

การพัฒนาแอปพลิเคชันเพื่อจัดการระบบฟาร์มอัจฉริยะ

APPLICATION DEVELOPMENT FOR SMART FARM MANAGEMENT



กาจบัณฑิต ช่วยศรีนวล
Kajbundit chuaisrinual

ภคิน ธาดาทังสกุล
Pakin Tadatangsakul

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต

สาขาวิชาวิศวกรรมสารสนเทศ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2565

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

APPLICATION DEVELOPMENT FOR SMART FARM MANAGEMENT



Kajbundit chuaisrinual

Pakin Tadatangsakul

THIS THESIS IS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
BACHELOR OF ENGINEERING IN INFORMATION ENGINEERING
SCHOOL OF ENGINEERING
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
ACADEMIC YEAR 2022

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อปริญญานิพนธ์

การพัฒนาแอปพลิเคชันเพื่อจัดการระบบฟาร์มอัจฉริยะ

รายนามนักศึกษา

นายกางบัณฑิต ช่วยศรีนวล

รหัสนักศึกษา 62010057

นายภคิน ธาดาทั้งสกุล

รหัสนักศึกษา 62010669

ปริญญา

วิศวกรรมศาสตรบัณฑิต

สาขาวิชา

วิศวกรรมสารสนเทศ

พ.ศ.

2565

อาจารย์ที่ปรึกษาปริญญานิพนธ์

ผศ.ไพศาล สิริธียาสากุล

ดร.ธนวิญญ์อนุวงศ์พินิจ

ปริญญานิพนธ์ฉบับนี้ได้รับการอนุมัติให้เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตร
วิศวกรรมศาสตรบัณฑิต คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร
ลาดกระบัง

(ผศ.ไพศาล สิริธียาสากุล)

(ดร.ธนวิญญ์อนุวงศ์พินิจ)

อาจารย์ผู้ควบคุมปริญญานิพนธ์

หัวข้อปริญญานิพนธ์	การพัฒนาแอปพลิเคชันเพื่อจัดการระบบฟาร์มอัจฉริยะ	
รายนามนักศึกษา	นายกจบัณฑิต ช่วยศรีนวล	รหัสนักศึกษา 62010057
	นายภคิน ธาดาทังสกุล	รหัสนักศึกษา 62010669
ปริญญา	วิศวกรรมศาสตรบัณฑิต	
สาขาวิชา	วิศวกรรมสารสนเทศ	
พ.ศ.	2565	
อาจารย์ที่ปรึกษาปริญญานิพนธ์	ผศ.ไพศาล สิทธิโยภาสกุล	
	ดร.ธนวิษณุณรงค์พินิจ	

บทคัดย่อ

โครงการเรื่อง การพัฒนาแอปพลิเคชันเพื่อจัดการระบบฟาร์มอัจฉริยะ มีวัตถุประสงค์เพื่อการพัฒนาแอปพลิเคชันที่มีความสามารถในการช่วยเหลือในด้านการติดตามการวัดผลของค่าต่างๆ ที่เป็นปัจจัยสำคัญในการส่งผลกระทบต่อฟาร์ม เช่น อุณหภูมิ ความชื้น เป็นต้น ซึ่งสามารถนำมาแสดงผลให้ผู้ใช้ได้ทั้งในรูปแบบต่างๆ เพื่อสามารถที่จะนำไปวิเคราะห์และแก้ไขจุดที่เป็นปัญหาภายในฟาร์มต่อไปได้

ผู้จัดทำได้ทำการพัฒนาแอปพลิเคชันโดยการใช้ชุดพัฒนา Flutter บนระบบปฏิบัติการแอนดรอยด์ และใช้คลาวด์ของ AWS โดยปัจจุบันมีความสามารถในแอปพลิเคชัน ได้แก่ การเข้าสู่ระบบ และการลงทะเบียนผ่านชื่อผู้ใช้และรหัสผ่าน การแสดงผลค่าจากอุปกรณ์แบบตามเวลาจริง ทั้งในรูปแบบของตัวเลข กราฟและไฟล์รายงาน การสั่งการเบื้องต้นที่เกี่ยวกับอุปกรณ์ เช่น การปิด-เปิดการรับค่าจากฮาร์ดแวร์ โดยการรับส่งค่าจากอุปกรณ์จะผ่านโปรโตคอล MQTT กับ HTTP/REST นอกจากนี้ ผู้จัดทำได้พัฒนาฟังก์ชันการทำงานแบบไร้เซิร์ฟเวอร์บน AWS เพื่อรองรับการทำงานตามเหตุการณ์ต่างๆ เช่น เมื่อมีค่าส่งมาจากอุปกรณ์ หรือ การดึงข้อมูลในฐานข้อมูลจากแอปพลิเคชัน ซึ่งสามารถรองรับการทำงานได้แบบต่อเนื่อง และสามารถทำงานได้แบบอัตโนมัติ ผลที่ได้จากการดำเนินการ คือ แอปพลิเคชันบนระบบแอนดรอยด์ที่มีความสามารถในการแสดงผลที่ค่อนข้างเร็วตามเวลาจริง ซึ่งอาจมีการชะลอบ้างในบางครั้งเนื่องด้วยจำนวนของข้อมูล รวมถึงสามารถควบคุมอุปกรณ์จากการสั่งการในแอปพลิเคชันได้และฟังก์ชันการทำงานแบบไร้เซิร์ฟเวอร์ที่สามารถจัดการข้อมูลจากองค์ประกอบได้อย่างรวดเร็วและเชื่อมต่อบริการอื่นๆ ของ AWS ได้

Thesis Title	Application development for smart farm management	
Student	Mr. Kajbundit chuaisrinual	Student ID. 62010057
	Mr. Pakin Tadatangsakul	Student ID. 62010669
Degree	Bachelor of Engineering	
Program	Information Engineering	
Year	2022	
Thesis Advisor	Asst.Prof.Paisan Sithiyopasakul	
	Dr.Thanavit Anuwongpinit	

ABSTRACT

The purpose of “Application Development for smart farm management” is to develop an application that can help in monitoring value measurement from crucial factors such as temperature, humidity, etc. which can display to the user as many forms for analyzing and fixing the critical point in the farm.

We developed an application using the Flutter development tool on the Android operating system and using the AWS cloud platform. In the current version, the features of the application are login and register through username and password, displaying the value in real-time in numbers, graphs, and report files, and basic hardware interaction such as turning the switch on-off to receive value from hardware or not to receive which was sent through MQTT protocol and HTTP/REST. In addition, we develop a serverless function on AWS to support such events as when a value is sent from hardware or fetching data in the database from the application, which supports continuous calls and automatic functions. The result from the proceeding were the android application that capable of nearly real-time value display which may be delayed sometimes by the amount of data and also can control the device from the application and the serverless functions that able to quickly manage data from components and connect to the other AWS services.

กิตติกรรมประกาศ

โครงการ การพัฒนาแอปพลิเคชันเพื่อจัดการระบบฟาร์มอัจฉริยะ สำเร็จได้ด้วยดี เพราะ
 เนื่องจากได้รับความกรุณาจาก ผศ.ไพศาล สิทธิโยภาสกุล ซึ่งเป็นอาจารย์ที่ปรึกษาที่ได้ให้คำแนะนำ
 ในเรื่องฮาร์ดแวร์ที่ใช้ในการทดลอง และดร.ธนวิษณุ อนุวงศ์พินิจ ซึ่งเป็นอาจารย์ที่ปรึกษา ที่ได้ให้
 คำแนะนำในด้านโครงสร้าง ภาพรวม และข้อเสนอแนะในการจัดทำโครงการจนสำเร็จไปได้ด้วยดี
 ด้วยเหตุนี้ผู้จัดทำจึงขอขอบพระคุณทุกท่านเป็นอย่างสูงมา ณ ที่นี้ด้วย

คณะผู้จัดทำหวังว่าปริญญาพนธ์ฉบับนี้จะมีประโยชน์ต่อผู้ที่สนใจไม่มากนักน้อยหากมี
 ข้อผิดพลาดประการใดขออภัยมา ณ ที่นี้ด้วย

กาจบัณฑิต ช่วยศรีนวล

ภคิน ธาดาทังสกุล



สารบัญ

บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	III
กิตติกรรมประกาศ	V
สารบัญ	IV
สารบัญตาราง	VII
สารบัญรูป	VIII
บทที่ 1 บทนำ	1
1.1 ความเป็นมาและความสำคัญของปัญหา	1
1.2 วัตถุประสงค์	1
1.3 ขอบเขตในการดำเนินงาน	1
1.4 ระยะเวลาการทำงาน	2
1.5 ประโยชน์ที่คาดว่าจะได้รับ	2
บทที่ 2 ทฤษฎีพื้นฐานที่ใช้	3
2.1 การพัฒนาแอปพลิเคชันบนสมาร์ตโฟน	3
2.1.1 ระบบปฏิบัติการบนมือถือ	3
2.1.2 โครงสร้างของแอปพลิเคชัน	3
2.1.3 เครื่องมือที่ใช้ในการพัฒนา	4
2.2 Internet of Things	4
2.2.1 ระบบไอโอที	4
2.2.2 โพรโทคอล	5
2.2.3 สมาร์ตฟาร์ม	7
2.3 Amazon Web Services	7
2.3.1 AWS Lambda / Serverless	8
2.3.2 AWS IoT Core	8
2.3.3 AWS Amplify	8
2.3.4 API Gateway	9
2.3.5 S3 (Simple Storage Service)	9
2.3.6 DynamoDB	9
2.3.7 IAM (Identity Access Management)	9
2.3.8 Cognito	10
2.3.9 CloudWatch	10

สารบัญ (ต่อ)

2.3.10 CloudFormation	10
2.4 Foreign Function Interface	10
บทที่ 3 การออกแบบโครงงาน	11
3.1 การออกแบบภาพรวม	11
3.2 สภาพแวดล้อมสำหรับการพัฒนา	12
3.3 สถาปัตยกรรมโครงสร้างของแอปพลิเคชัน	12
3.4 การออกแบบแอปพลิเคชันโดยรวม	13
3.5 การออกแบบฐานข้อมูล	14
3.6 การออกแบบหน้าต่างแอปพลิเคชัน	19
3.7 การออกแบบบริการต่างๆ	24
3.7.1 การเข้าสู่ระบบและการลงทะเบียน	24
3.7.2 การส่งข้อมูลจากฮาร์ดแวร์และการปล่อยข้อมูลทันทีให้ผู้ติดตาม	26
3.7.3 การแสดงข้อมูลผู้ใช้ ฟาร์ม และอุปกรณ์	27
3.7.4 การสั่งปิด-เปิดการรับค่าจากฮาร์ดแวร์	28
3.7.5 การแสดงผลค่าของอุปกรณ์และประวัติ	28
3.7.6 การแสดงผลในรูปแบบกราฟ	29
3.7.7 การค้นหาอุปกรณ์ในหน้าหลัก	30
3.7.8 การสร้างไฟล์รายงานผลแบบ PDF จากค่าของอุปกรณ์	32
3.7.9 การวิเคราะห์ค่าจากข้อมูลอุปกรณ์	33
3.7.10 การแจ้งเตือนผู้ใช้เมื่อค่าจากอุปกรณ์เกินจุดที่ตั้งค่าไว้	34
3.7.11 การส่งค่าไปควบคุมอุปกรณ์	35
บทที่ 4 ผลการดำเนินงาน	37
4.1 รายละเอียดการดำเนินการ	37
4.1.1 การติดตั้ง Flutter และ Android studio	37
4.1.2 การติดตั้ง Rust	47
4.1.3 การติดตั้ง Node.js เพื่อทำ Serverless	52
4.2 ขั้นตอนในการทำงาน	55
4.3 การใช้งานภายในแอปพลิเคชัน	57
4.3.1 การเข้าสู่ระบบ	57
4.3.2 การลงทะเบียน	58
4.3.3 หน้าสอนการใช้งาน	59

สารบัญ (ต่อ)

4.3.4 หน้าแดชบอร์ดหลัก	61
4.3.5 การเปลี่ยนฟาร์ม	63
4.3.6 การค้นหาอุปกรณ์	64
4.3.7 ประวัติการทำงานของอุปกรณ์	65
4.3.8 รายละเอียดของอุปกรณ์	66
4.3.9 การควบคุมอุปกรณ์	68
4.3.10 หน้าการวิเคราะห์	69
4.3.11 การส่งออกเอกสารค่าข้อมูลของอุปกรณ์	70
4.4 การใช้บริการต่างๆของ AWS ในระบบ	71
4.4.1 การตั้งค่าทำระบบยืนยันตัวตน	71
4.4.2 การทำฟังก์ชัน Serverless	75
4.4.2.1 การจัดการข้อมูลต่างๆ กับ AWS DynamoDB	85
4.4.2.2 การเชื่อมต่อกับ AWS IoT Core เพื่อจัดการข้อมูลจากอุปกรณ์	99
บทที่ 5 สรุปผล อภิปรายผล และข้อเสนอแนะ	130
5.1 สรุปการดำเนินการ	130
5.2 อภิปรายผล	131
5.3 ข้อเสนอแนะ	131
บรรณานุกรม	132

สารบัญตาราง

ตารางที่ 2.1 คำขอที่ใช้ใน HTTP 5

ตารางที่ 2.2 คำขอที่ใช้ใน REST 6

ตารางที่ 2.3 โค้ดแสดงสถานะ (Status code) 7



สารบัญรูป

รูปที่ 1.1	แผนการทำงานของโครงงาน	2
รูปที่ 3.1	ภาพรวมของระบบการทำงานแอปพลิเคชัน	11
รูปที่ 3.2	แผนภาพกรณีการใช้งานของแอปพลิเคชัน	14
รูปที่ 3.3	การเชื่อมต่อของฐานข้อมูล	14
รูปที่ 3.4	แอททริบิวต์ของตาราง FarmUser และตัวอย่างข้อมูลในรูปแบบ JSON	16
รูปที่ 3.5	แอททริบิวต์ของตาราง Farm และตัวอย่างข้อมูลในรูปแบบ JSON	17
รูปที่ 3.6	แอททริบิวต์ของตาราง FarmDevice และตัวอย่างข้อมูลในรูปแบบ JSON	18
รูปที่ 3.7	แอททริบิวต์ของตาราง FarmName และตัวอย่างข้อมูลในรูปแบบ JSON	18
รูปที่ 3.8	หน้า Splash screen ที่ได้ทำการออกแบบไว้	19
รูปที่ 3.9	หน้า login screen ที่ได้ทำการออกแบบไว้	20
รูปที่ 3.10	หน้าหลักของแอปพลิเคชัน	20
รูปที่ 3.11	popup เมื่อกด about this device	21
รูปที่ 3.12	หน้าในการดูประวัติของอุปกรณ์	21
รูปที่ 3.13	หน้าสอนใช้งานหน้าที่ 1	22
รูปที่ 3.14	หน้าสอนใช้งานหน้าที่ 2	22
รูปที่ 3.15	หน้าสอนใช้งานหน้าที่ 3	23
รูปที่ 3.16	หน้าสอนใช้งานหน้าที่ 4	23
รูปที่ 3.17	ผังงานของการเข้าสู่ระบบและลงทะเบียน	25
รูปที่ 3.18	ผังงานของการส่งข้อมูลจากฮาร์ดแวร์และการปล่อยข้อมูลทันทีให้ผู้ติดตาม	26
รูปที่ 3.19	ผังงานของการแสดงข้อมูลผู้ใช้ ฟาร์ม และอุปกรณ์	27
รูปที่ 3.20	ผังงานของการสั่งปิด-เปิดการรับค่าจากฮาร์ดแวร์	28
รูปที่ 3.21	ผังงานของการแสดงผลค่าของอุปกรณ์และประวัติ	29
รูปที่ 3.22	ผังงานของการแสดงผลในรูปแบบกราฟ	30
รูปที่ 3.23	ผังงานของการค้นหาอุปกรณ์ในหน้าหลัก	31
รูปที่ 3.24	ผังงานของการสร้างไฟล์รายงานผลแบบ PDF จากค่าของอุปกรณ์	32
รูปที่ 3.25	ผังงานของการวิเคราะห์ค่าจากข้อมูลอุปกรณ์	34
รูปที่ 3.26	ผังงานของการแจ้งเตือนผู้ใช้เมื่อค่าจากอุปกรณ์เกินจุดที่ตั้งค่าไว้	35
รูปที่ 3.27	ผังงานของการส่งค่าไปควบคุมอุปกรณ์	36
รูปที่ 4.1	เว็บไซต์ในการติดตั้ง Flutter	37
รูปที่ 4.2	หน้าต่างในการเพิ่ม path	38
รูปที่ 4.3	การเพิ่ม path ของ flutter	39

สารบัญรูป (ต่อ)

รูปที่ 4.4 การตรวจสอบสถานะการติดตั้งของ Flutter	39
รูปที่ 4.5 เว็บไซต์ในการติดตั้ง Android studio	40
รูปที่ 4.6 หน้าต่างในการดาวน์โหลด Android SDK	40
รูปที่ 4.7 หน้าต่างดาวน์โหลด plugin ของ flutter ภายใน android studio	41
รูปที่ 4.8 หน้าต่างในการเพิ่ม path ของ Android SDK	42
รูปที่ 4.9 Path ของ Android SDK ทั้งหมดที่ต้องเพิ่ม	43
รูปที่ 4.10 สถานะการติดตั้งของ flutter ภายในขั้นตอนที่ 2.2.3.	44
รูปที่ 4.11 การเลือกอุปกรณ์จำลองในการพัฒนา	44
รูปที่ 4.12 รุ่นระบบที่ต้องการติดตั้งในอุปกรณ์จำลอง	45
รูปที่ 4.13 หน้าต่างเลือกอุปกรณ์จำลอง	45
รูปที่ 4.14 ตัวอย่าง อุปกรณ์จำลองที่สร้างขึ้นมา	46
รูปที่ 4.15 หน้าต่างในการดาวน์โหลด NDK	47
รูปที่ 4.16 เว็บไซต์ในการติดตั้ง Rust	47
รูปที่ 4.17 เว็บไซต์ในการติดตั้ง LLVM	48
รูปที่ 4.18 คำสั่งในการเพิ่มไฟล์เดอร์ของ rust	48
รูปที่ 4.19 ชุดคำสั่งภายใน cargo.toml	49
รูปที่ 4.20 ชุดคำสั่งที่ต้องเพิ่มในหัวข้อ build-dependencies	49
รูปที่ 4.21 ชุดคำสั่งที่ต้องเพิ่มในไฟล์ lib.rs	49
รูปที่ 4.22 ชุดคำสั่งที่ในการเพิ่มแพลตฟอร์ม	50
รูปที่ 4.23 ชุดคำสั่งที่ต้องเพิ่มในไฟล์ android/app/build.gradle	51
รูปที่ 4.24 เว็บไซต์ในการติดตั้ง node.js	52
รูปที่ 4.25 บริการ IAM ของ AWS	52
รูปที่ 4.26 การสร้าง Users และ Add users	53
รูปที่ 4.27 การสร้าง Access keys	53
รูปที่ 4.28 หน้าต่างแสดง Access keys	54
รูปที่ 4.29 หน้าต่าง Terminal สำหรับใส่ access key และ secret key	55
รูปที่ 4.30 หน้าล็อกอินแอปพลิเคชัน	58
รูปที่ 4.31 หน้าลงทะเบียน	59
รูปที่ 4.32 หน้า Onboard	60
รูปที่ 4.33 การเชื่อมต่อจากหน้ายืนยันตัวตน หน้าสอนการใช้งาน และหน้าแดชบอร์ด	60
รูปที่ 4.34 หน้าหลักของแอปพลิเคชัน	61

สารบัญรูป (ต่อ)

รูปที่ 4.35 การเชื่อมต่อจากหน้าแดชบอร์ดไปยังหน้าคุณลักษณะอื่น	62
รูปที่ 4.36 หน้าหลักของแอปพลิเคชันเมื่อกดปุ่มเปลี่ยนฟาร์ม	63
รูปที่ 4.37 แสดงการค้นหาอุปกรณ์เมื่อมีตัวอักษรตรงกับตัวอักษรในชื่ออุปกรณ์	64
รูปที่ 4.38 หน้าประวัติของแอปพลิเคชัน	65
รูปที่ 4.39 หน้ารายละเอียดของอุปกรณ์	67
รูปที่ 4.40 หน้าเมนูการควบคุมอุปกรณ์	68
รูปที่ 4.41 หน้าวิเคราะห์การทำงานของอุปกรณ์	69
รูปที่ 4.42 หน้าการสร้างรายงาน	70
รูปที่ 4.43 หน้าแสดงผลไฟล์รายงาน	71
รูปที่ 4.44 การสร้าง บริการ serverless หรือ AWS Lambda	81
รูปที่ 4.45 ผลลัพธ์เมื่อกดที่ endpoint	81
รูปที่ 4.46 ฟังก์ชัน ภายใน AWS Lambda	82
รูปที่ 4.47 หน้าต่างภายในฟังก์ชัน	83
รูปที่ 4.48 แถบแสดงเมนู Code ของบริการ AWS Lambda	83
รูปที่ 4.49 แถบแสดงเมนู Test ของบริการ AWS Lambda	84
รูปที่ 4.50 แถบแสดงเมนู Monitor ของบริการ AWS Lambda	84
รูปที่ 4.51 แถบแสดงเมนู Configuration ของบริการ AWS Lambda	85
รูปที่ 4.52 แถบแสดงเมนู Version ของบริการ AWS Lambda	85
รูปที่ 4.53 สิทธิในฐานข้อมูลของแต่ละองค์ประกอบ	86
รูปที่ 4.54 แผนผังการเชื่อมต่อกับบริการ DynamoDB	86
รูปที่ 4.55 แผนผังการเชื่อมต่อกับบริการ IoT Core	99
รูปที่ 4.56 แผนผังช่วงการส่งค่าจากอุปกรณ์ฮาร์ดแวร์	100
รูปที่ 4.57 แผนผังช่วงการส่งค่าจากแอปพลิเคชันไปยังอุปกรณ์ฮาร์ดแวร์	100
รูปที่ 4.58 Settings ต่างๆ ของบริการ AWS IoT Core	102
รูปที่ 4.59 หน้า Certificates ของบริการ AWS IoT Core	102
รูปที่ 4.60 การสร้าง Certificate ของบริการ AWS IoT Core	103
รูปที่ 4.61 การดาวน์โหลด Certificate ของบริการ AWS IoT Core	103
รูปที่ 4.62 หน้า Things ของบริการ AWS IoT Core	104
รูปที่ 4.63 การสร้าง Thing ของบริการ AWS IoT Core	104
รูปที่ 4.64 การสร้าง Thing ขั้นตอนที่ 1 ของบริการ AWS IoT Core	105
รูปที่ 4.65 การสร้าง Thing ขั้นตอนที่ 2 ของบริการ AWS IoT Core	105

สารบัญรูป (ต่อ)

รูปที่ 4.66 หน้า Certificate แถบ Things ของบริการ AWS IoT Core	105
รูปที่ 4.67 การเพิ่ม Thing ไปที่ Certificate	106
รูปที่ 4.68 หน้า Certificate แถบ Things หลังเพิ่ม Thing	106
รูปที่ 4.69 หน้า Policy ของบริการ AWS IoT Core	107
รูปที่ 4.70 การสร้าง Policy ของบริการ AWS IoT Core	107
รูปที่ 4.71 การกำหนด Policy ในขั้นตอนการสร้าง	108
รูปที่ 4.72 หน้า Certificate แถบ Policies ของบริการ AWS IoT Core	108
รูปที่ 4.73 การเพิ่ม Policy ไปยัง Certificate	109
รูปที่ 4.74 หน้า Certificate แถบ Policies หลังเพิ่ม Policy	109
รูปที่ 4.75 หน้า Rule ของบริการ AWS IoT Core	111
รูปที่ 4.76 หน้าการสร้าง Rule ของบริการ AWS IoT Core	112
รูปที่ 4.77 หน้าการสร้าง SQL ของ Rule ของบริการ AWS IoT Core	112
รูปที่ 4.78 หน้าการเชื่อมโยงบริการอื่นของ Rule ของบริการ AWS IoT Core	113
รูปที่ 4.79 หน้าบทวนการสร้าง Rule ของบริการ AWS IoT Core	113
รูปที่ 4.80 หน้า AWS Lambda แถบ Configuration หัวข้อ Triggers	114
รูปที่ 4.81 แสดงการเพิ่ม Trigger ไปยังฟังก์ชัน	114
รูปที่ 4.82 หน้า AWS Lambda แถบ Configuration หลังเพิ่ม Trigger ใหม่	115
รูปที่ 4.83 เมนูการควบคุมในหน้าแดชบอร์ดกับอุปกรณ์ฮาร์ดแวร์ที่เกี่ยวข้อง	120
รูปที่ 4.84 การเปิดหน้าการส่งค่าในหน้าแดชบอร์ดกับอุปกรณ์ฮาร์ดแวร์ที่เกี่ยวข้อง	121
รูปที่ 4.85 หลังส่งค่าในหน้าแดชบอร์ดไปยังอุปกรณ์ฮาร์ดแวร์ที่เกี่ยวข้อง	121
รูปที่ 4.86 หน้า DynamoDB แสดงตารางอุปกรณ์	127
รูปที่ 4.87 หน้า DynamoDB แสดงตารางชื่อฟาร์ม	128
รูปที่ 4.88 หน้าข้อมูลอุปกรณ์ LIGHT01 ใน DynamoDB ตาราง Farmtest	128
รูปที่ 4.89 หน้าข้อมูลอุปกรณ์ LIGHT01 ใน DynamoDB ตาราง ITEFarm	129

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันได้เกิดการเปลี่ยนแปลงในด้านเศรษฐกิจ และสังคม ที่ทำให้รายได้จากการขายของภายในฟาร์มลดลง ทำให้เกิดผลกระทบกับบุคลากรที่ทำหน้าที่ในการดูแลฟาร์ม คือ มีค่าครองชีพไม่พอ ทำให้ประชาชนเริ่มหันไปทำงานอื่นๆที่มีรายได้มากกว่า ส่งผลให้ฟาร์มต่างๆที่อยู่ภายในประเทศไทยขาดแคลนบุคลากรในการดูแลฟาร์มในหลายๆแห่ง ทั้งฟาร์มที่เปิดมาเป็นเวลานานทำให้เจ้าของฟาร์มแห่งนั้นๆมีรายได้ที่ลดลงเป็นอย่างมาก โดยเฉพาะอย่างยิ่งผู้ที่ต้องการทำฟาร์มรายได้ใหม่ๆก็จะต้องใช้เงินทุนจำนวนมากในการลงทุนดูแลฟาร์มแห่งใหม่นั้น

เพื่อจัดการกับปัญหานี้ทางผู้จัดทำได้ทำการวางแผนในการแก้ไขปัญหาคือว่า จะสามารถจัดการกับฟาร์มได้อย่างไรโดยไม่ต้องเพิ่มบุคลากรหรือลดจำนวนบุคลากรแต่ยังทำให้การจัดการฟาร์มยังสามารถจัดการฟาร์มได้โดยง่าย โดยการที่ผู้จัดทำได้ออกแบบระบบในการดูแลฟาร์มที่ช่วยในการจัดการและตรวจสอบ ฟาร์มได้ในระยะทางไกลผ่านแอปพลิเคชันที่จะคอย รับและส่งข้อมูล ต่างๆที่ได้รับจากอุปกรณ์ที่ติดตั้งอยู่ภายในฟาร์ม โดยผ่านเซิร์ฟเวอร์ AWS ที่สามารถควบคุมผ่านทางอุปกรณ์พกพาของผู้ที่อยู่ในการดูแลฟาร์มได้เลย

1.2 วัตถุประสงค์

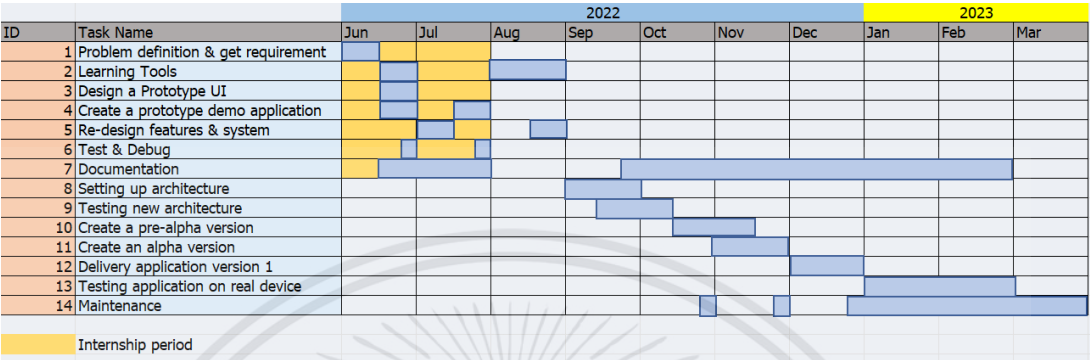
1. เพื่ออำนวยความสะดวกในการจัดการอุปกรณ์ต่างๆภายในฟาร์ม
2. เพื่อศึกษาการพัฒนาแอปพลิเคชันบนมือถือ
3. เพื่อนำความรู้ที่ได้จากการศึกษาไปใช้ในการปฏิบัติงานจริง

1.3 ขอบเขตในการดำเนินงาน

1. ทดสอบกับอุปกรณ์ทดลองภายในกลุ่มของผู้จัดทำ
2. ภายในระยะเวลาของปีการศึกษา พ.ศ. 2565 เทอม 1
3. สามารถทำการรับหรือส่งข้อมูลจากฮาร์ดแวร์ได้
4. พัฒนาแอปพลิเคชันบน flutter เนื่องจาก flutter เป็นชุดพัฒนาในการพัฒนาอย่างกว้างขวางทำให้สามารถพัฒนาได้รวดเร็วและมีประสิทธิภาพ และยังสามารถนำไปพัฒนาต่อในระบบ ios ได้อีกด้วย

5. ใช้ Amazon Web Services (AWS) เพื่อให้สามารถเป็นตัวกลางในการสื่อสารของระบบโดยในตัว AWS จะมี services หลากอย่างทำให้ง่ายต่อการจัดการ

1.4 ระยะเวลาการทำงาน



รูปที่ 1.1 แผนการทำงานของโครงการ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. เพื่ออำนวยความสะดวกในการจัดการอุปกรณ์ต่างๆภายในฟาร์ม
2. เพื่อศึกษาการพัฒนาแอปพลิเคชันบนมือถือ
3. เพื่อนำความรู้ที่ได้จากการศึกษาไปใช้ในการปฏิบัติงานจริง

บทที่ 2

ทฤษฎีพื้นฐานที่ใช้

2.1 การพัฒนาแอปพลิเคชันบนสมาร์ตโฟน

2.1.1 ระบบปฏิบัติการบนมือถือ

ระบบที่ออกแบบมาเพื่อควบคุมการทำงานต่างๆของโทรศัพท์มือถือ แท็บเล็ต สมาร์ทวอตช์หรืออุปกรณ์มือถืออื่นๆซึ่งมีจุดประสงค์เหมือนกันกับระบบปฏิบัติการบนคอมพิวเตอร์ แต่มีลักษณะเฉพาะอื่นๆ ที่เกี่ยวข้องกับมือถือ เช่น การรวมโมเด็มไร้สายไว้ในตัว หรือ ถาดใส่ซิมสำหรับโทรศัพท์ ในปัจจุบันมีระบบปฏิบัติการจำนวนมากที่ถูกใช้บนอุปกรณ์มือถือ ซึ่งมีอยู่ 2 ระบบใหญ่ๆ ที่เป็นที่รู้จัก ได้แก่ แอนดรอยด์ (Android) และ ไอโอเอส (iOS)

2.1.2 โครงสร้างของแอปพลิเคชัน

โครงสร้างของแอปพลิเคชัน คือ อินเทอร์เฟซ, เทคนิคทั้งหมดที่ใช้ในการออกแบบแอปพลิเคชันและ ลักษณะการทำงานโดยรวมของแอปพลิเคชันทั้งหมด โดยในการออกแบบแอปพลิเคชันให้มีประสิทธิภาพมากที่สุดจำเป็นต้องมีแผนการทำงานและคำแนะนำที่ครอบคลุมและแนวคิดที่ชัดเจนเกี่ยวกับโครงสร้างของแอปพลิเคชันบนอุปกรณ์เคลื่อนที่โครงสร้างของแอปพลิเคชันประกอบไปด้วย layers ที่ต่างกัน ดังนี้ [1]

- Presentation layer

ใน layer นี้ จะประกอบไปด้วย UI/UX ของแอปพลิเคชันทั้งหมด แสดงว่าในชั้นนี้จะเป็นชั้นที่ผู้ใช้สามารถทำความเข้าใจและใช้งานได้มากที่สุด

- Business layer

ใน layer นี้เป็นส่วนการจัดการทางด้านของธุรกิจนั้นหมายถึงต้องประกอบไปด้วยขั้นตอนการทำงาน (Work flow), โครงสร้างของการตลาด (Market structure) และตรรกะทางธุรกิจ (Business logic) ทั้งหมดไว้ใน layer นี้

- Data layer

ใน layer นี้มีความสำคัญเป็นอย่างมากเพราะ ต้องมีการจัดการข้อมูลของแอปพลิเคชันอย่างมีประสิทธิภาพ และ ยังมีระบบป้องกันความปลอดภัยอีกด้วย

โครงสร้างแอปพลิเคชันยังสามารถแบ่งได้เป็น 2 ประเภทดังนี้

Native Application คือ การออกแบบมาสำหรับระบบปฏิบัติการ (OS) โดยเฉพาะทำให้ผู้ใช้งานรู้สึกเป็นธรรมชาติ และ สะดวกกว่า ในการออกแบบและพัฒนาโครงสร้างแบบ Native Application เป็นการพัฒนาแอปพลิเคชันบนมือถือที่ทำงานบนระบบปฏิบัติการ

เฉพาะเท่านั้น ทำให้ผู้ใช้สามารถเข้าถึงได้จากบางแห่งเท่านั้นโดยส่วนใหญ่ผ่านทาง Google Play Store หรือ App Store

- การพัฒนา Native Application บน ระบบปฏิบัติการ Android

ส่วนใหญ่จะใช้ภาษา Kotlin, Java หรือ C++ ในการพัฒนาและมีเครื่องมืออื่นๆ ประกอบด้วย

- SDKs – Software Development Kits โดยรวมถึง Android Studio และ IDE
- Android Jetpack คือ ชุดส่วนประกอบของ Android
- Firebase คือ แพลตฟอร์มการพัฒนามือถือ
- เครื่องมือคำสั่งอื่นๆ สำหรับระบบปฏิบัติการ Windows, Mac และ Linux
- การพัฒนา Native Application บน ระบบปฏิบัติการ iOS

ส่วนใหญ่จะใช้ภาษา Objective-C หรือ Swift

Hybrid Application คือ การนำ Native Application มารวมกับ Web Application ทำให้ตัวแอปพลิเคชันสามารถทำงานได้ทุกแพลตฟอร์ม

2.1.3 เครื่องมือที่ใช้ในการพัฒนา

Flutter [2] เป็นชุดคำสั่งแบบโอเพนซอร์สที่ถูกพัฒนาขึ้นมาโดย Google จุดเด่นของ Flutter อยู่ที่สามารถสร้างแอปพลิเคชันบนมือถือได้ภายในหน้าเดียวและสามารถรองรับได้ทั้ง 2 แพลตฟอร์มหลักบนมือถือ คือ แอนดรอยด์ และไอโอเอส นอกจากนี้ยังสามารถนำไปพัฒนาให้ทำงานบนแพลตฟอร์มอื่นๆได้ เช่น ลินุกซ์ วินโดวส์ และ เว็บ โดยใช้ภาษาโปรแกรมมิ่งเพียงภาษาเดียวเท่านั้น ก็คือ ภาษา Dart ซึ่งทำงานร่วมกันกับ Kotlin และ Swift โดยที่ไม่จำเป็นต้องไปแก้ไขในส่วนที่เป็นภาษา Kotlin หรือ Swift นอกจากนี้ยังมีจุดเด่นอีก 1 อย่าง ก็คือ ความสามารถในการแสดงผลที่เร็วขณะที่กำลังเขียนโปรแกรมโดยที่ไม่ต้องไปเริ่มติดตั้งแอปพลิเคชันแล้วรันโปรแกรมใหม่ ซึ่งทำให้สามารถเห็นผลการเปลี่ยนแปลงของแอปพลิเคชันได้ทันที

2.2 Internet of Things

2.2.1 ระบบไอโอที

ระบบไอโอที (Internet of Things) คือ ระบบที่เชื่อมโยงอุปกรณ์ อิเล็กทรอนิกส์ต่างๆเข้าไว้ด้วยกัน เช่น เซนเซอร์ต่างๆโดยผ่านเครือข่ายอินเทอร์เน็ต การเชื่อมโยงนี้จะ ทำให้อุปกรณ์ในเครือข่ายสามารถสื่อสารกันได้ และยังทำให้ผู้ใช้สั่งการควบคุมอุปกรณ์ได้ ซึ่งข้อมูลที่เก็บมาได้จากอุปกรณ์จะนำมาเก็บไว้บนพื้นที่ เช่น บนคลาวด์ เป็นต้น และสามารถนำข้อมูลที่เก็บมาได้เหล่านั้นเพื่อไปวิเคราะห์ในภายหลัง การใช้ระบบไอโอทียังช่วยในเรื่องความแม่นยำของข้อมูล และความเป็นปัจจุบัน (Real-time) ทำให้ผู้ใช้สามารถมองเห็นถึงปัญหาได้ตลอดเวลา แต่ด้วยจำนวนข้อมูลที่มีจำนวนมากจึงมีโอกาสเสี่ยงในเรื่องของความปลอดภัยด้วยเช่นกัน

2.2.2 โพรโทคอล

โพรโทคอลคือข้อกำหนดในการสื่อสารกันระหว่างอุปกรณ์ต่างๆเพื่อให้เกิดการสื่อสารระหว่างกันเปรียบเสมือนกับภาษาที่ใช้ในการสื่อสารที่อุปกรณ์สามารถนำไปใช้ในการติดต่อกันได้ซึ่งในโพรโทคอลแต่ละชนิดก็มีจุดประสงค์ที่ทำงานแตกต่างกันไปในโปรเจกต์นี้ได้เน้นใช้โพรโทคอลสำหรับการแลกเปลี่ยนข้อมูล ได้แก่ MQTT และ HTTP/REST

2.2.2.1 MQTT

MQTT [3] (ย่อมาจาก MQ Telemetry Transport) เป็นโพรโทคอลที่ใช้ในการส่งข้อความ ซึ่งถูกใช้บ่อยในอุปกรณ์ประเภทไอโอที โพรโทคอลนี้ขับเคลื่อนด้วยเหตุการณ์ และ เชื่อมต่ออุปกรณ์ผ่านรูปแบบของการติดตาม-เผยแพร่ (Subscribe-Publish) ผู้ส่งและผู้รับจะสื่อสารกันผ่านหัวข้อที่กำหนดไว้ การเชื่อมต่อระหว่างผู้ใช้งานจะต้องผ่านคนกลางคือ ผู้ให้บริการ MQTT คนกลางนี้จะมีหน้าที่รับข้อความที่เข้ามาจากหัวข้อต่างๆ และกระจายไปยังผู้ที่ติดตามหัวข้อนั้นๆ

2.2.2.2 HTTP/REST

HTTP (Hypertext Transfer Protocol) [4] เป็นโพรโทคอลชั้นแอปพลิเคชันสำหรับการส่งเอกสารไฮเปอร์มีเดีย เช่น HTML โพรโทคอล HTTP นี้ถูกนำไปใช้ในการสื่อสารกันระหว่างเว็บเบราว์เซอร์ของผู้ใช้กับทรัพยากรบนอินเทอร์เน็ต HTTP เป็นโพรโทคอลที่ไร้สถานะ หมายถึง แต่ละคำขอจะไม่ถูกเก็บไว้ในแต่ละครั้งที่ใช้ และเป็นอิสระต่อกัน

ตารางที่ 2.1 คำขอที่ใช้ใน HTTP

Methods	Operations	Response
GET	ร้องขอข้อมูลที่แสดงผลในทรัพยากรนั้น	ข้อมูลที่ร้องขอ, Status Code
HEAD	เหมือนกันกับ GET แต่ไม่มีข้อมูล	ความเป็นมาของข้อมูล Status Code
POST	ส่งเอนทิทีไปยังทรัพยากรที่ระบุ ทำให้เกิดการเปลี่ยนแปลงในสถานะ	Status code
PUT	แทนที่การแสดงผลปัจจุบันของเป้าหมายด้วย payload ของคำร้องขอ	Status code
DELETE	ลบทรัพยากรที่ระบุ	Status Code

CONNECT	สร้างช่องทางไปยังเซิร์ฟเวอร์จากผู้รับด้วยคำร้องขอ	Status code
OPTIONS	อธิบายตัวเลือกช่องทางการสื่อสารของทรัพยากรเป้าหมาย	หัวเรื่องที่บ่งชี้ถึงตัวเลือกที่สามารถปรับใช้กับทรัพยากรเป้าหมายได้
TRACE	ทดสอบเส้นทางการเดินทางของข้อมูลที่จะเดินทางไปยังทรัพยากรเป้าหมาย	ค่า Max-forward ของ 0 ในคำร้องขอ
PATCH	นำส่วนที่ถูกปรับเล็กน้อยมาใช้กับทรัพยากร	Status code

REST (Representational State Transfer) [5] เป็นสถาปัตยกรรมซอฟต์แวร์ที่กำหนดเงื่อนไขการทำงานของ API ว่าต้องทำงานอย่างไร โดยมีหลักดังต่อไปนี้ 1. อินเทอร์เน็ตเป็นรูปแบบเดียวกันมีข้อจำกัด 4 ประการคือ คำขอควรระบุทรัพยากร ผู้ใช้มีข้อมูลเพียงพอในทรัพยากรที่จะแก้ไขหรือลบทรัพยากร ผู้ใช้ได้รับข้อมูลเกี่ยวกับวิธีการประมวลผลข้อมูลเพิ่มเติม และผู้ใช้ได้รับข้อมูลเกี่ยวกับทรัพยากรอื่นๆที่จำเป็นสำหรับการทำงานให้สมบูรณ์ 2. ความสำเร็จสถานะ เซิร์ฟเวอร์สามารถดำเนินการตามคำขอได้ทั้งหมดโดยแต่ละคำขอจะไม่ถูกเก็บ และไม่ขึ้นกับคำขอก่อนหน้า 3. ระบบที่แบ่งออกเป็นชั้น ผู้ใช้สามารถเชื่อมต่อกับตัวกลางอื่นๆได้ ที่ถูกยอมรับโดยทั้งผู้ใช้และเซิร์ฟเวอร์ และยังคงรอการตอบสนองจากเซิร์ฟเวอร์ 4. ความสามารถในการแคช ทรัพยากรจะถูกเก็บไว้เมื่อตอบสนองครั้งแรกทำให้เวลาในการตอบสนองเร็วมากขึ้น

ตารางที่ 2.2 คำขอที่ใช้ใน REST

Methods	Operations	Response
POST	ส่งเอนทิตีไปยังทรัพยากรที่ระบุ ทำให้เกิดการเปลี่ยนแปลงในสถานะ	Status code
GET	ร้องขอข้อมูลที่แสดงผลในทรัพยากรนั้น	ข้อมูลที่ร้องขอ, Status Code
PUT	แทนที่การแสดงผลปัจจุบันของเป้าหมายด้วย payload ของคำร้องขอ	Status code
PATCH	นำส่วนที่ถูกปรับเล็กน้อย	Status code

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

	มาใช้กับทรัพยากร	
DELETE	ลบทรัพยากรที่ระบุ	Status Code

ตารางที่ 2.3 โค้ดแสดงสถานะ (Status code) [6]

Class	Range
Informational response	100 - 199
Successful response	200 – 299
Redirection response	300 – 399
Client error response	400 – 499
Server error response	500 - 599

2.2.3 สมาร์ทฟาร์ม

สมาร์ทฟาร์ม คือ แนวคิดของระบบการจัดการที่มุ่งเน้นไปที่อุตสาหกรรมเกษตรโดยการผสมผสานกับเทคโนโลยีขั้นสูงต่างๆ ได้แก่ Big Data, Cloud และระบบไอโอทีซึ่งเป็นแกนกลางหลักของแนวคิดนี้ เพื่อที่จะใช้ในการติดตาม การทำระบบอัตโนมัติ และกระบวนการการวิเคราะห์ต่างๆ นอกจากนี้ยังช่วยลดทรัพยากรแรงงานให้น้อยลงและทำให้การควบคุมหรือค่าที่ได้จากการติดตามมีความแม่นยำขึ้นด้วย

2.3 Amazon Web Services

คลาวด์ (Cloud) เป็นการนำเครื่องเซิร์ฟเวอร์มาเชื่อมต่อทำงานร่วมกันเป็นกลุ่ม ในรูปแบบ Cluster ทำหน้าที่เป็นที่จัดเก็บข้อมูล การนำไปประมวลผล หรือ อื่นๆ โดยที่ผู้พัฒนาไม่ต้องมีตัวเครื่องเซิร์ฟเวอร์อยู่เลย กล่าวคือ การจำลองเซิร์ฟเวอร์ไว้บนเครื่องเซิร์ฟเวอร์อีกทีหนึ่ง ในงานนี้ได้เลือกใช้ Firebase ในช่วงทดสอบ และปรับเปลี่ยนมาใช้ AWS

AWS (Amazon Web Services) [7] เป็นแพลตฟอร์มคลาวด์ที่ให้บริการโดย Amazon ที่มีมากกว่า 200 บริการจากดาต้าเซนเตอร์รอบโลก เน้นการให้บริการออนไลน์ที่สามารถขยายระดับได้และมีแนวทางการประมวลผลของคลาวด์ที่คุ้มค่าต่อราคา บริการของ Amazon มีรายละเอียดได้แก่ บริการการประมวลผล ที่เก็บข้อมูล ฐานข้อมูล การทำเครือข่ายและการขนส่งข้อมูล เครื่องมือความปลอดภัย เครื่องมือในการพัฒนา และ เครื่องมือในการจัดการ เป็นต้น ในโปรเจกต์นี้ทางผู้จัดทำได้เลือกใช้เครื่องมือ ดังต่อไปนี้

- บริการการประมวลผล
 - o Lambda / Serverless
- Internet of Things

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- o IoT Core
- Front End Web & Mobile
 - o Amplify
 - o API Gateway
- ที่เก็บข้อมูล
 - o S3 (Simple Storage Service)
- ฐานข้อมูล
 - o DynamoDB
- เครื่องมือความปลอดภัย
 - o IAM (Identity Access Management)
 - o Cognito
- เครื่องมือการจัดการ
 - o CloudWatch
 - o CloudFormation

2.3.1 AWS Lambda / Serverless

AWS Lambda [8] คือการรันโค้ดโดยไม่ต้องใช้เซิร์ฟเวอร์มาจัดการ ซึ่งสามารถทำงานได้แบบอัตโนมัติและสามารถขยายระดับการทำงานได้ ทำให้ค่าใช้จ่ายลดลงเหลือแค่ช่วงที่โค้ดถูกรันเท่านั้น โค้ดเหล่านี้ที่เรียกว่า “Lambda Function” หรือ “Serverless Function” ประกอบไปด้วยฟังก์ชันหลายๆฟังก์ชันที่จะถูกเรียกใช้งานผ่านเหตุการณ์ต่างๆ เช่น การเรียกใช้งานผ่าน REST API หรือการอัปเดตข้อมูลในฐานข้อมูล เป็นต้น

2.3.2 AWS IoT Core

AWS IoT Core [9] เป็นบริการใน AWS Cloud ซึ่งทำหน้าที่ในการเชื่อมต่ออุปกรณ์ IoT เข้ากับ Cloud และที่อื่นๆ โดยมีการตั้งค่าการส่งข้อมูลต่างๆด้วยโปรโตคอล เช่น MQTT ทำให้อุปกรณ์ที่เชื่อมต่อสามารถรับข้อมูลหรือส่งผ่านมาด้วยหัวข้อต่างๆได้ แน่นอนว่าการส่งผ่านจะมีการใส่ Certificate ของอุปกรณ์มาก่อนที่จะเชื่อมต่อ ทำให้มั่นใจได้ว่า การรับ-ส่งข้อมูลจะเกิดขึ้นได้เฉพาะระหว่างอุปกรณ์ที่มีการยืนยันแล้วเท่านั้น

2.3.3 AWS Amplify

AWS Amplify [10] คือชุดเครื่องมือและบริการที่ใช้ในการพัฒนาแอปพลิเคชันหรือเว็บบน AWS โดยจะรวมไลบรารี องค์กรประกอบของ UI และชุดคำสั่ง command line ที่ใช้เพื่อสร้าง backend ของแอปพลิเคชันไว้ นอกจากนี้ยังสามารถเชื่อมต่อกับบริการอื่นๆของ AWS เช่น เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Cognito, Lambda และ S3 ได้โดยไม่ต้องใช้โค้ดหลายบรรทัด ซึ่งลดเวลาของการติดตั้งและการจัดการบริการของ backend

2.3.4 API Gateway

API Gateway [11] เป็นบริการที่ทำให้แอปพลิเคชันต่างๆสามารถเข้าถึงข้อมูล หรือการทำงานจากบริการ backend ได้ ซึ่งเป็นการสื่อสารแบบ 2 ทาง โดยมีวิธีการใช้ 2 วิธี ได้แก่ แบบ REST และ WebSocket นอกจากนี้ยังสนับสนุนการทำคอนเทนเนอร์และการทำงานแบบไร้เซิร์ฟเวอร์ (Serverless) API Gateway สามารถรับมือกับการเรียกใช้พร้อมกันได้สูงสุดถึง 10000 ครั้ง ซึ่งรวมถึงการจัดการการอนุญาตให้เข้าถึงข้อมูล CORS (Cross-origin resource sharing) การตรวจดูค่า และ การจัดการเวอร์ชัน

2.3.5 S3 (Simple Storage Service)

S3 [12] เป็นที่เก็บออบเจกต์ที่มีความยืดหยุ่น สามารถเก็บหรือส่งข้อมูลจากที่ไหนก็ได้และเป็นประเภทอะไรก็ได้ สามารถที่จะปรับขนาดของที่เก็บข้อมูลได้ตามต้องการ ข้อมูลที่ถูกเก็บจะมีการถูกสำรองเอาไว้เพื่อป้องกันข้อมูลที่สูญหาย และยังมีความคงทนของข้อมูลสูง กล่าวคือ ข้อมูลจะมีโอกาสหายได้ยาก

2.3.6 DynamoDB

DynamoDB [13] เป็นฐานข้อมูลแบบไม่มีความสัมพันธ์ (NoSQL) ที่เก็บข้อมูลในรูปแบบของกุญแจ-ค่า โดยมีจุดเด่นที่ความยืดหยุ่นของรายละเอียดข้อมูลที่ไม่จำกัดจำนวนของรายละเอียด ทำให้ในแต่ละรายละเอียดของข้อมูลสามารถมีหลายค่าได้ สามารถปรับขนาดได้อัตโนมัติตามจำนวนของข้อมูล และผู้ใช้ไม่ต้องทำการติดตั้งด้วยตัวเองเลย

2.3.7 IAM (Identity Access Management)

IAM [14] ใช้เพื่อกำหนดการเข้าถึง สิทธิต่างๆของแต่ละผู้ใช้ในแต่ละบทบาท ซึ่งจะถูกนำไปใช้ในการทำระบบการยืนยันตัวตนเพื่อป้องกันการเข้าถึงที่ไม่น่าเชื่อถือ และไม่ให้อัปเดตข้อมูล การกำหนดสิทธิของแต่ละบทบาท กลุ่มหรือผู้ใช้จะถูกกำหนดโดยนโยบาย นโยบายเหล่านี้จะเป็นตัวกำหนดการเข้าถึงทรัพยากรและบริการบน AWS ซึ่งจะทำการประเมินผู้ใช้ทุกครั้งที่มีร้องขอการเข้าถึง ผู้ใช้ที่นโยบายเหล่านี้ได้ไว้ที่บทบาทของตนจะสามารถเข้าถึงข้อมูลได้

2.3.8 Cognito

Cognito [15] ให้บริการในเรื่องระบบยืนยันตัวตนในการเข้าสู่ระบบหรือการลงทะเบียนเข้าสู่ระบบเพื่อเข้าที่แอปพลิเคชันด้วยการใช้ชื่อผู้ใช้และรหัสผ่าน โดยระบบยังสนับสนุนการเข้าสู่ระบบด้วยวิธีอื่นๆ เช่น เฟซบุ๊ก หรือ กูเกิ้ล เป็นต้น องค์ประกอบที่สำคัญของ Cognito มีอยู่ 2 อย่าง คือ user pool เป็นสมุดรายชื่อของผู้ใช้ ให้บริการการเข้าสู่ระบบและลงทะเบียน และ identity pool เป็นผู้ให้ credential กับผู้ใช้เพื่อการเข้าถึงข้อมูลและบริการต่างๆของ AWS และทำหน้าที่ยืนยันตัวตนของผู้ใช้

2.3.9 CloudWatch

CloudWatch [16] เป็นบริการที่ใช้ในการตรวจสอบทรัพยากรใน AWS Cloud และแอปพลิเคชันที่ทำงานบน AWS การใช้ CloudWatch สามารถเก็บและสะสมไฟล์บันทึกเหตุการณ์ที่เกิดจากการสร้างภายในแอปพลิเคชันได้ซึ่งสามารถใช้ในการแก้ปัญหาถ้าหากแอปพลิเคชันทำงานผิดพลาด

2.3.10 CloudFormation

CloudFormation [17] คือบริการที่ให้ผู้พัฒนาสามารถสร้างกลุ่มของทรัพยากรของ AWS และจากภายนอกมาจัดการให้เป็นกลุ่มเดียวกัน บริการนี้ถูกใช้งานในการติดตั้งปรับใช้ (deploy) ทำให้การติดตั้งสามารถทำได้อย่างรวดเร็ว และไม่มีข้อผิดพลาด และยังสามารถนำรูปแบบที่มีอยู่มาติดตั้งซ้ำได้ในพื้นที่ภูมิภาคอื่นๆ

2.4 Foreign Function Interface

Foreign Function Interface (ffi) เป็นตัวประสานที่ทำหน้าที่เชื่อมต่อภาษาที่ต่างกันให้สามารถทำงานร่วมกันได้ โดยในภาษาที่นำมาใช้นั้นต้องมีประเภทของตัวแปรคล้ายกันและเข้ากันได้ ถ้าหากภาษาที่นำมาใช้นั้นเข้ากันได้ ก็จะสามารถเรียกใช้ฟังก์ชันที่เขียนในอีกภาษาหนึ่งได้ ทำให้ช่วยเพิ่มประสิทธิภาพของโปรแกรมให้ดีขึ้น

ผู้จัดทำได้เลือกนำชุดพัฒนา Flutter ซึ่งเป็นภาษา Dart มีจุดเด่นในด้านการแสดงผลเร็วขณะทำการเขียนโปรแกรมและการรองรับได้ในหลายแพลตฟอร์มจากการเขียนเพียงครั้งเดียว แต่เมื่อรวมกับการทำระบบประมวลผลข้อมูลจำนวนมาก เช่น การคำนวณค่าเฉลี่ยของค่าจากอุปกรณ์ ทำให้ประสิทธิภาพในการทำงานของแอปพลิเคชันลดลงเป็นอย่างมาก ดังนั้น จึงนำ Foreign Function Interface เข้ามาแก้ไขปัญหาดังกล่าว โดยนำอีกภาษา ก็คือภาษา Rust [18] ซึ่งมีจุดเด่นในเรื่องของความเร็ว ไม่ใช้หน่วยความจำเยอะ และใช้พื้นที่น้อย มาช่วยในการทำการคำนวณเหล่านี้ ทำให้การแสดงผลในหน้าการวิเคราะห์ของแอปพลิเคชันสามารถแสดงผลและคำนวณไปพร้อมกันได้

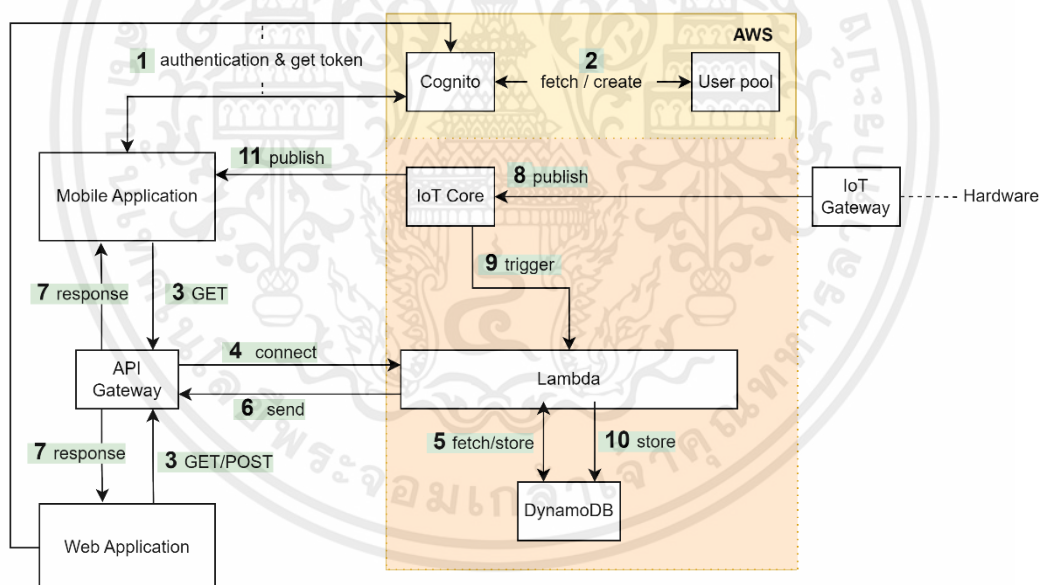
บทที่ 3

การออกแบบโครงงาน

3.1 การออกแบบภาพรวม

ภาพรวมของระบบที่ออกแบบประกอบไปด้วย 3 องค์ประกอบ คือ แอปพลิเคชัน AWS และ ส่วนของฮาร์ดแวร์ ในโครงงานนี้จะเน้นไปที่ส่วนของแอปพลิเคชันและ AWS โดยมีส่วนของการทำงานได้แก่ การยืนยันตัวตน การรับ-ส่งข้อมูล การบันทึกข้อมูลลงฐานข้อมูล

แอปพลิเคชันเชื่อมต่อบริการต่างๆของ AWS ได้แก่ ใช้ Cognito ในการทำระบบยืนยันตัวตน, รับ-ส่งข้อมูลกับฮาร์ดแวร์ผ่าน IoT Core และ เรียกใช้ฟังก์ชัน Lambda ผ่าน API Gateway ในแต่ละบริการของ AWS จะเชื่อมต่อกันผ่าน Lambda ซึ่งทำหน้าที่เป็นตัวกลางเปรียบเสมือนเป็น เซิร์ฟเวอร์ โดยบริการ Lambda จะเชื่อมต่อกับ DynamoDB ในการดึงและบันทึกข้อมูล และเชื่อมต่อกับ IoT Gateway เพื่อส่งต่อค่าจากฮาร์ดแวร์ หรือส่งค่าไปฮาร์ดแวร์



รูปที่ 3.1 ภาพรวมของระบบการทำงานแอปพลิเคชัน

3.2 สภาพแวดล้อมสำหรับการพัฒนา

ในการพัฒนาแอปพลิเคชันบนมือถือ จะต้องมีส่วนต่างๆดังต่อไปนี้

- Flutter (Channel stable, version 3.7.6 หรือสูงกว่า)
- Rust
- Node.js
- Windows (10 หรือ สูงกว่า)
- Android toolchain สำหรับการพัฒนาอุปกรณ์แอนดรอยด์ (SDK version 33.0.2 + NDK; Native)
- Android Studio
- VS Code
- อุปกรณ์ที่ใช้ในการรัน เช่น อุปกรณ์จำลอง หรือ มือถือ

โปรแกรมที่ใช้ในการเขียนโค้ด คือ Android Studio และ VS Code โดยจะเน้นใช้ไปที่ VS Code ส่วน Android Studio จะใช้ในการลง Android SDK / NDK ผ่าน SDK manager รวมถึงการสร้าง/จัดการไฟล์บนเครื่อง/ลบ อุปกรณ์จำลอง

3.3 สถาปัตยกรรมโครงสร้างของแอปพลิเคชัน

โครงสร้างในแอปพลิเคชันมีลักษณะเป็น widget tree โครงสร้างแบบต้นไม้คล้ายกับลักษณะของโพลีเดอริบนคอมพิวเตอร์ที่สามารถซ้อนกันได้ โดย parent แรกคือตัวแอปพลิเคชัน และไล่ลงไปเรื่อยๆ เป็น widget ต่างๆ ซึ่งคือหน้าของการใช้งาน แต่พบว่าเมื่อนำไปใช้แล้ว เกิดปัญหาที่ทำให้เมื่อต้องส่งค่าข้ามหน้าต่างต้องคอยเพิ่มพารามิเตอร์รองรับค่านั้นด้วย ซึ่งทำให้ใน 1 คลาสมีจำนวนพารามิเตอร์ที่เยอะเกินไป ดังนั้นจึงนำโมเดล BloC (Business Logic Component) [19] มาใช้ แบ่งออกเป็น 2 รูปแบบ คือ Cubit และ BloC โดยข้อแตกต่างคือ Cubit จะไม่มีการเก็บสถานะของ widget หมายถึง เมื่อมีการสร้าง widget นั้นๆใหม่จะทำให้ค่าที่หน้านั้นกลับไปเป็นแบบที่ค่าเริ่มต้นตั้งไว้ ในขณะที่ BloC จะมีการเก็บสถานะและสามารถเปลี่ยนแปลงได้ตลอด โมเดลนี้ถูกนำมาใช้ที่จุดมี Stateful widget เช่น หน้า dashboard หลัก เป็นต้น โมเดล BloC ช่วยในการแยกกระบวนการสร้าง widget กับการจัดการข้อมูลออกจากกัน ทำให้ 2 กระบวนการนี้ทำงานควบคู่กันไปแบบขนานจึงทำให้การทำงานของแอปพลิเคชันรวดเร็วยิ่งขึ้น

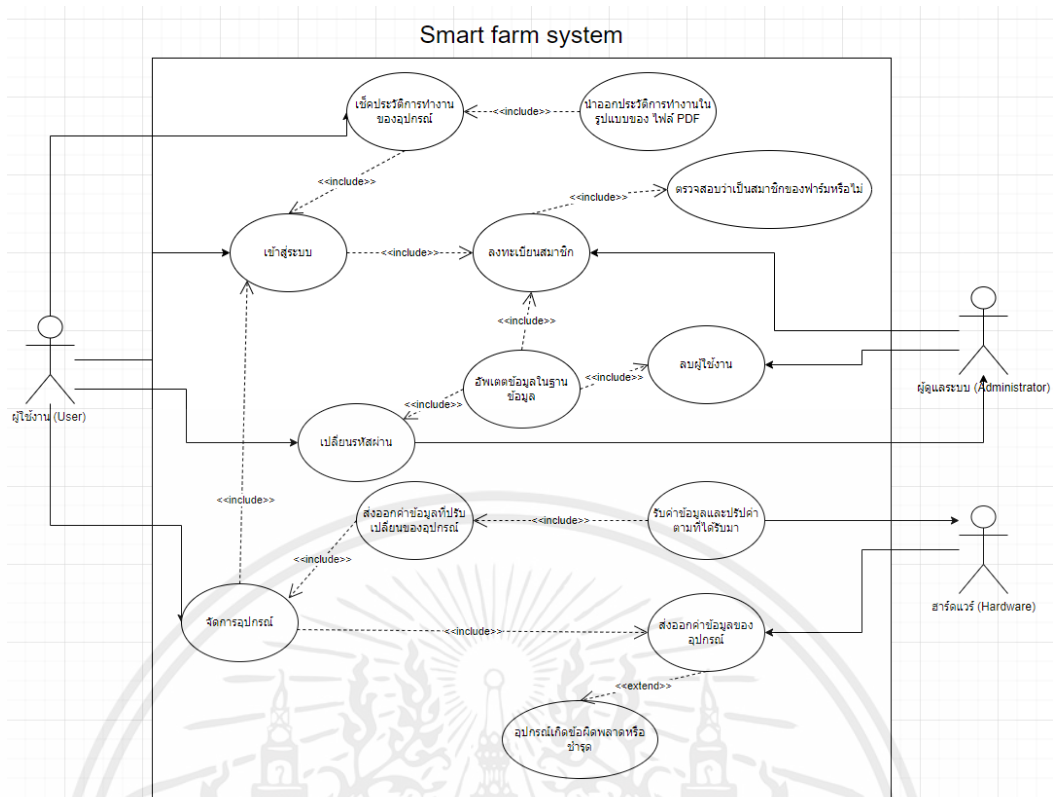
3.4 การออกแบบแอปพลิเคชันโดยรวม

การออกแบบแอปพลิเคชัน Smart Farm ประกอบด้วย 3 องค์ประกอบที่ทำงานร่วมกันดังต่อไปนี้

1. ผู้ใช้งาน (User)
2. ผู้ดูแลระบบ (Administrator)
3. ฮาร์ดแวร์ (Hardware)

ทางผู้จัดทำใช้ AWS Cloud ในการรับและส่งข้อมูลของทั้ง 3 องค์ประกอบ โดยจะมีหลักการคือ ผู้ดูแลระบบ (Administrator) ทำหน้าที่ในการเพิ่มหรือลบ ผู้ใช้งาน (User) เข้าไปในฟาร์มโดยการกำหนด บทบาท (role) ให้ หลังจากนั้น ผู้ใช้งาน (User) สามารถเข้าใช้งานได้แล้ว จะมีหน้าที่ในการจัดการ ฮาร์ดแวร์ (Hardware) ต่างๆที่อยู่ภายในฟาร์มที่ได้รับมอบหมายให้ดูแล และ ฮาร์ดแวร์ (Hardware) มีหน้าที่ในการส่งค่าข้อมูลเข้ามาในแอปพลิเคชันเพื่อให้ ผู้ใช้งาน (User) จัดการต่อไปฟังก์ชันต่างๆที่สามารถใช้งานได้มีดังนี้

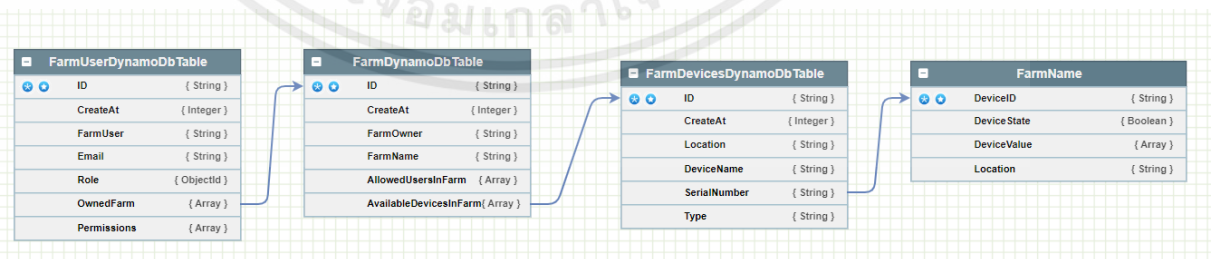
1. ผู้ใช้งาน (User)
 - 1.1 ดูข้อมูลของ ฮาร์ดแวร์ แสดงผลได้ 2 วิธีคือ รูปแบบตัวเลข และ รูปแบบกราฟ
 - 1.2 สามารถจัดการค่าข้อมูลต่างๆของ ฮาร์ดแวร์ ผ่านแอปพลิเคชัน
 - 1.3 สามารถเช็คประวัติการทำงานของ ฮาร์ดแวร์ ว่าเกิดข้อผิดพลาดขึ้นหรือป่าว
 - 1.4 สามารถนำออกประวัติการทำงาน หรือ ค่าของ ฮาร์ดแวร์ ในรูปแบบของไฟล์ PDF
2. ผู้ดูแลระบบ (Administrator)
 - 2.1 เพิ่มหรือลบ ผู้ใช้งานออกจากการใช้แอปพลิเคชันได้
 - 2.2 กำหนด บทบาทให้กับผู้ใช้งานในการดูแลฟาร์ม
 - 2.3 จัดการอุปกรณ์ทั้งหมดภายในฟาร์ม
3. ฮาร์ดแวร์ (Hardware)
 - 3.1 รับและส่งข้อมูลกับแอปพลิเคชัน



รูปที่ 3.2 แผนภาพกรณีการใช้งานของแอปพลิเคชัน

3.5 การออกแบบฐานข้อมูล

ฐานข้อมูลที่ได้ออกแบบไว้เป็น ฐานข้อมูลแบบ NoSQL ซึ่งไม่ได้มีการเชื่อมโยงกันของข้อมูลระหว่างตาราง แต่จะเชื่อมโยงกันผ่านการทำงานของฟังก์ชัน ฐานข้อมูลมีอยู่ 2 ส่วนหลักๆ คือ ส่วนที่ไม่เปลี่ยนแปลงบ่อย กับ ส่วนที่อัปเดตข้อมูลแบบตามเวลาจริง



รูปที่ 3.3 การเชื่อมต่อของฐานข้อมูล

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ส่วนที่ไม่เปลี่ยนแปลงบ่อประกอบไปด้วย 3 ตาราง ได้แก่ FarmUser, Farm, FarmDevices

1. FarmUser จะมีกุญแจหลักเป็น ID โดยที่ ID จะถูกสร้างเป็นแบบ UUIDv4 ซึ่งเป็นการสุ่ม และมีตัวแปร 6 อย่างคือ

- CreateAt บันทึกค่าเป็นตัวเลขซึ่งเป็นตัวบอกเวลาที่สร้างและบันทึกผู้ใช้โดยจะอยู่ในลักษณะของรูปแบบยูนิคซ์
- FarmUser เป็นชนิด String ใช้ในการบันทึกชื่อของผู้ใช้งาน
- Email เป็นชนิด String ใช้ในการบันทึกอีเมลล์ของผู้ใช้
- Role เป็นชนิด String ใช้ในการบ่งบอกถึงระดับของผู้ใช้ว่าอยู่ในระดับใด เช่น user, admin เป็นต้น
- OwnedFarm เป็น array ที่บันทึก String โดยแต่ละค่าจะเป็นค่าที่ถูกเข้ารหัสด้วยฐาน 64 ซึ่งค่าเหล่านี้เมื่อถอดรหัสออกมาแล้วจะได้เป็นชื่อของฟาร์มที่ผู้ใช้เป็นเจ้าของอยู่
- Permissions เป็น array ที่บันทึก String โดยจะบันทึกสิทธิการเข้าถึงของผู้ใช้ เช่น การเข้าถึงข้อมูลผู้ใช้ การยืนยันระดับของผู้ใช้ เป็นต้น

แนวคิดในการออกแบบในการเก็บข้อมูลของผู้ใช้งาน

ตารางนี้ถูกออกแบบมาเพื่อเก็บข้อมูลของผู้ใช้ต่างๆในแอปพลิเคชันทั้งบนเว็บและแอปพลิเคชันบนมือถือ โดยข้อมูลของตารางจะเก็บชื่อผู้ใช้ และอีเมลล์ รวมถึงว่าเป็นเจ้าของฟาร์มอะไรบ้าง โดยปกติอาจจะนำชื่อฟาร์มแยกเป็นชุดข้อมูลคนละชุดกัน แต่เนื่องจากว่าใน AWS DynamoDB สามารถเก็บข้อมูลในลักษณะของ array ได้ จึงทำให้ไม่ต้องแยกข้อมูลทีละชุด สมมติในกรณีที่ผู้ใช้มีจำนวนฟาร์มมากกว่า 1 ฟาร์ม จะนำชื่อฟาร์มต่างๆที่ถูกเข้ารหัสมาเพิ่มลงไป OwnedFarm ซึ่งเป็น array ที่สามารถเก็บ string ได้ ดังนั้นจึงทำให้การเพิ่มฟาร์มนั้นสามารถเพิ่มได้โดยไม่สนเรื่องของขนาด array ซึ่งในทำนองเดียวกัน Permissions ก็เป็น array เช่นกัน เพื่อให้ผู้ดูแลสามารถเพิ่มหรือลดสิทธิการเข้าถึงของผู้ใช้นั้นได้

FarmUserDynamoDbTable
+ ID: string
+ CreateAt: integer
+ FarmUser: string
+ Email: string
+ Role: string
+ OwnedFarm: array
+ Permissions: array

Example:

```
{
  "ID": "d957e981-bae0-4ba0-a3f3-52db9e38bab6",
  "CreateAt": 1664988278773,
  "Email": "pakintada@gmail.com",
  "FarmUser": "admin1",
  "OwnedFarm": [
    "MEFkRmFybTE=",
    "MEFkRmFybTI="
  ],
  "Permissions": [
    "WaitForFarmsDataConfirmation",
    "WaitForRoleConfirmation"
  ],
  "Role": "user"
}
```

รูปที่ 3.4 แอททริบิวต์ของตาราง FarmUser และตัวอย่างข้อมูลในรูปแบบ JSON

2. Farm จะมีกุญแจหลักเป็น ID โดยที่ ID จะถูกเข้ารหัสด้วยฐาน 64 ซึ่งในการเข้าถึงตารางนี้ได้สามารถทำได้โดยการนำไอดีมาจาก OwnedFarm ของตาราง FarmUser ในตารางนี้มีตัวแปรอีก 5 ตัวแปรได้แก่

- CreateAt บันทึกค่าเป็นตัวเลขซึ่งเป็นตัวบอกเวลาที่สร้างและบันทึกผู้ใช้โดยจะอยู่ในลักษณะของรูปแบบยูนิคซ์
- FarmOwner เป็นชนิด String ใช้เพื่อบ่งบอกว่าใครเป็นเจ้าของฟาร์มนี้
- FarmName เป็นชนิด String บันทึกชื่อของฟาร์ม
- AllowedUsersInFarm เป็น array ที่บันทึก String ซึ่งเป็นชื่อของผู้ใช้ที่มีสิทธิเข้าถึงข้อมูลของฟาร์มนี้
- AvailableDevicesInFarm เป็น array ที่บันทึก String ของเลขฐาน 16 โดยแต่ละค่าเป็นค่าที่ถูกสุ่มขึ้นมา โดยค่าเหล่านี้คือกุญแจหลักประจำตัวอุปกรณ์ ซึ่งบ่งบอกถึงอุปกรณ์ที่มีอยู่ในฟาร์มนั้นๆ

แนวคิดในการออกแบบในการเก็บข้อมูลของฟาร์ม

ตารางนี้ออกแบบมาเพื่อเก็บข้อมูลของแต่ละฟาร์ม โดยมีจุดสำคัญหลักๆ คือ เจ้าของฟาร์ม ผู้ที่สามารถเข้าถึงฟาร์มนี้ได้ และอุปกรณ์ที่มีอยู่ในฟาร์มนี้ ใช้หลักการเดียวกันกับตารางเก็บข้อมูลผู้ใช้ คือ ผู้ที่เข้าถึงฟาร์มนี้ได้ และ อุปกรณ์ภายในฟาร์มจะถูกเก็บในรูปแบบของ array ที่เก็บข้อมูลประเภท string ทำให้ไม่ต้องไปเพิ่มเป็นข้อมูลชุดใหม่ ในการเข้าถึงข้อมูลของแต่ละฟาร์มในตารางนี้ต้องมีไอดีซึ่งเป็นกุญแจหลักเพื่อใช้ในการดึงข้อมูล โดยสามารถนำไอดีดังกล่าวมาจาก OwnedFarm ในตารางผู้ใช้ได้

FarmDynamoDbTable	Example:
+ ID: string	{
+ CreateAt: integer	"ID": "MEFkRmFybTE=",
+ FarmOwner: string	"AllowedUsersInFarm": [
+ FarmName: string	"admin1"
+ AllowedUsersInFarm: array	],
+ AvailableDevicesInFarm: array	"AvailableDevicesInFarm": [
	"de967e15a70dda1d",
	"da877e4d7add5f34"
	],
	"CreateAt": 1664988838355,
	"FarmName": "AdFarm1",
	"FarmOwner": "admin1"
	}

รูปที่ 3.5 แอททริบิวต์ของตาราง Farm และตัวอย่างข้อมูลในรูปแบบ JSON

3. FarmDevice มีกุญแจหลักเป็น ID ซึ่งเป็นเลขฐาน 16 ซึ่งใช้ในการหาอุปกรณ์ภายในตารางนี้ เท่านั้นโดยสามารถนำมาจากตัวแปร AvailableDevicesInFarm ในตาราง Farm ซึ่งในตารางนี้มี ตัวแปรอีก 5 ตัวแปร ได้แก่

- CreateAt บันทึกค่าเป็นตัวเลขซึ่งเป็นตัวบอกเวลาที่สร้างและบันทึกอุปกรณ์ โดยจะอยู่ใน ลักษณะของรูปแบบยูนิคซ์
- Location เป็นประเภท String ซึ่งบอกถึงตำแหน่งที่ตั้งของอุปกรณ์นี้ ณ ปัจจุบัน
- DeviceName เป็นประเภท String ใช้ในการบันทึกชื่อของอุปกรณ์นั้นๆ
- SerialNumber เป็นประเภท String ใช้ในการบันทึกหมายเลขประจำตัวของอุปกรณ์
- Type เป็นประเภท String บันทึกประเภทของอุปกรณ์

แนวคิดในการออกแบบในการเก็บข้อมูลของอุปกรณ์

ตารางนี้ออกแบบมาเพื่อเก็บข้อมูลของแต่ละอุปกรณ์ โดยเก็บหมายเลขประจำตัวของ อุปกรณ์ ประเภท และสถานที่ที่อุปกรณ์ติดตั้งอยู่ โดยมีไอดีของแต่ละข้อมูลที่จะนำมาใช้เป็นกุญแจ หลักซึ่งไม่เกี่ยวข้องกับไอดีหรือหมายเลขประจำตัวของอุปกรณ์ โดยไอดีของอุปกรณ์สามารถ นำมาใช้ได้ด้วยการดึงมาจาก AvailableDevicesInFarm ในตารางฟาร์ม ดังนั้นเพื่อความปลอดภัย ของการดึงข้อมูล เมื่อต้องการดูรายละเอียดของข้อมูลอุปกรณ์ต้องเริ่มต้นที่ตารางผู้ใช้ เพื่อได้ไอดี ของฟาร์มที่ต้องการ ต่อการดึงข้อมูลของฟาร์มจากตารางฟาร์ม และนำไอดีของอุปกรณ์ที่มีในฟาร์ม มาเป็นไอดีหลักเพื่อดูรายละเอียดของอุปกรณ์ที่ตารางนี้

FarmDeviceDynamoDbTable
+ ID: string
+ CreateAt: integer
+ Location: string
+ DeviceName: string
+ SerialNumber: string
+ type: string

Example:

```
{
  "ID": "da877e4d7add5f34",
  "CreateAt": 1665047471186,
  "DeviceName": "PhuRasperry-02",
  "Location": "AdFarm1",
  "SerialNumber": "RP-TEST2",
  "Type": "MySmartIoTDevice"
}
```

รูปที่ 3.6 แอททริบิวต์ของตาราง FarmDevice และตัวอย่างข้อมูลในรูปแบบ JSON

ส่วนที่เปลี่ยนแปลงตามเวลาจริง จะใช้ชื่อตารางเป็นชื่อของฟาร์มนั้นๆ ซึ่งมีจุดประสงค์เพื่อการบันทึกค่าจากเซนเซอร์ตามเวลาจริง มีกฎหลักเป็น DeviceID ซึ่งก็คือ SerialNumber จากตาราง FarmDevice และมีตัวแปรอีก 3 ตัวแปรได้แก่

- DeviceState เป็นประเภท boolean ทำหน้าที่บอกสถานะของอุปกรณ์นั้นๆว่าพร้อมในการส่งค่าหรือไม่ หากมีค่าเป็นจริง หมายถึงอุปกรณ์นั้นสามารถส่งค่าต่างๆรวมถึงค่าที่วัดได้ แต่ในทางตรงกันข้าม หากมีค่าเป็นเท็จ หมายถึงอุปกรณ์นั้นจะส่งค่าต่างๆยกเว้นค่าที่วัดได้
- DeviceValue เป็นประเภท array ซึ่งบันทึก json ที่ส่งมาจากอุปกรณ์โดยแต่ละ json จะมีลักษณะดังนี้
 - o {"State": "", "TimeStamp": "", "Value": ""}
- Location เป็นประเภท String ซึ่งบอกถึงตำแหน่งที่ตั้งของอุปกรณ์นี้ ณ ปัจจุบัน

Example:

```
{
  "DeviceID": "RP-TEST2",
  "DeviceState": true,
  "DeviceValue": [
    {
      "State": true,
      "TimeStamp": 1666511953415,
      "Value": "30"
    },...
  ],
  "Location": "AdFarm1-west"
}
```

FarmName
+ DeviceID: string
+ DeviceState: boolean
+ DeviceValue: array
+ Location: string

รูปที่ 3.7 แอททริบิวต์ของตาราง FarmName และตัวอย่างข้อมูลในรูปแบบ JSON

3.6 การออกแบบหน้าต่างแอปพลิเคชัน

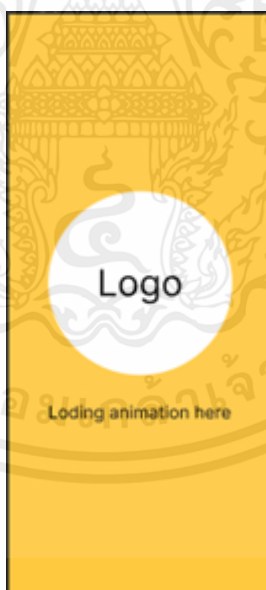
ในการพัฒนาแอปพลิเคชัน ในการทำให้ผู้ใช้งานทำความเข้าใจในการใช้งานแอปพลิเคชันได้อย่างง่ายดาย และให้ออกมามีความสวยงาม จะแบ่งเป็น 2 อย่าง คือการออกแบบ UX (User Experience) และ UI (User Interface)

1. UX (User Experience) คือ การออกแบบเพื่อให้ผู้ใช้งานแอปพลิเคชันใช้งานได้ง่าย ไม่สับสนในการใช้งาน โดยทางผู้จัดทำได้ออกแบบเพื่อให้ ผู้ใช้งานสามารถใช้งานได้ง่ายโดยการนำการจัดการอุปกรณ์ต่างๆ มาไว้ที่หน้าหลักเพื่อให้ง่ายต่อการ จัดการ

2. UI (User Interface) คือ ส่วนที่เชื่อมต่อระหว่างผู้ใช้งานกับระบบ หรือก็คือ หน้าตาและความสวยงามทั้งหมดของแอปพลิเคชันที่ผู้ใช้งานสามารถมองเห็นได้ โดยทางผู้จัดทำได้ใช้แอปพลิเคชัน Figma ในการออกแบบ

แอปพลิเคชัน Figma คือ แอปพลิเคชันที่ใช้ในการออกแบบหน้าต่าง ของ เว็บไซต์ และ แอปพลิเคชันเบื้องต้น โดยทางผู้จัดทำได้ออกแบบไว้ 3 ส่วนหลักๆ คือ

- Splash screen คือ หน้าที่ใช้ในการโชว์ระหว่างที่แอปพลิเคชันกำลังเปิดตัวโดยจะแสดงเพียงไม่กี่วินาทีให้ นิยมใส่เพียง โลโก้ หรือ ชื่อของแอปพลิเคชัน โดยได้ทำการออกแบบไว้ดังรูปที่ 3.4.1



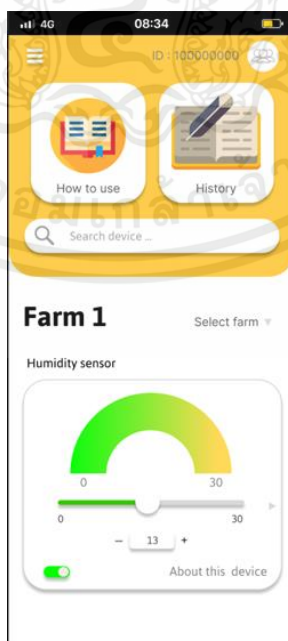
รูปที่ 3.8 หน้า Splash screen ที่ได้ทำการออกแบบไว้

- Login คือ หน้าที่ใช้การเข้าใช้งานแอปพลิเคชัน ผ่านรหัสผ่านที่ได้จาก ผู้ดูแลระบบ (Administrator) โดยได้ทำการออกแบบไว้ดังรูปที่ 3.4.2



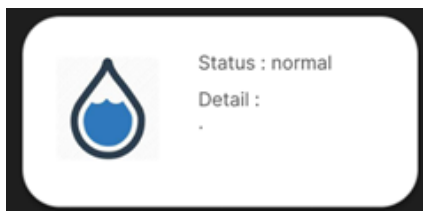
รูปที่ 3.9 หน้า login screen ที่ได้ทำการออกแบบไว้

- หน้าหลัก คือ หน้า que แสดงฟังก์ชันต่างๆของแอปพลิเคชัน โดยได้ทำการออกแบบไว้ดังรูปที่ 3.4.3 ,รูปที่ 3.4.4 และ รูปที่ 3.4.5



รูปที่ 3.10 หน้าหลักของแอปพลิเคชัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

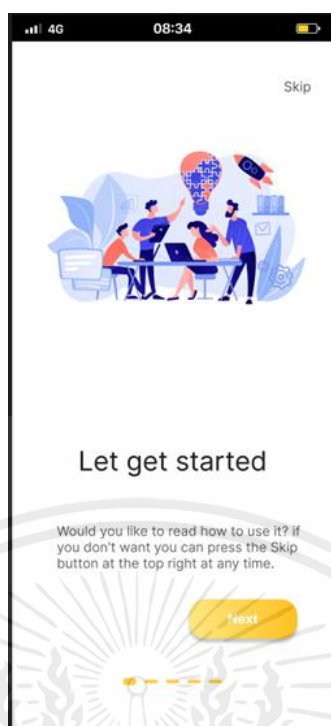


รูปที่ 3.11 popup เมื่อกด about this device



รูปที่ 3.12 หน้าในการดูประวัติของอุปกรณ์

- Onboarding คือ หน้าทีแสดงเมื่อผู้ใช้งานเข้าใช้งานแอปพลิเคชันเป็นครั้งแรก โดยจะเป็นการสอนระบบต่างๆภายในแอปพลิเคชัน โดยได้ทำการออกแบบไว้ดังรูปที่ 3.4.6 ,รูปที่ 3.4.7 ,รูปที่ 3.4.8 และ รูปที่ 3.4.9



รูปที่ 3.13 หน้าสอนใช้งานหน้าที่ 1

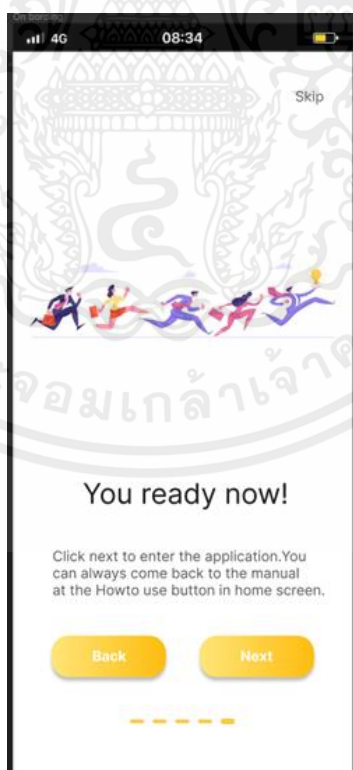


รูปที่ 3.14 หน้าสอนใช้งานหน้าที่ 2

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.15 หน้าสอนใช้งานหน้าที่ 3



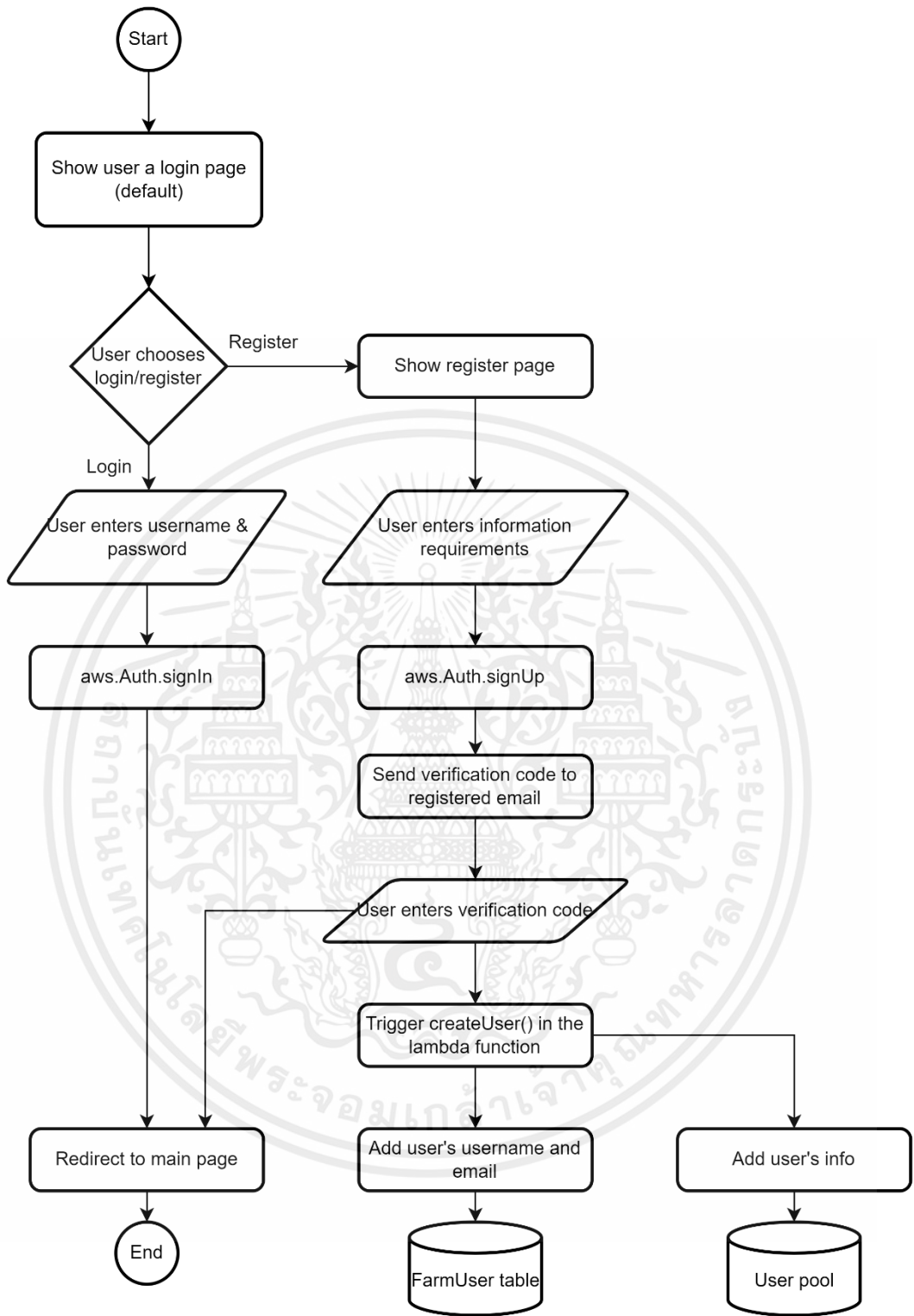
รูปที่ 3.16 หน้าสอนใช้งานหน้าที่ 4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.7 การออกแบบบริการต่างๆ

3.7.1 การเข้าสู่ระบบและการลงทะเบียน

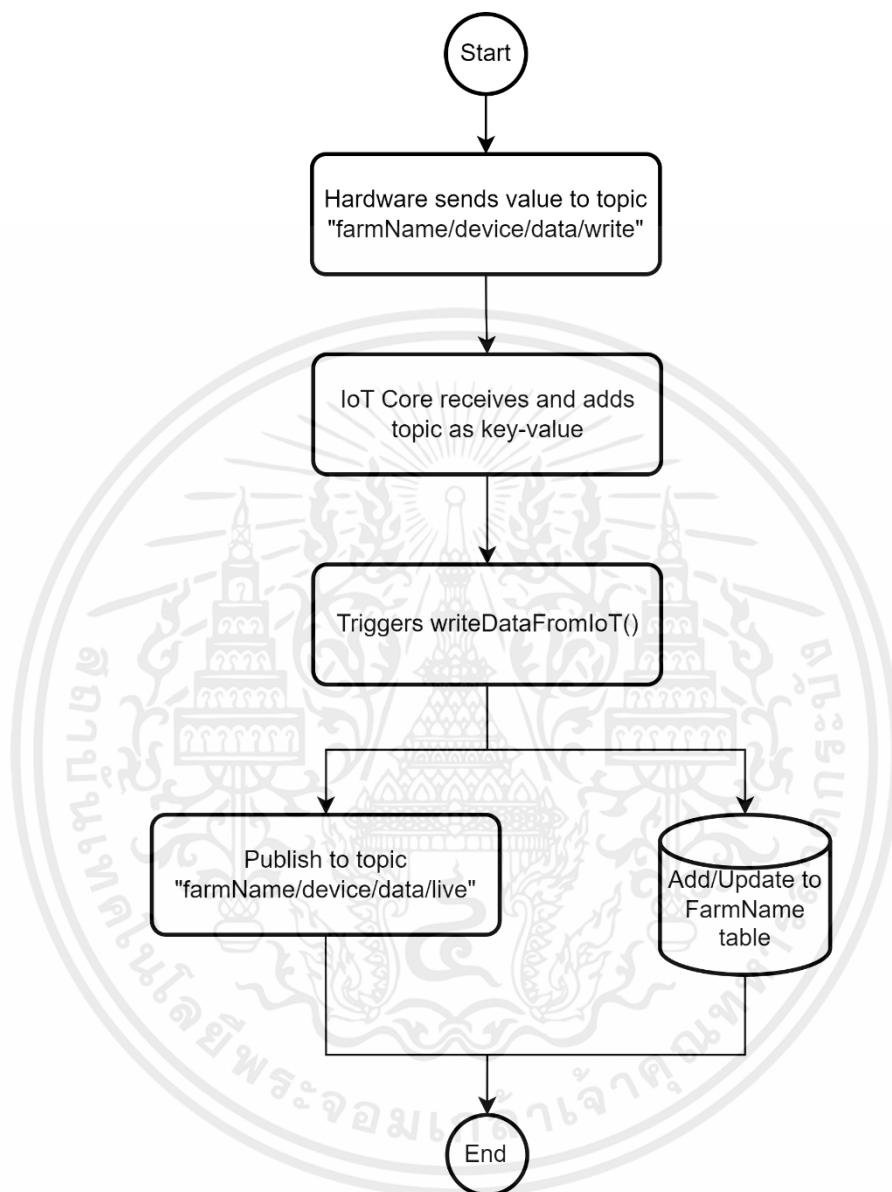
เมื่อผู้ใช้หน้าแอปพลิเคชันมา จะพบกับหน้า log in ผู้ใช้สามารถเลือกที่จะเข้าสู่ระบบได้ในหน้านี้ หรือถ้าหากยังไม่มีบัญชี สามารถกดที่ “Don’t have an account?” เพื่อเข้าสู่หน้าลงทะเบียน ในหน้าเข้าสู่ระบบ ผู้ใช้จะต้องใส่ชื่อผู้ใช้งานกับรหัสผ่านที่ได้ลงทะเบียนไว้ ถ้าหากลืมรหัสผ่าน สามารถกดที่ “forget password” เพื่อทำการตั้งรหัสผ่านใหม่ในการเข้าได้ เมื่อเข้าสู่ระบบสำเร็จ ก็จะเปลี่ยนไปที่หน้าหลัก ถ้าหากผู้ใช้เลือกที่จะลงทะเบียน ผู้ใช้จะต้องใส่ชื่อผู้ใช้ รหัสผ่าน ยืนยันรหัสผ่าน และอีเมลล์ เพื่อทำการลงทะเบียน แล้วเมื่อกดยืนยันก็จะเปลี่ยนเป็นหน้ายืนยัน โดยจะส่งโค้ดยืนยันไปที่อีเมลล์ ถ้าผู้ใช้กรอกโค้ดและยืนยันแล้ว ก็จะเปลี่ยนหน้าเป็นหน้าหลัก ในขณะเดียวกันที่เซิร์ฟเวอร์ก็จะส่งให้ฟังก์ชัน `lambda createUser()` ทำงาน ซึ่งจะทำหน้าที่บันทึกข้อมูลผู้ใช้บางส่วนลงในตาราง `FarmUser` และยังบันทึกข้อมูลผู้ใช้ที่ลงทะเบียนไว้ทั้งหมดลงใน `user pool` เพื่อใช้ในการเข้าสู่ระบบในครั้งถัดไป



รูปที่ 3.17 ฟังก์ชันของการเข้าสู่ระบบและลงทะเบียน

3.7.2 การส่งข้อมูลจากฮาร์ดแวร์และการปล่อยข้อมูลที่ให้ผู้ติดตาม

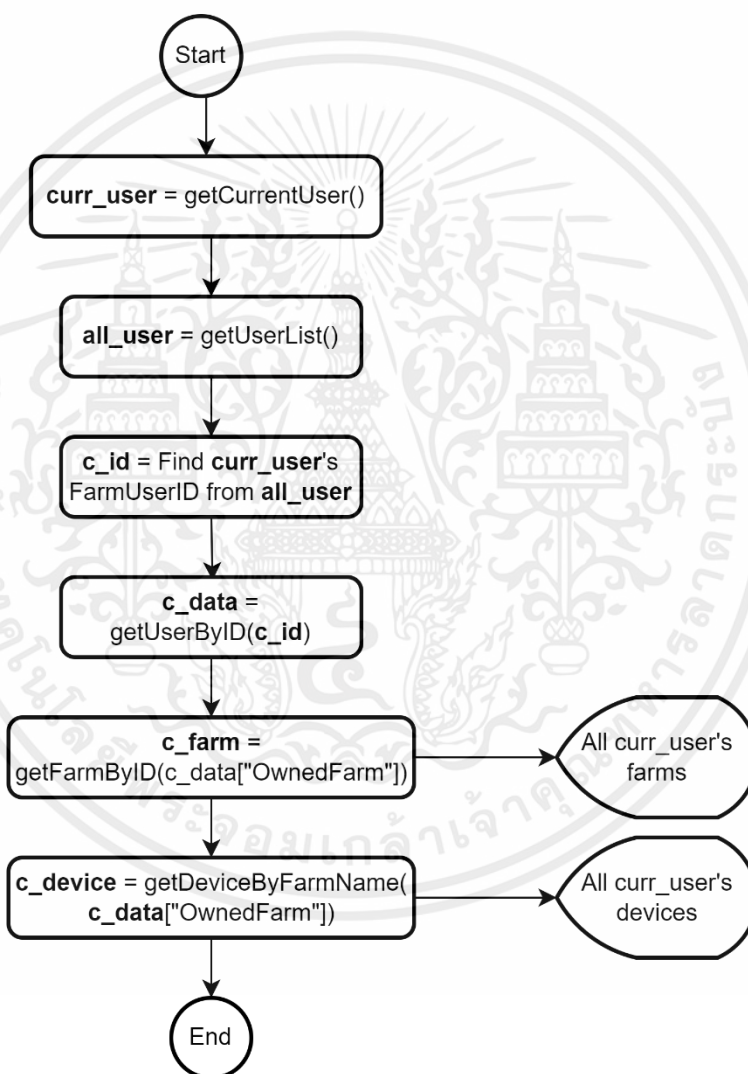
ฟังก์ชันนี้ถูกออกแบบมาเพื่อให้ฮาร์ดแวร์สามารถบันทึกหรืออัปเดตข้อมูลลงในฐานข้อมูลได้ ในขณะเดียวกัน เมื่อทำสำเร็จก็จะทำการเผยแพร่ข้อมูลนั้นให้ผู้ติดตามทันที



รูปที่ 3.18 ผังงานของการส่งข้อมูลจากฮาร์ดแวร์และการปล่อยข้อมูลที่ให้ผู้ติดตาม

3.7.3 การแสดงข้อมูลผู้ใช้ ฟาร์ม และอุปกรณ์

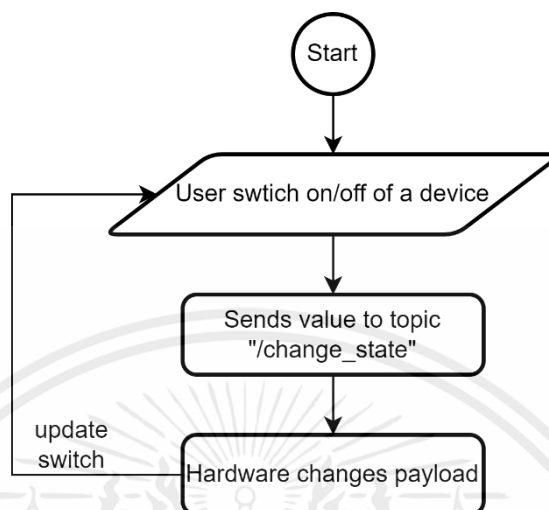
ฟังก์ชันที่ใช้ในการดึงข้อมูลมีอยู่ 5 ฟังก์ชัน โดยจะเริ่มจาก `getUserList` ซึ่งเป็นการดึงข้อมูลในตาราง `user` มาก่อน ในฟังก์ชันนี้ได้ออกแบบไว้ว่า ให้สามารถดึงได้แค่ 2 รายละเอียดคือ ไอดี UUIDv4 ที่เป็นกุญแจหลัก และชื่อ `username` จากนั้นนำไอดีของผู้ใช้ที่ต้องการไปเรียกฟังก์ชัน `getUserByID` เพื่อให้ได้รายละเอียดที่เหลือมา ในรายละเอียด `OwnedFarm` จะมีรายการไอดีฟาร์มอยู่ จากนั้นก็เรียกฟังก์ชัน `getFarmByID` โดยใส่ไอดีฟาร์มเข้าไปก็จะได้รายละเอียดฟาร์มทั้งหมดของฟาร์มนั้นมา นำชื่อฟาร์มมาเรียกฟังก์ชัน `getDevicesByFarmName` ซึ่งจะส่งอุปกรณ์ทั้งหมดที่มีรายละเอียดชื่อฟาร์มตรงกับชื่อฟาร์มที่เป็นเป็นอินพุต



รูปที่ 3.19 ผังงานของการแสดงข้อมูลผู้ใช้ ฟาร์ม และอุปกรณ์

3.7.4 การสั่งปิด-เปิดการรับค่าจากฮาร์ดแวร์

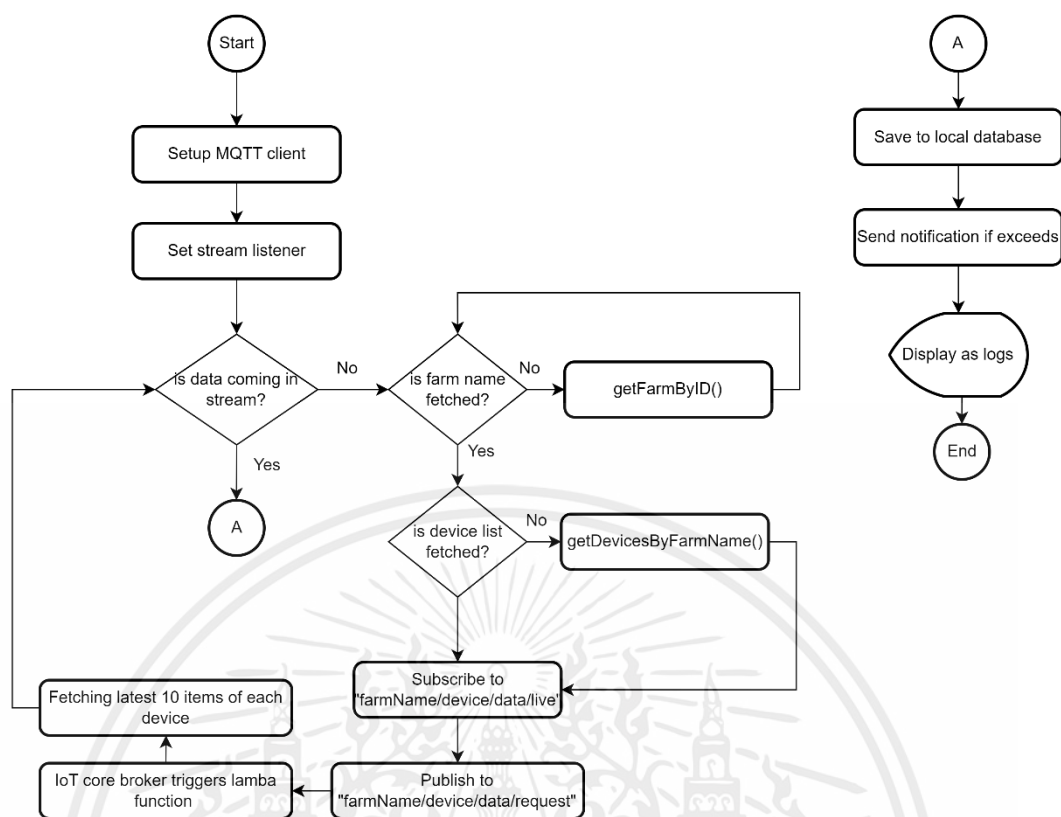
ฟังก์ชันนี้ถูกออกแบบมาเพื่อให้สามารถปิดหรือเปิดการรับค่าจากฮาร์ดแวร์ได้ โดยผู้ใช้สามารถเลือกกดที่สวิตช์ของแต่ละอุปกรณ์เพื่อปิดหรือเปิดได้



รูปที่ 3.20 ผังงานของการสั่งปิด-เปิดการรับค่าจากฮาร์ดแวร์

3.7.5 การแสดงผลค่าของอุปกรณ์และประวัติ

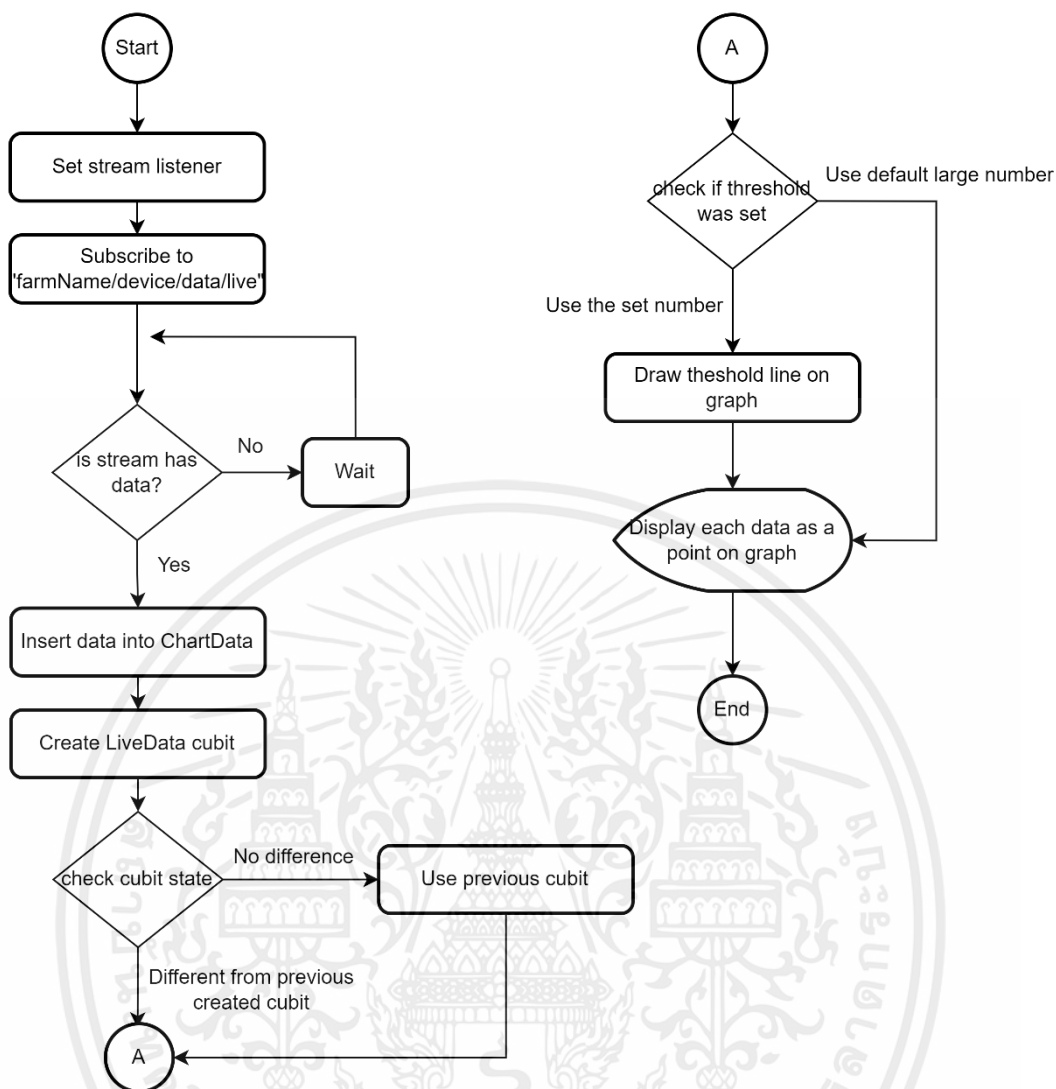
กระบวนการทำงานนี้จะเริ่มต้นที่การเชื่อมต่อผู้ใช้งานกับ endpoint ของ IoT Core เพื่อที่จะใช้โปรโตคอล MQTT ในการรับ-ส่งข้อมูล ในการรับข้อมูลได้อย่างต่อเนื่องไม่จำเป็นต้องทำการติดตามใหม่ทุกครั้งเหมือนการร้องขอแบบ REST โดยทำการตั้งตัวรับฟังข้อมูลประเภทสตรีมภายในแอป เมื่อข้อมูลเข้ามาในสตรีมของการติดตามหัวข้อนั้น จะทำการตรวจสอบว่ามีข้อมูลจริงหรือไม่ ถ้าใช่ ก็จะแสดงผลในรูปแบบของค่าจากอุปกรณ์นั้น ซึ่งรวมถึงการบันทึกลงในประวัติข้อมูลด้วย ในทางตรงกันข้าม ถ้าหากไม่มีข้อมูลเข้ามาในสตรีม ซึ่งอาจหมายถึงการที่ยังไม่ได้เริ่มติดตามหัวข้อ ทำการตรวจสอบว่ามีชื่อฟาร์มที่ดึงมาเก็บแล้วหรือไม่ ถ้ายังไม่มีให้ทำการเรียกฟังก์ชัน `getFarmByID` ในลักษณะเดียวกันกับ 3.5.3 เมื่อมีชื่อฟาร์มที่ต้องการแล้ว ทำการตรวจสอบว่ามีรายชื่ออุปกรณ์อยู่หรือไม่ ถ้ายังไม่มีให้ทำการเรียกฟังก์ชัน `getDevicesByFarmName` จากนั้นนำชื่อฟาร์มกับหมายเลขประจำอุปกรณ์มารวมกันกับ `"/data/live"` เพื่อสร้างหัวข้อในการติดตาม และรวมกับ `"data/request"` เพื่อเผยแพร่คำร้องขอแล้วดึงข้อมูลล่าสุด 10 อันมาทำเป็นประวัติเริ่มต้นของแต่ละอุปกรณ์ โดยในระหว่างที่ข้อมูลของอุปกรณ์เข้ามานั้น จะทำการบันทึกลงไปที่ตัวเครื่องของผู้ใช้งานเป็นรูปแบบฐานข้อมูล SQLite ทำให้ในครั้งถัดไปจะไม่ต้องดึงข้อมูลจาก AWS DynamoDB อีกครั้ง แต่จะดึงจากฐานข้อมูลนี้แทนและหลังจากนั้นเมื่อมีข้อมูลจากอุปกรณ์เข้ามาก็จะบันทึกลงฐานข้อมูลนี้โดยอัตโนมัติ ในระหว่างที่มีการบันทึกก็ตรวจสอบกับค่า Threshold ที่ผู้ใช้งานจะตั้งค่าไว้ ซึ่งถ้าพบว่าเกินหรืออยู่ในระยะใกล้กับค่าที่ตั้ง ก็ทำการแจ้งเตือน (ต่อที่ 3.5.10)



รูปที่ 3.21 ผังงานของการแสดงผลค่าของอุปกรณ์และประวัติ

3.7.6 การแสดงผลในรูปแบบกราฟ

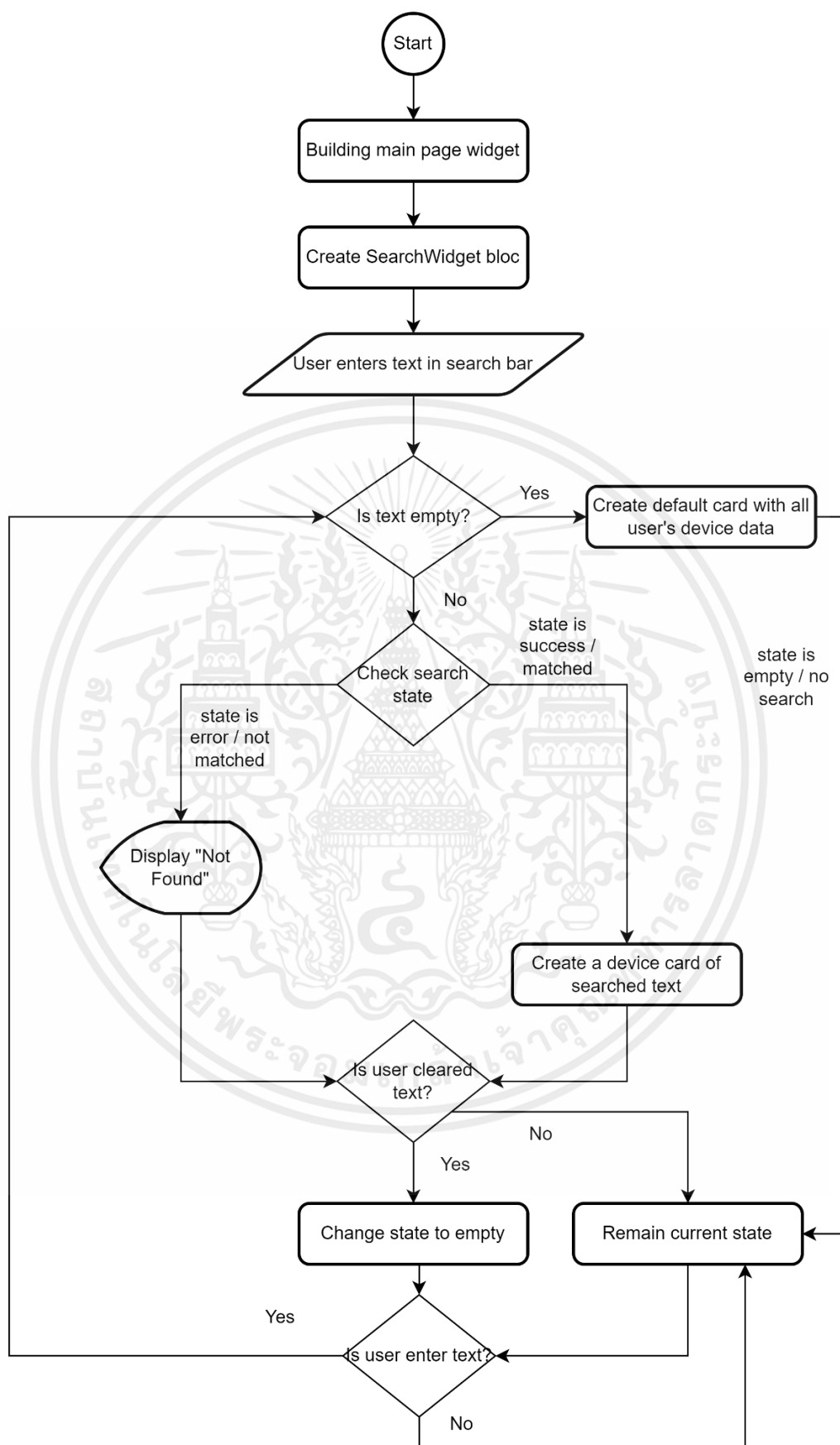
ผู้ใช้งานสามารถเข้าถึงหน้ากราฟของแต่ละอุปกรณ์ได้ผ่านการกดที่อุปกรณ์ที่แสดงผลบนหน้าหลัก หลังจากนั้นจะปรากฏหน้ารายละเอียดของอุปกรณ์นั้น โดยที่ 3 ส่วน ได้แก่ รายละเอียดต่างๆ ของอุปกรณ์ หน้ากราฟแบบตามเวลาจริง และการตั้งค่าการแจ้งเตือน ในหน้ากราฟจะมีการทำงานจะมีลักษณะคล้ายคลึงกับ 3.5.5 แต่จะแตกต่างกันที่ไม่ต้องทำการเผยแพร่ไปที่ “data/request” และการติดตามหัวข้อ “data/live” จะติดตามแค่ 1 อุปกรณ์เท่านั้น เมื่อได้ข้อมูลจากสตรีมมาแล้ว ก็จะทำให้กลายเป็นวัตถุ ChartData ที่เก็บข้อมูลสำหรับการทำกราฟ จากนั้นนำข้อมูลกับวัตถุไปสร้างเป็น cubit ซึ่งทำหน้าที่เป็นตัวตรวจสอบสถานะของวัตถุภายในแอปพลิเคชัน โดยทำการตรวจสอบว่าถ้าหากสถานะหรือค่าเหมือนกันกับค่าก่อนหน้า ก็จะไม่เปลี่ยนเป็นค่าใหม่ ถ้าไม่เหมือนเดิมก็เปลี่ยนเป็นค่าใหม่ จากนั้นค่าที่ได้ก็จะนำไปสร้างเป็นจุดบนกราฟ ในขณะเดียวกันถ้าหากพบว่าการตั้งค่าการแจ้งเตือนไว้ก็จะนำไปแสดงบนกราฟด้วย โดยจะแสดงเป็นเส้นตรงชื่อว่า Threshold นอกจากนี้จะสามารถสังเกตเส้นแนวโน้ม เส้นบอกค่าสูงสุดและต่ำสุดได้อีกด้วย



รูปที่ 3.22 ผังงานของการแสดงผลในรูปแบบกราฟ

3.7.7 การค้นหาอุปกรณ์ในหน้าหลัก

จุดมุ่งหมายของกระบวนการนี้คือการที่ผู้ใช้สามารถค้นหาอุปกรณ์เพื่อให้แสดงผลที่หน้าจอหลักได้ การทำงานของกระบวนการนี้จะเริ่มที่การสร้างวิดเจ็ตบนหน้าหลัก ในขณะที่สร้างแถบการค้นหา ก็จะสร้าง SearchWidget bloc ขึ้นมา โดยมีหน้าที่ในการดูสถานะของวิดเจ็ตและข้อมูล แตกต่างจาก cubit ที่สถานะของ bloc จะเปลี่ยนตามเหตุการณ์

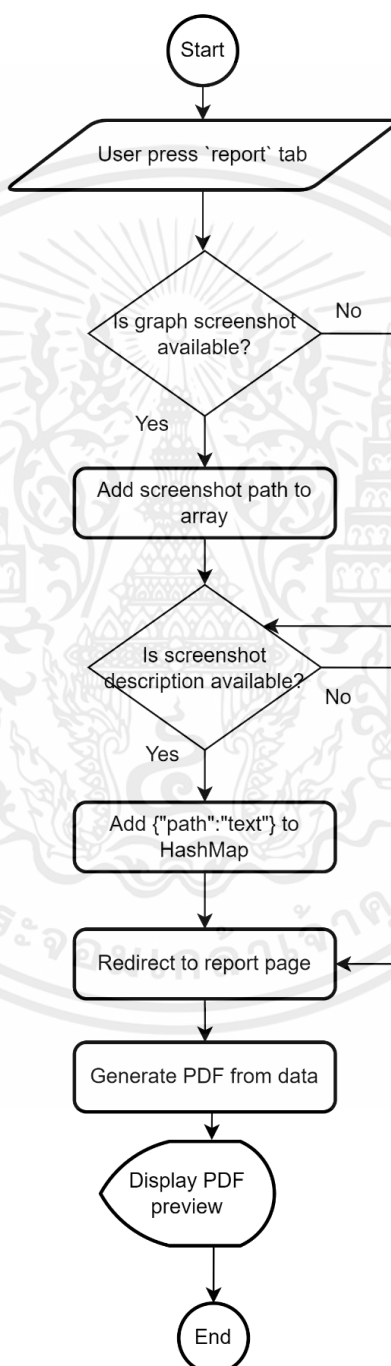


รูปที่ 3.23 ผังงานของการค้นหาอุปกรณ์ในหน้าหลัก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.7.8 การสร้างไฟล์รายงานผลแบบ PDF จากค่าของอุปกรณ์

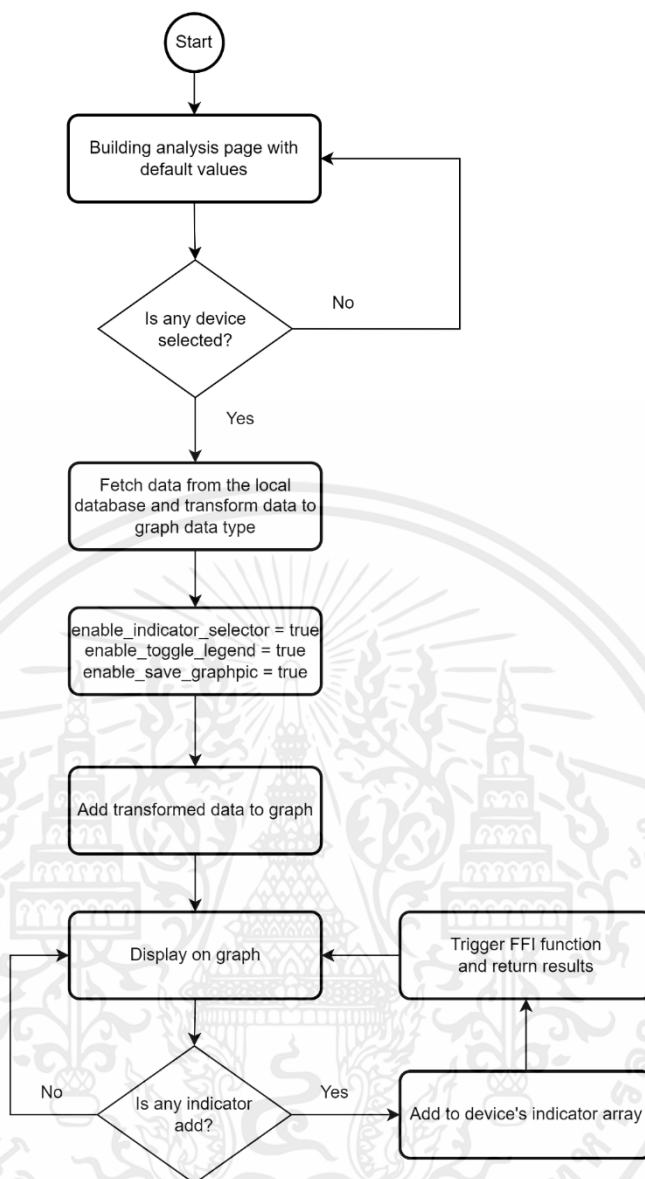
เมื่อผู้ใช้กดที่ปุ่ม รายงาน (Report) ก็จะทำข้อมูลล่าสุดที่มีอยู่ในประวัติปัจจุบันไปสร้างเป็นไฟล์รายงานในนามสกุลไฟล์ PDF โดยจะแสดงผลว่าเป็นรายงานของฟาร์มใด สร้างโดยใคร เมื่อไหร่ มีตารางชื่ออุปกรณ์และประเภท สถานที่ตั้งอุปกรณ์ และตารางแสดงค่าเฉลี่ยของแต่ละอุปกรณ์ นอกจากนี้ผู้ใช้อาจสามารถเพิ่มรูปภาพจากกราฟในหน้าวิเคราะห์พร้อมข้อความได้



รูปที่ 3.24 ผังงานของการสร้างไฟล์รายงานผลแบบ PDF จากค่าของอุปกรณ์

3.7.9 การวิเคราะห์ค่าจากข้อมูลอุปกรณ์

ผู้ใช้สามารถกดที่ปุ่ม Analysis เพื่อเข้าสู่หน้าการวิเคราะห์ได้ โดยประกอบไปด้วย 2 แท็บ คือ ส่วนที่แสดงเป็นหน้าวิเคราะห์ที่ผู้ใช้สามารถป้อนตัวชี้วัดเข้าไปเพื่อแสดงเส้นของตัวชี้ในกราฟของ อุปกรณ์ที่กำหนด และส่วนที่เข้าไปยังการสร้างรายงาน เมื่อผู้ใช้เข้าสู่หน้านี้ จะปรากฏ 3 ส่วน คือ ส่วนของกราฟ ตัวเลือกอุปกรณ์พร้อมค่าเฉลี่ยโดยรวม และปุ่มเพิ่มตัวชี้วัด ในการเข้าครั้งแรก จะไม่ปรากฏ หน้ากราฟและการเพิ่มตัวชี้วัดให้ผู้ใช้จนกว่าจะมีการเลือกอุปกรณ์ที่ตัวเลือกอุปกรณ์ โดยเมื่อกดเลือก อุปกรณ์แล้ว ก็จะแสดงผลข้อมูลทั้งหมดของอุปกรณ์นั้นที่ถูกเก็บไว้บนมือถือของผู้ใช้มาแสดงเป็นกราฟ เมื่อข้อมูลเข้ามาแล้วจะปรากฏขึ้นมาอีก 2 ปุ่มคือ ปุ่มปิด/เปิดแถบอธิบายเส้นกราฟ และปุ่มบันทึก กราฟ ซึ่งผู้ใช้สามารถกดบันทึกแล้วรูปภาพจะไปปรากฏในแท็บหน้ารายงานเพื่อรอกข้อมูลเพิ่มเติม ได้ ถ้าหากผู้ใช้ต้องการเพิ่มตัวชี้วัด ก็สามารถกดที่ปุ่มเพิ่มด้านล่าง โดยจะปรากฏหน้าต่างเลือกตัวชี้วัด ซึ่งใน รุ่นปัจจุบันจะมีอยู่ด้วยกัน 2 ตัว ได้แก่ SMA (Simple Moving Average) และ EMA (Exponential Moving Average) โดยเมื่อกดเลือกเสร็จสิ้น ก็จะปรากฏเส้นของตัวชี้วัดบนหน้ากราฟ และตัวเลือกของ ตัวชี้วัดนั้นซึ่งมีค่าเริ่มต้นอยู่ที่ 5 และสามารถเลือกเปลี่ยนได้ตาม 4 ตัวเลือกที่ปรากฏ หากผู้ใช้ไม่ ต้องการตัวชี้วัดในหน้ากราฟ สามารถกดค้างที่ตัวเลือกเพื่อนำออกได้



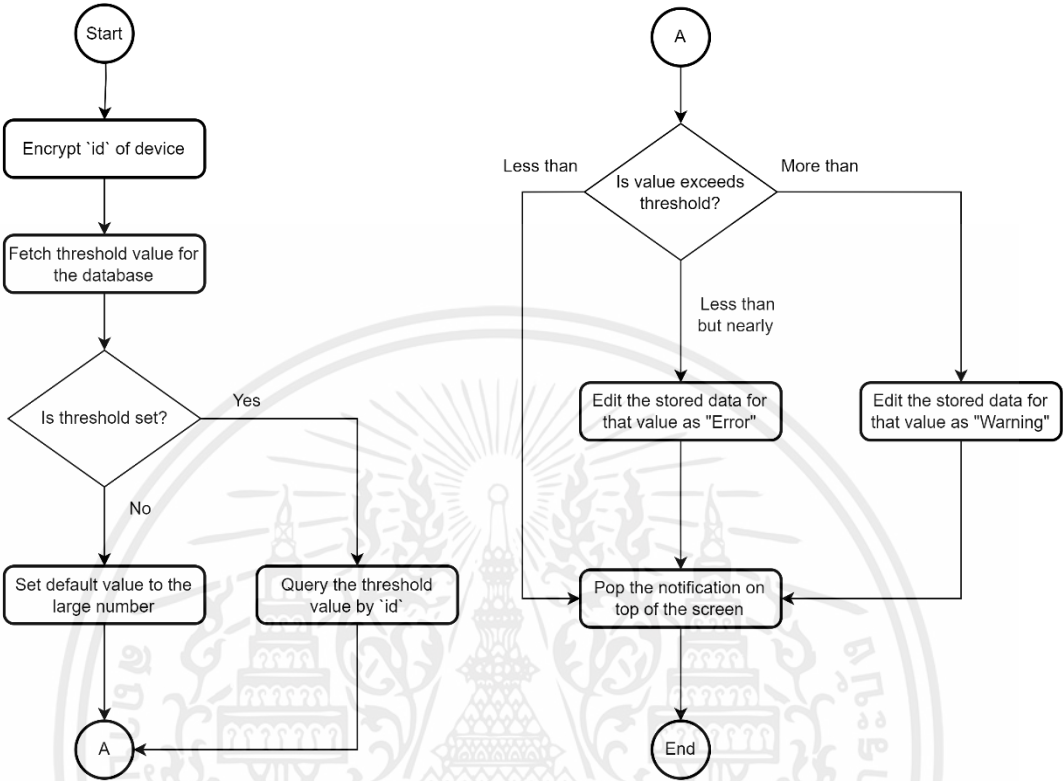
รูปที่ 3.25 ผังงานของการวิเคราะห์ค่าจากข้อมูลอุปกรณ์

3.7.10 การแจ้งเตือนผู้ใช้เมื่อค่าจากอุปกรณ์เกินจุดที่ตั้งค่าไว้

จาก 3.5.5 หลังจากมีการบันทึกข้อมูลลงในฐานข้อมูลบนเครื่อง ก็จะนำค่าเหล่านั้นมาตรวจสอบกับค่า Threshold ที่ผู้ใช้อาจจะตั้งค่าเอาไว้ โดยเมื่อพบว่ามีค่าใกล้กับค่าที่กำหนดน้อยกว่า ก็ จะทำการแจ้งเตือนผู้ใช้งานผ่านการแจ้งเตือนที่ด้านบนของหน้าจอว่าอยู่ในระดับ “Warning” หรือถ้าค่า นั้นเกินกว่าค่าที่กำหนด ก็จะแจ้งเตือนว่าอยู่ในระดับ “Error”

ในการตั้งค่า Threshold นี้จะทำได้ในหน้าของอุปกรณ์ซึ่งอยู่ในส่วนด้านล่างสุด ผู้ใช้สามารถ กรอกตัวเลขที่ต้องการลงไปในช่องได้ และกดบันทึกเพื่อทำการบันทึกค่าเหล่านี้ลงไปพื้นฐานข้อมูลค่า

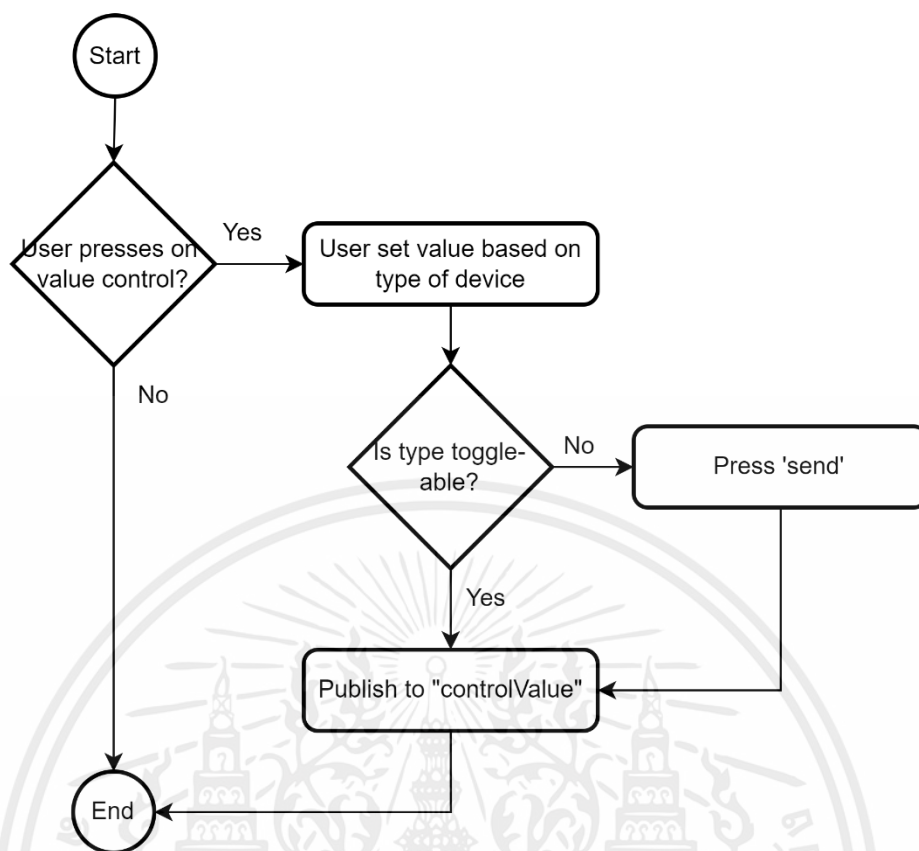
Threshold และหลังจากบันทึกแล้วค่าเหล่านี้ไปแสดงผลที่หน้ากราฟในหน้าของรายละเอียดอุปกรณ์ด้วย



รูปที่ 3.26 ผังงานของการแจ้งเตือนผู้ใช้เมื่อค่าจากอุปกรณ์เกินจุดที่ตั้งค่าไว้

3.7.11 การส่งค่าไปควบคุมอุปกรณ์

การควบคุมอุปกรณ์จะสามารถควบคุมได้ 2 วิธี คือ การปิด-เปิด (จาก 3.7.4) และ การควบคุมเป็นค่า การควบคุมค่าจะเป็นการสั่งควบคุมอุปกรณ์นั้นด้วยค่าตามประเภทของอุปกรณ์ ทั้งนี้ การควบคุมนี้จะไม่ีผลทันทีหลังการกดปุ่มส่งค่า เนื่องจากต้องตรวจสอบสถานะของอุปกรณ์กับฐานข้อมูลด้วย เพื่อป้องกันการส่งค่าซ้ำ



รูปที่ 3.27 ฟังก์ชันของการส่งค่าไปควบคุมอุปกรณ์

บทที่ 4

ผลการดำเนินงาน

การดำเนินการพัฒนาแอปพลิเคชันในครั้งนี้ มีผลการดำเนินงานและรายละเอียดขั้นตอนต่างๆ ดังต่อไปนี้

- 1. รายละเอียดการดำเนินการ
- 2. ขั้นตอนในการทำงาน
- 3. การใช้งานภายในแอปพลิเคชัน

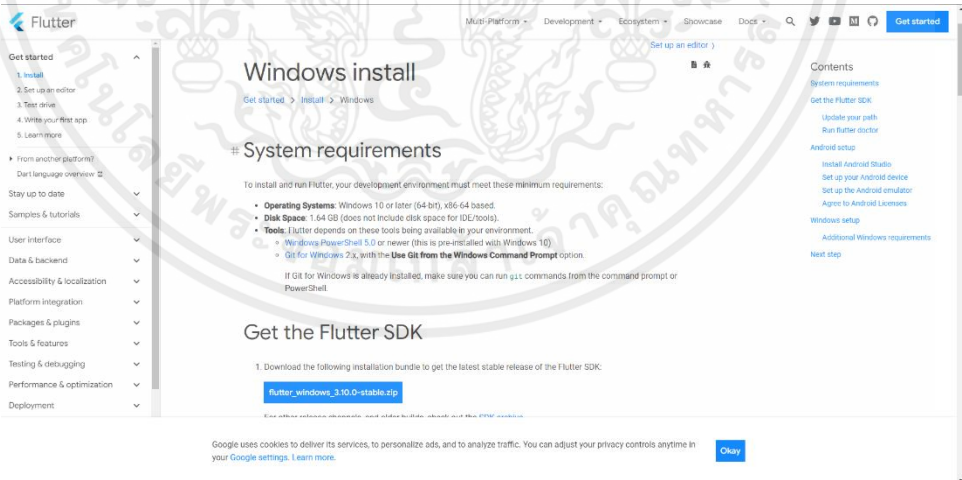
4.1 รายละเอียดการดำเนินการ

การติดตั้งภาษาที่ใช้ จะมีทั้งหมด 3 ส่วนด้วยกัน คือ Flutter, Rust และ Node.js สำหรับการ ทำ Serverless

4.1.1 การติดตั้ง Flutter และ Android studio

1. การติดตั้ง Flutter SDK

1.1. ติดตั้ง Flutter SDK โดยดาวน์โหลดได้ที่ [Windows install | Flutter](#)



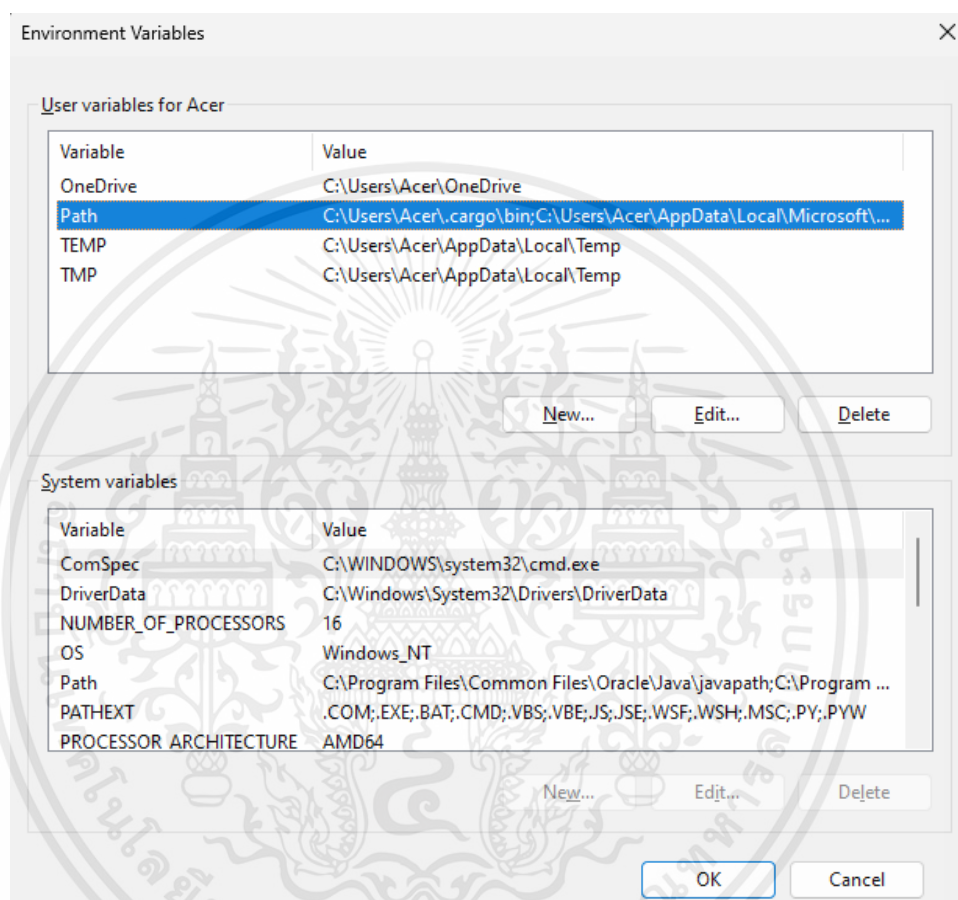
รูปที่ 4.1 เว็บไซต์ในการติดตั้ง Flutter

1.2. นำไฟล์ที่ดาวน์โหลดเสร็จแล้วแตกไฟล์และนำไปใส่ใน folder path ใหม่ เท่านั้น ห้ามวางใน C:\Program Files\ ตัวอย่างการสร้าง เช่น C:\src\flutter

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมีการดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1.3. สามารถดาวน์โหลดผ่าน Window power shell โดยใช้คำสั่งดังนี้ `git clone https://github.com/flutter/flutter.git -b stable`

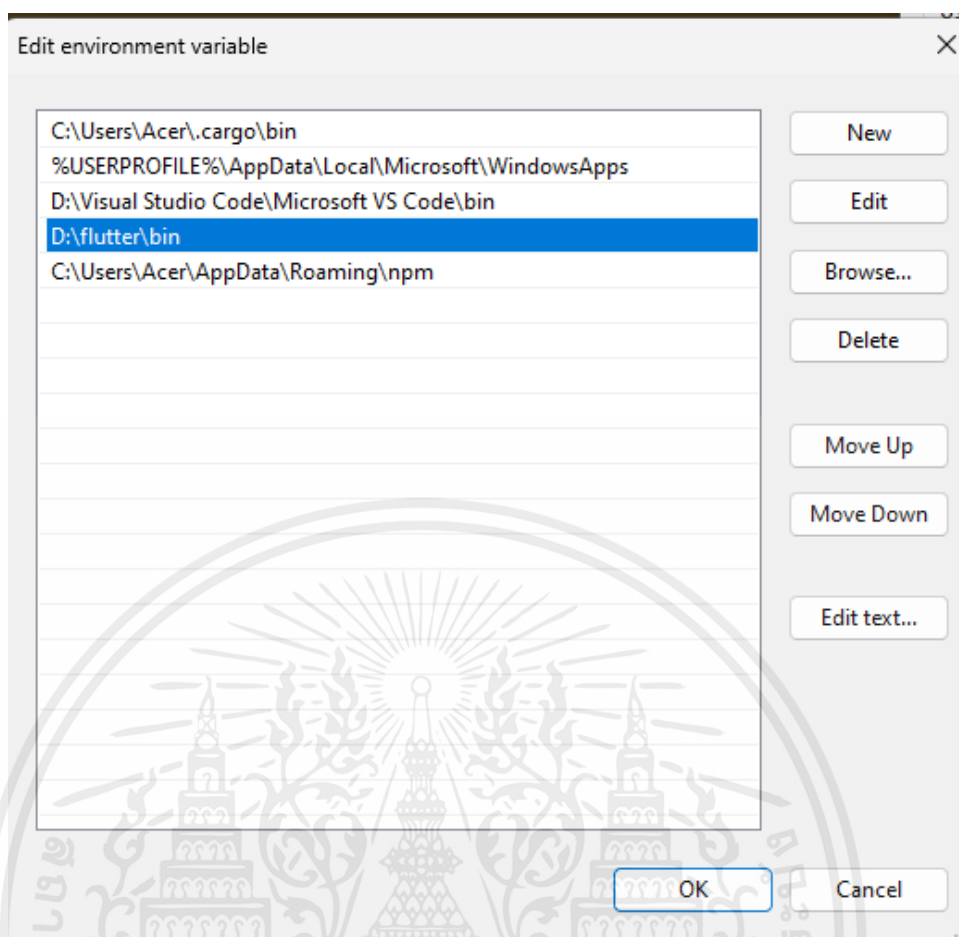
1.4 เมื่อสร้าง path ใหม่เสร็จแล้ว ให้เข้าไปที่ Settings ใส่คำว่า Edit environment variables for your account ในช่องค้นหาแล้ว กดเลือกที่ Advanced กดปุ่ม Environment Variables



รูปที่ 4.2 หน้าต่างในการเพิ่ม path

1.5 ไปที่ User variables คลิกที่ Path แล้วกดปุ่ม Edit... จากนั้น กดปุ่ม New

1.6 ไปที่โฟลเดอร์ตามข้อ 2 ที่สร้างขึ้นใหม่ กดเข้าไปจนเจอ flutter\bin แล้วนำ path ส่วนนั้นมาวาง แล้วกด OK



รูปที่ 4.3 การเพิ่ม path ของ flutter

1.7 ในการเช็คสถานะการติดตั้งของ flutter ให้เปิด Windows PowerShell แล้วใส่คำสั่งดังนี้
C:\src\flutter>flutter doctor

```
Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.7.12, on Microsoft Windows [Version 10.0.22621.1702], locale th-TH)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 32.1.0-rc1)
[✓] Chrome - develop for the web
[✓] Visual Studio - develop for Windows (Visual Studio Community 2022 17.2.2)
[✓] Android Studio (version 2021.2)
[✓] Connected device (3 available)
[✓] HTTP Host Availability

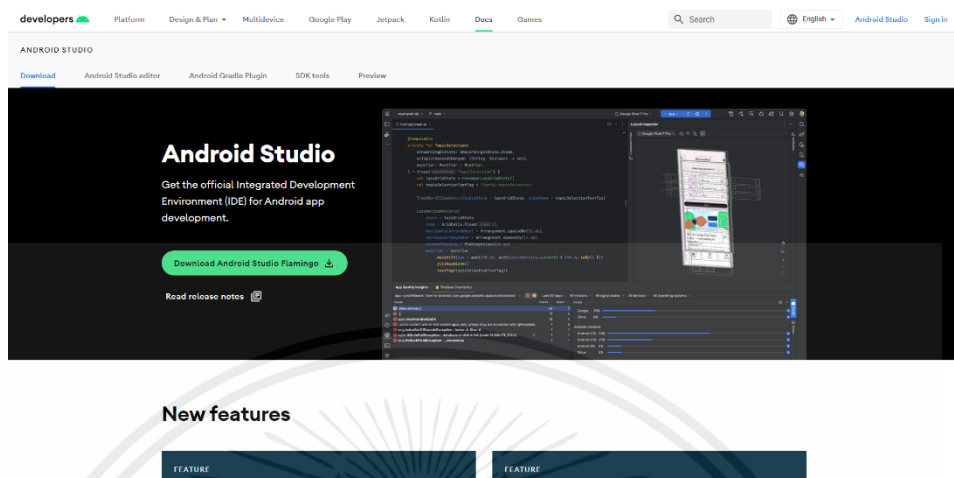
• No issues found!
```

รูปที่ 4.4 การตรวจสอบสถานะการติดตั้งของ Flutter

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. ติดตั้ง Android Studio

2.1 ติดตั้ง Android Studio โดยดาวน์โหลดได้ที่ [Download Android Studio & App Tools - Android Developers](#)

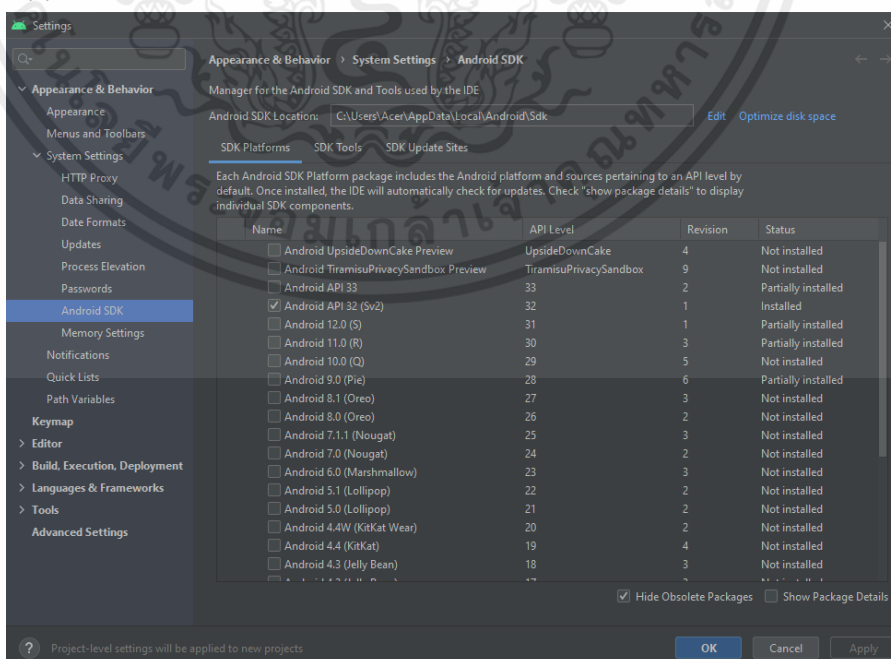


รูปที่ 4.5 เว็บไซต์ในการติดตั้ง Android studio

2.2 ในการติดตั้ง Android Studio ให้สมบูรณ์จำเป็นที่จะต้องติดตั้ง Android toolchain จะแบ่งเป็น 2 ส่วน คือ การติดตั้ง Android SDK และ Android Licens

2.2.1 ติดตั้ง Android SDK

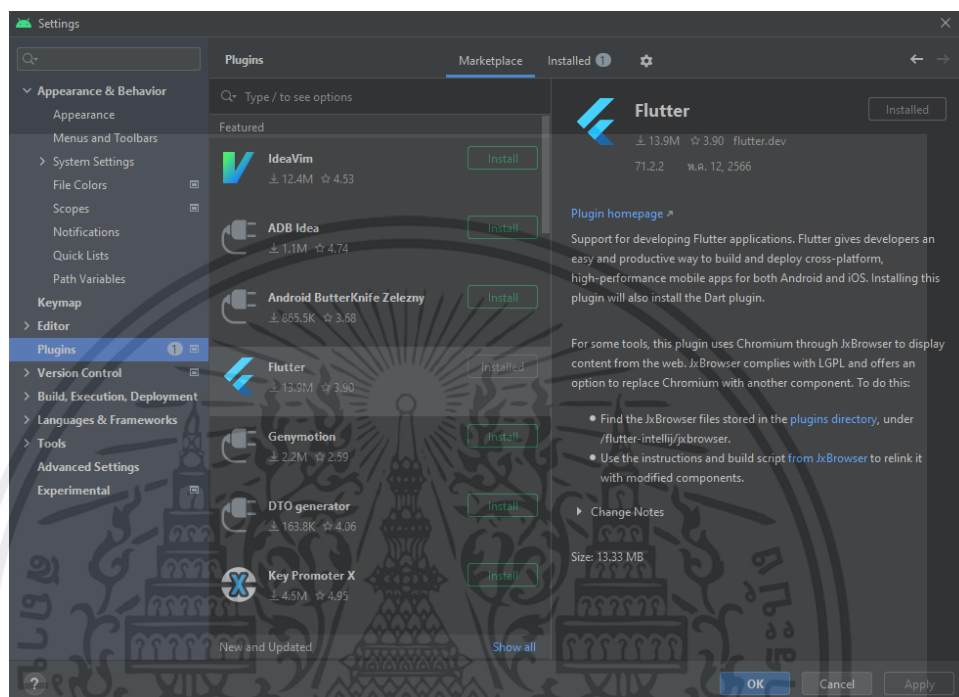
2.22.1 เปิด Android Studio แล้ว ไปที่ Configure > SDK Manager
ตรง Appearance & Behavior กดที่ System Settings > Android SDK



รูปที่ 4.6 หน้าต่างในการดาวน์โหลด Android SDK

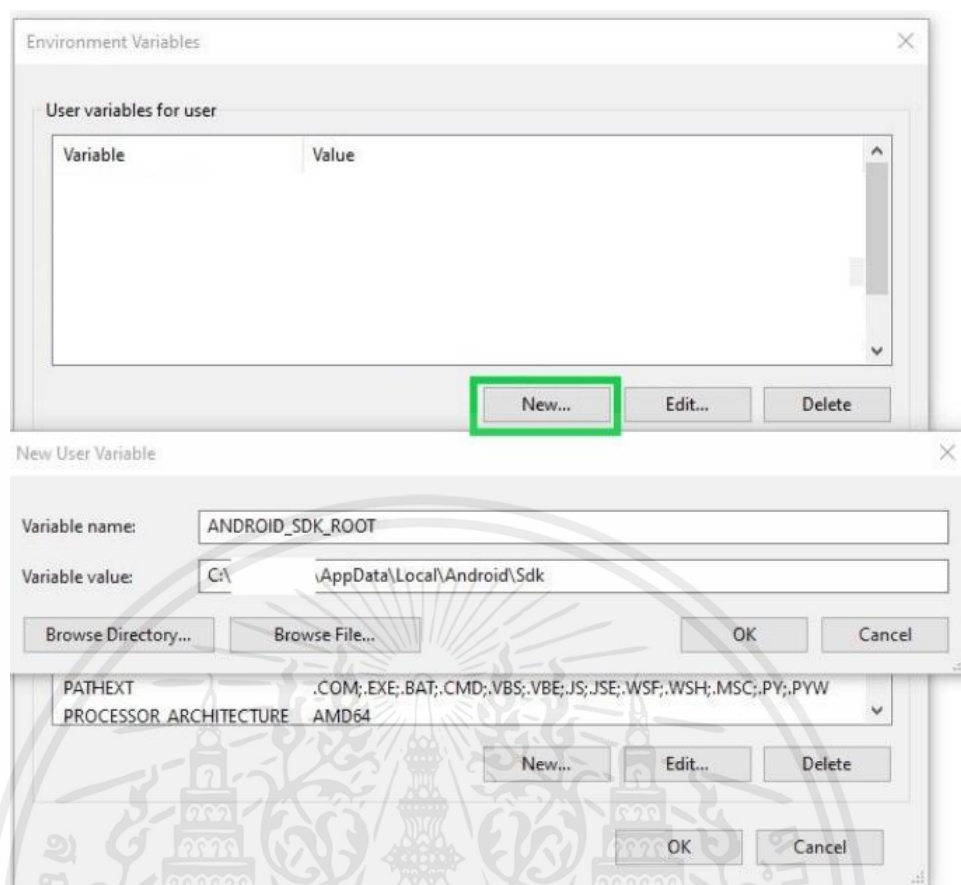
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เลือกเวอร์ชันที่ต้องการดาวน์โหลด โดยสามารถกดที่ Check Box และ Copy path ตรง Android SDK Location เอาไว้ ตรง Plugins ให้หาปลั๊กอินที่ชื่อ Flutter แล้วกด Install หลังจากนั้น โปรแกรมจะแจ้งว่า ต้องดาวน์โหลด ปลั๊กอิน Flutter จำเป็นต้องดาวน์โหลด Dart ด้วย กดดาวน์โหลดต่อไป



รูปที่ 4.7 หน้าต่างดาวน์โหลด plugin ของ flutter ภายใน android studio

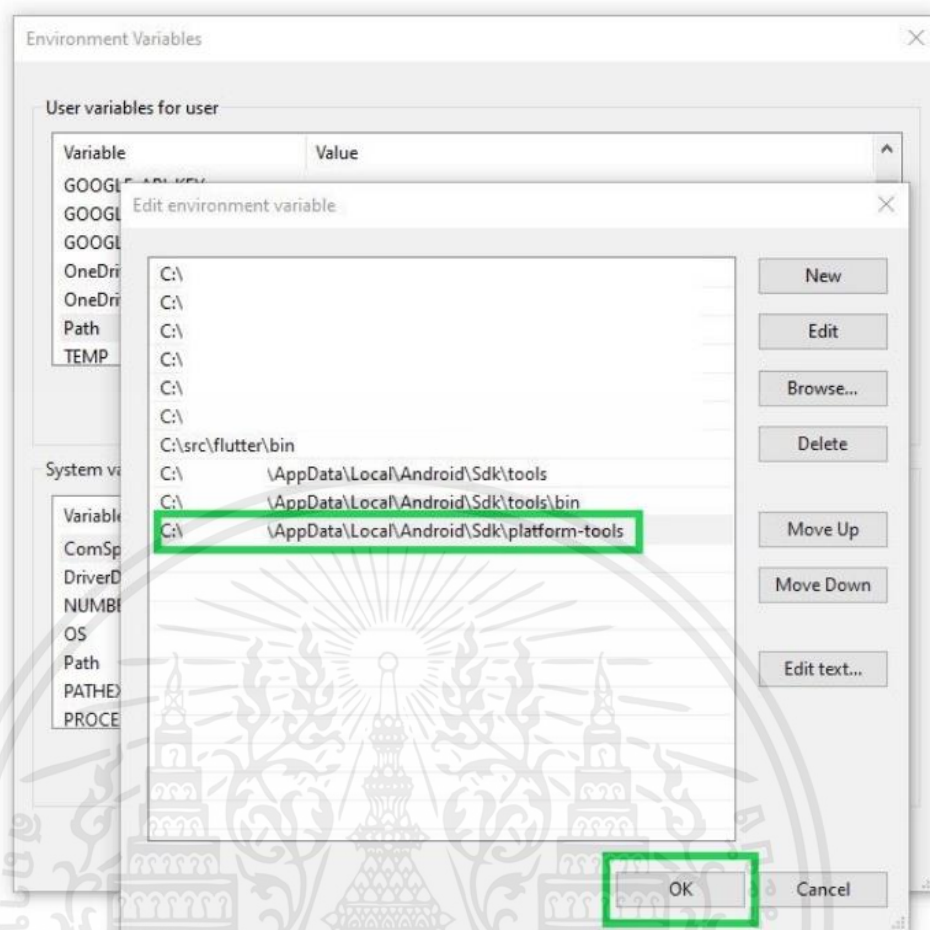
2.2.1.2 เมื่อดาวน์โหลดเสร็จสิ้นให้ เปิดโปรแกรม Edit The System Environment Variables ขึ้นมาแล้วไปที่ Advanced กดปุ่ม Environment Variables ตรง User variables ถ้าหากยังไม่มี การเพิ่ม path ของ Android SDK ให้กด New



รูปที่ 4.8 หน้าต่างในการเพิ่ม path ของ Android SDK

2.2.1.3 ในส่วน Variable name ให้ใส่ ANDROID_SDK_ROOT และ ตรง Variable value ใส่ path ที่ copy มาจาก Android SDK Location เสร็จแล้วกด OK

2.2.1.4 ในส่วน User variables ให้คลิกที่ Path แล้วกดปุ่ม Edit วาง path ที่ copy มาจาก Android SDK Location โดยต้องเพิ่ม path สามอัน คือ path แรก พิมพ์ \tools ต่อท้าย path สอง พิมพ์ \tools\bin ต่อท้าย และ path สุดท้าย พิมพ์ \platform-tools ต่อท้าย แล้วกด OK



รูปที่ 4.9 Path ของ Android SDK ทั้งหมดที่ต้องเพิ่ม

2.2.2 ติดตั้ง Android License

2.2.2.1. ต้องติดตั้งเพื่อยอมรับเงื่อนไขของ Android License

ทำได้โดยการใช้คำสั่งดังนี้

```
flutter doctor --android-licenses
```

2.2.2.2. ในกระบวนการนี้ หากระบบให้ตอบคำถามให้ กด Y (YES) ทุกครั้ง

2.2.2.3. เมื่อเสร็จสิ้น ให้เช็คสถานะการติดตั้งอีกที โดยใส่คำสั่ง `flutter doctor` แล้วควรจะขึ้นสถานะดังนี้

```

Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 1.22.5, on Microsoft Windows [Version 10.0.18363.1316], locale th-TH)

[✓] Android toolchain - develop for Android devices (Android SDK version 30.0.3)
[!] Android Studio (version 4.1.0)
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
[!] VS Code (version 1.52.1)
    X Flutter extension not installed; install from
      https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter
[!] Connected device
    ! No devices available

! Doctor found issues in 3 categories.

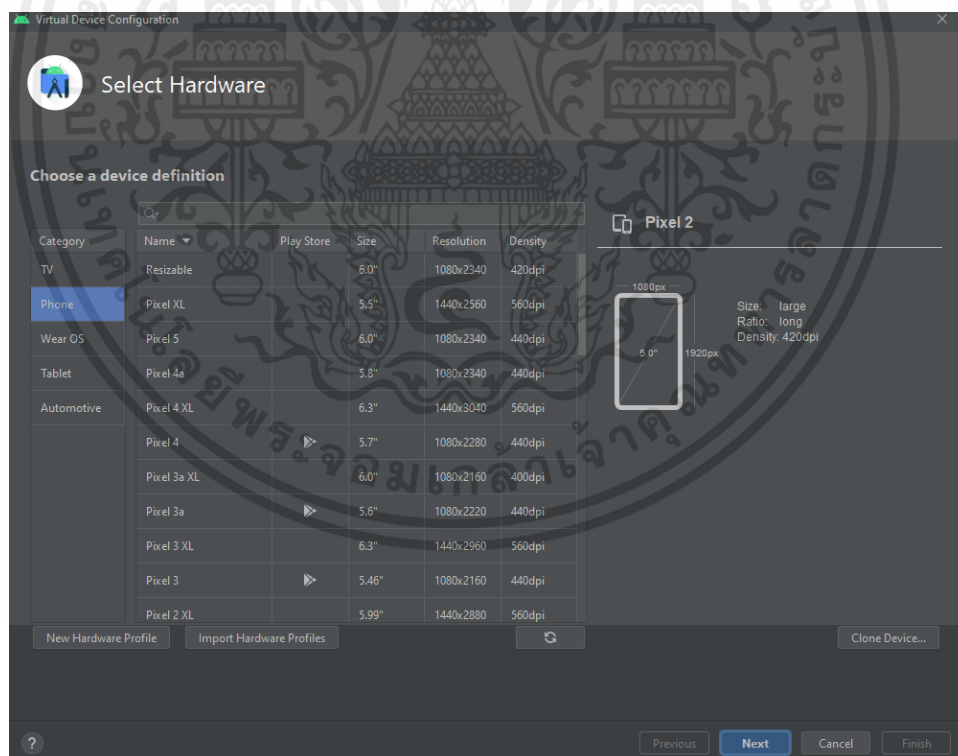
```

รูปที่ 4.10 สถานะการติดตั้งของ flutter ภายในขั้นตอนที่ 2.2.3.

2.3. สร้าง Android Virtual Device (AVD)

2.3.1 เปิด Android Studio แล้วไปที่ Configure > AVD Manager

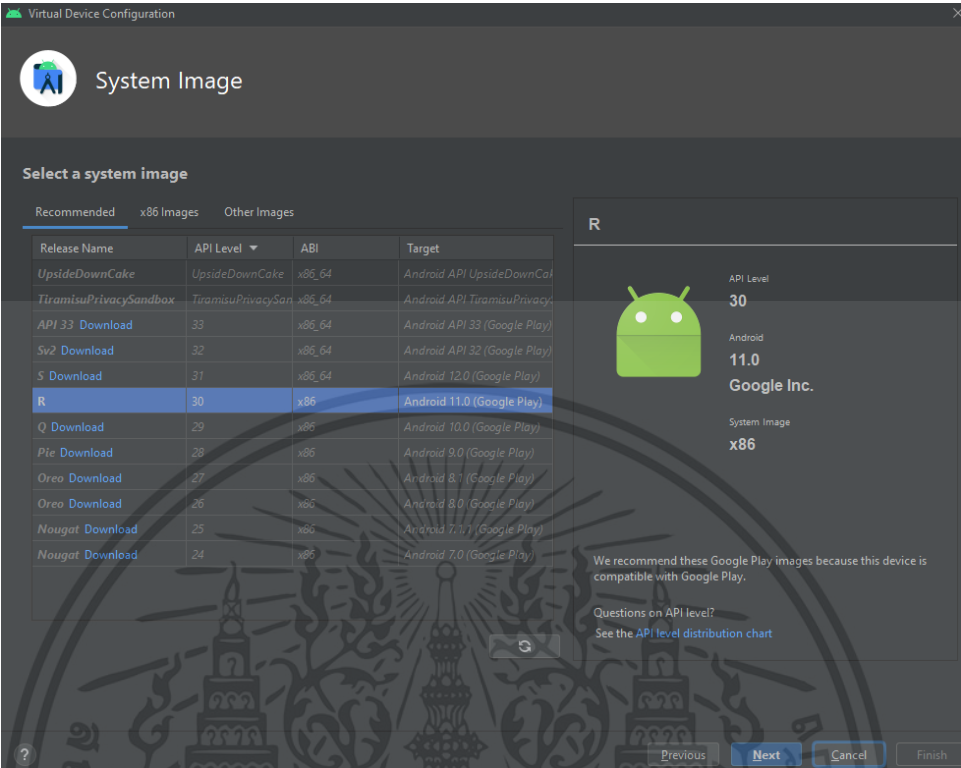
คลิกที่ Create Virtual Device ใน Category เลือก Phone แล้วเลือก pixel ที่ต้องการ จากนั้นกด Next คลิก Download รุ่นระบบที่ต้องการ จากนั้นกด Next



รูปที่ 4.11 การเลือกอุปกรณ์จำลองในการพัฒนา

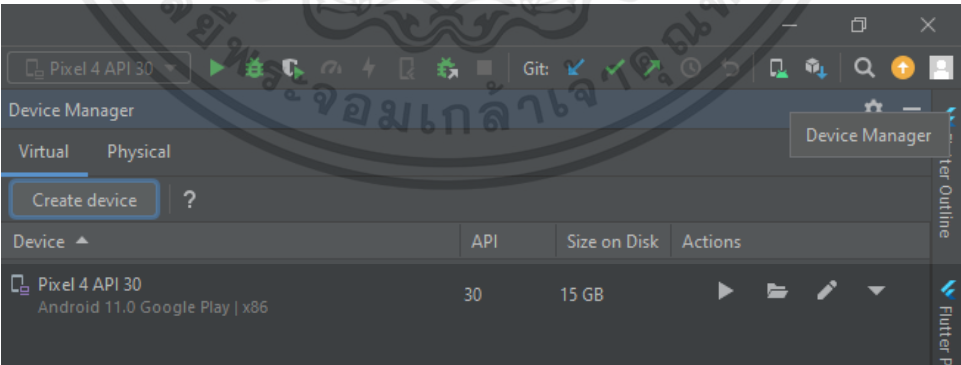
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3.2 คลิก Download รุ่นระบบที่ต้องการ จากนั้นกด Next



รูปที่ 4.12 รุ่นระบบที่ต้องการติดตั้งในอุปกรณ์จำลอง

2.3.2 เมื่อติดตั้งเสร็จแล้วก็สามารถที่จะเปิดตัวจำลองได้ที่ Device manager ได้แล้ว



รูปที่ 4.13 หน้าต่างเลือกอุปกรณ์จำลอง

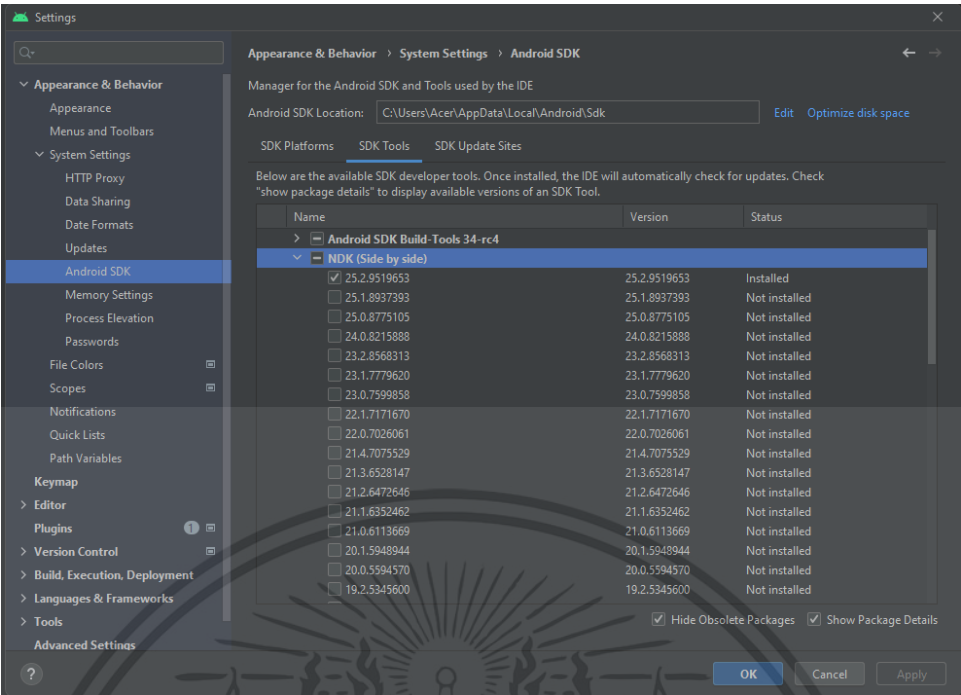
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.14 ตัวอย่าง อุปกรณ์จำลองที่สร้างขึ้น

2.4 ติดตั้ง NDK

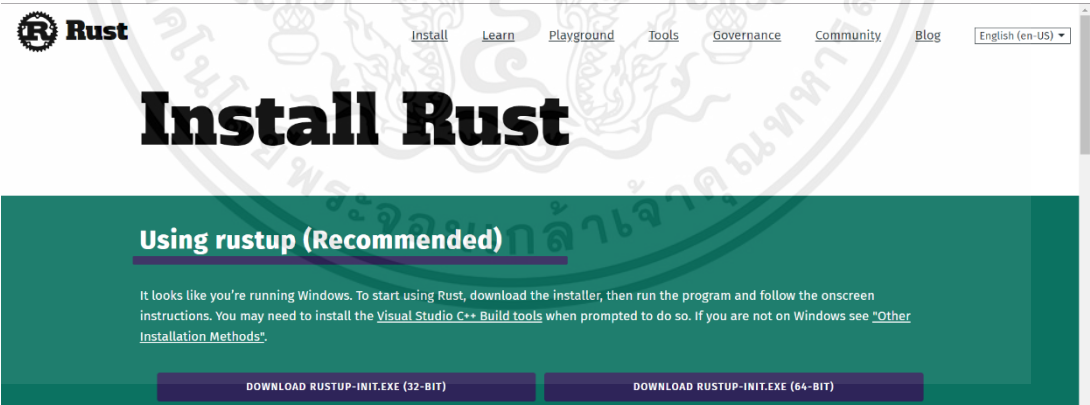
2.4.1 เข้า Android studio แล้วเข้าไปที่ File เลือก Setting จากนั้นในหัวข้อ Appearance & Behavior เลือก System setting จากนั้นเลือก Android SDK แล้วเลือกไปที่ SDK Tools และเลือก NDK (Side by side) และเลือก version ที่ต้องการดาวน์โหลดและทำการติดตั้งโดยการกด Apply หรือ OK



รูปที่ 4.15 หน้าต่างในการดาวน์โหลด NDK

4.1.2 การติดตั้ง Rust

- 1. ติดตั้ง Rust ลงบนเครื่องคอมพิวเตอร์ ผ่านเว็บ <https://www.rust-lang.org/tools/install>



รูปที่ 4.16 เว็บไซต์ในการติดตั้ง Rust

- ติดตั้ง LLVM ผ่านเว็บ <https://releases.llvm.org/download.html>

LLVM Download Page

Git Access

If you'd like access to the "latest and greatest" in LLVM development, please see the instructions for accessing the [LLVM Git Repository](#). The major changes and improvements that the development version contains relative to the previous release are listed in the [Release Notes](#) for the next release.

Download LLVM 16.0.0

Sources / Pre-Built Binaries / Doxygen:

These are available on the GitHub release [page](#).

Documentation:

- [LLVM \(release notes\)](#)
- [Clang \(release notes\)](#)
- [clang-tools-extra \(release notes\)](#)
- [LLD \(release notes\)](#)
- [libc++ \(release notes\)](#)
- [Polly \(release notes\)](#)
- [Flang \(release notes\)](#)

Download LLVM 15.0.7

4

รูปที่ 4.17 เว็บไซต์ในการติดตั้ง LLVM

- ลง generator ด้วยคำสั่ง `cargo install flutter_rust_bridge_codegen`
- เมื่อติดตั้งเสร็จแล้ว ใช้คำสั่ง “`cargo new native`” เพื่อสร้างโฟลเดอร์ของ rust ขึ้นมา

```
C:\Users\Pakin>cargo new native
Created binary (application) 'native' package
```

รูปที่ 4.18 คำสั่งในการเพิ่มโฟลเดอร์ของ rust

- หลังจากสร้างโฟลเดอร์เสร็จแล้ว ให้เปลี่ยนไฟล์ดังนี้
 - ลบไฟล์ `main.rs`

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ในโฟลเดอร์ src สร้างไฟล์ api.rs และ lib.rs โดยไฟล์ api.rs จะทำหน้าที่แทน main.rs ในการเขียนฟังก์ชันต่างๆ ส่วน lib.rs จะรวม api.rs และไฟล์ที่ใช้ในการแปลงซึ่งจะถูกสร้างอัตโนมัติในภายหลัง

6. ในไฟล์ cargo.toml ที่หัวข้อ lib ให้ใส่ดังนี้

```
[lib]
crate-type = ["lib","staticlib","cdylib",]
```

รูปที่ 4.19 ชุดคำสั่งภายใน cargo.toml

ที่หัวข้อ build-dependencies เพิ่ม flutter_rust_bridge_codegen = "1"

```
[build-dependencies]
flutter_rust_bridge_codegen = "1"
```

รูปที่ 4.20 ชุดคำสั่งที่ต้องเพิ่มในหัวข้อ build-dependencies

7. ในไฟล์ lib.rs เพิ่ม 2 บรรทัดนี้ลงไปไฟล์

```
native > src > lib.rs > ...
1  mod bridge_generated;
2  mod api;
3
```

รูปที่ 4.21 ชุดคำสั่งที่ต้องเพิ่มในไฟล์ lib.rs

8. ในไฟล์ api.rs เพิ่มบรรทัดเหล่านี้ลงไปไฟล์

```
pub enum Platform {
    Unknown,
    Android,
    Ios,
    Windows,
    Unix,
    MacIntel,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

MacApple,
Wasm,
}

pub fn platform() -> Platform {
    if cfg!(windows) {
        Platform::Windows
    } else if cfg!(target_os = "android") {
        Platform::Android
    } else if cfg!(target_os = "ios") {
        Platform::Ios
    } else if cfg!(all(target_os = "macos", target_arch = "aarch64")) {
        Platform::MacApple
    } else if cfg!(target_os = "macos") {
        Platform::MacIntel
    } else if cfg!(target_family = "wasm") {
        Platform::Wasm
    } else if cfg!(unix) {
        Platform::Unix
    } else {
        Platform::Unknown
    }
}

pub fn rust_release_mode() -> bool {
    cfg!(not(debug_assertions))
}

```

9. ในหน้าต่าง terminal พิมพ์คำสั่งดังต่อไปนี้ เพื่อเพิ่มแพลตฟอร์มเป้าหมายในสร้างไฟล์

```

rustup target add 1686-linux-android
rustup target add x86_64-linux-android
rustup target add armv7-linux-androideabi
rustup target add aarch64-linux-android

```

รูปที่ 4.22 ชุดคำสั่งที่ในการเพิ่มแพลตฟอร์ม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

10. พิมพ์คำสั่ง `cargo install cargo-ndk` เพื่อให้สามารถทำงานกับ Android NDK ได้
11. ที่ `android/app/build.gradle` เพิ่มบรรทัดดังต่อไปนี้

```
[
    Debug: null,
    Profile: '--release',
    Release: '--release'
].each {
    def taskPostfix = it.key
    def profileMode = it.value
    tasks.whenTaskAdded { task →
        if (task.name == "javaPreCompile$taskPostfix") {
            task.dependsOn "cargoBuild$taskPostfix"
        }
    }
    tasks.register("cargoBuild$taskPostfix", Exec) {
        workingDir "../native"
        environment ANDROID_NDK_HOME: "$ANDROID_NDK"
        commandLine 'cargo', 'ndk',
            // the 2 ABIs below are used by real Android devices
            '-t', 'armeabi-v7a',
            '-t', 'arm64-v8a',
            '-t', 'x86',
            '-t', 'x86_64',
            '-o', '../android/app/src/main/jniLibs', 'build'
        if (profileMode != null) {
            args profileMode
        }
    }
}
```

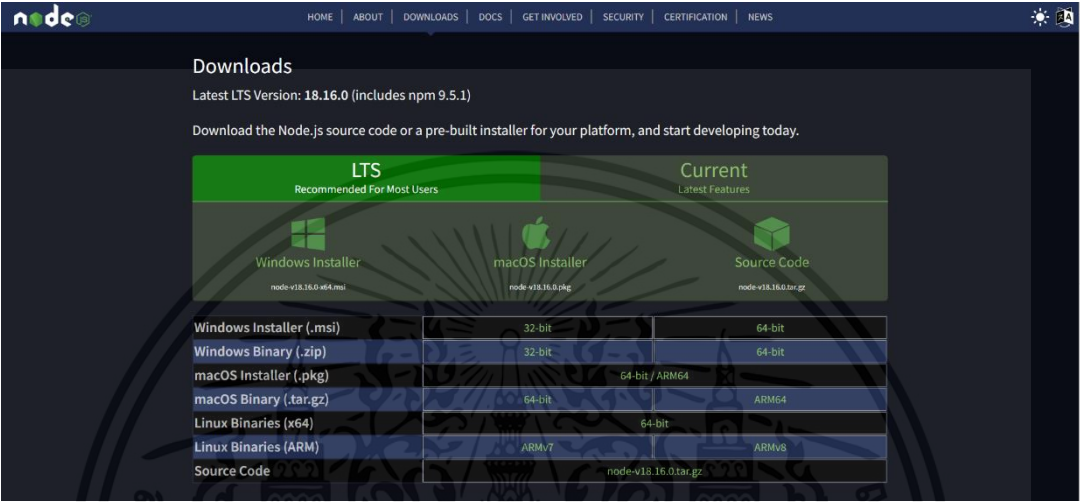
รูปที่ 4.23 ชุดคำสั่งที่ต้องเพิ่มในไฟล์ `android/app/build.gradle`

และที่ `android/gradle.properties` เพิ่ม “`ANDROID_NDK=`” โดยใส่ที่ที่ติดตั้ง NDK ลงไป

12. เพิ่ม dependencies ดังต่อไปนี้ในโปรเจกต์ flutter ในไฟล์ `pubspec.yaml`
 - `flutter pub add --dev ffigen && flutter pub add ffi`
 - `flutter pub add flutter_rust_bridge`
 - `flutter pub add -d build_runner`
 - `flutter pub add -d freezed`
 - `flutter pub add freezed_annotation`

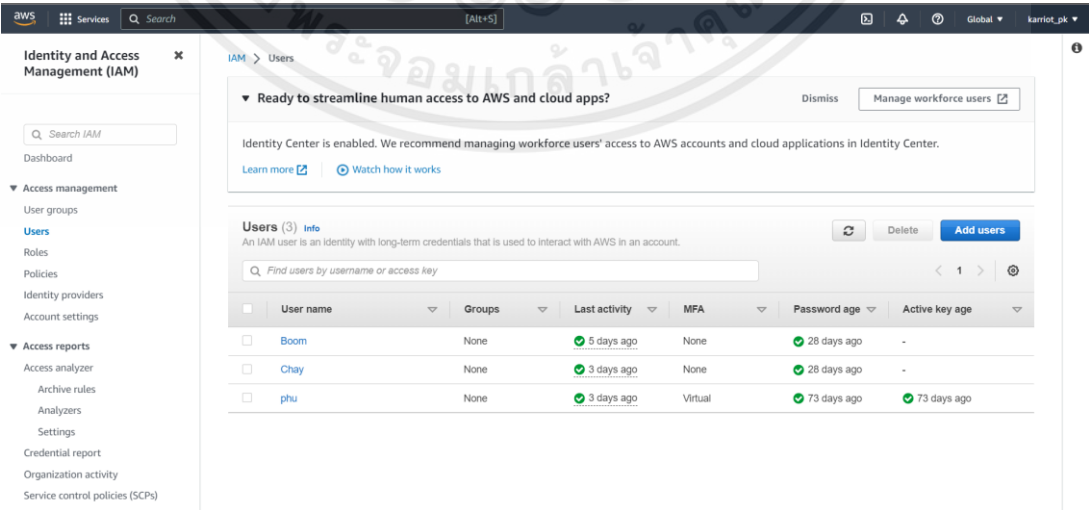
4.1.3 การติดตั้ง Node.js เพื่อทำ Serverless

1. ติดตั้ง node.js จาก <https://nodejs.org/en/download>



รูปที่ 4.24 เว็บไซต์ในการติดตั้ง node.js

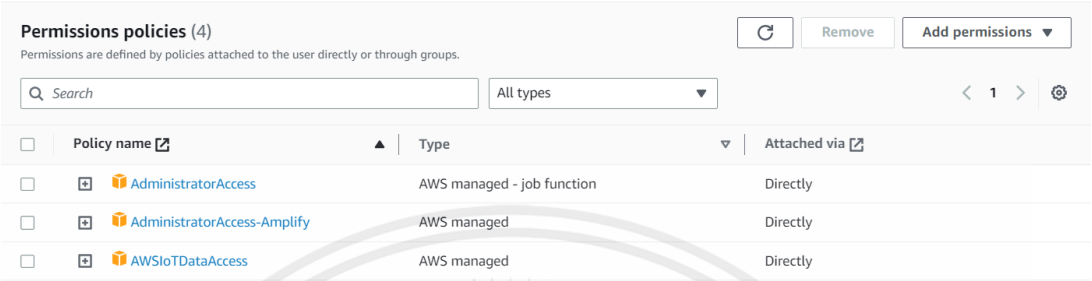
- 2. เมื่อติดตั้งเสร็จสิ้น ติดตั้ง serverless ผ่านคำสั่ง “npm i serverless@^3.28.1 -g”
- 3. เข้า <https://aws.amazon.com/> เพื่อสร้างบัญชีของ AWS ในกรณีที่ยังไม่มีบัญชี AWS
- 4. ไปที่ AWS Management console และไปที่บริการ IAM (Identity and Access Management)



รูปที่ 4.25 บริการ IAM ของ AWS

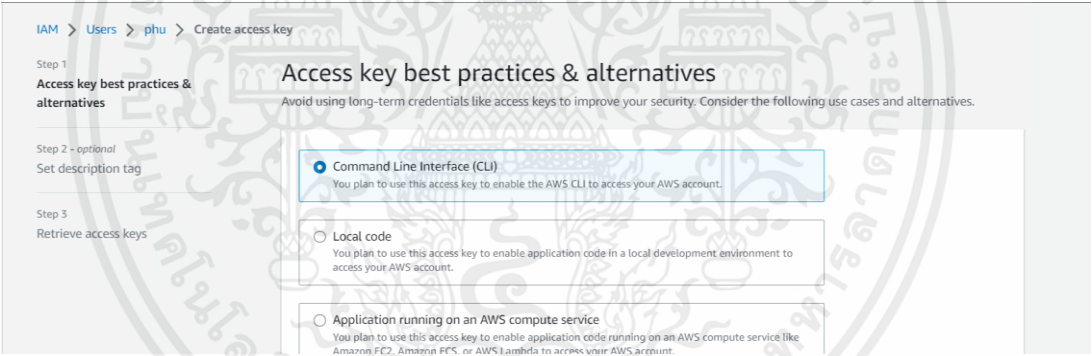
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5. สร้าง User โดยการกดที่ Users และ Add users โดยในระหว่างที่สร้างจะมีให้ตั้งค่าสิทธิการเข้าถึงต่างๆ ให้ใส่ AdministratorAccess, AdministratorAccess-Amplify และ AWSIoTDataAccess ลงไป

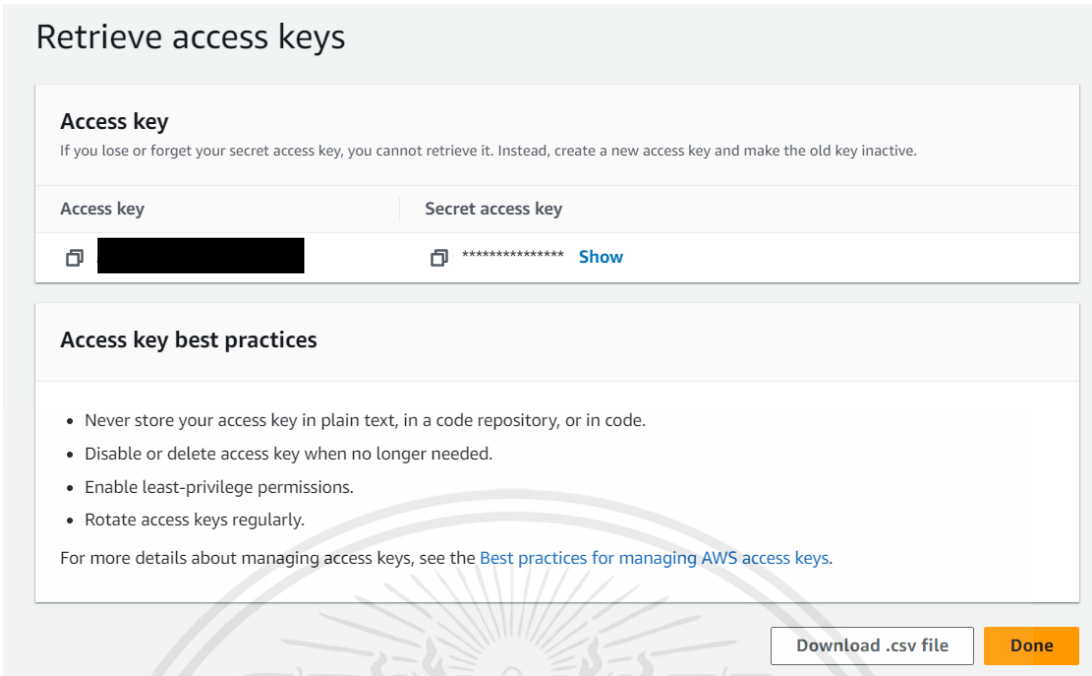


รูปที่ 4.26 การสร้าง Users และ Add users

6. เมื่อสร้าง user เสร็จแล้ว ให้ไปที่ user นั้น และกดที่ Security credentials, หาหัวข้อ Access keys และเลือกสร้าง access key ด้วยการกดที่ Create access key



รูปที่ 4.27 การสร้าง Access keys



รูปที่ 4.28 หน้าต่างแสดง Access keys

- 7. คัดลอก access key id และ secret access key
- 8. ในหน้าต่าง terminal พิมพ์คำสั่งดังต่อไปนี้ โดยแทน YOUR_AWS_ACCESS_KEY ด้วย access key id และ YOUR_AWS_SECRET_KEY ด้วย secret access key
serverless config credentials --provider aws --key YOUR_AWS_ACCESS_KEY --secret YOUR_AWS_SECRET_KEY
- 9. ต่อมา ติดตั้ง amplify-cli สำหรับการเพิ่มบริการในแอปพลิเคชัน ด้วยคำสั่ง
npm install -g @aws-amplify/cli [21]
- 10. เมื่อติดตั้งสำเร็จแล้ว ให้พิมพ์คำสั่ง amplify configure ซึ่งจะพาไปสู่หน้าคอนโซลของ aws โดยให้เข้าสู่ระบบด้วยบัญชี aws ที่มี
- 11. เมื่อเข้าสู่ระบบแล้ว ให้กลับมาที่ terminal และกด enter
- 12. เลือกภูมิภาคที่ต้องการ ในที่นี้เลือก ap-southeast-1
- 13. เมื่อเลือกภูมิภาคเสร็จสิ้น ตัวคอนโซลจะเปิดหน้าการสร้างผู้ใช้นี้มา ในที่นี้ถ้าหากมีผู้ใช้ที่สร้างไว้ใน iam แล้ว ไม่จำเป็นต้องสร้างใหม่
- 14. กลับมาที่หน้าต่าง terminal แล้วใส่ access key และ secret key จากขั้นตอนที่ 7 และตั้งชื่อโปรไฟล์ที่จะใช้สำหรับโปรเจกต์นี้

```

smart_iot_app amplify configure
Follow these steps to set up access to your AWS account:

Sign in to your AWS administrator account:
https://console.aws.amazon.com/
Press Enter to continue

Specify the AWS Region
? region: ap-southeast-1
Follow the instructions at
https://docs.amplify.aws/cli/start/install/#configure-the-amplify-cli

to complete the user creation in the AWS console
https://console.aws.amazon.com/iamv2/home#/users/create
Press Enter to continue

Enter the access key of the newly created user:
? accessKeyId: [hidden]

```

รูปที่ 4.29 หน้าต่าง Terminal สำหรับใส่ access key และ secret key

4.2 ขั้นตอนในการทำงาน

ในการทำแอปพลิเคชันบนมือถือประกอบไปด้วยหลายขั้นตอน เช่น การวางแผนก่อนลงมือทำ การออกแบบ การสร้าง เป็นต้น เพื่อพัฒนาประสิทธิภาพและการทำงานที่เร็วขึ้น พวกเราได้นำแนวคิดการทำงานในรูปแบบ Agile มาใช้ในกระบวนการพัฒนาของโปรเจกต์ เนื่องจากการพัฒนาแอปพลิเคชันจะแบ่งเป็น 2 ส่วนหลักๆ คือ คุณลักษณะต่างๆภายในแอปพลิเคชัน และ หน้าตาของแอปพลิเคชัน เพื่อให้ผู้จัดทำสามารถนำศักยภาพออกมาได้มากที่สุด และ เพื่อนำเสนอแนวทางที่มีโครงสร้างในการพัฒนาขึ้นมา แนวคิด Agile จะรวมชุดของขั้นตอนต่างๆ ที่จะมาช่วยเป็นแนวทางในการทำโปรเจกต์ โดยพวกเราได้ทำตามขั้นตอนตามแนวคิด Agile ดังนี้

1. วางแผนระบบภาพรวมทั้งหมด
 - a. ออกแบบภาพรวมของโปรเจกต์ว่าจะออกแบบระบบและ การทำงานออกมาในรูปแบบไหนและจะใช้ ฮาร์ดแวร์ เซิร์ฟเวอร์ ซอฟต์แวร์ อะไรบ้าง
 - b. ออกแบบคุณลักษณะและการทำงานของแต่ละฝั่ง ได้แก่ ฮาร์ดแวร์ เซิร์ฟเวอร์ ซอฟต์แวร์ ตามที่ได้ออกแบบไว้ ว่าแต่ละอย่างมีหน้าที่อะไรบ้าง
 - c. ค้นหาภาษาต่างๆที่ใช้ในการพัฒนา รวมถึงบริการ เซิร์ฟเวอร์ที่ต้องใช้ที่มีคุณลักษณะที่ตรงกับที่ออกแบบไว้
 - d. วางแผนการเชื่อมต่อระหว่างระบบว่าจะเชื่อมต่อโดยใช้โปรโตคอลอะไร
2. ออกแบบหน้าตาและการใช้งานภายในแอปพลิเคชัน
 - a. ร่างหน้าตาแอปพลิเคชันอ้างอิงจากคุณลักษณะที่กำหนดไว้ในการวางแผน
 - b. สร้างหน้าตาและตกแต่งภายในโปรแกรม Figma เพื่อเป็นแบบจำลองในการนำไปใช้ในแอปพลิเคชัน
 - c. ออกแบบหน้าตาในแอปพลิเคชันจริงตามแบบที่ได้ออกแบบไว้ในโปรแกรม Figma

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. สร้างเดโมของแอปพลิเคชัน

- a. สร้างโปรเจกต์ Flutter โดยเน้นไปที่การสร้างหน้าต่างแอปพลิเคชัน และการเชื่อมต่อระหว่างหน้าต่างๆภายในแอปพลิเคชัน
- b. เพิ่มคุณลักษณะต่างๆที่จะนำมาเป็นคุณลักษณะหลักของแอปพลิเคชันลงไป เช่น การยืนยันตัวตน และการเชื่อมต่อกับเซิร์ฟเวอร์
- c. สร้างบริการอื่นๆ ภายในแอปพลิเคชัน เช่น การดึงข้อมูลจากฐานข้อมูลบนเซิร์ฟเวอร์
- d. ทดสอบคุณลักษณะและบริการต่างๆ พร้อมแสดงผลให้อาจารย์ที่ปรึกษา และนำไปปรับเปลี่ยนตามข้อเสนอแนะ

4. วนซ้ำการออกแบบระบบและปรับเปลี่ยนหน้าต่างแอปพลิเคชัน

- a. ปรับปรุงในระบบและคุณลักษณะต่างๆที่ได้รับการเสนอแนะและเพิ่มคุณลักษณะที่ยังขาดเข้าไป รวมถึงปรับปรุงหน้าต่างของแอปพลิเคชันเพื่อให้ภาพรวมของแอปพลิเคชันสามารถนำไปใช้จริงได้มากที่สุด โดยจะดำเนินการไปคู่ขนานกัน
- b. ปรับปรุงคุณลักษณะ บริการ และหน้าต่างของแอปพลิเคชัน ตามการออกแบบที่ได้รับจากข้อเสนอแนะ
- c. ทดสอบการทำงานใหม่อีกครั้ง พร้อมกับแสดงให้อาจารย์ที่ปรึกษา และนำกลับไปแก้ไขตามข้อเสนอแนะ

5. ทดสอบความเสถียรของระบบ

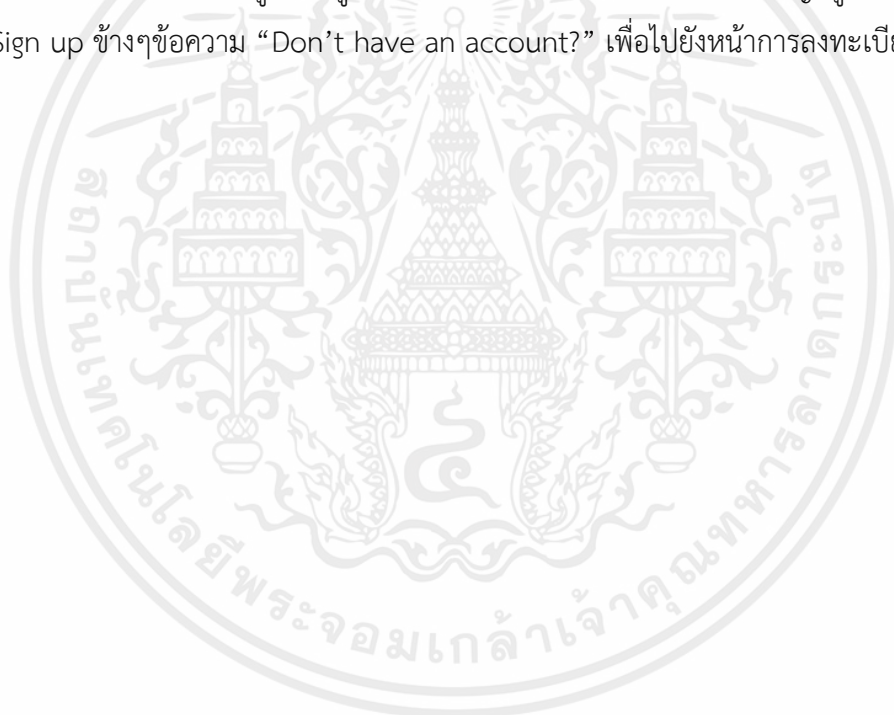
- a. ทดสอบการทำงานของระบบระหว่างองค์ประกอบต่างๆ เพื่อตรวจสอบว่ามีจุดใดมีปัญหาหรือไม่ โดยเริ่มทดสอบจากบนอุปกรณ์จำลองบนคอมพิวเตอร์ จำลองการทำงานบนโทรศัพท์ และการใช้งานแอปพลิเคชันบนโทรศัพท์จริง โดยในระหว่างแต่ละการทดสอบบนอุปกรณ์ ก็จะตรวจสอบการทำงานของฟังก์ชันบนเซิร์ฟเวอร์ด้วย
- b. แก้ไขจุดที่เป็นปัญหาที่เกิดขึ้น
- c. เมื่อแก้ไขเสร็จสิ้น นำกลับไปทดสอบไล่ตามอุปกรณ์อีกครั้ง ถ้าหากการทำงานบนโทรศัพท์จริงนั้นไม่มีปัญหา ก็สามารถส่งออกเป็นไฟล์ติดตั้ง apk บนโทรศัพท์ได้

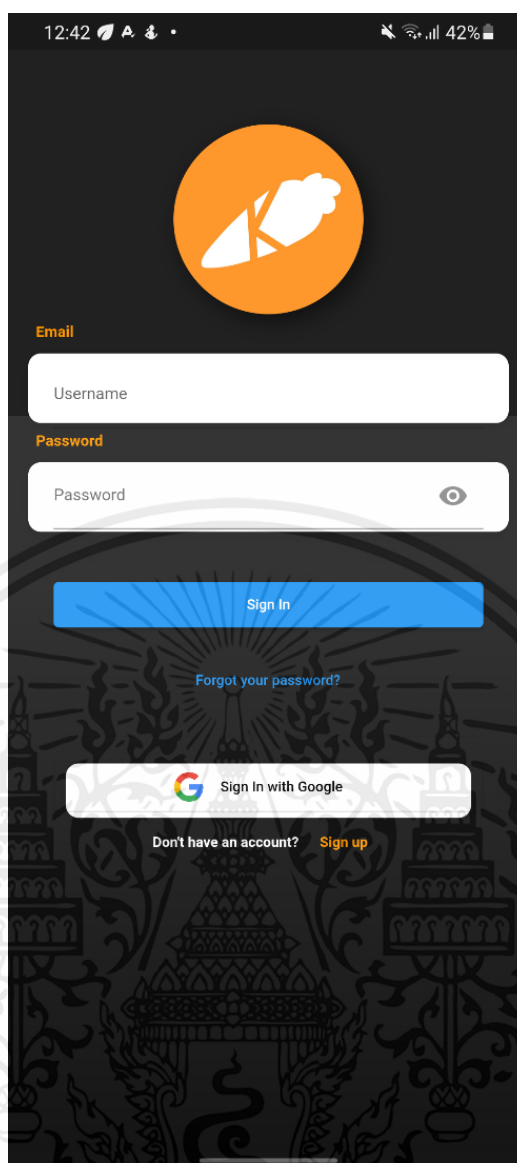
4.3 การใช้งานภายในแอปพลิเคชัน

ในส่วนนี้จะเป็นการแสดงการเชื่อมต่อระหว่างอินเตอร์เฟซของการทำงานในแต่ละหน้าและคุณลักษณะต่างๆภายในแอปพลิเคชันในภาพรวม ได้แก่ การเข้าสู่ระบบ การลงทะเบียน หน้าแดชบอร์ดหลัก หน้าประวัติ หน้าการวิเคราะห์ หน้ารายงาน หน้ารายละเอียดของอุปกรณ์

4.3.1 การเข้าสู่ระบบ

ในการเริ่มแอปพลิเคชัน หน้าเข้าสู่ระบบจะปรากฏให้ผู้ใช้เป็นหน้าแรก โดยผู้ใช้สามารถเลือกช่องทางการเข้าสู่ระบบได้ 2 ช่องทาง ได้แก่ อีเมลและรหัสผ่านที่สมัครไว้ หรือ ใช้บัญชีกูเกิ้ลในการเข้าสู่ระบบก็ได้ เมื่อกด Sign in แล้วก็จะไปสู่หน้า Onboarding ถ้าหากใช้แอปพลิเคชันนี้เป็นครั้งแรก หลังจากนั้นก็จะพาผู้ใช้เข้าสู่หน้าแดชบอร์ดหลัก หรือถ้าหากยังไม่มีบัญชีผู้ใช้ สามารถกดที่ปุ่ม Sign up ข้างๆข้อความ “Don’t have an account?” เพื่อไปยังหน้าการลงทะเบียนได้





รูปที่ 4.30 หน้าล็อกอินแอปพลิเคชัน

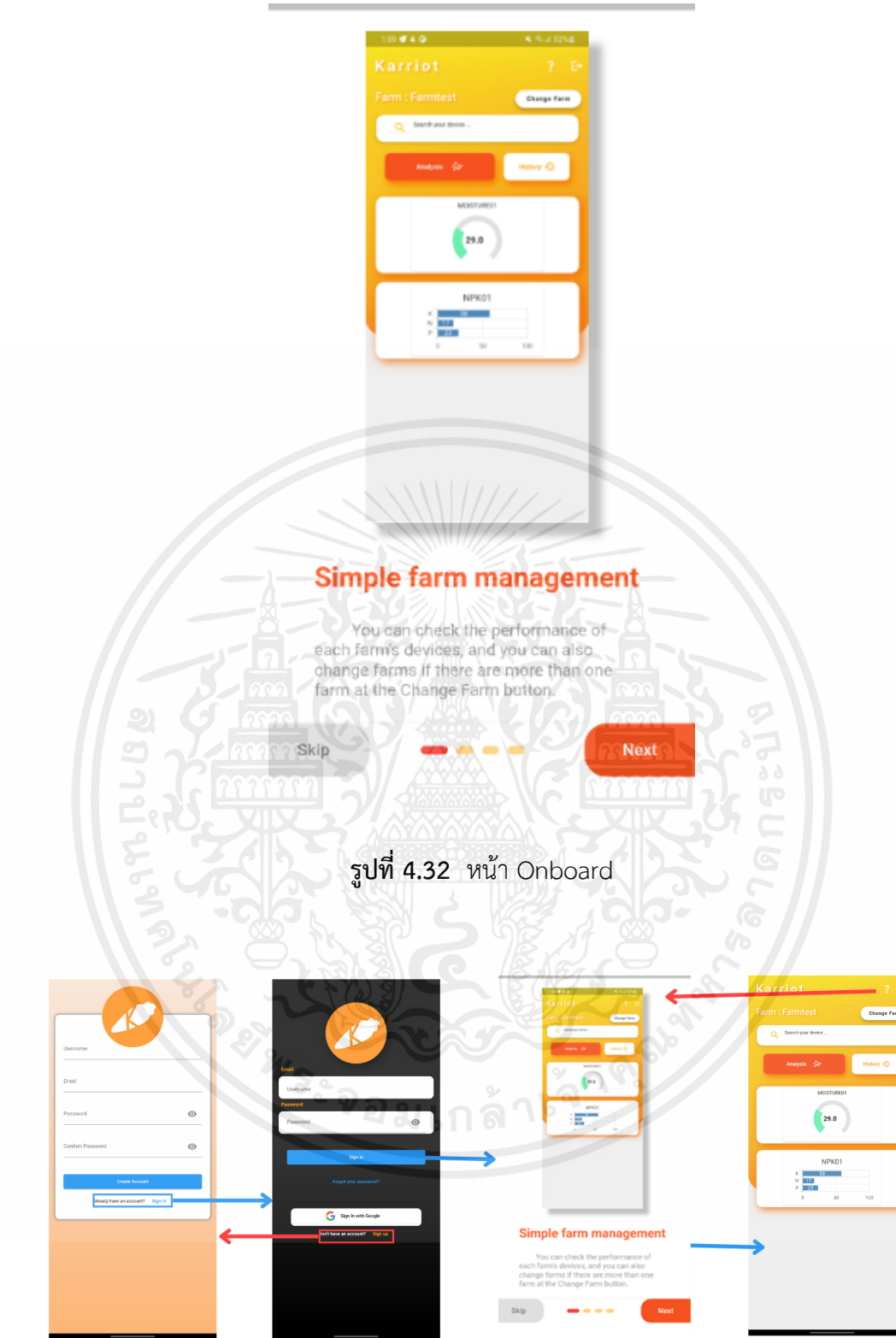
4.3.2 การลงทะเบียน

ในรูปภาพเป็นการแสดงหน้าลงทะเบียนภายในแอปพลิเคชัน หน้าการลงทะเบียนจะประกอบไปด้วย ชื่อผู้ใช้ อีเมล รหัสผ่าน และการยืนยันรหัสผ่าน เมื่อผู้ใช้กด Create Account จะเป็นการสร้างบัญชีผู้ใช้ โดยจะมีการยืนยันอีเมลด้วยการส่งเมลยืนยันไปที่อีเมลที่ผู้ใช้กรอกไว้ ซึ่งในเมลจะเป็นรหัสที่ใช้เพื่อในการกรอกตัวเลขลงในแอปพลิเคชันซึ่งปรากฏเป็นหน้ากรอกรหัส เมื่อกรอกรหัสเสร็จสิ้น ก็จะไปสู่หน้า Onboarding

รูปที่ 4.31 หน้าลงทะเบียน

4.3.3 หน้าสอนการใช้งาน

ในรูปภาพเป็นการแสดงหน้า Onboarding เพื่อสอนผู้ใช้งานเกี่ยวกับคุณลักษณะต่างๆ ภายในแอปพลิเคชัน ผู้ใช้สามารถกด Next เพื่อไปยังหน้าถัดไปได้ หรือจะกด Skip เพื่อข้ามไปยังหน้าสุดท้าย โดยผู้ใช้งานสามารถดูจำนวนหน้าที่เหลือได้ที่บริเวณด้านล่าง เมื่ออยู่ที่หน้าสุดท้ายแล้ว ปุ่ม Next จะเปลี่ยนเป็น Finish เพื่อสิ้นสุดหน้า Onboarding และหน้านี้จะไม่ปรากฏในครั้งถัดไปที่เปิดแอปพลิเคชันขึ้นมา



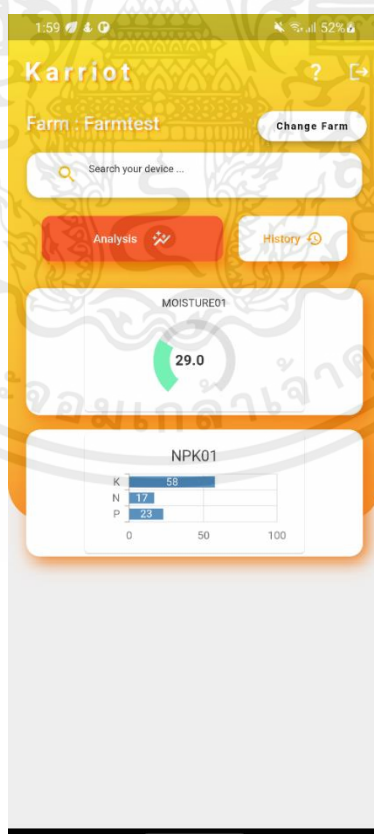
รูปที่ 4.33 การเชื่อมต่อจากหน้ายืนยันตัวตน หน้าสอนการใช้งาน และหน้าแดชบอร์ด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3.4 หน้าแดชบอร์ดหลัก

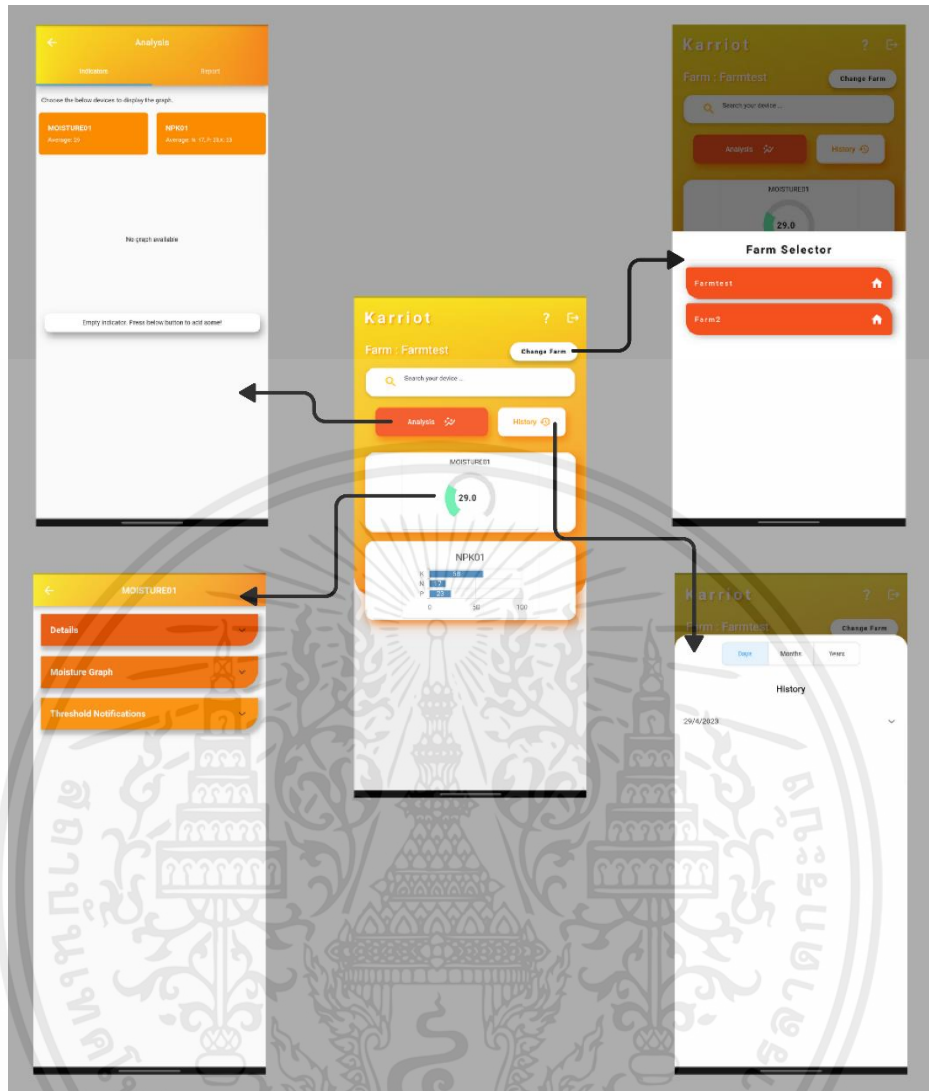
หน้าแดชบอร์ดจะเป็นหน้าหลักของแอปพลิเคชันนี้ โดยจะเป็นการแสดงผลค่าล่าสุดตามเวลาจริงของอุปกรณ์ในรูปแบบต่างๆ แยกตามประเภทอุปกรณ์ภายในฟาร์มนั้น โดยจะมีคุณลักษณะอื่นๆ ที่ผู้ใช้สามารถเข้าไปใช้ได้ดังนี้

- กดปุ่ม Analysis เพื่อไปยังหน้าวิเคราะห์ สำหรับการวิเคราะห์ข้อมูลและการสร้างรายงาน
- กดปุ่ม History เพื่อแสดงประวัติข้อมูล
- กดที่บริเวณที่แสดงผลของอุปกรณ์ เพื่อไปยังหน้ารายละเอียดของอุปกรณ์นั้น สำหรับการดูรายละเอียดต่างๆ การดูกราฟของข้อมูลทั้งหมดของอุปกรณ์นั้น และการตั้งค่าการแจ้งเตือน
- ผู้ใช้ยังสามารถค้นหาอุปกรณ์ได้ด้วยการพิมพ์ลงไปที่ย่อค้นหาสีขาวบริเวณด้านบน
- ผู้ใช้สามารถเลือกเปลี่ยนฟาร์มได้ที่บริเวณด้านบนขวาที่ปุ่ม Change Farm
- หากผู้ใช้ต้องการออกจากระบบสามารถกดปุ่มออกจากระบบได้ที่ปุ่มด้านขวามือ



รูปที่ 4.34 หน้าหลักของแอปพลิเคชัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.35 การเชื่อมต่อจากหน้าแดชบอร์ดไปยังหน้าคุณลักษณะอื่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3.5 การเปลี่ยนฟาร์ม

จากหน้าแดชบอร์ดหลัก เมื่อผู้ใช้งานกดที่ปุ่ม Change Farm ก็จะปรากฏเป็นหน้า Farm Selector โดยจะมีรายชื่อฟาร์มของผู้ใช้ขึ้นมา ผู้ใช้สามารถเลือกกดไปที่ชื่อฟาร์ม แล้วในหน้าแดชบอร์ดก็เปลี่ยนข้อมูลปัจจุบันที่แสดงอุปกรณ์ต่างๆภายในฟาร์มให้เป็นของฟาร์มที่ผู้ใช้เลือก

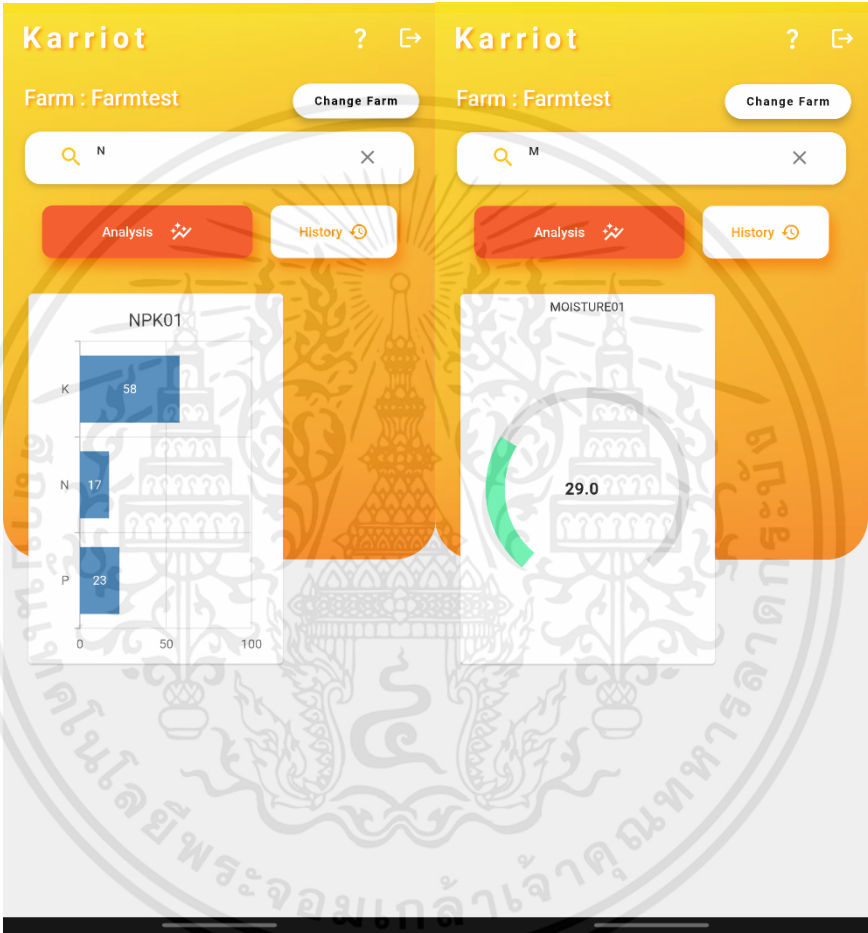


รูปที่ 4.36 หน้าหลักของแอปพลิเคชันเมื่อกดปุ่มเปลี่ยนฟาร์ม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3.6 การค้นหาอุปกรณ์

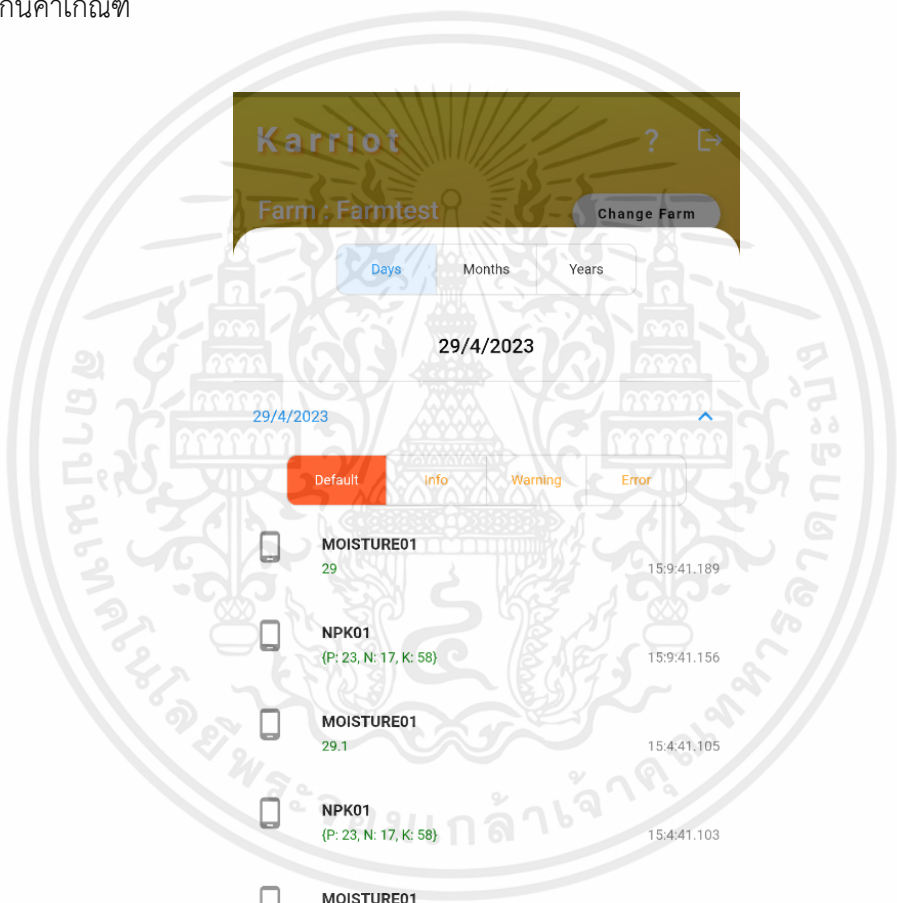
ในหน้าแดชบอร์ดหลัก จะมีช่องคำค้นหาอยู่ที่บริเวณด้านบน ผู้ใช้สามารถค้นหาอุปกรณ์ต่างๆได้ โดยการพิมพ์ตัวอักษรหรือคาลงไปในช่อง หน้าแดชบอร์ดจะนำอุปกรณ์ที่มีอักษรหรือคำตรงกันกับชื่ออุปกรณ์มาแสดงทั้งหมด ถ้าหากการค้นหาไม่พบข้อมูล ก็จะแสดงข้อความขึ้นมาว่า ไม่พบอุปกรณ์ที่ตรงกับคำที่ค้นหา



รูปที่ 4.37 แสดงการค้นหาอุปกรณ์เมื่อมีตัวอักษรตรงกับตัวอักษรในชื่ออุปกรณ์

4.3.7 ประวัติการทำงานของอุปกรณ์

จากหน้าแดชบอร์ดหลัก ผู้ใช้สามารถเข้าหน้าประวัติได้ด้วยการกดปุ่ม History ที่ได้ช่องการค้นหา ในหน้าประวัติจะแสดงข้อมูลของอุปกรณ์ในฟาร์มปัจจุบันที่แสดงในหน้าหลัก ผู้ใช้สามารถเลือกกรองข้อมูลในหน้าประวัติได้ 2 จุด คือ ด้านบนเป็นการกรองตาม วัน เดือน และ ปี โดยมีค่าเริ่มต้นที่ วัน และในฝั่งด้านล่างจะปรากฏเป็นรายการตามที่เราเลือกกรองไว้ เมื่อผู้ใช้กดเข้าไปที่แต่ละรายการ ก็จะปรากฏการกรองอีกอัน เป็นการกรองตามประเภทของข้อมูล ได้แก่ ข้อมูลรวม ข้อมูลปกติ ข้อมูลในระดับแจ้งเตือน (ใกล้ค่าเกณฑ์ที่ผู้ใช้กำหนด ถ้าผู้ใช้ได้ตั้งค่าไว้) และข้อมูลถึงและเกินค่าเกณฑ์

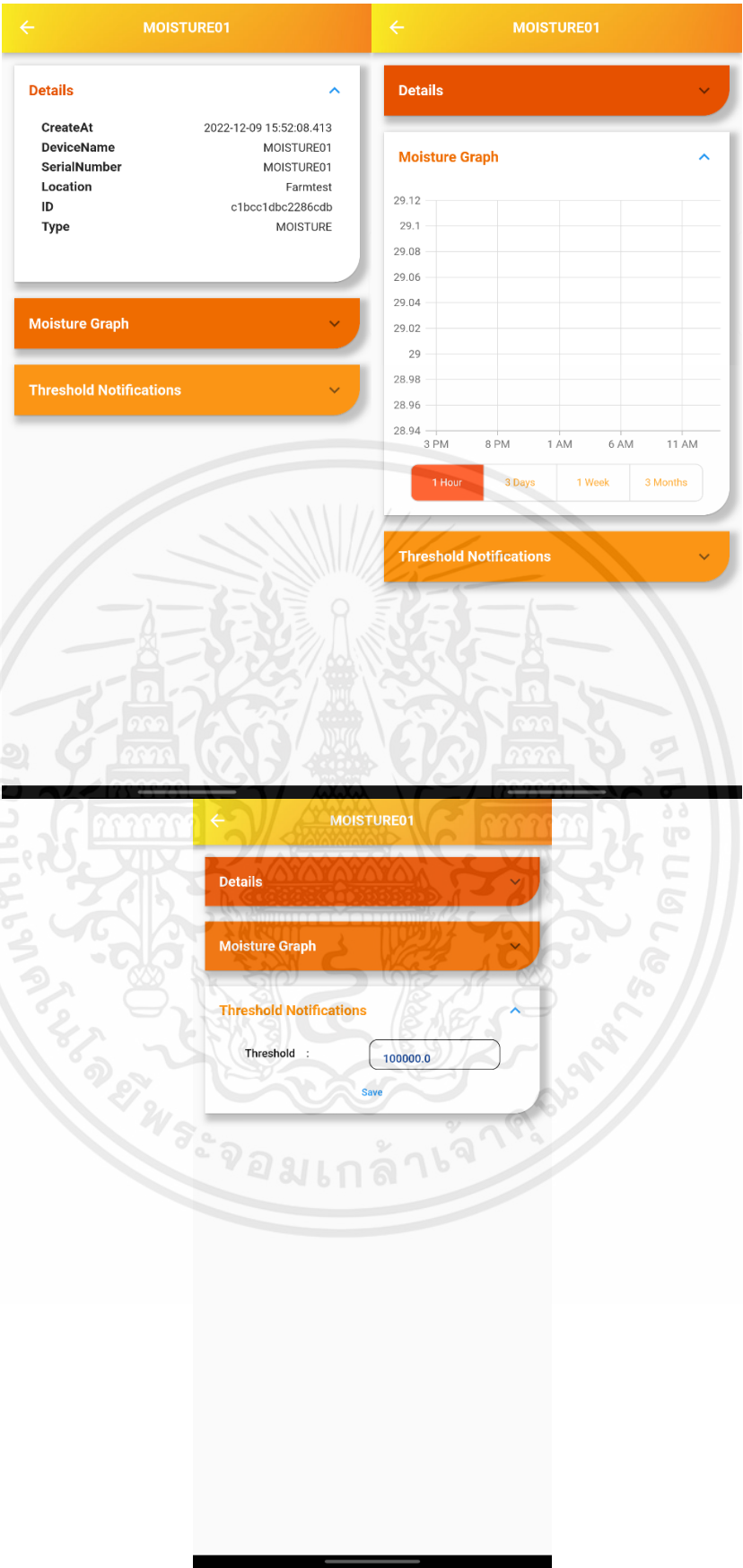


รูปที่ 4.38 หน้าประวัติของแอปพลิเคชัน

4.3.8 รายละเอียดของอุปกรณ์

ในการเข้าดูหน้ารายละเอียดของแต่ละอุปกรณ์ ผู้ใช้งานต้องกดที่อุปกรณ์ในหน้าหลัก ในหน้ารายละเอียดนี้จะประกอบไปด้วย 3 ส่วน ได้แก่ รายละเอียด กราฟ และ การตั้งค่าเกณฑ์การแจ้งเตือน

- รายละเอียด จะแสดงเกี่ยวกับข้อมูลอุปกรณ์ เช่น ไอดีอุปกรณ์ หมายเลขประจำตัวอุปกรณ์ เป็นต้น
- กราฟ เป็นการแสดงผลของข้อมูลอุปกรณ์ในรูปแบบกราฟเส้นตามเวลาจริง โดยมีข้อมูลแรกเริ่มเป็นข้อมูลที่เก็บไว้บนเครื่อง และผู้ใช้สามารถปรับข้อมูลในกราฟด้วยการกดปุ่มได้ กราฟเพื่อรองรับให้กราฟเลื่อน
- การตั้งค่าเกณฑ์การแจ้งเตือน ผู้ใช้สามารถตั้งค่าโดยการพิมพ์ลงในช่อง โดยค่าที่ตั้งนี้ จะทำงานเพื่อแจ้งเตือนผู้ใช้เป็นข้อความแจ้งเตือนใน 3 กรณี ได้แก่ เมื่อค่าที่เข้ามาของอุปกรณ์ใกล้กับค่าที่ตั้งไว้ เท่ากับค่าที่ตั้ง และเกินค่าที่ตั้งไว้



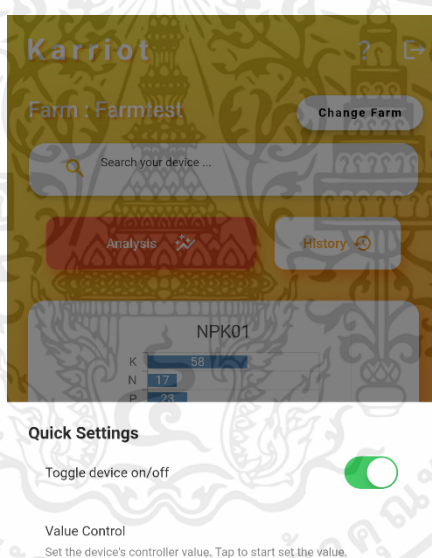
รูปที่ 4.39 หน้ารายละเอียดของอุปกรณ์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.3.9 การควบคุมอุปกรณ์

ผู้ใช้สามารถเข้าถึงการควบคุมอุปกรณ์ได้ด้วยการกดค้างที่อุปกรณ์ในหน้าแดชบอร์ดหลัก แล้วหน้าการตั้งค่าจะเด้งขึ้นมา ในหน้านี้จะประกอบไปด้วย การเปิด-ปิดอุปกรณ์ และการส่งค่าไปควบคุมอุปกรณ์ โดยในแต่ละส่วนทำหน้าที่ดังนี้

- การเปิด-ปิดอุปกรณ์ จะส่งค่าไปให้อุปกรณ์เพื่อเปิดปิดอุปกรณ์นั้น ทำให้อุปกรณ์นั้นไม่ส่งค่ากลับมา ผู้ใช้จะต้องรอการอัปเดตสถานะจากอุปกรณ์นั้นอีกครั้ง
- การส่งค่าไปควบคุมอุปกรณ์ ค่าที่ปรากฏให้สามารถตั้งค่าได้จะอ้างอิงจากอุปกรณ์ว่าเป็นประเภทอะไร ในส่วนนี้จะแตกต่างจากการเปิด-ปิดยกเว้นในกรณีที่อุปกรณ์นั้นทำหน้าที่เปิด-ปิดอย่างเดียว



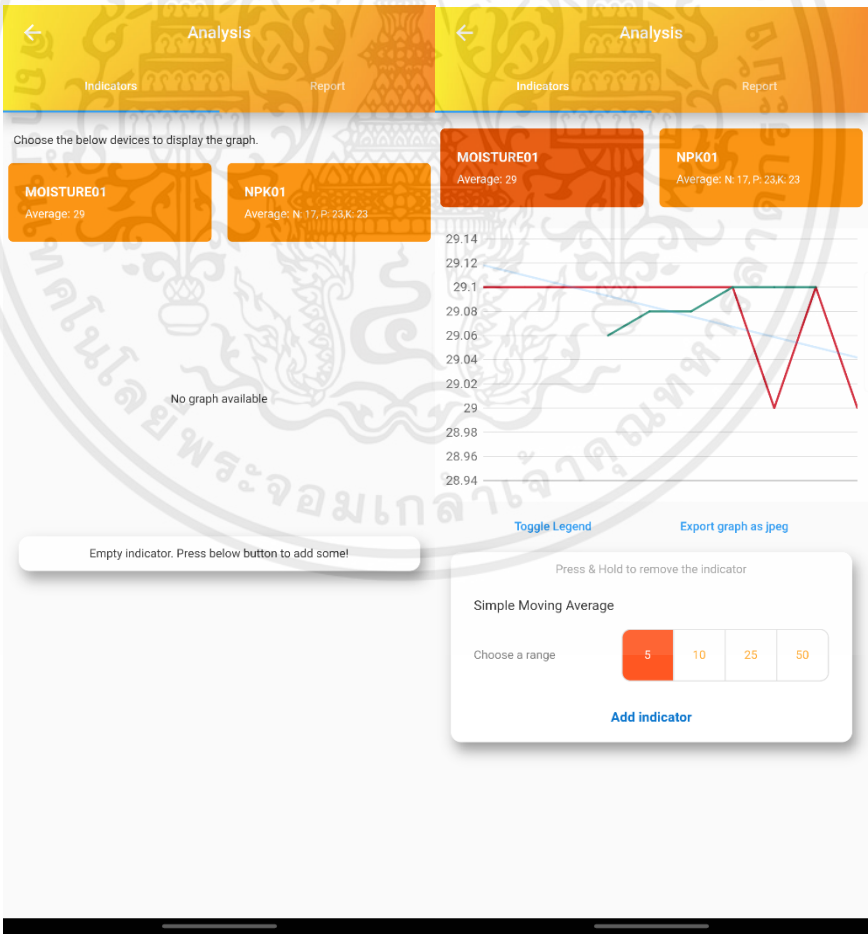
รูปที่ 4.40 หน้าเมนูการควบคุมอุปกรณ์

4.3.10 หน้าการวิเคราะห์

จากหน้าแดชบอร์ดหลักผู้ใช้สามารถกดที่ปุ่ม Analysis เพื่อเข้าสู่หน้านี้ ในหน้านี้จะมีด้วยกัน 3 ส่วนคือ ปุ่มเลือกอุปกรณ์ กราฟ และปุ่มเพิ่มตัวชี้วัด โดยถ้าเข้าหน้านี้นี้ครั้งแรก จะไม่มีการแสดงผลใดๆนอกจากปุ่มที่มีชื่ออุปกรณ์และค่าเฉลี่ยของอุปกรณ์นั้น ผู้ใช้จะต้องเลือกอุปกรณ์ใดๆก่อน แล้วหลังจากนั้นส่วนของกราฟและปุ่มเพิ่มตัวชี้วัดก็จะปรากฏขึ้นมา

- ในกราฟเป็นการแสดงข้อมูลทั้งหมดจากประวัติบนเครื่องในรูปแบบของกราฟเส้น และผู้ใช้สามารถกดเลือกแสดงเส้นได้ โดยการกดที่เครื่องหมายข้างๆกราฟ
- เมื่อผู้ใช้กดปุ่มเพิ่มตัวชี้วัด จะแสดงหน้าต่างเลือกตัวชี้วัด ซึ่งในปัจจุบันมี 2 ตัวชี้วัดได้แก่ SMA (Simple Moving Average) และ EMA (Exponential Moving Average)

โดยหลังจากที่กดเพิ่ม ตัวชี้วัดก็ถูกเพิ่มเข้าไปในกราฟ นอกเหนือจากที่มีส่วนของกราฟและตัวชี้วัดแสดงขึ้นมาหลังการเลือกอุปกรณ์ ก็จะมีการแสดงปุ่มเพิ่มเติม ได้แก่ ปุ่มแสดงหรือซ่อนข้อความกำกับกราฟ และปุ่มส่งออกรูปภาพ ณ ปัจจุบัน ซึ่งจะถูกนำไปใช้หน้ารายงาน

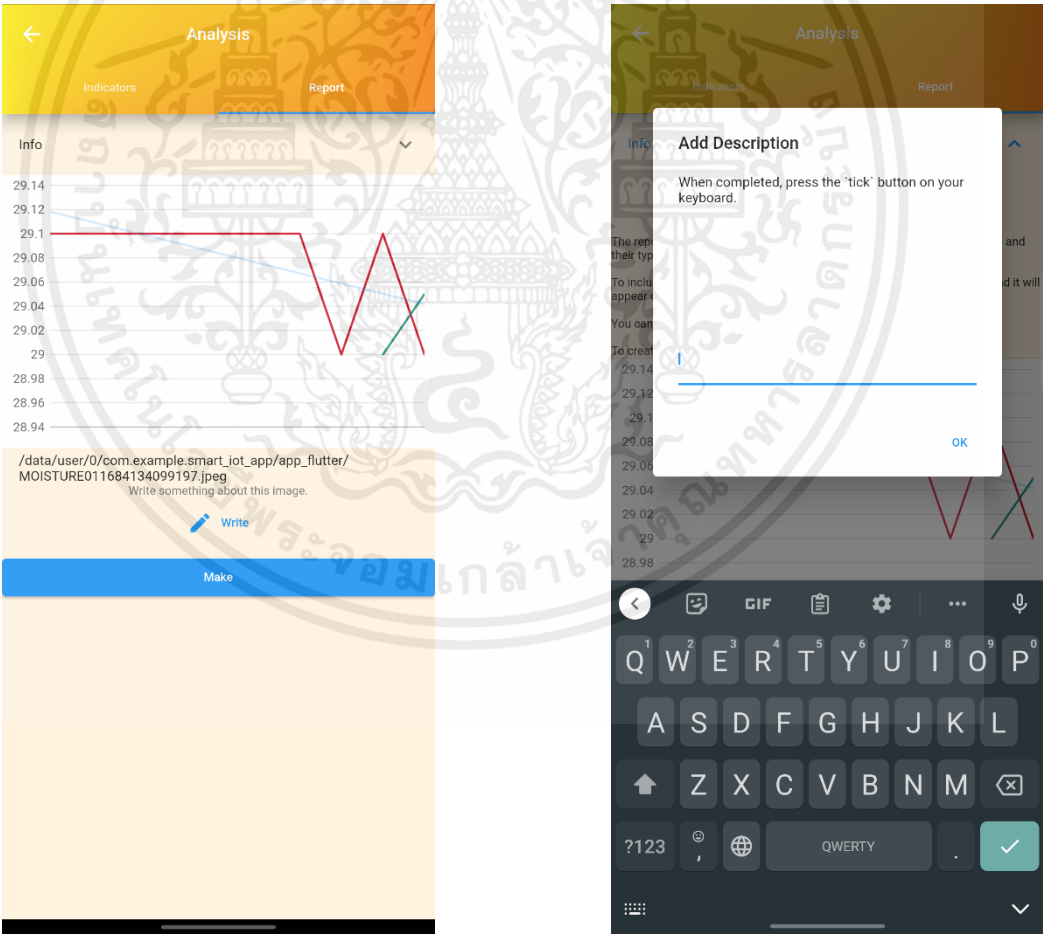


รูปที่ 4.41 หน้าวิเคราะห์การทำงานของอุปกรณ์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

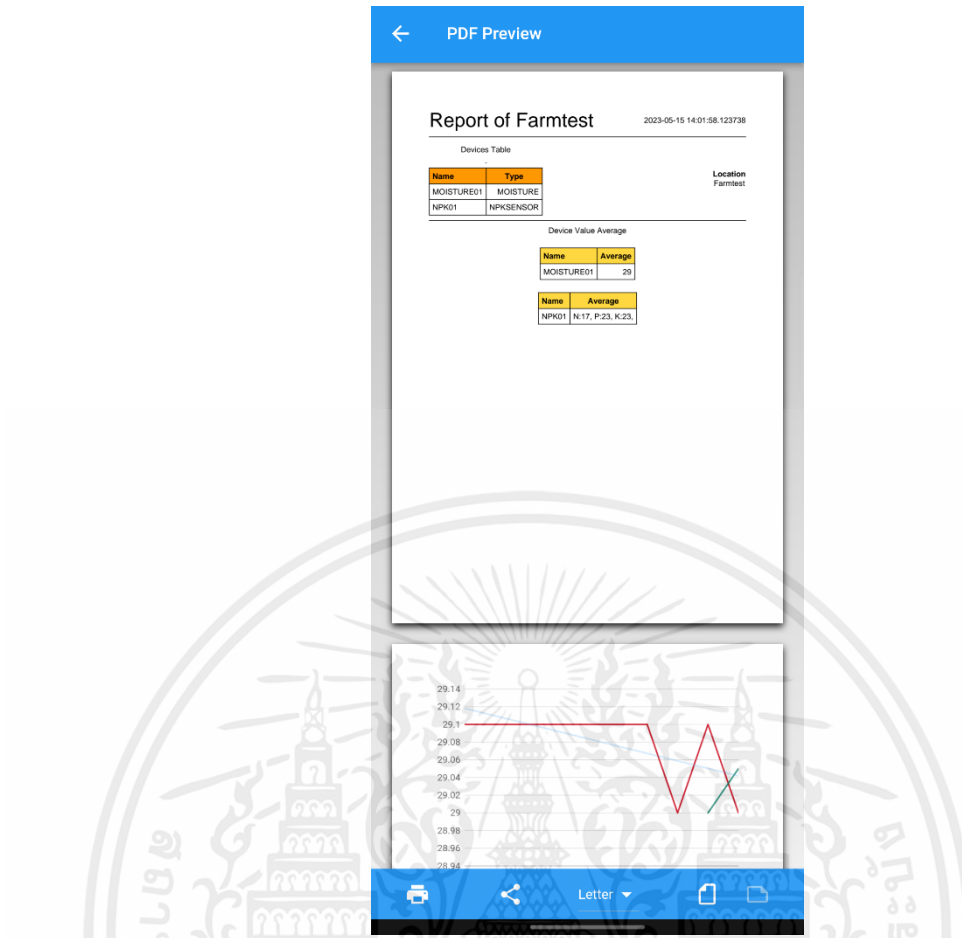
4.3.11 การส่งออกเอกสารค่าข้อมูลของอุปกรณ์

ผู้ใช้สามารถเข้าถึงหน้าการสร้างรายงานได้โดยการเข้ามาที่หน้าวิเคราะห์ก่อน และเลือกแถบ Report ในแรกเริ่มหน้านี้จะประกอบไปด้วย คำอธิบายการใช้ และปุ่มการสร้างรายงาน ถ้าหากผู้ใช้ไม่ได้กดเลือกอุปกรณ์ใดๆเลยในหน้าวิเคราะห์ จะเป็นการสร้างรายงานที่ไม่มีรูปภาพ ถ้าหากผู้ใช้ต้องการเพิ่มรูปภาพลงไป ในรายงาน ผู้ใช้ต้องกดส่งออกรูปจากหน้าวิเคราะห์แถบ Indicator ก่อนแล้วรูปก็จะมาปรากฏในหน้านี้ ซึ่งผู้ใช้สามารถเพิ่มอธิบายสำหรับกราฟนั้นๆได้โดยการกดที่ปุ่ม Write ภายใต้รูปภาพ หลังจากผู้ใช้พิมพ์ข้อความที่ต้องการเสร็จสิ้น ผู้ใช้ต้องกดเครื่องหมายถูกบนแป้นพิมพ์เพื่อยืนยันอีกครั้งแล้วจึงกด OK เมื่อผู้ใช้ต้องการสร้างรายงาน ให้กดที่ปุ่ม Make เพื่อนำไปสู่หน้ารายงานที่จะถูกสร้างขึ้นตามข้อมูลอุปกรณ์ ซึ่งสามารถบันทึกไฟล์ออกมาเป็นไฟล์สกุล pdf ได้



รูปที่ 4.42 หน้าการสร้างรายงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.43 หน้าแสดงผลไฟล์รายงาน

4.4 การใช้บริการต่างๆของ AWS ในระบบ

ในส่วนนี้จะเป็นการแสดงการใช้ AWS ในการให้บริการต่างๆ ที่ใช้เชื่อมต่อกับแอปพลิเคชัน ไม่ว่าจะเป็น AWS Cognito, AWS DynamoDB, AWS Amplify, AWS IoT Core ซึ่งทั้งหมดนี้ได้นำมาเชื่อมต่อเข้าด้วยกันผ่าน AWS Lambda โดยสามารถเรียกใช้ผ่าน API Gateway ด้วย REST API ได้ รวมถึงการนำบริการเหล่านี้มาใช้ในแอปพลิเคชันด้วย

4.4.1 การตั้งค่าทำระบบยืนยันตัวตน

หลังจากติดตั้ง amplify-cli เสร็จแล้ว ให้ทำการเพิ่มบริการที่ต้องการใช้ลงไปในโปรไฟล์ปัจจุบันด้วยขั้นตอนดังนี้ [21]

- 1. พิมพ์คำสั่ง amplify init เมื่อป้อนคำสั่งแล้ว จะมีการสร้างตัวเลือกมาให้ดังนี้
 - Name ชื่อของแอปพลิเคชัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- Environment ชื่อสภาพแวดล้อมสำหรับการพัฒนาโปรเจกต์
- Default editor โปรแกรมแก้ไขโค้ด
- App type ประเภทแอปพลิเคชัน
- Configuration file location ที่อยู่ของไฟล์บันทึกการตั้งค่าที่จะเก็บไว้ที่ไหน

จุดที่สำคัญมี App type ที่ต้องกำหนดเป็น flutter และ Configuration file location ที่กำหนดให้อยู่ในโฟลเดอร์เดียวกันกับโฟลเดอร์ lib หลังจากตั้งค่าเสร็จแล้ว จะมีการถามถึงการยืนยันตัวตนของผู้พัฒนาว่า ต้องการยืนยันด้วยวิธีใด ให้เลือก AWS access keys และนำ access key id กับ secret key ที่ได้สร้างไว้มาใส่

2. เมื่อเสร็จสิ้น ให้พิมพ์คำสั่ง `amplify add auth` ซึ่งจะขึ้นมาให้ถามดังต่อไปนี้

- ต้องการใช้ค่าเริ่มต้นของการยืนยันตัวตนหรือไม่ ให้เลือก Default configuration
- วิธีที่ใช้เข้าสู่ระบบ ให้เลือก Username
- การตั้งค่าเพิ่มเติม สามารถเลือก No, I am done หรือตั้งค่าต่อไปได้

ในกรณีที่ต้องการปรับเปลี่ยนการตั้งค่าอีกครั้ง ให้พิมพ์คำสั่ง `amplify update auth`

3. เมื่อตั้งค่าเสร็จสิ้นแล้ว พิมพ์คำสั่ง `amplify push` เพื่อนำการตั้งค่าปัจจุบันบันทึกลงเซิร์ฟเวอร์

ในแอปพลิเคชันจะมีหน้าการยืนยันตัวตน อันประกอบด้วยหน้าลงทะเบียนและหน้าการเข้าสู่ระบบ ซึ่งจำเป็นต้องใช้ library เพิ่มเติม โดยสามารถติดตั้งด้วยคำสั่ง `flutter pub add` และตามด้วยชื่อดังต่อไปนี้

- `amplify_flutter`
- `amplify_auth_cognito`
- `amplify_authenticator`

เมื่อติดตั้ง library ที่ต้องการเสร็จสิ้นแล้ว ให้ไปที่ไฟล์ที่ต้องการทำหน้าการยืนยันตัวตน และใส่โค้ดดังนี้ เมื่อ `AmplifyConfig` เป็นตัวแปรคงที่ที่นำเข้ามาจากไฟล์ `amplifyconfiguration.dart` ซึ่งถูกสร้างขึ้นจากตอนที่เพิ่มบริการ

```
import '../amplifyconfiguration.dart';
```

```
Future<void> _configureAmplify() async {
  final authPlugin = AmplifyAuthCognito();
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

await Amplify.addPlugin(authPlugin);

try{
    await Amplify.configure(amplifyConfig);
} on AmplifyAlreadyConfiguredException {
    print("Tried to re-configure; Android app was restarted");
}
}

```

โค้ดส่วนนี้จะเป็นการเพิ่มบริการการยืนยันตัวตนลงไปในแอปพลิเคชัน ซึ่งสามารถเรียกใช้ในหน้ายืนยันตัวตนได้ ในลักษณะนี้

```

class Login extends StatefulWidget{
    Login({Key? key, ... }): super(key: key);

    @override
    State<Login> createState() => _Login();
}

class _Login extends State<Login>{

    @override
    void initState(){
        super.initState();
        _configureAmplify();
    }
}

```

จากโค้ดจะเป็นการตั้งค่าบริการยืนยันตัวตนก่อนจะสร้างหน้ายืนยันตัวตน โดยเมื่อ widget นี้ถูกนำเข้าสู่โครงสร้างหลัก จะเรียกใช้ initState และก็จะไปเรียกฟังก์ชัน _configureAmplify ให้ทำงาน หลังจากตั้งค่าเสร็จสิ้นก็จะไปสร้างหน้าการยืนยันตัวตนด้วยวิธี build ซึ่งเป็นวิธีที่บังคับให้ทุก widget ต้อง override เพื่อสร้าง widget นั้นๆ โดยสามารถสร้างหน้าการยืนยันตัวตนด้วยการใช้ library amplify_authenticator ดังนี้

```

class _Login extends State<Login>{

```

```

...

@override
Widget build(BuildContext context){
  return Authenticator(
    authenticatorBuilder: (p0, p1){
      switch(p1.currentStep){
        case AuthenticatorStep.signUp:
          return SignUpForm();
        case AuthenticatorStep.signIn:
          return SignInForm();
        case ...: // other cases
          break;
      }
    },
    child: MaterialApp(
      builder: Authenticator.builder(),
      home: ...
    ),
  );
}
}

```

ด้วยการใช้ library นี้จะทำให้ไม่ต้องไปทำการยืนยันตัวตนด้วยตนเอง และสามารถใช้ widget สำเร็จรูปได้เลยทันทีที่ Authenticator จะเป็น widget หลักที่ทำหน้าที่คุมการเปลี่ยนสถานะของการยืนยันตัวตนว่ากำลังอยู่ในขั้นตอนใด โดยแรกเริ่มจะบังคับเป็นการเข้าสู่ระบบ ใน widget นี้จะมี 1 ค่าที่ต้องการ คือ child เพื่อใช้เป็นผลลัพธ์ที่ต้องการจะแสดงหลังจากยืนยันตัวตนได้สำเร็จ จะเห็นได้ว่าได้มีการเพิ่มค่าเข้าไปที่ widget นี้อีก 1 ค่า ก็คือ authenticatorBuilder ซึ่งเป็นค่าที่ไม่ต้องกำหนดก็ได้ ในค่านี้จะเป็นการใส่ฟังก์ชันเพื่อควบคุมการเปลี่ยนแปลงขั้นตอนการยืนยันตัวตน ให้แอปพลิเคชันแสดงผลตามหน้าที่เราต้องการ โดยปกติแล้ว widget นี้จะคุมขั้นตอนเอง จึงไม่จำเป็นต้องมาเพิ่มฟังก์ชันเข้าไป แต่ถ้าหากในกรณีที่ต้องการปรับเปลี่ยนรูปแบบหน้าตา จึงจำเป็นต้องมีฟังก์ชันนี้เพิ่มเข้ามาด้วย ซึ่งในแต่ละเคสของขั้นตอนต้องมีปุ่มที่ใช้เพื่อการเปลี่ยนหน้า ในกรณีตัวอย่าง เช่น เมื่อผู้ใช้ใช้แอปพลิเคชันนี้เป็นครั้งแรก ซึ่งอาจจะยังไม่มีบัญชีที่สมัครไว้ จึงต้อง

ไปลงทะเบียนก่อน แต่ในแอปพลิเคชันยังแสดงผลเป็นหน้าเข้าสู่ระบบ ในกรณีนี้จึงจำเป็นต้องทำปุ่มเปลี่ยนหน้าซึ่งเปลี่ยนขั้นตอนการยืนยันตัวตนได้ด้วย และสามารถทำได้ด้วยการเพิ่ม widget ดังนี้

```
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const Text(
      "Don't have an account? ",
      style: TextStyle(
        color: Colors.white,
        fontWeight: FontWeight.bold,
        fontFamily: 'OpenSans',
        fontSize: 14,
      ),
    ),
    TextButton(
      onPressed: () => p1.changeStep(AuthenticatorStep.signUp),
      child: const Text(
        "Sign up",
        style: TextStyle(
          color: Colors.orange,
          fontWeight: FontWeight.bold,
          fontFamily: 'OpenSans',
          fontSize: 14,
        ),
      ),
    ),
  ],
),
```

4.4.2 การทำฟังก์ชัน Serverless

หลังจากติดตั้ง serverless ผ่าน npm แล้ว ในขั้นตอนถัดไปจะเป็นการตั้งค่าไฟล์ต่างๆ เพื่อนำฟังก์ชันต่างๆขึ้นไปยังบน AWS ซึ่งมีขั้นตอนดังต่อไปนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1. ตั้งค่า Serverless บนเครื่องคอมพิวเตอร์ ด้วยคำสั่ง
 - a. `serverless config credentials --provider aws --key (1) --secret (2)`
 - b. (1) คือ access key id และ (2) คือ secret key ซึ่งสามารถใช้ร่วมกันกับที่เคยสร้างได้ หรือจะสร้างใหม่ก็ได้
2. เลือกที่อยู่ของโปรเจกต์ และพิมพ์คำสั่ง `serverless create --template aws-nodejs --path` (ใส่ที่อยู่ที่ต้องการ)

เมื่อเสร็จจากขั้นตอนที่ 2 จะได้ไฟล์มา 2 ไฟล์ คือ `serverless.yaml` และ `handler.js` โดยในแต่ละจะมีหน้าที่ คือ `serverless.yaml` จะเป็นไฟล์ที่ใช้ในการตั้งค่าบริการ AWS Lambda ว่าในแต่ละฟังก์ชันนั้นมีที่อยู่ไหน และสามารถเรียกใช้ได้อย่างไร รวมถึงการกำหนดทรัพยากรที่สามารถเข้าถึงได้ และไฟล์ `handler.js` จะเป็นไฟล์ที่ใช้ในการเขียนฟังก์ชันต่างๆ โดยในโปรเจกต์นี้จะใช้ภาษา javascript ในการเขียนฟังก์ชัน [22]

ในไฟล์ `serverless.yaml` มีจุดที่ต้องแก้ ดังต่อไปนี้

- หัวข้อ provider
 - region ให้แก้เป็นภูมิภาคที่ได้ deploy บริการต่างๆ ไว้ ในที่นี้ทำได้ทำไว้ที่ภูมิภาค `ap-southeast-1`
 - iam หากไม่มีให้เพิ่มส่วนนี้

iam:

role:

statements:

- Effect: "Allow"

Action:

- "s3:ListBucket"

Resource: { "Fn::Join" : ["", ["arn:aws:s3:::", { "Ref" : "ServerlessDeploymentBucket" }]] }

- Effect: "Allow"

Action:

- "s3:PutObject"

Resource:

Fn::Join:

- ""

- - "arn:aws:s3:::"

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- "Ref" : "ServerlessDeploymentBucket"
 - "/*"
 - Effect: "Allow"
 - Action:
 - dynamodb:Query
 - dynamodb:Scan
 - dynamodb:GetItem
 - dynamodb:PutItem
 - dynamodb:CreateTable
 - dynamodb:DeleteItem
 - หัวข้อ package สำหรับการบอกว่าไฟล์ไหนเกี่ยวข้องและถูกนำมาใช้ในการทำ Serverless package:
 - patterns:
 - '!'**'
 - handler.js
 - หัวข้อ resources สำหรับทรัพยากรบน AWS ที่ต้องนำมาใช้หรือสร้างขึ้นใหม่ ยกตัวอย่าง เช่น การสร้างฐานข้อมูลฟาร์ม จะสามารถใส่ลงไปได้ในลักษณะนี้
- resources:
- Resources:
- FarmDynamoDbTable:
- Type: "AWS::DynamoDB::Table"
- DeletionPolicy: Retain
- Properties:
- AttributeDefinitions:
- AttributeName: "ID"
 - AttributeType: "S"
- KeySchema:
- AttributeName: "ID"
 - KeyType: "HASH"
- ProvisionedThroughput:
- ReadCapacityUnits: 1
- WriteCapacityUnits: 1
- TableName: FarmDynamoDbTable

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในหัวข้อย่อยนี้จะถูกนำไปสร้างในระหว่างที่ Serverless กำลังนำข้อมูลไปสร้างโดยใช้ AWS CloudFormation ซึ่งในการสร้างฐานข้อมูลจะมีอยู่ 3 หัวข้อย่อย ได้แก่ Type, DeletionPolicy และ Properties

- Type จะเป็นการกำหนดว่าทรัพยากรนี้จะเป็นทรัพยากรประเภทอะไร ในที่นี้ต้องการจะสร้างตารางใน AWS DynamoDB จึงเป็น “AWS::DynamoDB::Table”
- DeletionPolicy คือคุณสมบัติที่จะนำมาใช้เมื่อทรัพยากรที่ได้ deploy ขึ้นไปบน CloudFormation ถูกลบออก จะทำอย่างไรกับทรัพยากรนี้ที่ถูกสร้างขึ้นมาด้วยทรัพยากรที่เพิ่งถูกลบออกไป ในที่นี้ใช้ Retain ซึ่งหมายความว่า แม้ทรัพยากรจะถูกลบออกไป แต่ตัวตารางนี้จะไม่ถูกลบไปด้วย
- Properties คือคุณสมบัติของตาราง
 - AttributeDefinitions คือการกำหนดว่าแอททริบิวต์ชื่อนี้เป็นประเภทอะไร ส่วนนี้ต้องสอดคล้องกับ KeySchema เพื่อกำหนดประเภทของกุญแจหลักว่าเป็นประเภทอะไร และในส่วนนี้ไม่สามารถสร้างแอททริบิวต์อื่นๆได้นอกจากกุญแจหลัก แต่ด้วยความยืดหยุ่นของฐานข้อมูล ทำให้การสร้างค่าอื่นๆนั้นสามารถเพิ่มขึ้นมาได้ภายหลัง
 - KeySchema คือการกำหนดว่าแอททริบิวต์ที่ต้องการจะสร้างนั้นเป็นกุญแจหลักประเภทอะไร ระหว่าง HASH และ RANGE ซึ่ง DynamoDB จะสามารถสร้างกุญแจหลักได้ใน 2 รูปแบบ คือ HASH และ HASH+RANGE โดยในแบบหลังนั้นต้องใช้ 2 กุญแจหลัก
 - ProvisionedThroughput คือการกำหนดว่าความสามารถในการอ่าน-เขียนที่เกี่ยวข้องกับตารางนี้จะสามารถอ่าน-เขียนได้กี่ข้อมูลพร้อมๆกัน
 - TableName คือชื่อของตาราง

ถ้าหากได้มีการกำหนดว่าจะสร้างฐานข้อมูลขึ้นมาใหม่เมื่อทำการ deploy ต้องเพิ่มหัวข้อ Resource ลงไปในหัวข้อ iam เป็นหัวข้อย่อย ในรูปแบบดังนี้

iam:

role:

statements:

...

Resource:

- "Fn::GetAtt": [FarmDynamoDbTable, Arn]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

และเพื่อความสะดวกในการเขียนฟังก์ชัน โดยที่ไม่ต้องไปเรียกชื่อตารางใหม่ทุกครั้ง สามารถทำให้ชื่อของตารางกลายเป็นตัวแปรในสภาพแวดล้อมของทุกครั้งฟังก์ชันทำงานได้ ซึ่งจะทำให้ฟังก์ชันสามารถเรียกใช้งานชื่อตารางได้ทันที สามารถทำได้ คือ เพิ่มหัวข้อย่อย environment ลงไปที่หัวข้อ provider และตามด้วยชื่อตัวแปรที่ต้องกับ Ref เป็นค่าของตัวแปรนั้น เช่น

environment:

FARM_TABLE:

Ref: FarmDynamoDbTable

ต่อไปจะเป็นส่วนของฟังก์ชัน ในไฟล์ handler.js โดยในครั้งแรกที่จะ deploy นั้นจะเป็นการสร้างฟังก์ชันเพื่อทดสอบการทำงาน โดยจะเขียนฟังก์ชันทดสอบการรับ-ส่ง ดังนี้

ให้ผลลัพธ์มีลักษณะดังต่อไปนี้

```
function getFarm() {
  return [
    {
      Name: 'first farm',
      ID: 'farm01',
      Owner: 'Adam',
      AllowedUsers: [
        "Adam",
        "John",
        "Robert"
      ],
      AvailableDevices: [
        "farm01.DEVICE_01",
        "farm01.DEVICE_02"
      ]
    },
  ];
}
```

และฟังก์ชันที่จะใช้ในการเรียกข้อมูลนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

module.exports.getFarmExample = (event, context, callback) => {
  let example = getFarm();
  var headers = {
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Credentials': true
  };
  const response = {
    statusCode: 200,
    headers,
    body: JSON.stringify(example),
  };
  callback(null, response);
};

```

ในฟังก์ชันจะถูกส่งออกไปในลักษณะของโมดูล ที่มีชื่อฟังก์ชันคือ `getFarmExample` ซึ่งมีพารามิเตอร์ที่ AWS กำหนดไว้ ได้แก่ `event`, `context` และ `callback` มีหน้าที่การทำงานคือ

- `event` เป็นส่วนที่ AWS จะส่งค่ามา เช่น ในกรณีที่มีการเรียกใช้ผ่านเว็บแอปพลิเคชัน และมีการส่งค่ามาใน payload เป็น `{“body”: “12345”}` ฟังก์ชันจะสามารถเข้าถึง “12345” ได้ด้วยการใช้ `event.body` เป็นต้น
- `context` เป็นวัตถุที่บันทึกค่าต่างๆที่เกี่ยวกับฟังก์ชันนี้ว่าทำงานอย่างไร ด้วยการใช้ภาษาใด ใช้หน่วยความจำในการประมวลผลเท่าไร และที่อยู่ของฟังก์ชันบน AWS เป็นต้น
- `callback` เป็นฟังก์ชันที่จะถูกเรียกใช้งานหลังการทำงานเสร็จสิ้น โดยจะมี 2 องค์ประกอบ ได้แก่ `Error` และผลลัพธ์ที่จะส่งกลับไป

ในฟังก์ชันนี้ จะเรียกฟังก์ชัน `getFarm` มาซึ่งเป็นผลลัพธ์ที่ต้องการมาเก็บไว้เป็นตัวแปร `example` และส่งกลับไปด้วยการใช้ `callback` ที่ให้ `Error` เป็นค่า `null` และผลลัพธ์เป็น `json` ที่มีโครงสร้างอันประกอบไปด้วย `statusCode` ที่ใช้บ่งบอกว่าผลลัพธ์นี้เป็นสถานะอะไร, `headers` คือหัวของโครงสร้างนี้ ใช้บ่งบอกคุณสมบัติของผลลัพธ์นี้ซึ่งจะเป็นต่อเว็บแอปพลิเคชัน แต่ถ้าหากเป็นแอปพลิเคชันบนมือถือก็ไม่ต้องส่ง `headers` ไปด้วย และ `body` ที่นำตัวแปร `example` มาแปลงจากวัตถุ `json` ให้เป็น `json string` เมื่อเขียนฟังก์ชันได้ดังนี้แล้ว ให้เพิ่มวิธีการเรียกใช้ฟังก์ชันได้ด้วยการเพิ่มที่อยู่ลงในไฟล์ `serverless.yaml`

functions:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

getFarmExample:
  handler: handler.getFarmExample
  events:
    - http:
        path: /farm/get/example
        method: GET
        cors: true

```

ในไฟล์ serverless.yaml ระหว่างหัวข้อ package กับ resources ให้เพิ่มหัวข้อ functions ลงไป ในลักษณะแบบด้านบน โดยกำหนดชื่อฟังก์ชัน หลังจากนั้นตามด้วยหัวข้อย่อย ได้แก่ handler และ events ใน handler จะเป็นการบอกว่าสามารถเรียกใช้ฟังก์ชันนี้ได้อย่างไร ในที่นี้ฟังก์ชันนี้อยู่ในไฟล์ handler.js มีชื่อฟังก์ชันว่า getFarmExample ดังนั้นที่หัวข้อนี้ จึงใส่ไปว่า

handler.getFarmExample ส่วนหัวข้อ events ให้เพิ่มหัวข้อย่อยเป็น http หมายถึงว่า จะเรียกใช้ฟังก์ชันนี้ด้วยวิธีอะไร ซึ่งเลือกเป็นโปรโตคอล HTTP และมีหัวข้อย่อยดังต่อไปนี้

- path หมายถึง ที่อยู่ของฟังก์ชันว่าจะสามารถเรียกใช้ฟังก์ชันนี้ได้ด้วยที่อยู่แบบใด ในส่วนนี้สามารถกำหนดได้โดยไม่ต้องคำนึงถึงที่อยู่ของฟังก์ชันภายในไฟล์
- method คือวิธีที่จะเรียกใช้ฟังก์ชันนี้ ในที่นี้คือวิธีการต่างๆของโปรโตคอล HTTP ในกรณีนี้เป็นการเรียกเพื่อแสดงผลลัพธ์โดยไม่ต้องส่งค่าใดๆไป จึงใช้ GET
- cors คือการบอกว่าฟังก์ชันนี้สามารถเรียกใช้ได้แม้ว่าผู้เรียกใช้กับฟังก์ชันจะอยู่คนละโดเมนกัน โดยกำหนดให้ค่าเป็น true เพื่ออนุญาต นอกจากนี้สามารถกำหนดให้เป็นโดเมนที่ต้องการโดยเฉพาะได้ด้วยการใส่ชื่อของโดเมนนั้นๆแทน

เมื่อเขียนฟังก์ชัน การตั้งค่าต่างๆ เสร็จสิ้นแล้ว ต่อไปจะเป็นการ deploy ครั้งแรกเพื่อให้ได้มีการสร้างฐานข้อมูลและสร้าง stack ที่บันทึกฟังก์ชัน serverless ลงไปบน CloudFormation ทำให้ครั้งถัดไปในการ deploy จะไม่เสียเวลามาทำหรือสร้างตารางต่างๆใหม่ โดยใช้คำสั่ง คือ serverless deploy เพื่อเริ่มการสร้างบริการ serverless หรือ AWS Lambda แต่หากมีการ deploy ไว้แล้ว สามารถใช้คำสั่ง serverless deploy -f <ชื่อฟังก์ชัน> แทนได้

เมื่อทำงานเสร็จสิ้น จะได้ที่อยู่ของฟังก์ชันมาในลักษณะนี้

```

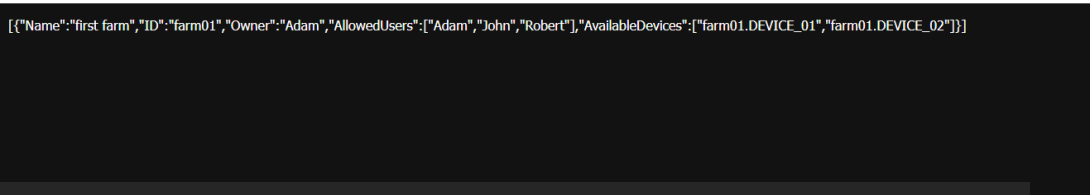
Service Information
service: smart-iot-app
stage: dev
region: ap-southeast-1
stack: smart-iot-app-dev
resources: 156
api keys:
  None
endpoints:
  GET - https://[redacted].execute-api.ap-southeast-1.amazonaws.com/dev/farm/get/example

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

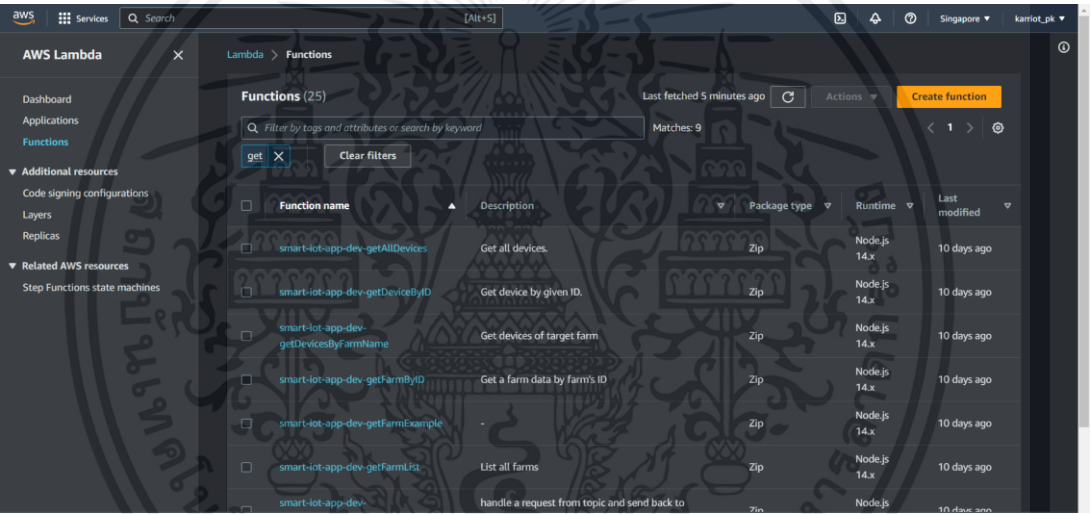
รูปที่ 4.44 การสร้าง บริการ serverless หรือ AWS Lambda

เมื่อเข้าไปตามลิงค์ของ endpoint ก็จะได้ผลลัพธ์ดังนี้



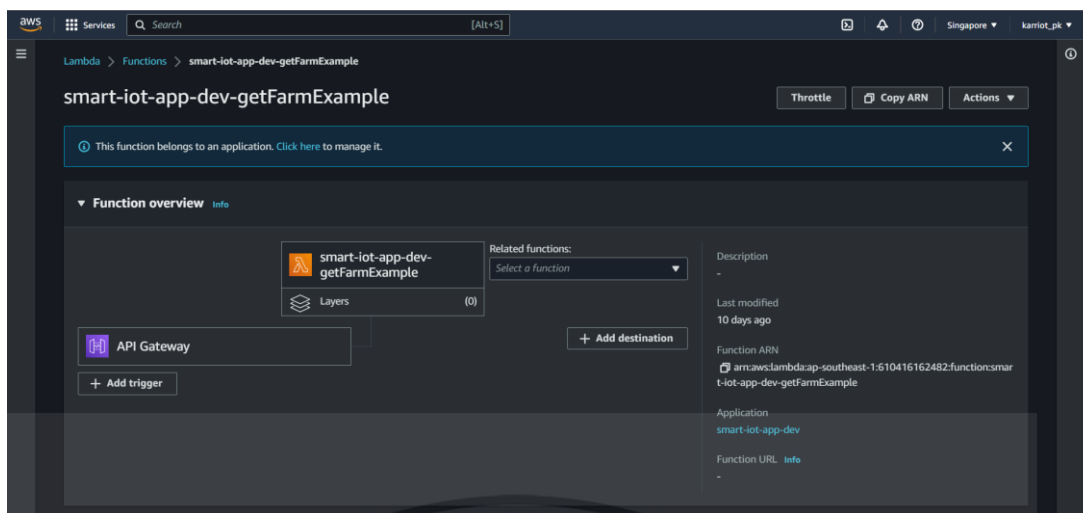
รูปที่ 4.45 ผลลัพธ์เมื่อกดที่ endpoint

ฟังก์ชันที่ deploy แล้วจะปรากฏบนหน้า AWS Lambda บน AWS console ดังนี้



รูปที่ 4.46 ฟังก์ชัน ภายใน AWS Lambda

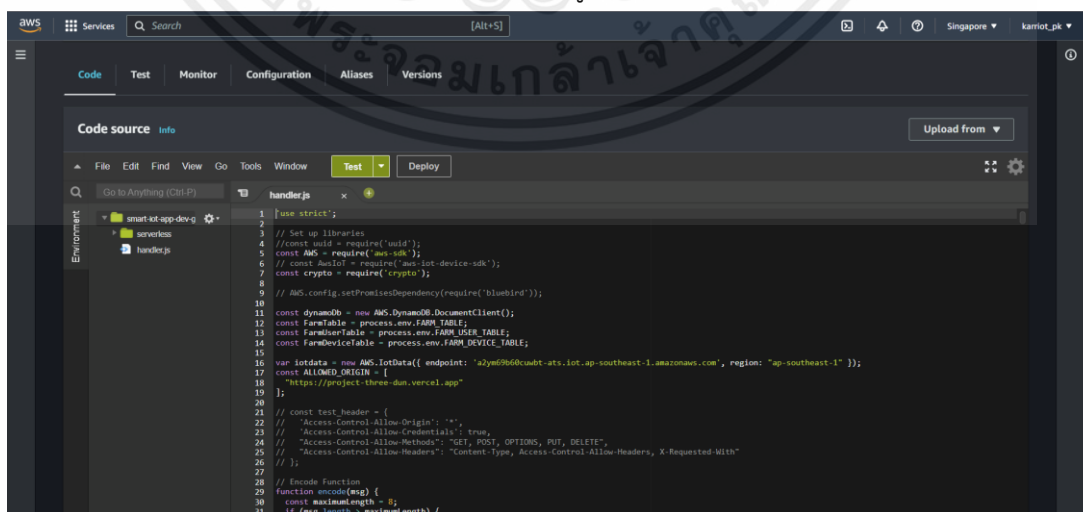
เมื่อเข้าไปในฟังก์ชันจะปรากฏเป็นหน้าดังนี้ โดยจะแสดงหน้าต่างของฟังก์ชันว่ามีการเชื่อมต่อจาก อินพุตที่ใช้ในการเรียกฟังก์ชันได้อย่างไรบ้างและเอาท์พุตเชื่อมต่อไปที่ใดบ้าง



รูปที่ 4.47 หน้าต่างภายในฟังก์ชัน

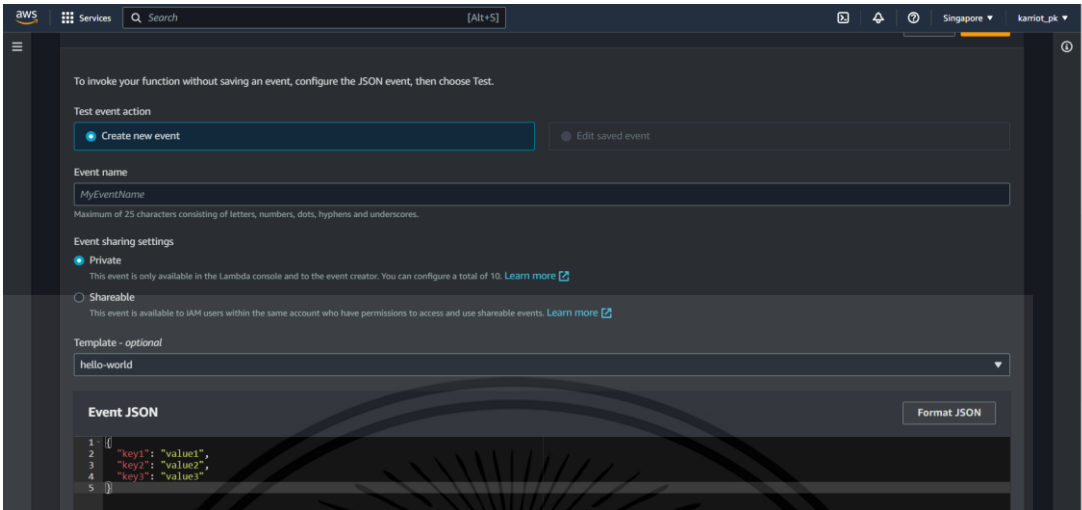
เมื่อเลื่อนลงมา จะเจอแถบเมนูในฟังก์ชันต่างๆ เพื่อใช้ในการดูฟังก์ชัน โดยแถบเมนูที่สำคัญมีดังนี้

- Code จะเป็นการแสดงไฟล์ที่มีโค้ดของฟังก์ชันนั้นๆ
- Test จะเป็นการทดสอบส่งข้อมูล json ไปที่ฟังก์ชันนั้น เพื่อทดสอบการทำงาน ในเมนูนี้จะเป็นการส่งภายใน AWS ทำให้ไม่สามารถเทียบการทดสอบจากการส่งแบบเว็บแอปพลิเคชันได้
- Monitor จะเป็นการบันทึกข้อมูลการทำงานของฟังก์ชันว่าทำงานมีผลลัพธ์อย่างไรบ้าง ในแถบนี้จะเป็นการใช้บริการ CloudWatch ซึ่งสามารถทำให้แจ้งเตือนการทำงานได้ถ้าหากพบว่ามีข้อผิดพลาด แต่ก็คิดค่าบริการเพิ่มด้วย
- Configuration จะเป็นการตั้งค่าฟังก์ชันนั้นๆ เช่น การตั้งค่าว่าจะมีอินพุตที่ทำให้ฟังก์ชันนี้ทำงานจากอะไรได้บ้าง เป็นต้น ซึ่งสามารถดูลิงค์ API จากแถบได้เช่นกัน

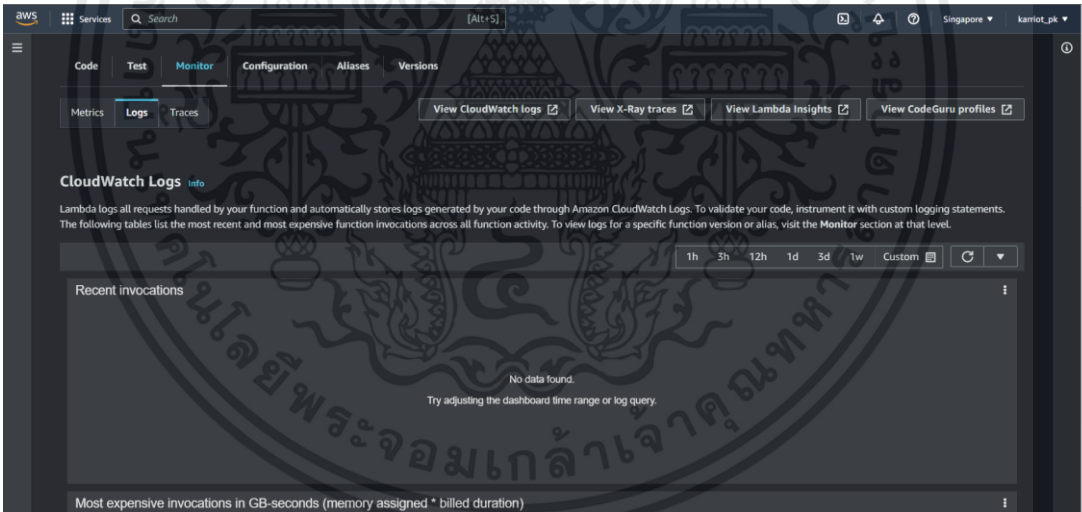


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

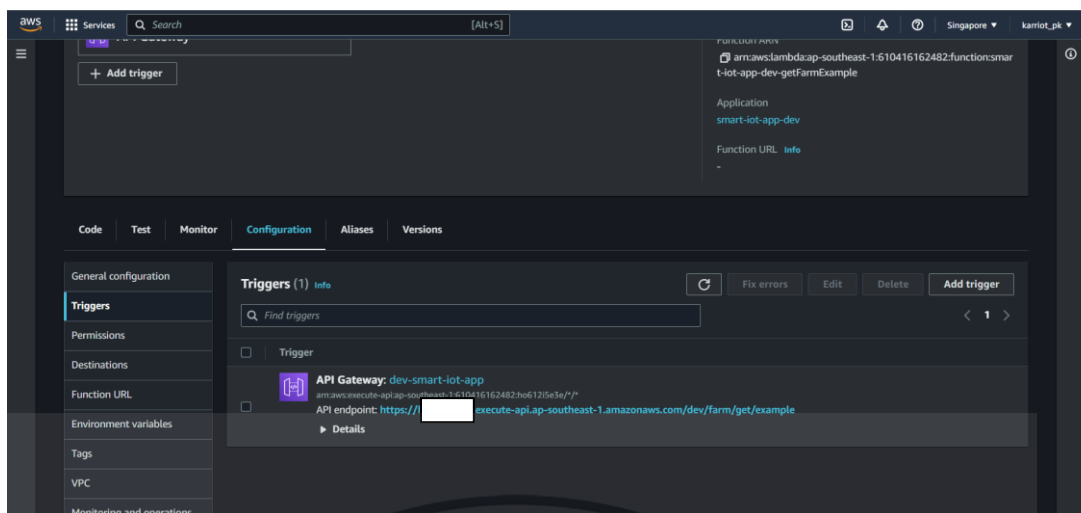
รูปที่ 4.48 แถบแสดงเมนู Code ของบริการ AWS Lambda



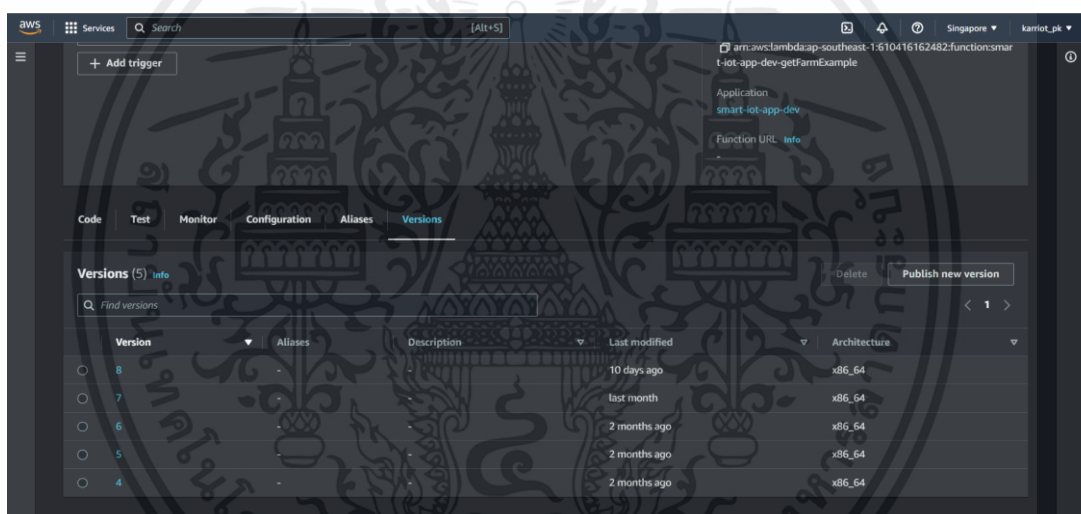
รูปที่ 4.49 แถบแสดงเมนู Test ของบริการ AWS Lambda



รูปที่ 4.50 แถบแสดงเมนู Monitor ของบริการ AWS Lambda



รูปที่ 4.51 แถบแสดงเมนู Configuration ของบริการ AWS Lambda



รูปที่ 4.52 แถบแสดงเมนู Version ของบริการ AWS Lambda

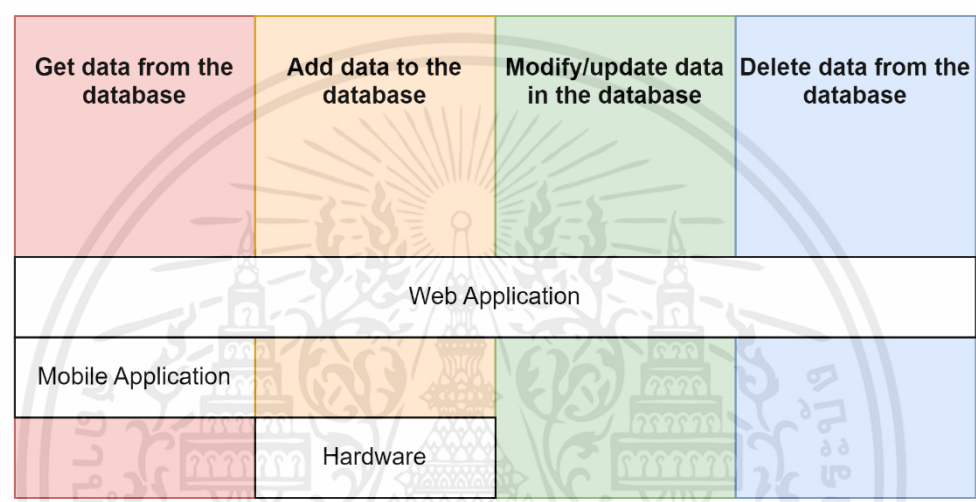
4.4.2.1 การจัดการข้อมูลต่างๆ กับ AWS DynamoDB

การเชื่อมต่อกับบริการ AWS DynamoDB นั้นมีจุดประสงค์เพื่อทำหน้าที่ในการเก็บ แก้ไข ลบและดึงข้อมูล ในแต่ละองค์ประกอบของระบบก็จะมีสิทธิของหน้าที่นี้ไม่เท่ากัน ดังนี้

- ฮาร์ดแวร์ จะไม่เข้าถึงบริการ AWS DynamoDB โดยตรง แต่จะผ่าน rule ของ AWS IoT Core อีกที ซึ่งจะมีสิทธิได้แก่ การเก็บข้อมูล เพียงอย่างเดียวเท่านั้น

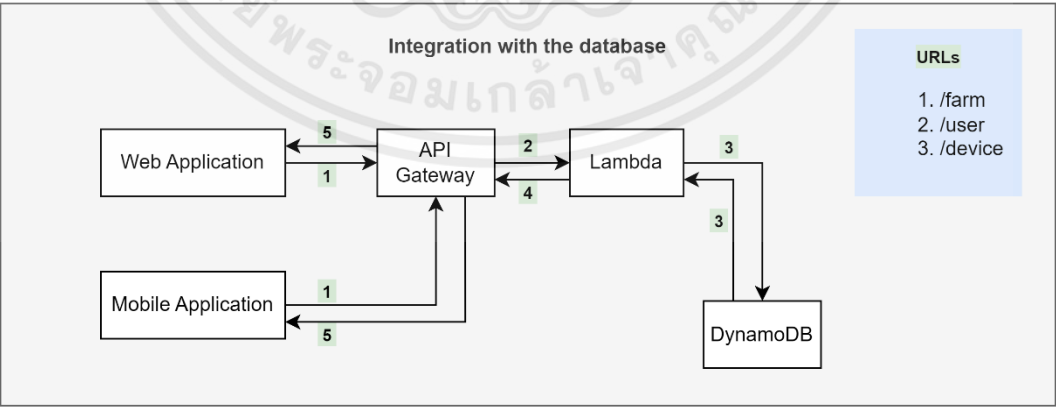
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- เว็บแอปพลิเคชัน จะทำหน้าที่เป็นคอนโซลในการจัดการข้อมูลในฐานข้อมูลต่างๆ รวมถึงการควบคุมอุปกรณ์ด้วย ซึ่งสิทธิในการเข้าถึงนี้จะได้ครบตามจุดประสงค์ แต่การเรียกใช้บนเว็บจะต้องผ่านฟังก์ชัน AWS Lambda เท่านั้น
- แอปพลิเคชันบนมือถือ จะหน้าที่แสดงผลค่า และควบคุมการทำงาน ดังนั้นสิทธิในการเข้าถึงข้อมูลจะมี การดึงข้อมูล เพียงอย่างเดียว ส่วนในเรื่องการบันทึกข้อมูลนั้นจะไม่มี เพราะอาจมีเหตุการณ์ในกรณีที่บันทึกข้อมูลพร้อมกันกับฝั่งฮาร์ดแวร์ และเพื่อลดการทำงานของฟังก์ชันที่ซ้ำกันหลายรอบ



รูปที่ 4.53 สิทธิในฐานข้อมูลของแต่ละองค์ประกอบ

จากรูป เมื่อนำมาเขียนเป็นแผนผังฟังก์ชันการใช้งาน จะได้ลักษณะนี้



รูปที่ 4.54 แผนผังการเชื่อมต่อกับบริการ DynamoDB

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากรูปจะเห็นได้ว่า มีทั้งหมด 5 ขั้นตอนด้วยกัน เริ่มจากขั้นตอนที่ 1 แอปพลิเคชันส่งคำร้องขอ GET หรือ POST ไปที่ url ที่เป็นที่อยู่ของฟังก์ชันหรือก็คือลิงค์ endpoint นั้นเอง ลิงค์ดังกล่าวจะมีลักษณะเป็น API ที่สามารถใช้ในการเรียกฟังก์ชันต่อได้ ในขั้นตอนที่ 2 ก็จะเป็นการเรียกฟังก์ชันตามที่อยู่ที่กำหนดไว้ ในขั้นตอนที่ 3 เมื่อฟังก์ชันถูกเรียกใช้งาน ในที่นี้จะเป็นการทำการกระบวนกรของฐานข้อมูล DynamoDB ซึ่งเมื่อได้ผลลัพธ์เสร็จสิ้น ก็จะส่งกลับไปที่ฟังก์ชัน ในขั้นตอนที่ 4 จะเป็นการส่งผลลัพธ์กลับมาจากฟังก์ชันที่ทำงานเสร็จแล้วและนำไปแสดงผลลัพธ์บนแอปพลิเคชัน ในรูปนี้จะไม่มีการรวมส่วนของฮาร์ดแวร์มาเกี่ยวข้องเนื่องจากการใช้บริการของ AWS IoT Core ซึ่งจะอยู่ในหัวข้อถัดไป

ในส่วนของ url ที่ใช้เป็นลิงค์สำหรับการเรียก API มาใช้งานนั้นจะแบ่งออกตามตารางของข้อมูลซึ่งมีทั้งหมด 3 ตารางด้วยกัน คือ ฟาร์ม ผู้ใช้และอุปกรณ์ โดยในแต่ละลิงค์ url จะมีการต่อด้วย “/” และตามหลังด้วยชื่อ ซึ่งหมายถึงว่าลิงค์นี้จะใช้ในการทำอะไรกับข้อมูลจากตารางนั้นๆ ฟังก์ชันของแต่ละตารางจะมีดังนี้

- ฟาร์ม “/farm”
 - “/farm/list/all” ใช้ GET แสดงรายชื่อฟาร์มทั้งหมด
 - “/farm/get/{ID}” ใช้ GET แสดงรายละเอียดของฟาร์มนั้นๆจากไอดีของฟาร์ม
 - “/farm/create” ใช้ POST สร้างรายการฟาร์มใหม่ลงไปตารางฟาร์ม
 - “/farm/create_as_table” ใช้ POST สร้างตารางฟาร์มที่เป็นชื่อฟาร์มนั้นๆ
- ผู้ใช้ “/user”
 - “/user/create” ใช้ POST สร้างผู้ใช้ใหม่ลงไปตาราง
 - “/user/list/all” ใช้ GET แสดงรายชื่อผู้ใช้ทั้งหมดในตาราง
 - “/user/get/{ID}” ใช้ GET แสดงรายละเอียดของผู้ใช้ตามไอดี
- อุปกรณ์ “/device”
 - “/device/all” แสดงอุปกรณ์ทั้งหมดในตาราง
 - “/device/in/{ID}” แสดงอุปกรณ์ตามไอดี
 - “/device/getByFarm” แสดงอุปกรณ์ตามชื่อฟาร์ม

ส่วนที่อัปเดตหรือเปลี่ยนแปลงข้อมูลของทุกตารางจะใช้ร่วมกันคือ “/any/targetTable/update” ด้วยวิธีการ POST ส่วนวิธีอื่นได้แก่ การเก็บข้อมูลลงในตารางของฮาร์ดแวร์ การลงทะเบียนอุปกรณ์ การจัดการข้อมูลจากหัวข้อ MQTT และการควบคุมค่าของอุปกรณ์จะรวมอยู่ในหัวข้อที่ 4.4.2.2 การเชื่อมต่อกับ AWS IoT Core เพื่อจัดการข้อมูลจากอุปกรณ์

การดำเนินการในแต่ละฟังก์ชันที่เกี่ยวข้องกับ AWS DynamoDB

ก่อนจะเพิ่มฟังก์ชันใหม่ ให้เพิ่มบรรทัดเหล่านี้เข้าไปในไฟล์ handler.js

```
// required for development;

const AWS = require('aws-sdk');

// dynamodb;

const dynamoDb = new AWS.DynamoDB.DocumentClient();

// process environments;

// created in serverless.yaml in "environment" section.

const FarmTable = process.env.FARM_TABLE;

const FarmUserTable = process.env.FARM_USER_TABLE;

const FarmDeviceTable = process.env.FARM_DEVICE_TABLE;
```

- AWS เป็นตัวแปรคงที่ที่ใช้เข้าถึงบริการต่างๆของ AWS
- dynamoDb เป็นตัวแปรคงที่ที่ใช้ในการจัดการฐานข้อมูล DynamoDB
- ค่าตัวแปรคงที่ 3 ตัวที่ใช้ในการอ้างอิงถึงชื่อตารางที่กำหนดไว้ในไฟล์ serverless.yaml โดยสามารถเรียกใช้ได้ด้วย process.env และตามชื่อที่ประกาศไว้ในไฟล์

1. การดึงข้อมูลในตารางทั้งหมด ในส่วนนี้จะกล่าวถึง “/farm/list/all”, “/user/list/all” และ “/device/all” ซึ่งมีลักษณะในฟังก์ชันที่คล้ายคลึงกันแต่แตกต่างกันที่ชื่อตารางและผลลัพธ์ที่ส่งกลับมา โดยในตารางฟาร์มและผู้ใช้จะส่งกลับมาแค่ไอดีและชื่อ ในขณะที่อุปกรณ์จะส่งกลับมา

ทั้งหมด ในการดึงข้อมูลทั้งหมดในตารางนี้ จะใช้วิธี scan [23] ของ DynamoDB ซึ่งเป็นการดึงข้อมูลทั้งหมดภายในตารางนั้นๆ สามารถทำได้ ดังนี้

```
dynamoDb.scan({
  TableName: // Table name from process.env
  ProjectionExpression: "ID, Name"
}, (err, data) => {
  if(err){
    callback(err);
  } else {
    return callback(null, {
      statusCode: 200,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    headers,
    body: JSON.stringify({
      results: data.Items
    })
  });
}
}).promise();

```

วิธี scan จะมี 2 ค่าที่ต้องการ ได้แก่วัตถุประเภท json ที่บังคับต้องมีกุญแจชื่อ TableName โดยนำตัวแปรคงที่ที่กำหนดไว้ เช่น FarmTable มาใส่ที่จุดนี้ได้ และฟังก์ชัน callback ซึ่งจริงๆ แล้วไม่จำเป็นต้องใส่ก็ได้ แต่ในที่นี้ต้องใส่ไว้เพื่อในกรณีที่ค่าที่ได้เป็นค่าผิดพลาดจะทำให้สามารถจัดการได้ว่าควรส่งผลลัพธ์กลับไปให้ผู้เรียกใช้อย่างไร นอกจากนี้จะเห็นได้ว่าวัตถุ json ว่ามีกุญแจในวัตถุอีก 1 ค่า คือ “ProjectionExpression” ซึ่งใช้ในการกำหนดว่าหลังจากทำการสแกนเสร็จแล้ว จะให้เหลือค่าใดบ้างที่จะส่งกลับไปเป็นผลลัพธ์

เมื่อได้ดังนี้แล้วสามารถนำไปสร้างฟังก์ชันได้ 3 ฟังก์ชันดังนี้

```

module.exports.getFarmList = (event, context, callback) => {
  // above dynamodb function
};

module.exports.getUserList = (event, context, callback) => {
  // above dynamodb function
};

module.exports.getAllDevices = (event, context, callback) => {
  // above dynamodb function
};

```

และไปสร้างที่อยู่ API ในไฟล์ serverless.yaml หัวข้อ functions ดังนี้

functions:

getFarmList:

handler: handler.getFarmList

description: List all farms

events:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- http:

path: /farm/list/all

method: GET

cors: true

getUserList:

handler: handler.getUserList

description: Return a list of users with id

events:

- http:

path: /user/list/all

method: GET

cors: true

getAllDevices:

handler: handler.getAllDevices

description: Get all devices.

events:

- http:

path: /device/all

method: GET

cors: true

เมื่อได้ดังนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิงค์ endpoint ของแต่ละฟังก์ชันมา

2. การดึงของข้อมูลในตารางที่ตรงกับไอดี ในส่วนนี้จะกล่าวถึง “/farm/get/{ID}”, “/user/get/{ID}” และ “/device/in/{ID}” โดยจะเป็นการนำไอดีของรายการที่ต้องการมาใส่ไว้ใน url เลย จะได้ผลลัพธ์ออกมาเป็นรายละเอียดข้อมูลที่มีไอดีตรงกับที่ใส่ไว้ใน url ในการดึงข้อมูลลักษณะนี้จะใช้ get [23] เพราะว่าไอดีที่ใช้เป็นกุญแจหลักซึ่งไม่ซ้ำกับรายการอื่นๆ และต้องการผลลัพธ์เพียงอันเดียว สามารถทำได้ดังนี้

```
dynamoDb.get({
  TableName: // Table name from the process.env
  Key: {
    ID: event.pathParameters.ID
  }
}).promise().then(
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

result => {
  const response = {
    statusCode: 200,
    headers,
    body: JSON.stringify(result.Item),
  };
  callback(null, response);
}
).catch(error => {
  callback(new Error("couldn't fetch by given ID"))
});

```

จุดที่แตกต่างจากการดึงข้อมูลทั้งหมด ก็คือ ในส่วนนี้ต้องการดึงข้อมูลเฉพาะของรายการที่ต้องการที่รู้ไอดีหรือกุญแจหลักของรายการนั้นแล้ว จึงดึงมาทั้งหมดโดยไม่ได้กำหนดว่าจะจำกัดการแสดงค่าของรายการนั้น ในวิธี get นี้จะใช้ค่า 2 ค่าคือ วัตถุ json ที่มีกุญแจชื่อ TableName และ Key โดยกำหนดกุญแจเป็นไอดีที่ผู้เรียกใส่มาในพารามิเตอร์ของ url ซึ่งสามารถเข้าถึงค่าใน url ได้ด้วยการใช้ event.pathParameter.ID และได้ใส่ promise() ต่อท้ายเข้าไปด้านหลังเหมือนการดึงข้อมูลทั้งหมด แต่ในคราวนี้มีเพิ่ม then และ catch เข้าไป เพื่อจัดการผลลัพธ์หลังจากที่ get เสร็จสิ้น ในที่นี้ then จะเป็นการนำผลลัพธ์ดังกล่าวมาจัดการต่อ ไม่สนใจว่าผลลัพธ์นั้นจะผิดพลาดหรือไม่ โดยนำมาสร้างเป็นค่าที่จะส่งกลับไปยังผู้เรียกใช้ฟังก์ชันนี้ และจบด้วย callback แต่ในกรณีที่มีข้อผิดพลาดเกิดขึ้น ก็ใช้ catch ต่อท้ายเพื่อป้องกันค่าผิดพลาดที่จะเกิดขึ้น ซึ่งจะเป็นการส่งค่าผิดพลาดว่า ไม่สามารถค้นหารายการตามไอดีที่ผู้เรียกให้มาได้

เมื่อได้ดังนี้แล้วสามารถนำไปสร้างฟังก์ชันได้ 3 ฟังก์ชันดังนี้

```

module.exports.getFarmByID = (event, context, callback) => {
  // above dynamodb function
};

module.exports.getUserByID = (event, context, callback) => {
  // above dynamodb function
};

module.exports.getDeviceByID = (event, context, callback) => {
  // above dynamodb function
};

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
};
```

และไปสร้างที่อยู่ API ในไฟล์ serverless.yaml หัวข้อ functions ดังนี้

getFarmByID:

handler: handler.getFarmByID

description: Get a farm data by farm's ID

events:

- http:

path: /farm/get/{ID}

method: GET

cors: true

getUserByID:

handler: handler.getUserByID

description: Get a data of user by given ID

events:

- http:

path: /user/get/{ID}

method: GET

cors: true

getDeviceByID:

handler: handler.getDeviceByID

description: Get device by given ID.

events:

- http:

path: /device/in/{ID}

method: GET

cors: true

เมื่อได้ดังนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิงค์ endpoint ของแต่ละฟังก์ชันมา

3. การดึงข้อมูลจากตารางด้วยชื่อของฟาร์ม ในส่วนนี้จะกล่าวถึง “/device/getByFarm” จะมีลักษณะคล้ายกันกับการดึงข้อมูลจากตารางด้วยไอดี แต่จะใช้ชื่อฟาร์มแทน ผลลัพธ์ที่ได้จะเป็น

ข้อมูลอุปกรณ์ที่อยู่ในฟาร์มนั้นทั้งหมด วิธีนี้จะใช้ได้ในส่วนแอปพลิเคชันบนมือถือเพื่อแสดงผลในหน้าแดชบอร์ดหลัก สามารถทำได้ดังนี้

```
module.exports.getDevicesByFarmName = async (event, context, callback) => {
  let data = JSON.parse(event.body);
  const location = data.DeviceLocation;
  let params = {
    TableName: FarmDeviceTable,
  };
  var raw = await dynamoDb.scan(params).promise();
  let resultList = {
    Devices: [],
  };
  raw.Items.forEach((device, index, arr) => {
    if(device.Location === location){
      resultList.Devices.push(device);
    }
  });
  callback(null, {
    statusCode: 200,
    body: JSON.stringify(resultList.Devices),
  });
};
```

ในฟังก์ชันนี้จะสามารถแบ่งขั้นตอนการทำงานได้ 3 ขั้นตอนด้วยกัน คือ การสแกนข้อมูลในตาราง การเทียบชื่อที่อยู่ตรงกัน และ การส่งค่าผลลัพธ์กลับไปยังผู้เรียกใช้ ในการสแกนข้อมูลจะใช้เหมือนกันกับการสแกนข้อมูลทั้งหมด แต่จะไม่มีใส่ Key และ ProjectionExpression ในการสแกนครั้งนี้ ต้องใส่ await ไว้ด้วย เพื่อให้เก็บข้อมูลหลังจากการสแกนทั้งหมด ถ้าไม่ใส่ await จะทำให้ในบรรทัดทำงานแต่จะไม่รอผลลัพธ์ ทำให้อาจเกิดข้อผิดพลาดได้ เช่น ข้อมูลส่งมาคนละเวลากัน เป็นต้น เมื่อได้ข้อมูลจากการสแกนมาแล้ว ในขั้นตอนถัดมาจะเป็นการเปรียบเทียบด้วยการใช้ forEach ซึ่งเป็นการวนซ้ำเพื่อนำข้อมูลแต่ละตัวออกมา เมื่อข้อมูลแต่ละอันมีที่อยู่ตรงกับชื่อฟาร์มที่กำหนด ก็จะเพิ่มข้อมูลนั้นลงไป ใน array ที่จะส่งกลับ และในขั้นตอนสุดท้ายก็จะส่ง array ที่มีข้อมูลที่มีที่อยู่ตรงกันทั้งหมดกลับไป

และไปสร้างที่อยู่ API ในไฟล์ serverless.yaml หัวข้อ functions ดังนี้

```
getDevicesByFarmName:
  handler: handler.getDevicesByFarmName
  description: Get devices of target farm
  events:
    - http:
        path: /device/getByFarm
        method: POST
```

เมื่อได้ดังนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิงค์ endpoint ของฟังก์ชันขึ้นมา

4. การสร้างข้อมูลในตาราง ในส่วนนี้จะกล่าวถึง “/farm/create” และ “/user/create” ใช้ในการสร้างข้อมูลฟาร์มและผู้ใช้งานไปในตารางตามลำดับ ในส่วนนี้จะถูกใช้โดยเว็บแอปพลิเคชันเท่านั้น เพื่อให้ผู้ดูแลจัดการข้อมูลได้ ในการสร้างข้อมูลในฐานข้อมูล จะใช้ put [23] ในการสร้างข้อมูล สามารถสร้างฟังก์ชันได้ดังนี้

สร้างโครงสร้างที่จะนำไปใส่ตามค่าในตารางฟาร์ม และผู้ใช้ โดยสร้างเป็นตัวแปรคงที่ที่จะรับค่าในลักษณะของฟังก์ชันและส่งกลับมาเป็นวัตถุ json

```
const farmInfo = (name, owner, allowed, availableDevices) => {
  const timestamp = new Date().getTime();
  return {
    ID: encode(name),
    FarmName: name,
    FarmOwner: owner,
    AllowedUsersInFarm: allowed,
    AvailableDevicesInFarm: availableDevices,
    CreateAt: timestamp,
  };
}
```

```
const userInfo = (user, email, owned) => {
  const id = crypto.randomUUID();
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

const timestamp = new Date().getTime();

return {
  ID: id,
  FarmUser: user,
  Email: email,
  OwnedFarm: owned,
  Role: "user", // default
  Permissions: ["WaitForFarmsDataConfirmation", "WaitForRoleConfirmation"],
  CreateAt: timestamp,
};
};

```

เมื่อได้โครงสร้างแล้ว ก็ใช้ put เพื่อนำโครงสร้างเหล่านี้เข้าไปในฐานข้อมูลได้ ดังนี้

```

dynamoDb.put({
  TableName: // FarmTable, FarmUserTable
  Item: // farmInfo(name, owner, allowed, availableDevices), userInfo(user, email, owned),
}).promise().then(res => {
  callback(null, {
    statusCode: 200,
    headers,
    body: JSON.stringify({
      message: "Successfully created " + // res.FarmName, res.FarmUser,
      id: res.id
    }),
  })
}).catch(err => {
  callback(null, {
    statusCode: 500,
    headers,
    body: JSON.stringify({
      message: "Unable to create"
    })
  })
});

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โดยวิธี put จะมี 2 ค่าที่ใช้ คือ TableName แทนด้วยชื่อตารางจาก process.env และ Item ให้นำโครงสร้างไปใส่ที่ Item จากนั้นเมื่อใส่ข้อมูลสำเร็จ ให้ส่ง json ที่มีข้อความกับชื่อที่นำเข้าไปในตารางกลับไปให้ผู้เรียกใช้ แต่ถ้าหากทำไม่สำเร็จ จะไม่ส่งค่าข้อผิดพลาดกลับไป แต่จะใช้เป็นการส่งโค้ดสถานะ 500 กลับไปแทนซึ่งหมายถึงการกระทำในเซิร์ฟเวอร์ผิดพลาด และสามารถนำไปสร้างเป็นโมดูลฟังก์ชันได้ดังนี้

```
module.exports.createFarm = async (event, context, callback) => {
  // insert check data type ...
  // generate structure to put
  const farmInfo = () => {};
  dynamoDb.put({...}).promise().then().catch();
};

module.exports.createUser = async (event, context, callback) => {
  // insert check data type ...
  // generate structure to put
  const userInfo = () => {};
  dynamoDb.put({...}).promise().then().catch();
};
```

และไปสร้างที่อยู่ API ในไฟล์ serverless.yaml หัวข้อ functions ดังนี้

```
createFarm:
  handler: handler.createFarm
  description: Send a farm information
  events:
    - http:
        path: /farm/create
        method: POST
        cors: true
createUser:
  handler: handler.createUser
  description: Send an user information
  events:
    - http:
        path: /user/create
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

method: POST

cors: true

เมื่อได้ดังนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิงค์ endpoint ของฟังก์ชันขึ้นมา

5. การสร้างตารางฟาร์มตามชื่อฟาร์มที่กำหนด ในส่วนนี้จะกล่าวถึง “/farm/create_as_table”

ใช้สำหรับการสร้างฟาร์มที่เก็บค่าตามเวลาจริงจากอุปกรณ์ ซึ่งจะไม่สร้างขึ้นตั้งแต่ตอน deploy ครั้งแรกแต่จะถูกสร้างผ่านการจัดการของเว็บแอปพลิเคชันเท่านั้น โดยเริ่มจากการกำหนดค่าต่างๆ ที่ใช้ในการสร้างตารางเป็นตัวแปรคงที่ ดังนี้

```
const params = {
  AttributeDefinitions: [
    {
      AttributeName: 'DeviceID',
      AttributeType: 'S'
    }
  ],
  KeySchema: [
    {
      AttributeName: 'DeviceID',
      KeyType: 'HASH',
    }
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1
  },
  TableName: table_name_from_body,
  StreamSpecification: {
    StreamEnabled: false
  }
};
```

เช่นเดียวกันกับการสร้างตารางตอน deploy ในไฟล์ serverless.yaml ครั้งแรก ได้กำหนดว่า
 กฎหลักในตารางนี้เป็น DeviceID และมีความสามารถในการอ่าน-เขียนเป็น 1 ข้อมูลต่อครั้ง ใน
 การสร้างตารางจะไม่ใช้ตัวแปร dynamoDb ที่ได้สร้างไว้แล้ว แต่จะสร้างตัวแปรใหม่ที่เป็นวัตถุ
 บริการ DynamoDB ดังนี้

```
let ddb = new AWS.DynamoDB();
```

ข้อแตกต่างจากตัวแปรที่สร้างไว้แล้วคือ ตัวแปร dynamoDb จะเป็นการยุ่งเกี่ยวกับตารางใดๆ
 ในขณะที่ตัวแปร ddb นี้จะเป็นยุ่งเกี่ยวกับบริการ DynamoDB โดยตรง จากนั้นสามารถนำไปใช้
 สร้างตารางได้ ดังนี้

```
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error: ", err);
  } else {
    console.log("Success => ", data);
  }
})
callback(null, {
  statusCode: 200,
  headers,
  body: JSON.stringify({ message: "create farm as table successfully" })
});
```

เมื่อนำส่วนต่างๆมารวมกันก็จะได้โมดูลฟังก์ชันดังนี้

```
module.exports.createFarmAsTable = (event, context, callback) => {
  const params = ...;
  let ddb = new AWS.DynamoDB();
  ddb.createTable(params, (err, data) {...});
  callback(null, {...});
};
```

และไปสร้างที่อยู่ API ในไฟล์ serverless.yaml หัวข้อ functions ดังนี้

```
registerDevice:
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
 ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
handler: handler.registerDevice

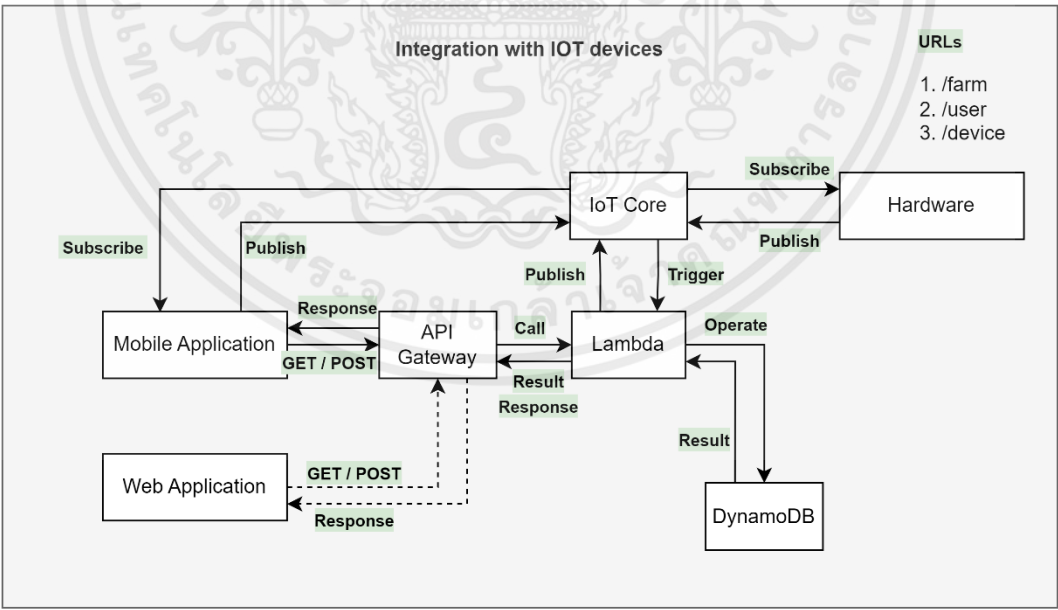
description: Use for first time register of new device.

events:
  - http:
      path: /device/register
      method: POST
      cors: true
```

เมื่อได้ตั้งนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิ้งค์ endpoint ของฟังก์ชันขึ้นมา

4.4.2.2 การเชื่อมต่อกับ AWS IoT Core เพื่อจัดการข้อมูลจากอุปกรณ์

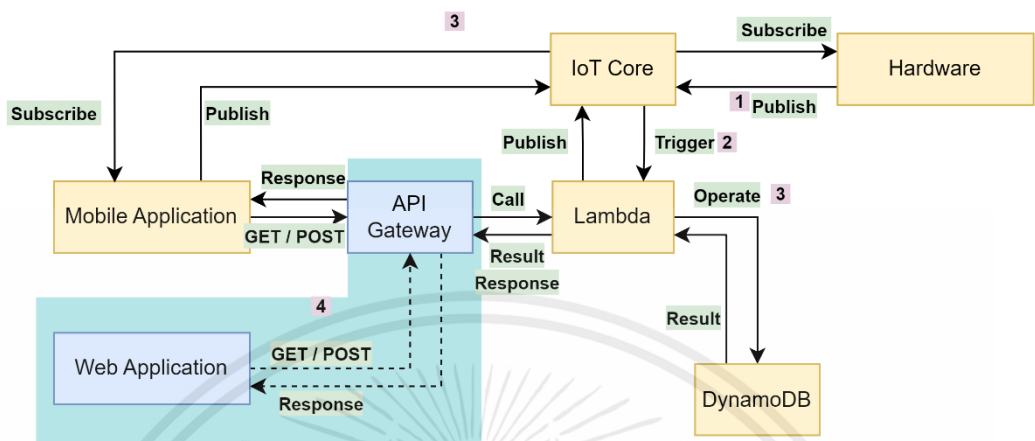
การเชื่อมต่อกับบริการ AWS IoT Core เป็นการทำให้ข้อมูลจากอุปกรณ์ส่งและบันทึกลงในฐานข้อมูล ในขณะเดียวกัน ก็เป็นการแสดงผลไปที่แอปพลิเคชัน เช่น การดูค่าของอุปกรณ์ในแอปพลิเคชันบนมือถือ หรือ เว็บแอปพลิเคชัน โดยในส่วนของบนมือถือจะเป็นการใช้โปรโตคอล MQTT ควบคู่กับ AWS Lambda และเว็บจะใช้ AWS Lambda อย่างเดียวเท่านั้น



รูปที่ 4.55 แผนผังการเชื่อมต่อกับบริการ IoT Core

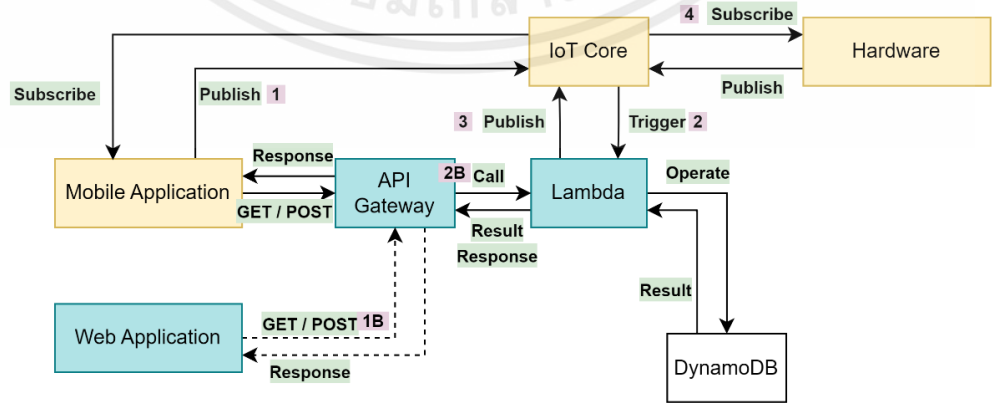
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากแผนผังจะสามารถแบ่งการทำงานออกเป็น 2 ช่วง ได้แก่ เมื่อฮาร์ดแวร์ส่งค่าจากอุปกรณ์มา และ ช่วงที่แอปพลิเคชันส่งข้อมูลจากอุปกรณ์ไปหาฮาร์ดแวร์



รูปที่ 4.56 แผนผังช่วงการส่งค่าจากอุปกรณ์ฮาร์ดแวร์

ในช่วงการทำงานนี้จะประกอบไปด้วย 3 ขั้นตอนสำหรับฮาร์ดแวร์ไปยังแอปพลิเคชันบนมือถือ และ 4 ขั้นตอนสำหรับแอปพลิเคชันบนเว็บ เมื่ออุปกรณ์เริ่มส่งค่าไปที่โบรกเกอร์ MQTT บน IoT Core ก็จะมี rule ที่กำหนดไว้สำหรับหัวข้อการส่งค่า และ rule นั้นก็จะสั่งใช้งานฟังก์ชันบน AWS Lambda ซึ่งจะบันทึกข้อมูลดังกล่าวลงไปยังฐานข้อมูลตารางตามชื่อฟาร์มที่กำหนดไว้ และส่งข้อมูลนั้นไปยังหัวข้อที่แอปพลิเคชันบนมือถือ ในขณะที่เว็บแอปพลิเคชันจะไม่มีารรับ-ส่งค่าด้วยโปรโตคอล MQTT นี้ แต่จะเป็นการเรียกใช้ฟังก์ชันจาก AWS Lambda เพื่อดึงค่าข้อมูลล่าสุดในตารางนั้นๆมาแทน



รูปที่ 4.57 แผนผังช่วงการส่งค่าจากแอปพลิเคชันไปยังอุปกรณ์ฮาร์ดแวร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในการส่งค่าไปยังอุปกรณ์ฮาร์ดแวร์จะมี 2 กรณีด้วยกัน คือ จากแอปพลิเคชันบนมือถือ และจากเว็บแอปพลิเคชัน ในกรณีของแอปพลิเคชันบนมือถือนั้นจะทำการส่งผ่านโปรโตคอล MQTT ด้วยหัวข้อของอุปกรณ์นั้นๆ และ rule ที่ตั้งไว้ใน IoT Core ก็จะส่งให้ฟังก์ชันหนึ่งบน AWS Lambda ทำงานเพื่อไปจัดการการตรวจสอบข้อมูลและส่งต่อไปในหัวข้อที่อุปกรณ์ติดตามอยู่ ในขณะที่บนเว็บแอปพลิเคชันนั้นจะเป็นการเรียกใช้ฟังก์ชันใน AWS Lambda โดยตรงเมื่อฟังก์ชันทำงานเสร็จสิ้นก็จะเป็นการส่งข้อมูลไปยังอุปกรณ์เช่นเดียวกัน

ในส่วนของ url ที่ใช้เป็นลิงค์สำหรับการเรียก API มาใช้งานและฟังก์ชันที่ถูก rule ของ IoT Core เรียกใช้นั้น จะมีทั้งหมด 4 ฟังก์ชันด้วยกัน ได้แก่

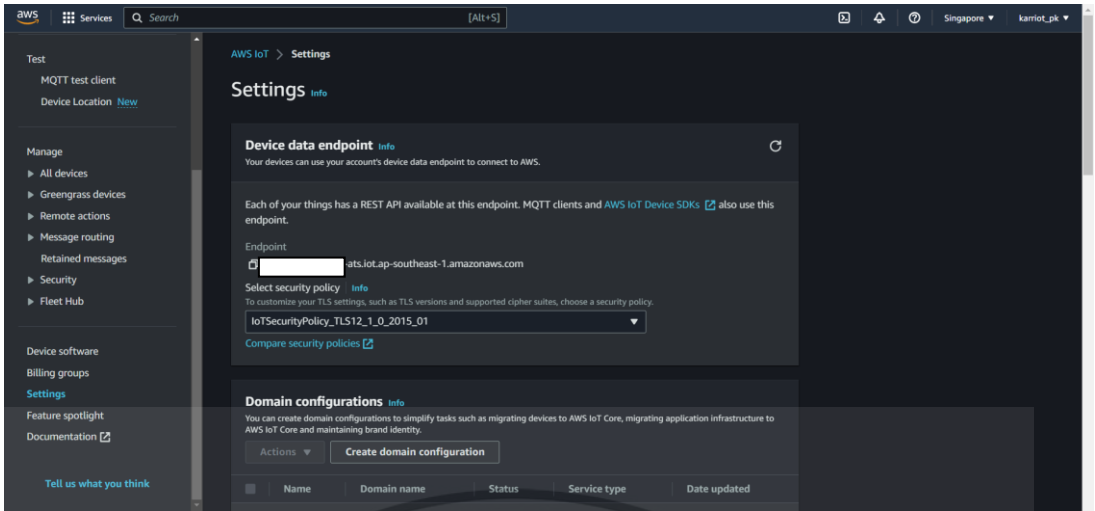
- “/device/register” ใช้สำหรับเว็บแอปพลิเคชันและอุปกรณ์เพื่อลงทะเบียนข้อมูลอุปกรณ์นั้นๆลงไปยังตารางอุปกรณ์
- “/liveData” ใช้สำหรับการดึงข้อมูลในตารางที่มีการอัปเดตค่าจากอุปกรณ์ตามเวลาจริง โดยฟังก์ชันนี้จะผูกไว้กับ rule ของ IoT Core ในหัวข้อ MQTT ที่ชื่อ “+/+/data/request” เมื่อเครื่องหมาย + คือฟาร์มและอุปกรณ์ที่ต้องการตามลำดับ และนอกจากนั้นก็สามารถเรียกใช้ผ่าน API ได้เช่นกัน
- “/farm/device/controlValue” ใช้สำหรับการควบคุมอุปกรณ์ที่ต้องการโดยการส่งค่าจากแอปพลิเคชัน
- “/farm/write” ใช้เฉพาะอุปกรณ์เท่านั้นในการส่งและเพิ่มหรืออัปเดตข้อมูลในตารางที่เป็นชื่อฟาร์มเฉพาะ

การดำเนินการในแต่ละฟังก์ชันที่เกี่ยวข้องกับ AWS IoT Core

ก่อนจะสร้างฟังก์ชันให้เพิ่มส่วนนี้ลงไปในไฟล์ handler.js

```
var iotdata = new AWS.IotData({ endpoint: 'iot endpoint', region: "ap-southeast-1" });
```

โดยแทนที่ “iot endpoint” ด้วยโดเมนจาก IoT Core ในคอนโซล AWS ให้ไปที่ Settings และหัวข้อ Device data endpoint จะปรากฏ Endpoint ที่สามารถนำไปใช้ได้



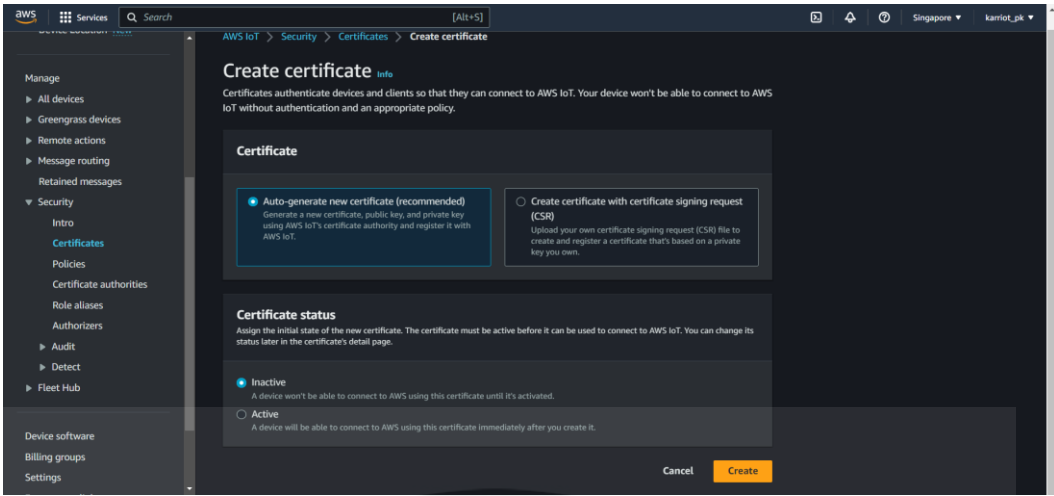
รูปที่ 4.58 Settings ต่างๆ ของบริการ AWS IoT Core

ในแอปพลิเคชันบนมือถือ จะมีการใช้ MQTT ด้วย ดังนั้นจึงต้องมี certificate ของบริการนี้ ให้ไปที่ Security ในหน้า IoT Core เลือกหัวข้อ Certificates และกดที่ Add certificate



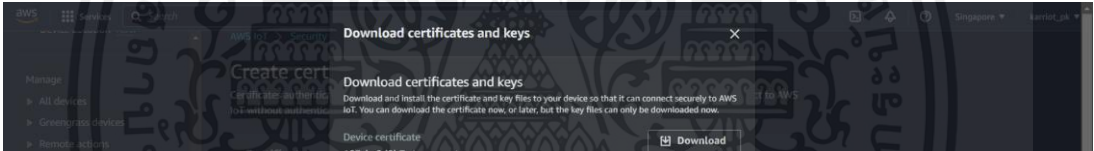
รูปที่ 4.59 หน้า Certificates ของบริการ AWS IoT Core

กดสร้าง certificate ที่ปุ่ม Create โดยเลือก Certificate เป็น Auto-generate และเลือกสถานะเป็น Inactive โดยสถานะนี้สามารถมาปรับให้ Active ได้ในภายหลัง



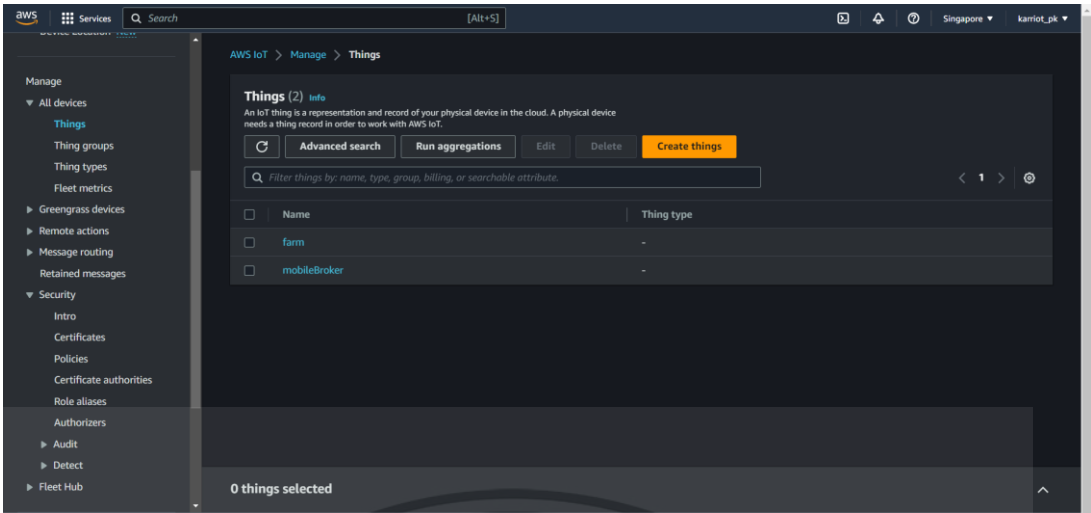
รูปที่ 4.60 การสร้าง Certificate ของบริการ AWS IoT Core

เมื่อกดสร้าง certificate แล้วจะปรากฏหน้าต่างดาวน์โหลด certificate และ keys ต่างๆขึ้นมาดังนี้ ให้กดดาวน์โหลดให้หมดในทุกหัวข้อ โดยเฉพาะหัวข้อ key เพราะจะสามารถดาวน์โหลดได้เพียงครั้งเดียวเท่านั้น ถ้าหากไม่ได้เก็บไว้ อาจจะต้องลบ certificate นี้ และดำเนินการสร้างใหม่

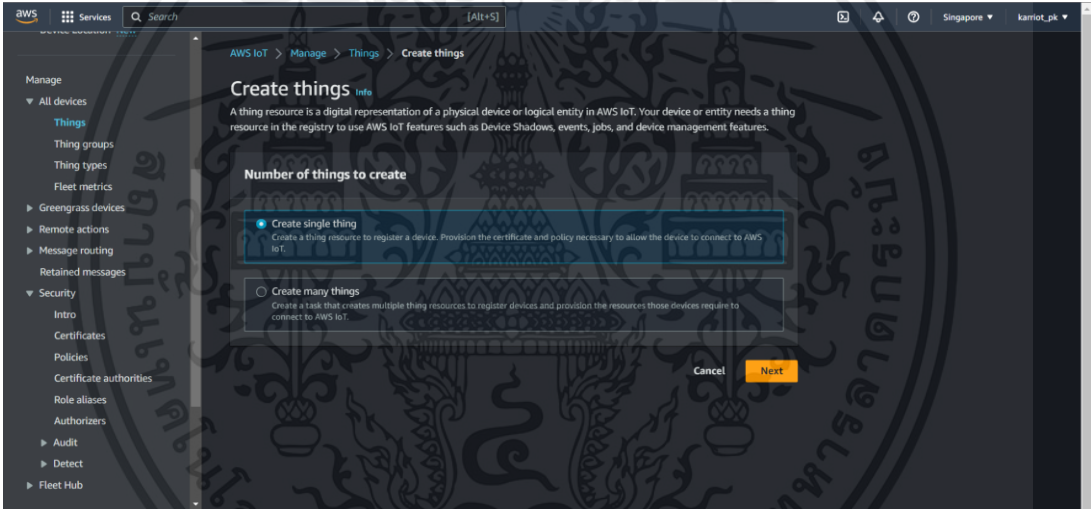


รูปที่ 4.61 การดาวน์โหลด Certificate ของบริการ AWS IoT Core

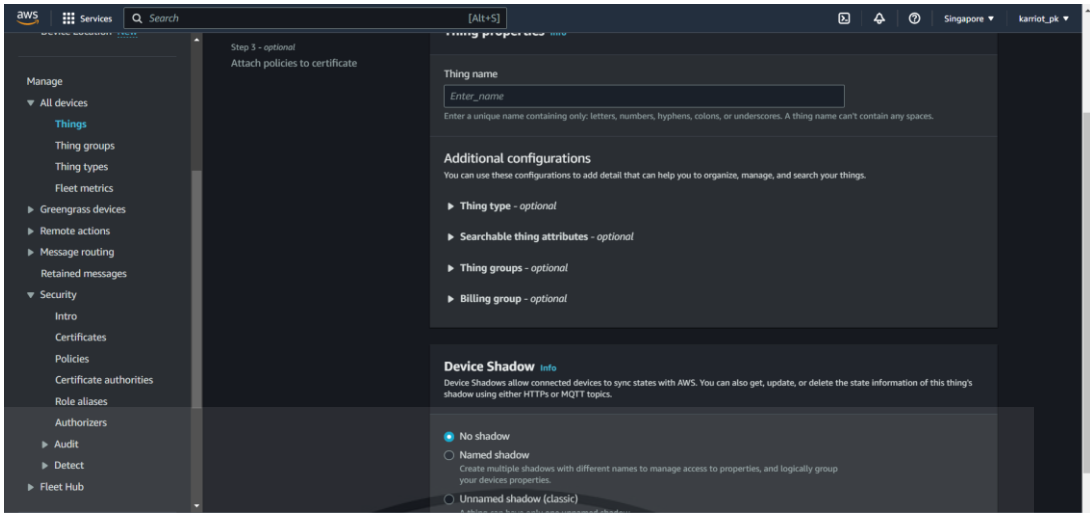
จากนั้นให้ไปที่หน้า Things เพื่อสร้าง Thing มารองรับทั้งฮาร์ดแวร์และแอปพลิเคชัน ให้เลือกสร้าง Single Thing แล้วกด Next ใส่ชื่อและเลือก No Shadow จากนั้นในหน้าที่ถามถึง device certificate ให้เลือก Skip และกดปุ่ม Create thing



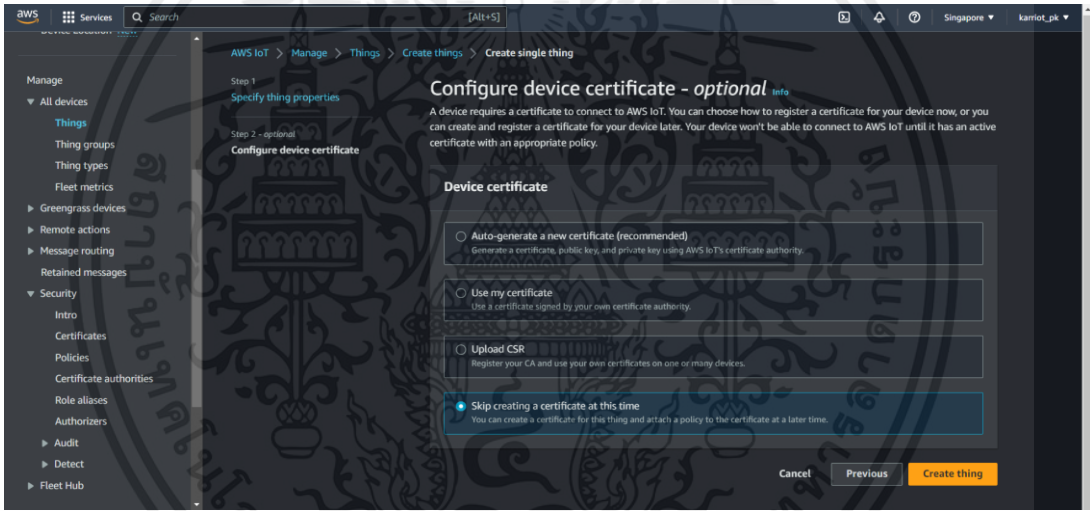
รูปที่ 4.62 หน้า Things ของบริการ AWS IoT Core



รูปที่ 4.63 การสร้าง Thing ของบริการ AWS IoT Core

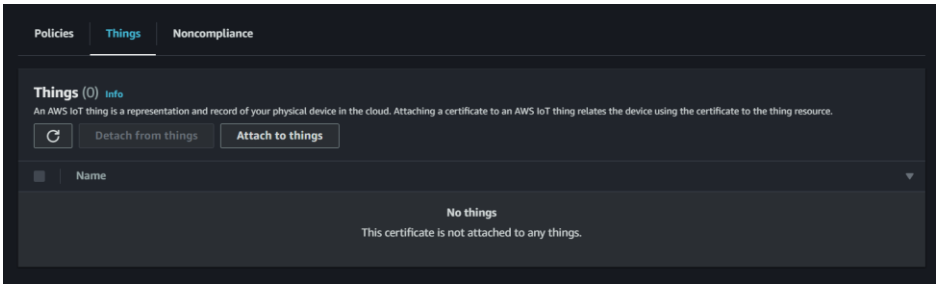


รูปที่ 4.64 การสร้าง Thing ขั้นตอนที่ 1 ของบริการ AWS IoT Core



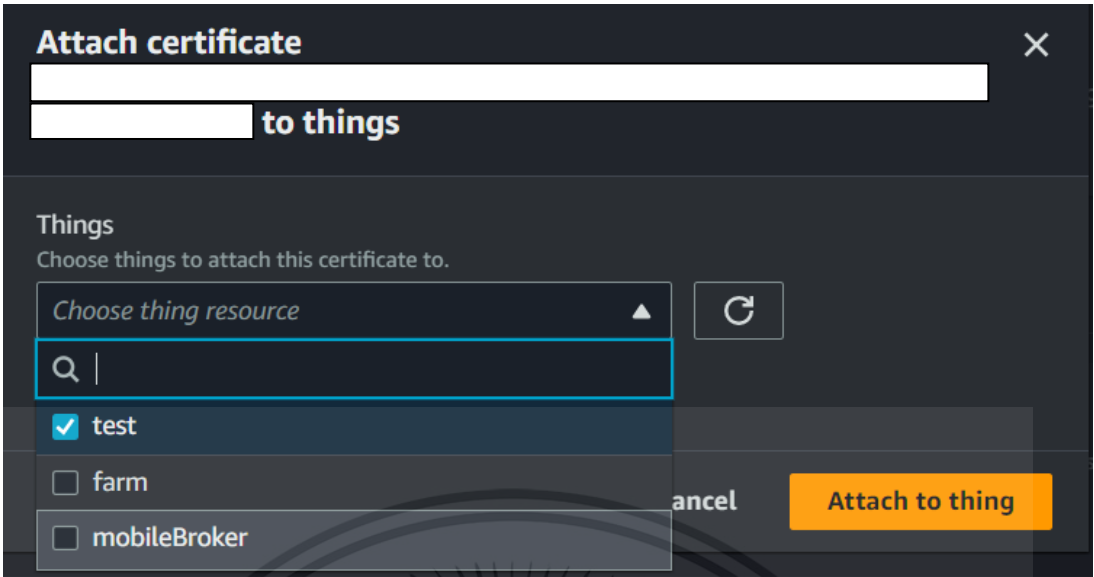
รูปที่ 4.65 การสร้าง Thing ขั้นตอนที่ 2 ของบริการ AWS IoT Core

จากนั้นให้กลับไปหน้า certificate และเลือก certificate ที่สร้าง ไปที่แถบ Things และกด Attach to things และเลือก Thing ที่สร้างไว้

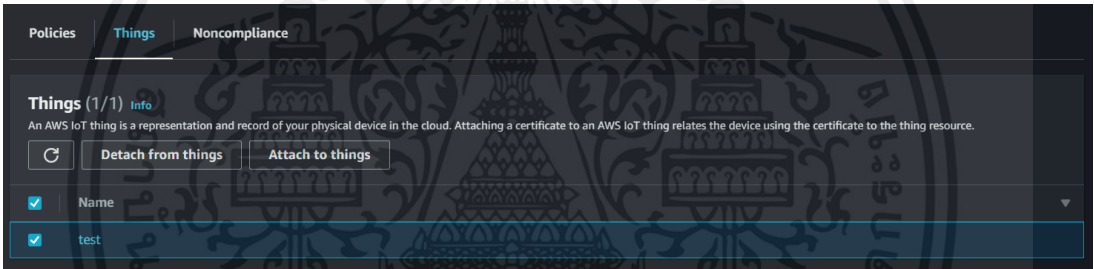


รูปที่ 4.66 หน้า Certificate แถบ Things ของบริการ AWS IoT Core

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.67 การเพิ่ม Thing ไปที่ Certificate

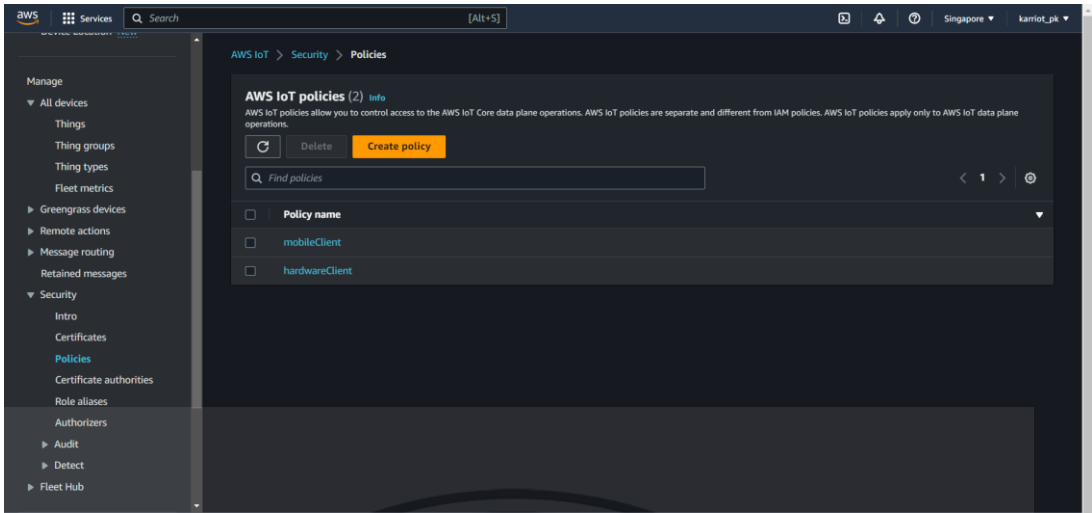


รูปที่ 4.68 หน้า Certificate แถบ Things หลังเพิ่ม Thing

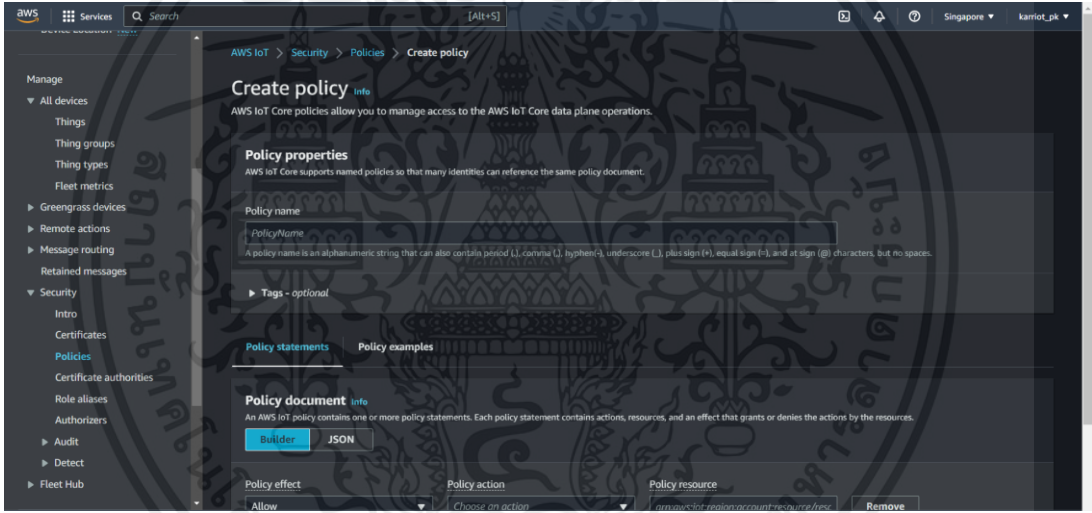
จากนั้นไปที่หน้า Policies เลือกสร้าง Policy ที่ปุ่ม Create policy กำหนดชื่อและเงื่อนไขมาให้เลือกหัวข้อดังนี้

- iot:Connect
- iot:Publish
- iot:Receive
- iot:Subscribe

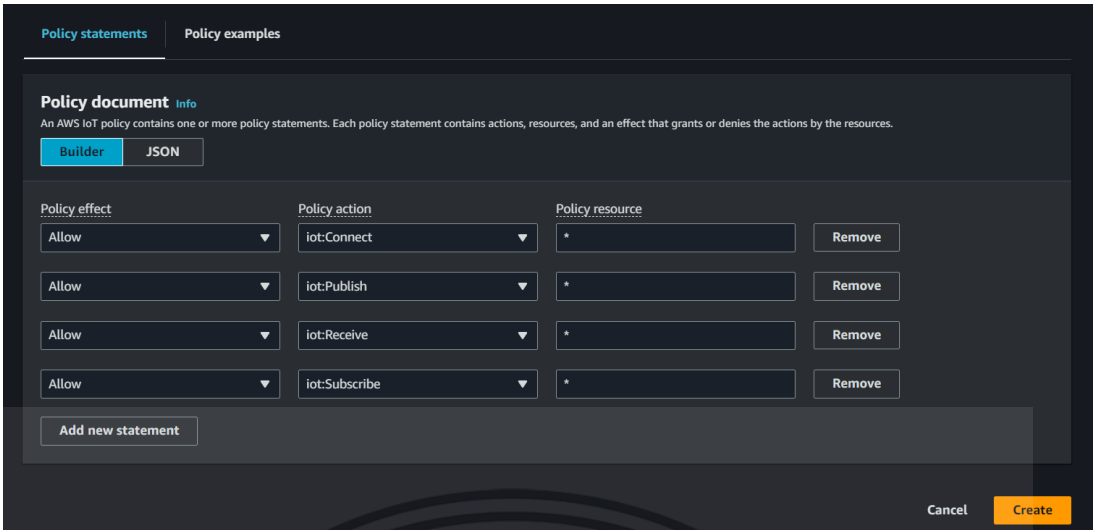
โดยที่ทุกอันนั้นเป็น allow และ Policy Resource เป็น * หมายถึงสามารถเข้าถึงทรัพยากรอื่นๆได้แบบไม่จำกัดว่าเป็นบริการอะไร เมื่อได้ดังนี้แล้ว ให้กด Create



รูปที่ 4.69 หน้า Policy ของบริการ AWS IoT Core

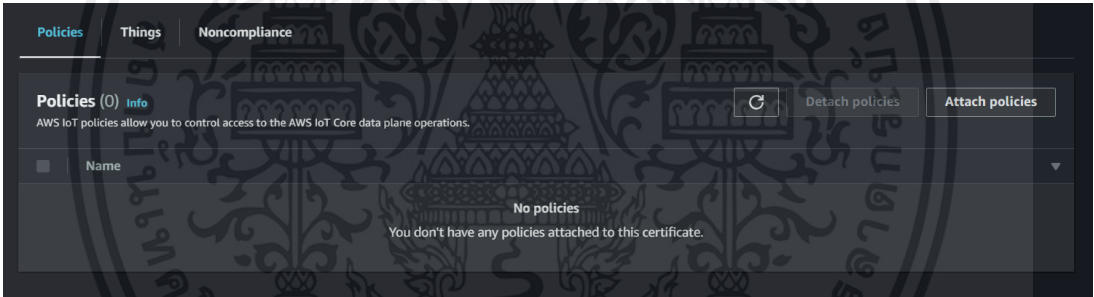


รูปที่ 4.70 การสร้าง Policy ของบริการ AWS IoT Core

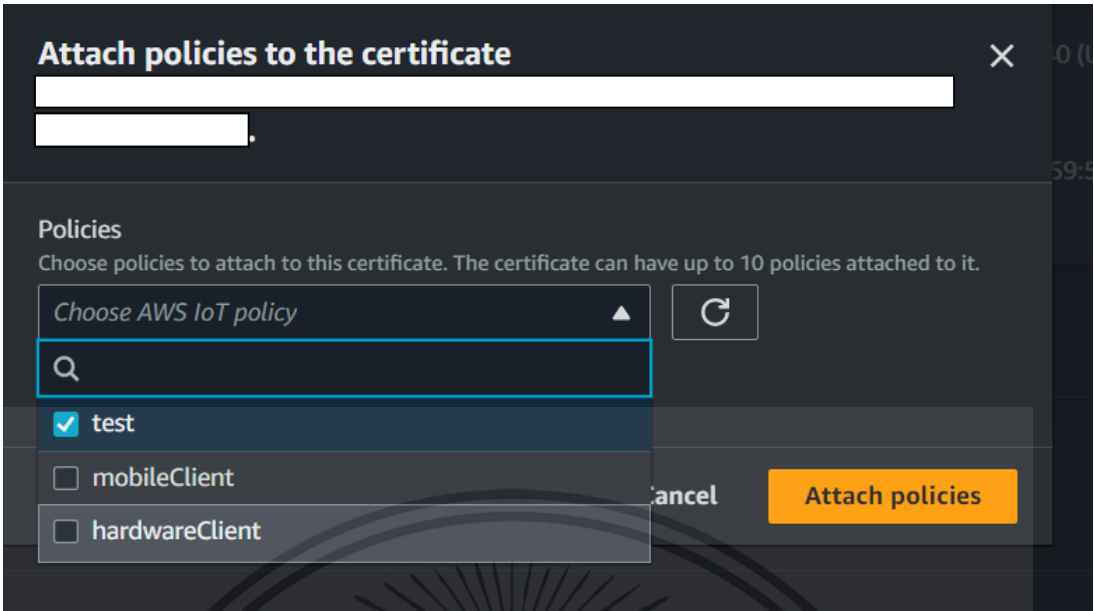


รูปที่ 4.71 การกำหนด Policy ในขั้นตอนการสร้าง

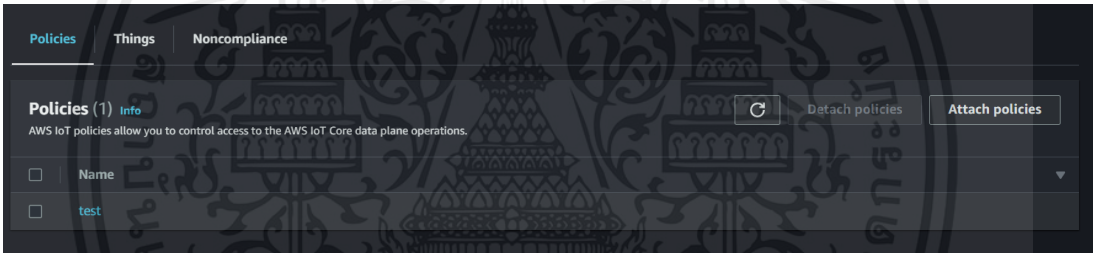
เมื่อสร้าง policy เสร็จแล้ว ให้กลับไปหน้าจอ certificate และเลือก certificate ที่สร้างไว้ ไปที่แถบ Policies เลือก Attach policies และเลือก policy ที่สร้าง



รูปที่ 4.72 หน้า Certificate แถบ Policies ของบริการ AWS IoT Core



รูปที่ 4.73 การเพิ่ม Policy ไปยัง Certificate



รูปที่ 4.74 หน้า Certificate แล่น Policies หลังเพิ่ม Policy

1. การลงทะเบียนอุปกรณ์ ในส่วนนี้จะกล่าวถึง “/device/register” ซึ่งใช้ในการลงทะเบียนอุปกรณ์ลงไปพื้นฐานข้อมูล โดยสามารถเรียกใช้ผ่านเว็บแอปพลิเคชันและตัวอุปกรณ์เอง ในการสร้างฟังก์ชันนี้ จะใช้ในลักษณะเดียวกันกับการสร้างข้อมูลในตาราง เริ่มจากสร้างโครงสร้างที่จะนำไปใส่ตามค่าในตารางอุปกรณ์ โดยสร้างเป็นตัวแปรคงที่ที่จะรับค่าในลักษณะของฟังก์ชันและส่งกลับมาเป็นวัตถุ json

```
const deviceInfo = (DeviceSerial, DeviceName, DeviceType, FarmName) => {  
  const timestamp = new Date().getTime();  
  return {  
    ID: gen_id, // encode to generate id  
    SerialNumber: DeviceSerial,  
    DeviceName: DeviceName,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    Type: DeviceType,
    Location: FarmName,
    CreateAt: timestamp
  };
};

```

เมื่อได้โครงสร้างแล้ว ก็ใช้ put เพื่อนำโครงสร้างเหล่านี้เข้าไปในฐานข้อมูลได้ ดังนี้

```

dynamoDb.put({
  TableName: // FarmDeviceTable
  Item: // deviceInfo(DeviceSerial, DeviceName, DeviceType, FarmName),
}).promise().then(res => {
  callback(null, {
    statusCode: 200,
    headers,
    body: JSON.stringify({
      message: "Successfully created " + // res.FarmName, res.FarmUser,
      id: res.id
    }),
  })
}).catch(err => {
  callback(null, {
    statusCode: 500,
    headers,
    body: JSON.stringify({
      message: "Unable to create"
    })
  })
});

```

โดยวิธี put เหมือนการสร้างข้อมูลในตาราง ให้ TableName แทนด้วยชื่อตารางอุปกรณ์จาก process.env และ Item ให้นำโครงสร้างไปใส่ที่ Item จากนั้นเมื่อใส่ข้อมูลสำเร็จ ให้ส่ง json ที่มีข้อความกับชื่อที่นำเข้าไปในตารางกลับไปให้ผู้เรียกใช้ แต่ถ้าหากทำไม่สำเร็จ จะไม่ส่งค่าข้อผิดพลาดกลับไป แต่จะใช่เป็นการส่งโค้ดสถานะ 500 กลับไปแทนซึ่งหมายถึงการกระทำในเซิร์ฟเวอร์ผิดพลาด และสามารถนำไปสร้างเป็นโมดูลฟังก์ชันได้ดังนี้

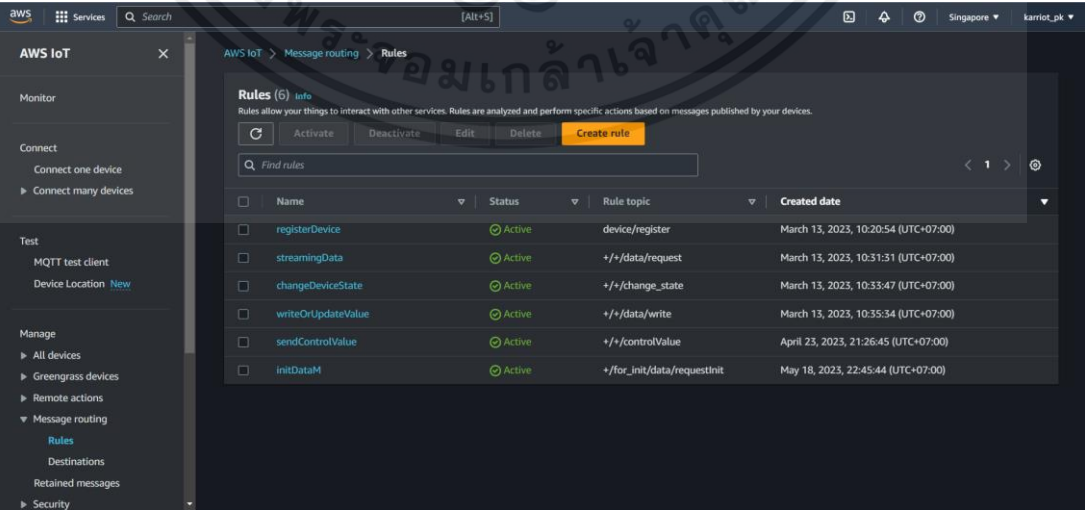
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
module.exports.registerDevice = async (event, context, callback) => {  
  // insert check data type ...  
  // generate structure to put  
  const deviceInfo = () => {};  
  dynamoDb.put({...}).promise().then().catch();  
};
```

และไปสร้างที่อยู่ API ในไฟล์ serverless.yml หัวข้อ functions ดังนี้

```
registerDevice:  
  handler: handler.registerDevice  
  description: Use for first time register of new device.  
  events:  
    - http:  
      path: /device/register  
      method: POST  
      cors: true
```

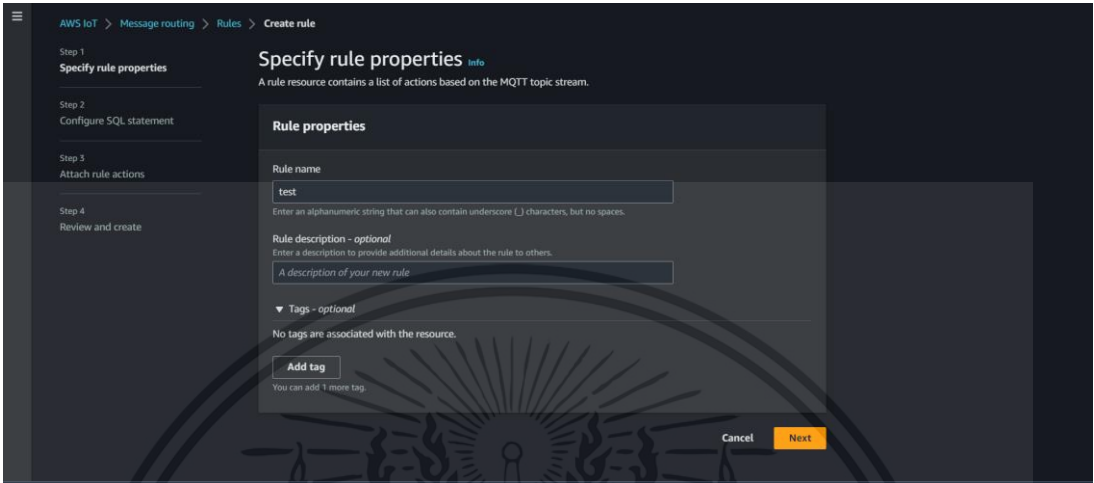
เมื่อได้ดังนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิงค์ endpoint ของฟังก์ชันขึ้นมา แต่จุดที่ต่างออกไปจากการสร้างข้อมูลในตาราง นั่นก็คือ ต้องทำให้ฟังก์ชันนี้ทำงานได้จากอุปกรณ์ด้วย ดังนั้น ให้ไปที่คอนโซล AWS และไปที่บริการ AWS IoT Core เมื่อไปที่หน้าบริการแล้วให้ไปที่ Message Routing และเลือก Rules



รูปที่ 4.75 หน้า Rule ของบริการ AWS IoT Core

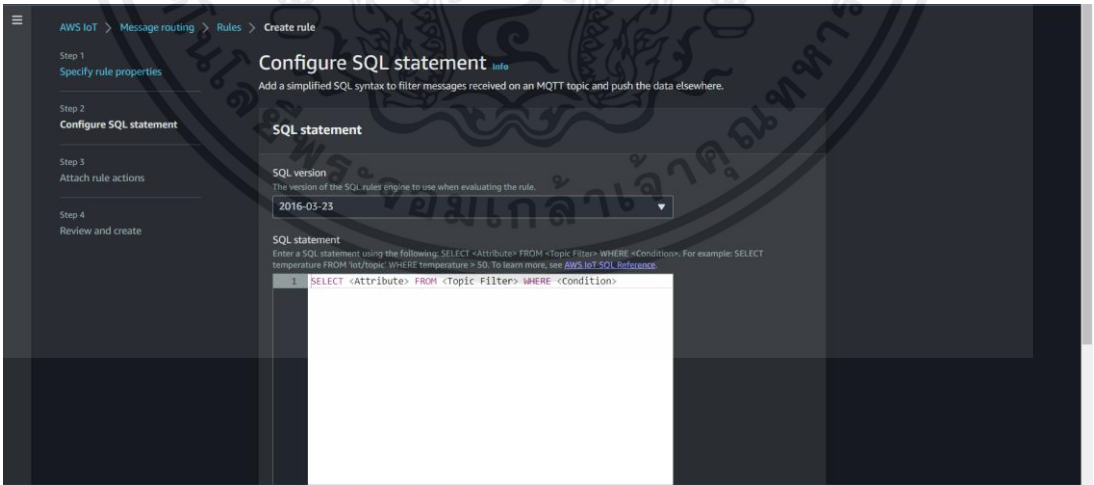
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในหน้านี้จะเป็นการกำหนดกฎต่างๆที่จะเกิดขึ้นเมื่อมีข้อมูลเข้ามาในหัวข้อ MQTT ที่กำหนด ซึ่งสามารถตั้งค่าได้ว่า จะให้ไปส่งให้บริการอื่นๆทำงานด้วยหรือไม่ ให้กดที่ Create Rule เพื่อสร้างกฎ



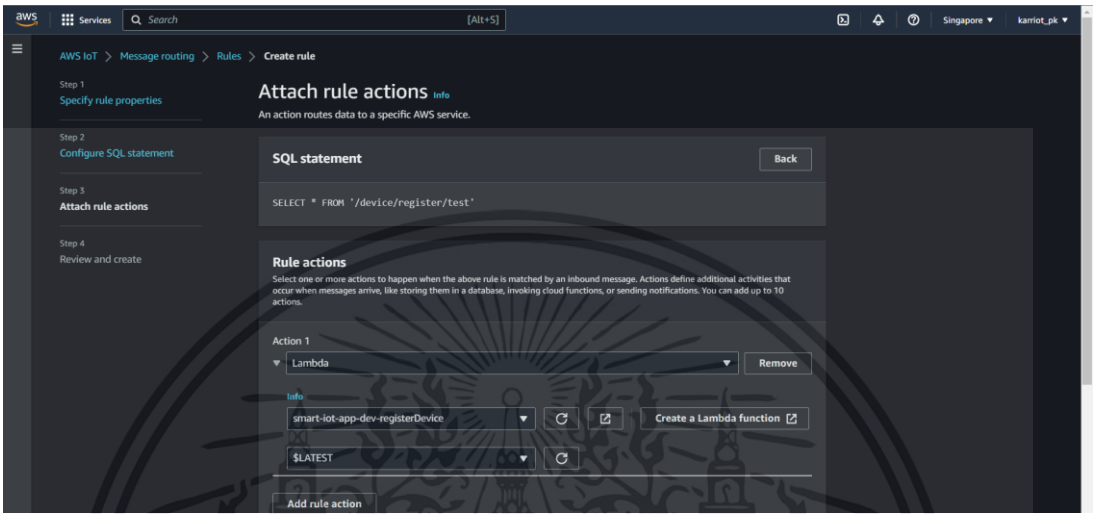
รูปที่ 4.76 หน้าการสร้าง Rule ของบริการ AWS IoT Core

เมื่อกด Next จะเข้าสู่หน้าการใส่ SQL ในหน้านี้จะเป็นการกรองข้อมูลที่ได้มาจากหัวข้อ MQTT ในกรณีนี้เป็นการลงทะเบียนอุปกรณ์ลงไปในตารางซึ่งหัวข้อ MQTT คือ “/device/register” ให้ใส่ SQL ไปดังนี้ `SELECT * FROM 'device/register'`

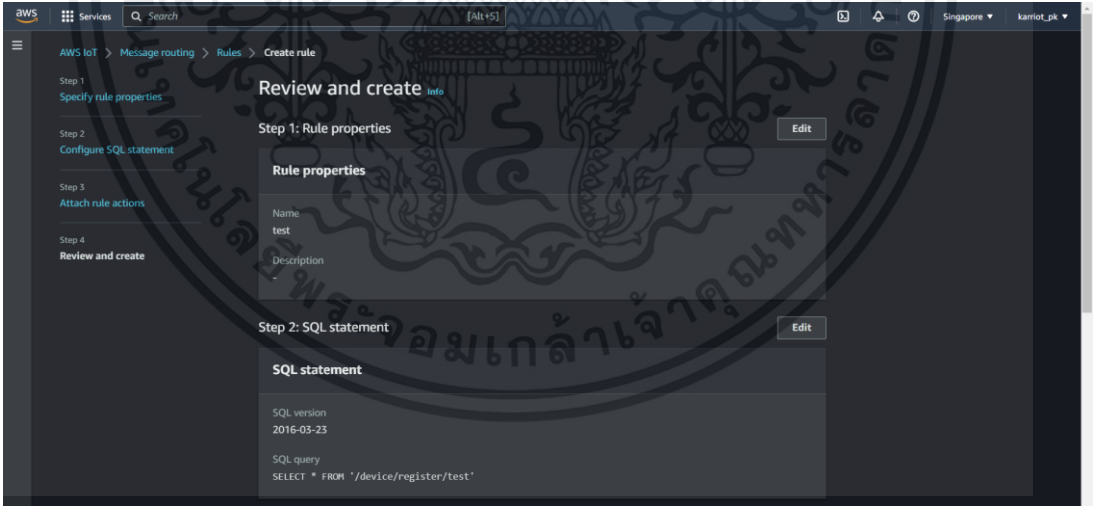


รูปที่ 4.77 หน้าการสร้าง SQL ของ Rule ของบริการ AWS IoT Core

ในหน้าถัดไปจะเป็นเชื่อมต่อไปยังบริการอื่นๆของ AWS ให้เลือกตรง Rule Actions ที่ Action 1 เป็น Lambda หลังจากนั้นจะมีรายชื่อของฟังก์ชันขึ้นมาให้เลือก เลือก “registerDevice” ที่ deploy ไป เมื่อเสร็จสิ้นให้กดหน้าถัดไป เพื่อตรวจสอบการสร้างกฎนี้ และกด Create ที่ด้านล่างสุดของหน้า Review เพื่อสร้าง



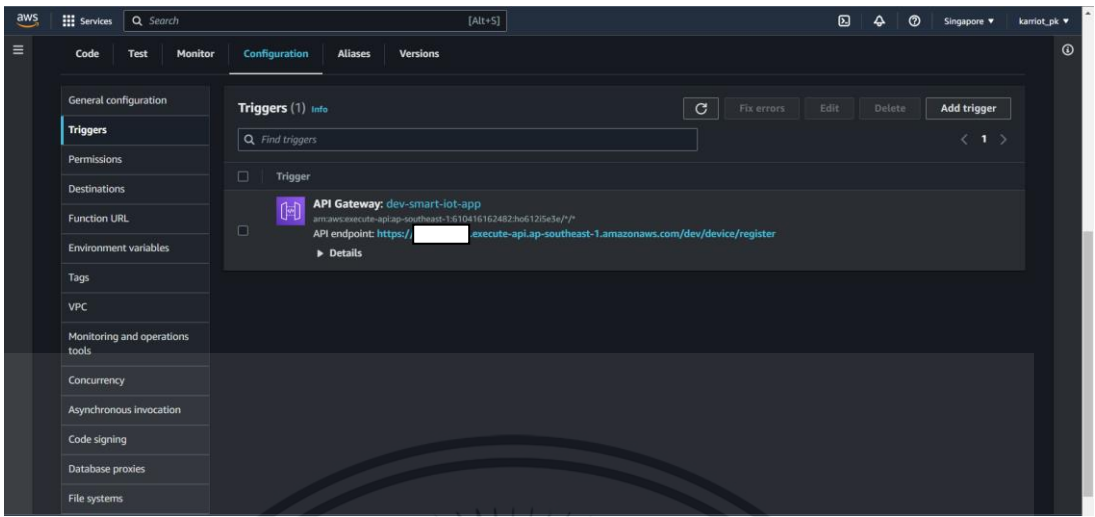
รูปที่ 4.78 หน้าการเชื่อมไปบริการอื่นๆของ Rule ของบริการ AWS IoT Core



รูปที่ 4.79 หน้าบทวนการสร้าง Rule ของบริการ AWS IoT Core

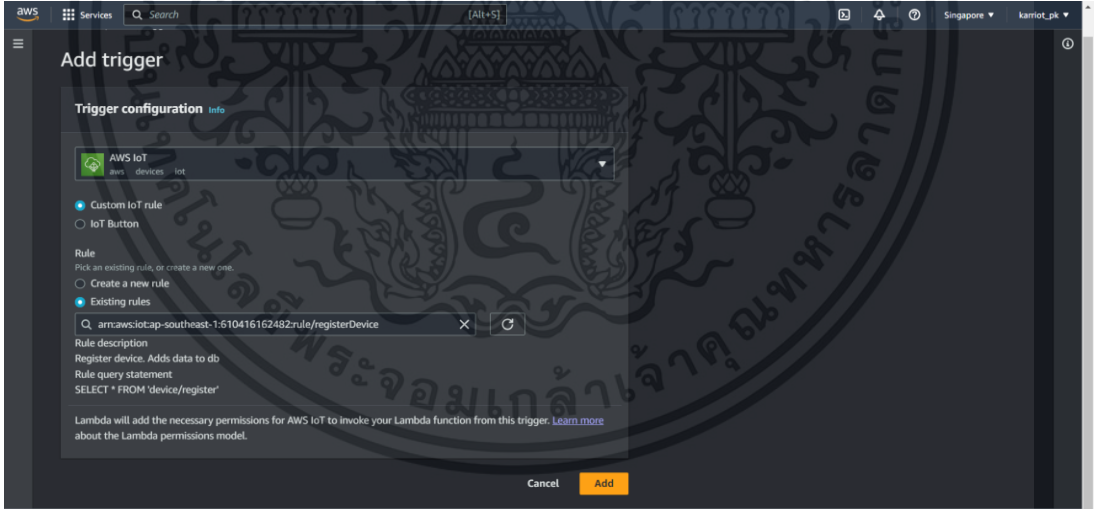
ในกรณีนี้จะไม่ทำให้การส่งข้อมูลจากอุปกรณ์มาเข้าสู่กฎนี้ได้ เพราะว่าส่วนของฟังก์ชันยังไม่ได้ตั้งค่าการรับส่งใช้งานจากบริการ IoT Core นี้ ให้ไปที่หน้าบริการ AWS Lambda และเลือกฟังก์ชันที่ต้องการ จากนั้นเลือกแถบ Configuration และเลือก Trigger จะปรากฏในลักษณะนี้ และให้กดที่ Add Trigger เพื่อทำการเพิ่มบริการเข้าไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



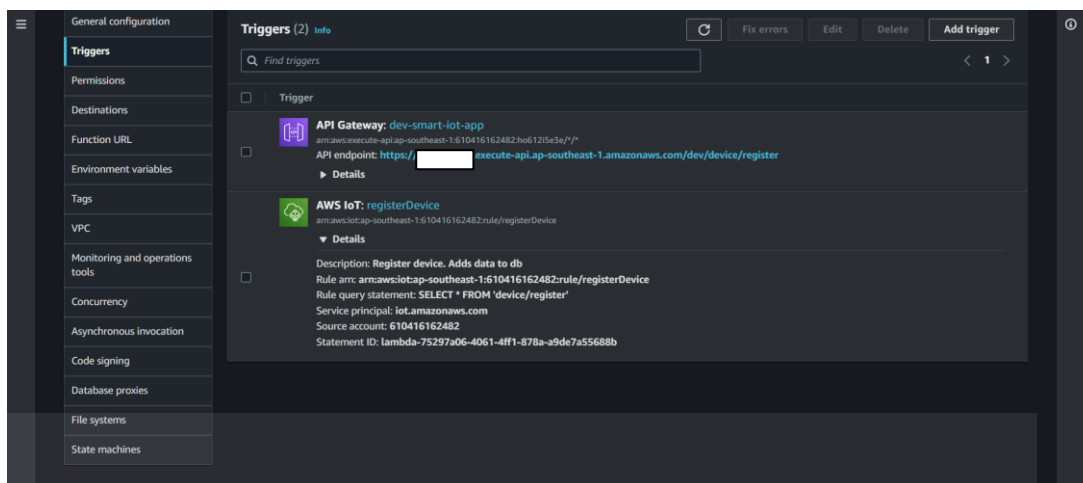
รูปที่ 4.80 หน้า AWS Lambda แถบ Configuration หัวข้อ Triggers

ในหน้า Trigger ให้กดเลือกบริการ AWS IoT จากนั้นเลือก Custom IoT Rule แล้วเลือก Existing Rule และเลือกกฎที่ได้สร้างไว้ ซึ่งเมื่อเลือกเสร็จ จะปรากฏเป็นคำอธิบายของกฎและ SQL ให้ทำการตรวจสอบความถูกต้องแล้วจึงกด Add



รูปที่ 4.81 แสดงการเพิ่ม Trigger ไปยังฟังก์ชัน

เมื่อเพิ่มสำเร็จจะปรากฏ Trigger อันใหม่ในแถบ Configuration หัวข้อ Trigger ดังนี้



รูปที่ 4.82 หน้า AWS Lambda แถบ Configuration หลังเพิ่ม Trigger ใหม่

2. การดึงข้อมูลของอุปกรณ์ในฟาร์มใดๆ ในส่วนนี้จะกล่าวถึง “/liveData” ซึ่งใช้ในการดึงข้อมูลมาแสดงผล การทำงานจะแบ่งออกเป็น 2 กรณี คือ ในแอปพลิเคชันบนมือถือจะเป็นการนำมาใช้เพื่อทำข้อมูลเริ่มต้นของหน้าแสดงผลในแดชบอร์ด ในขณะที่บนเว็บแอปพลิเคชันจะใช้เพื่อแสดงผลล่าสุดเพียงอย่างเดียวเท่านั้น ด้วยการใช้วิธี get จากตารางชื่อฟาร์มใดๆ สามารถทำได้ดังนี้

```
const data = await dynamoDb.get({
  TableName: request.topic.split("/")[0],
  Key: {
    "DeviceID": request.topic.split("/")[1],
  },
  ProjectionExpression: "DeviceValue",
}).promise();
```

ถ้ากำหนดให้ rule ใน IoT Core ส่งหวัข้อมูลด้วยก็จะทำให้สามารถประหยัดพื้นที่ในการส่งข้อมูลได้ด้วยการไม่ต้องแยกส่งชื่อฟาร์มและชื่ออุปกรณ์มา แต่เป็นการนำชื่อหัวข้อแยกออกจากกันก็จะสามารถได้ชื่อฟาร์มพร้อมชื่ออุปกรณ์ได้เช่นเดียวกัน เมื่อได้ดังนี้ ใช้ get ที่ใส่ค่าชื่อตาราง กฎหลักและค่าที่ต้องการไป และเก็บไว้เป็นตัวแปรคงที่ ในกรณีที่ข้อมูลมีจำนวนมากเกินไป ก็อาจจะส่งผลต่อการประมวลผลของข้อมูลที่ค้างไว้อย่างจำกัด ทำให้ต้องใช้การลดทอนจำนวนลง เพื่อให้สามารถส่งข้อมูลได้เร็วขึ้นด้วยวิธีการดังนี้

```
var itemToReturn = { "LatestItems": [] };
if (data.Item["DeviceValue"].length >= 10) {
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    itemToReturn.LatestItems = data.Item["DeviceValue"].slice(data.Item["DeviceValue"].length - 10,
data.Item["DeviceValue"].length);
  } else {
    itemToReturn.LatestItems = data.Item["DeviceValue"];
  }
}

```

ยกตัวอย่าง ให้ตัดทอนข้อมูลเหลือแค่ 10 ชุดข้อมูล สามารถใช้วิธี slice โดยต้องการชุดข้อมูลล่าสุด ในฐานข้อมูลนี้ ชุดข้อมูลจากอุปกรณ์จะถูกเก็บไว้ในลักษณะของ array ทำให้ข้อมูลล่าสุดก็คือ ข้อมูลที่อยู่ท้ายสุด ดังนั้นระยะที่ต้องการก็คือ length - 10 ไปจนถึง length และนำข้อมูลที่ถูกลดทอนมาใส่ในโครงสร้างเพื่อเตรียมส่งค่ากลับ

ในการส่งค่ากลับ จะมีอยู่ 2 กรณีด้วยกัน คือ ส่งกลับแบบใช้ MQTT หรือ ไม่ใช้ ซึ่งการใช้ MQTT นั้นจะใช้กับแอปพลิเคชันบนมือถือเท่านั้น ในขณะที่แบบเว็บแอปพลิเคชัน จะใช้ callback และส่งผลลัพธ์กลับพร้อมโค้ดสถานะ โดยในแต่ละวิธีสามารถทำได้ ดังนี้ เริ่มจากแบบไม่ใช้ MQTT จะได้ว่า

```

if (noMqtt) {
  const noMqttResponse = {
    statusCode: 200,
    headers,
    body: JSON.stringify(itemToReturn.LatestItems),
  };
  callback(null, noMqttResponse);
}

```

ในโค้ดนี้จะมีการตรวจสอบตัวแปร noMqtt ซึ่งเว็บแอปพลิเคชันจำเป็นต้องส่งมาใน json เพื่อให้ส่งกลับแบบไม่ใช้ MQTT ซึ่งจะใช้ callback แบบฟังก์ชันอื่นๆ ในขณะที่ใช้ MQTT จะทำได้ ดังนี้

```

var paramsResponse = {
  topic: farmName + '/' + targetDevice + '/' + 'data/live',
  payload: JSON.stringify(itemToReturn.LatestItems),
  qos: 1
};
const pub = iotdata.publish(paramsResponse);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
pub.on('success', () => console.log("success")).on('error', () => console.log("error"));
return new Promise(() => pub.send(function (err, data) {
  if (err) {
    console.log(err);
  } else {
    console.log(data);
  }
}));
```

เริ่มต้นด้วยที่จะต้องกำหนดค่าที่จำเป็นต่อการส่งกลับไปได้แก่ หัวข้อ, ผลลัพธ์ข้อมูล และ ความสำคัญของข้อมูล โดยหัวข้อจะใช้ชื่อฟาร์มและชื่ออุปกรณ์ที่แยกออกมาในขั้นตอนแรกที่ใช้ get ต่อด้วย “/data/live” ผลลัพธ์ของข้อมูลที่ส่งกลับมาก็คือ ชุดข้อมูลที่ถูกลดทอนขนาด และ ความสำคัญของข้อมูล (Quality of Service) ซึ่งให้เป็น 1 หมายถึงการที่ข้อมูลนั้นจะต้องส่งอย่างน้อย 1 ครั้ง เพื่อเป็นการรับประกันได้ว่าข้อมูลจะไปถึงแอปพลิเคชัน ต่อมาจะเป็นการใช้วิธี publish [23] โดยนำค่าที่กำหนดไว้มาใส่ และกำหนด callback ของแต่ละกรณี ได้แก่ success และ error ด้วยการใช้ on ในที่นี้จะใช้เพื่อเป็นการตรวจสอบการทำงานบน CloudWatch และจบการทำงานของฟังก์ชันด้วยการใช้ send เพื่อเริ่มการส่งข้อมูล และสามารถนำไปสร้างโมดูลฟังก์ชันได้ ดังนี้

```
module.exports.liveDataHandler = (event, context, callback) => {
  // insert data type check ...
  const data = ...;
  var itemToReturn = ...;
  if(noMqtt) ...
  var paramsResponse =
  ...
  return Promise(...);
};
```

และไปสร้างที่อยู่ API ในไฟล์ serverless.yaml หัวข้อ functions ดังนี้

liveDataHandler:

handler: handler.liveDataHandler

description: handle a request from topic and send back to application

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

events:

- http:
 - path: /liveData
 - method: POST
 - cors: true

เมื่อได้ดังนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิงค์ endpoint ของฟังก์ชันขึ้นมา เช่นเดียวกันกับการลงทะเบียนอุปกรณ์ ต้องตั้งค่าเพิ่มเติม ดังนี้

- ที่ IoT Core สร้าง rule โดยมีรายละเอียด ได้แก่
 - SQL `SELECT *, topic() as topic FROM '+/+/data/request'`
 - เชื่อมต่อไปยังฟังก์ชัน liveData
- ที่ Lambda ฟังก์ชัน liveData แถบ Configuration เพิ่ม trigger เป็น IoT Core และเลือก Custom IoT Rule และเลือก rule ที่สร้างไว้

หัวข้อที่จะใช้สำหรับแต่ละองค์ประกอบจะมีลักษณะดังนี้ เมื่อ + คือชื่อฟาร์มและอุปกรณ์ ตามลำดับ

- แอปพลิเคชันบนมือถือ ส่งในหัวข้อ “+/+/data/request”
- เว็บแอปพลิเคชัน ส่งในหัวข้อ “liveData” และเพิ่ม RequestNoMqtt ในข้อมูลว่าเป็นระหว่าง “yes” หรือ “no” ที่ส่งมาด้วย

3. การควบคุมอุปกรณ์ ในส่วนนี้จะกล่าวถึง “/farm/device/controlValue” ซึ่งใช้ในการส่งค่าที่ต้องการไปควบคุมอุปกรณ์ผ่านแอปพลิเคชัน ในการเขียนฟังก์ชันนี้จะมีลักษณะคล้ายกันกับการดึงข้อมูลจากอุปกรณ์ใดๆ สามารถเขียนเป็นโมดูลฟังก์ชันได้ ดังนี้

```
module.exports.sendControlValue = (event, context, callback) => {
  // insert data type check ...
  var response = {
    topic: farmName + '/' + targetDevice + '/' + 'control/listen/request',
    payload: JSON.stringify({
      "value": request.controlValue,
    }),
    qos: 1
  };
};
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

const pub = iotdata.publish(response);

pub.on('success', () => ...).on('error', () => ...);

return new Promise(() => pub.send((err, data) {...}));

};

```

และไปสร้างที่อยู่ API ในไฟล์ serverless.yaml หัวข้อ functions ดังนี้

sendControlValue:

```

handler: handler.sendControlValue

description: Send a value to control the device

events:
  - http:
      path: /farm/device/controlValue
      method: POST
      cors: true

```

เมื่อได้ดังนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิ้งค์ endpoint ของฟังก์ชันขึ้นมา เช่นเดียวกันกับการลงทะเบียนอุปกรณ์ ต้องตั้งค่าเพิ่มเติม ดังนี้

- ที่ IoT Core สร้าง rule โดยมีรายละเอียด ได้แก่
 - SQL **SELECT *, topic() as topic FROM "+/+controlValue"**
 - เชื่อมต่อไปยังฟังก์ชัน sendControlValue
- ที่ Lambda ฟังก์ชัน sendControlValue แถบ Configuration เพิ่ม trigger เป็น IoT Core และเลือก Custom IoT Rule และเลือก rule ที่สร้างไว้

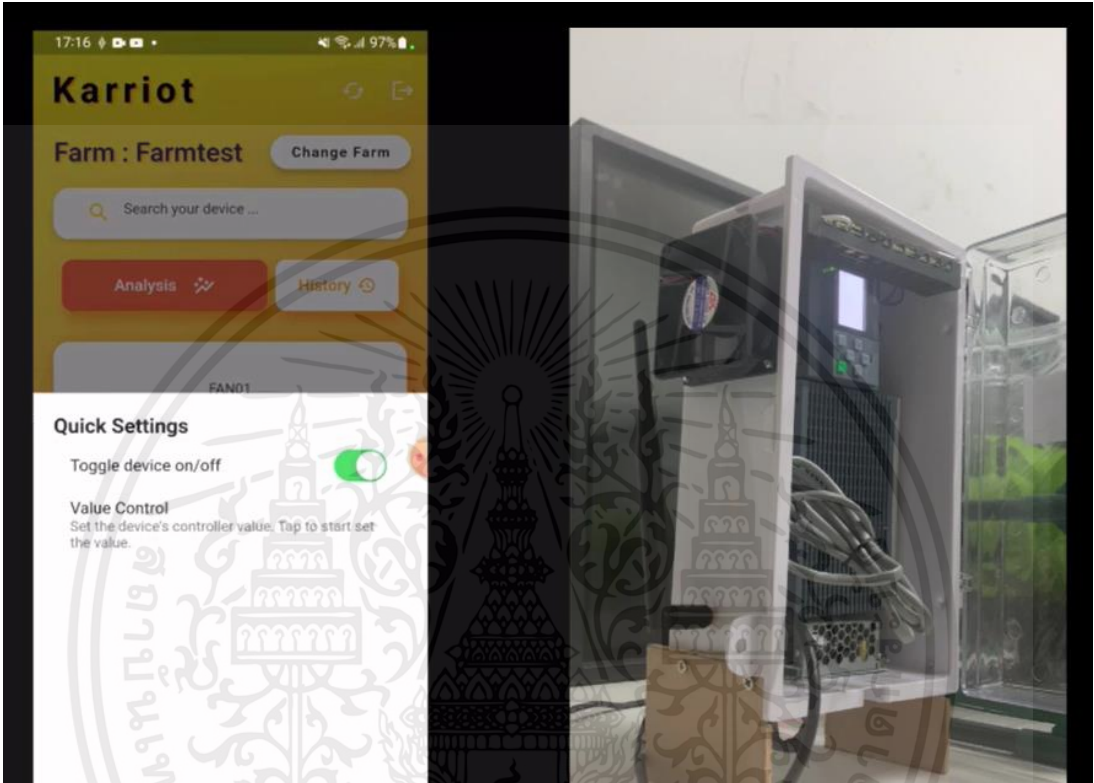
หัวข้อที่จะใช้สำหรับแต่ละองค์ประกอบจะมีลักษณะดังนี้ เมื่อ + คือชื่อฟาร์มและอุปกรณ์ ตามลำดับ

- ฮาร์ดแวร์ ติดตามหัวข้อ “+/+/control/listen/request”
- แอปพลิเคชันบนมือถือ ส่งในหัวข้อ “+/+/controlValue”
- เว็บแอปพลิเคชัน ส่งในหัวข้อ “farm/device/controlValue” และเพิ่ม topic ในข้อมูลที่ส่งมาด้วย

ทดสอบการเชื่อมต่อระหว่างแอปพลิเคชันบนมือถือไปยังอุปกรณ์ด้วยฟังก์ชันควบคุม

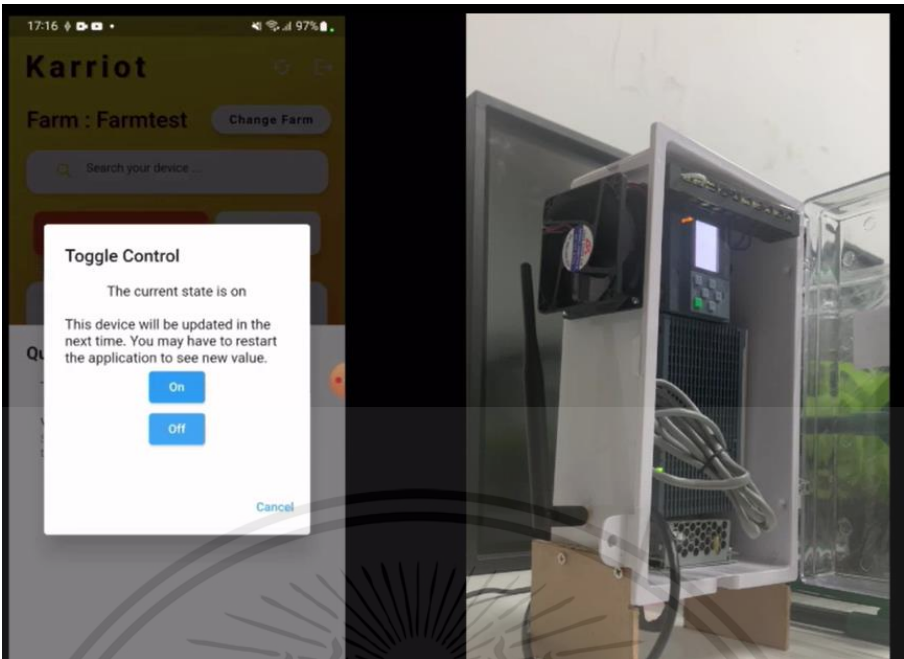
ในการทดสอบนี้จะเป็นการส่งค่าปิด-เปิดไปยังอุปกรณ์ก็คือ พัดลม ซึ่งจะมีการเปลี่ยนแปลงในตัวอุปกรณ์คือ เมื่อผู้ใช้กดเปิด พัดลมจะทำงานและหน้าจอจะเป็นสีขาว ในขณะที่เมื่อกดปิด หน้าจอจะเปลี่ยนเป็นสีแดง

สถานะปัจจุบันของพัดลมในรูปเป็นเปิด และผู้ใช้กดค้างที่ FAN01 เพื่อเปิดหน้าเมนูควบคุม



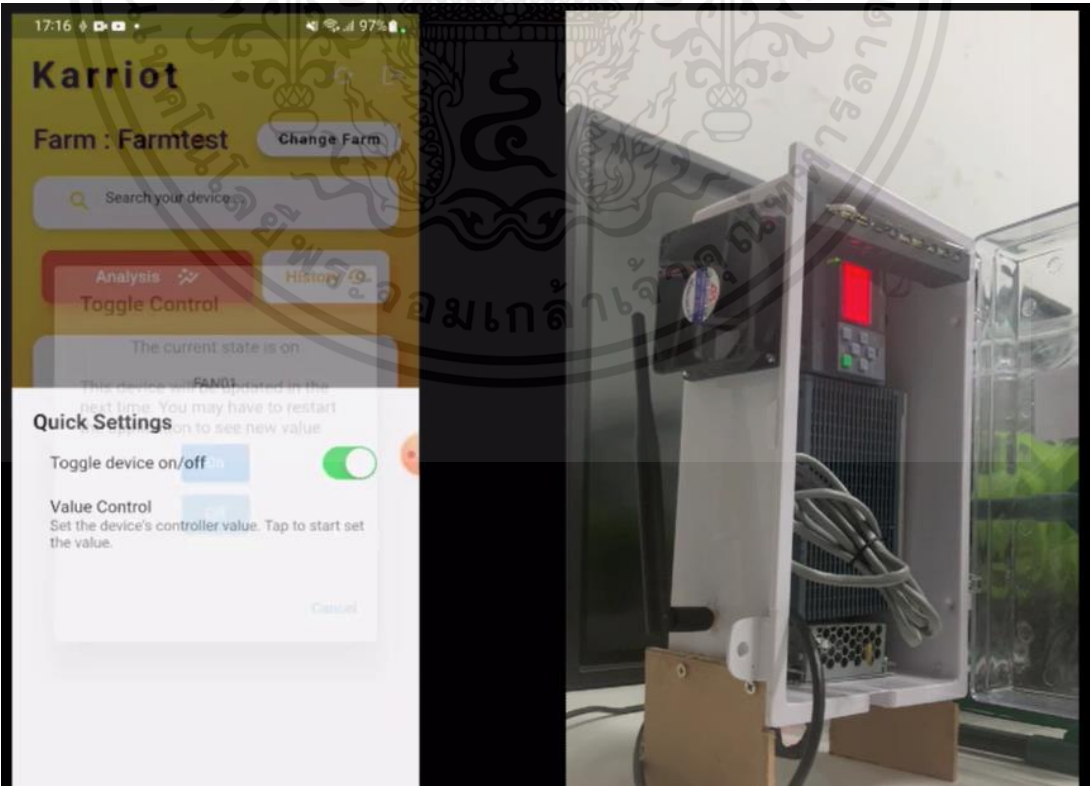
รูปที่ 4.83 เมนูการควบคุมในหน้าแดชบอร์ดกับอุปกรณ์ฮาร์ดแวร์ที่เกี่ยวข้อง

ผู้ใช้กดที่ Value Control เพื่อเปิดหน้าเมนูควบคุม ซึ่งจะปุ่มควบคุมได้ 2 ปุ่ม คือ เปิด และปิด การควบคุมนี้จะเป็นการส่งค่าไปยัง rule ที่ตั้งไว้ และไปเรียกใช้งานฟังก์ชัน Lambda



รูปที่ 4.84 การเปิดหน้าการส่งค่าในหน้าแดชบอร์ดกับอุปกรณ์ฮาร์ดแวร์ที่เกี่ยวข้อง

เมื่อผู้ใช้กดปุ่มปิดการทำงาน หน้าต่างการควบคุมก็จะเด้งออกไปยังหน้าเมนูควบคุมและส่งค่าสถานะ “ปิด” ไปยัง rule และไปเรียกใช้ Lambda ซึ่งผลลัพธ์คือการส่งค่าไปยังหัวข้อ “+/FAN01/control/listen/request” จากนั้นปิดพัดลมและเปลี่ยนหน้าจอเป็นสีแดง



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 4.85 หลังส่งค่าในหน้าแดชบอร์ดไปยังอุปกรณ์ฮาร์ดแวร์ที่เกี่ยวข้อง

4. การเพิ่มหรืออัปเดตข้อมูลจากอุปกรณ์ ในส่วนนี้จะกล่าวถึง “/farm/write” ซึ่งใช้ในการส่งค่าจากอุปกรณ์ไปยังตารางชื่อฟาร์มที่อุปกรณ์นั้นติดตั้งอยู่ โดยอุปกรณ์จะถูกตรวจสอบว่ามีข้อมูลอยู่ในตารางอุปกรณ์ด้วยหรือไม่ ถ้าไม่มี การส่งนั้นจะไม่ปรากฏในฐานข้อมูล และถ้าหากข้อมูลมีการเพิ่มเข้าไปในตารางแล้ว การส่งค่าก็จะผ่านไปยังขั้นตอนถัดไป คือ การตรวจสอบว่าการส่งค่านี้เพื่อไปบันทึกลงในตารางจะเป็นการเพิ่มหรือการอัปเดต แล้วจึงค่อยนำข้อมูลลงไปในตาราง เริ่มต้นการสร้างโครงสร้างที่จะใช้เก็บข้อมูลดังนี้

```
const deviceInfo = (DeviceSerial, DeviceState, FarmName) => {
  return {
    DeviceID: DeviceSerial,
    DeviceState: DeviceState,
    Location: FarmName,
    DeviceValue: [
      {
        "Value": event.payload.Data,
        "TimeStamp": timestamp,
        "State": DeviceState
      }
    ],
  };
};
```

จากนั้นนำโครงสร้างนี้ไปตรวจสอบกับตารางอุปกรณ์ ถ้าหากพบว่ามีไม่ปรากฏในตาราง ก็จะให้หยุดการทำงานของฟังก์ชันทันที

```
await dynamoDb.scan({ TableName: FarmDeviceTable }).promise().then(res => {
  if (res.Items.every((i) => i.DeviceName !== targetDevice)) {
    console.info([res.Items.every((i) => i.DeviceName !== targetDevice), targetDevice]);
    throw new Error("Id not exist in the table");
  }
});
```

จากนั้นก็ใช้ `get` เพื่อตรวจสอบข้อมูลในตารางชื่อฟาร์มว่ามีข้อมูลอยู่หรือไม่ ถ้าไม่มีก็จะตั้งค่าตัวแปรเป็น `true` ในทางตรงกันข้าม ถ้ามีข้อมูลอยู่แล้ว ก็จะเป็น `false`

```
var paramsDeviceInFarm = {
  TableName: farmName,
  Key: {
    DeviceID: targetDevice
  },
  ProjectionExpression: "DeviceID, DeviceValue"
};

await dynamoDb.get(paramsDeviceInFarm).promise().then(
  result => {
    console.info(result.Item);
    if (result.Item == undefined || result.Item.DeviceValue == undefined) {
      noData = true;
    }
  })
  .catch((err) => {
    console.error(err);
    noData = true;
  });
```

จากนั้นสร้างตัวแปรคงที่ ที่มีค่าเป็นฟังก์ชันที่รับค่าเป็นโครงสร้าง `deviceInfo` โดยในตัวฟังก์ชันจะมีการสร้างโครงสร้างสำหรับตารางด้วย ดังนี้

```
const deviceInfo = {
  TableName: farmName,
  Item: device
};

const deviceUpdateInfo = {
  TableName: farmName,
  Key: {
    DeviceID: targetDevice
  },
  UpdateExpression: "SET #attrName = list_append(#attrName, :attrValue)",
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

ExpressionAttributeNames: {
  "#attrName": "DeviceValue"
},
ExpressionAttributeValues: {
  ":attrValue": device["DeviceValue"]
},
ReturnValues: "ALL_NEW"
};

```

deviceInfo ในที่นี้ใช้เพื่อเพิ่มข้อมูลลงไปในการอัปเดตข้อมูลในตาราง โดยจุดสำคัญได้แก่ deviceUpdateInfo จะใช้เพื่อเป็นการอัปเดตข้อมูลในตาราง โดยจุดสำคัญได้แก่

- UpdateExpression ใช้เพื่อกำหนดว่าในการอัปเดตจะใช้เป็นวิธีใด ในที่นี้ใช้ SET ในการตั้งค่าให้ค่า DeviceValue ในตารางให้เป็นการเพิ่มข้อมูลลงไปเป็น array ด้วยการ ใช้ list_append ที่ต้องการ 2 ค่า ได้แก่ ชื่อของหลัก และค่าที่ต้องการตามลำดับ
- ExpressionAttributeNames คือการกำหนดชื่อของตัวแปรที่นำไปแทนใน UpdateExpression เป็นหลักของตาราง
- ExpressionAttributeValues คือการกำหนดชื่อของตัวแปรที่นำไปแทนใน UpdateExpression เป็นค่าที่ต้องการอัปเดตของหลักนั้นในตาราง
- ReturnValues คือการบอกว่าจะทำอย่างไรหลังอัปเดตเสร็จสิ้น ซึ่งในที่นี้ใช้ “ALL_NEW” หมายถึง ใช้ค่าใหม่ทั้งหมดแทนค่าเก่า

เมื่อได้โครงสร้าง 2 อันนี้แล้ว สร้างเงื่อนไขเพื่อตรวจสอบตัวแปรจากที่ตรวจสอบการมีอยู่ของข้อมูล จะได้ว่า

```

if (noData == true) {
  console.log("data not found");
  return dynamoDb.put(deviceInfo).promise().then(res => device);
}
else {
  console.log("update");
  return dynamoDb.update(deviceUpdateInfo).promise().then(res => device);
}

```

จากนั้นกำหนดค่าที่จะส่งกลับไปยังแอปพลิเคชันด้วยโปรโตคอล MQTT ในหัวข้อ “+/+/data/live” เมื่อ + คือชื่อฟาร์มและอุปกรณ์ตามลำดับ โดยจะส่งหัวข้อ ข้อมูลและความสำคัญของข้อมูล (QoS) ในลักษณะ ดังนี้

```
var paramsResponse = {
  topic: farmName + '/' + targetDevice + '/' + 'data/live',
  payload: JSON.stringify([
    {
      "Value": event.payload.Data,
      "TimeStamp": timestamp,
      "State": event.DeviceState
    }
  ]),
  qos: 1
};

const pub = iotdata.publish(paramsResponse);
pub.on('success', () => console.log("[Success] send data from IOT")).on('error', () =>
console.log("[Error] something went wrong..."));
```

ในข้อมูลที่ส่งกลับไปยังอยู่ลักษณะของ array ที่มี json เป็นองค์ประกอบ ใน json จะมี 3 ค่า คือ ค่าของอุปกรณ์ เวลา และสถานะอุปกรณ์ (ปิด-เปิด) จากนั้นก็นำค่าที่กำหนดเหล่านี้ไปเป็นค่าที่เตรียมจะส่งด้วย publish [23] พร้อมด้วยการเพิ่ม callback และจบการทำงานด้วยการส่งข้อมูล ดังนี้

```
await submitDevice(deviceInfo(targetDevice, event.DeviceState, event.DeviceLocation)).then(res
=> {
  callback(null, event)
}).then(() => {
  return new Promise(() => pub.send(function (err, data) {
    if (err) {
      console.log("[Error] ", err);
    } else {
      console.log(data);
    }
  })
});
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    });
  }).catch(err => {
    console.log(err);
    callback(null, {
      statusCode: 500,
      body: JSON.stringify({
        message: "Unable to write or publish device's data"
      })
    });
  });
});

```

ด้วยการนำตัวแปรคงที่มีค่าเป็นฟังก์ชันที่สร้างไว้ในการเพิ่มหรืออัปเดตข้อมูลตามการตรวจสอบการมีอยู่ของข้อมูลโดยใส่ค่าเข้าไปเป็นโครงสร้าง deviceInfo เมื่อตรวจสอบเสร็จก็ค่อยเริ่มส่งข้อมูลนี้ไปยังหัวข้อ “+/+/data/live” และแสดงผลบนแอปพลิเคชันบนมือถือ ดังในหัวข้อที่ 4.3.4 และนำมารวมกันเป็นโมดูลฟังก์ชัน ดังนี้

```

module.exports.writeDataFromIOT = async (event, context, callback) => {
  // insert data type check ...
  var noData = false;
  const deviceInfo = ...;
  var paramsDeviceInFarm = ...;
  await dynamoDb.scan(...).promise().then(...);
  await dynamoDb.get(paramsDeviceInFarm).promise().then(...);
  const submitDevice = device => {
    const deviceInfo = ...;
    const deviceUpdateInfo = ...;
    if(noData){...} else {...}
  };
  var paramsResponse = ...;
  const pub = ...;
  pub.on(...).on(...);
  await submitDevice(...).then(...).catch(...);
};

```

และไปสร้างที่อยู่ API ในไฟล์ serverless.yml หัวข้อ functions ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

writeDataFromIoT:

handler: handler.writeDataFromIoT

description: Handle write and update of value from iot device in the farm table

events:

- http:
 - path: /farm/write
 - method: POST

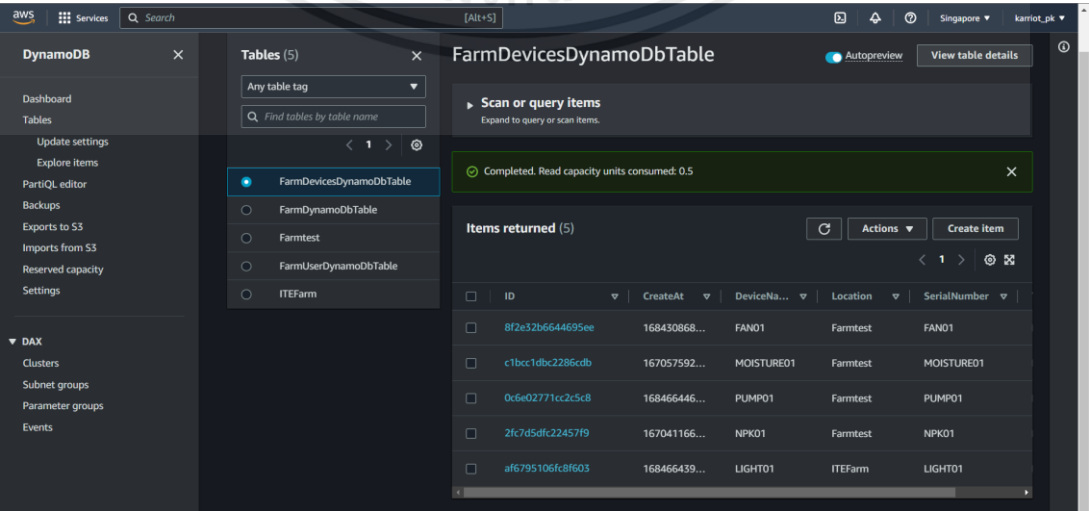
เมื่อได้ดังนี้แล้วให้ทำการ deploy ขึ้นไปยัง AWS ด้วยคำสั่ง serverless deploy เมื่อ deploy เสร็จแล้วจะได้ลิงค์ endpoint ของฟังก์ชันขึ้นมา เช่นเดียวกันกับการลงทะเบียนอุปกรณ์ ต้องตั้งค่าเพิ่มเติม ดังนี้

- ที่ IoT Core สร้าง rule โดยมีรายละเอียด ได้แก่
 - SQL `SELECT *,topic() as topic FROM '+/+/data/write'`
 - เชื่อมต่อไปยังฟังก์ชัน writeDataFromIoT
- ที่ Lambda ฟังก์ชัน writeDataFromIoT แถบ Configuration เพิ่ม trigger เป็น IoT Core และเลือก Custom IoT Rule และเลือก rule ที่สร้างไว้

หัวข้อที่จะใช้สำหรับแต่ละองค์ประกอบจะมีลักษณะดังนี้ เมื่อ + คือชื่อฟาร์มและอุปกรณ์ ตามลำดับ

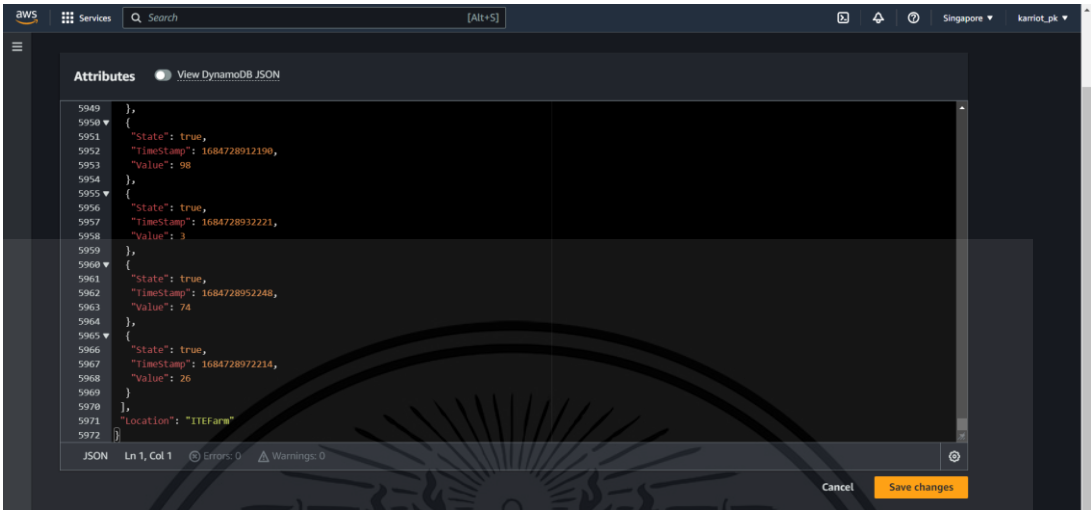
- ฮาร์ดแวร์ ส่งในหัวข้อ “+/+/data/write”
- แอปพลิเคชันบนมือถือ ติดตามในหัวข้อ “+/+/data/live”

และได้ผลจากการเชื่อมต่อกับอุปกรณ์ ดังนี้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 4.88 หน้าข้อมูลอุปกรณ์ LIGHT01 ใน DynamoDB ตาราง Farmtest



รูปที่ 4.89 หน้าข้อมูลอุปกรณ์ LIGHT01 ใน DynamoDB ตาราง ITEFarm

จาก 2 รูป จะเห็นได้ว่า รูปแรกคือข้อมูลเดิมในฟาร์ม Farmtest ในขณะที่ข้อมูลปัจจุบันอยู่ที่ ITEFarm มีจำนวนข้อมูลแตกต่างกันอย่างเห็นได้ชัด เพราะว่าเงื่อนไขที่เขียนไว้ในฟังก์ชันนั่นเอง ที่มีการตรวจสอบชื่ออุปกรณ์ในฟาร์มด้วยการใช้วิธี scan [23] ทำให้ข้อมูลของ LIGHT01 ที่ถูกย้ายไปนั้นไม่ไปอัปเดตที่ข้อมูลฟาร์มเก่า และการแสดงผลของค่าที่ส่งมาสามารถดูได้ทั้งหน้าแดชบอร์ดหรือหน้าประวัติ ในแอปพลิเคชันบนมือถือ ดังในหัวข้อ 4.3.4 และ 4.3.7

บทที่ 5

สรุปผล อภิปรายผล และข้อเสนอแนะ

การพัฒนาแอปพลิเคชันเพื่อจัดการระบบฟาร์มอัจฉริยะ มีวัตถุประสงค์เพื่อพัฒนาแอปพลิเคชันในระบบฟาร์มอัจฉริยะที่สามารถปรับตัวได้ตามความต้องการของผู้ใช้ โดยมีการพัฒนาทั้งในส่วน of แอปพลิเคชันบนมือถือ และฟังก์ชันบริการบนเซิร์ฟเวอร์ ซึ่งในแอปพลิเคชันบนมือถือจะมีหน้าที่หลักคือการดูข้อมูลของอุปกรณ์รวมถึงประวัติของค่านั่น ควบคุมการทำงานของอุปกรณ์ ควบคุมให้เปิดหรือปิดการทำงาน การ ในขณะที่ฟังก์ชันบนเซิร์ฟเวอร์จะมีหน้าที่จัดการการทำงาน ให้บริการต่างๆ เชื่อมต่อกัน ได้แก่ การยืนยันตัวตน ฐานข้อมูล และ MQTT ซึ่งผู้จัดทำได้สรุปการดำเนินการ อภิปรายผล และข้อเสนอแนะ ดังต่อไปนี้

5.1 สรุปการดำเนินการ

1. การพัฒนาแอปพลิเคชันบนระบบปฏิบัติการแอนดรอยด์ สำหรับการจัดการระบบฟาร์มอัจฉริยะโดยใช้เครื่องมือชุดพัฒนา Flutter ในการพัฒนามีการเชื่อมต่อไปยังระบบอื่น ได้แก่ เซิร์ฟเวอร์ และฮาร์ดแวร์ โดยในแอปพลิเคชันมีความสามารถในการทำงาน ดังนี้
 - 1.1. การรับ-ส่งค่าด้วยโปรโตคอล MQTT
 - 1.2. การแสดงผลค่าอุปกรณ์แบบตามเวลาจริงในรูปแบบต่างๆ
 - 1.3. การเก็บประวัติค่าต่างๆ
 - 1.4. การยืนยันตัวตน
 - 1.5. การแจ้งเตือนผู้ใช้
 - 1.6. การสร้างรายงาน
 - 1.7. การค้นหาอุปกรณ์ในหน้าแดชบอร์ด
2. การพัฒนาฟังก์ชันบนเซิร์ฟเวอร์ AWS ด้วย AWS Lambda ที่ใช้โครงสร้างการทำงานแบบ Serverless เพื่อใช้ในการทำระบบอัตโนมัติที่รองรับการทำงานจากฮาร์ดแวร์และไปเชื่อมต่อกับบริการอื่นๆ โดยมีฟังก์ชันการทำงานหลักดังต่อไปนี้
 - 2.1. การบันทึกค่าจากอุปกรณ์ลงฐานข้อมูลที่ AWS DynamoDB
 - 2.2. การรับค่าและส่งต่อไปที่ฮาร์ดแวร์สำหรับการควบคุมอุปกรณ์
 - 2.3. การจัดการข้อมูลในฐานข้อมูล เช่น การดึงข้อมูลฟาร์ม, การสร้างตารางฟาร์มสำหรับการเก็บค่าตามเวลา เป็นต้น
 - 2.4. การจัดการข้อมูลในขณะที่กำลังมีการส่งค่าจากฮาร์ดแวร์เพื่อใช้ในการแสดงผลบนแอปพลิเคชัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.2 อภิปรายผล

จากการศึกษาแอปพลิเคชันที่ใช้ในระบบฟาร์มอัจฉริยะ พบว่า มีหลากหลายปัจจัยที่ส่งผลทำให้เกิดข้อจำกัดภายในแอปพลิเคชัน เช่น การเพิ่มจำนวนอุปกรณ์ที่ทำได้จำกัดและทำได้แค่เพียง 1 ฟาร์มอันเนื่องมาจากมาจากพื้นที่บนเซิร์ฟเวอร์หรือคลาวด์มีจำนวนจำกัดส่งผลให้การเก็บข้อมูลทำได้น้อยลงและจะต้องใช้จ่ายเพื่อเพิ่มพื้นที่ให้มากขึ้นด้วย หรืออาจติดปัญหาในเรื่องของการได้รับค่าของอุปกรณ์ล่าช้า ดังนั้นจึงได้ออกแบบและสร้างแอปพลิเคชันและระบบบนคลาวด์ขึ้นมาเพื่อรองรับการปรับขนาดของข้อมูลต่างๆ ด้วยการใช้ชุดเครื่องมือ Flutter ในการพัฒนาแอปพลิเคชันบนมือถือ และใช้ Serverless ในการพัฒนาฟังก์ชัน หลังการสร้างระบบดังกล่าวพบว่า อุปกรณ์สามารถส่งค่าและแสดงผลในแอปพลิเคชันได้ตามเวลาจริง และในกรณีที่มีการเพิ่มอุปกรณ์ของผู้ใช้ในฟาร์มก็จะปรากฏในแอปพลิเคชันทันที โดยที่ผู้ใช้ไม่ต้องไปเพิ่มอุปกรณ์ด้วยตนเอง และเมื่อดูค่าใช้จ่ายของคลาวด์ว่า การทำงานด้วยการใช้บริการต่างๆ นี้จะต้องมีค่าใช้จ่ายเท่าไร พบว่าเมื่อเปรียบเทียบกับราคาของบริการจากแอปพลิเคชันอื่นๆ แล้วนั้นมีราคาถูกกว่ามาก เพราะบริการใน AWS มีลำดับขั้นของการใช้งานที่ถูกแบ่งออกมา ถ้ายังไม่เกินจำนวนที่กำหนดก็จะยังไม่คิดราคาเพราะยังถือว่าอยู่ในเกณฑ์ของระดับฟรี-tier สมมติในกรณีที่คิดราคาแล้วสำหรับ 1 อุปกรณ์ที่ส่งข้อมูลทุกๆ 5 นาที จะมีค่าใช้จ่ายอยู่ที่ประมาณ 120 บาทต่อเดือน ทั้งนี้ราคานี้เป็นเพียงการคาดการณ์ของ 1 อุปกรณ์เท่านั้นและไม่ได้รวมการขยายระดับขนาดของข้อมูลซึ่งคาดว่าจะขึ้นไปถึงประมาณ 3,000 บาทหรือมากกว่าได้

5.3 ข้อเสนอแนะ

1. ทำแอปพลิเคชันให้สามารถรองรับการประเภของอุปกรณ์ให้หลากหลายมากขึ้น เพื่อรองรับการแสดงผลในหน้าแดชบอร์ดให้หลากหลาย
2. รองรับการแสดงผลเป็นภาษาไทยในแอปพลิเคชัน เพื่อให้ผู้ใช้เข้าใจง่ายมากขึ้น
3. เพิ่มตัวชี้วัดในหน้าวิเคราะห์ให้มากขึ้น เพื่อให้สามารถวิเคราะห์กราฟได้หลากหลายมากขึ้น
4. เพิ่มข้อมูลเพิ่มเติมในหน้ารายงานให้มากขึ้น
5. ทำแอปพลิเคชันให้รองรับระบบปฏิบัติการอื่นบนมือถือ เช่น ios
6. ทำการตั้งค่าให้เป็นระบบอัตโนมัติเมื่อถึงเกณฑ์ที่ผู้ใช้กำหนด ทำให้สามารถควบคุมอุปกรณ์เองได้โดยไม่ต้องผ่านผู้ใช้
7. พัฒนาการวิเคราะห์ให้เป็นการทำ machine learning เพื่อทำให้วิเคราะห์และคาดเดาในเรื่องต่างๆ เช่น ผลผลิต หรือ ค่าเกณฑ์ที่ผู้ใช้ควรจะต้องตั้ง

บรรณานุกรม

- [1] Aykut Bal. 2023. **Native Mobile App Architecture – Everything You Need to Know** [Online]. Available: <https://www.storyly.io/post/native-mobile-app-architecture-everything-you-need-to-know>
- [2] **Flutter Documentation** [Online]. 2023. Available: <https://docs.flutter.dev/>
- [3] **MQTT Essentials** [Online]. 2023. Available: <https://www.hivemq.com/mqtt-essentials/>
- [4] **HTTP** [Online]. 2023, May 15. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>
- [5] Lokesh Gupta. 2022, April 7. **What is REST** [Online]. Available: <https://restfulapi.net/>
- [6] **HTTP response status codes** [Online]. 2023, April 11. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
- [7] **What is AWS** [Online]. 2023. Available: https://aws.amazon.com/what-is-aws/?nc1=f_cc
- [8] **AWS Lambda FAQs** [Online]. 2023. Available: <https://aws.amazon.com/lambda/faqs/>
- [9] **AWS IoT Core FAQs** [Online]. 2023. Available: <https://aws.amazon.com/iot-core/faqs/>
- [10] **AWS Amplify FAQs** [Online]. 2023. Available: <https://aws.amazon.com/amplify/faqs/?nc=sn&loc=6&dn=2>
- [11] **Amazon API Gateway FAQs** [Online]. 2023. Available: <https://aws.amazon.com/api-gateway/faqs/>
- [12] **Amazon S3 FAQs** [Online]. 2023. Available: <https://aws.amazon.com/s3/faqs/?nc=sn&loc=7>
- [13] **Amazon DynamoDB FAQs** [Online]. 2023. Available: <https://aws.amazon.com/dynamodb/faqs/>
- [14] **AWS Identity and Access Management (IAM) FAQs** [Online]. 2023. Available: <https://aws.amazon.com/iam/faqs/?nc=sn&loc=5>
- [15] **Amazon Cognito** [Online]. 2023. Available: https://aws.amazon.com/cognito/?nc2=h_ql_prod_se_cog

- [16] **Amazon CloudWatch FAQs** [Online]. 2023. Available:
<https://aws.amazon.com/cloudwatch/faqs/>
- [17] **AWS CloudFormation FAQs** [Online]. 2023. Available:
<https://aws.amazon.com/cloudformation/faqs/>
- [18] **Flutter Rust bridge** [Online]. 2023. Available:
https://cjycode.com/flutter_rust_bridge/index.html
- [19] **BLoC** [Online]. 2023. Available: <https://bloclibrary.dev/#/coreconcepts>
- [20] **Amplify CLI** [Online]. 2022, October 29. Available: <https://docs.amplify.aws/cli/>
- [21] **Authentication using Amplify Auth** [Online]. 2023, May 9. Available:
<https://docs.amplify.aws/lib/auth/getting-started/q/platform/flutter/>
- [22] **Serverless Framework Documentation** [Online]. 2022. Available:
<https://www.serverless.com/framework/docs>
- [23] **AWS SDK for JavaScript** [Online]. 2023, May 30. Available:
<https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/index.html>