IoT authorization with web application

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor’s thesis

Degree Program in Automation Engineering

Spring 2021

Nam Huynh

Automation and Electrical Engineering,
Valkeakoski

Author	Nam Huynh	**Year** 2021
Subject	IoT authorization with web application	
Supervisor(s)	Juhani Henttonen	

ABSTRACT

Today, there are many devices which are connected to the internet in buildings, homes or public places. People cannot deny the convenience provided by IOT. However, lots of risks also come along as security, effectiveness.

This project is about building a web platform that helps the administration by allowing or prohibiting users from controlling connected devices. With Amazon Web Services including IoT, authorization, server; the website can guarantee a high level of security.

Keywords Amazon Web Services, IoT authorization, MQTT protocol, AWS IoT.

Pages 30 pages including references 02 pages and appendices 01 page.

CONTENTS

1 INTRODUCTION

Nowadays, IoT becomes more and more popular because of its significance and productive efficiency on human activities. When all the devices share connected to the network, communication and connection between devices become very easy. They can be controlled and managed. Although IoT is currently applied mostly in the industries like agriculture, smart cities, logistics, …, there are still a lot of rooms for the advantages of IOT in daily lives. For example, in public places like university buildings, devices should be applied with IoT so that lecturers, students can control devices or be notified about device's status on their smartphones. There are many benefits if IoT is applied such as saving energy, receiving notification immediately when something wrong happened like temperature is too high or booking room, … However, security and management are huge troubles that makes IoT not be applied in public places. If everybody has the right to access the IoT network, it would be complicated to manage users and the system could be hacked by non-identified people.

To solve the situation, I build this project by myself for users to log in to the IoT network easily by their Google accounts and be managed by admin at the same time. Basically, between users and the IoT network there will be another layer called "admin". Admin position has the role to control everything including devices and normal users. Therefore, this project is for 2 types of users: admin (e.g. IoT manager) and normal users (e.g. lecturers, students). Admin has access to all devices and is able to allow or prevent normal users from accessing the IoT network. This is why the thesis is called "IoT Authorization", basically the only way that normal users can access the network is to be authorized by admin.

The project was built like a web app so people can interact with the app on their phones or computers. To build the app, there are some technologies which are researched and used. All the frameworks, programming languages and technologies used are Reactjs (front-end library), Nodejs (back-end Javascript runtime), MongoDb (non-SQL database), AWS (for services like IoT, hosting server, domain name register). To run and experiment this project in real time, a Raspberry Pi and 3 LEDs were used for the simulation.

In this document, research and implementation of the system will be described. There are 4 main points that will be dealt with as follows:

1. Understand the general picture of IoT and MQTT protocol
2. How to design the system from front-end to back-end and everything in between
3. Security of the network
4. Steps of implementing everything from software to hardware

2 IOT AUTHORIZATION AND WHY AWS IOT

2.1 From devices to IoT gateway

Smart devices continuously communicate with each other and with the cloud using different types of wireless communication protocols. While creating an adaptive IoT application, the communication process can also expose IoT security flaws and open channels for malicious agents and unintentional data leaks. To protect users, devices, and businesses, IoT devices must be secure and protected. The foundation of IoT security lies in controlling, managing, and establishing connections between devices. Correct protection helps keep data private, limits access to cloud devices and resources, provides secure methods of connecting to the cloud, and checks device usage. An IoT security strategy minimizes vulnerabilities through policies such as device identity management, encryption, and access control.

Security vulnerabilities are weaknesses that can be exploited to compromise the integrity or availability of IoT applications. IoT devices are inherently vulnerable. The IoT group is comprised of devices with diverse functions, long-standing, and geographically distributed. These characteristics, along with the growing number of devices, raise questions about how to address the security risks posed by IoT devices.

Each IoT machine needs a unique identity when connected to a gateway or central server to prevent malicious actors from gaining control of the system. This is done by associating the identity with a unique, cryptographic key. The X.509 certificate authenticates between AWS IoT system and client connection. The X.509 certificate provides a range of advantages over other methods of identification and authentication. Since the private key never leaves the system, X.509 certificate offers great client security over the systems. (What is an X.509 certificate?)

2.2 User authorization

User authorization and authentication framework, Open Authorization (OAuth) has become a popular and convenient way to securely access IoT devices. Users can log in to access their IoT devices using third-party accounts such as Amazon, Google or Facebook. Essentially OAuth allows users to securely authorize access to their data without having to enter a shared username and password with another third party. (Understand IoT Authentication and Authorization with “OAuth 2.0”)

OAuth 2 uses authorization and token-based authentication. Users are redirected to the proxy server when they log in to the application. With AWS Amplify, developer can integrate OAuth 2 to the front end with built-in components like sign up, log in, ... AWS Amplify is a set of tools and services that can be used concurrently or individually to help front-end web developers build fully powered, scalable, stackable applications, issued by AWS.

Amplify supports popular web frameworks including React (which will be used in this project), Angular, Vue. Amplify helps set up protected authentication streams easily and provides connectivity with AWS IoT so that users can easily control devices from the web app.

2.3 Why AWS

On the market, there are multiple of IoT service providers with different pros and cons. Three most popular IoT platforms now are Google Cloud, AWS and Microsoft Azure. AWS stands out due to its diverse services, besides that AWS provides services for all layers of security, including preventive security mechanisms, such as encryption and access control to device data, along with a service for continuous monitoring and configuration checking. AWS IoT is built on a proven, secure, and scalable cloud infrastructure that scales with billions of devices and trillions of messages. AWS IoT integrates other AWS services to developers can build a complete solution. (Top IoT Development Platform & Tools with Comparison)

AWS IoT Core is one of the most important service within AWS IoT. AWS IoT Core allows to connect IoT devices to the AWS cloud without provisioning or managing servers. AWS IoT Core can support reliably and securely process and deliver messages to AWS endpoints and to other devices. With AWS IoT Core, apps will have the ability to monitor and communicate with all of the devices, at all times, even when they're not connected.

AWS IoT Core also makes it easy to use AWS and Amazon services. Also, it will collect, process, analyse and manipulate the data generated by the connected devices without having to manage any infrastructure. AWS Amplify is also one of the reasons to choose using AWS for web app. With ready-to-use OAuth 2.0 and ecosystem powered by AWS, users can access to real-time device data from browser.

3 OVERVIEW OF IOT AND MQTT PROTOCOL

3.1 Overview of IoT network

For a very long time, the Internet of Things (IoT) has not been around. Since the early 1800s, however, there have been visions of machines interacting with one another. Since the telegraph was established in the 1830s and 1840s, computers have been providing direct communication. As a term, the Internet of Things was not publicly listed until 1999 by Kevin Ashton, the Executive Director if Auto-ID Labs at MIT. (A Brief History of the Internet of Things)

The network of IoT generally consists of connected devices, cloud services and apps.

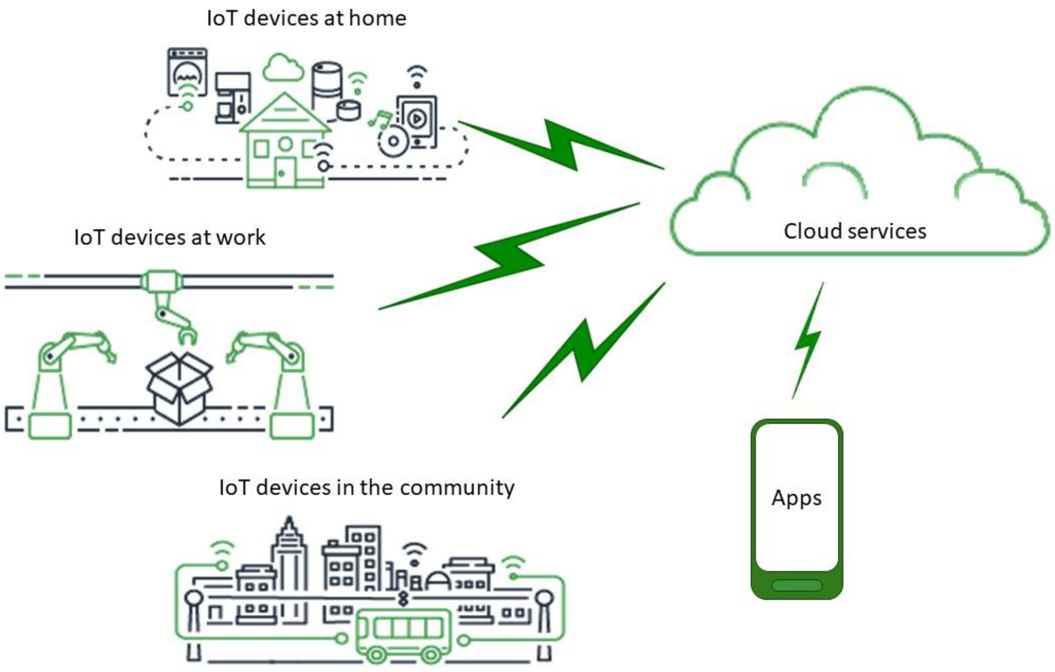


Figure 1 Overview of IoT network (How AWS IoT works - The IoT universe)

End users can access to IoT devices and the features provided by the cloud services through apps (desktop app, phone app or web app). In this thesis, a web app is built to demonstrate how to control connected devices.

Cloud services provide large-scale, distributed data storage and processing services connected to the Internet (AWS IoT core Developer guide – p. 4). AWS IoT is an example for the cloud service which is specific for IoT connection and management. AWS IoT supports MQTT, the main protocol applied for IoT.

3.2 MQTT protocol

In 1999, Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link) invented the MQTT protocol. To communicate with oil pipelines via satellite, they required a protocol for minimal battery loss and bandwidth. They specified some requirements for the protocol:

- Simple implementation
- Quality of Service data delivery
- Lightweight and bandwidth efficient
- Data agnostic

- Continuous session awareness

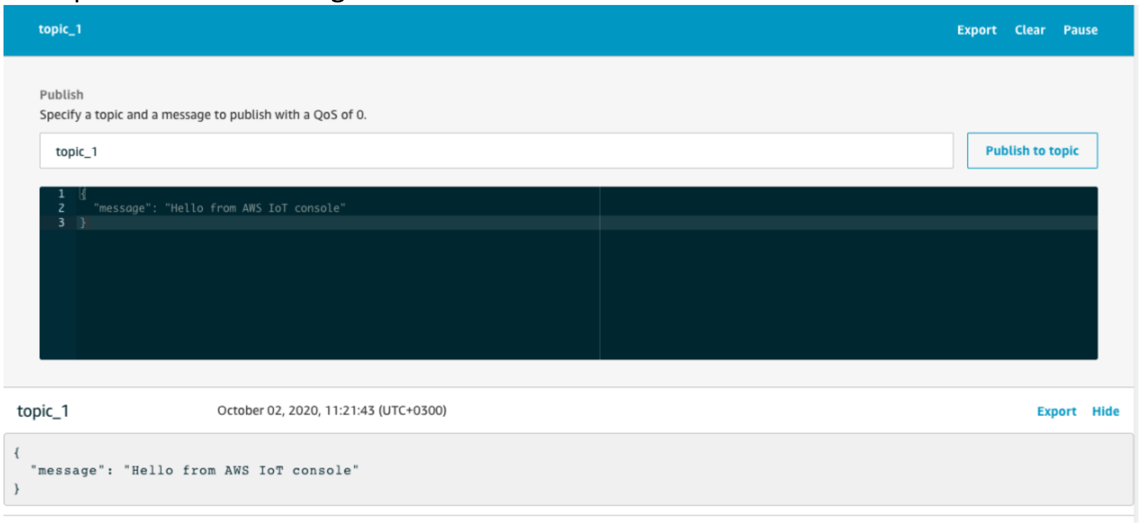
These priorities are also at the center of MQTT. However, the primary focus of the protocol has shifted from proprietary embedded systems to open IoT use cases.

“MQ” refers to the MQ Series, a product which IBM has built to support of MQ telemetry. In 1999, when Andy and Arlen developed their protocol, they named it after the IBM product, “MQ Telemetry Transport”. IBM used the protocol internally for the next ten years before they launched MQTT 3.1 in 2010 as a free version. (Introducing MQTT protocol)

MQTT (Message Queuing Telemetry Transport) is a publish/subscribe delivery protocol used for IOT devices with low bandwidths, a high reliability and the ability to be used in unstable networks.

In a system using the MQTT protocol, many station nodes (called MQTT clients – referred to as the clients) connect to an MQTT server (called broker). Each client will subscribe to several channels (topic). This sign up process is called “subscribe”. Each client will receive data when any other station sends data and the registered channel. When a client sends data to that channel, it is called “publish” (MQTT definition - https://mqtt.org/).

Example of demonstrating in AWS IoT console with MQTT:



Specify a topic and a message to publish with a QoS of 0.

topic_1

```
{
  "message": "Hello from AWS IoT console"
}
```

topic_1 October 02, 2020, 11:21:43 (UTC+0300)

```
{
  "message": "Hello from AWS IoT console"
}
```

Figure 2 AWS IoT Console with MQTT

In the example, a JSON format message is sent to topic “topic_1” and subscriber will receive the content of the message immediately.

3.3 Device shadow MQTT topics

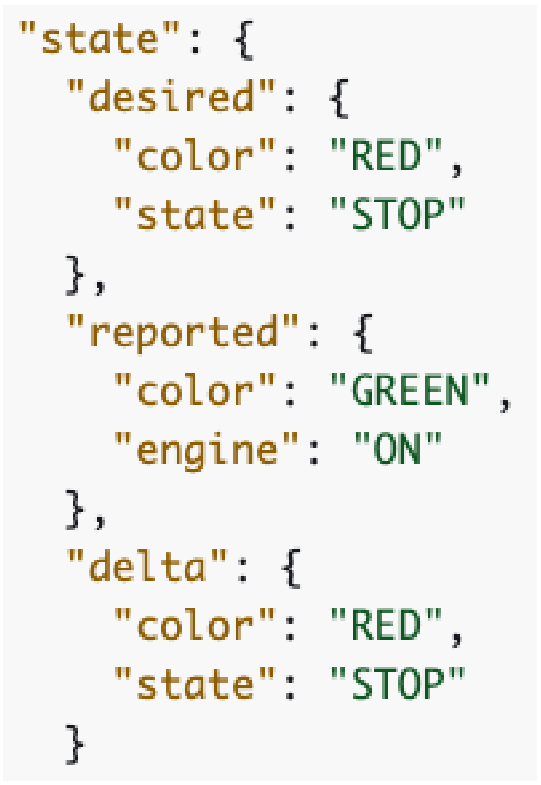
Device shadow is basically a "JSON State Document", which simply means device shadow is like an intermediary database used to store the status of devices represented by “things”, in JSON string format and architecture, rules have been built by AWS. User can interact with the device shadow using the MQTT.

The data stored in device shadow represents the state of the device, AWS IoT divided into two types of states to store: the “reported” state of the device and the “desired” state of the device.

The reported state of the device is the current state of the physical device that device shadow is describing, stored in the reported field of the JSON string. Therefore, only physical devices query the reported field and other applications will rely on the values in this field to retrieve the device's status in real time without fear of conflicts or confusion.

Desired status of device is the state that other applications want to send to the device, for example data from remote control applications by phone, website, virtual assistant, etc. This type of data will be stored separately in the desired field of the JSON string. The physical device will rely on the value in this field to change to a new state and then update the reported field (AWS IoT core Developer guide – p. 499).

Both reported and desired fields will be inside the state field. So, in general, the data in device shadow will have the following JSON string format:



```
"state": {
  "desired": {
    "color": "RED",
    "state": "STOP"
  },
  "reported": {
    "color": "GREEN",
    "engine": "ON"
  },
  "delta": {
    "color": "RED",
    "state": "STOP"
  }
```

Figure 3 Desired and reported state (AWS IoT Device Shadow service documents – Delta state)

Any topic starts with “$” sign is reserved by AWS IoT. Below is the table of shadow topics that are reserved by AWS IoT to allow client to get, update or delete device state information.

Topic	Client operations allowed	Description
Prefix/delete	Publish/Subscribe	To delete shadow
Prefix/delete/accepted	Subscribe	When a shadow is deleted, a message will be sent to this topic
Prefix/delete/rejected	Subscribe	When a delete shadow request is rejected, a message will be sent to this topic
Prefix/get	Publish/Subscribe	Publish an empty message to this topic to get a shadow
Prefix/get/accepted	Subscribe	When a request for a shadow is made successfully, a message will be sent to this topic
Prefix/get/rejected	Subscribe	When a request for a shadow is rejected, a message will be sent to this topic
Prefix/update	Publish/Subscribe	To update shadow
Prefix/update/accepted	Subscribe	When an update is made successfully, a message will be sent to this topic
Prefix/update/rejected	Subscribe	When an update is rejected, a message will be sent to this topic
Prefix/update/delta	Subscribe	When the reported and desired sections are different, a message with that different content will be sent to this topic
Prefix/update/documents	Subscribe	When an update is made successful, a state document will be sent to this topic

Figure 4 Shadow topics (AWS IoT MQTT reserved topics – Shadow topics)

Prefix = $aws/things/thingName/shadow

In AWS IoT, a thing means a device connected to the service. For the demonstration, a thing called “led” was created. Firstly, sending a message to $aws/things/led/shadow/update with the following JSON content:

```
{
  "state": {
    "reported": {
      "color": "red",
      "power": "off"
    }
  }
}
```

Two messages from AWS IoT service will be sent to $aws/things/led/shadow/update/accepted and $aws/things/led/shadow/update/documents to report that current device's state is: color is red and power is off.

Next, sending a new message to $aws/things/led/shadow/update:

```
{
  "state": {
    "desired": {
      "color": "yellow"
    }
  }
}
```

The color is desired from red to yellow, therefore there will be a message sent to topic $aws/things/led/shadow/update/delta from AWS IoT:

```
{
  "version": 22,
  "timestamp": 1601707724,
  "state": {
    "color": "yellow"
  },
  "metadata": {
    "color": {
      "timestamp": 1601707724
    }
  }
}
```

The delta shadow only receives the different information between reported state and desired state. Things can depend on this topic to update required states. After updating, thing need to send a message to $aws/things/led/shadow/update:

```
{
  "state": {
    "reported": {
      "color": "yellow"
    },
    "desired": null
  }
}
```

This message will report the new color and update the desired state to null since color is updated successfully.

In addition to the state, device shadow can also contain fields such as metadata, timestamp, version, ... to show specific information about the time, version, ... of device shadow, helping manage and verify easier. These fields will be automatically managed by AWS, user’s main task is to focus on designing the system and the state field (including reported and desired).

4 DESIGN SYSTEM

4.1 Common way

Nowadays, IoT is applied a lot in different fields such as agriculture, smart city, manufacturing industry, … People who maintain, operate the IoT system in these fields are engineers or managers. Normal workers (famers in this example) are not able to enter the network because of security, costs, … This system is 1-layer system (engineers/managers). Below is the example of a farm with IoT applied.

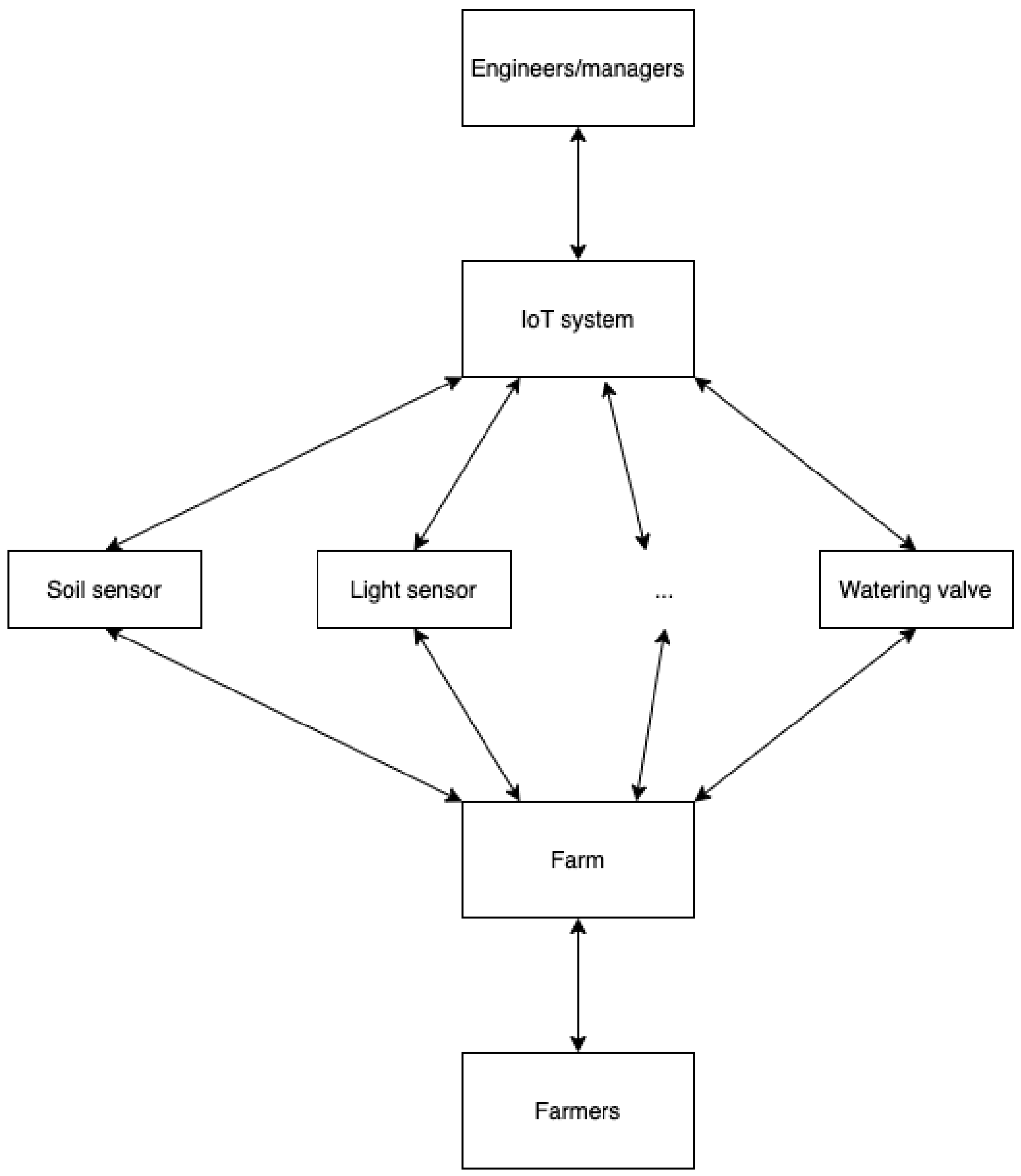


Figure 5 General IoT system of a farm

On a farm applied IoT technology, there are many different kinds of sensors, controllers, … which are connected to the IoT system managed by engineers. Farmers are the ones who work directly on the farm so they need to have access to the network in order to monitor and give decision on time. There are two ways:

- Farmers have the admin rights, meaning they have their own account which has the same rights as admin. This way takes risk at giving too many accesses to the system.
- Second way is if farmer want to make a decision they will contact to managers / engineers. The disadvantage is time consuming; late decision may lead to bad consequence.

With authorized IoT, after farmers log in the application with their Google account, they can request to have access to the system and admin will be able to approve or deny. This means admins will be in between farmers and the network, so if there are any problems with the account or the system, admins can immediately suspend the farmer account.

4.2 With Administration in between

In public places, IoT authorization play an even more important role. For example, in university building, if IoT is applied to the devices in the building, not only professors or lecturers can control devices but students also should be able to access the system with limited rights. This is a 2-layer system (Admins -> users). Below is a diagram example of an IoT system applied in the university building.

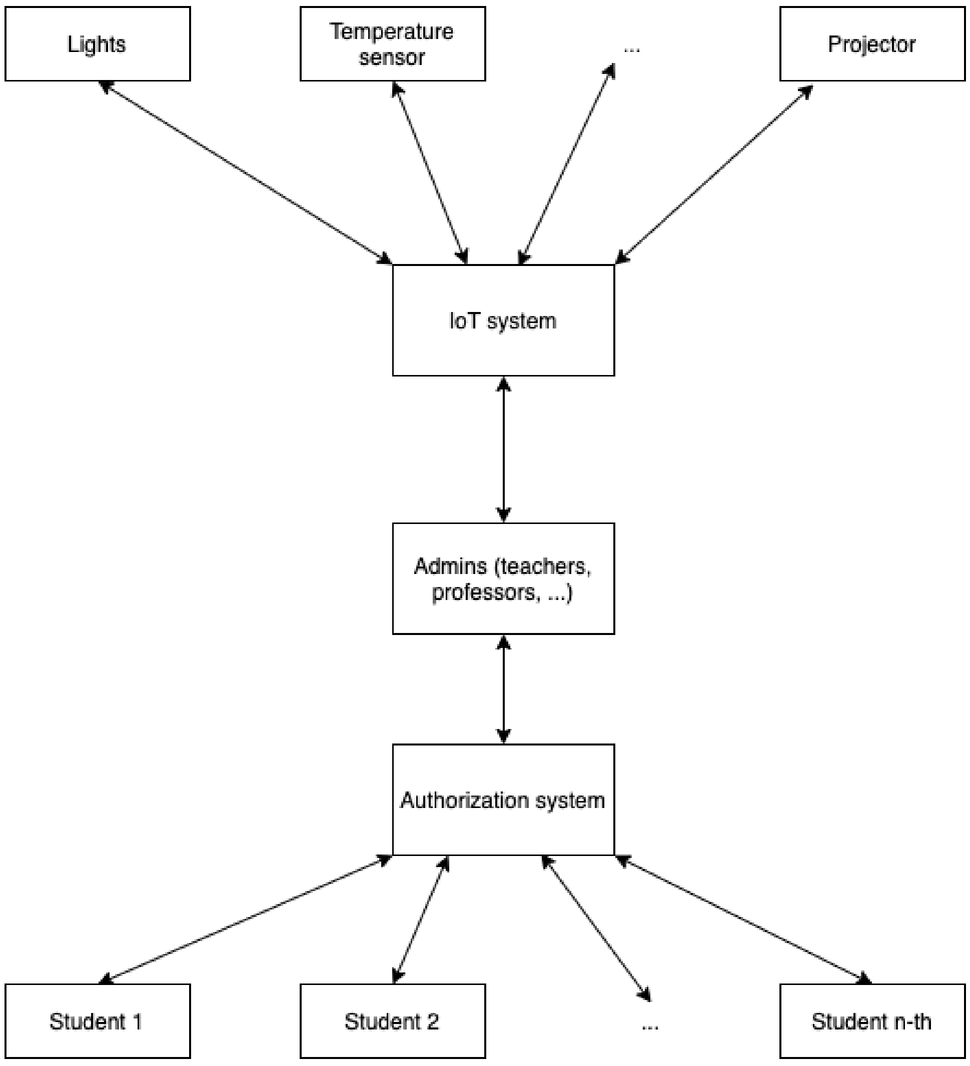
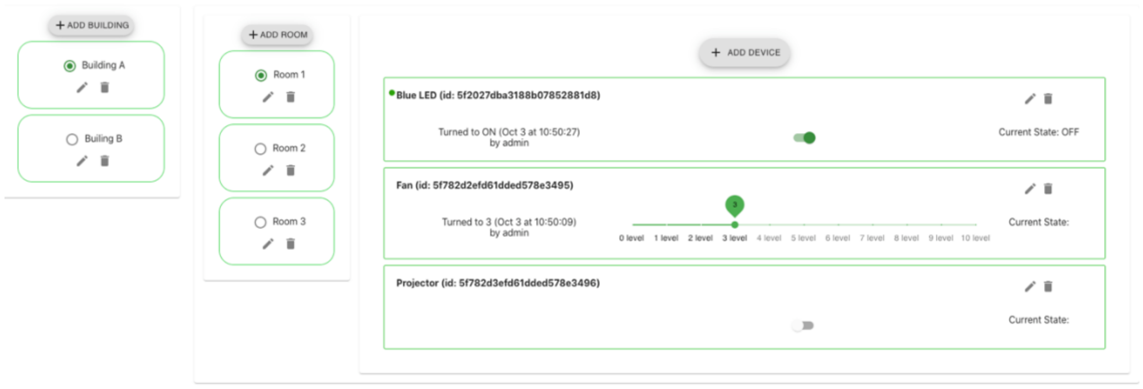


Figure 6 IoT system in a University building

With this 2-layer system, students can have limited rights like turning the lights on or off. At the same time admin can suspend the users' account because they are in the middle of the layer.

4.3 Manage devices in Hamk buildings

For the project, a system of buildings and connected devices was built to manage easier.



Blue LED (id: 5f2027dba3188b07852881d8)
Turned to ON (Oct 3 at 10:50:27) by admin
Current State: OFF

Fan (id: 5f782d2efd61dded578e3495)
Turned to 3 (Oct 3 at 10:50:09) by admin
Current State:

Projector (id: 5f782d3efd61dded578e3496)
Current State:

Figure 7 Buildings manager

In a building there are rooms, in each room there are devices with unique id.

There are two kinds of users: admins (e.g. professor, teacher, facility maintainer) and normal users (e.g. students).

Role	Role description
Admin	• Add/edit/delete building • Add/edit/delete room • Add/edit/delete device • Control devices • Accept/reject user from controlling devices
Users	• Request admin to control devices • Able to control devices if admin accepted

Figure 8 Admin and users' role description

After logging in with Google account, normal users will send a request to control devices, admin will know who is requesting by their email address. Admin has the right to accept or decline the request. If the request is accepted, users can control devices.

During the time while user has the permission to control devices, admin can still decline the user's permission anytime. In case user get rejected from admin, he/she can request again.

4.4 How system works

To connect things to AWS IoT, a Raspberry Pi is chosen. This diagram shows how the system works together:

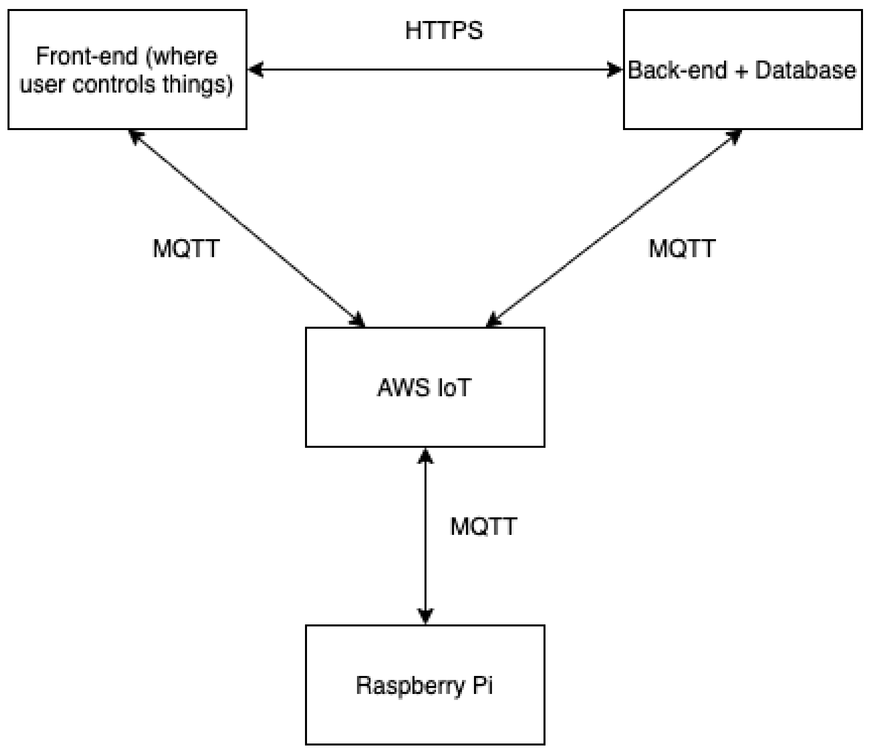


Figure 9 How system connects

For website front-end, Reactjs is chosen. This framework for a client-side web page is a suitable choice to help the user to interact with devices. For server (back-end), Nodejs and Mongo Database are used. As an asynchronous event-driven JavaScript runtime, Nodejs is designed to build scalable network application. Mongo Database stores data in flexible, JSON-like documents. Below is the diagram which presents the frameworks and technologies chosen for the project:

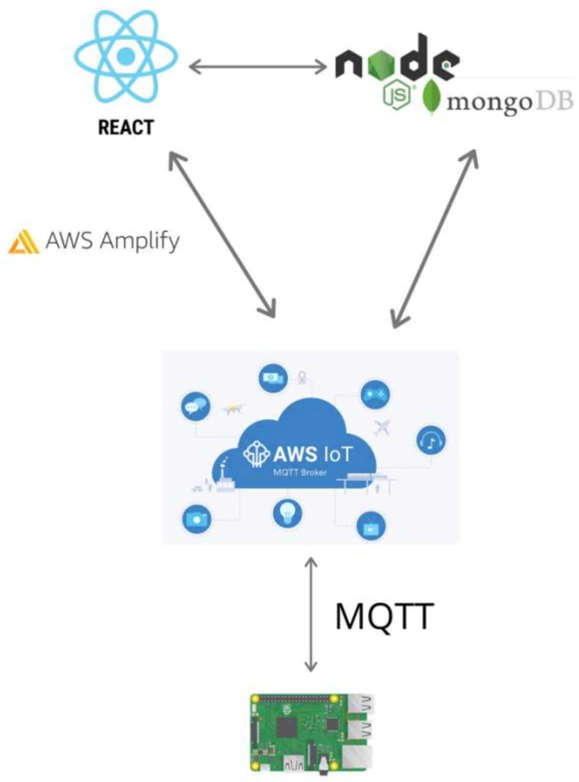


Figure 10 Frameworks and technologies chosen

In the picture, AWS Amplify displays the connection between web front-end Reactjs and AWS IoT.

4.5 AWS Amplify and Google Authentication plugin

The AWS Amplify Authentication modules support APIs to developers who want to construct apps with real-world production-ready user authentication. The Amplify system uses Amazon Cognito as the key authentication provider. Amazon Cognito is a robust directory service for users that manages user registration, authentication, recovery of accounts and other activities.

Amplify interfaces with Cognito to store user data provided by third-party authorization such as Apple, Google, Facebook as well as identity provider and OpenId Connect.

Amplify makes it easy to authenticate users, securely store data and user metadata, authorize selective access to data, analyse application metrics and execute server-side code. (Nader Dabit - The complete guide to user authentication with the Amplify Framework)

For the thesis's purpose, AWS Amplify Authorization with Google is applied to the application.

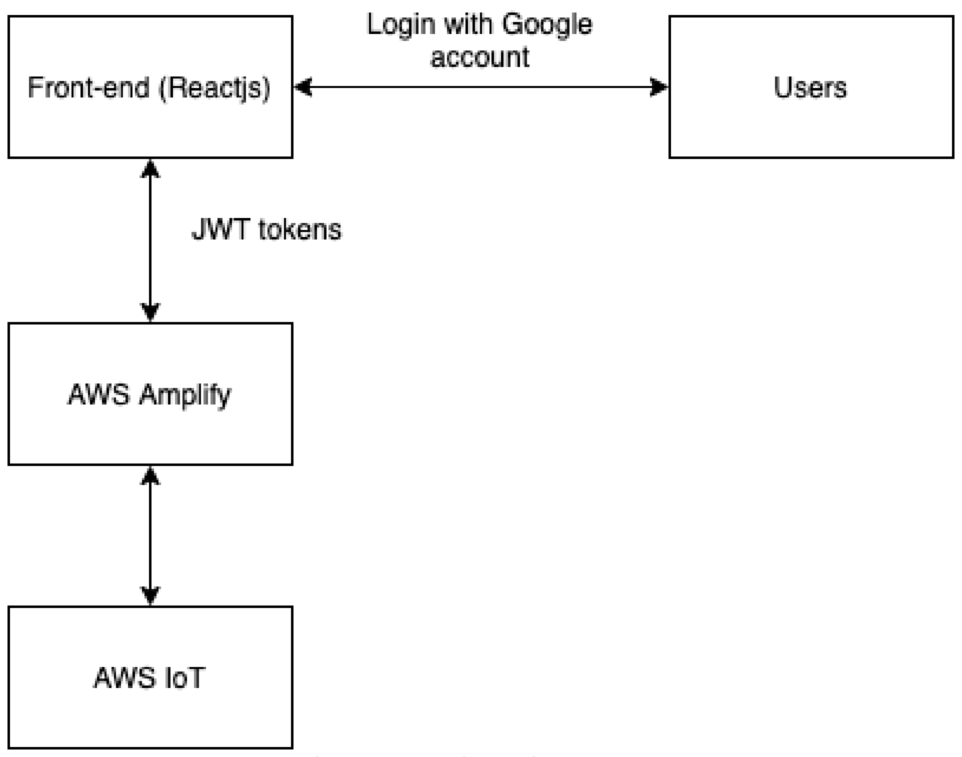


Figure 11 Authentication with Google

After user logins with Google account, AWS amplify will return JWT (JSON Web Token). User uses this token to verify and access to IoT service. (AWS Amplify Authentication Concept)

4.6 Domain name and server

To run the project in real life, a domain name was registered on Amazon Route 53: https://hamkiot.nam-huynh.com/. And server is running on Amazon Elastic Compute Cloud (AWS EC2).

Amazon Route 53 is a cloud-based, highly scalable, Domain Name System (DNS) web service. This service is designed to provide developers and businesses with an extremely reliable and cost-effective way of routing end users to Internet applications by converting names like www.example.com to a numeric IP address like 192.0.2.1 that computers use to connect to each other. Amazon Route 53 also fully complies with IPv6. (Amazon Route 53 Developer Guide, p. 2)

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure and flexible computing power in the cloud. This service is designed to make it easier for developers to use cloud computing at the web scale.

Amazon EC2's simple web service interface allows user to gain and configure capacity with minimal collisions. This service gives full control of computing resources and helps running on Amazon's proven computing environment. Amazon EC2 reduces the time it takes to retrieve and start new server versions to minutes, allowing user to quickly increase or decrease the capacity scale according to changes in computing requirements. Amazon EC2 changes the economics of computing by allowing user to pay for only the energy user actually use. Amazon EC2 provides developers with many tools to build applications that are more resistant to bugs and do not let them fall into common error situations. (AWS EC2 User's Guide – What is Amazon EC2?)

5 SECURITY

5.1 Thing (Raspberry Pi)

When creating a thing on AWS IoT console, a certificate, a public key and a private key will be created for the client to download. The public key and private key can only be retrieved one time. Things with these certificate and keys will be controlled by client.

The Raspberry Pi is imported with these credentials, make it to be controlled securely by the right client.

5.2 Normal users

Normal users can control things with the acceptance from admin. Every user after logging in will have a Cognito Identity Id (a unique Id from AWS) which will be sent to admin if the user requests to control devices. If admin accepts, the Id will be attached to the predefined policy (a policy allows some abilities such as controlling only certain things, getting data from devices without controlling them). For this specific project, the policy will allow user with full control to every device. Below is the flow chart that demonstrates the flow of requesting from user:

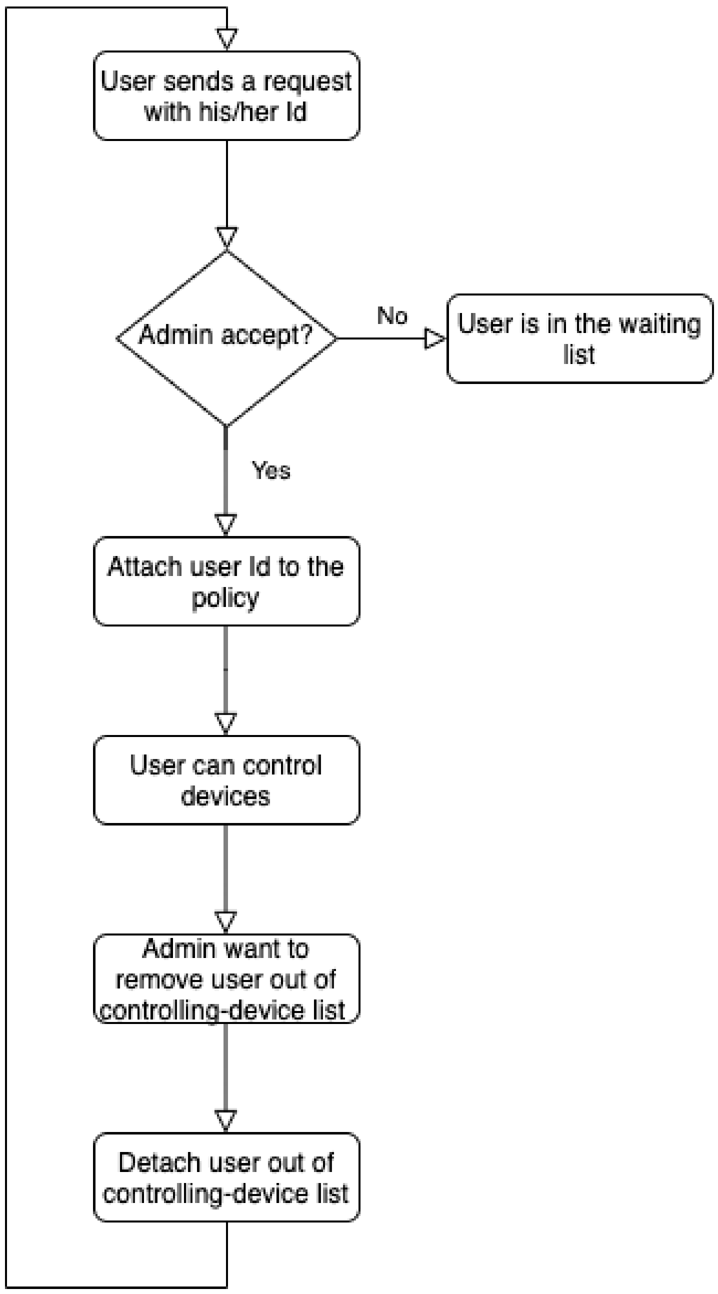

User sends a request with his/her Id
Admin accept?
No
User is in the waiting list
Yes
Attach user Id to the policy
User can control devices
Admin want to remove user out of controlling-device list
Detach user out of controlling-device list

Figure 12 Request from user

5.3 SSL certificate

After running the Nodejs server on AWS EC2, front-end web browser transfers data with server via HTTP (Hypertext Transfer Protocol). For the security, SSL certificate need to be obtained and applied so that data transferred via HTTPS.

SSL stands for Secure Socket Layer is a protocol that allows to communicate securely over the network.

The connection between a web browser to any point on the Internet goes through many independent systems without any protection for the information on the line. No one, either the user or the web server, has any control over the path of the data or can control whether someone enters information on the link.
To protect confidential information on the Internet or any TCP / IP network, SSL has combined the following factors to establish a secure transaction:

- Authentication: ensure the authenticity of the page on the other end of the connection. Also, web pages need to check the authenticity of the user.
- Encryption: ensures information cannot be accessed by third parties. To eliminate eavesdropping on sensitive information when it is transmitted over the Internet, the data must be encrypted so that it cannot be read by anyone other than the sender and receiver.
- Data integrity: ensuring information is not misleading and it must accurately represent the original information sent. (What is an SSL certificate?)

6 IMPLEMENTATION

6.1 Devices

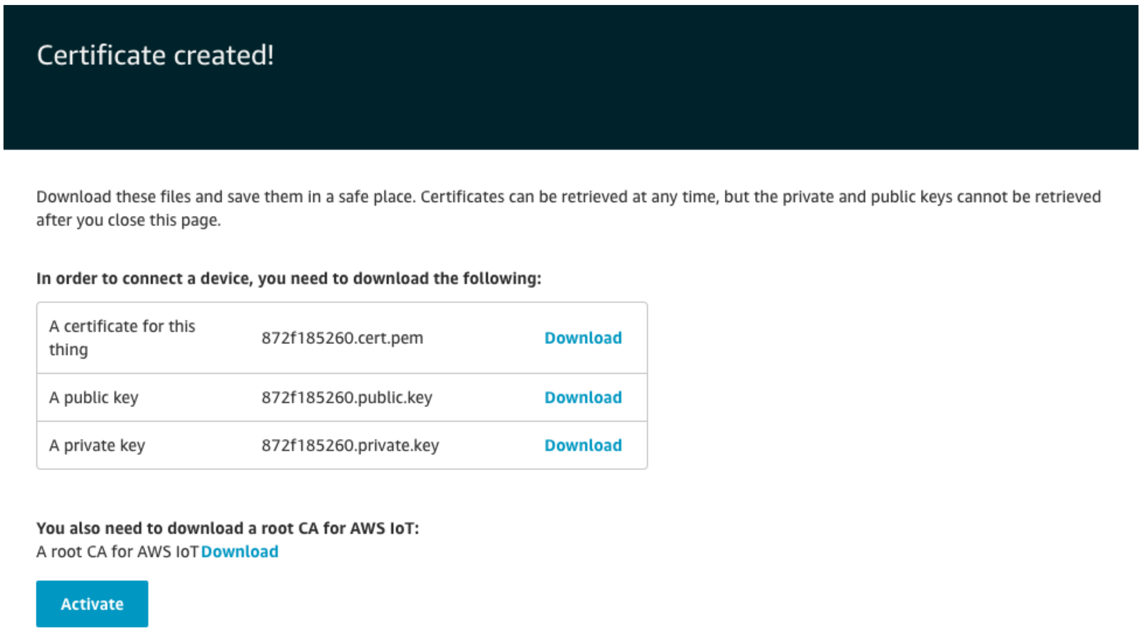
For the thesis's demo, there are some devices needed:

- Raspberry Pi
- 3 LEDs (red, blue, yellow)
- 3 resistors
- Electrical wires

These devices will reflect how things (LEDs) connect to AWS IOT and be controlled in real time.

6.2 Setup AWS IoT

Creating a thing on AWS IoT, user will be able to download a certificate, a public key, a private key and root CA (certificate authority). These certificates and keys will be needed later for the Raspberry Pi.



Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	872f185260.cert.pem	Download
A public key	872f185260.public.key	Download
A private key	872f185260.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT Download

Figure 13 Certificates and keys from AWS

Also attaching a policy to the certificate, this means thing (Raspberry Pi) with this certificate will be allowed to do some defined tasks. (Connect Raspberry Pi to AWS IoT service)

6.3 Setup Raspberry Pi

Connect 3 LEDs to the Raspberry Pi and create 3 programs for each LED to connect to AWS IoT. In each program, declaring the certificates and keys from AWS IoT to be able to connect with AWS IoT service.

For controlling the LED, use the following topics:

Topic	Operation	Description
$aws/things/thingId/shadow/update/delta	Subscribe	Get the desired different data to update LED
$aws/things/thingId/shadow/update	Publish	Publish reported data after updating the LED

Figure 14 Shadow topics for Raspberry Pi

6.4 Server-side code

For the server, a set of RESTful APIs were created. RESTful API is a standard used in designing API for web applications to facilitate the management of resources. It focuses on system resources (text files, images, audio, video or dynamic data, ...), including resource states that are formatted and transmitted over HTTP.

The APIs in server are mainly for creating, updating and deleting buildings, rooms and devices. Some APIs are for saving new users to database, creating requests.

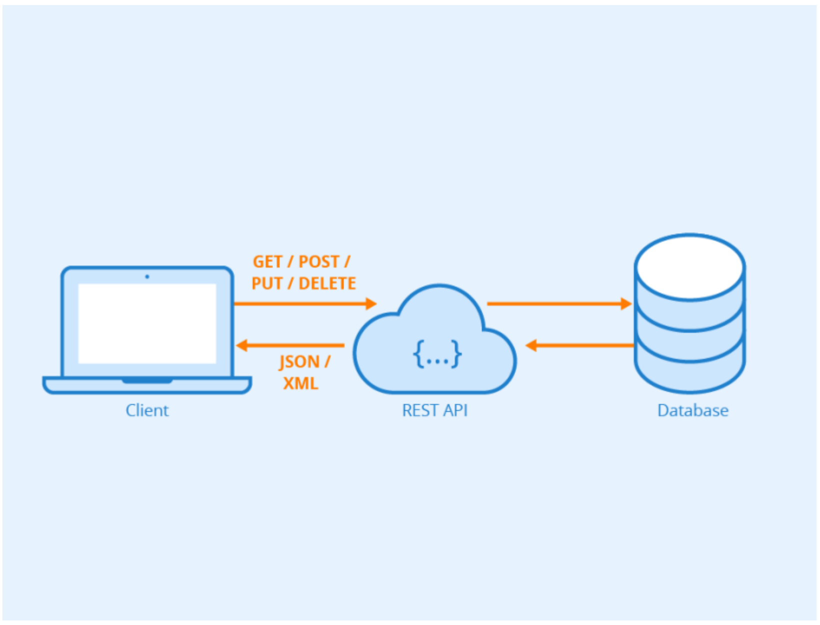


Figure 15 Rest API (Rest APIs: Definition, Working, Benefits, and Design Principals)

Moreover, server has a function of checking the thing is connected or disconnected to AWS IoT service. AWS IoT publishes a message to the following topic if:

- $aws/events/presence/connected/thingId: the thing is connected
- $aws/events/presence/disconnected/thingId: the thing is disconnected

After receiving a message in any one of two cases, server will publish a message to $aws/things/thingId/shadow/update with the following format:

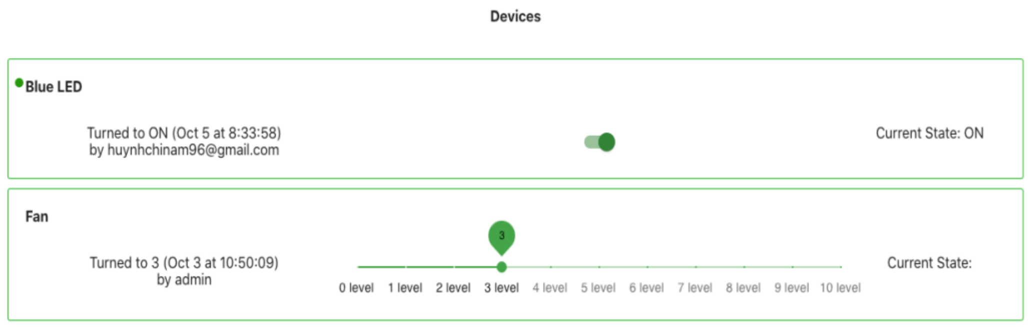
```
{
 "state": {
   "reported": {
       connected: true/false (true if device connected and vice versa)
   }
  }
}
```

6.5 Setup front-end code

AWS Amplify will be applied to front-end so that user can log in with Google account. Besides that, front-end in web browser also subscribe and publish to some AWS IoT topics.

On the web page, the following data will be shown on the device:

Information	Controller	State	Time	Current State
Value	Admin/user email	ON/OFF (for digital device)	Time at the latest that the device state was updated	Connected/disconnected/state
		Level (for analog device)		


Devices

Blue LED

Turned to ON (Oct 5 at 8:33:58) by huynhchinam96@gmail.com

Current State: ON

Fan

Turned to 3 (Oct 3 at 10:50:09) by admin

Current State:

Figure 16 Devices data shown from webpage

To be able to receive updated status and report new states of the devices, front end needs to publish and subscribe the following AWS IoT shadow topics:

Topic	Operation	Description
$aws/things/thingId/shadow/update	Publish	Publish the new states of the device and user email address of the controller
$aws/things/thingId/shadow/update/documents	Subscribe	Subscribe to new states of device
$aws/things/thingId/shadow/update/get/accepted	Subscribe	Get states of the devices after logging in
$aws/things/thingId/shadow/update/get	Publish	Publish an empty message to this topic to receive message from $aws/things/thingId/sh

		adow/update/get/accepted

Figure 17 Shadow topics for front-end

Multiple users can publish and subscribe to shadow topics concurrently.

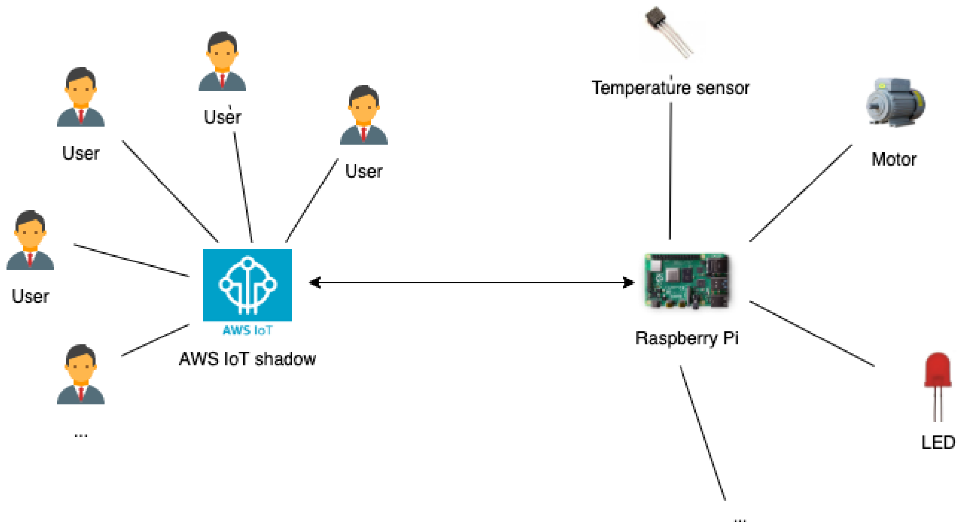
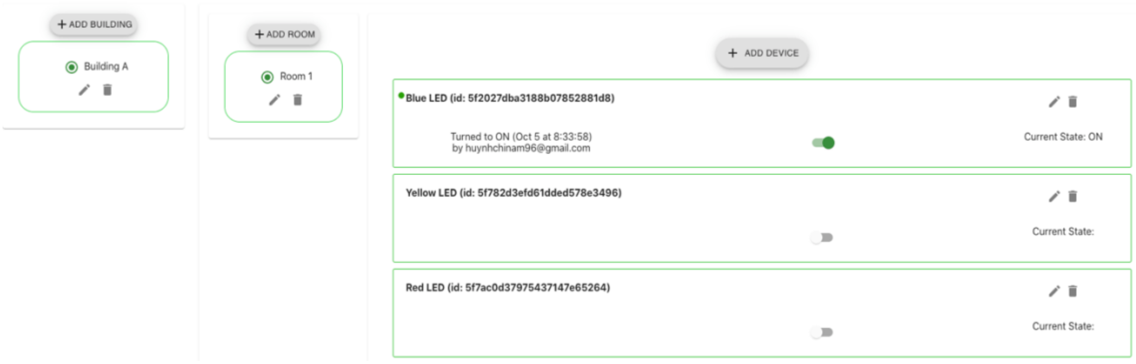


Figure 18 Multiple users control concurrently

6.6 Working flows of application

Step 1/ Go to https://hamkiot.nam-huynh.com/. Admin logs in by Google account. AWS Amplify will send a unique Cognito Username (e.g. Google_109485232010340394385023). Saving this Cognito Username in database so next time server can confirm the admin.

Step 2/ Creating a building, in the building creating a room and, in the room, creating devices. For the project, creating 3 digital devices for 3 LEDs (red, blue, yellow). After creating, Mongo Database will automatically send back 3 unique ids for each LED.



Blue LED (id: 5f2027dba3188b07852881d8)
Turned to ON (Oct 5 at 8:33:58)
by huynhchinam96@gmail.com
Current State: ON

Yellow LED (id: 5f782d3efd61dded578e3496)
Current State:

Red LED (id: 5f7ac0d37975437147e65264)
Current State:

Figure 19 Unique Ids sent back from MongoDb

Step 3/ Log in as a normal user and request to control devices. Database will save the following information of user:

Field	Type	Description
isAccepted	Boolean	True if admin accepted
isRequesting	Boolean	True if user requested to control devices
cognitoUsername	String	This data is from AWS service to check if the client is admin or normal user
Email	String	"Name" of the user, showing the list of users to admin and who controlled the device
cognitoIdentityId	String	This data is from AWS service to attach/detach user from the controlling-device list

Figure 20 Information saved in database after user requested

Step 4/ If admin accepted, server will attach the "cognitoIdentityId" to the predefined policy to allow user to control device and set "isAccecpted" to true and "isRequesting" to false. Front end web page will show the interface for user to interact with the devices.

Step 5/ After logging in, an empty message will be sent to $aws/things/blueLEDId/shadow/get so that user can get the current state of the device from topic $aws/things/blueLEDId/shadow/get/accepted:

```
{
  "state": {
    "reported": {
      "connected": true,
      "state": "OFF",
      "controller": "admin"
    }
  },
  "metadata": {
    "reported": {
      "connected": {
        "timestamp": 1601876017
      },
      "state": {
        "timestamp": 1601882830
      },
      "controller": {
        "timestamp": 1601882091
      }
    }
  },
  "version": 790,
  "timestamp": 1601884777
}
```

Reactjs will update the interface according to the information from reported message. In this message, the blue LED is connected and currently turned off by admin from timestamp 1601882830.

Step 6/ Turn the Blue LED on from the web page interface. A message will be sent to $aws/things/blueLEDId/shadow/update:

```
{
  state: {
    desired: {
      state: "ON",
      controller: "user1@gmail.com"
    }
}
```

Step 7/ Raspberry Pi is now subscribing to topic $aws/things/blueLEDId/shadow/update/delta and it receives the message:

```
{
  "version": 787,
  "timestamp": 1601882091,
  "state": {
    "state": "ON",
    "controller": "user1@gmail.com"
  },
  "metadata": {
    "state": {
      "timestamp": 1601882091
    },
    "controller": {
      "timestamp": 1601882091
    }
  }
}
```

The Raspberry Pi will look at this message and update the blue LED by the state in the message. If the blue LED was turned on successfully, the Raspberry Pi will send a message to $aws/things/blueLEDId/shadow/update:

```
{
  "state": {
    "reported": {
      "state": "ON"
    },
    "desired": NULL
  }
}
```

This message notifies that the blue LED was updated successfully. Currently its state is ON and no desired state from users. All the users and admin are subscribing to topic $aws/things/blueLEDId/shadow/update/documents will receive the message:

```
{
 "current": {
  "state": {
   "reported": {
    "connected": true,
    "state": "ON",
    "controller": "admin"
   }
  },
  "metadata": {
   "reported": {
    "connected": {
     "timestamp": 1601876017
    },
    "state": {
     "timestamp": 1601882830
    },
    "controller": {
     "timestamp": 1601882091
    }
   }
  },
  "version": 790
 },
 "timestamp": 1601882830
}
```

The message shows information of the device’s current state and Reactjs will update this information to users and admin on the web page interface.

Step 8/ If admin wants to remove the user out of the control-device list, server will detach user from the policy and set “isAccepted” field in the database to false. Reactjs will also change the interface so that user is not able to interact with the devices.

7 CONCLUSION

The main purpose of this project is that a user can control the device easily only by being through Google authentication and admin authorization, which was achieved by combining ReactJs in the frontend, NodeJs in the backend and AWS services in between.

Moreover, there are still lots of functions that can be developed and applied, for example, alert user if temperature is too high/low, set timer for the devices or devices can only be controlled by users who reserve them. Conclusively, by applying technologies smartly and securely, life and work will become easier and more productive.

REFERENCES

What is an X.509 certificate? https://www.ssl.com/faqs/what-is-an-x-509-certificate/

Understand IoT Authentication and Authorization with “OAuth 2.0” https://prodea.com/2017/07/13/understand-iot-authentication-authorization-need-know-oauth-2-0/

Top IoT Development Platform & Tools with Comparison https://www.intuz.com/blog/top-iot-development-platforms-and-tools

Design and development of IoT-based latency-optimized augmented reality framework in home automation and telemetry for smart life style (Proposed system architecture) https://link.springer.com/epdf/10.1007/s40860-020-00106-1?sharing_token=t1JK1IUPAhlS1PxT9EkuS_e4RwlQNchNByi7wbcMAY5G9dP2lMA9JAGvRNXXF-K0kLBiJluZ_QBObpd7jBGUHnv97E3oeS3mPGiCIsLj7K79h30_7WjKZKI5PwwgiaIyB1aq2ZPx1ZcecNSKXEVwWe1y1-H8s3IgTMcLI5qTI24%3D

Nader Dabit - The complete guide to user authentication with the Amplify Framework https://dev.to/dabit3/the-complete-guide-to-user-authentication-with-the-amplify-framework-2inh

A Brief History of the Internet of Things https://www.dataversity.net/brief-history-internet-things

Introducing MQTT protocol https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/

AWS EC2 User’s Guide https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

MQTT definition http://mqtt.org

AWS Amplify Authentication Concept https://docs.amplify.aws/lib/auth/overview/q/platform/js#social-provider-federation

Connect Raspberry Pi to AWS IoT service https://docs.aws.amazon.com/iot/latest/developerguide/connecting-to-existing-device.html

How AWS IoT works https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html

AWS IoT MQTT reserved topics
https://docs.aws.amazon.com/iot/latest/developerguide/reserved-topics.html

Rest APIs: Definition, Working, Benefits, and Design Principals
https://www.astera.com/type/blog/rest-api-definition/

AWS IoT Device Shadow service documents
https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-document.html

Amazon Route 53 Developer Guide
https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/route53-dg.pdf#Welcome

What is an SSL certificate? https://www.cloudflare.com/learning/ssl/what-is-an-ssl-certificate/

APPENDICES

Appendix 1. Back-end code Github link https://github.com/namhuynh96/v2-iot-ts-backend

Appendix 2. Front-end code Github link https://github.com/namhuynh96/v2-iot-ts-frontend