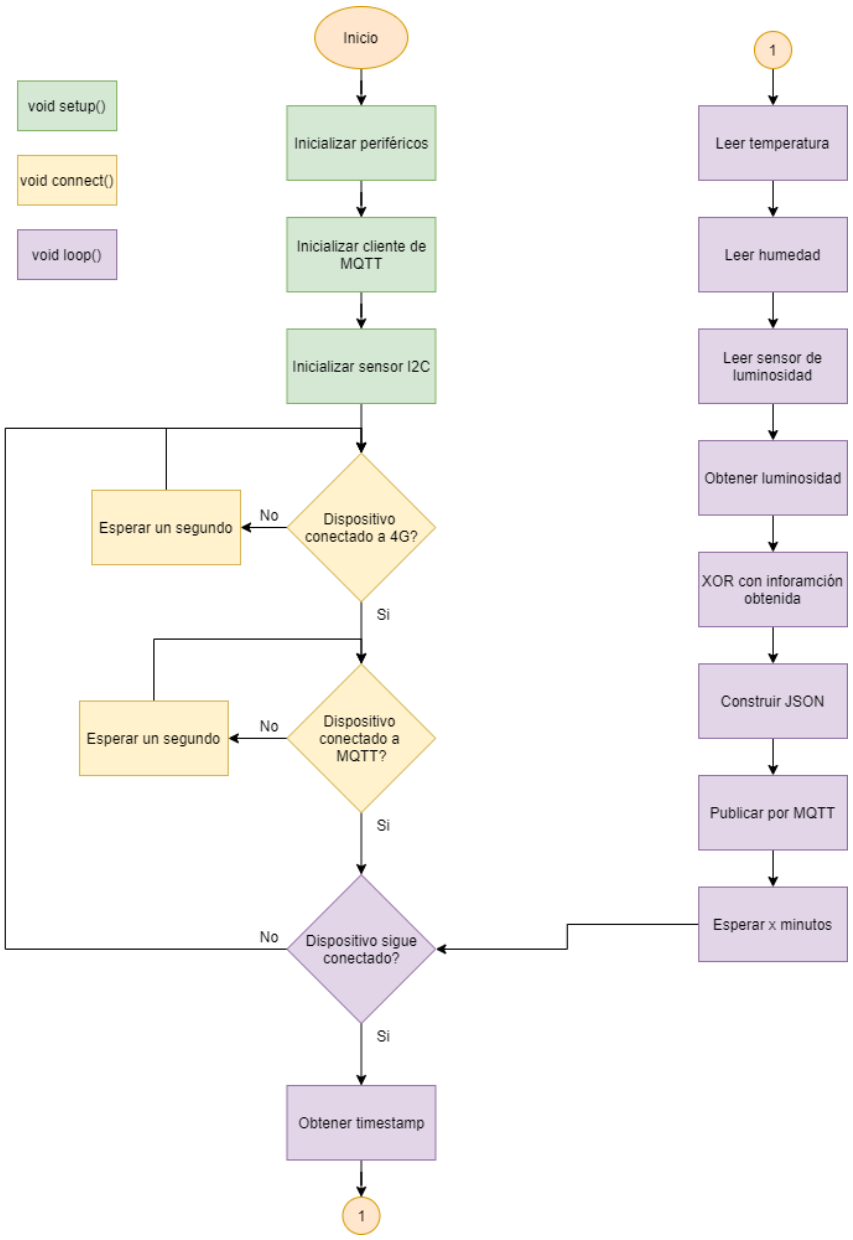

Figura 12. Diagrama de flujo del código de Arduino.

En la figura 12 se puede observar el proceso que lleva a cabo el Arduino, el cual inicializa el módulo celular, y el sensor de I2C (El sensor de luminosidad no requiere de inicialización), y después intenta conectarse a la red celular, y después al servidor MQTT, una vez que se logra esto, se obtiene la información de los sensores de humedad, temperatura, luminosidad, y también obtiene el *timestamp* del módulo celular, construye el mensaje en formato JSON, para lo cual, convierte todas las variables a *string*, y finalmente lo publica por MQTT al servidor de AWS. Por último, después de esperar x tiempo, definido por el usuario, revisa la que la

conexión con el servidor MQTT, o con la red GSM, no se haya perdido; en caso de falla en la conexión, repite el proceso inicial, de lo contrario, continúa leyendo los sensores.

En el proceso de desarrollo del código de Arduino, se descubrió que, debido a un bug del módulo celular, este no acepta certificados, necesarios para la comunicación por SSL, y por lo tanto con AWS. Para resolver este problema se configuró el Arduino para que este publicara, por MQTT, a un servidor de pruebas público llamado Mosquitto, el cual no requiere de conexión segura para recibir la información. Por otro lado, se desarrolló un código en Python el cual se conecta a Mosquitto y a AWS, y envía la información recibida de el primer servidor al segundo, utilizando TLS v1.2.

Lambda

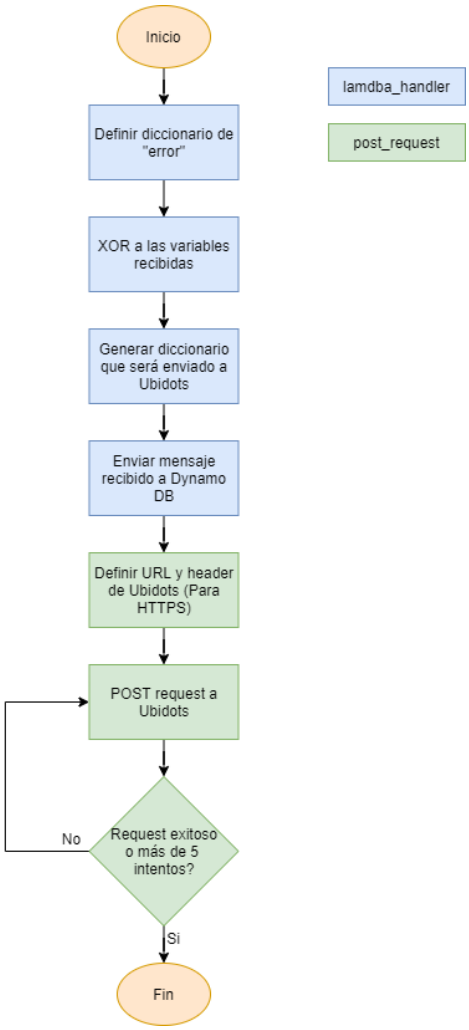

Figura 13. Diagrama de flujo de la función Lambda.

En la figura 13, se muestra el proceso de la función Lambda al recibir un mensaje por MQTT. Este mensaje se recibe como un diccionario de Python, aunque éste se enviado como una cadena de caracteres, y se envía a *Dynamo DB*. También, se extrae la información necesaria para Ubidots, y se envía por medio de HTTPS. Originalmente, esta transmisión iba a ser por medio de MQTT, utilizando la librería de *paho-mqtt*, pero no funcionaba correctamente, sin razón alguna, y la información no siempre la recibía Ubidots, por lo que se decidió utilizar el protocolo HTTP.

Repositorio en GitHub

https://github.com/Leonarpiof/IoT-ITESO-Prim2020

Modelo de negocios

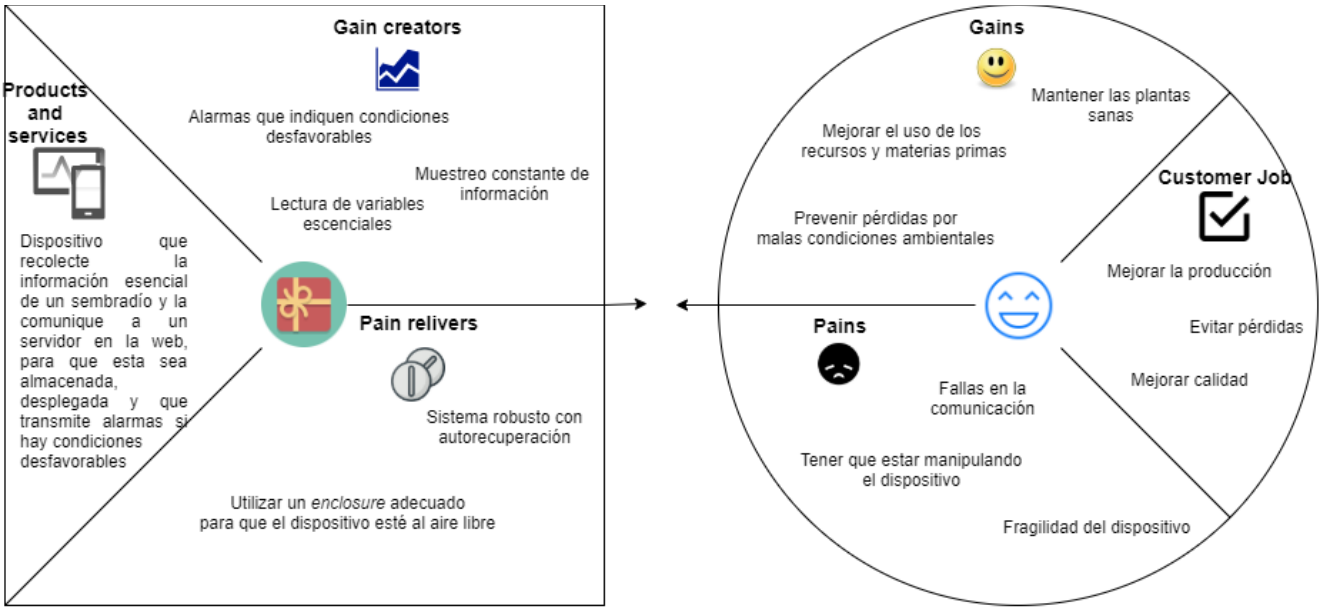

Figura 14. Canvas *de propuesta de valor.*

<table>
<tr>
<td rowspan="2">Problemas

Falta de dispositivos que permitan la comunicación de información de los sembradios.

Alto desperdicio en los sembradíos de recursos y materias primas.

Dificultad de acceso a información sobre las plantas que permita la acción en caso de que haya condiciones desfavorables.</td>
<td>Solución
Dispositivo que recolecte la información esencial de un sembradío y la comunique a un servidor en la web, para que esta sea almacenada, desplegada y que transmite alarmas si hay condiciones desfavorables</td>
<td rowspan="2">Propuesta de valor
Sencillez de uso del dispositivo, diseñado para ser plug and play de manera que el cliente solamente encienda el dispositivo y este se comunique automáticamente con el servidor y sea capaz de recuperarse en caso de eventos inesperados.</td>
<td>Ventaja especial
Sencillez de uso de la aplicación y accesibilidad para cambios de presentación de los datos y alarmas.</td>
<td rowspan="2">Segmento de clientes
Agricultores con necesidad de optimizar el uso de los recursos dedicados a su cultivo y de mejorar la calidad de sus productos.</td>
</tr>
<tr>
<td>Métricas clave
Venta de dispositivos
Reducción de costos del cliente
Solicitudes de lectura de nuevas variables</td>
<td>Canales
Correo
Celular</td>
</tr>
<tr>
<td colspan="2">Costos
• Componentes electrónicos
• Producción de la PCB
• Desarrollo de software y hardware
• Uso de la aplicación
• Tarjeta SIM y saldo para la comunicación celular</td>
<td colspan="3">Ingresos
• Compra del producto final
• Mantenimiento del dispositivo y aplicación
• Pruebas de concepto con diferentes sensores
• Cambios en hardware y software para nuevas soluciones.</td>
</tr>
</table>

Figura 15. Lean canvas.

PERSONA **FERNANDO (FER)** PROJECT **IOT PROJECT** EXPORT DATE **09.04.2020** *Created with Smaply Trial Version* smaply

Fernando

SHORT NAME
FER

IMAGE

OCCUPATION
Agricultor

NATIONALITY
Mexicano

MARITAL STATUS
Casado

AGE
31

GENDER
Masculino

CULTIVO(S)
Lechuga (Hibernadero)

DESCRIPTION
Fernando es un agricultor que quiere mejorar la calidad de sus productos, y quiere optimizar la producción de sus cultivos, no tiene el mayor conocimiento sobre tecnología, pero quiere una solución que le ayude a reducir sus gastos, disminuir sus pérdidas y aumentar la calidad. Esto le ayudará a mejorar la calidad de vida de su familia y trabajadores, brindando mejores oportunidades a futuro, además, quiere crecer cada vez más su negocio.

MOTIVACIONES
- Familia
- Trabajadores
- Negocio

FRUSTRACIONES
- Macroempresas (Competencia)
- Incertidumbre económica
- Falta de oportunidades educativas

Figura 16. Arquetipo de persona.

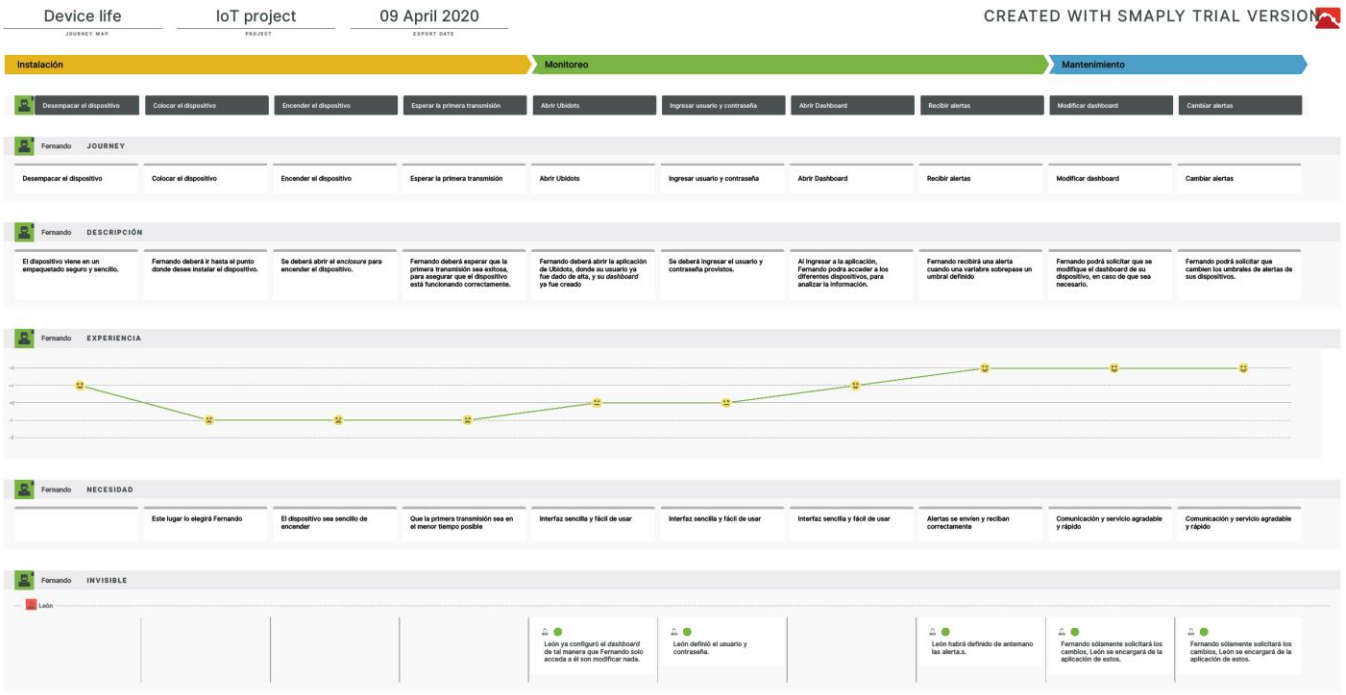

Figura 17. Customer Journey (Vida del dispositivo).

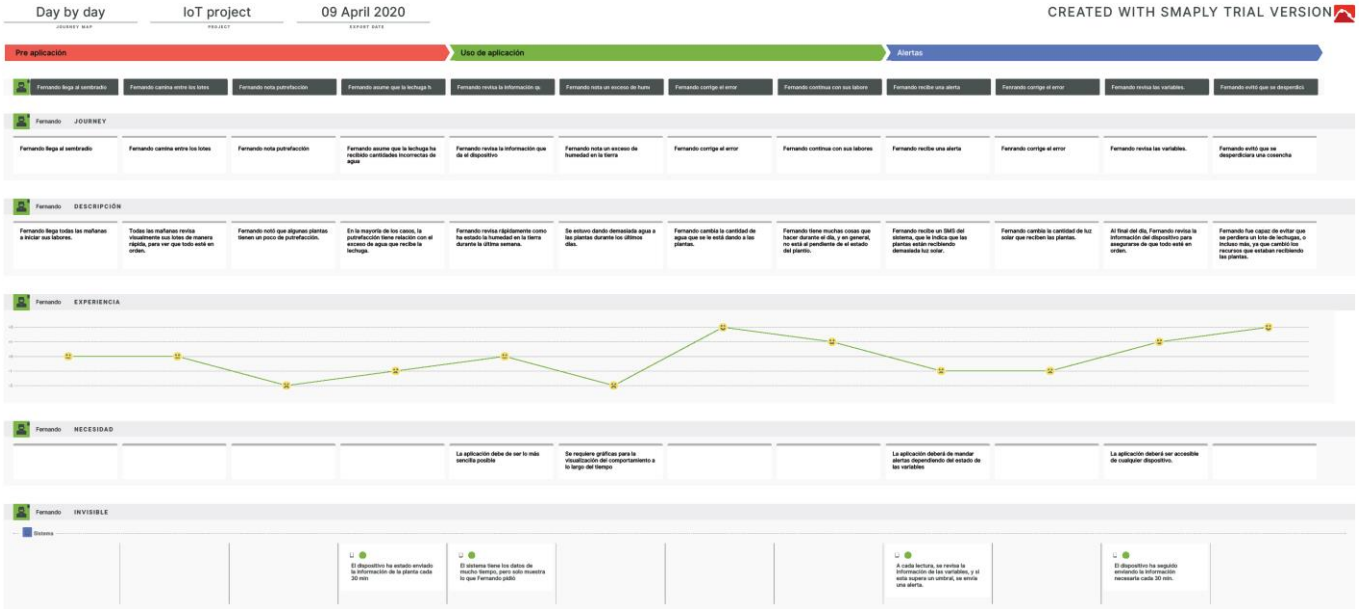

Figura 18. Customer Journey (Día a día).

Next steps

Para lograr concluir este proyecto, lo primero es revisar una alternativa para el módulo celular, especialmente otras alternativas del mismo fabricante, y la misma línea, para verificar si es que existe otro módulo que utilice el mismo *instruction set* de comandos AT, y que sea *pin-to-pin compatible*, de manera que el dispositivo se pueda comunicar directamente con AWS, es decir, utilizando SSL/TLS. En caso de que no exista un módulo que cumpla estos requerimientos, se deberá buscar otro módulo de comunicación celular que permita lograr una conexión por MQTT segura.

Otro paso importante que debe de ejecutarse es el de la producción de un prototipo de la PCB, y la prueba de este, para asegurarse de que esta fue diseñada correctamente, y en caso de que haya algún error, corregirlo, para poder producir el dispositivo en masa, ya con una PCB funcional. De igual manera, aunque este paso se deberá adecuar a cada cliente, encontrar un contenedor para el dispositivo que permita que este pueda ser instalado al aire libre sin el riesgo a que el agua, el sol u otras variables afecten al dispositivo.

Un posible paso, que dependerá de que tanto se quiera escalar el proyecto, sería el de generar librerías específicas para el control del dispositivo, de manera que se optimice el funcionamiento del microcontrolador, y que se exploten todos los recursos que este contiene, como *watchdog*, RTC, unidad de bajo consumo, y otras, pudiendo así generar una solución mucho más robusta.

Conclusiones

De este proyecto se concluye la importancia de todo lo que se tiene que hacer antes y después de desarrollar un dispositivo o un servicio, es decir, la parte innovadora de un producto no es solo el dispositivo, o el desarrollo tecnológico que hay detrás de este, si no la toma correcta de decisiones para lograr algo que se vaya a vender, esto implica hacer un análisis asertivo de las personas a quienes vaya dirigido el producto o servicio, y hacer todo el desarrollo siguiendo este análisis, pues si no, es altamente probable que no se venda, y que se desperdicie todo el tiempo, dinero y esfuerzo dedicados.

También, siempre es importante tener en mente las dificultades que se pueden presentar, y estar preparados para pivotear o incluso cambiar radicalmente uno o varios elementos durante el desarrollo, como ejemplo, es posible que, para generar un producto que se pueda vender,

se tenga que volver a desarrollar casi toda la parte electrónica por un *bug* del módulo celular, e incluso es posible que sea conveniente cambiar de tecnología de comunicación, por causa de un problema que difícilmente se iba a conocer de antemano, pues la manera de descubrirlo fue haciendo pruebas, y revisando muy a fondo los foros, pues el fabricante no anuncia este problema.

Estos cambios, ante los cuales se tiene que estar abierto, no solo pueden venir por problemas en el desarrollo, sino también por necesidades del mercado, es decir, puede ser que algún mercado más grande requiera de la medición de otras variables, aunque esto impacte en el costo del dispositivo, o tal vez se descubre que también las personas con interés de tener cultivos en casa son un mercado más amplio que el de agricultores. Todas estas posibilidades de cambio siempre están presentes en todos los proyectos, por lo que es importante tener en cuenta diferentes planes de acción, tanto para cambios pequeños como para cambios grandes, de manera que se pueda modificar el proyecto rápidamente, y lograr un producto o servicio exitoso.

Anexos

Código de Arduino

```
/*************** INCLUDES ************************************************/
#include <I2CSoilMoistureSensor.h>
#include <Wire.h>
#include <String.h>
#include <GSMSSLClient.h>
#include <GSMSecurity.h>
#include <GSMModem.h>
#include <GPRS.h>
#include <PubSubClient.h>
#include <MQTT.h>

#include "secrets.h"
/*************** INCLUDES ************************************************/



/*************** DEFINES *************************************************/
/** Use Telcel's APN*/
#define TELCEL                  (1)
/** Use AT&T's APN (Mexico)*/
#define AT_T              (0)
/** Use Movistar's APN*/
#define MOVISTAR          (0)

/** Sets a test message to the buffer*/
```

```
#define TEST_MSG          (0)

/** Enables or disables the use of SSL*/
#define SSL_EN                  (0)

/** Use Mosquitto test broker DO NOT PUBLISH SENSITIVE DATA*/
/** Mosquitto test broker is public, so anyone may be listening,
do not publish any sensitive data.*/
#define MOSQUITTO_BRK     (1)
/** Use AWS broker*/
#define AWS_BRK                 (0)

/** Port to which the device will connect.
      Usually:
      1883 -> MQTT not secure
      8883 -> MQTT secure*/
#define PORT              (1883)
/** Max message size*/
#define MSG_SIZE          (100)
/** IMEI size*/
#define IMEI_SIZE         (16)
/** Size for the temp buffer*/
#define TEMP_BUFF_SIZE    (20)

/** Serial port baud rate*/
#define SER_PORT_BR       (115200)

/** Defines the time between two messages (ms)*/
#define DELAY_TIME        (60000)
/** Defines the delay between two I2C readings (ms)*/
#define I2C_DELAY         (50)

/** Defines the value to set decimal base (itoa function)*/
#define DECIMAL_BASE      (10)

/** Defines the digital pin 0*/
#define DIG_0                   (0)
/** Defines the digital pin 1*/
#define DIG_1                   (1)
/** Defines the digital pin 2*/
#define DIG_2                   (2)
/** LED off state for pull-down*/
#define LED_OFF                 (LOW)
/** LED on stat for pull-down*/
#define LED_ON                  (HIGH)

/** Reference voltage for the ADC*/
#define V_REF                   (3.3)
/** ADC resolution of the MKR GSM 1400*/
#define ADC_RES_BITS      (1023)
/** Resistor of the luminosity sensor*/
#define LIGHT_SENSOR_R    (68000)
/** Value to define 1 micro A*/
#define uA_1              (0.000001)

/** Defines a delay of 1 second*/
#define ONE_SEC_DELAY     (1000)
```

```
/** Defines the time the publish LED will be on (ms)*/
#define PUB_LED_DELAY     (2000)

/*************** DEFINES ************************************************/



/*************** VARIABLES **********************************************/
/** Object for the I2C sensor*/
I2CSoilMoistureSensor sensor;

/** Analog pin for the light sensor*/
int light_pin = A1;
/** Variable for the light sensor reading*/
int light_read = 0;
/** Variable to calculate the luminosity*/
double light_calc = 0;

/** Variable for the humidity*/
int humidity = 0;
/** Variable for the temperature*/
int temperature = 0;
/** Variable to store light as an integer*/
int luminosity = 0;

/** Variable for the timestamp*/
unsigned long timestamp = 0;

/** LED for when de device is connected to GPRS*/
int gprs_led = DIG_0;
/** LED for when the device has published*/
int publish_led = DIG_1;
/** LED for when the device has connected to MQTT*/
int mqtt_led = DIG_2;

/** SIM pin*/
const char pin[]  = "1111";

#if(TELCEL)
/** Telcel APN*/
const char apn[]  = "internet.itelcel.com";
const char login[] = "webgprs";
const char password[] = "webgprs2002";
#elif(AT_T)
/** AT&T APN*/
const char apn[]  = "modem.nexteldata.com.mx";
/** AT&T no tiene usuario ni contraseña para la APN
    utilizar NULL para ambos parámetros*/
#else
/** Movistar APN*/
const char apn[] = "internet.movistar.mx";
const char login[] = "movistar";
const char password[] = "movistar";
#endif

#if(MOSQUITTO_BRK)
/** Mosquitto server address*/
```

```
const char mqtt_server[] = "test.mosquitto.org";
#else
/** AWS server address*/
const char mqtt_server[] = "your-aws.broker.address.dom";
#endif

/** Variables to store the device's IMEI*/
String imei_str;
char IMEI[IMEI_SIZE] = {0};

/** MQTT topic into which the data will be published*/
const char topic[] = "/your/topic";

/** Array to store the message to be sent*/
char msg_to_be_sent[MSG_SIZE] = {0};
/** Buffer to construct the message and then send it*/
String msg_buff = "";
/** Length of the message to be sent*/
int msg_to_be_sent_len = 0;
/** Temporal buffer to convert from integers to ASCII*/
char temp_buff[TEMP_BUFF_SIZE] = {0};
/** Length of the content of the temporal buffer*/
int temp_buff_len = 0;

/** Password to encrypt temperature value*/
int temperature_pswd = 0x4D3;
/** Password to encrypt humidity value*/
int humidity_pswd = 0x6FA;
/** Password to encrypt luminosity value*/
int luminosity_pswd = 0xC37B8;

/** Counter for the XOR application*/
int xor_counter = 0;

/** JSON string for IMEI*/
const char imei_json[] = "{\"IMEI\":";
/** JSON string for timestamp*/
const char time_json[] = "\"timestamp\":";
/** JSON string for temperature*/
const char temp_json[] = "\"temp\":";
/** JSON string for humidity*/
const char hum_json[] = "\"hum\":";
/** JSON string for luminosity*/
const char lum_json[] = "\"lum\":";
/** Quotes character for JSON message*/
const char quotes[] = "\"";
/** Comma character for JSON message*/
const char comma[] = ",";
/** Closing braces for JSON message*/
const char eom[] = "}";

#if(SSL_EN)

/** Object for the SSL client*/
GSMSSLClient net_client;
/** Secure MQTT SSL client*/
PubSubClient mqttClient(mqtt_server, PORT, net_client);
```

```
#else

/** Object for the client (Not SSL)*/
GSMClient net_client;
/** Object for the MQTT client*/
MQTTClient client;

#endif
/** Object for security*/
GSMSecurity profile;
/** Object to access IMEI*/
GSMModem modem;
/** Object to attach to a network*/
GPRS gprs;
/** Object to acces the SIM*/
GSM gsmAccess;

/** Variable to keep count of time*/
unsigned long lastMillis = 0;
/**************** VARIABLES ***************************************************/



/*************** CONNECT ****************************************************/
/** Connects to both cellular network, and MQTT server*/
void connect()
{
	/** Connection state*/
	bool connected = false;

	Serial.print("\nconnecting to cellular network ...");

	/** After starting the modem with gsmAccess.begin()
	attach to the GPRS network with the APN, login and password */
	while (!connected)
	{
	    if ((gsmAccess.begin(pin) == GSM_READY) &&
			(gprs.attachGPRS(apn, login, password) == GPRS_READY))
	      {
			connected = true;
	      }

	      else
	      {
			Serial.print(".");
			delay(ONE_SEC_DELAY);
	      }
	      /** Repeats process until connected to network*/
	}
	/** Turns on connectivity LED*/
	digitalWrite(gprs_led, LED_ON);

	Serial.print("\nconnecting...");

	/** Connects to the MQTT server*/
#if(SSL_EN)
```

```
	while (!mqttClient.connect(IMEI))
#else
	while (!client.connect(IMEI))
#endif
	{
		Serial.print(".");
		delay(ONE_SEC_DELAY);
	}

	/** Turns on MQTT connection LED*/
	digitalWrite(mqtt_led, LED_ON);

	Serial.println("\nconnected!");
}
/*************** CONNECT ************************************************/



/*************** SETUP **************************************************/
/** Setup functions*/
void setup()
{
	/** Initializes serial communication through USB*/
	Serial.begin(SER_PORT_BR);
	Serial.print("Initialized\n");

	/** Initializes the 3 LED pins*/
	pinMode(gprs_led, OUTPUT);
	pinMode(publish_led, OUTPUT);
	pinMode(mqtt_led, OUTPUT);

	/** Turns off the 3 LEDs*/
	digitalWrite(gprs_led, LED_OFF);
	digitalWrite(publish_led, LED_OFF);
	digitalWrite(mqtt_led, LED_OFF);

#if(!SSL_EN)
	/** Connects to MQTT broker*/
	client.begin(mqtt_server, PORT, net_client);
#endif
	/** Initializes modem functions*/
	modem.begin();

	/** Gets the device's IMEI, and sets it to a char variable*/
	imei_str = (modem.getIMEI());
	imei_str.toCharArray(IMEI, sizeof(IMEI));
	Serial.print(imei_str);

	/** Initializes I2C for sensor reading*/
	Wire.begin();
	/** Initializes the I2C Soil Moisture sensor*/
	sensor.begin();

	/** Connects to cellular network and MQTT broker*/
	connect();
}
/*************** SETUP **************************************************/
```

```
/*************** LOOP ****************************************************/
/** Loop forever*/
void loop()
{
	/** MQTT loop*/
	client.loop();

	/** If the device is disconnected, tries to connect
	again */
#if(SSL_EN)
	if (!mqttClient.connected())
#else
	if (!client.connected())
#endif
	{
		/** Turns off both MQTT and GPRS connection LEDs*/
		digitalWrite(gprs_led, LED_OFF);
		digitalWrite(mqtt_led, LED_OFF);
		connect();
	}

	/** Publish a message roughly every minute*/
	if(DELAY_TIME < (millis() - lastMillis))
	{
		/** Gets current time*/
		lastMillis = millis();

		/** Gets the timestamp
			The timestamp format for MKR GSM 1400 is the seconds
			passed since January 1st, 1970*/
		timestamp = gsmAccess.getTime();

		/** Reads the temperature*/
	    temperature = sensor.getTemperature();
	    /** Gives a small delay for the I2C buffer to respond*/
	    delay(I2C_DELAY);
	    /** Reads the humidity*/
	    humidity = sensor.getCapacitance();

		/** Reads the light sensor*/
		light_read   = analogRead(light_pin);
		/** Gets the voltage value read by the ADC*/
		light_calc = ((double)(light_read) * V_REF) / (ADC_RES_BITS);
		/** Calculates the luminosity in Lux*/
		light_calc = (light_calc / (10 * LIGHT_SENSOR_R * uA_1));
		light_calc = pow(10, light_calc);
		Serial.println(light_calc);

		/** Cast into integer to be sent by the device*/
		luminosity = (int)(light_calc);

		/** Pass the variables through the passwords*/
		temperature ^= temperature_pswd;
		humidity ^= humidity_pswd;
```

```
            luminosity ^= luminosity_pswd;


#if(TEST_MSG)
            msg_to_be_sent[0] = 'W';
            msg_to_be_sent[1] = 'o';
            msg_to_be_sent[2] = 'r';
            msg_to_be_sent[3] = 'l';
            msg_to_be_sent[4] = 'd';
            msg_to_be_sent_len = 5;
#else
            /** Erases string buffers*/
            memset(msg_to_be_sent, '\0', sizeof(msg_to_be_sent));

            /** Sets the first variable {"IMEI":"value"*/
            strcat(msg_to_be_sent, imei_json);
            strcat(msg_to_be_sent, quotes);
            strcat(msg_to_be_sent, IMEI);
            strcat(msg_to_be_sent, quotes);

            strcat(msg_to_be_sent, comma);

            /** Sets the timestamp "timestamp":value*/
            strcat(msg_to_be_sent, time_json);
            memset(temp_buff, '\0', sizeof(temp_buff));
            itoa(timestamp, temp_buff, DECIMAL_BASE);
            strcat(msg_to_be_sent, temp_buff);

            strcat(msg_to_be_sent, comma);

            /** Sets the temperature "temp":value*/
            strcat(msg_to_be_sent, temp_json);
            memset(temp_buff, '\0', sizeof(temp_buff));
            itoa(temperature, temp_buff, DECIMAL_BASE);
            strcat(msg_to_be_sent, temp_buff);

            strcat(msg_to_be_sent, comma);

            /** Sets the humidity "hum":value*/
            strcat(msg_to_be_sent, hum_json);
            memset(temp_buff, '\0', sizeof(temp_buff));
            itoa(humidity, temp_buff, DECIMAL_BASE);
            strcat(msg_to_be_sent, temp_buff);

            strcat(msg_to_be_sent, comma);

            /** Sets the luminosity "lum":value*/
            strcat(msg_to_be_sent, lum_json);
            memset(temp_buff, '\0', sizeof(temp_buff));
            itoa(luminosity, temp_buff, DECIMAL_BASE);
            strcat(msg_to_be_sent, temp_buff);

            strcat(msg_to_be_sent, eom);

            /** Gets the string length*/
            msg_to_be_sent_len = strlen(msg_to_be_sent);
#endif
```

```
            /** Publishes a message*/
#if(SSL_EN)
            mqttClient.publish(topic, msg_to_be_sent, msg_to_be_sent_len);
#else
            client.publish(topic, msg_to_be_sent, msg_to_be_sent_len);
#endif

            /** Turns off the publish LED only for one second*/
            digitalWrite(publish_led, LED_ON);
            delay(PUB_LED_DELAY);
            digitalWrite(publish_led, LED_OFF);

            Serial.println("\nPublish!");
      }
}
/*************** LOOP ****************************************************/
```

Script de comunicación entre Mosquitto y AWS:

```
import paho.mqtt.client as mqtt
import datetime
import json
import ssl
import logging, traceback
import sys
import time

# Port for Mosquitto connection
PORT = 1883
# Port for AWS connection
SSL_PORT = 443

# Mosquitto broker address
mosq_broker = "test.mosquitto.org"
# Mosquitto client ID (Random string)
mosq_client_id = "e275e7835e404b4a7ca"

# ALPN protocol name (AWS)
IoT_protocol_name = "x-amzn-mqtt-ca"
# AWS broker address
aws_broker = "aws.broker.example.com"
# AWS client ID (Configured in AWS policies)
aws_client_id = "AWS_Client_ID_Example"

# CA certificate path
ca = "C:/your/path/to/cartificate/cert_name.pem"
# Client certificate path
```

```
cert = "C:/your/path/to/cartificate/cert_name.pem.crt"
# Private Key path
private = "C:/your/path/to/key/key_name.pem.key"

# Variable to define if a message must be sent
msg_ready = False
# Message to be sent
msg_to_be_sent = ""

# Topic to which the data will be published and received from
topic = "/topic/to/publish/example"

# Callback for Mosquitto connection
def on_connect_mosq(client, userdata, flags, rc):
  # Message to state a connection
  print("Connected to Mosquitto with result code " + str(rc))

  # Subscribe to Mosquitto topic to which the data will be
  # published by Arduino
  client.subscribe(topic)

# Callback for when a message is received from Mosquitto
def on_message_mosq(client, userdata, msg):
  global msg_ready
  global msg_to_be_sent

  # Sets the message to be sent as ready
  msg_ready = True
  # Sets the message to be sent
  msg_to_be_sent = msg.payload

# Sets the SSL configuration for a secure connection
def ssl_alpn():
    try:
        # Creates an SSL context
        ssl_context = ssl.create_default_context()
    # Sets the AWS protocol for ALPN
        ssl_context.set_alpn_protocols([IoT_protocol_name])
    # Sets the CA certificate
        ssl_context.load_verify_locations(cafile = ca)
    # Sets the client certificate and the private key
        ssl_context.load_cert_chain(certfile= cert, keyfile = private)

    # Returns the SSL context to be configured into MQTT
        return  ssl_context

    except Exception as e:
```

```
        print("exception ssl_alpn()")
        raise e

if __name__ == '__main__':

  try:
    # Creates the Mosquitto MQTT aws_client
    mosq_client = mqtt.Client(mosq_client_id)
    # Sets the callbacks
    mosq_client.on_connect = on_connect_mosq
    mosq_client.on_message = on_message_mosq

    # Connects to Mosquitto
    mosq_client.connect(mosq_broker, PORT)

    # Creates the AWS MQTT client
    aws_client = mqtt.Client(aws_client_id)
    # Configures the SSL context
    ssl_ctx = ssl_alpn()
    # Sets the SSL context
    aws_client.tls_set_context(context = ssl_ctx)
    # Connects to AWS
    aws_client.connect(aws_broker, port = SSL_PORT)

    # Starts both MQTT connections
    aws_client.loop_start()
    print("Connected to AWS")
    mosq_client.loop_start()

    while True:

      # When a message is received
      if True == msg_ready:
        # Sets the message as sent
        msg_ready = False
        print("Publish")
        # Publishes the message to Topic in AWS
        aws_client.publish(topic, msg_to_be_sent)

  # Exception to exit at CTRL + C
  except KeyboardInterrupt:
    # Disconnects from both servers
    aws_client.disconnect()
    print("Disconnected from AWS")
    mosq_client.disconnect()
    print("Disconnected from Mosquitto")
```

```
    print("END")
```

Código de función Lambda:

```
import json
import requests
import time
import boto3

# Ubidots access token
TOKEN = "your-ubidots-token"
# Ubidots stem address
address = "http://industrial.api.ubidots.com"

# Passwords to decrypt the message
temp_pswd = 0x4D3
hum_pswd = 0x6FA
lum_pswd = 0xC37B8

# Object to manage DynamoDB
dynamodb = boto3.resource('dynamodb')
# Object to manage the table IoT_DB
table = dynamodb.Table('IoT_DB')

# Posts via HTTP to ubidots
def post_request(payload, device_label):
  # sets the address and the label to the URL
  url = "{}/api/v1.6/devices/{}".format(address, device_label)
  # Headers for the post request
  headers = {"X-Auth-Token": TOKEN, "Content-Type": "application/json"}

  # Status of the post request
  status = 400
  # Number of attempts to post
  attempts = 0

  # If the status is not successful, tries to post_request
  # 5 times
  while status >= 400 and attempts <= 5:
    # Post request
    req = requests.post(url=url, headers=headers, json=payload)
    # Gets the request status
    status = req.status_code
    # Retries in 1 second
```

```
    attempts += 1
    time.sleep(1)

  # If could not publish in 5 trues, returns error
  if status >= 400:
    return False

  # Otherwise, returns success
  return True

# Lambda function handler
def lambda_handler(event, context):
  # Dictionary to be sent to Ubidots with error values
  msg_to_be_sent = {"temp":-1, "hum":-1, "lum":-1}

  # Decrypts the messages
  event["temp"] ^= temp_pswd
  event["hum"] ^= hum_pswd
  event["lum"] ^= lum_pswd

  # Sets the dictionary to be sent to Ubidots
  msg_to_be_sent["temp"] = event["temp"]
  msg_to_be_sent["hum"] = event["hum"]
  msg_to_be_sent["lum"] = event["lum"]

  # Puts data received into DynamoDB
  table.put_item(Item = event)

  # Publishes the message to ubidots via HTTP, to the proper device
  post_request(msg_to_be_sent, event["IMEI"])

    # Return value
  return {'statusCode': 200, 'body': json.dumps(msg_to_be_sent)}
```

Hojas de datos consultadas

I2C Soil Moisture Sensor: https://www.tindie.com/products/miceuz/i2c-soil-moisture-sensor/

SEN0114: https://datasheetspdf.com/pdf-file/770442/DFROBOT/SEN0114/1

SHT11: https://www.seeedstudio.com/Soil-Moisture-Temperature-Sensor-p-1356.html

LM35: http://www.ti.com/lit/ds/symlink/lm35.pdf?HQS=TI-null-null-mousermode-df-pf-null-wwe

MAX6627: https://www.mouser.mx/datasheet/2/256/MAX6627-MAX6628-370438.pdf

VEML7700: https://www.vishay.com/docs/84286/veml7700.pdf

APDS9007: https://docs.broadcom.com/doc/AV02-0512EN

GA1AS202WP: https://pdf1.alldatasheet.com/datasheet-pdf/view/582599/SHARP/GA1A1S202WP.html

Li-Yuan Conductivity, temperature and humidity: https://es.aliexpress.com/item/32793236869.html

Arduino Conductivity Sensor:

http://vi.raptor.ebaydesc.com/ws/eBayISAPI.dll?ViewItemDescV4&item=132746823890&category=57520&pm=1&ds=0&t=1559965893000&ver=0

Isolated EC probe: https://ufire.co/shop/

Arduino Uno: https://components101.com/microcontrollers/arduino-uno https://store.arduino.cc/usa/arduino-uno-rev3

Arduino MKR FOX 1200: https://store.arduino.cc/usa/arduino-mkrfox1200

Arduino MKR GSM 1400: https://store.arduino.cc/usa/mkr-gsm-1400

NXP FRDM-K64F: https://www.mouser.com/datasheet/2/813/K64P144M120SF5RM-1074828.pdf, https://os.mbed.com/media/uploads/GregC/frdm-k64f_ug_rev0.1.pdf

Referencias

(n.d.). Retrieved from http://www.infoagro.com/hortalizas/lechuga.htm

Zen Cart® Team. (n.d.). Guía: La luz en tus plantas. Retrieved from https://www.hydroenv.com.mx/catalogo/index.php?main_page=page&id=221

REVISTA DE CIENCIAS AGRÍCOLAS. (2014, November 11). PRODUCTIVIDAD DE LECHUGA Lactuca sativa EN CONDICIONES DE MACROTÚNEL EN SUELO Vitric haplustands [Dataset]. Retrieved May 21, 2019, from http://www.scielo.org.co/pdf/rcia/v31n2/v31n2a08.pdf

Casanova, M. (2017). ¿Qué es LoRa? Marzo 13, 2020, de Alfa IoT Sitio web: https://alfaiot.com/blog/ultimas-noticias-2/post/que-es-lora-2

García, C. (2012). Zigbee, Comunicación para Dispositivos. Marzo 13, 2020, de SG Sitio web: https://sg.com.mx/content/view/310

Ferrer, V. (2019). Qué es Sigfox. Marzo 13, 2020, de Vicente Ferrer Sitio web: https://vicentferrer.com/sigfox/

Munera, S. (2019). Modelado y evaluación de la tecnología Sigfox para NS3. (Tesis Ingeniería Informática). Marzo 13, 2020. Sitio web:

https://pdfs.semanticscholar.org/a425/374d9b01f457c5f448247af9b7428e05febe.pdf

Villagómez, C. (2018). Introducción a wifi (802.11 o WiFi). Marzo 13, 2020, de CMM Sitio web: https://es.ccm.net/contents/789-introduccion-a-wifi-802-11-o-wifi

Buchanan, B. (2019). So what is the encryption in 3G/4G networks?. Marzo 13, 2020, de Medium Sitio web: https://medium.com/asecuritysite-when-bob-met-alice/so-what-is-the-encryption-in-3g-4g-networks-139b8c0da3eb

Kroodo, M. (2019). 2G / 3G / 4G / 5G / LPWAN – Which to Choose for Your IoT Project?. Marzo 13, 2020, de 1oT Sitio web: https://1ot.mobi/resources/blog/gsm-3g-4g-lte-which-one-to-choose-for-your-iot-project?gclid=Cj0KCQjw6sHzBRCbARIsAF8FMpWvagfQZ-9xAXuXuw-kQYQ--0R_Y7GuCW4r-eOfSl0XPxCtXnD-BsUaAlX-EALw_wcB