Applied Computer Science, vol. 17, no. 2, pp. 14–27
doi: 10.23743/acs-2021-10

Submitted: 2021-06-01 | Revised: 2021-06-20 | Accepted: 2021-06-22

AWS, IoT, MQTT

Elizabeth PÉREZ [0000-0002-7250-0579]*, *Juan C. ARAIZA* [0000-0002-6188-6867]*, *Dreysy POZOS* [0000-0002-7797-1230]*, *Edmundo BONILLA* [0000-0003-2062-4219]*, *José C. HERNÁNDEZ* [0000-0001-7245-9564]*, *Jesús A. CORTES***

APPLICATION FOR FUNCTIONALITY AND REGISTRATION IN THE CLOUD OF A MICROCONTROLLER DEVELOPMENT BOARD FOR IOT IN AWS

Abstract

The use of the Amazon Web Services cloud enables new functionalities that are not possible with traditional solutions: low latency, local data processing and storage, and direct connectivity to other cloud services. Reimagining the way IoT connectivity services are presented by combining AWS cloud technology with mobile connectivity offers rapid prototyping to help connect devices natively over Wi-Fi. For this, the MQTT communication protocol is used to interact with the IoT device and exchange data, which allows controlling the basic functions of a sensor node. The installation is realized through a software development kit (SDK), which allows the creation of an application for Android devices. This solution gives the option to integrate together, improving the connectivity of the IoT system. The results enable board logging and network configuration, and can also be used to control the IoT device. The embedded firmware provides the required security functions.

1. INTRODUCTION

The results presented in this article are part of a development platform for Internet of Things (IoT) Solutions. But, are cloud computing technologies really necessary to create an IoT scenario?

If talk about the technical requirements, the immediate answer is no. Data processing and commands can be performed locally through an internet connection. However, there are interesting benefits for IoT applications if we are going to use the cloud. To mention some of them, there is scalability, system availability, the ability to aggregate large amounts of data, reduced infrastructure costs, etc.

*Computer Systems Department, TecNM/Campus Apizaco Tlaxcala, Mexico, m19371374@apizaco.tecnm.mx, m19371363@apizaco.tecnm.mx, m19371373@apizaco.tecnm.mx, edmundo.bh@apizaco.tecnm.mx, crispin.hh@apizaco.tecnm.mx
**Microside Technology, Software Development Department, Tlaxcala de Xicohténcatl, México, contacto@microside.com

IoT (Internet of things) is an emerging architecture based on the global Internet that facilitates the exchange of goods and services between supply chain networks and has a significant impact on the security and privacy of the actors involved(Salazar & Silvestre, n.d.). Is an extensive term that has infinite applications, however, that devices are connected or have some degree of intelligence is not new, the question is how IoT proposes to do it today, especially with the great progress it has gained with the use of mobile technology.

Amazon Web Services as a cloud computing solution became the most adopted and comprehensive platform in the world, which offers more than 200 integral services of data centers globally (AWS, n.d.).

The AWS architecture for this proposal supports devices to report their status by publishing messages using the MQTT protocol to exchange data with the cloud and the AWS mobile application (Guth et al., 2018). The message is sent to AWS IoT Device Shadow, and sends it to all users subscribed to that topic. Each device has a shadow object, or in other words a virtual representation of the device, which is used to save and retrieve information about the current state of the target in a JSON document, divided into a last reported state and a desired state (AWS Device Shadow, n.d.).

This application can send a request with the current state of the device or a change in the status of the device. When the message gets to the agent, the rules engine provides message processing and integration with other AWS services. The rest of this article presents the most significant results obtained and is organized as follows: section 2, related works and state of the problem; section 3, methodology and design procedure; section 4, results obtained; sections 5 and 6 refer to discussion and conclusion of this work.

2. LITERATURE REVIEW AND PROBLEM STATEMENT

The research studies consulted for this paper adopted AWS Serverless architecture as the cloud solution, focusing on a framework of mobile applications that control microcontroller devices and give way to IoT scenarios.

Wasoontarajaroen et al. (Wasoontarajaroen, Pawasan & Chamnanphrai, 2017) present an IoT device that was developed to monitor electrical energy consumption in a building. As a low cost IoT device, it consists of three modules available, including electrical power sensors, an Arduino Nano Mini microcontroller and a Wi-Fi board. The IoT device was tested by performing electrical energy measurement for a week. The test results confirmed that the device performed successfully, which could collect electrical energy data to support efficient energy management in the future.

In this paper, different techniques used for data classification in IoT devices using services provided by the AWS Kinesis platform were analyzed (Quadri & Yadav, 2018). Using Amazon Kinesis, the complete IoT architecture was analyzed in a real-world scenario, where the object of study was an agricultural station used to predict soil moisture. Subsequently, it was proposed to implement algorithms and methods such as an API for more efficient data classification and also to improve security measures, (AWS Kinesis, n.d.).

The main objective proposed by (Muñoz Carrasco & Garrido Márquez, 2018) focuses on the development of a mobile application that displays sensor data in real time and through which it is possible to act on them. All this depends on the development of two applications for the Arduino and Raspberry boards, the last one with Android Things. A program was developed on Arduino to read analog and digital sensors, connect to Wi-Fi and both receive and send messages using MQTT.

Guha Roy et al. (Guha Roy et al., 2018) proposes a trade-off solution to explore the unusual behavior of Wireless Sensor Network gateways with theoretical expressions and apply dynamic gateway selection formulas to reduce end-to-end delay as well as increase delivery rate of packages. The project was developed using sensor devices, Arduino Uno, Raspberry Pi 3, Amazon web services, the MQTT broker, and the private cloud platform in their Mobile Cloud computer lab.

Amazon Web Services has gained popularity with IoT Core Platform, which integrates all the functions necessary to develop an IoT system. (AWS IoT Core, n.d.). The objective of (Imtiaz Jaya & Hossain, 2019) focused on exploring some of these functions and their integration into the proposed project, which consisted of the development and presentation of a prototype of an airflow control system for home automation, using AWS IoT Core and the MQTT protocol on the WebSocket server.

Kim et al. (Kim et al., 2019) proposes an MQTT as a communication protocol between Amazon Web Services and a mobile application. The problem detected in this research is the hot water wasted at sea by the electric companies. For this purpose, the hot water thermal energy management system used in an Internet of Things based fish farming system is developed. A control and monitoring system for an intelligent fish farm was proposed by building an intelligent fish that has the function of detecting and monitoring by various sensors elements such as oxygen, temperature and water level.

Villamil & Guarda (2019) focuses on the development of a mobile App with MIT app inventor, based on the analysis of an agile methodology for the development of mobile applications for the Internet of Things. In addition, it can be controlled from a LAN or WAN network by interfacing with the free hardware development boards Arduino Uno and Arduino Ethernet. The technical space is a data center in which temperature, humidity, sound recording and reproduction can be monitored by means of sensors.

The system proposed by (Mullapudi et al., 2020) is an advanced weather monitoring solution that uses AWS IoT to make real-time data easily accessible over a very wide range. The system monitors: the weather and its changes such as temperature, humidity, pressure, altitude using sensors and a Raspberry pi card as a controller. Data from these sensors is hosted in AWS Dynamo DB, which can handle more than 10 billion requests per day and can support peaks of more than 20 million requests per second (AWS DynamoDB, n.d.).

In this document (Wu et al., 2020), a new IoT fake app detection method, called MSimDroid, based on multidimensional similarities is proposed to mitigate the threat. This method focuses on application markets, and consists of complete application similarity, resource similarity, code similarity and their joint strategy. For the similarity calculation, a distinctive algorithm was designed based on the characteristic of different false patterns. Experiments showed that the accuracy of MSimDroid is over 99.31% on the ground truth dataset and 97.43% in the wild.

3. DESIGN PROCEDURE

3.1. Methodology

Three essential phases are essential in the method used for the development of the proposed model. These phases are: Design, Development and Validation, which are described below in Fig. 1.
In the Design phase, the general requirements and planning of the IoT solution are defined, wich are described in section 3.4. The design of each of the components is carried out, getting the screen design of the mobile interface, the design of the physical infrastructure and the choice of AWS cloud services. In the next phase, the development of each of the components of the system is completed, following the designs and requirements of the Design phase. The Validation phase includes a process of test protocols, which allows the verification of the correct operation of each component and then the integration of these components to build the system, validating the full compliance of the requirements stipulated in the Design phase.

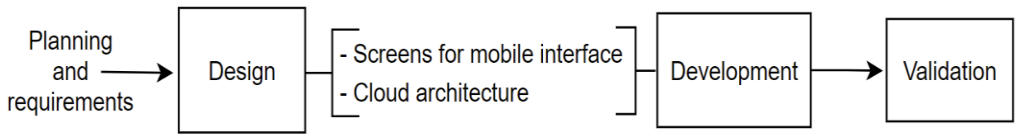


Fig. 1. Methodology block diagram

3.2. Hardware Requirements

The following hardware components are required for this work: Wi-Fi microcontroller development board, mobile device with android operating system with version 6.0 or higher and a Wi-Fi router.

3.2.1. WiFi microcontroller development board

These days, there is a wide range of development boards that can be classified in several ways. According, development platforms can be classified into four groups: microcontroller-based, microprocessor-based, Field Programmable Gate Array (FPGA) and based and hybrid development platforms (Auer & Zutin, 2018).
The design of this work uses a development board includes a preloaded IEEE 802.11 b/g/n network controller for Internet of Things (IoT) applications. The main components that describe this kind of hardware are presented below in Fig.2:

– Qualified IEEE 802.11 b/g/n Network Controller Module,
– On-Board Host Microcontroller (SAML21G18B),
– CryptoAuthentication Device (ATECC608A),
– On-Board Li-Ion/Li-Po Charge Management Controller (MCP73833),
– On-Board Environment Sensor (BME280),
– Sensor de luz integrado (VEML6030).

The development board is designed to be powered by a USB interface. The central part of the board is the SAML21G18B microcontroller, which controls all the other devices that make up the board. The network controller module provides Wi-Fi capabilities. The ATECC608A handles certificate storage and authentication.

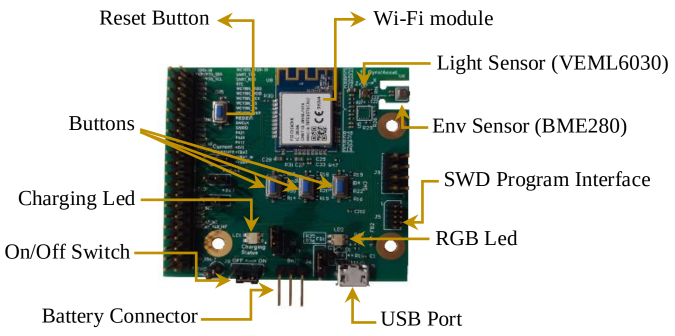


Fig. 2. Wi-Fi microcontroller development board

The development board is designed to be powered by a USB interface. The central part of the board is the SAML21G18B microcontroller, which controls all the other devices that make up the board. The network controller module provides Wi-Fi capabilities. The ATECC608A handles certificate storage and authentication.

3.3. Boot Manager Settings

A boot-loader is an application whose primary purpose is to allow a system's software to be updated without the use of specialized hardware. They can communicate over a variety of protocols such as USART, CAN, I2C, Ethernet, USB and the list goes on with all the protocols that exist. Systems with bootloaders have at least two program images coexisting on the same microcontroller and must include fork code that performs a check to see if an attempt is being made to update the software (Beningo, 2015).

For this project, the boot-loader allows programming of MCU firmware via USB port using Atmel Studio. The application firmware is used to test connection to AWS IoT. The firmware connects the board to the AWS IoT Cloud, updates the sensor data, gets the status of the leds, and publishes it to AWS Shadow. This is so that devices can communicate their status to AWS IoT and AWS IoT publishes MQTT messages to inform devices and applications of changes and events. Precisely the Message Queuing Telemetry Transport (MQTT) protocol is designed for light use of publish / subscribe messaging transport (Tsai, Hsu & Lo, 2020).

It´s important to emphasize that this service adds shadows to AWS IoT objects. Shadows can make a device's mode available for apps and other services, whether or not the device is connected to AWS IoT.

Boot-Loader mode is activated by pressing and holding the SW1 switch before turn on the board. The Samba GUI is used to load a firmware on the board using the USB port. The bootloader is saved at memory address 0x0-0x2000. The memory address for booting the firmware of an application is 0x2000.

The boot-loader can be reutilized even after modifying the application firmware and programming a new application firmware. Fig.3. shows a state machine that summarizes the bootloader process when Samba starts.

When loading the binary image into the device, only the upper part of flash starting at 0x2000 address should be programmed. Any attempt to write to the Samba Boot region using Samba GUI commands will be aborted and will return an error.

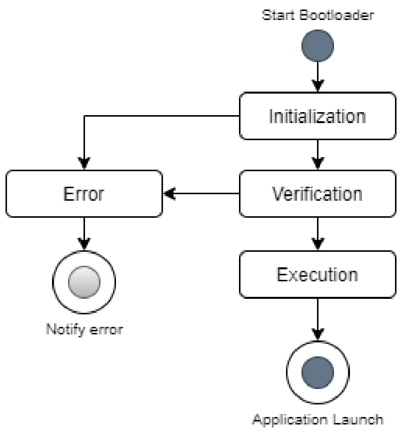


Fig. 3. Boot-Loader process

3.3.1. Firmware

The firmware of a microcontroller unit is a program consisting of a series of simple instructions intended to process data. For the most part it works like this: read input information, monitor the current state, develop a series of output instructions and send those output instructions. For this case, the option to use the MCU will be to turn on an LED when a certain condition is satisfied.

There are middleware modules between the application layer and the device driver layer. The middleware consists of different agents, such as resource manager, access controller scheduler, etc. Agents are small programs or sets of rules to execute a task. These agents depend on both the software and hardware of the connected device (Chang et al., 2020).

To develop the application, the use of middleware APIs was required to configure the device drivers. The following Tab.1. shows the middleware APIs required for this project:

Tab. 1. Middleware APIS

Middleware Modules	Function
Button	Read button value.
Cloud Wrapper	Connect/Disconnect cloud, publish/subscribe MQTT channel.
Env_Sensor	Read sensor value.
IoT Message	Generate or parson JSON message to/from the AWS Shadow.
Led	Control Led LD2 color and blinking behavior.
Socket	Network socket API.
WLAN	Wlan features like app scanning, app connection, and so on.

3.4. Software Requirements

The software used to successfully complete the objectives of the work are described: Samba v2.18 for programming the Application firmware of the board. Atmel Studio was used for the board firmware compilation procedure. Python scripts were used to provision the Wi-Fi board. The development of the mobile application required the official integrated development environment Android Studio. In the same way, AWS services were used for the serverless architecture of the project. Amazon Cognito was used to control users, Lambda to manage the policies that include an access token that contains the mobile application and DynamoDB to store information from the board registration process.

3.4.1. Serverless architecture design with AWS

There is a way to build and deploy APIs, where there is no need to manage or maintain servers. These types of architectures are known as Serverless Architectures. Serverless computing has emerged as a new paradigm for the deployment of applications and services where the application logic is divided into functions that are executed in response to events (Parres-Peredo, Piza-Davila & Cervantes, 2019). To build our mobile back-end on AWS IoT Core, the main services for this architecture are described: Amazon Cognito, Amazon DynamoDB and AWS Lambda.

Amazon Cognito was used to provide user identity to end users of the product, including registration and login functions in the mobile application. With Amazon Cognito, the user can register and login to the account and control the development board after successful authentication. Amazon Cognito returns user group tokens to the application. For the case, tokens can be used to grant users access to their own server-side resources or the Amazon API Gateway as represented in the Fig.5.

The DynamoDB table collects the user account id, the device id and the device name. The mobile application saves this information in the DynamoDB table during the board check-in procedure.

The Lambda function browses the AWS DynamoDB table to find the element id of the device that corresponds to the account id of the application. Then, the Lambda function code updates the value to AWS IoT Shadow and gets the device's AWS IoT Shadow to control the led.

To exchange data from the device to the IoT backend, messages must be published via the MQTT protocol to the AWS IoT Core message broker, this process is shown in the description of the solution in Fig. 6.

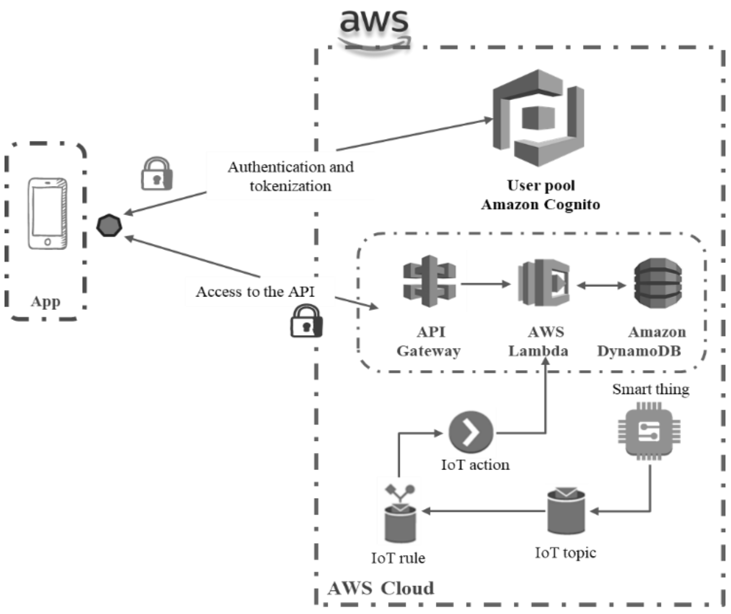


Fig. 5. Serverless architecture in AWS

The AWS IoT Device Shadow service collects and retrieves the current status information from the board. The board updates the sensor and LED data in AWS IoT by posting an MQTT message in Shadow MQTT and thus obtains the latest LED status from AWS IoT by subscribing to Shadow MQTT/update/delta topics. The shared message is shown in the Fig.7. This is how the mobile application controls the LED color by posting an MQTT message on Shadow.

The MCU runs AWS IoT sdk to connect the AWS IoT cloud board. AWS IoT sdk includes the cloud connection API and the MQTT signature and publishing API. The MQTT client ID is required for the connection and the subject key identifier of the device certification, which is used as the MQTT client ID. The board connects to the AWS IoT cloud with the TLS encryption suite 1. 2. The device certificate and private key are stored in the cryptographic authentication device.

The device's certificate and private key are stored in the Crypto Authentication ECC608 device. To perform authentication with the cloud, the certificate and key in ECC608 are used. After Cloud Configuration, Id's values corresponding to the identity group are collected from the Cognito console identity group, the AWS IoT Endpoint, which can be found in the AWS IoT console configuration page, to build the application apk file, as shown in Fig. 8.

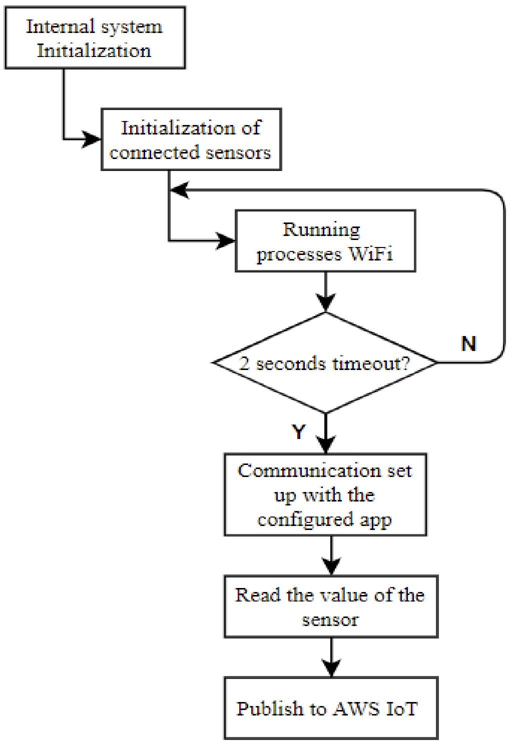
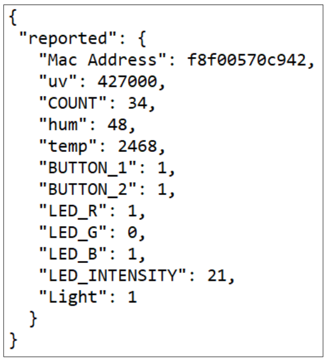
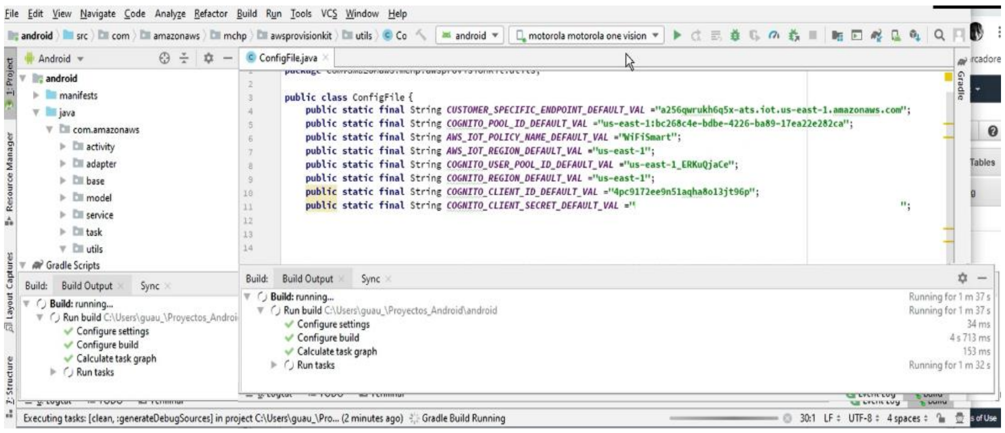

Internal system Initialization
Initialization of connected sensors
Running processes WiFi
2 seconds timeout?
N
Y
Communication set up with the configured app
Read the value of the sensor
Publish to AWS IoT

Fig. 6. Application flowchart



```
{
 "reported": {
   "Mac Address": f8f00570c942,
   "uv": 427000,
   "COUNT": 34,
   "hum": 48,
   "temp": 2468,
   "BUTTON_1": 1,
   "BUTTON_2": 1,
   "LED_R": 1,
   "LED_G": 0,
   "LED_B": 1,
   "LED_INTENSITY": 21,
   "Light": 1
  }
}
```

Fig.7. Shared shadow message



```
public class ConfigFile {
    public static final String CUSTOMER_SPECIFIC_ENDPOINT_DEFAULT_VAL ="a256qwrukh6q5x-ats.iot.us-east-1.amazonaws.com";
    public static final String COGNITO_POOL_ID_DEFAULT_VAL ="us-east-1:bc268c4e-bdbe-4226-ba89-17ea22e282ca";
    public static final String AWS_IOT_POLICY_NAME_DEFAULT_VAL ="WiFiSmart";
    public static final String AWS_IOT_REGION_DEFAULT_VAL ="us-east-1";
    public static final String COGNITO_USER_POOL_ID_DEFAULT_VAL ="us-east-1_ERKuQjaCe";
    public static final String COGNITO_REGION_DEFAULT_VAL ="us-east-1";
    public static final String COGNITO_CLIENT_ID_DEFAULT_VAL ="4pc9172ee9n51aqha8o13jt96p";
    public static final String COGNITO_CLIENT_SECRET_DEFAULT_VAL ="                    ";
```

```
Build: running...                                              Running for 1 m 37 s
  Run build C:\Users\guau_\Proyectos_Android\android           Running for 1 m 37 s
    Configure settings                                         34 ms
    Configure build                                            4 s 713 ms
    Calculate task graph                                       153 ms
    Run tasks                                                  Running for 1 m 32 s
```

```
Executing tasks: [clean, :generateDebugSources] in project C:\Users\guau_\Pro... (2 minutes ago)   Gradle Build Running
```

Fig. 8. Mobile app source file

4. SOLUTION DESCRIPTION AND RESULTS

When the development board is turned on, the internal MCU system and all connected sensors go through the initialization stage. After this stage, the board tries to connect to the configured app. LD2 flashes blue every 500 ms during this process. When the board successfully connects to the app and obtains the IP address, the LED flashes blue every 100 ms. Then, the board tries to connect to AWS IoT. After successful connection to the cloud, LD2 remains solid blue as shown in Fig. 11. The board tests the sensor data every two seconds.If the sensor data changes, the data is published to AWS IoT Shadow, this process is described in the flowchart Fig. 6. At this point, can control the microcontroller device with the mobile application.

Led LD2 shows the operation of the board. When the board connects to the AWS cloud, LD2 turns on and the color can be changed using the mobile app; Tab. 2 describes the behavior depending the status of LD2 led.

Tab. 2. LD2 States

LD2	Status	Description
Red	Blinks for every 0.5 seconds	Network provisioning mode.
	Blinks for every 100 milliseconds	The mobile app successfully connects to the card for Network Configuration.
Blue	Blinks for every 0.5 seconds	Attempts to connect to the Application.
	Blinks for every 100 milliseconds	It successfully connects to the Application, and attempts to connect to AWS IoT Cloud.
	Turn on	The Wi-Fi sensor board successfully connects to the AWS IoT cloud.
Green	Blinks for every 0.5 seconds	Firmware update mode.

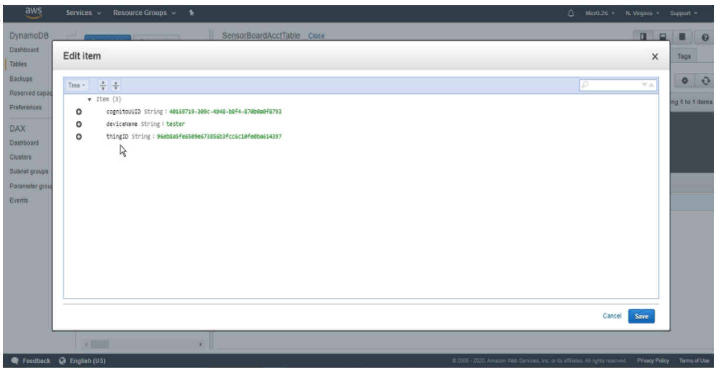
The following process is used to register the card and configure the network with the application: The board is connected to the network and allows control of the board using the mobile application account, for this it is necessary to register as a new user.

This record will be reflected in our User Group in the Amazon Cognito management console. For the test, have the user *testuser* item a) Fig. 13. During this process, the board must be prepared for board registration mode and network configuration. The LD2 Led flashes red.

Then, the mobile application acts as a Wi-Fi station to connect the board. A TCP tunnel is created with port 8899. The SSID of the target network and the password phrase are transferred from the mobile application to the target in this tunnel. When the board successfully obtains the SSID and password phrase, it switches to Wi-Fi station mode and connects to the target App using the Wi-Fi credential.

In this process, the Device Thing ID is also transferred from the board to the mobile application. When the mobile application obtains the Device Thing ID Fig.9, joins the Device Thing ID with the account ID of the mobile application and the name of the device on the board in an AWS DynamoDB table. Fig.12.

When the user sign in to the mobile app, the mobile app scans the table in AWS DynamoDB to display the dashboard information that corresponds to their mobile app account ID.



```
▼ Item {3}
    cognitoUUID String : 40169719-309c-4b48-b8f4-870b0a0f8793
    deviceName String : tester
    thingID String : 96eb8a5fe6509e673856b3fcc6c10fe0ba614397
```

Fig. 9. Device thing Id

Fig. 13. show the process that the application follows to register a new device, item a) of the same figure, shows the parameters of the card, which are the color of the LED LD2, the current temperature and humidity.

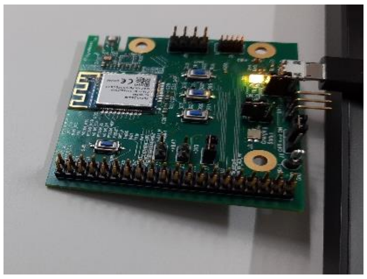


Fig. 10. Provisioning status

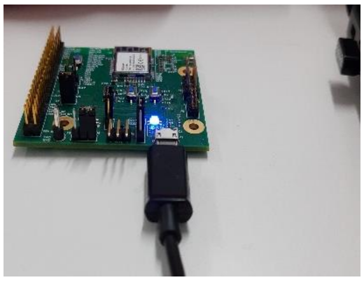
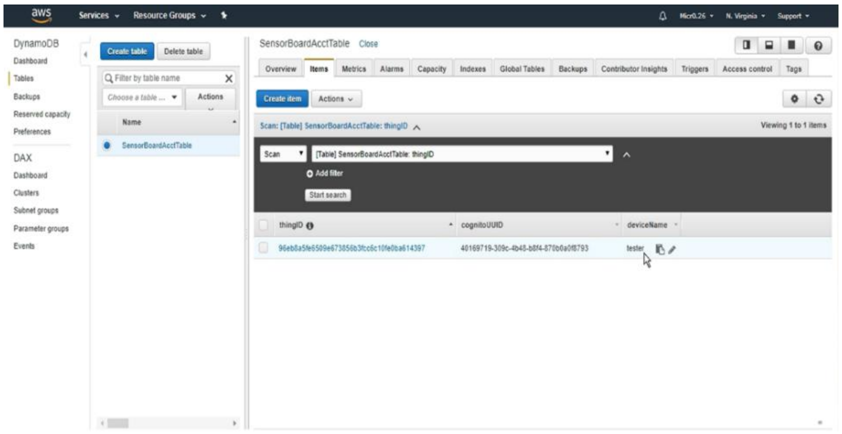


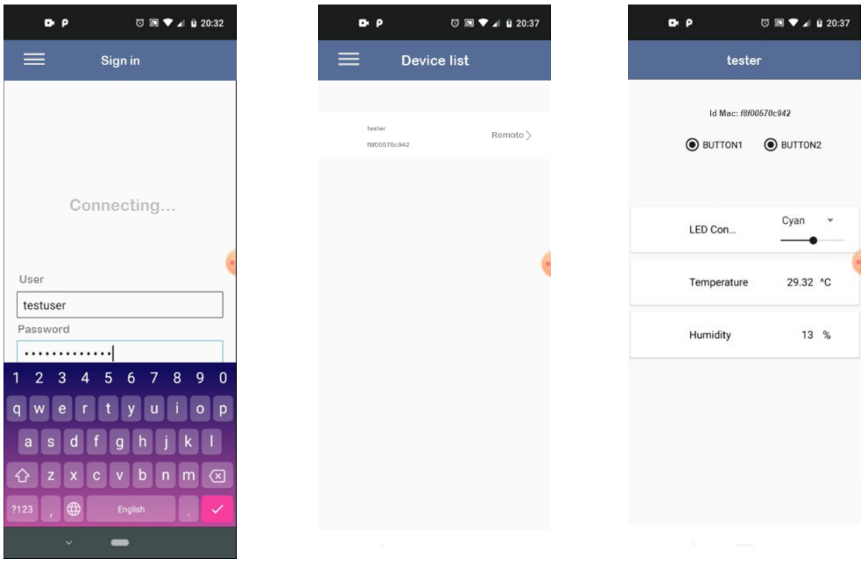
Fig. 11. Successful connection status


SensorBoardAcctTable

Scan: [Table] SensorBoardAcctTable: thingID — Viewing 1 to 1 items

thingID	cognitoUUID	deviceName
96eb8a5fe6509e673856b3fcc6c10fe0ba614397	40169719-309c-4b48-b8f4-870b0a0f8793	tester

Fig. 12. Table with thing information in DynamoDB


Sign in

Connecting...

User

testuser

Password

•••••••••••••

Device list

tester — f8f00570c942 — Remoto

tester

Id Mac: f8f00570c942

BUTTON1 BUTTON2

LED Con...	Cyan
Temperature	29.32 °C
Humidity	13 %

a) Login **b) Paired devices** **c) Device control**

Fig. 13. App Screenshots

5. DISCUSSION

The percentage of progress of the project is calculated based on (DocIRS Document Information Retrieval Systems, n.d.), and the subprojects to be completed for the second part of this project are listed in Tab.3.

It's important to emphasize that the AWS cloud registration goal is successfully completed, as well as the basic control of Led LD2, and the knowledge of the humidity and temperature of the microcontroller board. With the cloud registration of an object, is possible manage the number of connected and controlled devices and ensure a successful connection for the subsequent implementation of the Alexa voice assistant. The mobile application only presented compatibility issues for higher versions of Android 9 (Pie), so these requests will be resolved subsequently.

$$AP = \sum_{i=1}^{n} B_i \cdot AS_i \tag{1}$$

$$AS = \frac{N^\circ\ completed\ subproject\ processes}{N^\circ\ total\ subproject\ processes\ completed} \cdot 100 \tag{2}$$

where: *AP* – project progress measured as a percentage,
n – no. of subprojects in the project,
B – weighting (ratio of processes in the subproject/total number of processes),
AS – subproject progress in percentage,
i – summation index, *i* = 1,2, 3…*n*.

Tab. 3. Project progress

Subprojects	**% state of progress**
Planning ✓ Data collection ✓ Analysis ✓ Requirements	100
Design ✓ AWS Architecture ✓ Interface App mobile	100
Development ✓ Boot-Loader Firmware ✓ Application Firmware ✓ AWS Provision ✓ Mobile App AWS Architecture Alexa Skills	66.66
Testing ✓ System testing ✓ Document problems found Correct issues found	66.66
AP= 71.42%	

5. CONCLUSIONS AND FUTURE WORK

The proposed architecture in AWS to manage IoT devices had a positive effect, for the continuous registration of devices, after having tested with different certificates. This system has the ability to scale the functionality of different microcontroller development cards for Wi-Fi. As well as the technological interaction of users with connected devices. Future research areas include:

1. Incorporate the AWS Amplify framework as a user administrator resource to easily manage application content outside of the AWS console.
2. Integration of the Amazon Alexa Voice Assistant.
3. Lambda functions to process Alexa skill directives.

REFERENCES

Auer, M.E., & Zutin, D.G. (2018). *Online Engineering & Internet of Things. Proceedings of the 14th International Conference on Remote Engineering and Virtual Instrumentation REV 2017, held 15–17 March 2017, Columbia University, New York, USA.* Springer.

AWS. (n.d.). *Amazon Web Services*. Retrieved March 18, 2021 from https://aws.amazon.com/es/what-is-aws

AWS Kinesis. (n.d.). *Amazon Web Services*. Retrieved March 18, 2021 from https://aws.amazon.com/es/kinesis

AWS Device Shadow. (n.d.). *Amazon Web Services*. Retrieved March 18, 2021 from https://docs.aws.amazon.com/es_es/iot/latest/developerguide/iot-device-shadows.html

AWS DynamoDB. (n.d.). *Amazon Web Services*. Retrieved March 18, 2021 from https://aws.amazon.com/es/dynamodb

AWS IoT Core. (n.d.). *Amazon Web Services*. Retrieved March 18, 2021 from https://aws.amazon.com/es/iot-core/

Beningo, B.J. (2015). *Bootloader design for MCUs in Embedded Systems, Rev A2*. Beningo Engineering.

Chang, K.C., Chu, K.C., Wang, H.C., Lin, Y.C., & Pan, J.S. (2020). Agent-based middleware framework using distributed CPS for improving resource utilization in smart city. *Future Generation Computer Systems, 108*, 445–453. https://doi.org/10.1016/j.future.2020.03.006

DocIRS Document Information Retrieval Systems. (n.d.). *DocIRS Technology*. Retrieved March 18, 2021 from https://www.docirs.cl

Guha Roy, D., Mahato, B., De, D., & Buyya, R. (2018). Application-aware end-to-end delay and message loss estimation in Internet of Things (IoT) — MQTT-SN protocols. *Future Generation Computer Systems, 89*, 300–316. https://doi.org/10.1016/j.future.2018.06.040

Guth, J., Breitenbücher, U., Falkenthal, M., Fremantle, P., Kopp, O., Leymann, F., & Reinfurt, L. (2018). *A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences*. In B. Di Martino, K.C. Li, L. Yang, A. Esposito (Eds.), *Internet of Everything. Internet of Things (Technology, Communications and Computing)* (pp. 81–101). Springer. https://doi.org/10.1007/978-981-10-5861-5_4

Imtiaz Jaya, N., & Hossain, M.F. (2019). A Prototype Air Flow Control System for Home Automation Using MQTT over Websocket in AWS IoT Core. In *Proceedings – 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2018* (pp. 111–117). IEEE. https://doi.org/10.1109/CyberC.2018.00032

Kim, Y., Lee, N., Kim, B., & Shin, K. (2019). Realization of IoT based fish farm control using mobile app. In *Proceedings - 2018 International Symposium on Computer, Consumer and Control* (pp. 189–192). IEEE. https://doi.org/10.1109/IS3C.2018.00055

Mullapudi, R.K., Mallika, G., Nikitha, K., & Bhanu Sree, M. (2020). Weather Monitoring using AWS Cloud Computing. *International Journal of Advance Research and Innovation*, *8*(2), 205–208.

Muñoz Carrasco, J., & Garrido Márquez, D. (2018). *IoT home automation application using Android Things*. https://riuma.uma.es/xmlui/bitstream/handle/10630/17551/Mu%C3%B1oz%20Carrasco%2C%20Juan%20Memoria.pdf?sequence=1&isAllowed=y

Parres-Peredo, A., Piza-Davila, I., & Cervantes, F. (2019). Building and evaluating user network profiles for cybersecurity using serverless architecture. In *2019 42nd International Conference on Telecommunications and Signal Processing, TSP 2019* (pp. 164–167). IEEE. https://doi.org/10.1109/TSP.2019.8768825

Quadri, N.S., & Yadav, K. (2018). Efficient Data Classification for IoT Devices using AWS Kinesis Platform. In *21st Saudi Computer Society National Computer Conference, NCC 2018* (pp. 1–5). IEEE. https://doi.org/10.1109/NCG.2018.8593105

Salazar, J., & Silvestre, Y. S. (n.d.). *Internet de las cosas*. https://core.ac.uk/download/pdf/81581111.pdf

Tsai, M.H., Hsu, Y.C., & Lo, N.W. (2020). An Efficient Blockchain-based Firmware Update Framework for IoT Environment. In *Proceedings - 2020 15th Asia Joint Conference on Information Security, AsiaJCIS 2020* (pp. 121–127). IEEE. https://doi.org/10.1109/AsiaJCIS50894.2020.00030

Villamil, X., & Guarda, T. (2019). Mobile application developed with agile methodology for IoT controlled from a LAN / WAN network with a free hardware development board (Arduino). *Iberian Journal of Information Systems and Technologies*, *17*, 379–392.

Wasoontarajaroen, S., Pawasan, K., & Chamnanphrai, V. (2017). Development of an IoT device for monitoring electrical energy consumption. In *2017 9th International Conference on Information Technology and Electrical Engineering, ICITEE 2017* (pp. 1–4). IEEE. https://doi.org/10.1109/ICITEED.2017.8250475

Wu, P., Liu, D., Wang, J., Yuan, B., & Kuang, W. (2020). Detection of Fake IoT App Based on Multidimensional Similarity. *IEEE Internet of Things Journal, 7*(8), 7021–7031. https://doi.org/10.1109/JIOT.2020.2981693