แพลตฟอร์มจัดการอุปกรณ์ IoT : การต่อประสานผ่านเว็บที่ปรับแต่งได้สำหรับการควบคุมและการติดตามสถานะ

IoT DEVICE MANAGEMENT PLATFORM : A CUSTOMIZABLE WEB INTERFACE FOR CONTROL AND MONITORING

โดย

นายกษม มิ่งเมือง

นางสาวชนิดาภา ชาติศักดิ์

นายภัทรจาริน นภากาญจน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2567

แพลตฟอร์มจัดการอุปกรณ์ IoT : การต่อประสานผ่านเว็บที่ปรับแต่งได้สำหรับการควบคุมและการติดตามสถานะ

IoT DEVICE MANAGEMENT PLATFORM : A CUSTOMIZABLE WEB INTERFACE FOR CONTROL AND MONITORING

โดย

นายกษม	มิ่งเมือง	64010027
นางสาวชนิดาภา	ชาติศักดิ์	64010153
นายภัทรจาริน	นภากาญจน์	64011224

อาจารย์ที่ปรึกษา

รศ.ดร. เวธิต ภาคย์พิสุทธิ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2567

ปริญญานิพนธ์ปีการศึกษา 2567

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง แพลตฟอร์มจัดการอุปกรณ์ IoT : การต่อประสานผ่านเว็บที่ปรับแต่งได้สำหรับการควบคุมและการติดตามสถานะ

IoT DEVICE MANAGEMENT PLATFORM : A CUSTOMIZABLE WEB INTERFACE FOR CONTROL AND MONITORING

ผู้จัดทำ

1.	นายกษม	มิ่งเมือง	64010027
2.	นางสาวชนิดาภา	ชาติศักดิ์	64010153
3.	นายภัทรจาริน	นภากาญจน์	64011224

..... **อาจารย์ที่ปรึกษา**

(รศ.ดร. เวธิต ภาคย์พิสุทธิ์)

กิตติกรรมประกาศ

ปริญญานิพนธ์เรื่อง “แพลตฟอร์มจัดการอุปกรณ์ IoT : การต่อประสานผ่านเว็บที่ปรับแต่งได้สำหรับการควบคุมและการติดตามสถานะ” ฉบับนี้จะไม่สามารถสำเร็จลุล่วงไปได้หากไม่ได้รับความอนุเคราะห์และการสนับสนุนอย่างดียิ่งจากหลายฝ่าย

ขอขอบพระคุณ รศ.ดร. เวธิต ภาคย์พิสุทธิ์ อาจารย์ที่ปรึกษา ที่กรุณาให้คำปรึกษา คำแนะนำที่เป็นประโยชน์ รวมถึงการชี้แนะแนวทางในการแก้ไขปัญหาต่าง ๆ ตลอดระยะเวลาที่ทำการศึกษาวิจัย จนทำให้ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้อย่างสมบูรณ์

ขอขอบพระคุณคณาจารย์และเจ้าหน้าที่ประจำภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่าน ที่ได้มอบความรู้ ประสบการณ์ และการอบรมสั่งสอนประสิทธิ์ประสาทวิชา ความรู้และประสบการณ์ที่เป็นประโยชน์ตลอดระยะเวลาที่ศึกษาอยู่ในสถาบันแห่งนี้ที่ให้แก่ผู้จัดทำ

ท้ายที่สุดนี้ผู้จัดทำขอขอบพระคุณ บิดา มารดา และครอบครัว ที่ให้ความรักความห่วงใย และเป็นกำลังใจที่สำคัญตลอดมา อีกทั้งยังสนับสนุนโอกาสทางการศึกษาที่มีคุณค่าอันยิ่งใหญ่ให้แก่ผู้จัดทำ

นายกษม มิ่งเมือง
นางสาวชนิดาภา ชาติศักดิ์
นายภัทรจาริน นภากาญจน์
ผู้จัดทำ

แพลตฟอร์มจัดการอุปกรณ์ IoT : การต่อประสานผ่านเว็บที่ปรับแต่งได้สำหรับการควบคุมและการติดตามสถานะ

IoT DEVICE MANAGEMENT PLATFORM : A CUSTOMIZABLE WEB INTERFACE FOR CONTROL AND MONITORING

โดย นายกษม มิ่งเมือง 64010027
นางสาวชนิดาภา ชาติศักดิ์ 64010153
นายภัทรจาริน นภากาญจน์ 64011224

อาจารย์ที่ปรึกษา รศ.ดร. เวธิต ภาคย์พิสุทธิ์

บทคัดย่อ

ปริญญานิพนธ์นี้มีวัตถุประสงค์เพื่อออกแบบและพัฒนาแพลตฟอร์มจัดการอุปกรณ์ IoT โดยการต่อประสานผ่านเว็บที่ปรับแต่งได้สำหรับการควบคุมและการติดตามสถานะ โดยรองรับอุปกรณ์หลากหลายประเภท เช่น รถบังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ แพลตฟอร์มถูกออกแบบให้สามารถสั่งการและตรวจสอบสถานะของอุปกรณ์ได้ผ่านเว็บแอปพลิเคชัน โดยที่อุปกรณ์ IoT สามารถปรับเปลี่ยนเครือข่ายที่เชื่อมต่อได้โดยไม่ต้องแก้ไขโปรแกรมหรืออัปโหลดโปรแกรมลงอุปกรณ์ใหม่ อีกทั้งยังสามารถนำเข้าอุปกรณ์จากผ่านนอกเข้ามาร่วมใช้งานกับแพลตฟอร์ม และสามารถปรับแต่งปุ่มการสั่งงานและการติดตามสถานะตามชนิดของอุปกรณ์จากความต้องการของผู้ใช้งาน เพื่อเพิ่มความสะดวกในการจัดการ ควบคุม และเฝ้าติดตามอุปกรณ์จากระยะไกลอย่างมีประสิทธิภาพและปลอดภัย

ABSTRACT

This project aims to design and develop an IoT device management platform with a customizable web interface for control and monitoring. The platform supports a wide range of devices, such as remote-controlled cars, robotic arms, plant watering devices, and smoke detectors. It is designed to allow users to operate and monitor devices through a web application, enabling IoT devices to switch between networks without requiring program modifications or firmware uploads. Additionally, external devices can be integrated into the platform, and the control buttons and status monitoring features can be customized based on the type of device and user preferences. This enhances the convenience, efficiency, and security of remote device management, control, and monitoring.

สารบัญ

		หน้า
กิตติกรรมประกาศ		I
บทคัดย่อ		II
สารบัญ		IV
สารบัญรูป		XI
สารบัญตาราง		XXVII
บทที่ 1	**บทนำ**	
	1.1 ความเป็นมาและความสำคัญของปัญหา	1
	1.2 วัตถุประสงค์	1
	1.3 ขอบเขตของปริญญานิพนธ์	1
บทที่ 2	**ทฤษฎีและหลักการที่เกี่ยวข้อง**	
	2.1 IoT (Internet of Thing)	3
	2.1.1 สถาปัตยกรรม Internet of Things	3
	2.2 MQTT (Message Queuing Telemetry Transport)	5
	2.2.1 Publish-Subscribe Model	5
	2.2.2 โครงสร้างแพ็กเกจของ MQTT	8
	2.2.3 QoS	9
	2.2.4 กระบวนการเชื่อมต่อระหว่าง Client และ Broker (Server)	10
	2.2.4.1 HiveMQ	13

สารบัญ (ต่อ)

หน้า

2.2.4.2 Adafruit 14

2.3 บอร์ดไมโครคอนโทรลเลอร์ ESP32 14

2.3.1 โปรแกรมการสั่งงาน 15

2.3.2 ESP32 DevKit v1 16

2.3.3 Flash Memory 18

2.3.4 EEPROM 19

2.3.5 การเชื่อมต่อผ่าน Wi-Fi 20

2.4 ฮาร์ดแวร์ส่วนรถบังคับ 21

2.4.1 DC Gear Motor 21

2.4.2 โมดูลขับมอเตอร์ L298N 22

2.5 ฮาร์ดแวร์ส่วนแขนกล 23

2.5.1 Micro Servo MG90S 23

2.6 ฮาร์ดแวร์ส่วนอุปกรณ์รดน้ำต้นไม้ 25

2.6.1 โมดูลเซนเซอร์วัดความชื้นในดิน 25

2.6.2 Micro Submersible Water Pump 26

2.6.3 รีเลย์ JQC-3F-5VDC 28

2.7 ฮาร์ดแวร์ส่วนอุปกรณ์ตรวจจับควันไฟ 29

2.7.1 MQ-2 Gas Sensor Module 29

2.7.2 เซนเซอร์วัดอุณหภูมิและความชื้น SHTC3 30

2.7.3 โมดูล Active Buzzer 31

สารบัญ (ต่อ)

	หน้า
2.8 เว็บแอปพลิเคชัน	32
2.8.1 เว็บแอปพลิเคชัน	32
2.8.2 เว็บเบราว์เซอร์	34
2.8.3 เว็บเซิร์ฟเวอร์	34
2.8.4 ฐานข้อมูล	34
2.8.5 HTTP (Hypertext Transfer Protocol)	34
2.8.6 API (Application Programming Interface)	35
2.8.7 REST	35
2.8.8 WebSocket	36
2.8.9 Next.js	37
2.8.10 Tailwind CSS	37
2.8.11 JSON (JavaScript Object Notation)	38
2.8.12 Cookies	38
2.8.13 ภาษาคอมพิวเตอร์	38
2.9 ข้อมูล	40
2.9.1 ชนิดของข้อมูล (Type of data)	40
2.10 ฐานข้อมูล	41
2.10.1 ฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Non-Relational Database)	41
2.10.2 ฐานข้อมูลไม่ใช่เชิงสัมพันธ์ประเภท Document Database	41
2.10.3 MongoDB	42

สารบัญ (ต่อ)

		หน้า
	2.11 Feature Model	42
	2.11.1 Feature Diagram	43
บทที่ 3	**การออกแบบและการจัดทำปริญญานิพนธ์**	
	3.1 การออกแบบอุปกรณ์ IoT	47
	3.1.1 การเก็บชื่อ Wi-Fi เริ่มต้นของอุปกรณ์ IoT	47
	3.1.2 การเปลี่ยนชื่อ Wi-Fi ของอุปกรณ์ IoT	48
	3.1.3 การรับและส่งข้อมูลของอุปกรณ์ IoT	53
	3.1.4 การออกแบบอุปกรณ์ IoT	55
	3.2 การออกแบบแพลตฟอร์มตัวกลางควบคุมและจัดการอุปกรณ์ IoT	68
	3.2.1 การทำงานและฟังก์ชันการใช้งานเว็บแอปพลิเคชัน	72
	3.2.2 การนำเข้าอุปกรณ์ตัวอย่าง	102
	3.2.3 การนำเข้าอุปกรณ์จากภายนอก	107
	3.2.4 การปรับแต่งปุ่มสั่งการและหน้าแสดงผล	112
	3.2.5 การกำหนดคำสั่งจากหน้าเว็บแอปพลิเคชัน	119
	3.2.6 การรับค่าการทำงานของอุปกรณ์ IoT	126
	3.2.7 ระบบฐานข้อมูลเพื่อเก็บข้อมูลของอุปกรณ์ IoT และผู้ใช้งาน	128
	3.3 เครื่องมือที่ใช้ในการทดสอบ	135
	3.3.1 คอมพิวเตอร์วางตัก	135
	3.3.2 Postman	137

สารบัญ (ต่อ)

		หน้า
	3.3.3 HiveMQ Cloud Web Client	137
	3.3.4 Adafurit	137
	3.3.5 Google Chrome	138
	3.3.6 Arduino IDE	138
	3.4 การจัดเก็บผลการทดลอง	138
	3.4.1 การทดสอบการทำงานอุปกรณ์	138
	3.4.2 การทดสอบการเชื่อมต่อกับ Broker	138
	3.4.3 การทำงานของเว็บแอปพลิเคชัน	138
	3.4.4 การทดสอบการนำเข้าอุปกรณ์	139
	3.4.5 การทดสอบการเปลี่ยนการเชื่อมต่อ Wi-Fi	139
	3.4.6 การทำงานปรับแต่งปุ่มควบคุมและการแสดงข้อมูลอุปกรณ์ IoT	139
บทที่ 4	**ผลการทดลอง**	
	4.1 ผลการทำงานของอุปกรณ์	140
	4.1.1 รถยนต์บังคับ	140
	4.1.2 แขนกล	147
	4.1.3 อุปกรณ์รดน้ำต้นไม้	156
	4.1.4 อุปกรณ์ตรวจจับควันไฟ	160
	4.1.5 ตัวอย่างอุปกรณ์ควบคุมใด ๆ	167
	4.1.6 ตัวอย่างอุปกรณ์รับข้อมูลใด ๆ	171

สารบัญ (ต่อ)

	หน้า
4.2 การทดสอบการเชื่อมต่อกับ MQTT Broker	176
4.2.1 ผลการเชื่อมต่อ HiveMQ Broker กับเว็บแอปพลิเคชัน	176
4.2.2 ผลการเชื่อมต่อ HiveMQ Broker กับบอร์ดไมโครคอนโทรลเลอร์ ESP32	177
4.2.3 ผลการเชื่อมต่อ Adafruit Broker กับเว็บแอปพลิเคชัน	178
4.2.4 ผลการเชื่อมต่อ Adafruit Broker กับบอร์ดไมโครคอนโทรลเลอร์ ESP32	180
4.3 ผลการทำงานของเว็บแอปพลิเคชัน	181
4.3.1 หน้าเว็บแอปพลิเคชัน (Front-End)	181
4.3.2 ระบบจัดการเว็บแอปพลิเคชัน (Back-End)	191
4.4 การนำเข้าอุปกรณ์	211
4.4.1 ผลทดสอบการนำเข้าอุปกรณ์ตัวอย่าง	211
4.4.2 ผลทดสอบการนำเข้าอุปกรณ์ใด ๆ	214
4.5 การเปลี่ยนการเชื่อมต่อ Wi-Fi	217
4.5.1 ผลทดสอบการเปลี่ยนเครือข่าย Wi-Fi ใหม่ผ่านหน้าเว็บแอปพลิเคชัน	217
4.5.2 ผลการเปลี่ยนเครือข่าย Wi-Fi กลับไปค่าเริ่มต้นผ่านหน้าเว็บแอปพลิเคชัน	219
4.6 การปรับแต่งปุ่มควบคุมและการแสดงข้อมูลอุปกรณ์ IoT	221
4.6.1 ผลการทดสอบการปรับแต่งปุ่มในการสั่งการ	221
4.6.2 ผลการทดสอบการปรับแต่งหน้าจอแสดงผลข้อมูลอุปกรณ์ IoT	230

สารบัญ (ต่อ)

		หน้า
บทที่ 5	**สรุปผลและข้อเสนอแนะ**	
	5.1 สรุปผล	234
	5.2 ข้อเสนอแนะ	235
บรรณานุกรม		**236**

สารบัญรูป

หน้า

รูปที่

2.1 โครงสร้างแบบ 3 Layer (Three-layer) 4
2.2 โครงสร้างแบบ 5 Layer (Five-layer) 5
2.3 การทำงานของ Publish-Subscribe Model 6
2.4 โครงสร้างแพ็กเกจของ MQTT 9
2.5 QoS level ใน MQTT 10
2.6 กระบวนการเชื่อมต่อระหว่าง Client และ Broker (Server) 13
2.7 รายละเอียดขา ESP32 DevKit v1 18
2.8 การทำงาน Station Mode 21
2.9 DC Gear Motor 21
2.10 โมดูลขับมอเตอร์ L298N 22
2.11 Micro Servo MG90S 24
2.12 โมดูลเซนเซอร์วัดความชื้นในดิน 25
2.13 Micro Submersible Water Pump 27
2.14 รีเลย์ JQC-3F-5VDC 28
2.15 MQ-2 Gas Sensor Module 29
2.16 เซนเซอร์วัดอุณหภูมิและความชื้น SHTC3 31
2.17 โมดูล Active Buzzer 32
2.18 Three-Tier Architecture 33
2.19 การเชื่อมต่อผ่านเว็บเซิร์ฟเวอร์ 34
2.20 การทำงานของ API 35
2.21 การเชื่อมต่อระหว่างคลไอเอนต์และเซิร์ฟเวอร์ 37
2.22 ตัวอย่างข้อมูลจะถูกจัดเก็บในรูปแบบเอกสาร JSON 41
2.23 โครงสร้างพื้นฐานของ Document Database 42
2.24 ความสัมพันธ์แบบ Mandatory 43

สารบัญรูป (ต่อ)

หน้า

รูปที่

2.25 ความสัมพันธ์แบบ Optional 44
2.26 ความสัมพันธ์แบบ Or 44
2.27 ความสัมพันธ์แบบ Alternative 44
2.28 ข้อจำกัดที่เชื่อมโยงคุณสมบัติแบบ Requires 45
2.29 ข้อจำกัดที่เชื่อมโยงคุณสมบัติแบบ Excludes 45
3.1 บล็อกไดอะแกรมภาพรวมของปริญญานิพนธ์ 46
3.2 คำสั่งโปรแกรมการบันทึก SSID และรหัสผ่าน Wi-Fi เริ่มต้น 47
3.3 คำสั่งโปรแกรมนำค่าเริ่มต้น Wi-Fi เพื่อใช้ในการเชื่อมต่อ 47
3.4 คำสั่งโปรแกรมในการเชื่อมต่อ Wi-Fi เริ่มต้น 48
3.5 คำสั่งโปรแกรมเมื่อรีเซ็ตกลับไปเป็น Wi-Fi เริ่มต้น 48
3.6 กระบวนการเปลี่ยนการเชื่อมต่อ Wi-Fi 49
3.7 คำสั่งโปรแกรมการทำงานรับ Payload จาก MQTT 50
3.8 คำสั่งโปรแกรมตรวจสอบข้อมูลในหน่วยความจำ EEPROM 51
3.9 คำสั่งโปรแกรมแยกข้อความ 52
3.10 คำสั่งโปรแกรมเพื่อบันทึกข้อมูลเข้า EEPROM 52
3.11 การสื่อสารผ่าน MQTT Protocol 53
3.12 ตัวอย่างคำสั่งโปรแกรมส่งข้อมูล (Publish) 53
3.13 ตัวอย่าง Subscribe ไปยัง Topic และฟังก์ชัน Callback อ่านค่าข้อมูล 54
3.14 การจำลองวงจรอุปกรณ์ควบคุมใด ๆ 55
3.15 การจำลองวงจรอุปกรณ์รับข้อมูลใด ๆ 56
3.16 คำสั่งโปรแกรมนำเข้าไลบรารี 57
3.17 คำสั่งโปรแกรมกำหนดตัวแปรเก็บข้อมูล 57
3.18 คำสั่งโปรแกรมกำหนดขนาด EEPROM และเชื่อมต่อกับ MQTT Broker 57
3.19 คำสั่งโปรแกรมอ่านข้อมูลจาก EEPROM 58

สารบัญรูป (ต่อ)

หน้า

รูปที่

3.20 คำสั่งโปรแกรมตรวจสอบข้อมูลจาก EEPROM และทำการเชื่อมต่อกับ Wi-Fi 59
3.21 คำสั่งโปรแกรมเชื่อมต่อ ESP32 กับ Wi-Fi 59
3.22 คำสั่งโปรแกรมเชื่อมต่อ ESP32 กับ MQTT broker 60
3.23 ฟังก์ชันลบข้อมูลเก่าจาก EEPROM และแยกข้อมูล Wi-Fi 61
3.24 คำสั่งโปรแกรมนี้จะทำการเขียนข้อมูล Wi-Fi ใหม่ลงใน EEPROM 61
3.25 คำสั่งโปรแกรมนี้จะทำการลบข้อมูลที่เก็บไว้ใน EEPROM เมื่อได้รับคำสั่ง "defaultwifi" 62
3.26 คำสั่งโปรแกรมเช็คข้อความที่ได้รับจาก MQTT Broker เริ่มต้นด้วย "ctrl/" 62
3.27 คำสั่งโปรแกรมจะส่งข้อความ "online" ไปยัง MQTT broker 63
3.28 การจำลองวงจรรถยนต์บังคับ 64
3.29 การจำลองวงจรแขนกล 65
3.30 การจำลองวงจรอุปกรณ์รดน้ำต้นไม้ 66
3.31 การจำลองวงจรอุปกรณ์ตรวจจับควันไฟ 67
3.32 แผนภาพคุณสมบัติภายในเว็บแอปพลิเคชัน 68
3.33 ขั้นตอนการใช้งานภายในระบบ 71
3.34 กระบวนการทำงานของเว็บแอปพลิเคชัน 72
3.35 โครงร่างเว็บแอปพลิเคชันหน้า Sign up 75
3.36 กระบวนการทำงานของเว็บแอปพลิเคชันหน้า Sign up 76
3.37 โครงร่างเว็บแอปพลิเคชันหน้า Login 77
3.38 กระบวนการทำงานของเว็บแอปพลิเคชันหน้า Login 78
3.39 โครงร่างเว็บแอปพลิเคชันหน้าหลัก 79
3.40 แถบนำทาง (Navigation Bar) 79
3.41 โครงร่างเว็บแอปพลิเคชันหน้า Document 80
3.42 โครงร่างเว็บแอปพลิเคชันหน้า Device 81
3.43 โครงร่างเว็บแอปพลิเคชัน Import Example Device 81

สารบัญรูป (ต่อ)

หน้า

รูปที่

3.44 โครงร่างเว็บแอปพลิเคชันเมื่อนำเข้าอุปกรณ์ตัวอย่าง 82
3.45 กระบวนการทำงานในการนำเข้าอุปกรณ์ตัวอย่าง 83
3.46 โครงร่างเว็บแอปพลิเคชันหน้าควบคุมอุปกรณ์ 84
3.47 กระบวนการทำงานในการควบคุมอุปกรณ์ 85
3.48 โครงส่วนหน้าเริ่มต้นของอุปกรณ์ 85
3.49 โครงร่างเว็บแอปพลิเคชันการ Custom Button Feature 86
3.50 กระบวนการทำงานในการ Custom Button Feature 87
3.51 โครงร่างเว็บแอปพลิเคชันการ Custom Button Feature 88
3.52 โครงร่างหน้าต่างปรับแต่งหน้าปัดแสดงผลวงกลม 88
3.53 กระบวนการทำงานในการ Custom Monitor Display Feature 89
3.54 โครงร่างเว็บแอปพลิเคชันหน้า Import New Device 90
3.55 กระบวนการทำงานหน้า Import New Device 91
3.56 โครงร่างเว็บแอปพลิเคชันหน้าตั้งค่า Wi-Fi 92
3.57 กระบวนการทำงานในการตั้งค่า Wi-Fi 93
3.58 โครงร่างเว็บแอปพลิเคชันหน้า Product 94
3.59 โครงร่างเว็บแอปพลิเคชันหน้า About Us 95
3.60 โครงร่างเว็บแอปพลิเคชันหน้าข้อมูลผู้ใช้งาน 95
3.61 กระบวนการนำเข้าอุปกรณ์ตัวอย่าง 103
3.62 คำสั่งโปรแกรมดึงข้อมูลอุปกรณ์ของผู้ใช้ผ่าน getDeviceByUser 104
3.63 คำสั่งโปรแกรมเพื่อระบุว่าการโหลดข้อมูลเสร็จสมบูรณ์แล้ว 104
3.64 คำสั่งโปรแกรมเมื่อผู้ใช้ต้องการเพิ่มอุปกรณ์ใหม่ 105
3.65 คำสั่งโปรแกรมตรวจสอบข้อมูลอุปกรณ์ตรวจสอบ 105
3.66 คำสั่งโปรแกรมตรวจสอบสถานะอุปกรณ์และสร้าง UUID สำหรับอุปกรณ์และ WiFi 105
3.67 คำสั่งโปรแกรม POST Method โดยจะส่งข้อมูลเข้าฐานข้อมูล 106

สารบัญรูป (ต่อ)

หน้า

รูปที่

3.68 คำสั่งโปรแกรมเพื่อบันทึกข้อมูลไปยัง API 106
3.69 คำสั่งโปรแกรมส่งคำขอ PUT ไปยัง API 107
3.70 คำสั่งโปรแกรมตรวจสอบว่าการเพิ่มอุปกรณ์สำเร็จ 107
3.71 กระบวนการนำเข้าอุปกรณ์จากภายนอก 108
3.72 คำสั่งโปรแกรมเก็บค่าการเชื่อมต่อกับ MQTT Broker 109
3.73 คำสั่งโปรแกรมสร้างหน้าต่างเลือกการเชื่อมต่อ MQTT Broker 109
3.74 คำสั่งโปรแกรมสร้างช่องรับข้อมูลในการเชื่อมต่อ HiveMQ Broker 110
3.75 คำสั่งโปรแกรมสร้างช่องรับข้อมูลในการเชื่อมต่อ Adafruit Broker 111
3.76 คำสั่งโปรแกรมดึงข้อมูลอุปกรณ์ที่นำเข้ามาตาม ID ของผู้ใช้งาน 111
3.77 คำสั่งโปรแกรมเพื่อเรียก getExternalDevice(userId) 111
3.78 คำสั่งโปรแกรมแสดงข้อมูลของอุปกรณ์นำเข้า 112
3.79 กระบวนการทำงานของฟังก์ชันปรับแต่งปุ่มสั่งการ 113
3.80 คำสั่งโปรแกรมในการเก็บข้อมูลที่ผู้ใช้เลือก 113
3.81 ตัวอย่างคำสั่งโปรแกรมในการให้ผู้ใช้เลือกค่าต่าง ๆ 114
3.82 คำสั่งโปรแกรมในการให้ผู้ใช้กำหนดคำสั่งการ 114
3.83 คำสั่งโปรแกรมส่งข้อมูลของปุ่มใหม่ไปยังเซิร์ฟเวอร์ผ่าน API 115
3.84 กระบวนการทำงานของฟังก์ชันปรับแต่งหน้าแสดงผล 116
3.85 คำสั่งโปรแกรมกำหนดค่าเบื้องต้น 116
3.86 คำสั่งโปรแกรมเลือกแผนภูมิโดนัท 117
3.87 คำสั่งโปรแกรมเลือกแผนภูมิวงกลม 117
3.88 คำสั่งโปรแกรมเลือกแสดงผลเป็นวงกลม 117
3.89 คำสั่งโปรแกรมตัวเลือกสีพื้นหลัง 118
3.90 คำสั่งโปรแกรมตัวเลือกสีตัวหนังสือและหน่วยแสดงผล 118
3.91 กระบวนการทำงานของคำสั่งที่กำหนดไว้ภายในเว็บแอปพลิเคชัน 119

สารบัญรูป (ต่อ)

หน้า

รูปที่

3.92 คำสั่งโปรแกรมสร้างหน้าต่างตัวเลือกคำสั่ง	120
3.93 ฟังก์ชันบันทึกปุ่ม	121
3.94 ฟังก์ชันโปรแกรมส่งค่าคำสั่งขึ้น Broker	121
3.95 คำสั่งโปรแกรมดึงข้อมูลปุ่มจาก API โดยใช้ ID ของปุ่มเป็นตัวระบุ	122
3.96 คำสั่งโปรแกรมแสดงข้อมูลปุ่ม	122
3.97 กระบวนการทำงานของคำสั่งที่ผู้ใช้กำหนดเอง	123
3.98 คำสั่งโปรแกรมสร้าง Input field รับค่าคำสั่ง	123
3.99 ฟังก์ชันบันทึกปุ่ม	124
3.100 ฟังก์ชันโปรแกรมส่งค่าคำสั่งขึ้น Broker	124
3.101 คำสั่งโปรแกรมดึงข้อมูลปุ่มจาก API โดยใช้ ID ของปุ่มเป็นตัวระบุ	125
3.102 คำสั่งโปรแกรมแสดงข้อมูลปุ่ม	125
3.103 กระบวนการทำงานของการรับค่าการทำงานของอุปกรณ์ IoT	126
3.104 คำสั่งโปรแกรมเชื่อมต่อ MQTT Broker	127
3.105 คำสั่งโปรแกรม Subscribe Topic	127
3.106 คำสั่งโปรแกรมแยกข้อความและจับคู่ค่าที่ตรงกัน	128
3.107 คำสั่งโปรแกรมแสดงผลในผ่าน Device log	128
3.108 ความสัมพันธ์กันระหว่างข้อมูลภายในเว็บแอปพลิเคชัน	134
3.109 ACER NITRO V 15 ANV15-51-574G	135
3.110 LENOVO IDEAPAD 330S-14IKB-81F401MKTA	136
3.111 ASUS TUF GAMING F15 FX506LH-HN002T	137
4.1 รถยนต์บังคับ	140
4.2 คำสั่งโปรแกรมกำหนดขาการทำงาน	141
4.3 คำสั่งโปแกรมตรวจสอบเงื่อนไขแต่ละคำสั่ง	141
4.4 คำสั่งโปรแกรมฟังก์ชัน forward	141

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.5 ผลการทำงานขา INP1 และ INP3 ได้รับสัญญาณ HIGH 142
4.6 ผลการทำงานขา INP2 และ INP4 ได้รับสัญญาณ LOW 142
4.7 คำสั่งโปรแกรมฟังก์ชัน backward 143
4.8 ผลการทำงานขา INP2 และ INP4 ได้รับสัญญาณ HIGH 143
4.9 ผลการทำงานขา INP1 และ INP3 ได้รับสัญญาณ LOW 143
4.10 คำสั่งโปรแกรมฟังก์ชัน left 144
4.11 ผลการทำงานขา INP2 และ INP3 ได้รับสัญญาณ HIGH 144
4.12 ผลการทำงานขา INP1 และ INP4 ได้รับสัญญาณ LOW 145
4.13 คำสั่งโปรแกรมฟังก์ชัน right 145
4.14 ผลการทำงานขา INP1 และ INP4 ได้รับสัญญาณ HIGH 145
4.15 ผลการทำงานขา INP2 และ INP3 ได้รับสัญญาณ LOW 146
4.16 คำสั่งโปรแกรมฟังก์ชัน stop 146
4.17 ผลการทำงานขา INP1 และ INP2 ได้รับสัญญาณ LOW 146
4.18 ผลการทำงานขา INP3 และ INP4 ได้รับสัญญาณ LOW 147
4.19 แขนกล 147
4.20 คำสั่งโปรแกรมกำหนดขาการทำงาน 148
4.21 คำสั่งโปรแกรม forward 148
4.22 ผลการทำงานมอเตอร์ M2 เมื่อได้รับข้อความ forward 148
4.23 คำสั่งโปรแกรม backward 149
4.24 ผลการทำงานมอเตอร์ M2 เมื่อได้รับข้อความ backward 149
4.25 ผลการทำงานมอเตอร์ M2 เมื่อต้องหยุดหมุน 150
4.26 คำสั่งโปรแกรม up 150
4.27 ผลการทำงานมอเตอร์ M3 เมื่อได้รับข้อความ up 151
4.28 คำสั่งโปรแกรม down 151

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.29 ผลการทำงานมอเตอร์ M3 เมื่อได้รับข้อความ down 151
4.30 ผลการทำงานมอเตอร์ M3 เมื่อต้องหยุดหมุน 152
4.31 คำสั่งโปรแกรม left 152
4.32 ผลการทำงานมอเตอร์ M1 เมื่อได้รับข้อความ left 153
4.33 คำสั่งโปรแกรม right 153
4.34 ผลการทำงานมอเตอร์ M3 เมื่อได้รับข้อความ right 153
4.35 ผลการทำงานมอเตอร์ M3 เมื่อต้องหยุดหมุน 154
4.36 คำสั่งโปรแกรม holdON 154
4.37 ผลการทำงานมอเตอร์ M4 เมื่อได้รับข้อความ holdON 155
4.38 คำสั่งโปรแกรม holdOFF 155
4.39 ผลการทำงานมอเตอร์ M4 เมื่อได้รับข้อความ holdOFF 155
4.40 อุปกรณ์รดน้ำต้นไม้ 156
4.41 คำสั่งโปรแกรมกำหนดขาการทำงาน 156
4.42 คำสั่งโปรแกรมตรวจสอบเงื่อนไขข้อความ 157
4.43 คำสั่งโปรแกรมที่ควบคุมรีเลย์และวัดความชื้นในดิน 157
4.44 คำสั่งโปรแกรมฟังก์ชัน startPump 158
4.45 คำสั่งโปรแกรมฟังก์ชัน stopPump 158
4.46 ผลการทำงานของรีเลย์เมื่อรับคำสั่ง on และมีความชื้นในดิน 0 เปอร์เซ็นต์ 158
4.47 คำสั่งโปรแกรมตรวจสอบเงื่อนไขข้อความ 158
4.48 คำสั่งโปรแกรมฟังก์ชัน stopPump 159
4.49 ผลการทำงานของรีเลย์เมื่อรับคำสั่ง on และมีความชื้นในดิน 60 เปอร์เซ็นต์ 159
4.50 คำสั่งโปรแกรมในการส่งค่าให้อยู่ในรูปแบบ value 159
4.51 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value 159
4.52 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน 160

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.53 อุปกรณ์ตรวจจับควันไฟ 160
4.54 คำสั่งโปรแกรมกำหนดขาการทำงาน 161
4.55 คำสั่งโปรแกรมอ่านข้อมูลอุณหภูมิและความชื้นจากเซนเซอร์ SHTC3 161
4.56 อุณหภูมิและความชื้นสัมพันธ์ผ่าน Serial Monitor 161
4.57 ฟังก์ชันเปรียบเทียบเซนเซอร์ MQ-2 162
4.58 ฟังก์ชันอ่านค่าความต้านทาน 163
4.59 ฟังก์ชันคำนวณความเข้มข้นของแก๊ส 163
4.60 คำสั่งโปรแกรมอ่านค่าเซนเซอร์ MQ-2 และตรวจสอบเงื่อนไข 164
4.61 ค่าที่วัดได้สภาพอากาศปกติไม่มีควันไฟหรือการเผาไหม้ 164
4.62 ค่าที่วัดได้เมื่อเกิดการเผาไหม้เกิดขึ้น 164
4.63 คำสั่งโปรแกรมส่งค่าจากอุปกรณ์ตรวจจับควันขึ้น Broker 165
4.64 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value 165
4.65 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน 165
4.66 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value 166
4.67 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน 166
4.68 ตัวอย่างอุปกรณ์ควบคุมใด ๆ 167
4.69 คำสั่งโปรแกรมกำหนดขาการทำงาน 167
4.70 คำสั่งโปรแกรมกำหนดการทำงานเริ่มต้น 168
4.71 ฟังก์ชันที่ใช้ควบคุมรีเลย์และไฟ RGB LED 168
4.72 คำสั่งโปรแกรม red_on 168
4.73 ผลการทำงานของรีเลย์เมื่อ true และไฟ LED สีแดง true 169
4.74 คำสั่งโปรแกรม green_on 169
4.75 ผลการทำงานของรีเลย์เมื่อ true และไฟ LED สีแดง true 169
4.76 คำสั่งโปรแกรม blue_on 170

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.77 ผลการทำงานของรีเลย์เมื่อ true และไฟ LED สีแดง true 170
4.78 คำสั่งโปรแกรม stop 170
4.79 ผลการทำงานของรีเลย์เมื่อ false และไฟ LED สีแดง false 171
4.80 ผลการทำงานของไฟ LED สีเขียวและไฟ LED สีน้ำเงิน false 171
4.81 ตัวอย่างอุปกรณ์รับข้อมูลใด ๆ 172
4.82 คำสั่งโปรแกรมกำหนดขาการทำงาน 172
4.83 คำสั่งโปรแกรมอ่านค่าจากเซนเซอร์ MQ-6 172
4.84 คำสั่งโปรแกรมคำนวณค่าความต้านทานของเซนเซอร์ 173
4.85 คำสั่งโปรแกรมคำนวณความเข้มข้นของก๊าซปิโตรเลียมเหลวในหน่วย ppm 173
4.86 เงื่อนไขในการตรวจสอบก๊าซปิโตรเลียมเหลวเกินค่าที่กำหนด 174
4.87 ค่าที่วัดได้สภาพอากาศปกติไม่มีแก๊สรั่วไหล 174
4.88 ค่าที่วัดได้เมื่อเกิดแก๊สรั่วไหล 174
4.89 คำสั่งโปรแกรมส่งค่าจากอุปกรณ์ตรวจจับก๊าซปิโตรเลียมเหลว 174
4.90 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value 175
4.91 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน 175
4.92 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value 175
4.93 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน 176
4.94 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker 176
4.95 คำสั่งโปรแกรมการเชื่อมต่อ 177
4.96 ผลการเชื่อมต่อผ่าน Console เบราว์เซอร์ 177
4.97 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker 177
4.98 คำสั่งโปรแกรมเชื่อมต่อบอร์ดไมโครคอนโทรลเลอร์ ESP32 กับ HiveMQ Broker 178
4.99 ผลการเชื่อมต่อผ่าน Serial Monitor 178
4.100 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker 179

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.101 คำสั่งโปรแกรมการเชื่อมต่อ 179
4.102 ผลการเชื่อมต่อผ่าน Console เบราว์เซอร์ 179
4.103 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker 180
4.104 คำสั่งโปรแกรมเชื่อมต่อบอร์ดไมโครคอนโทรลเลอร์ ESP32 กับ HiveMQ Broker 180
4.105 ผลการเชื่อมต่อผ่าน Serial Monitor 181
4.106 เว็บแอปพลิเคชันหน้า Sign up 181
4.107 กล่องข้อมูล Sign up 182
4.108 กล่องโต้ตอบเมื่อสมัครบัญชีผู้ใช้ 182
4.109 เว็บแอปพลิเคชันหน้า Login 182
4.110 กล่องข้อมูล Login 183
4.111 กล่องโต้ตอบเมื่อลงชื่อเข้าใช้ 183
4.112 ข้อมูลบัญชีผู้ใช้ที่ได้ลงชื่อเข้าใช้งานเว็บแอปพลิเคชัน 183
4.113 เว็บแอปพลิเคชันหน้าหลัก 184
4.114 เว็บแอปพลิเคชันหน้า Device เมื่อไม่มีการนำเข้าอุปกรณ์ตัวอย่าง 184
4.115 ปุ่ม Import Example Device ในเว็บแอปพลิเคชันหน้า Device 185
4.116 กล่องข้อความในการนำเข้าอุปกรณ์ตัวอย่าง 185
4.117 กล่องโต้ตอบเมื่อเพิ่มอุปกรณ์ 185
4.118 เว็บแอปพลิเคชันหน้า Device เมื่อนำเข้าอุปกรณ์ตัวอย่าง 186
4.119 เว็บแอปพลิเคชันหน้า Car 186
4.120 เว็บแอปพลิเคชันหน้า Robotic Arm 187
4.121 เว็บแอปพลิเคชันหน้า Pump 188
4.122 เว็บแอปพลิเคชันหน้า Smoke Detector 188
4.123 เว็บแอปพลิเคชันหน้า Import New Device ของตัวอย่างอุปกรณ์ควบคุมใด ๆ 189
4.124 เว็บแอปพลิเคชันหน้า Import New Device ของตัวอย่างอุปกรณ์รับข้อมูลใด ๆ 189

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.125 เว็บแอปพลิเคชันหน้า Product 190
4.126 เว็บแอปพลิเคชันหน้า Documentation 191
4.127 ทดสอบ Method GET ผ่าน Postman 191
4.128 ผลการทดสอบเมื่อใช้ Method GET 192
4.129 ทดสอบ Method POST ผ่าน Postman 192
4.130 ผลการทดสอบเมื่อใช้ Method POST 192
4.131 ทดสอบ Method PUT ผ่าน Postman 193
4.132 ข้อมูลเดิม 193
4.133 ข้อความอัปเดตเพื่อ PUT เข้าฐานข้อมูล 193
4.134 ผลการทดสอบเมื่อใช้ Method POST 194
4.135 ทดสอบ Method GET ด้วย ID ผ่าน Postman 194
4.136 ผลการทดสอบเมื่อใช้ Method GET 195
4.137 ทดสอบ Method POST ผ่าน Postman 195
4.138 ข้อมูล JSON ที่ POST 195
4.139 ผลการทดสอบเมื่อใช้ Method POST 196
4.140 ทดสอบ Method PUT ผ่าน Postman 196
4.141 ข้อมูล JSON ที่ PUT 196
4.142 ข้อมูลเดิม 197
4.143 ผลการทดสอบเมื่อใช้ Method PUT 197
4.144 ทดสอบ Method GET ด้วย ID ผ่าน Postman 197
4.145 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID 198
4.146 ทดสอบ Method GET ผ่าน Postman 198
4.147 ผลการทดสอบเมื่อใช้ Method GET 199
4.148 ทดสอบ Method POST ผ่าน Postman 199

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.149 ข้อมูล JSON ที่ POST 199
4.150 ผลการทดสอบเมื่อใช้ Method POST 200
4.151 ทดสอบ Method DELETE ผ่าน Postman 200
4.152 ผลการทดสอบเมื่อใช้ Method DELETE 200
4.153 ทดสอบ Method GET ด้วย ID ผ่าน Postman 201
4.154 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID 201
4.155 ทดสอบ Method GET ผ่าน Postman 202
4.156 ผลการทดสอบเมื่อใช้ Method GET 202
4.157 ทดสอบ Method POST ผ่าน Postman 202
4.158 ข้อมูล JSON ที่ POST 203
4.159 ผลการทดสอบเมื่อใช้ Method POST 203
4.160 ทดสอบ Method PUT ผ่าน Postman 203
4.161 ข้อมูล JSON ที่ POST 204
4.162 ผลการทดสอบเมื่อใช้ Method PUT 204
4.163 ทดสอบ Method GET ด้วย ID ผ่าน Postman 204
4.164 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID 204
4.165 ทดสอบ Method GET ด้วย deviceId ผ่าน Postman 205
4.166 ผลการทดสอบเมื่อใช้ Method GET 205
4.167 ทดสอบ Method POST ผ่าน Postman 206
4.168 ข้อมูล JSON ที่ POST 206
4.169 ผลการทดสอบเมื่อใช้ Method POST 206
4.170 ทดสอบ Method DELETE ผ่าน Postman 207
4.171 ผลการทดสอบเมื่อใช้ Method DELETE 207
4.172 ทดสอบ Method GET ด้วย ID ผ่าน Postman 207

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.173 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID 208
4.174 ทดสอบ Method GET ผ่าน Postman 208
4.175 ผลการทดสอบเมื่อใช้ Method GET 209
4.176 ทดสอบ Method POST ผ่าน Postman 209
4.177 ข้อมูล JSON ที่ POST 209
4.178 ผลการทดสอบเมื่อใช้ Method POST 210
4.179 ทดสอบ Method DELETE ผ่าน Postman 210
4.180 ผลการทดสอบเมื่อใช้ Method DELETE 210
4.181 ทดสอบ Method GET ด้วย ID ผ่าน Postman 211
4.182 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID 211
4.183 หน้าต่างรับข้อมูลรถยนต์บังคับ 212
4.184 หน้ารวมแสดงข้อมูลอุปกรณ์ที่ได้เพิ่มภายในบัญชี 212
4.185 กล่องตอบโต้เมื่อเพิ่มอุปกรณ์สำเร็จ 212
4.186 กล่องตอบโต้เมื่อเพิ่มอุปกรณ์ไม่สำเร็จ 212
4.187 หน้าต่างรับข้อมูลแขนกล 213
4.188 หน้าต่างรับข้อมูลอุปกรณ์รดน้ำต้นไม้ 213
4.189 หน้าต่างรับข้อมูลอุปกรณ์ตรวจจับควันไฟ 213
4.190 หน้ารวมอุปกรณ์ 214
4.191 ปุ่ม Import New Device 214
4.192 หน้าเว็บแอปพลิเคชัน Import New Device 215
4.193 หน้าต่างรับค่าเพื่อเชื่อมต่อกับ HiveMQ Broker 215
4.194 หน้าต่างรับค่าเพื่อเชื่อมต่อกับ Adafruit Broker 216
4.195 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker 216
4.196 หน้ารวมอุปกรณ์นำเข้าจากภายนอก 217

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.197 ค่า Wi-Fi เริ่มต้นที่แสดงผลในหน้าเว็บแอปพลิเคชัน 218
4.198 Wi-Fi เมื่อผู้ใช้ทำการเปลี่ยนผ่านหน้าเว็บแอปพลิเคชัน 218
4.199 SSID และ Password ที่ถูก Publish จากหน้าเว็บแอปพลิเคชัน 219
4.200 บันทึก SSID และ Password ลงใน EEPROM 219
4.201 หน้าเว็บแอปพลิเคชันเพื่อเปลี่ยนค่ากลับไปเป็น Wi-Fi เริ่มต้น 220
4.202 ข้อความที่ถูก Publish จากหน้าเว็บแอปพลิเคชัน 220
4.203 ลบข้อมูล SSID และ Password ออกจาก EEPROM 220
4.204 ข้อความเมื่อ Wi-Fi ของอุปกรณ์กลับมาเป็นค่าเริ่มต้น 220
4.205 หน้าต่างเพิ่มอุปกรณ์ 221
4.206 หน้าต่างตัวเลือกประเภทอุปกรณ์ 222
4.207 หน้าต่างตัวเลือกประเภทการทำงานของอุปกรณ์ 222
4.208 หน้าต่างการลงทะเบียนเพิ่มอุปกรณ์ 223
4.209 หน้าแสดงข้อมูลอุปกรณ์ของผู้ใช้ 223
4.210 หน้าเริ่มต้นของอุปกรณ์ 224
4.211 ทดสอบกดปุ่มสั่งงานให้เดินหน้า 225
4.212 คำสั่งเข้า Cloud Broker 225
4.213 หน้าแสดงผลเมื่อกดปุ่ม Custom Button 225
4.214 หน้าแสดงผลเมื่อกดปุ่ม Add Button 226
4.215 หน้าต่างตัวเลือกในการปรับแต่งปุ่ม 226
4.216 หน้าต่างตัวเลือกชนิดการทำงานของอุปกรณ์ 227
4.217 หน้าต่างตัวเลือกประเภทการทำงานงานของปุ่ม 227
4.218 หน้าต่างตัวเลือกรูปแบบปุ่ม 227
4.219 หน้าต่างเลือกส่งคำสั่งค่าเริ่มต้น 228
4.220 หน้าต่างเมื่อปรับแต่งปุ่มเสร็จสิ้น 228

สารบัญรูป (ต่อ)

หน้า

รูปที่

4.221 หน้าควบคุมอุปกรณ์เมื่อปรับแต่งปุ่มเสร็จสิ้น 229
4.222 คำสั่งเข้า Cloud Broker 229
4.223 หน้าต่างเลือกส่งคำสั่งใหม่ 229
4.224 ทดสอบกดปุ่มสั่งงาน poweron 230
4.225 คำสั่งเข้า Cloud Broker 230
4.226 หน้าเว็บแอปพลิเคชันอุปกรณ์รดน้ำต้นไม้ 231
4.227 หน้าต่างรับค่าเลือกประเภทการแสดงผลการทำงานของอุปกรณ์ 231
4.228 แผนภูมิโดนัทที่แสดงผลข้อมูลการทำงานของอุปกรณ์ 231
4.229 แผนภูมิวงกลมที่แสดงผลข้อมูลการทำงานของอุปกรณ์ 232
4.230 หน้าต่างเลือกสีพื้นหลังของหน้าปัด 232
4.231 หน้าต่างเลือกสีตัวหนังสือของหน้าปัด 233
4.232 หน้าต่างรับหน่วยแสดงผลข้อมูล 233
4.233 หน้าปัดวงกลมที่แสดงผลข้อมูลการทำงานของอุปกรณ์ 233

สารบัญตาราง

หน้า

ตารางที่

2.1 ตัวอย่าง Topic 7
2.2 ตัวอย่างรูปแบบ Wildcard 7
2.3 คุณลักษณะของบอร์ด ESP32 DevKit v1 16
2.4 คุณลักษณะ Flash Memory 19
2.5 คุณลักษณะ DC Gear Motor 22
2.6 คุณลักษณะโมดูลขับมอเตอร์ L298N 23
2.7 คุณลักษณะ Micro Servo MG90S 24
2.8 คุณลักษณะโมดูลเซนเซอร์วัดความชื้น 26
2.9 คุณลักษณะ Micro Submersible Water Pump 27
2.10 คุณลักษณะรีเลย์ JQC-3F-5VDC 28
2.11 คุณลักษณะ MQ-2 Gas Sensor Module 30
2.12 คุณลักษณะเซนเซอร์วัดอุณหภูมิและความชื้น SHTC3 31
2.13 คุณลักษณะ Active Buzzer 32
2.14 HTTP Methods ที่ใช้งานใน REST API 36
3.1 ตารางข้อมูล User 129
3.2 ตารางข้อมูล Product 129
3.3 ตารางข้อมูล Device 130
3.4 ตารางข้อมูล ExternalDevice 131
3.5 ตารางข้อมูล Wi-Fi 132
3.6 ตารางข้อมูล Button 132
3.7 ตารางข้อมูล Button 133

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เนื่องจากเทคโนโลยี IoT (Internet of Things) มีการเติบโตอย่างรวดเร็วในยุคปัจจุบัน โดยช่วยในการเชื่อมต่ออุปกรณ์ต่าง ๆ ให้สามารถทำงานร่วมกันได้ผ่านอินเทอร์เน็ต การพัฒนาเว็บแอปพลิเคชันที่สามารถควบคุมอุปกรณ์เหล่านี้จึงเป็นปัจจัยสำคัญในการเพิ่มความสะดวกสบายและประสิทธิภาพในการใช้งาน ปริญญานิพนธ์นี้จริงมีจุดประสงค์เพื่อควบคุมและสั่งการอุปกรณ์ IoT ผ่านเว็บแอปพลิเคชันซึ่งเป็นสิ่งที่ผู้ใช้งานต้องการมากขึ้น โดยเฉพาะในระบบที่มีการจัดการอุปกรณ์หลายตัว เพื่อตอบสนองต่อความต้องการของผู้ใช้ในยุคที่เทคโนโลยีเชื่อมต่อกันมากขึ้น การควบคุมอุปกรณ์หลายตัวในคราวเดียวกันผ่านเว็บแอปพลิเคชันจะช่วยเพิ่มประสิทธิภาพในการทำงาน ลดเวลาที่ใช้ในการจัดการอุปกรณ์แต่ละตัว และทำให้การทำงานของอุปกรณ์ต่าง ๆ สามารถทำงานร่วมกันได้ดี อีกทั้งยังตัวกลางในการเชื่อมต่อกับอุปกรณ์ภายนอกที่ผู้ใช้งานกำลังพัฒนา โดยสามารถปรับแต่งการควบคุมและการแสดงผลตามความต้องการของผู้ใช้งาน นอกจากนี้การควบคุมผ่านเว็บแอปพลิเคชันยังเพิ่มความสะดวกในการใช้งาน ผู้ใช้สามารถสั่งการและควบคุมอุปกรณ์จากที่ใดก็ได้ที่มีอินเทอร์เน็ต

1.2 วัตถุประสงค์

1) เพื่อพัฒนาเว็บแอปพลิเคชันให้สามารถควบคุมและสั่งการอุปกรณ์ Internet of Things จากระยะไกล
2) เพื่อออกแบบอุปกรณ์ Internet of Things หลายชนิด โดยสามารถปรับเปลี่ยนเครือข่ายที่เชื่อมต่อได้

1.3 ขอบเขตของปริญญานิพนธ์

ปริญญานิพนธ์นี้เป็นการพัฒนาเว็บแอปพลิเคชันสำหรับการจัดการอุปกรณ์ Internet of Things (IoT) โดยแบ่งเป็น 2 ส่วน คือ

1) อุปกรณ์ Internet of Things ที่ควบคุมการทำงานจากเว็บแอปพลิเคชัน โดยสามารถเปลี่ยนอุปกรณ์ที่ควบคุมได้ด้วยการสั่งการผ่านเว็บแอปพลิเคชันได้ทันที โดยแบ่งอุปกรณ์ 4 ชิ้นเพื่อใช้ในการแสดงการเชื่อมต่อและควบคุม ได้แก่ รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ โดยอุปกรณ์รถยนต์บังคับและแขนกลเป็นการแสดงการทำงานแบบทันทีที่มีการสั่งการผ่านเว็บแอปพลิเคชันเพื่อควบคุมอุปกรณ์ให้เดินหน้า ถอยหลัง หรือทำให้สิ่งที่ผู้ใช้สั่งการ ส่วนอุปกรณ์รดน้ำต้นไม้ตามการสั่งงานจะมีเซนเซอร์วัดความชื้นในดิน โดยจะแสดงผลความชื้นในดินผ่านเว็บแอปพลิเคชันให้แก่ผู้ใช้ และยังสามารถส่งงานให้ตัวอุปกรณ์ทำการรดน้ำได้จากการควบคุมของผู้ใช้ เช่นเดียวกันกับอุปกรณ์ตรวจจับควันไฟ ที่จะมีเซนเซอร์ตรวจจับควันไฟจากการเผาไหม้ที่ทำให้เกิดแก๊สคาร์บอนไดออกไซด์ และเมื่อเกิดควันไฟจากการเผาไหม้จะมีการแจ้งเตือนเข้ามาในหน้าเว็บแอปพลิเคชันของผู้ใช้งาน พร้อมทั้งมีกระดิ่งแจ้งเตือนเมื่อเกิดควันไฟให้แก่ผู้คนที่อยู่ใกล้กับที่เกิดควันไฟ นอกจากนี้ยังสามารถเปลี่ยนการเชื่อมต่อเครือข่าย Wi-Fi ได้ โดยเริ่มจากผู้ใช้ป้อนชื่อและรหัสผ่าน Wi-Fi ผ่านเว็บแอปพลิเคชัน ระบบส่งข้อมูลผ่าน Cloud Broker ไปยังบอร์ดไมโครคอนโทรลเลอร์ ESP32 ซึ่งเชื่อมต่อกับ Wi-Fi เครือข่ายเริ่มต้นหรือเครือข่ายเดิม บอร์ดไมโครคอนโทรลเลอร์ ESP32 จะทำการถอดรหัสและบันทึกข้อมูลเครือข่าย Wi-Fi ใหม่ลงใน Flash Memory จากนั้นเชื่อมต่อกับ Wi-Fi เครือข่ายใหม่ และส่งสถานะกลับไปยังเว็บแอปพลิเคชันผ่าน MQTT Protocol กระบวนการนี้ช่วยให้เปลี่ยนการเชื่อมต่อ Wi-Fi ได้ง่ายและรวดเร็วโดยไม่ต้องแก้ไขโค้ดโปรแกรมหรือเริ่มการทำงานของอุปกรณ์ใหม่

2) เว็บแอปพลิเคชันโดยพัฒนาเพื่อควบคุมและสั่งงานการทำงานของอุปกรณ์ผ่านเฟรมเวิร์ค Next.JS นำมาพัฒนาทั้งหน้าต่างเว็บแอปพลิเคชันสำหรับผู้ใช้งานและระบบการใช้งานซอฟต์แวร์ในตัวเดียวกัน เป็นตัวกลางที่เชื่อมต่อระหว่างผู้ใช้กับ Internet of Things เพื่อควบคุมอุปกรณ์ Internet of Things ผ่าน Cloud Broker ที่เป็นตัวกลางสื่อสารแบบ WebSocket Server และส่งข้อมูลผ่าน Wi-fi มาที่เว็บแอปพลิเคชัน โดยจะมีการออกแบบส่วนหน้า (Front-End) ให้มีความเข้าใจง่ายต่อการใช้งานและไม่ซับซ้อนจนเกินไปสำหรับผู้ใช้ อีกทั้งมีฟังก์ชันการปรับแต่งปุ่มควบคุมให้เหมาะแก่การควบคุมตามชนิดอุปกรณ์ นอกจากนี้ยังมีการแสดงผลของความชื้นของดินในกระถาง การแจ้งเตือนเมื่อเกิดควันไฟจากการเผาไหม้ และสถานะการเชื่อมต่อของอุปกรณ์ Internet of Things กับเว็บแอปพลิเคชัน

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 IoT (Internet of Thing)

Internet of Things (IoT) [1] หมายถึง การที่อุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ สามารถเชื่อมโยงและแลกเปลี่ยนข้อมูลกันได้ผ่านอินเทอร์เน็ต โดยไม่ต้องพึ่งพาการป้อนข้อมูลจากมนุษย์ การเชื่อมต่อดังกล่าวทำให้สามารถควบคุมและใช้งานอุปกรณ์เหล่านี้ผ่านเครือข่ายอินเทอร์เน็ตได้อย่างง่ายดาย แนวคิดนี้ถูกเสนอขึ้นครั้งแรกโดย Kevin Ashton ในปี 1999 ซึ่งพัฒนาจากโครงการ Auto-ID Center ที่ใช้เทคโนโลยี RFID (Radio Frequency Identification) เป็นมาตรฐานสำหรับการจับสัญญาณจากเซนเซอร์ต่าง ๆ Kevin ได้นิยามว่าอุปกรณ์อิเล็กทรอนิกส์ที่สามารถสื่อสารกันได้ถือเป็นอุปกรณ์ "internet-like" และได้เลือกใช้คำว่า "Things" เพื่อแทนอุปกรณ์อิเล็กทรอนิกส์ที่เชื่อมโยงกันในระบบ IoT

2.1.1 สถาปัตยกรรม Internet of Things

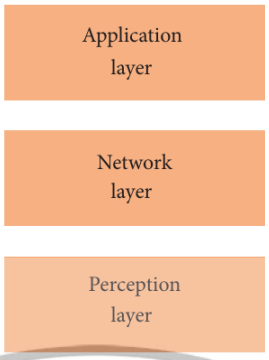
สถาปัตยกรรมของ Internet of Things (IoT) [2] สามารถแบ่งออกเป็นหลายชั้น (Layer) โดยมีโครงสร้างที่นิยมใช้ 2 แบบ คือ โครงสร้างแบบ 3 Layer (Three-layer) ดังรูปที่ 2.1 และโครงสร้างแบบ 5 Layer (Five-layer) ดังรูปที่ 2.2 ซึ่งแต่ละชั้นมีหน้าที่เฉพาะในกระบวนการเชื่อมโยง การประมวลผล และการแสดงผลข้อมูลจากอุปกรณ์ IoT

2.1.1.1 3 Layer (Three-layer)

1) Perception Layer เป็นชั้นที่ทำหน้าที่รับรู้สภาพแวดล้อมและเก็บรวบรวมข้อมูลจากเซนเซอร์และอุปกรณ์ต่าง ๆ ข้อมูลเหล่านี้อาจเป็นข้อมูลทางกายภาพ เช่น อุณหภูมิ ความชื้น หรือแสง และจะถูกส่งต่อไปยัง Network Layer เพื่อทำการประมวลผลต่อไป

2) Network Layer ทำหน้าที่ในการเชื่อมต่อข้อมูลจาก Perception Layer กับเครือข่าย โดยอาศัยเทคโนโลยีการสื่อสาร เช่น Wi-Fi Bluetooth LAN หรือเครือข่ายมือถือ (3G 4G) เพื่อส่งข้อมูลไปยังชั้นถัดไปหรือเซิร์ฟเวอร์สำหรับการประมวลผล

3) Application Layer เป็นชั้นที่ใกล้ชิดกับผู้ใช้ ทำหน้าที่ประมวลผลและแสดงผลข้อมูลในรูปแบบที่ผู้ใช้สามารถเข้าใจได้ เช่น แอปพลิเคชันสำหรับควบคุมการทำงานของอุปกรณ์ IoT ในบ้าน การติดตามสุขภาพ หรือการจัดการพลังงานในเมือง



รูปที่ 2.1 โครงสร้างแบบ 3 Layer (Three-layer) [2]

2.1.1.2 5 Layer (Five-layer)

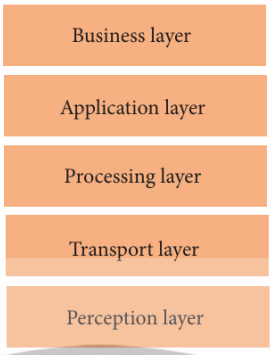
1) Perception Layer ชั้นนี้ยังคงทำหน้าที่เช่นเดียวกับในโครงสร้างแบบ 3 ชั้น คือการเก็บรวบรวมข้อมูลจากเซนเซอร์และอุปกรณ์ก เช่น อุณหภูมิ ความชื้น หรือการเคลื่อนไหวของวัตถุ โดยเป็นจุดเริ่มต้นของการนำข้อมูลเข้าสู่ระบบ IoT

2) Network Layer ทำหน้าที่รับข้อมูลจาก Perception Layer แล้วส่งต่อไปยัง Processing Layer ข้อมูลอาจถูกส่งผ่านเครือข่ายแบบมีสายหรือไร้สาย เช่น IPv6 6LoWPAN Wi-Fi หรือเครือข่ายมือถือ โดยโปรโตคอลที่ใช้ขึ้นอยู่กับลักษณะการเชื่อมต่อ

3) Processing Layer เป็นชั้นที่ทำการประมวลผลข้อมูลที่ได้รับจาก Network Layer อาจใช้วิธีประมวลผลข้อมูลที่อุปกรณ์ข้างเคียง (Edge Computing) หรือส่งไปยังเซิร์ฟเวอร์คลาวด์ (Cloud Computing) โปรโตคอลที่ใช้ในชั้นนี้ได้แก่ MQTT สำหรับการส่งข้อมูลแบบ Publish/Subscribe และ CoAP สำหรับอุปกรณ์ที่มีทรัพยากรจำกัด

4) Application Layer ชั้นนี้จะนำเสนอข้อมูลที่ประมวลผลแล้วให้ผู้ใช้ผ่านแอปพลิเคชัน โดยโปรโตคอลที่ใช้ในชั้นนี้ได้แก่ HTTP/HTTPS สำหรับการรับส่งข้อมูลหรือ WebSocket สำหรับการสื่อสารแบบเรียลไทม์ โปรโตคอล AMQP ใช้สำหรับระบบที่ต้องการความปลอดภัยสูง

5) Business Layer เป็นชั้นที่ทำหน้าที่จัดการข้อมูลเชิงธุรกิจ เช่น การวิเคราะห์ข้อมูลที่ได้จากระบบ IoT เพื่อตัดสินใจเชิงกลยุทธ์ การวิเคราะห์การใช้งานของลูกค้า การจัดการต้นทุน และการควบคุมความปลอดภัยของข้อมูล โดยใช้โปรโตคอลการวิเคราะห์ข้อมูลขนาดใหญ่ เช่น Hadoop หรือ Spark



รูปที่ 2.2 โครงสร้างแบบ 5 Layer (Five-layer) [2]

2.2 MQTT (Message Queuing Telemetry Transport)

MQTT (Message Queuing Telemetry Transport) [3] เป็นโปรโตคอลการสื่อสารที่มีขนาดเล็กและออกแบบมาให้เหมาะสมกับการใช้งานในระบบ IoT ซึ่งมีข้อจำกัดด้านพลังงานและการเชื่อมต่อ นิยมใช้ในการสื่อสารแบบ M2M (Machine to Machine) โดยโปรโตคอลนี้ใช้หลักการทำงานแบบ Publish/Subscribe ซึ่งช่วยเพิ่มประสิทธิภาพในการส่งข้อมูลและประหยัดพลังงานในการสื่อสาร การทำงานของ MQTT สามารถรองรับการเชื่อมต่อสื่อสารแบบสองทาง (Bi-directional communication) และการส่งข้อมูลแบบไม่สูญหายผ่านเครือข่ายต่าง ๆ MQTT จะสามารถทำงานร่วมกับโปรโตคอลเครือข่ายหลากหลายชนิดได้ ซึ่ง MQTT จัดอยู่ใน Application Layer ในโมเดล TCP/IP [4] ซึ่งเป็นชั้นบนสุดที่ทำหน้าที่จัดเตรียมโปรโตคอลและอินเทอร์เฟซสำหรับการแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์ต่าง ๆ ทำให้การสื่อสารระหว่างอุปกรณ์ IoT ให้มีความเสถียรและมีประสิทธิภาพมากยิ่งขึ้น

2.2.1 Publish-Subscribe Model

Publish-Subscribe Model [3] ในโปรโตคอล MQTT เป็นรูปแบบการสื่อสารที่แยกตัวอุปกรณ์ (client) [5] ออกจากกันในการรับรู้การมีอยู่ของอุปกรณ์อื่น ๆ ภายในระบบ กล่าวคือ อุปกรณ์แต่ละตัวไม่จำเป็นต้องรู้จักอุปกรณ์อื่น ๆ ที่อยู่ในระบบเดียวกัน อุปกรณ์แต่ละตัวสามารถสมัครสมาชิก (subscribe) ในหัวข้อที่เกี่ยวข้อง (topic) เพื่อรับข้อความที่เกี่ยวข้องกับหัวข้อนั้น และสามารถส่งข้อความ (publish) ไปยังหัวข้อที่ระบุได้ เพื่อให้ข้อมูลนั้นถูกส่งต่อไปยังอุปกรณ์อื่น ๆ ที่สมัครสมาชิกในหัวข้อเดียวกัน

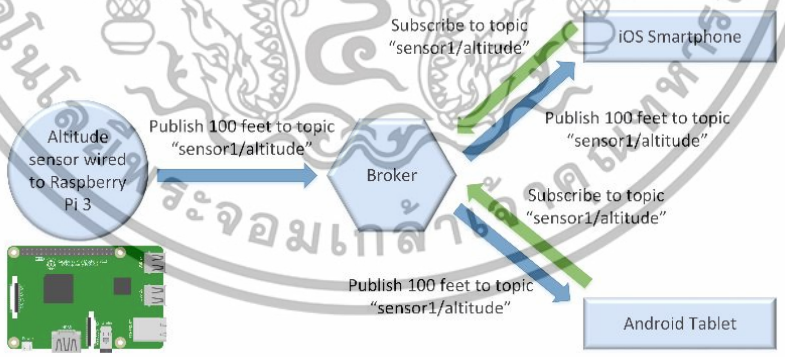
2.2.1.1 การทำงาน Publish-Subscribe

การทำงาน Publish-Subscribe รูปแบบการสื่อสารที่มีการแยกตัว (decoupling) ระหว่างผู้ส่งข้อมูล (Publisher) และผู้รับข้อมูล (Subscriber) โดยทั้งสองฝ่ายไม่จำเป็นต้องรู้ถึงการมีอยู่ของกันและกันโดยตรง แต่จะใช้ Broker หรือ ตัวกลาง ในการส่งผ่านข้อมูล ดังรูปที่ 2.3 ทำให้มีความยืดหยุ่นสูงในการเชื่อมต่ออุปกรณ์หลาย ๆ ตัวในระบบเครือข่าย โดยมีหลักการดังนี้

1) Publisher (ผู้ส่งข้อมูล) เป็นอุปกรณ์หรือโปรแกรมที่ทำหน้าที่ส่งข้อมูลไปยัง Broker โดยข้อมูลที่ส่งจะถูกกำหนดในรูปแบบของหัวข้อ (Topic) เช่น sensor1/altitude โดย Publisher สามารถส่งข้อมูลได้โดยไม่ต้องรู้ว่ามี Subscriber รอรับข้อมูลอยู่หรือไม่

2) Subscriber (ผู้รับข้อมูล) เป็นอุปกรณ์หรือโปรแกรมที่สนใจและสมัครสมาชิกในหัวข้อที่ต้องการรับข้อมูลจาก Broker ตัวอย่างเช่น หาก Subscriber สมัครสมาชิกกับหัวข้อ sensor/# จะได้รับข้อมูลจากทุกหัวข้อที่เกี่ยวข้องค่าที่ sensor เก็บผลได้

3) Broker (ตัวกลาง) ทำหน้าที่เป็นศูนย์กลางในการรับและกระจายข้อมูล โดยจะรับข้อมูลจาก Publisher ที่ส่งเข้ามาตามหัวข้อ และส่งต่อข้อมูลนั้นไปยัง Subscriber ที่สมัครสมาชิกไว้ในหัวข้อที่ตรงกัน โดยที่ Broker ไม่จำเป็นต้องเก็บข้อมูลทั้งหมด แต่อาจเก็บเฉพาะข้อมูลสำคัญหรือข้อมูลที่ต้องการให้ Subscriber ใหม่ได้รับเมื่อสมัครเข้ามาภายหลัง



รูปที่ 2.3 การทำงานของ Publish-Subscribe Model [6]

2.2.1.2 Topics

Topics [7] เป็นส่วนสำคัญที่ใช้ในการจัดระเบียบและแบ่งกลุ่มข้อความที่ถูกส่งออกจากอุปกรณ์ต่าง ๆ โดยเป็นตัวกำหนดว่าข้อความเหล่านั้นอยู่ในหมวดหมู่หรือกลุ่มใด หัวข้อมีลักษณะเป็นสตริงที่ใช้รหัส Unicode Transformation Format 8-bit (UTF-8) ซึ่งทำให้สามารถรองรับตัวอักษรได้หลากหลาย Topic ใน MQTT สามารถสร้างขึ้นเป็นลำดับชั้น (hierarchical) โดยใช้เครื่องหมาย / เป็นตัวแบ่งระหว่างแต่ละระดับของหัวข้อ ดังตารางที่ 2.1

ตารางที่ 2.1 ตัวอย่าง Topic

Topic	**คำอธิบาย**
sensor1/temp	ส่งข้อมูลอุณหภูมิ (temp) จาก sensor1 ผ่าน topic
sensor1/moisture	ส่งข้อมูลความชื้น (moisture) จาก sensor1 ผ่าน topic
Sensor2/angle	ส่งข้อมูลค่ามุม (angle) จาก sensor2 ผ่าน topic

2.2.1.3 Wildcard

Wildcards [8] ใน MQTT คือเครื่องหมายที่ใช้ในการสมัครรับข้อมูล (Subscribe) เพื่อช่วยให้เราสามารถรับข้อมูลจากหลายหัวข้อ (Topic) ได้ในคราวเดียว โดยไม่ต้องระบุหัวข้อทั้งหมดอย่างละเอียด ซึ่งมี 2 รูปแบบ ดังตารางที่ 2.2

ตารางที่ 2.2 ตัวอย่างรูปแบบ Wildcard

รูปแบบ Wildcard	**ลักษณะการทำงาน**	**ตัวอย่าง**
Single-level (+)	ใช้เพื่อแทนที่หัวข้อเฉพาะ 1 ลำดับชั้น	@msg/home/+/light จะรับข้อมูลจาก @msg/home/bathroom /fan
Multi-level (#)	ใช้เพื่อแทนที่หัวข้อหลายลำดับชั้นต่อจากจุดที่กำหนด และครอบคลุมทุกลำดับชั้นที่เหลือ	@msg/home/# จะรับข้อมูลจาก @msg/home@msg/home/bethroom /fan

2.2.2 โครงสร้างแพ็กเกจของ MQTT

แพ็กเกจของโปรโตคอล MQTT [3] นั้นถูกแบ่งออกเป็น 3 ส่วน ได้แก่ Fixed Header Variable Header และ Payload ดังรูปที่ 2.4 โดยแต่ละส่วนมีหน้าที่แตกต่างกันและประกอบไปด้วยข้อมูลต่าง ๆ ที่จำเป็นต่อการสื่อสารและการทำงานของโปรโตคอล

2.2.2.1 Fixed Header

Fixed Header เป็นส่วนที่กำหนดประเภทของแพ็กเกจและจัดเก็บข้อมูลพื้นฐานที่จำเป็นในการสื่อสาร เช่น ประเภทของข้อความและสถานะของการส่งข้อมูล

- 4 Byte แรกจะแสดงถึงประเภทของแพ็กเกจ เช่น CONNECT PUBLISH หรือ SUBSCRIBE ซึ่งจะกำหนดว่าข้อมูลในแพ็กเกจนี้จะถูกนำไปใช้งานอย่างไร
- 1 Byte ถัดมาจะเป็น flag ที่ใช้ระบุว่าแพ็กเกจนี้เป็นการส่งซ้ำหรือไม่ (Duplicate flag) ซึ่งจะถูกใช้งานในกรณีที่ต้องการส่งข้อมูลซ้ำ เช่น ในกรณีของการเชื่อมต่อล้มเหลว
- 2 Byte ต่อมาใช้สำหรับกำหนดระดับ QoS (Quality of Service) ที่บอกถึงระดับความปลอดภัยและการส่งข้อมูลว่าจะต้องถูกส่งมากกว่าหนึ่งครั้งหรือไม่
- 1 Byte สุดท้าย เป็น flag RETAIN ซึ่งใช้ในการระบุว่าข้อมูลนี้ต้องการให้ Broker เก็บไว้สำหรับการใช้ในอนาคตหรือไม่

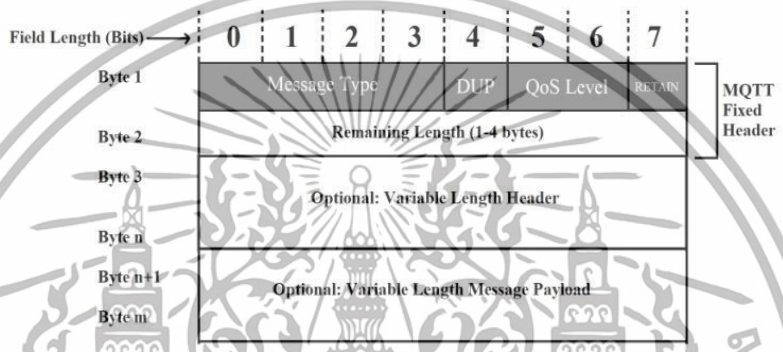
2.2.2.2 Variable Header

Variable Header เป็นส่วนที่มีข้อมูลเพิ่มเติมตามประเภทของแพ็กเกจที่ถูกส่ง เช่น การบอกว่าแพ็กเกจนี้มีข้อมูล Username และ Password อยู่ใน Payload หรือไม่ ส่วนแพ็กเกจ PUBLISH จะมีข้อมูลที่แตกต่างออกไป ซึ่งจะประกอบไปด้วย

- ชื่อของหัวข้อ (Topic Name) เป็นข้อมูลที่บอกว่าแพ็กเกจนี้เกี่ยวข้องกับหัวข้อใดใน Broker ที่จะถูกใช้ในการสื่อสาร
- Most and Least Significant Bytes ใช้ในการระบุความยาวของสตริงชื่อหัวข้อ
- Message ID สำหรับการยืนยันหรือควบคุมการส่งซ้ำของแพ็กเกจ

2.2.2.3 Payload

ส่วน Payload เป็นส่วนที่บรรจุข้อมูลหลักที่ต้องการส่งไปยัง Broker หรือข้อมูลที่ถูกส่งไปให้กับอุปกรณ์อื่นๆ ซึ่งในแพ็กเกจ PUBLISH Payload จะประกอบไปด้วยข้อมูลที่ต้องการเผยแพร่ ส่วนในแพ็กเกจ SUBSCRIBE จะประกอบไปด้วยชื่อหัวข้อที่ต้องการติดตามจาก Broker โดยที่ Payload จะมีความยาวแตกต่างกันไปตามประเภทของแพ็กเกจและข้อมูลที่ต้องการส่ง ข้อมูลนี้จะถูกจัดการตามการตั้งค่าของ QoS และ RETAIN ที่ถูกกำหนดไว้ใน Fixed Header



รูปที่ 2.4 โครงสร้างแพ็กเกจของ MQTT [9]

2.2.3 QoS

QoS (Quality of Service) [10] เป็นระดับคุณภาพในการส่งข้อความในโปรโตคอล MQTT ที่กำหนดความน่าเชื่อถือในการส่งและรับข้อมูลระหว่างผู้ส่ง (Publisher) และผู้รับ (Subscriber) ผ่านทาง Broker ดังรูปที่ 2.5 ในการส่งข้อมูล MQTT มี 3 ระดับ ดังนี้

2.2.3.1 QoS 0

การส่งข้อมูลในระดับนี้เป็นการส่งแบบไม่รับประกันว่าข้อมูลจะถูกส่งหรือได้รับสำเร็จ หากเกิดปัญหาระหว่างการส่ง เช่น การสูญเสียสัญญาณ ข้อมูลอาจจะไม่ถูกส่งถึงผู้รับได้เลยไม่มีการยืนยันจากฝั่งผู้รับ (Broker หรือ Subscriber) เมื่อรับข้อมูลเหมาะสำหรับการส่งข้อมูลที่ไม่จำเป็นต้องการรับประกันการส่ง เช่น การส่งข้อมูลเซนเซอร์ที่ส่งบ่อย ๆ

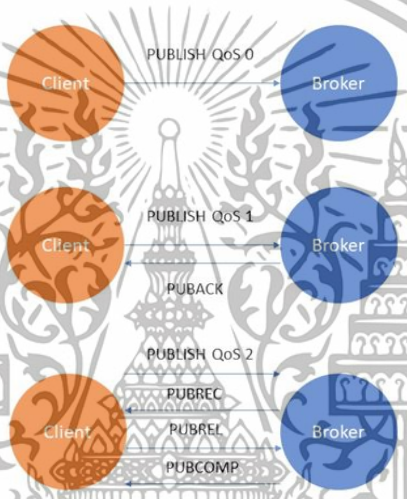
2.2.3.2 QoS 1

ข้อมูลจะถูกส่งให้ถึงผู้รับอย่างน้อย 1 ครั้ง แต่มีโอกาสที่จะได้รับข้อมูลซ้ำ เมื่อผู้รับได้รับข้อมูลจะส่งสัญญาณยืนยันกลับมาว่ารับข้อมูลแล้ว แต่หากผู้ส่งไม่ได้รับการยืนยันผู้ส่งจะทำการส่งซ้ำจนกว่าจะได้รับการยืนยันเหมาะสำหรับกรณีที่ข้อมูลต้องถูกส่งให้ถึงปลายทาง

แต่ไม่จำเป็นต้องหลีกเลี่ยงการซ้ำกัน เช่น ข้อมูลการแจ้งเตือน การแจ้งเตือนที่ต้องการให้ส่งไปถึงผู้รับทุกครั้ง ไม่ว่าสัญญาณจะขาดหายหรือไม่

2.2.3.3 QoS 2

เป็นระดับสูงสุดและมีความน่าเชื่อถือที่สุด ข้อมูลจะถูกส่งและรับเพียง 1 ครั้งเท่านั้น ไม่มีโอกาสที่จะเกิดการส่งซ้ำหรือขาดหาย ใช้ขั้นตอนการตรวจสอบเพิ่มเติมเพื่อให้แน่ใจว่าข้อมูลถูกส่งและรับเพียงครั้งเดียว ซึ่งอาจทำให้เกิดความหน่วงเล็กน้อย เหมาะสำหรับกรณีที่ข้อมูลต้องมีความสำคัญสูงและไม่สามารถมีข้อมูลซ้ำได้ เช่น การทำธุรกรรมทางการเงิน



รูปที่ 2.5 QoS level ใน MQTT [2]

2.2.4 กระบวนการเชื่อมต่อระหว่าง Client และ Broker (Server)

การเชื่อมต่อระหว่าง Client และ Broker ใน MQTT [3] มีขั้นตอนที่ต้องดำเนินการเพื่อให้การสื่อสารระหว่างกันเกิดขึ้นได้ ซึ่งประกอบด้วยการแลกเปลี่ยนข้อมูลต่าง ๆ เพื่อยืนยันตัวตน การตรวจสอบ และการจัดการการเชื่อมต่อ ดังรูปที่ 2.6 ข้อมูลที่ใช้ในการสื่อสารเหล่านี้จะถูกบรรจุในรูปแบบที่เรียกว่า Control Packet ซึ่งแบ่งออกเป็นประเภทต่าง ๆ ขึ้นอยู่กับวัตถุประสงค์ของการใช้งาน โดยมีขั้นตอนการเชื่อมต่อและประเภทของ Control Packet ที่ใช้ในแต่ละขั้นตอนดังนี้

1) การเชื่อมต่อ TCP

ก่อนการเชื่อมต่อในโปรโตคอล MQTT จะเกิดขึ้น อุปกรณ์ Client จะต้องเริ่มต้นการเชื่อมต่อกับ Broker ผ่านโปรโตคอล TCP โดยกระบวนการนี้จะเปิดช่องทางการสื่อสารระหว่างทั้งสองฝ่ายเพื่อให้สามารถส่งและรับข้อมูล MQTT ได้

2) การส่ง CONNECT Packet

เมื่อ Client เริ่มต้นการเชื่อมต่อกับ Broker ผ่าน TCP ได้สำเร็จ Client จะส่ง CONNECT Packet ไปยัง Broker เพื่อเริ่มต้นเซสชันการเชื่อมต่อ ภายใน CONNECT Packet จะมีข้อมูลที่จำเป็นสำหรับการยืนยันตัวตนและการตั้งค่าต่าง ๆ ดังนี้

- ClientId รหัสระบุตัวตนเฉพาะของ Client ที่ต้องใช้ในการสื่อสารกับ Broker ทุกครั้ง
- CleanSession ฟิลด์ boolean ที่ใช้กำหนดว่าข้อมูลการสมัครสมาชิก (subscription) และสถานะของการเชื่อมต่อควรถูกเก็บไว้หรือไม่
- UserName และ Password ใช้สำหรับการยืนยันตัวตน Client หากต้องการระบุข้อมูลการยืนยันตัวตน Broker จะทำการตรวจสอบข้อมูลเหล่านี้ก่อนการเชื่อมต่อ
- ProtocolLevel ระบุเวอร์ชันของโปรโตคอล MQTT ที่ใช้ในการเชื่อมต่อ
- KeepAlive ระบุช่วงเวลา (s) ที่ Client ต้องส่ง Control Packet ไปยัง Broker หากไม่มีข้อมูลให้ส่งเพื่อรักษาการเชื่อมต่อให้ยังคงอยู่

3) การตอบกลับของ Broker (CONNACK)

เมื่อ Broker ได้รับ CONNECT Packet จะทำการตรวจสอบข้อมูลที่ได้รับจาก Client โดยจะตรวจสอบตัวตนและความถูกต้องของข้อมูล หากการตรวจสอบผ่าน Broker จะตอบกลับด้วย CONNACK Packet ซึ่งเป็นการยืนยันว่าการเชื่อมต่อสำเร็จและการสื่อสารระหว่าง Client และ Broker สามารถเริ่มต้นได้ มีพารามิเตอร์ในการตอบกลับของ Broker (CONNACK) ดังนี้

- SessionPresent ระบุสถานะของเซสชันที่ยังคงอยู่ ถ้า CleanSession ของ Client เป็น 1 ฟิลด์นี้จะมีค่าเป็น 0 ถ้า CleanSession เป็น 0

และมีเซสชันเก่าคงอยู่ ฟิลด์นี้จะมีค่าเป็น 1 และถ้า CleanSession เป็น 0 และมีเซสชันเก่าคงอยู่ ฟิลด์นี้จะมีค่าเป็น 1

- ReturnCode แสดงสถานะของการเชื่อมต่อ ถ้าการเชื่อมต่อสำเร็จ ReturnCode จะเป็น 0 หากการเชื่อมต่อล้มเหลว จะมีค่าเป็น 1-5 เพื่อระบุข้อผิดพลาด

4) การสมัครสมาชิก (SUBSCRIBE)

หลังจากที่การเชื่อมต่อได้รับการยืนยัน Client สามารถเริ่มกระบวนการสมัครสมาชิกหัวข้อ (topic) ที่สนใจ โดยการส่ง SUBSCRIBE Packet ไปยัง Broker ภายใน SUBSCRIBE Packet จะระบุหัวข้อที่ต้องการสมัครรับข้อมูล

5) การยืนยันการสมัครสมาชิก (SUBACK)

เมื่อ Broker ได้รับคำขอสมัครสมาชิกจาก Client จะตอบกลับด้วย SUBACK Packet ซึ่งเป็นการยืนยันว่าการสมัครสมาชิกสำเร็จแล้ว Client จะสามารถเริ่มรับข้อมูลจากหัวข้อที่สมัครไว้ผ่าน Broker ได้ทันที

6) การแลกเปลี่ยนข้อมูล (PUBLISH)

เมื่อการสมัครสมาชิกสำเร็จแล้ว Client จะสามารถส่งและรับข้อมูลผ่านหัวข้อที่กำหนดไว้ โดยข้อมูลจะถูกแลกเปลี่ยนกันในรูปแบบของ PUBLISH Packet ซึ่งจะถูกส่งไปยัง Broker และเผยแพร่ไปยัง Client อื่น ๆ ที่สมัครรับข้อมูลในหัวข้อเดียวกัน

7) การยกเลิกการสมัครสมาชิก (UNSUBSCRIBE)

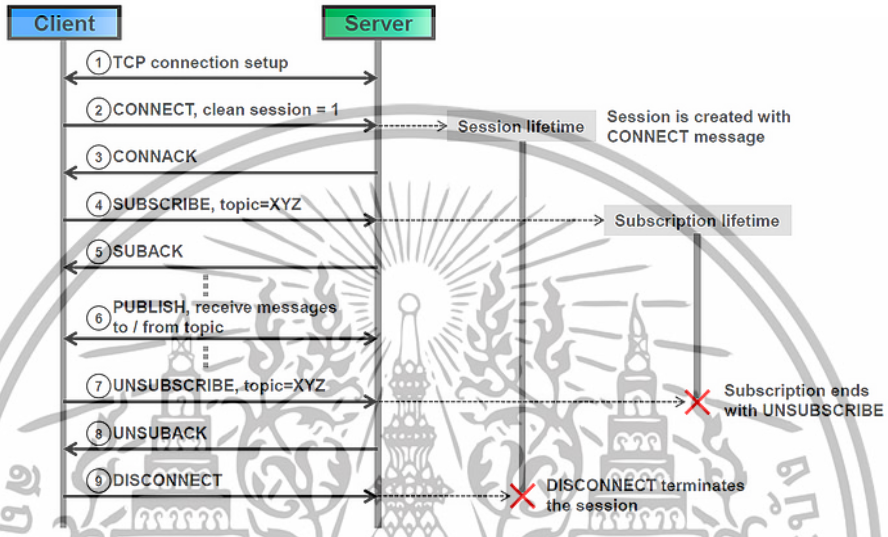
เมื่อ Client ต้องการหยุดรับข้อมูลจากหัวข้อที่เคยสมัครไว้ จะส่ง UNSUBSCRIBE Packet ไปยัง Broker เพื่อยกเลิกการสมัครหัวข้อ ระบบ MQTT อนุญาตให้ Client สามารถสมัครและยกเลิกการสมัครสมาชิกหัวข้อได้ตลอดเวลาในระหว่างเซสชันการเชื่อมต่อ

8) การยืนยันการยกเลิกการสมัครสมาชิก (UNSUBACK)

หลังจาก Broker ได้รับคำขอยกเลิกการสมัครสมาชิกจาก Client จะตอบกลับด้วย UNSUBACK Packet เพื่อยืนยันว่าการยกเลิกสำเร็จแล้ว Client จะไม่ได้รับข้อมูลจากหัวข้อที่ยกเลิกอีกต่อไป

9) การยกเลิกการเชื่อมต่อ (DISCONNECT)

เมื่อ Client ต้องการยกเลิกการเชื่อมต่อกับ Broker อย่างปลอดภัย Client จะส่ง DISCONNECT Packet ไปยัง Broker เพื่อแจ้งให้ Broker ยกเลิกการเชื่อมต่อ หลังจากที่ได้รับ DISCONNECT Packet การเชื่อมต่อจะถูกปิด และเซสชันจะสิ้นสุดลง



รูปที่ 2.6 กระบวนการเชื่อมต่อระหว่าง Client และ Broker (Server) [11]

2.2.4.1 HiveMQ

HiveMQ [12] เป็นแพลตฟอร์มที่ออกแบบมาเพื่อรองรับโปรโตคอล MQTT ซึ่งเป็นมาตรฐานสำหรับการแลกเปลี่ยนข้อมูลในระบบ IoT โดยเฉพาะ HiveMQ จะช่วยให้การส่งข้อมูลระหว่างอุปกรณ์ IoT ได้อย่างรวดเร็ว มีประสิทธิภาพ และปลอดภัย

1) HiveMQ Cloud [13] เป็นแพลตฟอร์มคลาวด์ที่ช่วยจัดการการส่งข้อมูลระหว่างอุปกรณ์ IoT ได้อย่างรวดเร็วและปลอดภัย โดยรองรับโปรโตคอลมาตรฐาน MQTT ที่ถูกออกแบบมาสำหรับการสื่อสารในระบบ IoT ทำให้ผู้ใช้สามารถสร้าง Cluster สำหรับเชื่อมต่ออุปกรณ์ IoT ได้อย่างรวดเร็วโดยไม่ต้องติดตั้งหรือดูแลเซิร์ฟเวอร์เอง อีกทั้งยังสามารถปรับขนาดระบบได้อัตโนมัติเพื่อรองรับการใช้งานที่เพิ่มขึ้น โดยที่ HiveMQ Cloud มีระบบรักษาความปลอดภัยระดับองค์กรที่รวมถึงการเข้ารหัส TLS และการยืนยันตัวตนของอุปกรณ์ ทำให้มีความปลอดภัยในการใช้งานในระบบ IoT ขนาดใหญ่

2) HiveMQ Web Client [14] เป็นเครื่องมือที่ช่วยให้ผู้ใช้สามารถเชื่อมต่อและจัดการกับอุปกรณ์ IoT ผ่านโปรโตคอล MQTT ได้โดยตรงจากเว็บเบราว์เซอร์ โดยไม่ต้องติดตั้งซอฟต์แวร์เพิ่มเติม โดยสามารถทดสอบการเชื่อมต่อของการส่งและรับข้อมูลจากอุปกรณ์ IoT แบบเรียลไทม์ผ่านอินเทอร์เฟซของ HiveMQ Web Client จึงเหมาะสำหรับการทดสอบและการดีบักระบบ IoT ช่วยลดความซับซ้อนในการตรวจสอบการทำงานของระบบ และสามารถเชื่อมต่อกับ HiveMQ Cloud หรือเซิร์ฟเวอร์ MQTT อื่น ๆ ได้

2.2.4.2 Adafruit

Adafruit IO MQTT Broker [15] เป็นบริการฟรีที่ Adafruit ให้ไว้สำหรับผู้ใช้งานแพลตฟอร์ม Adafruit IO เพื่อช่วยในการเชื่อมต่ออุปกรณ์ IoT ผ่านโปรโตคอล MQTT Broker นี้ทำหน้าที่เป็นศูนย์กลางรับส่งข้อมูลระหว่างอุปกรณ์และ Adafruit IO ทำให้ผู้ใช้สามารถส่งข้อมูลจากอุปกรณ์ไปยัง Adafruit IO เพื่อจัดเก็บ ประมวลผล หรือแสดงผลได้ง่าย นอกจากนี้ยังสามารถรับข้อมูลจาก Adafruit IO ไปควบคุมอุปกรณ์ได้อีกด้วย จุดเด่นของ Adafruit IO MQTT Broker คือ ใช้งานง่าย ไม่ต้องตั้งค่า Server เอง มี Dashboard สำหรับจัดการข้อมูล และมี Library ที่รองรับการใช้งานกับบอร์ดไมโครคอนโทรลเลอร์หลากหลายประเภท อย่างไรก็ตาม Adafruit IO MQTT Broker ก็มีข้อจำกัดบางประการ เช่น ปริมาณข้อมูลที่สามารถรับส่งได้ฟรีมีจำกัด และอาจมี Latency สูงกว่า Broker ที่มีการจัดการเอง

2.3 บอร์ดไมโครคอนโทรลเลอร์ ESP32

ESP32 [16] เป็นไมโครคอนโทรลเลอร์ที่พัฒนาโดยบริษัท Espressif Systems ที่รวมเอาความสามารถด้านการสื่อสารไร้สายทั้ง Wi-Fi และ Bluetooth เข้ามาในชิปเดียว โดยรองรับทั้ง Bluetooth Classic และ Bluetooth Low Energy (BLE) ทำให้ ESP32 เป็นที่นิยมอย่างมากในการพัฒนาอุปกรณ์ IoT (Internet of Things) เนื่องจากมีประสิทธิภาพสูงและการจัดการพลังงานที่ดี นอกจากนี้ ESP32 ยังมาพร้อมกับซีพียูแบบ Xtensa® dual-core หรือ single-core LX6 ซึ่งสามารถทำงานได้ที่ความเร็วสูงสุด 240 MHz และหน่วยความจำในตัวที่หลากหลาย ด้วยความสามารถในการสื่อสารที่ครอบคลุมและความหลากหลายในการใช้งาน ESP32 สามารถนำไปประยุกต์ใช้ได้ในหลาย ๆ ด้าน เช่น การควบคุม Smart Home ระบบตรวจสอบระยะไกล และการประมวลผลแบบ Real time ทำให้เหมาะสมสำหรับการพัฒนาโซลูชัน IoT ที่ต้องการ

เชื่อมต่ออุปกรณ์ผ่านอินเทอร์เน็ตหรือเครือข่ายไร้สาย โดยสามารอัปโหลดโปรแกรมไปยังบอร์ด ESP32 และเชื่อมต่อวงจรเพื่อแสดงการทำงานของโปรแกรม

2.3.1 โปรแกรมการสั่งงาน

2.3.1.1 Arduino IDE

Arduino IDE (Integrated Development Environment) [17] เป็นซอฟต์แวร์ Open source ที่ใช้สำหรับการพัฒนาโปรแกรมและอัปโหลดโปรแกรมลงในบอร์ดไมโครคอนโทรลเลอร์ทำให้ผู้ใช้สามารถเขียนโค้ดในภาษา C และ C++ รวมถึงการควบคุมการทำงานของฮาร์ดแวร์ผ่านไมโครคอนโทรลเลอร์ได้ง่าย ๆ

ฟังก์ชันหลักของ Arduino IDE

1) ตัวแก้ไขโค้ด (Code Editor) ใช้สำหรับเขียนโค้ดโปรแกรมที่ควบคุมบอร์ด Arduino รองรับภาษา C และ C++ พร้อมฟังก์ชันไลบรารีเพิ่มเติมเพื่อการทำงานที่สะดวกขึ้น

2) คอมไพเลอร์ (Compiler) ทำหน้าที่แปลงโค้ดที่เขียนในภาษา C/C++ ให้เป็นภาษาเครื่องที่บอร์ด Arduino สามารถเข้าใจและประมวลผลได้

3) การอัปโหลดโค้ด (Uploader) ใช้ในการอัปโหลดโค้ดที่เขียนไปยังบอร์ด Arduino ผ่านพอร์ต USB

โครงสร้างพื้นฐานของภาษา C ที่ใช้กับ Arduino IDE

1) Header ส่วนนี้จะมีหรือไม่มีก็ได้ ขึ้นอยู่กับความจำเป็นของโปรแกรม ส่วนของ Header จะอยู่ในช่วงเริ่มต้นของโปรแกรมและอาจประกอบไปด้วย

2) ฟังก์ชัน setup() เป็นฟังก์ชันบังคับที่ต้องมีในทุกโปรแกรมของ Arduino แม้ว่าบางโปรแกรมจะไม่ต้องการทำงานในฟังก์ชันนี้ก็ตาม ฟังก์ชัน setup() จะทำงานเพียงครั้งเดียวตอนเริ่มต้นการทำงานของโปรแกรม โดยจะใช้สำหรับการตั้งค่าเริ่มต้นต่างๆ เช่น การกำหนดโหมดการทำงานของพิน (pinMode) หรือการตั้งค่าการสื่อสารผ่าน Serial

3) ฟังก์ชัน loop() เป็นฟังก์ชันที่ทำหน้าที่ในการประมวลผลคำสั่งซ้ำ ๆ อย่างต่อเนื่องตลอดเวลาที่โปรแกรมทำงาน ซึ่งเป็นส่วนหลักของการควบคุมการทำงานของโปรแกรม

2.3.2 ESP32 DevKit v1

ESP32 DevKit v1 [18] เป็นบอร์ดพัฒนาที่ใช้ไมโครคอนโทรลเลอร์ ESP32 จากบริษัท Espressif ซึ่งออกแบบมาเพื่อตอบโจทย์การพัฒนาโพรเจกต์ด้าน Internet of Things (IoT) บอร์ดนี้ได้รับความนิยมอย่างมาก มีขาเชื่อมต่อ 30 ขา เหมาะสำหรับการพัฒนา IoT เนื่องจากมีฟังก์ชันครบครันทั้ง Wi-Fi และ Bluetooth ภายในโมดูลเดียว ขนาดกะทัดรัดและสามารถใช้งานได้ง่ายผ่านการเชื่อมต่อกับพอร์ต GPIO ต่าง ๆ โดยมีคุณลักษณะของบอร์ด ESP32 DevKit v1 ดังตารางที่ 2.3

ตารางที่ 2.3 คุณลักษณะของบอร์ด ESP32 DevKit v1

คุณลักษณะ	รายละเอียด
ขนาดบอร์ด	51.5 x 29 x 5 มม.
จำนวนขา (Pins)	30 ขา
ชิปหลัก (SoC)	ESP32-D0WD
ความถี่ซีพียู	สูงสุด 240 MHz
หน่วยความจำแฟลช (Flash)	4MB (external flash)
หน่วยความจำ SRAM	520 KB
Wi-Fi	802.11 b/g/n (2.4 GHz) ความเร็วสูงสุด 150 Mbps
Bluetooth	Bluetooth v4.2 (BR/EDR + BLE)
การจัดการพลังงาน	รองรับโหมด Active Modem-sleep Light-sleep Deep-sleep และ Hibernation
แรงดันไฟฟ้าทำงาน	3.0V – 3.6V
การเชื่อมต่อไฟฟ้า (IO)	3.3V TTL

2.3.2.1 การทำงานของขา

ESP32 DevKit v1 มีขาพินทั้งหมด 30 ขา ดังรูปที่ 2.7 โดยแต่ละขาสามารถใช้ได้หลากหลายฟังก์ชัน เช่น GPIO (General Purpose Input/Output) ADC (Analog-to-Digital Converter) UART (Universal Asynchronous Receiver-Transmitter) PWM (Pulse Width Modulation) และอื่น ๆ การแบ่งขาของ ESP32-WROOM-32 (30 Pin) ตามประเภทการทำงานสามารถแบ่งได้ตามการใช้งานหลักของขา ดังนี้

1) ขาจ่ายไฟ มีหน้าที่ในการจ่ายพลังงานให้กับโมดูลและส่วนประกอบภายใน ซึ่งสำคัญต่อการทำงานของอุปกรณ์ ได้แก่ ขา 3V3 และขา GND

2) ขา GPIO (General Purpose Input/Output Pins) ขาอเนกประสงค์ที่สามารถตั้งค่าให้ทำงานได้ทั้งในโหมดการรับข้อมูล (Input) หรือส่งข้อมูล (Output) ขาเหล่านี้มีความสำคัญอย่างยิ่งในการเชื่อมต่อ ESP32 กับอุปกรณ์ภายนอก ได้แก่ GPIO23 GPIO22 GPIO21 GPIO19 GPIO18 GPIO17 GPIO16 GPIO4 GPIO0 GPIO2 GPIO5 GPIO13 GPIO12 GPIO14 GPIO15 GPIO25 GPIO26 GPIO27 GPIO32 GPIO33 GPIO34 GPIO35 GPIO36 และ GPIO39

3) ขา UART (Universal Asynchronous Receiver-Transmitter Pins) เป็นขาที่ใช้สำหรับการสื่อสารแบบอนุกรม (Serial Communication) ซึ่งเป็นหนึ่งในวิธีที่นิยมใช้ในการส่งและรับข้อมูลระหว่าง ESP32 กับอุปกรณ์ภายนอก ได้แก่ TXD0 RXD0 TXD2

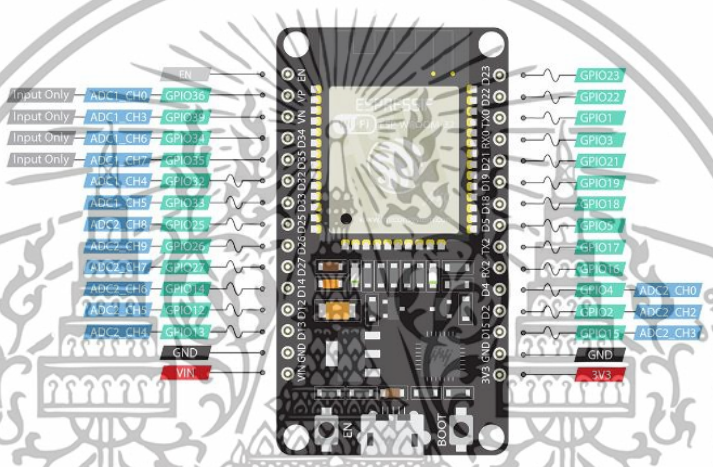
4) ขา ADC (Analog-to-Digital Converter Pins) ใช้สำหรับการแปลงสัญญาณอนาล็อก (Analog Signal) เป็นสัญญาณดิจิทัล (Digital Signal) ซึ่งทำให้สามารถอ่านค่าจากเซนเซอร์ที่ให้สัญญาณในรูปแบบอนาล็อก ได้แก่ GPIO32 GPIO33 GPIO34 GPIO35 GPIO36 และ GPIO39

5) ขา DAC (Digital-to-Analog Converter Pins) ใช้สำหรับการแปลงสัญญาณดิจิทัล (Digital Signal) เป็นสัญญาณอนาล็อก (Analog Signal) ทำให้สามารถควบคุมอุปกรณ์ที่ต้องการสัญญาณอนาล็อก ได้แก่ GPIO25 และ GPIO26

6) ขา I2C รองรับการใช้งาน I2C โดยไม่มีขาที่กำหนดเฉพาะ แต่สามารถตั้งค่าให้ GPIO ใด ๆ เป็น SDA และ SCL ได้แก่ GPIO21 และ GPIO22

7) ขา SPI มี SPI จำนวน 3 ชุด (SPI HSPI VSPI) โดยสามารถใช้งานได้ทั้งในโหมด Slave และ Master ได้แก่ MOSI (Master Out Slave In) ที่ขา GPIO23 MISO (Master In Slave Out) ที่ขา GPIO19 และ SCK (Serial Clock) ที่ขา GPIO18

8) ขา Capacitive Touch ซึ่งสามารถตรวจจับการเปลี่ยนแปลงของความจุไฟฟ้าที่เกิดจากการสัมผัส โดยสามารถนำไปประยุกต์ใช้งานในการทำแป้นสัมผัส (Touch Pad) หรือเปิด-ปิดอุปกรณ์ด้วยการแตะเบา ๆ ได้แก่ GPIO4 GPIO0 GPIO2 GPIO15 GPIO13 GPIO12 GPIO14 GPIO27 GPIO33 และ GPIO32



รูปที่ 2.7 รายละเอียดขา ESP32 DevKit v1 [19]

2.3.3 Flash Memory

ESP32 DevKit v1 รุ่น 30 ขา มี Flash Memory [20] (หน่วยความจำแฟลช) ที่ติดตั้งอยู่ในตัว โดยหน่วยความจำแฟลชนี้เป็น SPI Flash ขนาด 4 MB ซึ่งใช้สำหรับเก็บ Firmware โปรแกรม และข้อมูลอื่น ๆ ที่จำเป็นในการทำงานของไมโครคอนโทรลเลอร์ หน่วยความจำแฟลชนี้ทำงานร่วมกับหน่วยประมวลผลและส่วนต่าง ๆ ของ ESP32 เพื่อให้สามารถบันทึกและอ่านข้อมูลได้อย่างมีประสิทธิภาพ มีคุณลักษณะดังตารางที่ 2.4

ตารางที่ 2.4 คุณลักษณะ Flash Memory

คุณลักษณะ	รายละเอียด
ขนาดหน่วยความจำแฟลช	4 MB (32 Mbit)
ประเภทหน่วยความจำ	SPI Flash (Serial Peripheral Interface)
การเชื่อมต่อ	ใช้ GPIO6 ถึง GPIO11
การใช้งานหลัก	- เก็บโปรแกรมหลักและเฟิร์มแวร์ของ ESP32 - เก็บข้อมูลผู้ใช้ (User Data)

2.3.4 EEPROM

EEPROM (Electrically Erasable Programmable Read-Only Memory) [20] คือ หน่วยความจำที่สามารถเก็บข้อมูลในระยะยาว โดยไม่ต้องใช้ไฟฟ้าเพื่อรักษาข้อมูล (non-volatile) ใน ESP32 มีการจำลอง EEPROM ด้วยการใช้พื้นที่ใน Flash Memory ซึ่งช่วยให้สามารถเก็บข้อมูลในลักษณะของ key-value เช่น ค่าสถานะการเชื่อมต่อ Wi-Fi (SSID และ Password) โดยสามารถอ่านและเขียนข้อมูลได้ มีรายละเอียด ดังนี้

- Non-Volatile โดนที่ EEPROM สามารถเก็บข้อมูลได้โดยไม่ต้องมีไฟจ่ายเข้า เมื่ออุปกรณ์ปิดไฟ ข้อมูลที่ถูกจัดเก็บอยู่ใน EEPROM จะยังคงอยู่
- ข้อมูลใน EEPROM สามารถเขียนทับได้หลายครั้ง แต่มีข้อจำกัดในการเขียนซ้ำประมาณ 100,000 ครั้งต่อเซลล์
- มีโครงสร้างแบบ key-value จึงสามารถเก็บข้อมูลในรูปแบบของ key-value ทำให้การเข้าถึงข้อมูลทำได้ง่าย
- มีพื้นที่จัดเก็บ 512 KB (0 - 255) หรือ 128 IP addresses

2.3.4.1 ฟังก์ชันการทำงานหลักของ EEPROM

1) การเขียนข้อมูล ฟังก์ชัน EEPROM.write(address, value) ใช้ในการเขียนข้อมูลไปยังตำแหน่งที่กำหนดใน EEPROM โดยที่

- address เป็นตำแหน่งใน EEPROM ที่ต้องการเขียนข้อมูล (ตั้งแต่ 0 ถึงขนาดสูงสุดของ EEPROM)Most and Least Significant Bytes ใช้ในการระบุความยาวของสตริงชื่อหัวข้อ
- value คือข้อมูลที่ต้องการเขียน (0-255 byte)

2) การอ่านข้อมูล ฟังก์ชัน EEPROM.read(address) ใช้ในการอ่านข้อมูลจากตำแหน่งที่กำหนดใน EEPROM โดยที่

- address เป็นตำแหน่งใน EEPROM ที่ต้องการอ่านข้อมูล ฟังก์ชันนี้จะคืนค่าข้อมูลที่อยู่ในตำแหน่งที่กำหนดในรูปแบบ byte (0-255)

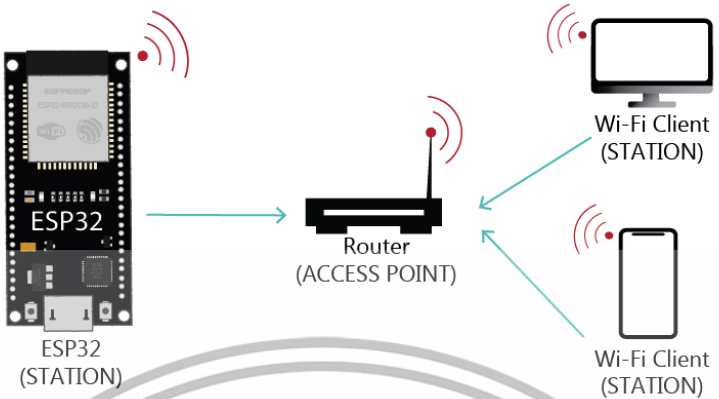
3) การบันทึกการเปลี่ยนแปลง หลังจากที่มีการเขียนข้อมูลลงใน EEPROM ฟังก์ชัน EEPROM.commit() จะถูกเรียกใช้เพื่อบันทึกการเปลี่ยนแปลงอย่างถาวร

2.3.5 การเชื่อมต่อผ่าน Wi-Fi

ESP32 [21] เป็นไมโครคอนโทรลเลอร์ที่มีฟีเจอร์การเชื่อมต่อ Wi-Fi 802.11b/g/n รองรับการเชื่อมต่อความถี่ 2.4 GHz พร้อมความสามารถในการรับส่งข้อมูลด้วยความเร็วสูงสุดถึง 150 Mbps ซึ่งเหมาะสำหรับการใช้งานที่ต้องการการเชื่อมต่ออินเทอร์เน็ตแบบไร้สายในอุปกรณ์ IoT

2.3.5.1 โหมดการทำงาน Station Mode

การทำงานโหมดนี้ทำให้ ESP32 เชื่อมต่อกับเครือข่าย Wi-Fi อื่น ๆ เช่น Router เพื่อเข้าถึงอินเทอร์เน็ตหรือเครือข่ายภายใน โดยเมื่อ ESP32 เชื่อมต่อกับ Router จะได้รับ IP Address ที่จัดสรรโดย DHCP (Dynamic Host Configuration Protocol) จาก Router นั้น โหมดนี้จะใช้งานในกรณีที่ต้องการให้ ESP32 เชื่อมต่อกับเครือข่าย Wi-Fi ที่มีอยู่ เช่น การเข้าถึงข้อมูลจาก API ส่งข้อมูลไปยังเซิร์ฟเวอร์ หรือควบคุมอุปกรณ์อื่น ๆ ที่เชื่อมต่ออยู่ในเครือข่ายเดียวกันจึงเหมาะสำหรับโพรเจกต์ IoT ที่ต้องการส่งข้อมูลจาก ESP32 ไปยังเซิร์ฟเวอร์หรือรับข้อมูลจากอินเทอร์เน็ต

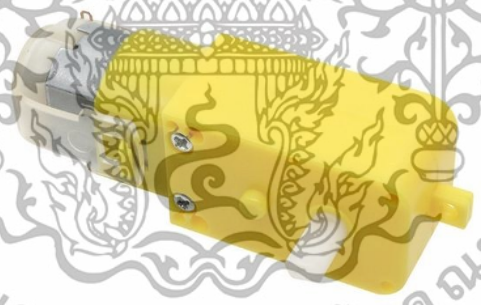


รูปที่ 2.8 การทำงาน Station Mode [21]

2.4 ฮาร์ดแวร์ส่วนรถบังคับ

2.4.1 DC Gear Motor

DC Gear Motor [22] ดังรูปที่ 2.9 เป็นมอเตอร์ที่มีแรงบิดสูง น้ำหนักเบา และ RPM ต่ำ ทำให้เหมาะสำหรับการใช้งานในงานที่ต้องการแรงขับเคลื่อนสูงและความแม่นยำในการควบคุมการเคลื่อนที่ มีคุณลักษณะดังตารางที่ 2.5



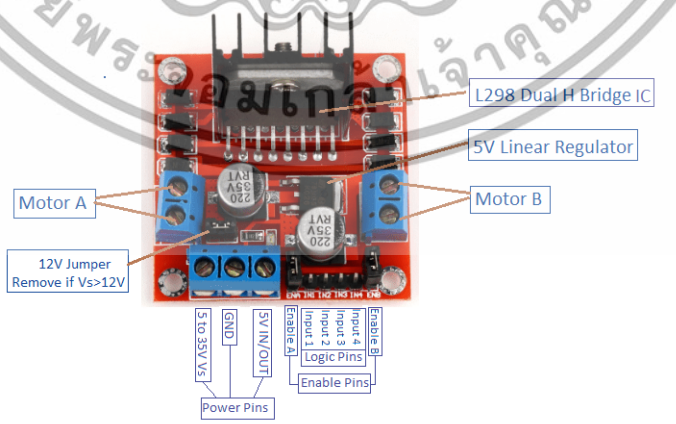
รูปที่ 2.9 DC Gear Motor [23]

ตารางที่ 2.5 คุณลักษณะ DC Gear Motor

คุณลักษณะ	รายละเอียด
แรงดันไฟฟ้าของมอเตอร์	3 - 12 V
กระแสมอเตอร์	70 mA (ทั่วไป)
กระแสสูงสุด	250 mA (สูงสุด)
แรงบิด	สูงสุด 0.8 Kg
อัตราทดเกียร์	1:48
ขนาด	7 cm. *2 cm. *2.5 cm.
แรงดันไฟฟ้าที่ทำงานทั่วไป	6 V

2.4.2 โมดูลขับมอเตอร์ L298N

โมดูลไดรเวอร์ L298N [24] ดังรูปที่ 2.10 ใช้ชิป ST L298N ซึ่งเป็น Dual H-Bridge driver ที่สามารถขับมอเตอร์ DC สองตัวหรือมอเตอร์ Stepper แบบ 2 เฟส สามารถขับมอเตอร์ DC รับแรงดันไฟฟ้าจาก 5V ถึง 35V ซึ่งทำให้เหมาะสำหรับการใช้งานมอเตอร์ DC ขนาด 3V ถึง 30V และมีพอร์ตเอาต์พุต 5V สำหรับจ่ายพลังงาน สามารถควบคุมมอเตอร์ DC ได้ทั้งความเร็วและทิศทาง รวมถึงสามารถควบคุมมอเตอร์Stepper แบบ 2 เฟสได้ ซึ่งเป็นสิ่งจำเป็นสำหรับรถยนต์บังคับ มีคุณลักษณะดังตารางที่ 2.6



รูปที่ 2.10 โมดูลขับมอเตอร์ L298N [25]

ตารางที่ 2.6 คุณลักษณะโมดูลขับมอเตอร์ L298N

คุณลักษณะ	รายละเอียด
ชิปไดรเวอร์	L298N Dual H-Bridge
แรงดันไฟฟ้าจ่ายสำหรับไดรเวอร์ (VMS)	+5 V ~ +35 V
กระแสพีค (Io) ต่อช่อง	2A / Bridge
แรงดันไฟฟ้า (Vss)	4.5 - 5.5 V
กระแสทำงาน	0 ~ 36 mA
ช่วงแรงดันไฟฟ้าสัญญาณควบคุม	4.5 - 5.5 V (LOW 0V HIGH)
กำลังไฟสูงสุดที่ใช้	20W
อุณหภูมิในการเก็บรักษา	-25 ~ +130 °C
ขนาดของบอร์ดไดรเวอร์	55 mm × 60 mm × 30 mm
น้ำหนักของบอร์ดไดรเวอร์	33 g

2.5 ฮาร์ดแวร์ส่วนแขนกล

2.5.1 Micro Servo MG90S

Micro Servo MG90S [26] ดังรูปที่ 2.11 เป็นเซอร์โวมอเตอร์ขนาดเล็กที่อัปเกรดจากรุ่น SG90 โดยมีความสามารถในการหมุน 180 องศา และ 360 องศา ซึ่งเหมาะสำหรับการใช้งานในโครงงานต่าง ๆ เช่น รถยนต์ขนาดเล็ก เรือ เฮลิคอปเตอร์และการควบคุมจากระยะไกล เซอร์โวมอเตอร์นี้มีแรงบิดที่สูงและความเร็วในการทำงานที่ดี ทำให้เป็นตัวเลือกที่น่าสนใจสำหรับการควบคุมการเคลื่อนไหวที่ต้องการความแม่นยำ มีคุณลักษณะดังตารางที่ 2.7

รูปที่ 2.11 Micro Servo MG90S [27]

ตารางที่ 2.7 คุณลักษณะ Micro Servo MG90S

คุณลักษณะ	**รายละเอียด**
ขนาด	22.8 mm × 12.2 mm × 28.5 mm
น้ำหนัก	13.4 g
แรงบิด (Stall Torque)	1.8 kg/cm (ที่ 4.8V) และ 2.2 kg/cm (ที่ 6.0V)
ความเร็วในการทำงาน	0.10 วินาที / 60 องศา (ที่ 4.8V)
	0.08 วินาที / 60 องศา (ที่ 6.0V)
แรงดันไฟฟ้าสำหรับการทำงาน	4.8V – 6.0V
ช่วงอุณหภูมิในการทำงาน	–30 ถึง +60 องศาเซลเซียส
ประเภทการควบคุม	PWM Control

2.5.1.1 การควบคุมมอเตอร์ด้วย PWM

PWM (Pulse Width Modulation) [28] หรือ การมอดูเลตความกว้างของพัลส์ เป็นเทคนิคในการควบคุมสัญญาณดิจิทัลโดยการปรับความกว้างของพัลส์ เพื่อควบคุมพลังงานที่ส่งไปยังอุปกรณ์ โดยการสลับสัญญาณระหว่าง สถานะ HIGH (ON) และ สถานะ LOW (OFF) ในช่วงเวลาหนึ่ง ซึ่งเรียกว่า period โดยการปรับความกว้างของช่วงเวลาในสถานะ HIGH (ON) และสถานะ LOW (OFF) ให้แตกต่างกัน จะช่วยควบคุมปริมาณพลังงานที่ส่งไปยังอุปกรณ์

มอเตอร์ใช้เทคนิค Pulse Width Modulation (PWM) ในการควบคุมมุมการหมุน ซึ่งมีการส่งสัญญาณที่มีลักษณะเป็นพัลส์สี่เหลี่ยม (square wave) โดยมีความถี่ (frequency) ที่เหมาะสมสำหรับการทำงานของเซอร์โวมอเตอร์ ในกรณีของ MG90S คือ 50 Hz ซึ่งหมายความว่าระยะเวลาของพัลส์ทั้งหมดใน 1 วินาทีคือ 50 ครั้ง

โดยจะคำนวณคาบ (Period) ได้จากสมการที่ (2.1)

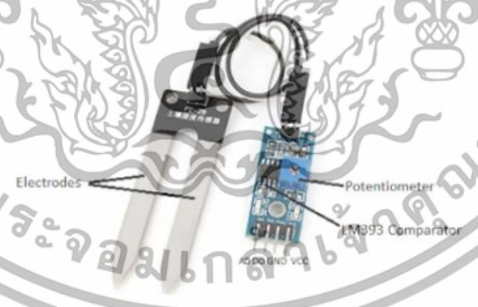
$$T = \frac{1}{f} \tag{2.1}$$

เมื่อคำนวณหาคาบจากสมการที่ (2.1) จะได้ 20 ms มอเตอร์จะได้รับพัลส์ใหม่เพื่อปรับมุมการหมุนตามที่ตั้งค่า และจะปรับค่ามุมตามเวลา On-Time และ Off-Time

2.6 ฮาร์ดแวร์ส่วนอุปกรณ์รดน้ำต้นไม้

2.6.1 โมดูลเซนเซอร์วัดความชื้นในดิน

โมดูลเซนเซอร์วัดความชื้นในดิน [29] ดังรูปที่ 2.12 เป็นอุปกรณ์ที่ใช้สำหรับตรวจวัดความชื้นในดิน โดยจะวัดปริมาณน้ำในดินและส่งค่าระดับความชื้นที่ตรวจวัดได้เป็นผลลัพธ์ออกมา โมดูลนี้มีทั้งขาเอาต์พุตแบบดิจิทัล (Digital) และแบบอนาล็อก (Analog) รวมถึงมีปุ่มปรับค่าแรงดันขอบเขตที่ใช้สำหรับการปรับความไวในการตรวจจับ มีคุณลักษณะดังตารางที่ 2.8



รูปที่ 2.12 โมดูลเซนเซอร์วัดความชื้นในดิน [30]

ตารางที่ 2.8 คุณลักษณะโมดูลเซนเซอร์วัดความชื้นในดิน

คุณลักษณะ	รายละเอียด
แรงดันไฟฟ้าทำงาน	3.3V ถึง 5V DC
กระแสไฟฟ้าที่ใช้งาน	15mA
เอาต์พุตดิจิทัล (DO)	0V ถึง 5V (ปรับระดับ trigger ได้จาก potentiometer)
เอาต์พุตอนาล็อก (AO)	0V ถึง 5V ตามปริมาณความชื้นในดิน
ขนาด	3.2cm × 1.4cm
การออกแบบใช้ LM393	ใช้วงจรเปรียบเทียบแรงดันไฟฟ้า LM393
ปรับความไวได้	มี potentiometer สำหรับปรับระดับแรงดันและความไวของการตรวจจับ

2.6.1.1 ส่วนประกอบสำคัญ

1) LM393 Comparator IC เป็นวงจรเปรียบเทียบแรงดันไฟฟ้าที่ใช้ในการตรวจจับความชื้น โดยขา 2 ของ LM393 เชื่อมต่อกับตัวปรับค่าแรงดัน (Potentiometer) และขา 3 เชื่อมต่อกับเซนเซอร์วัดความชื้น

2) Moisture Sensor Probes หัววัดความชื้นประกอบด้วยโพรบ 2 ชิ้นที่ใช้ตรวจวัดความชื้นในดิน โดยการส่งกระแสไฟฟ้าผ่านดินเพื่อตรวจสอบค่าความต้านทาน ซึ่งค่าความต้านทานจะเปลี่ยนแปลงตามปริมาณน้ำในดิน

3) Potentiometer (ตัวปรับค่าแรงดัน) ใช้สำหรับปรับระดับความไวในการตรวจจับของเอาต์พุตดิจิทัล (DO) และ เอาต์พุตแอนนาล็อก (DO)

2.6.2 Micro Submersible Water Pump

Micro Submersible Water Pump [31] ดังรูปที่ 2.13 เป็นอุปกรณ์ที่ถูกออกแบบมาเพื่อใช้ในการสูบน้ำภายใต้สภาพแวดล้อมที่ต้องการการทำงานใต้น้ำ โดยมีการออกแบบที่ใช้งานง่ายและมีความทนทาน ทำให้เหมาะสมกับการใช้งานในที่ที่มีการสัมผัสน้ำอยู่ตลอดเวลาและใช้พลังงานน้อย มีคุณลักษณะดังตารางที่ 2.9

รูปที่ 2.13 Micro Submersible Water Pump [31]

ตารางที่ 2.9 คุณลักษณะ Micro Submersible Water Pump

คุณลักษณะ	**รายละเอียด**
แรงดันไฟฟ้า	2.5V - 6V DC
กระแสไฟฟ้า	0.1 - 0.2 A
ระยะการยกน้ำสูงสุด	40 - 110 cm (15.75" - 43.4")
อัตราการไหล	80 - 120L/H
เส้นผ่านศูนย์กลางภายนอกของท่อ	7.5 mm (0.3")
เส้นผ่านศูนย์กลางภายในของท่อ	5 mm (0.2")
ขนาดตัวปั๊ม	45 mm x 30 mm x 25 mm
อุณหภูมิน้ำสูงสุดที่รองรับ	80 องศาเซลเซียส
การขับเคลื่อน	แบบแม่เหล็กและใช้ไฟฟ้ากระแสตรง

2.6.3 รีเลย์ JQC-3F-5VDC

รีเลย์ (Relay) JQC-3F-5VDC [32] ดังรูปที่ 2.14 เป็นอุปกรณ์สวิตช์ไฟฟ้าที่ทำงานโดยใช้สัญญาณควบคุมไฟฟ้าขนาดเล็กเพื่อเปิด-ปิดวงจรไฟฟ้าที่มีกระแสไฟฟ้าสูงกว่า เป็นตัวกลางที่ช่วยให้การควบคุมการเปิด-ปิดอุปกรณ์ไฟฟ้าเกิดขึ้นได้อย่างปลอดภัยและแม่นยำโดยทำงานด้วยการใช้ขดลวดแม่เหล็กไฟฟ้า เมื่อมีสัญญาณไฟฟ้าเข้าไปกระตุ้นที่ขดลวด แม่เหล็กไฟฟ้าจะสร้างแรงดูดให้กับคอนแทคภายในรีเลย์เพื่อเปลี่ยนสถานะการเชื่อมต่อของขา ควบคุมอุปกรณ์ไฟฟ้าด้วยการสั่งงานจากสัญญาณแรงดันต่ำ (3V-3.3V) ผ่าน High-Level Trigger โดยโมดูลรีเลย์นี้มีคุณสมบัติเด่นโดยการใช้ Optocoupler Isolator ที่ช่วยป้องกันการรบกวนของสัญญาณ การตอบสนองที่รวดเร็วและอายุการใช้งานที่ยาวนาน มาพร้อมกับวงจรป้องกันไดโอดสำหรับความปลอดภัยในการใช้งาน มีคุณลักษณะดังตารางที่ 2.10

รูปที่ 2.14 รีเลย์ JQC-3F-5VDC [32]

ตารางที่ 2.10 คุณลักษณะรีเลย์ JQC-3F-5VDC

คุณลักษณะ	รายละเอียด
แรงดันไฟฟ้าทำงาน	3V - 3.3V DC
กระแสไฟทำงาน (VCC)	65mA
กระแสสัญญาณ Trigger (IN)	3mA
วงจรป้องกันไดโอด	มีวงจร Flyback Diode Protection
จำนวนช่องควบคุม	1 ช่อง
โหลดที่รองรับ	10A 250VAC / 10A 30VDC
เวลาตอบสนอง	น้อยกว่า 20ms

2.6.3.1 หลักกการทำงาน NC NO และ COM

1) COM เป็นขาหลักที่ใช้สำหรับเชื่อมต่อกับแรงดันไฟฟ้าหรือแหล่งจ่ายไฟฟ้าที่ต้องการควบคุมกระแสไฟจะเข้าสู่รีเลย์ผ่านขา COM และการทำงานของรีเลย์จะเป็นตัวกำหนดว่ากระแสไฟจะถูกส่งไปที่ขา NC หรือ NO

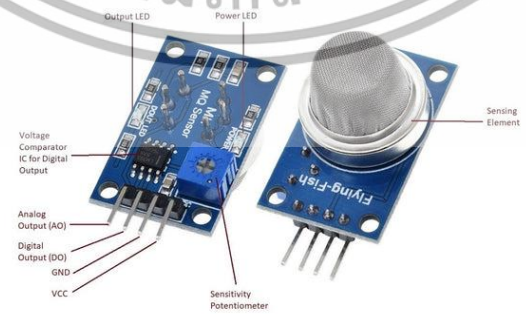
2) NC เป็นขาที่อยู่ในสถานะเชื่อมต่อกับขา COM เมื่อรีเลย์ยังไม่ถูกสั่งให้ทำงาน และเมื่อต่อวงจรผ่านขา NC จะทำให้กระแสไฟไหลผ่านไปยังอุปกรณ์ตลอดเวลา เนื่องจากขา NC ถูกเชื่อมต่อกับ COM ในสถานะปกติเมื่อรีเลย์ได้รับสัญญาณควบคุม HIGH จากไมโครคอนโทรลเลอร์ รีเลย์จะเปลี่ยนสถานะ ทำให้ขา NC ถูกตัดการเชื่อมต่อและหยุดการจ่ายกระแสไฟฟ้าไปยังอุปกรณ์

3) NO เป็นขาที่จะอยู่ในสถานะไม่เชื่อมต่อกับขา COM เมื่อรีเลย์ยังไม่ถูกสั่งให้ทำงาน กระแสไฟจะไม่ไหลผ่านขา NO จนกว่ารีเลย์จะได้รับสัญญาณ HIGH เพื่อเปิดการทำงาน ซึ่งจะทำให้ขา NO เชื่อมต่อกับ COM เมื่อขา NO เชื่อมต่อกับ COM กระแสไฟจะไหลไปยังอุปกรณ์ที่เชื่อมต่อกับขา NO และอุปกรณ์จะทำงาน

2.7 ฮาร์ดแวร์ส่วนอุปกรณ์ตรวจจับควันไฟ

2.7.1 MQ-2 Gas Sensor Module

MQ-2 [33] เป็นโมดูลเซนเซอร์ที่ออกแบบมาสำหรับตรวจจับก๊าซที่ติดไฟได้และควัน ดังรูปที่ 2.15 โดยมีคุณสมบัติการตรวจจับก๊าซหลากหลาย เช่น LPG Propane Hydrogen Methane Butane และSmoke ในช่วงความเข้มข้นของก๊าซที่ 200-10,000 ppm มีคุณลักษณะดังตารางที่ 2.11



รูปที่ 2.15 MQ-2 Gas Sensor Module [34]

ตารางที่ 2.11 คุณลักษณะ MQ-2 Gas Sensor Module

คุณลักษณะ	รายละเอียด
แรงดันไฟฟ้าทำงาน	5V DC
ความต้านทานโหลด (RL)	20 kΩ
ความต้านทานฮีตเตอร์ (RH)	33Ω ± 5%
ความต้านทานการตรวจจับ (RS)	10 kΩ – 60 kΩ
ช่วงความเข้มข้นที่ตรวจจับ	200 - 10,000 ppm
ขนาด	3.2cm × 2.0cm
แก๊สที่ตรวจจับได้	ควัน (Smoke) และก๊าซติดไฟได้
ความไว	มีวงจร Potentiometer สำหรับปรับค่าความไวของการตรวจจับควัน

2.7.1.1 หลักการทำงาน

เมื่อเซนเซอร์ MQ-2 ถูกให้ความร้อนจนถึงอุณหภูมิที่เหมาะสม ชั้นผิวของ Tin Dioxide (SnO_2) จะดูดซับออกซิเจนจากอากาศ ในสภาพอากาศบริสุทธิ์จะเกิดชั้นป้องกันการนำไฟฟ้า (Potential Barrier) บนผิว SnO_2 ทำให้ความต้านทานเพิ่มขึ้น เมื่อมีก๊าซที่ติดไฟได้ เช่น ควันหรือ LPG ก๊าซเหล่านี้จะทำปฏิกิริยากับออกซิเจนบนผิว SnO_2 ลดความหนาแน่นของออกซิเจน และปล่อยอิเล็กตรอนเข้าสู่ SnO_2 ส่งผลให้ความต้านทานลดลงและกระแสไฟฟ้าไหลผ่านเซนเซอร์ได้ง่ายขึ้น

2.7.2 เซนเซอร์วัดอุณหภูมิและความชื้น SHTC3

เซนเซอร์ SHTC3 [34] ดังรูปที่ 2.16 เป็นเซนเซอร์วัดอุณหภูมิและความชื้นที่มีความแม่นยำสูงที่ใช้เทคโนโลยีดิจิทัลในการวัดค่าทั้งสอง ด้วยการใช้งานแบบ I2C ทำให้สามารถรับค่าความชื้นและอุณหภูมิได้แม่นยำและทันสมัย โดยเซนเซอร์นี้มีการทำงานโดยการส่งข้อมูลผ่านพอร์ต I2C ซึ่งมีสองสายคือ SDA (Serial Data) สำหรับส่งข้อมูลและ SCL (Serial Clock) สำหรับการตั้งเวลาในการส่งข้อมูล ตัวเซนเซอร์ทำงานในช่วงแรงดันไฟฟ้า 3.3V ถึง 5.5V และสามารถวัดความชื้น

ในช่วง 0 ถึง 100% RH ด้วยความแม่นยำ ± 2% RH และสามารถวัดอุณหภูมิในช่วง -40 ถึง 125°C มีคุณลักษณะดังตารางที่ 2.12

รูปที่ 2.16 เซนเซอร์วัดอุณหภูมิและความชื้น SHTC3 [35]

ตารางที่ 2.12 คุณลักษณะเซนเซอร์วัดอุณหภูมิและความชื้น SHTC3

คุณลักษณะ	**รายละเอียด**
แรงดันไฟฟ้าทำงาน	3.3 - 5V DC
การสื่อสาร	I2C (Serial Data, Serial Clock)
ช่วงการวัดความชื้น	0 ~ 100% RH
ความแม่นยำของความชื้น	± 2% RH
ช่วงการวัดอุณหภูมิ	-40°C ถึง 125°C
ความแม่นยำของอุณหภูมิ	± 2°C
ขนาด	2.3cm × 1.2cm

2.7.3 โมดูล Active Buzzer

โมดูล Active Buzzer [36] ดังรูปที่ 2.17 เป็นอุปกรณ์ที่ใช้สร้างเสียงเตือนในระบบอิเล็กทรอนิกส์ได้ง่ายดายเพียงแค่จ่ายไฟและสัญญาณดิจิทัลเข้าไปที่ขาควบคุม (I/O) เมื่อจ่ายแรงดันไฟฟ้า (3.3V - 5V DC) ไปที่ขา VCC และสัญญาณดิจิทัลที่ขาควบคุม (I/O) ถูกตั้งค่าเป็น HIGH โมดูลจะเริ่มสร้างเสียงโดยอัตโนมัติ เนื่องจากมีวงจรภายในที่ทำหน้าที่สร้างความถี่เสียง เมื่อโมดูลรับสัญญาณดิจิทัล ระบบจะขยายสัญญาณด้วยทรานซิสเตอร์เบอร์ 9012 ซึ่งช่วยเพิ่มความดังของเสียง มีคุณลักษณะดังตารางที่ 2.13

รูปที่ 2.17 โมดูล Active Buzzer [37]

ตารางที่ 2.13 คุณลักษณะ Active Buzzer

คุณลักษณะ	รายละเอียด
แรงดันไฟฟ้าทำงาน	3.3 - 5V DC
กระแสไฟฟ้า	ต่ำกว่า 30 mA
ทรานซิสเตอร์	PNP Transistor 9012
เสียงที่สร้างได้	ประมาณ 85 dB
การควบคุม	สัญญาณดิจิทัล (HIGH/LOW) ที่ขา I/O
ขนาด	3.3cm × 1.3cm
ความถี่เสียง	ประมาณ 2300 Hz

2.8 เว็บแอปพลิเคชัน

เว็บแอปพลิเคชัน [38] คือ แอปพลิเคชันที่ถูกพัฒนาขึ้นเพื่อให้ผู้ใช้สามารถเข้าถึงและใช้งานผ่านเว็บเบราว์เซอร์โดยตรง โดยไม่จำเป็นต้องติดตั้งซอฟต์แวร์ลงในเครื่องของผู้ใช้ ซึ่งมีข้อดีหลายประการ เช่น การใช้ทรัพยากรต่ำ การอัปเดตที่สะดวกสบาย และการเข้าถึงได้จากอุปกรณ์ต่าง ๆ ที่มีการเชื่อมต่ออินเทอร์เน็ต เว็บแอปพลิเคชันประกอบด้วยหลายส่วนที่ทำงานร่วมกันในการให้บริการและประสบการณ์การใช้งานที่ดีแก่ผู้ใช้ โดยแต่ละส่วนมีหน้าที่ที่สำคัญ ดังนี้

2.8.1 เว็บแอปพลิเคชัน

เว็บแอปพลิเคชันเป็นโปรแกรมซอฟต์แวร์ที่ทำงานบนเซิร์ฟเวอร์และสามารถเข้าถึงได้ผ่านเว็บเบราว์เซอร์ โดยไม่จำเป็นต้องดาวน์โหลดหรือติดตั้งแอปพลิเคชันลงในเครื่องของผู้ใช้ ซึ่งเป็นการให้บริการที่สะดวกและประหยัดทรัพยากร เนื่องจากสามารถใช้งานได้จากอุปกรณ์และ

ระบบปฏิบัติการที่หลากหลายทำหน้าที่เป็นตัวกลางในการรับข้อมูลจากผู้ใช้ ซึ่งอาจรวมถึงการป้อนข้อมูล การคำนวณต่าง ๆ ระบบการชำระเงิน และฟังก์ชันการทำงานอื่น ๆ ที่ผู้ใช้สามารถเข้าถึงได้ เช่น Google Docs Facebook หรือระบบการจัดการต่าง ๆ ที่ใช้ในองค์กร

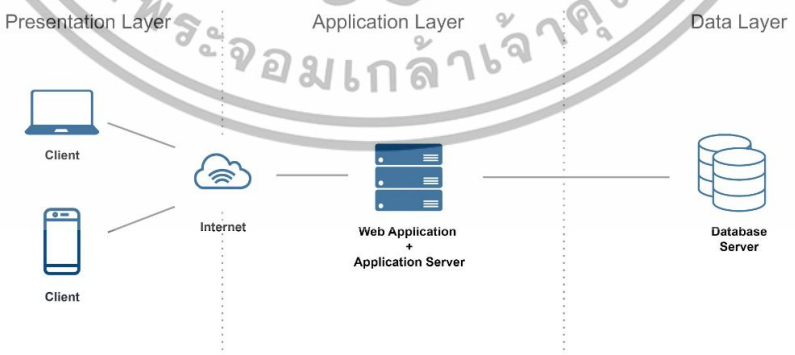
2.8.1.1 โครงสร้างของเว็บแอปพลิเคชัน

Three-Tier Architecture [39] เป็นโครงสร้างการออกแบบซอฟต์แวร์ที่ใช้กันอย่างแพร่หลายในการพัฒนาเว็บแอปพลิเคชัน โดยการแบ่งการทำงานออกเป็นสามชั้นหลัก ได้แก่ Presentation Tier Application Tier และ Data Tier การแยกส่วนเหล่านี้ช่วยให้การพัฒนา การบำรุงรักษา และการปรับขนาดระบบมีประสิทธิภาพมากขึ้น ดังรูปที่ 2.18

1) Presentation Tier เป็นชั้นที่ผู้ใช้มีปฏิสัมพันธ์ โดยแสดงข้อมูลและรับข้อมูลจากผู้ใช้ โดยมักใช้ภาษาทางโปรแกรม เช่น HTML CSS และ JavaScript ในการพัฒนาเว็บเพจหรือแอปพลิเคชันมือถือ เช่น หน้าของเว็บไซต์ eCommerce ที่มีการแสดงผลิตภัณฑ์และให้ผู้ใช้สามารถทำการสั่งซื้อได้

2) Application Tier ทำหน้าที่เป็นชั้นกลางที่ประมวลผลข้อมูลจาก Presentation Tier และสื่อสารกับ Data Tier โดยมีการดำเนินการต่าง ๆ เช่น การคำนวณ การตรวจสอบสิทธิ์ผู้ใช้ (authentication) และการจัดการธุรกรรม เช่น API ที่พัฒนาด้วย Node.js หรือ Python ที่รับข้อมูลจากหน้าเว็บแอปพลิเคชันและส่งข้อมูลไปยังฐานข้อมูล

3) Data Tier เป็นชั้นที่จัดเก็บข้อมูลและควบคุมการเข้าถึงข้อมูล โดยใช้ฐานข้อมูลเชิงสัมพันธ์ หรือ NoSQL เช่น ฐานข้อมูลที่เก็บข้อมูลผู้ใช้ ประวัติการสั่งซื้อและข้อมูลผลิตภัณฑ์



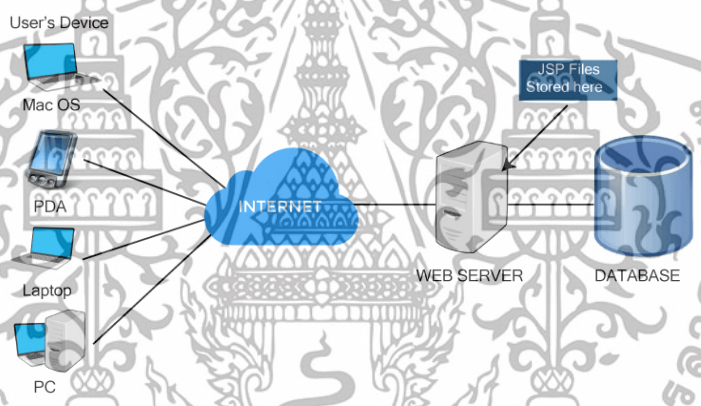
รูปที่ 2.18 Three-Tier Architecture [40]

2.8.2 เว็บเบราว์เซอร์

เว็บเบราว์เซอร์ [41] (Web Browser) คือ ซอฟต์แวร์หรือแอปพลิเคชันที่ใช้สำหรับเข้าถึงและแสดงผลเนื้อหาที่อยู่บนอินเทอร์เน็ต เช่น เว็บไซต์ เว็บแอปพลิเคชัน และข้อมูลอื่น ๆ ที่สามารถแสดงผลผ่าน HTTP หรือ HTTPS [42] โดยเว็บเบราว์เซอร์จะส่งคำขอไปยังเว็บเซิร์ฟเวอร์ จากนั้นจะแสดงผลข้อมูลให้ผู้ใช้ในรูปแบบที่เข้าใจได้ง่าย

2.8.3 เว็บเซิร์ฟเวอร์

เว็บเซิร์ฟเวอร์ (Web Server) [43] ทำหน้าที่ในการรับส่งข้อมูลระหว่างผู้ใช้และเว็บแอปพลิเคชัน โดยจะให้บริการแก่เว็บไซต์และเว็บแอปพลิเคชันที่ต้องการเข้าถึงข้อมูล มีการเชื่อมต่อผ่านเว็บเซิร์ฟเวอร์ ดังรูปที่ 2.19 เว็บเซิร์ฟเวอร์ที่นิยมใช้ ได้แก่ Apache HTTP Server Nginx Microsoft IIS



รูปที่ 2.19 การเชื่อมต่อผ่านเว็บเซิร์ฟเวอร์ [43]

2.8.4 ฐานข้อมูล

ฐานข้อมูล [44] ทำหน้าที่เป็นแหล่งเก็บข้อมูลทั้งหมดที่เว็บแอปพลิเคชันต้องการในการทำงาน เช่น ข้อมูลผู้ใช้ ข้อมูลสินค้า และข้อมูลอื่น ๆ ที่เกี่ยวข้องกับระบบหรือผู้ใช้ การเชื่อมต่อกับฐานข้อมูลช่วยให้เว็บแอปพลิเคชันสามารถจัดการข้อมูลได้แบบเรียลไทม์ ทั้งการเพิ่มข้อมูล แก้ไข ลบ หรือค้นหาข้อมูล

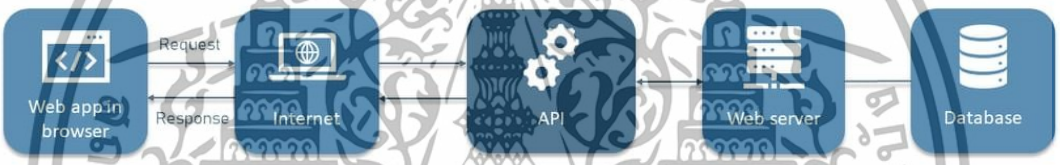
2.8.5 HTTP (Hypertext Transfer Protocol)

HTTP (Hypertext Transfer Protocol) [45] คือ โปรโตคอลพื้นฐานที่ใช้ในการถ่ายโอนไฟล์และข้อมูลต่างๆ บน World Wide Web ไม่ว่าจะเป็นข้อความ ภาพกราฟิก เสียง วิดีโอ

หรือไฟล์มัลติมีเดียอื่นๆ เมื่อผู้ใช้งานเปิดเว็บไซต์บนเบราว์เซอร์ HTTP จะถูกใช้โดยอัตโนมัติเพื่อสื่อสารระหว่างเบราว์เซอร์ (Client) และเซิร์ฟเวอร์ที่เก็บข้อมูลของเว็บไซต์นั้น HTTP เป็นโปรโตคอลชั้นแอปพลิเคชัน (Application Protocol) ที่ทำงานบนพื้นฐานของโปรโตคอล TCP (Transmission Control Protocol) ซึ่งเป็นโปรโตคอลในชั้นเครือข่ายที่รับประกันการส่งข้อมูลระหว่างผู้ใช้งานและเซิร์ฟเวอร์

2.8.6 API (Application Programming Interface)

API [46] คือ เครื่องมือที่ทำหน้าที่เป็นตัวกลางในการเชื่อมต่อระหว่างระบบหนึ่งกับอีกระบบหนึ่ง ช่วยให้ซอฟต์แวร์หรือแอปพลิเคชันภายนอกสามารถเข้าถึง อัปเดต และจัดการข้อมูลในระบบนั้นได้โดยอัตโนมัติ เมื่อมีการส่งคำสั่งมาจากผู้ใช้หรือแอปพลิเคชันอื่น API จะทำหน้าที่รับคำสั่งนั้น ประมวลผล และส่งผลลัพธ์กลับไปยังผู้สั่งในรูปแบบที่เข้าใจได้ง่าย มีการทำงานดังรูปที่ 2.20



รูปที่ 2.20 การทำงานของ API [46]

2.8.7 REST

REST (Representational State Transfer) หรือ RESTful API [47] เป็นรูปแบบการสร้าง Web Service ที่ใช้กันอย่างแพร่หลาย ซึ่งทำงานบนโปรโตคอล HTTP เพื่อสื่อสารและทำงานระหว่าง Client และ Server โดย REST นั้นมีคุณสมบัติเด่นคือการทำงานแบบ stateless หมายความว่า Server จะไม่เก็บสถานะของการเชื่อมต่อแต่ละคำขอ (request) ไว้

REST API รองรับการรับส่งข้อมูลในหลายรูปแบบ เช่น JSON XML และ Text ทำให้มีความยืดหยุ่นสูงในการใช้งานและสามารถพัฒนาด้วยภาษาโปรแกรมต่าง ๆ ได้ โดย RESTful Web Services จะอ้างอิงถึง URI/URL ในการค้นหาและประมวลผลข้อมูล เมื่อส่งคำสั่งไปยัง API จะได้รับ response กลับมา โดยในการใช้ Methods ใน HTTP จะขึ้นอยู่กับการใช้งาน ดังตารางที่ 2.14

ตารางที่ 2.14 HTTP Methods ที่ใช้งานใน REST API

HTTP Method	ความหมาย
GET	ดึงข้อมูลจาก Server
POST	สร้างข้อมูลใหม่บน Server
PUT	แก้ไขข้อมูลทั้งหมดของ Resource
PATCH	แก้ไขข้อมูลบางส่วนของ Resource
DELETE	ลบข้อมูลจาก Server

2.8.8 WebSocket

WebSocket [48] เป็นโปรโตคอลที่ออกแบบมาเพื่อให้การสื่อสารระหว่างเว็บแอปพลิเคชันและเซิร์ฟเวอร์เป็นไปรวดเร็ว โดยสามารถรองรับ Full Duplex หรือ Bidirectional Communication ซึ่งแตกต่างจากโปรโตคอล HTTP ที่อนุญาตให้ส่งข้อมูลได้เพียงทิศทางเดียวเท่านั้น ในการใช้งาน WebSocket หลังจากที่มีการสร้างการเชื่อมต่อผ่านการอัปเกรดจาก HTTP ช่องทางการสื่อสารจะเปิดอยู่ตลอดเวลา ทำให้สามารถส่งและรับข้อมูลได้ทันทีโดยไม่ต้องสร้างการเชื่อมต่อใหม่ทุกครั้ง โปรโตคอล WebSocket ช่วยให้การพัฒนาแอปพลิเคชันที่ต้องการการอัปเดตข้อมูลในเวลาจริง มีกระบวนการเชื่อมต่อดังรูปที่ 2.21

2.8.8.1 การเชื่อมต่อระหว่างคลไอเอนต์และเซิร์ฟเวอร์

1) คลไอเอนต์เริ่มต้นการเชื่อมต่อ HTTP

2) WebSocket จะทำการ handshake ระหว่างคลไอเอนต์และเซิร์ฟเวอร์ ซึ่งทำให้ WebSocket สามารถทำงานร่วมกับพอร์ต HTTP (80 และ 443) และพร็อกซีได้ การ handshake เกิดขึ้นโดยการส่ง HTTP upgrade request header จากคลไอเอนต์ไปยังเซิร์ฟเวอร์ เพื่อขอเปลี่ยนโปรโตคอลจาก HTTP เป็น WebSocket เซิร์ฟเวอร์จะตอบกลับด้วย upgrade response เพื่อยืนยันการเปลี่ยนแปลง โดยการตอบกลับนี้จะมี “Upgrade” header และ “Connection” header ซึ่งทั้งสองจะบ่งบอกถึงโปรโตคอล WebSocket

3) ทั้งคลไอเอนต์และเซิร์ฟเวอร์สามารถแลกเปลี่ยนข้อความได้อย่างอิสระผ่านการเชื่อมต่อที่เปิดอยู่และถาวร โดยไม่จำเป็นต้องทำการ polling หรือคำขอ HTTP ใหม่อย่าง

ต่อเนื่องการสื่อสารนี้จะใช้โมเดลการสื่อสารแบบข้อความ ซึ่งแต่ละข้อความจะอยู่ใน WebSocket frame ที่มีข้อมูลข้อความจริง พร้อมกับข้อมูลเพิ่มเติม เช่น ประเภทและความยาวของ frame ประเภทของข้อความทั้งแบบ Binary หรือข้อความ ขึ้นอยู่กับความต้องการของแอปพลิเคชัน

4) การเชื่อมต่อจะยังคงเปิดอยู่จนกว่าฝ่ายใดฝ่ายหนึ่งจะปิดช่องทางนี้ ซึ่งเมื่อการเชื่อมต่อถูกปิด การสื่อสารระหว่าง Client และเซิร์ฟเวอร์จะสิ้นสุดลง

รูปที่ 2.21 การเชื่อมต่อระหว่างคลไอเอนต์และเซิร์ฟเวอร์ [49]

2.8.9 Next.js

Next.js [50] คือ JavaScript Framework ที่สร้างขึ้นบนพื้นฐานของ เพื่อช่วยในการพัฒนาเว็บแอปพลิเคชันแบบ Full-Stack ซึ่งสามารถทำงานได้ทั้งฝั่ง Client และ Server โดยมันมีฟีเจอร์ที่ช่วยเพิ่มประสิทธิภาพและแก้ปัญหาการพัฒนาเว็บที่ซับซ้อนได้ เช่น Server-Side Rendering (SSR) และ Static Site Generation (SSG) ซึ่งทำให้เว็บโหลดเร็วและแสดงผลได้ดี นอกจากนี้ยังมี API Routes ที่ช่วยให้นักพัฒนาสร้าง API ได้ภายในโพรเจกต์เดียวโดยไม่ต้องใช้เซิร์ฟเวอร์อื่น

การจัดการเส้นทาง (Routing) ใน Next.js ก็ง่ายและทำได้อัตโนมัติผ่านโครงสร้างไฟล์ ทำให้นักพัฒนาสามารถสร้างหน้าใหม่ ๆ หรือเส้นทางแบบไดนามิกได้โดยไม่ต้องเขียนโค้ดเพิ่ม Next.js ยังทำงานร่วมกับ React จึงทำให้สามารถพัฒนาเว็บได้ทั้งแบบ Single Page และ Multi-Page Application

2.8.10 Tailwind CSS

Tailwind CSS [51] คือ CSS Framework แบบ utility-first เป็นเครื่องมือที่มีประสิทธิภาพสำหรับการพัฒนา UI ที่ช่วยให้การสร้างและออกแบบเว็บแอปพลิเคชันอย่างง่าย

รวดเร็ว และมีประสิทธิภาพ ลดความซับซ้อนของคลาส โดยเน้นการใช้คลาส CSS ที่มีลักษณะเฉพาะ (utility classes) เพื่อกำหนดลักษณะและรูปแบบขององค์ประกอบ (elements) บนหน้าเว็บ

2.8.11 JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) [52] เป็นรูปแบบการแลกเปลี่ยนข้อมูลที่ง่ายและมีขนาดเล็ก ซึ่งเหมาะสำหรับการใช้งานบนอินเทอร์เน็ตและ API ต่าง ๆ โดย JSON ได้รับความนิยมอย่างแพร่หลายตั้งแต่ต้นยุค 2000 เนื่องจากโครงสร้างที่อ่านและเข้าใจได้ง่าย ทั้งยังสามารถใช้งานร่วมกับภาษาโปรแกรมหลายภาษา เช่น Python และ JavaScript ข้อมูลใน JSON ประกอบด้วยประเภทต่าง ๆ เช่น ข้อความ (String) ตัวเลข (Number) ค่าจริงหรือเท็จ (Boolean) กลุ่มข้อมูล (Array) ชุดข้อมูลแบบ key-value (Object) และค่า null จุดเด่นของ JSON คือมีโครงสร้างที่ไม่ซับซ้อน จึงทำให้ทั้งมนุษย์และเครื่องจักรสามารถเข้าถึงและประมวลผลได้รวดเร็ว นอกจากนี้ JSON ยังมีขนาดข้อมูลที่เล็กเมื่อเทียบกับรูปแบบอื่น ๆ อย่าง XML ทำให้เป็นที่นิยมและกลายเป็นมาตรฐานสำหรับการแลกเปลี่ยนข้อมูลในระบบดิจิทัล

2.8.12 Cookies

Cookies [53] คือข้อมูลขนาดเล็กที่เซิร์ฟเวอร์ส่งไปยังเบราว์เซอร์ของผู้ใช้เมื่อผู้ใช้เข้าเว็บไซต์ ซึ่งเบราว์เซอร์จะเก็บข้อมูลนี้ไว้และส่งกลับไปยังเซิร์ฟเวอร์เมื่อผู้ใช้เข้ามาเยี่ยมชมเว็บไซต์อีกครั้ง คุกกี้ถูกใช้เพื่อจดจำสถานะต่าง ๆ ของผู้ใช้ เช่น การลงชื่อเข้าใช้ การตั้งค่าภาษาหรือธีม รวมถึงติดตามพฤติกรรมการใช้งานของผู้ใช้ โดยคุกกี้แบ่งออกเป็นสองประเภทหลัก ได้แก่ Session Cookies ที่จะถูกลบเมื่อปิดเบราว์เซอร์ และ Permanent Cookies ที่มีวันหมดอายุหรือถูกกำหนดเวลาไว้ล่วงหน้า เพื่อเพิ่มความปลอดภัย คุกกี้สามารถตั้งค่าให้ทำงานเฉพาะในโปรโตคอลที่เข้ารหัส (HTTPS) และไม่สามารถเข้าถึงได้ผ่าน JavaScript โดยใช้คุณสมบัติ HttpOnly เพื่อป้องกันการโจมตีแบบ XSS (cross-site scripting)

2.8.13 ภาษาคอมพิวเตอร์

ภาษาคอมพิวเตอร์ [54] คือ โปรแกรมหรือชุดคำสั่งที่โปรแกรมเมอร์เขียนเพื่อสั่งงานให้คอมพิวเตอร์ทำงานตามที่ต้องการ ซึ่งแบ่งออกเป็น 5 ระดับ

1) ภาษาเครื่อง (Machine Language) คือ ภาษาที่ใช้ระบบเลขฐานสอง (Binary) โดยใช้ตัวเลขเพียงสองตัว คือ 0 และ 1 ในการสื่อสารกับคอมพิวเตอร์โดยตรง ซึ่งหมายความว่าคำสั่งทุกอย่างที่ป้อนให้คอมพิวเตอร์จะถูกแปลงเป็นชุดของตัวเลขเหล่านี้เพื่อให้คอมพิวเตอร์เข้าใจ

2) ภาษาระดับต่ำ (Low Level Language) เขียนตามลักษณะการทำงานของเครื่อง เป็นภาษาที่ใกล้เคียงกับภาษาเครื่องมากกว่า มักใช้สำหรับควบคุมฮาร์ดแวร์โดยตรง เช่น Device Drivers หรือ Embedded Systems

3) ภาษาระดับสูง (High Level Language) มีคำสั่งเป็นภาษาอังกฤษที่ทำความเข้าใจได้ง่าย โดยใช้ตัวแปลภาษาเป็นภาษาที่มนุษย์เข้าใจได้ง่าย แต่ยากสำหรับเครื่องคอมพิวเตอร์ จึงจำเป็นต้องแปลเป็นภาษาเครื่องด้วย Interpreter หรือ Compiler ก่อน ตัวอย่างภาษาระดับสูง ได้แก่ Python Java C++ และ JavaScript

4) ภาษาระดับสูงมาก (Very High-Level Language) ผู้เขียนไม่จำเป็นต้องบอกวิธีการทำงานละเอียด เพียงระบุคำสั่งให้ทำงานสั้น ๆ ได้ เช่น ภาษา SQL

5) ภาษาธรรมชาติ (Natural Language) ใช้ระบบฐานความรู้และกฎอ้างอิง เพียงป้อนคำถามให้คอมพิวเตอร์ คอมพิวเตอร์จะทำการวิเคราะห์คำถามและค้นหาคำตอบจากระบบฐานความรู้

2.8.13.1 ภาษาซี (C)

ภาษาซี (C) [55] เป็นภาษาการเขียนโปรแกรมที่มีโครงสร้างชัดเจนและมีความสามารถในการใช้งานที่หลากหลาย มันถือว่าเป็นภาษาระดับสูงที่มีลักษณะคล้ายกับภาษาพีชคณิต ซึ่งทำให้โปรแกรมเมอร์สามารถเขียนคำสั่งได้อย่างมีระเบียบ จึงเป็นภาษาที่ใช้เขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์อย่างแพร่หลาย

2.8.13.2 ภาษา JavaScript

ภาษา JavaScript [56] คือ ภาษาคอมพิวเตอร์ที่ได้รับความนิยมสูงสำหรับการพัฒนาเว็บไซต์และแอปพลิเคชันบนระบบอินเทอร์เน็ต เป็นภาษาสคริปต์เชิงวัตถุ (Object-Oriented Scripting Language) ที่ใช้ในการเขียนโปรแกรมเพื่อเพิ่มความสามารถในการโต้ตอบ (interactivity) ของหน้าเว็บ โดย JavaScript สามารถทำงานร่วมกับ HTML และ CSS เพื่อทำให้เว็บไซต์ตอบสนองต่อผู้ใช้ได้ดียิ่งขึ้น

2.8.13.3 ภาษา TypeScript

ภาษา TypeScript [57] เป็นภาษาเขียนโปรแกรมที่พัฒนาโดยบริษัท Microsoft โดยออกแบบมาเพื่อขยายความสามารถของ JavaScript โดยเฉพาะในการจัดการประเภทข้อมูล (Data Type) ซึ่งช่วยให้นักพัฒนาสามารถเขียนโค้ดที่มีความเข้มงวดและตรวจสอบ

ข้อผิดพลาดได้ดีกว่า JavaScript ในขณะที่ยังคงรองรับการพัฒนาแอปพลิเคชันที่สามารถทำงานได้ทั้งฝั่งไคลเอนต์ (Client-side) และเซิร์ฟเวอร์ (Server-side) ผ่านการแปลงโค้ด TypeScript ให้เป็น JavaScript

2.9 ข้อมูล

ข้อมูล [58] หมายถึง ข้อเท็จจริงต่าง ๆ ที่ถูกจัดเก็บลงในระบบฐานข้อมูล หรือระบบแฟ้มข้อมูล โดยข้อมูลอาจถูกจัดเก็บอยู่ในรูปของตัวเลข ข้อความ หรือสื่อมัลติมีเดียร์ต่าง ๆ ได้ในโลกปัจจุบัน ข้อมูลถือได้ว่าเป็นส่วนประกอบที่สำคัญ ดังนั้นข้อมูลที่ดีจะต้องมีความถูกต้อง (Accuracy) สมบูรณ์ (Integrity) และน่าเชื่อถือ (Reliable)

2.9.1 ชนิดของข้อมูล (Type of data)

ข้อมูลที่จัดเก็บลงในแฟ้มข้อมูลหรือฐานข้อมูล นอกจากจะเป็นแบบข้อความแล้ว ในปัจจุบันยังมีข้อมูลชนิดอื่น ๆ ที่สามารถนำมาใช้ประกอบร่วมกัน โดยชนิดของข้อมูลยังแบ่งออกเป็นรูปแบบต่าง ๆ ได้แก่

1) ข้อมูลชนิดข้อความ (Text) คือ ข้อมูลที่ประกอบด้วยตัวอักขระ เช่น ตัวอักษรและตัวเลข รวมกันเป็นคำหรือประโยค โดยไม่ต้องตีความเพิ่มเติม เช่น ชื่อพนักงานหรือที่อยู่ปัจจุบันของพนักงาน

2) ข้อมูลที่เป็นรูปแบบ (Formatted Data) คือ ข้อมูลที่ประกอบด้วยตัวอักขระที่ถูกกำหนดรูปแบบอย่างชัดเจน มักใช้รหัสเพื่อประหยัดพื้นที่จัดเก็บ เช่น รหัสสาขา “TE” แทนสาขาโทรคมนาคม หรือ “TE101” แทนรหัสเซคชันวิชา

3) ข้อมูลรูปภาพ (Images) คือ ข้อมูลที่แสดงในรูปภาพ ซึ่งสามารถช่วยเสริมความเข้าใจข้อมูลข้อความ เช่น ภาพถ่ายนักศึกษาหรือภาพสินค้าพร้อมรายละเอียด ข้อมูลรูปภาพต้องใช้พื้นที่จัดเก็บมากกว่า และมีหลายรูปแบบ เช่น BMP JPG TIFF GIF และ PNG

4) ข้อมูลชนิดเสียง (Audio) คือ ไฟล์เสียงที่ถูกแปลงเป็นดิจิทัล เช่น เสียงพูดหรือเสียงดนตรี สามารถเปิดฟังผ่านอุปกรณ์คอมพิวเตอร์ เช่น ไฟล์ MIDI หรือไฟล์เสียงดิจิทัลทั่วไป

2.10 ฐานข้อมูล

ฐานข้อมูล [59] ระบบที่ใช้จัดเก็บ จัดการ และเข้าถึงข้อมูลในรูปแบบที่มีการจัดระเบียบ ทำให้สามารถค้นหาและจัดการข้อมูลได้อย่างมีประสิทธิภาพ ฐานข้อมูลสามารถใช้ในการเก็บข้อมูลประเภทต่าง ๆ

2.10.1 ฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Non-Relational Database)

ฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Non-Relational Database) [60] ฐานข้อมูลประเภทนี้ออกแบบมาเพื่อจัดการข้อมูลที่ไม่เป็นเชิงสัมพันธ์ หรือข้อมูลที่มีความซับซ้อน เช่น ข้อมูลที่มีโครงสร้างหลากหลายหรือไม่แน่นอน ฐานข้อมูล NoSQL มักใช้ในการจัดเก็บข้อมูลขนาดใหญ่และข้อมูลที่มีความเร็วในการเข้าถึงสูง เช่น MongoDB Cassandra และ Amazon Web Services

2.10.2 ฐานข้อมูลไม่ใช่เชิงสัมพันธ์ประเภท Document Database

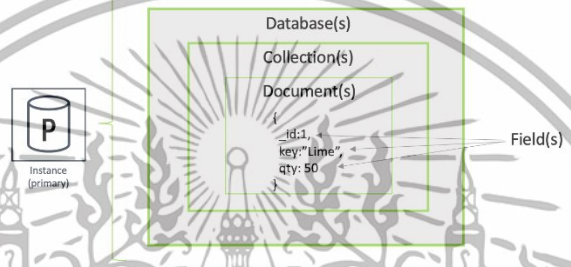
Document Database [61] หรือฐานข้อมูลเอกสาร เป็นประเภทหนึ่งของ NoSQL Database ที่ออกแบบมาเพื่อจัดเก็บและจัดการข้อมูลในรูปแบบของเอกสาร (Document) ซึ่งโดยทั่วไปจะใช้รูปแบบ JSON (JavaScript Object Notation) ดังรูปที่ 2.22 BSON (Binary JSON) หรือ XML ในการเก็บข้อมูล Document Database มีความยืดหยุ่นสูงและเหมาะสำหรับการทำงานกับข้อมูลที่มีโครงสร้างไม่แน่นอนหรือเปลี่ยนแปลงบ่อย

รูปที่ 2.22 ตัวอย่างข้อมูลจะถูกจัดเก็บในรูปแบบเอกสาร JSON

2.10.2.1 โครงสร้างพื้นฐานของ Document Database

Document Database จัดเก็บข้อมูลในรูปแบบเอกสาร (Document) ซึ่งมีลักษณะคล้ายกับ JSON (JavaScript Object Notation) ซึ่งเป็นรูปแบบการแลกเปลี่ยนข้อมูลที่นิยมใช้กัน เอกสารแต่ละชิ้นประกอบด้วยคู่ key-value โดยที่ "key" ทำหน้าที่เป็นชื่อของฟิลด์ และ "value" คือข้อมูลที่จัดเก็บในฟิลด์นั้นๆ ข้อมูลที่จัดเก็บอาจมีได้หลากหลายรูปแบบ เช่น

ข้อความ ตัวเลข วันที่ หรือแม้แต่เอกสารอื่นๆ ที่ซ้อนกันอยู่ภายในเอกสารหลัก ทำให้สามารถจัดเก็บข้อมูลที่มีโครงสร้างซับซ้อนได้ เอกสารเหล่านี้จะถูกจัดกลุ่มไว้ใน "คอลเล็กชัน" (Collection) ที่อยู่ภายในฐานข้อมูล ดังรูปที่ 2.23 ซึ่งมีหน้าที่คล้ายกับตารางในฐานข้อมูลเชิงสัมพันธ์ แต่มีความยืดหยุ่นมากกว่า ตรงที่แต่ละเอกสารในคอลเล็กชันไม่จำเป็นต้องมีโครงสร้างที่เหมือนกัน ทำให้ Document Database เหมาะสำหรับการจัดเก็บข้อมูลที่ไม่มีโครงสร้างที่แน่นอน หรือมีการเปลี่ยนแปลงโครงสร้างอยู่บ่อยครั้ง



รูปที่ 2.23 โครงสร้างพื้นฐานของ Document Database [61]

2.10.3 MongoDB

MongoDB [62] เป็นโปรแกรมฐานข้อมูลที่ใช้หลักการของ NoSQL โดยรองรับการทำงานแบบข้ามแพลตฟอร์ม และมีการออกแบบในลักษณะของฐานข้อมูลเชิงเอกสาร (Document-Oriented Database) ซึ่งข้อมูลจะถูกจัดเก็บในรูปแบบเอกสารที่มีโครงสร้างคล้ายกับ JSON (JavaScript Object Notation) โดยโครงสร้างของเอกสารเหล่านี้สามารถกำหนดหรือไม่กำหนดรูปแบบได้ตามความต้องการ จึงเป็นฐานข้อมูลที่มีความยืดหยุ่นสูงและได้รับความนิยมในการพัฒนาระบบที่ต้องการการจัดการข้อมูลในลักษณะที่ไม่จำเป็นต้องมีโครงสร้างตายตัว ทำให้มีความสามารถในการจัดการข้อมูลขนาดใหญ่ได้อย่างมีประสิทธิภาพ

2.11 Feature Model

Feature Model [63] คือ โมเดลที่ใช้ในการอธิบายและจัดการคุณสมบัติของผลิตภัณฑ์ในรูปแบบที่เป็นระบบ โดยแสดงให้เห็นว่าผลิตภัณฑ์แต่ละรุ่นมีคุณสมบัติอะไรบ้าง และคุณสมบัติ เหล่านั้นมีความสัมพันธ์กันอย่างไร เป็นเครื่องมือหลักในการพัฒนาผลิตภัณฑ์แบบ Product Line Engineering (PLE) ซึ่งมุ่งเน้นการสร้างกลุ่มผลิตภัณฑ์ที่มีความคล้ายคลึงกัน แต่มีความแตกต่างในคุณสมบัติ (Feature) เพื่อตอบสนองความต้องการที่หลากหลายของลูกค้า

โดย Feature Model จะแสดงรายละเอียดของคุณสมบัติต่าง ๆ ของผลิตภัณฑ์ พร้อมทั้งความสัมพันธ์ระหว่างคุณสมบัติเหล่านั้น

2.11.1 Feature Diagram

Feature Diagram [64] เป็นเครื่องมือที ่ใช้ในการแสดงภาพรวมของ Feature Model ซึ่งเป็นส่วนสำคัญในการพัฒนาผลิตภัณฑ์แบบ Product Line Engineering (PLE) โดย Feature Diagram จะแสดงคุณสมบัติต่าง ๆ ของผลิตภัณฑ์และความสัมพันธ์ระหว่างคุณสมบัติเหล่านั้นในรูปแบบกราฟิก ทำให้ง่ายต่อการเข้าใจและจัดการความหลากหลายของผลิตภัณฑ์ โดยมีโครงสร้างและความสัมพันธ์ระหว่างคุณสมบัติ ดังนี้

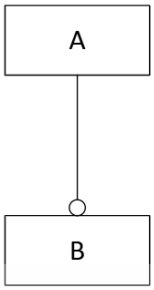
1) คุณสมบัติหลัก (Parent Feature) เป็นคุณสมบัติหลักหรือหัวข้อหลักของผลิตภัณฑ์ ซึ่งมักแสดงอยู่ด้านบนสุดของไดอะแกรม

2) ความสัมพันธ์ระหว่างคุณสมบัติหลัก (Parent Feature) และคุณสมบัติย่อย (Child Feature) ได้แก่

- Mandatory relation คุณสมบัติย่อยนี้ต้องถูกเลือก หากคุณสมบัติหลักถูกเลือก เปรียบเสมือนส่วนประกอบที่ขาดไม่ได้ของผลิตภัณฑ์ มีสัญลักษณ์แสดงความสัมพันธ์ดังรูปที่ 2.24

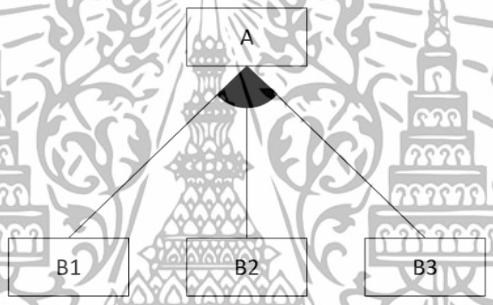
รูปที่ 2.24 ความสัมพันธ์แบบ Mandatory

- Optional relation คุณสมบัติย่อยนี้อาจถูกเลือกหรือไม่ก็ได้ขึ้นอยู่กับความต้องการของแต่ละผลิตภัณฑ์ เปรียบเสมือนอุปกรณ์เสริม มีสัญลักษณ์แสดงความสัมพันธ์ดังรูปที่ 2.25



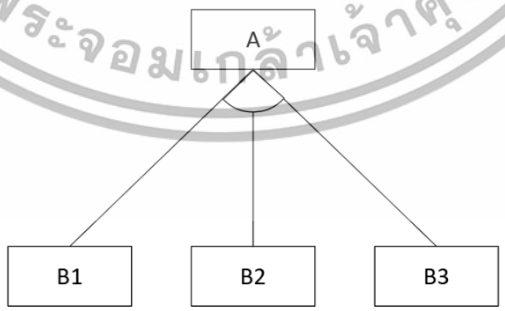
รูปที่ 2.25 ความสัมพันธ์แบบ Optional

- Or relation ต้องเลือกคุณสมบัติย่อยอย่างน้อยหนึ่งคุณสมบัติจากกลุ่มนี้ เปรียบเสมือนการเลือกอย่างใดอย่างหนึ่ง มีสัญลักษณ์แสดงความสัมพันธ์ ดังรูปที่ 2.26



รูปที่ 2.26 ความสัมพันธ์แบบ Or

- Alternative relation ต้องเลือกคุณสมบัติย่อยเพียงหนึ่งคุณสมบัติจากกลุ่มนี้ เปรียบเสมือนการเลือกเพียงหนึ่งเดียว มีสัญลักษณ์แสดงความสัมพันธ์ ดังรูปที่ 2.27



รูปที่ 2.27 ความสัมพันธ์แบบ Alternative

3) Cross-tree Constraints ซึ่งเป็นข้อจำกัดที่เชื่อมโยงคุณสมบัติที่อยู่ในส่วนต่าง ๆ ของ Feature Model เข้าด้วยกัน ข้อจำกัดที่พบบ่อย คือ

- Requires การเลือกคุณสมบัติ A จำเป็นต้อง เลือกคุณสมบัติ B ด้วย มีสัญลักษณ์แสดงความสัมพันธ์ ดังรูปที่ 2.28

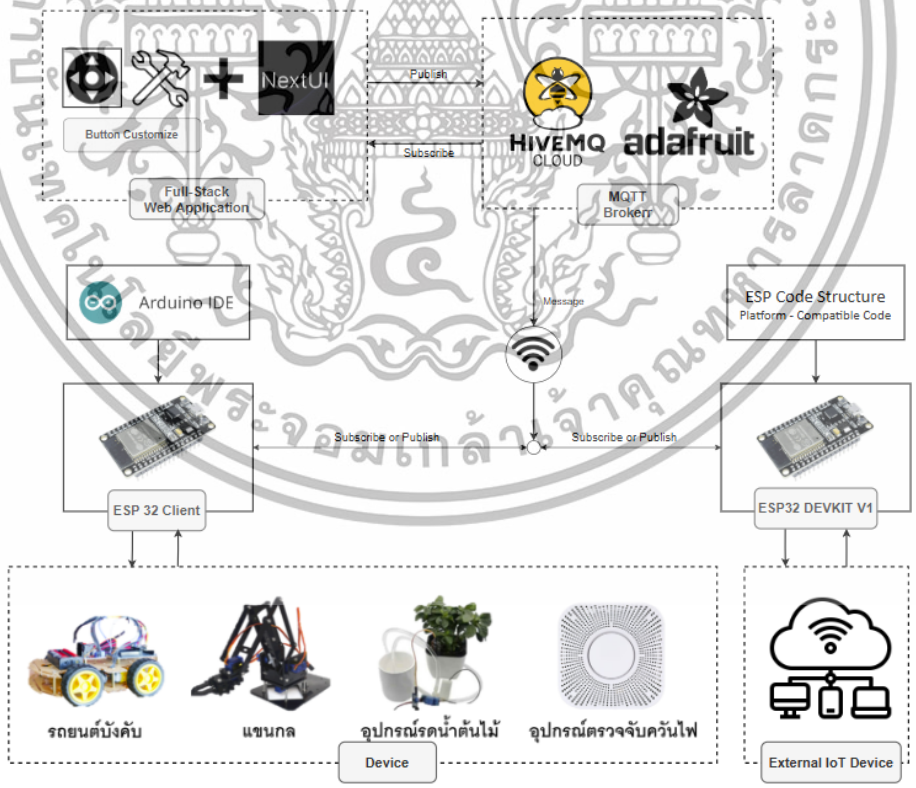
รูปที่ 2.28 ข้อจำกัดที่เชื่อมโยงคุณสมบัติแบบ Requires

- Excludes การเลือกคุณสมบัติ A และคุณสมบัติ B ไม่สามารถ ถูกเลือกพร้อมกันได้ มีสัญลักษณ์แสดงความสัมพันธ์ ดังรูปที่ 2.29

รูปที่ 2.29 ข้อจำกัดที่เชื่อมโยงคุณสมบัติแบบ Excludes

บทที่ 3
การออกแบบและการจัดทำปริญญานิพนธ์

ปริญญานิพนธ์แพลตฟอร์มจัดการอุปกรณ์ IoT ประกอบด้วยองค์ประกอบหลักหลายส่วน โดยเริ่มจาก Full-Stack Web Application ซึ่งทำหน้าที่เป็นโปรแกรมประสานสำหรับผู้ใช้ในการแสดงผลและควบคุมอุปกรณ์ โดยใช้หลักการสื่อสารแบบPublish/Subscribe ผ่าน MQTT Broker ที่ใช้แพลตฟอร์ม HiveMQ และ Adafruit เป็นตัวกลางในการส่งและรับข้อความระหว่างเว็บแอปพลิเคชันและอุปกรณ์ปลายทาง ในส่วนของอุปกรณ์ควบคุม ESP32 จะถูกพัฒนาและอัปโหลดโค้ดผ่าน Arduino IDE เพื่อทำหน้าที่เป็น ESP32 Client ที่สามารถรับและส่งข้อมูลผ่าน MQTT ได้ ในกรณีที่เป็นอุปกรณ์ IoT จากภายนอกต้องใช้ ESP32 DEVKIT V1 โดยในแต่ละตัวต้องใช้โค้ดที่รองรับการใช้งานร่วมกับแพลตฟอร์ม อุปกรณ์ที่รองรับการควบคุมภายในโครงงานประกอบด้วย รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ ซึ่งทั้งหมดทำงานร่วมกับ ESP32 และสื่อสารผ่าน MQTT Broker ดังแสดงในรูปที่ 3.1



รูปที่ 3.1 บล็อกไดอะแกรมภาพรวมของปริญญานิพนธ์

3.1 การออกแบบอุปกรณ์ IoT

3.1.1 การเก็บชื่อ Wi-Fi เริ่มต้นของอุปกรณ์ IoT

ออกแบบมาเพื่อให้ ESP32 เชื่อมต่อ Wi-Fi โดยอัตโนมัติ โดยบันทึกค่า SSID และรหัสผ่าน Wi-Fi ลงใน EEPROM ซึ่งเป็นหน่วยความจำถาวรแม้เมื่ออุปกรณ์ปิดหรือรีเซ็ต ข้อมูลเหล่านี้จะช่วยให้ ESP32 ใช้ค่า Wi-Fi เดิมที่บันทึกไว้เพื่อเชื่อมต่อใหม่โดยไม่ต้องตั้งค่าซ้ำ ซึ่งจะช่วยให้สลับเครือข่ายได้ง่ายและเหมาะในสถานการณ์ที่ต้องการสลับการเชื่อมต่อ Wi-Fi หรือใช้การเชื่อมต่อ Wi-Fi เริ่มต้นของ Wi-Fi ที่ได้ออกแบบไว้ โดยมีแนวคิดการออกแบบ ดังนี้

1) กำหนดค่าเริ่มต้นของ Wi-Fi โดย SSID = "Default" และรหัสผ่าน = "12345678" ดังรูปที่ 3.2

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <EEPROM.h>

// WiFi
const char *ssid = "Default";
const char *password = "12345678";
```

รูปที่ 3.2 คำสั่งโปรแกรมการบันทึก SSID และรหัสผ่าน Wi-Fi เริ่มต้น

2) เมื่อ ESP32 เริ่มต้นทำงาน โค้ดจะสั่งให้ตรวจสอบค่าที่บันทึกไว้ใน EEPROM ดังรูปที่ 3.3 หากพบข้อมูล Wi-Fi จะดึงค่าที่เก็บไว้ออกมาและใช้ในการเชื่อมต่อโดยอัตโนมัติ ดังรูปที่ 3.4 แต่ถ้าหากใน EEPROM ยังไม่มีข้อมูล เช่น ในการเริ่มต้นใช้งานครั้งแรกหรือเมื่อทำการล้างข้อมูล ESP32 จะใช้ค่า SSID และรหัสผ่านที่กำหนดไว้ในโค้ดเป็นค่าเริ่มต้นแทน

```
// Read WiFi details from EEPROM
String getwfn;
for (int i = 0; true; i++) {
    char c = EEPROM.read(i);
    if (c == '\0') break;
    getwfn += c;
}

// Read getwfp from EEPROM
String getwfp;
int pwdStart = getwfn.length() + 1;
for (int i = pwdStart; true; i++) {
    char c = EEPROM.read(i);
    if (c == '\0') break;
    getwfp += c;
}
```

รูปที่ 3.3 คำสั่งโปรแกรมนำค่าเริ่มต้น Wi-Fi เพื่อใช้ในการเชื่อมต่อ

```
    // Connect to WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        digitalWrite(statusconnect, HIGH);  // LED ON
        delay(500);
        digitalWrite(statusconnect, LOW);   // LED OFF
        delay(500);
        Serial.println("-------  Connecting to WiFi..  -------");
    }
    digitalWrite(statusconnect, HIGH);  // LED ON
    Serial.println("-------  Connected to the WiFi network  -------");
}
```

รูปที่ 3.4 คำสั่งโปรแกรมในการเชื่อมต่อ Wi-Fi เริ่มต้น

3) แต่ในกรณีการรีเซ็ตกลับไปใช้ค่า Wi-Fi เริ่มต้น ถ้าหากใน EEPROM ยังไม่มีข้อมูล เช่น ในการเริ่มต้นใช้งานครั้งแรกหรือเมื่อทำการล้างข้อมูล ESP32 จะใช้ค่า SSID และรหัสผ่านที่กำหนดไว้ในที่เป็นค่าเริ่มต้นแทน โดยทำการลบข้อมูล Wi-Fi จาก EEPROM เมื่อได้รับคำสั่งข้อความ defaultwifi คำสั่งโปรแกรมจะทำการลบข้อมูลใน EEPROM และเชื่อมต่อด้วยค่าเริ่มต้น ดังรูปที่ 3.5

```
if (message.startsWith("defaultwifi")) {
    for (int i = 0; i < EEPROM_SIZE; i++) {
        EEPROM.write(i, 0);
    }
    EEPROM.commit();
```

รูปที่ 3.5 คำสั่งโปรแกรมเมื่อรีเซ็ตกลับไปเป็น Wi-Fi เริ่มต้น

3.1.2 การเปลี่ยนชื่อ Wi-Fi ของอุปกรณ์ IoT

การออกแบบขั้นตอนการเปลี่ยนชื่อ Wi-Fi เริ่มต้นที่ติดตั้งในอุปกรณ์ หรือการรีเซ็ตชื่อ Wi-Fi ให้กลับไปเป็นค่าเริ่มต้น มีการพิจารณาจากปัญหาที่อาจเกิดขึ้นเมื่อผู้ใช้ต้องการนำอุปกรณ์ไปใช้งานในเครือข่าย Wi-Fi ที่มีชื่อและรหัสผ่านแตกต่างกัน ขณะเดียวกัน อุปกรณ์จำเป็นต้องเชื่อมต่อกับ Wi-Fi เพื่อใช้เป็นช่องทางหลักในการเชื่อมต่อกับ Cloud Broker จึงมีการออกแบบวิธีแก้ไขปัญหาในลำดับขั้นตอน ดังนี้

1) อุปกรณ์ทุกตัวที่ออกแบบมาจะมีค่าเริ่มต้นของ Wi-Fi เหมือนกัน โดยกำหนดชื่อ Wi-Fi เป็น “Default” และรหัสผ่านเป็น “12345678” เพื่อให้ผู้ใช้สามารถเชื่อมต่อกับอุปกรณ์ได้ง่ายในครั้งแรก

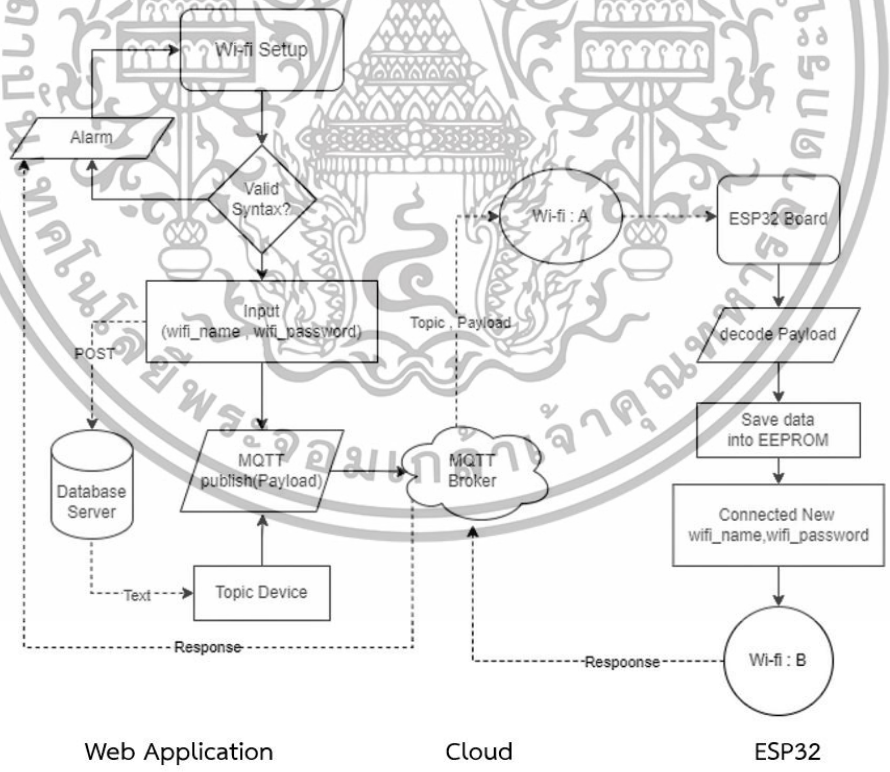
2) ผู้ใช้ต้องแชร์ Wi-Fi Hotspot โดยตั้งค่าชื่อและรหัสผ่านให้ตรงกับค่าที่อุปกรณ์กำหนดไว้ เพื่อทำการเชื่อมต่อในครั้งแรก

3) เมื่อเชื่อมต่อกับอุปกรณ์เรียบร้อยแล้ว ผู้ใช้จะต้องผูกอุปกรณ์กับบัญชีของตนในเว็บแอปพลิเคชัน จากนั้นสามารถแก้ไขรหัสผ่าน Wi-Fi ผ่านเว็บแอปพลิเคชัน โดยระบุเป็น Wi-Fi ของผู้ใช้ที่ต้องการเชื่อมต่อ

4) หลังจากทำการเปลี่ยนค่า Wi-Fi เสร็จสิ้น ผู้ใช้จะต้องกดปุ่ม Reset บนอุปกรณ์เพื่อให้เชื่อมต่อกับ Wi-Fi ใหม่ที่ได้ตั้งค่าไว้

5) หากผู้ใช้ต้องการเปลี่ยนค่า Wi-Fi กลับไปเป็นค่าเริ่มต้น สามารถกดปุ่ม Reset Wi-Fi บนอุปกรณ์ หรือเลือกรีเซ็ตจากหน้าเว็บแอปพลิเคชัน ซึ่งจะทำให้ชื่อและรหัสผ่าน Wi-Fi กลับไปเป็นค่าเริ่มต้นตามที่กำหนดไว้

โดยจากขั้นตอนการอออกแบบจึงได้กระบวนการทำงานระบบเปลี่ยนการเชื่อมต่อ Wi-Fi ดังรูปที่ 3.6



รูปที่ 3.6 กระบวนการเปลี่ยนการเชื่อมต่อ Wi-Fi

3.1.2.1 การออกแบบการรับ Payload จาก MQTT

บอร์ดไมโครคอนโทรลเลอร์ ESP32 ได้รับการออกแบบให้รองรับการรับส่งข้อมูลจากแพลตฟอร์มผ่านการเชื่อมต่อ MQTT โดยข้อมูลที่ส่งเข้ามาจะประกอบด้วย 3 ตัวแปรหลัก ได้แก่ Topic Payload และ Length มีรายละเอียดคำสั่งโปรแกรม ดังรูปที่ 3.7

- Topic ทำหน้าที่เป็นคีย์ในการเชื่อมต่อระหว่างอุปกรณ์และแพลตฟอร์ม โดยข้อมูลที่รับเข้ามาใน Topic จะอยู่ในรูปแบบของตัวอักษร (character) และมีความยาวไม่เกิน 30 ตัว ซึ่งสามารถมองว่า Topic เปรียบเสมือนหัวข้อหลัก (header) ในการเข้าถึงข้อมูลภายในเฟรมของแพ็กเกจข้อมูล หากตรวจสอบพบว่า Topic ถูกต้อง ระบบจะทำการรับและอ่านข้อมูลที่เหลือจาก Payload
- Payload เป็นข้อมูลที่รับเข้ามาในรูปแบบของ byte ซึ่งในขั้นตอนการประมวลผลจะมีการวนลูปตามจำนวน byte ที่ระบุในตัวแปร Length ขณะวนลูป ข้อมูล Payload ที่รับเข้ามาจะถูกเก็บไว้ในตัวแปร Message และมีการแปลงข้อมูลจาก byte เป็น character เพื่อให้อ่านข้อมูลได้ในรูปแบบของข้อความตัวอักษร เมื่อวนลูปครบแล้ว จะได้ข้อความที่ถูกส่งมาจาก Payload
- Length เป็นตัวแปรที่บ่งบอกถึงความยาวของข้อมูล (byte) ที่มากับ Payload โดยค่าที่ได้จะถูกนำมาใช้ในการควบคุมจำนวนรอบในการวนลูปเพื่อประมวลผลข้อมูล

```
void callback(char *topic, byte *payload, unsigned int length) {
    Serial.printf("\nMessage arrived in topic: ");
    Serial.print(topic);
    String message;
    String wfn;
    String wfp;
    for (int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
```

รูปที่ 3.7 คำสั่งโปรแกรมการทำงานรับ Payload จาก MQTT

3.1.2.2 การออกแบบเพื่ออ่านการเปลี่ยนค่า Wi-Fi

ออกแบบให้หลังจากที่ข้อมูลใน Payload ถูกแปลงแล้ว โปรแกรมได้สร้างเงื่อนไขเพื่อดักจับข้อความที่เข้ามา หากข้อความที่รับมามีค่าตรงกับคำว่า wfn ระบบจะเข้าสู่โหมดการเปลี่ยนค่า Wi-Fi โดยทันที เมื่อเข้าสู่ฟังก์ชันการเปลี่ยนค่า Wi-Fi โปรแกรมจะทำการวนลูปเพื่อตรวจสอบข้อมูลในหน่วยความจำ EEPROM และทำการล้างค่าข้อมูลเก่าทั้งหมดก่อน เพื่อเตรียมสำหรับการบันทึกข้อมูล Wi-Fi ใหม่ โดยมีคำสั่งโปรแกรมตรวจสอบข้อมูลในหน่วยความจำ EEPROM ดังรูปที่ 3.8

```
if(message.startsWith("wfn")){
    for (int i = 0; i < EEPROM_SIZE; i++) {
        EEPROM.write(i, 0);
    }
    EEPROM.commit();
    Serial.println("\n\n========== [ EEPROM Clearing Old data... ] ==========\n");
```

รูปที่ 3.8 คำสั่งโปรแกรมตรวจสอบข้อมูลในหน่วยความจำ EEPROM

3.1.3.3 การเก็บข้อมูล SSID และรหัสผ่าน Wi-Fi เข้า EEPROM

วิธีการเก็บข้อมูล SSID และรหัสผ่านของเครือข่าย Wi-Fi ลงในหน่วยความจำ EEPROM ของ ESP32 เพื่อให้สามารถใช้ข้อมูลนี้ในการเชื่อมต่อกับ Wi-Fi ได้แม้หลังจากมีการรีสตาร์ทอุปกรณ์ ข้อมูล SSID และรหัสผ่านที่ได้รับผ่านทางข้อความจะถูกนำมาประมวลผลและจัดเก็บลงใน EEPROM ซึ่งเป็นหน่วยความจำถาวรที่สามารถคงค่าไว้ได้แม้ไม่มีการจ่ายไฟ กระบวนการนี้ช่วยให้ระบบสามารถเชื่อมต่อกับ Wi-Fi ที่ถูกต้องได้ทุกครั้งที่เริ่มต้นการทำงานของอุปกรณ์ โดยมีวิธีการการเก็บข้อมูล SSID และรหัสผ่าน Wi-Fi เข้า EEPROM ดังนี้

1) แยกข้อความที่ได้รับ โดยเริ่มจากการรับข้อความที่มีรูปแบบ wfn:xxxxxx ในการเก็บค่า SSID ของ Wi-Fi ใหม่ และ wfp:xxxxxxxx ในการเก็บรหัสผ่านของ Wi-Fi โดยใช้เครื่องหมาย "," เพื่อแยกข้อมูลออกเป็นสองส่วน คือ SSID ที่ขึ้นต้นด้วย wfn: และรหัสผ่าน ที่ขึ้นต้นด้วย wfp: ดังรูปที่ 3.9

2) เก็บข้อมูลในตัวแปร โดยใช้ while loop เพื่อแยกข้อมูล key และ value ของแต่ละส่วน โดยแบ่งด้วยเครื่องหมาย ":" เพื่อให้ได้ค่า SSID และรหัสผ่านแยกกัน จากนั้นเก็บค่าในตัวแปรเพื่อเตรียมใช้งานในขั้นตอนถัดไป

3) แสดงผลและตรวจสอบข้อผิดพลาด เมื่อแยกข้อมูลสำเร็จ ระบบจะแสดงผลข้อมูลที่ถูกแยกใน console เพื่อให้เห็นค่าที่ได้รับ หากเกิดข้อผิดพลาดในการรับข้อมูล ข้อมูลจะไม่ถูกแสดงเพื่อป้องกันการใช้งานค่าที่ผิดพลาด

4) บันทึกข้อมูลใน EEPROM หลังจากตรวจสอบว่าข้อมูลถูกต้อง จะวนลูปข้อมูลทั้งหมดเพื่อนำไปเก็บใน EEPROM ของ ESP32 โดยใช้คำสั่งในการเขียนข้อมูลลงใน EEPROM ซึ่งจะเก็บค่าที่จำเป็นสำหรับการเชื่อมต่อ Wi-Fi ใหม่ในครั้งถัดไป ดังรูปที่ 3.10

```
char buffer[100];
message.toCharArray(buffer, 100); // Convert String to char array
char *pair = strtok(buffer, ",");
char key[50], value[50];
while (pair != NULL) {
    sscanf(pair, "%49[^:]:%49s", key, value);
    if (strcmp(key, "wfn") == 0) {
        wfn = value;
    } else if (strcmp(key, "wfp") == 0) {
        wfp = value;
    }
    pair = strtok(NULL, ",");
}
Serial.printf("[Payload Detail] >> ");
Serial.print(message);
Serial.printf("\n New Wi-fi Name : ");
Serial.print(wfn);
Serial.printf("\n New Wi-fi Password : ");
Serial.print(wfp);
```

รูปที่ 3.9 คำสั่งโปรแกรมแยกข้อความ

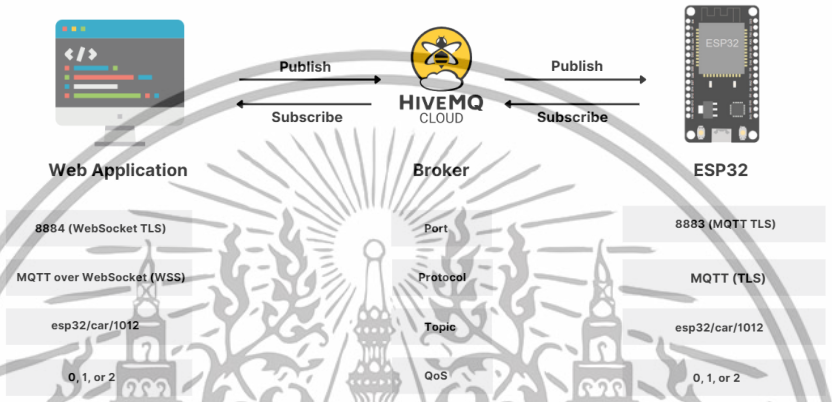
```
for (int i = 0; i < wfn.length(); i++) {
    EEPROM.write(i, wfn[i]);
}
EEPROM.write(wfn.length(), '\0');  // Null terminator for string

// Write wfp to EEPROM starting from address after username
int pwdStart = wfn.length() + 1;
for (int i = 0; i < wfp.length(); i++) {
    EEPROM.write(pwdStart + i, wfp[i]);
}
EEPROM.write(pwdStart + wfp.length(), '\0');  // Null terminator for string
EEPROM.commit();
```

รูปที่ 3.10 คำสั่งโปรแกรมเพื่อบันทึกข้อมูลเข้า EEPROM

หลังจากนี้เมื่อผู้ใช้ได้ทำการเปิดหรือปิดอุปกรณ์ใหม่ ตัวอุปกรณ์จะจดจำค่า wi-fi อันใหม่ของผู้ใช้ไปตลอดจนกว่าจะมีการ Reset เกิดขึ้น ซึ่งการออกแบบนี้เป็นประโยชน์ต่อการเชื่อมต่อไร้สายในวง LAN ที่มีความหลากหลายและแตกต่างกันในเรื่องของการเข้าถึง

3.1.3 การรับและส่งข้อมูลของอุปกรณ์ IoT

การรับและส่งข้อมูลของอุปกรณ์ IoT สื่อสารกันผ่าน MQTT Protocol ทำให้อุปกรณ์สามารถสื่อสารกันได้โดยใช้โครงสร้างแบบ Publisher/Subscriber ผ่านตัวกลางที่เรียกว่า MQTT Broker ซึ่งทำหน้าที่รับส่งและกระจายข้อมูลไปยังอุปกรณ์ที่ต้องการ โดยที่อุปกรณ์ต้องมีการเชื่อมต่ออินเทอร์เน็ตเพื่อจะได้สื่อสารกันได้ ดังรูปที่ 3.11



รูปที่ 3.11 การสื่อสารผ่าน MQTT Protocol

อุปกรณ์ IoT ที่ต้องการส่งข้อมูล (Publisher) จะสร้างการเชื่อมต่อกับ MQTT Broker ผ่านเครือข่ายอินเทอร์เน็ต จากนั้นจะทำการ Publish ข้อมูลไปยัง Topic ที่กำหนดไว้ โดยข้อมูลที่ส่งจะอยู่ในรูปแบบของข้อความ (String) หรือ JSON เมื่อข้อมูลถูกส่งไปยัง MQTT Broker แล้ว ตัวกลางนี้จะจัดเก็บและรอให้ Subscriber ที่สมัครรับข้อมูลจาก Topic ดังกล่าวสามารถดึงข้อมูลไปใช้งานได้ สามารถเขียนคำสั่งโปรแกรมเพื่อให้ส่งข้อมูลของอุปกรณ์ IoT ที่ต้องการได้ ดังรูปที่ 3.12

```
String soilMoistureMessage ="value1:" + String(soilMoistureValue);
client.publish(topic,soilMoistureMessage.c_str() );
Serial.print("Soil moisture value : ");
Serial.println(soilMoistureMessage);
```

รูปที่ 3.12 ตัวอย่างคำสั่งโปรแกรมส่งข้อมูล (Publish)

กระบวนการรับข้อมูลของอุปกรณ์ IoT ทำหน้าที่เป็น Subscriber จะต้องเชื่อมต่อกับ MQTT Broker ผ่านเครือข่ายอินเทอร์เน็ตเช่นเดียวกับการ Publish ข้อมูล และทำการ Subscribe ไปยัง Topic ที่ต้องการ เมื่อมีคำสั่งใหม่ถูกส่งมายัง Topic นี้จากแอปพลิเคชันหรือผู้ใช้ MQTT Broker จะส่งข้อมูลดังกล่าวไปยังอุปกรณ์ IoT ที่สมัครรับข้อมูลอยู่ทันที อุปกรณ์ IoT ที่ได้รับข้อมูลจะต้องมีฟังก์ชัน Callback ที่สามารถอ่านค่าข้อมูลที่ได้รับและดำเนินการตามคำสั่งที่กำหนดไว้ ดังรูปที่ 3.13

```
  client.subscribe(topic);

}
void callback(char *topic, byte *payload, unsigned int length) {
  Serial.printf("\nMessage arrived in topic: ");
  Serial.print(topic);
  String message;
  String wfn;
  String wfp;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
```

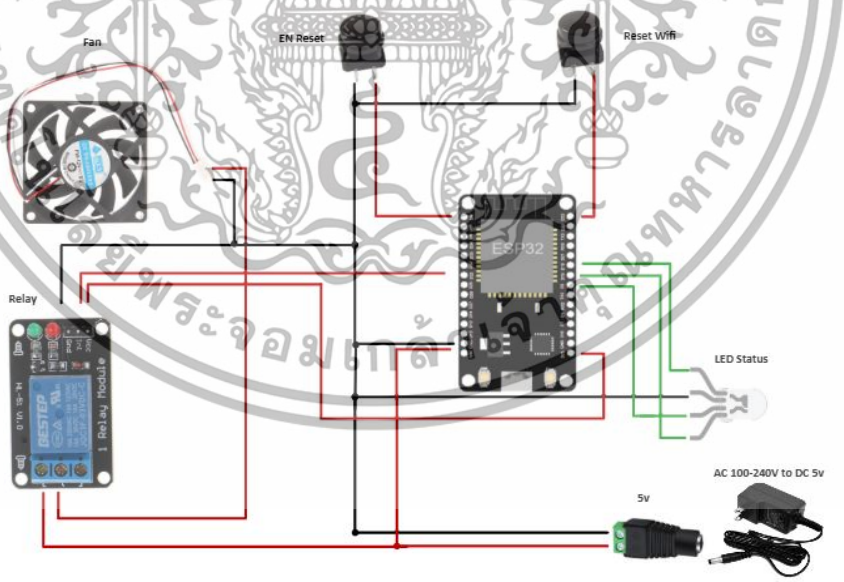
รูปที่ 3.13 ตัวอย่าง Subscribe ไปยัง Topic และฟังก์ชัน Callback อ่านค่าข้อมูล

3.1.4 การออกแบบอุปกรณ์ IoT

ในการออกแบบอุปกรณ์ IoT จะทำการจำลองการเชื่อมต่อขิงอุปกรณ์ โดยมีออกแบบอุปกรณ์ใด ๆ เพื่อนำมาใช้งานร่วมกับแพลตฟอร์ม และออกแบบอุปกรณ์ตัวอย่างเพื่อแสดงการทำงานและการเชื่อมต่อผ่านแพลตฟอร์มจัดการ IoT ที่สร้างขึ้น

3.1.4.1 ตัวอย่างการออกแบบอุปกรณ์ควบคุมใด ๆ

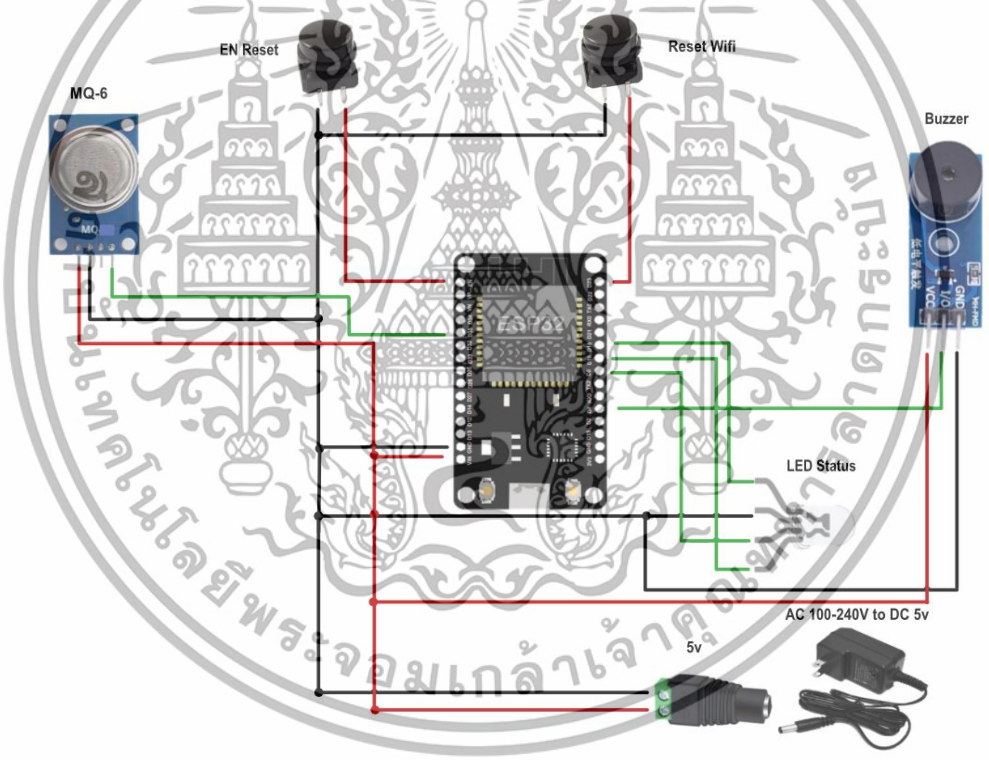
ออกแบบการทำงานของอุปกรณ์ควบคุมใด ๆ ตัวอย่างให้บอร์ดไมโครคอนโทรลเลอร์ ESP32 เป็นส่วนการควบคุมอุปกรณ์ต่าง ๆ ภายในวงจร โดยใช้แหล่งจ่ายไฟ 5V DC ภายในระบบมีการเชื่อมต่อ Relay Module เพื่อควบคุมอุปกรณ์ไฟฟ้าจากแพลตฟอร์ม ดังรูปที่ 3.14 ใช้รีเลย์ในการควบคุมพัดลม ซึ่งมีสายไฟเลี้ยงเชื่อมต่อจากแหล่งจ่ายไฟและสายควบคุมจากขา GPIO ของ ESP32 นอกจากนี้ยังมีปุ่ม EN Reset และปุ่ม Reset Wi-Fi ซึ่งเชื่อมต่อกับขาของ ESP32 โดยปุ่ม EN Reset จะใช้สำหรับรีเซ็ตการทำงานของบอร์ด ขณะที่ปุ่ม Reset Wi-Fi ออกแบบมาเพื่อให้ผู้ใช้สามารถลบค่าการตั้งค่า Wi-Fi และตั้งค่าใหม่ได้ นอกจากนี้ ESP32 ยังสามารถควบคุมสถานะของ LED Status ซึ่งต่อเข้ากับขา GPIO ผ่านสายสัญญาณเพื่อนำมาใช้แสดงผลตามสถานะการทำงานของระบบ



รูปที่ 3.14 การจำลองวงจรอุปกรณ์ควบคุมใด ๆ

3.1.4.2 ตัวอย่างการออกแบบอุปกรณ์รับข้อมูลใด ๆ

ออกแบบการทำงานของอุปกรณ์รับข้อมูลใด ๆ ตรวจจับก๊าซปิโตรเลียมเหลว โดยใช้บอร์ดไมโครคอนโทรลเลอร์ ESP32 โดยมีการับค่าการทำงานของเซนเซอร์ MQ-6 ซึ่งเป็นเซนเซอร์ในการตรวจจับก๊าซปิโตรเลียมเหลวต่าง ๆ ในอากาศ เมื่อตรวจจับก๊าซปิโตรเลียมเหลวได้ Buzzer ที่อยู่ภายในอุปกรณ์จะดังขึ้นเพื่อแจ้งเตือนให้กับผู้ใช้ ภายในวงจรจะมีจ่ายไฟผ่านตัวแปลงไฟจากไฟ 220 โวลต์ เป็นไฟ 9 โวลต์ เพื่อจ่ายเข้าสู่วงจร มี LED RGB ในการแสดงผลสีไฟตามสถานะการเชื่อมต่อ Wi-Fi และมีสวิตช์ต่อเข้ากับขา EN ของบอร์ดไมโครคอนโทรลเลอร์ ESP32 เพื่อรีเซ็ตค่าการทำงานตามการสั่งงานต่อไป โดยวงจรการทำงานของตัวอย่างการออกแบบอุปกรณ์รับข้อมูลใด ๆ มีการเชื่อมต่อ ดังรูปที่ 3.15



รูปที่ 3.15 การจำลองวงจรอุปกรณ์รับข้อมูลใด ๆ

โดยในการออกแบบอุปกรณ์ต่าง ๆ ที่ใช้งานในแพลตฟอร์มร่วมกันกับเว็บแอปพลิเคชัน ต้องมีการอัปโหลดคำสั่งโปรแกรมเพื่อให้เชื่อมต่อและใช้งานฟังก์ชัน มีรายละเอียด ดังนี้

1) นำเข้า (include) ไลบรารีต่าง ๆ ดังรูปที่ 3.16 ที่จำเป็นสำหรับการทำงานของโปรแกรม

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <EEPROM.h>
```

รูปที่ 3.16 คำสั่งโปรแกรมนำเข้าไลบรารี

2) การกำหนดตัวแปรเก็บข้อมูลเพื่อให้ ESP32 สามารถเชื่อมต่อกับเครือข่าย Wi-Fi และ MQTT Broker ได้อย่างถูกต้อง ดังรูปที่ 3.17

```
// WiFi
const char *ssid = "Default";
const char *password = "12345678";

// MQTT Broker
const char *mqtt_broker =
const char *topic = "esp32
const char *mqtt_username
const char *mqtt_password
const int mqtt_port = 8883;
```

.hivemq.cloud";

รูปที่ 3.17 คำสั่งโปรแกรมกำหนดตัวแปรเก็บข้อมูล

3) กำหนดขนาดของ EEPROM ที่ใช้เก็บข้อมูลเป็น 512 ไบต์ และตั้งค่าการเชื่อมต่อ Wi-Fi และ MQTT ผ่านการใช้การเชื่อมต่อแบบปลอดภัย (SSL/TLS) โดยใช้ตัวแปร espClient และ client ดังรูปที่ 3.18 เพื่อการเชื่อมต่อที่ปลอดภัยกับ MQTT broker

```
#define EEPROM_SIZE 512

WiFiClientSecure espClient;
PubSubClient client(espClient);
```

รูปที่ 3.18 คำสั่งโปรแกรมกำหนดขนาด EEPROM และเชื่อมต่อกับ MQTT Broker

4) คำสั่งโปรแกรมเพื่ออ่านข้อมูลจาก EEPROM เพื่อดึงข้อมูลที่เกี่ยวกับชื่อ Wi-Fi และรหัสผ่าน Wi-Fi ที่ถูกบันทึกไว้ใน EEPROM ของ ESP32 ดังรูปที่ 3.19 ซึ่งจะใช้ในการเชื่อมต่อกับเครือข่าย Wi-Fi โดยอัตโนมัติเมื่อเริ่มทำงาน

```
if (!EEPROM.begin(EEPROM_SIZE)) {
    Serial.println("Failed to initialize EEPROM");
    return;
}

// Read WiFi details from EEPROM
String getwfn;
for (int i = 0; true; i++) {
    char c = EEPROM.read(i);
    if (c == '\0') break;
    getwfn += c;
}

// Read getwfp from EEPROM
String getwfp;
int pwdStart = getwfn.length() + 1;
for (int i = pwdStart; true; i++) {
    char c = EEPROM.read(i);
    if (c == '\0') break;
    getwfp += c;
}
```

รูปที่ 3.19 คำสั่งโปรแกรมอ่านข้อมูลจาก EEPROM

5) คำสั่งโปรแกรมใช้เพื่อเชื่อมต่อ ESP32 กับ Wi-Fi ดังรูปที่ 3.20 โดยตรวจสอบข้อมูลชื่อ Wi-Fi และรหัสผ่านจาก EEPROM และทำการเชื่อมต่อกับ Wi-Fi ตามข้อมูลที่บันทึกไว้ หากไม่มีข้อมูล Wi-Fi ใน EEPROM ก็จะแสดงข้อความเพื่อให้ทราบว่ากำลังเชื่อมต่อกับ Wi-Fi ค่าเริ่มต้น

```
// Check Wi-Fi default
if (getwfn.length() > 0 && getwfp.length() > 0) {
    Serial.print("WiFi-name: ");
    Serial.println(getwfn);
    Serial.print("Password: ");
    Serial.println(getwfp);
    Serial.println("\n============  [CUSTOMIZE] New Wi-fi  ==========\n");

    // Connect to WiFi
    WiFi.begin(getwfn, getwfp);
    while (WiFi.status() != WL_CONNECTED) {
        digitalWrite(statusconnect, HIGH);
        delay(500);
        digitalWrite(statusconnect, LOW);
        delay(500);
        Serial.println(">>>>  Connecting to [Local] WiFi..");
    }
    digitalWrite(statusconnect, HIGH);
    Serial.println(">>>>  Connected to the WiFi network");
}
else {
    Serial.println("\n============  [INITIAL] Default Wi-fi  ==========\n");
```

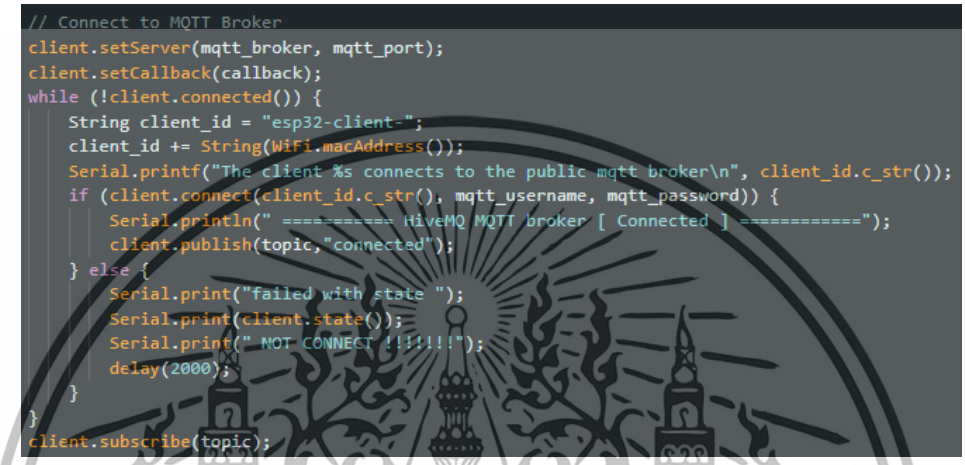
รูปที่ 3.20 คำสั่งโปรแกรมตรวจสอบข้อมูลจาก EEPROM และทำการเชื่อมต่อกับ Wi-Fi

6) คำสั่งโปรแกรมเชื่อมต่อ ESP32 กับ Wi-Fi ดังรูปที่ 3.21 เมื่อเชื่อมต่อสำเร็จจะเปิดการเชื่อมต่อที่ปลอดภัยแบบ TLS/SSL โดยไม่ใช้การตรวจสอบใบรับรอง

```
    // Connect to WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        digitalWrite(statusconnect, HIGH);
        delay(500);
        digitalWrite(statusconnect, LOW);
        delay(500);
        Serial.println("-------  Connecting to WiFi..  -------");
    }
    digitalWrite(statusconnect, HIGH);
    Serial.println("-------  Connected to the WiFi network  -------");
}

// Open TLS/SSL connection
espClient.setInsecure();
```

รูปที่ 3.21 คำสั่งโปรแกรมเชื่อมต่อ ESP32 กับ Wi-Fi

7) คำสั่งโปรแกรมเชื่อมต่อ ESP32 กับ MQTT broker และใช้ client_id ที่สร้างจาก MAC address ของ ESP32 เพื่อระบุตัวตนในการเชื่อมต่อ โดยจะแสดงข้อความสถานะการเชื่อมต่อ และเมื่อเชื่อมต่อสำเร็จจะ Subscribe รับข้อมูลจาก topic และส่งข้อความ Connected เข้าสู่ Broker ดังรูปที่ 3.22



```
// Connect to MQTT Broker
client.setServer(mqtt_broker, mqtt_port);
client.setCallback(callback);
while (!client.connected()) {
    String client_id = "esp32-client-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public mqtt broker\n", client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
        Serial.println(" ========== HiveMQ MQTT broker [ Connected ] ==========");
        client.publish(topic,"connected");
    } else {
        Serial.print("failed with state ");
        Serial.print(client.state());
        Serial.print(" NOT CONNECT !!!!!!");
        delay(2000);
    }
}
client.subscribe(topic);
```

รูปที่ 3.22 คำสั่งโปรแกรมเชื่อมต่อ ESP32 กับ MQTT broker

8) ฟังก์ชันนี้จะทำงานเมื่อได้รับข้อความจาก MQTT broker ที่มีข้อมูล Wi-Fi ใหม่ โดยจะทำการลบข้อมูลเก่าจาก EEPROM โดยการแยกข้อมูล Wi-Fi ใหม่ออกจากข้อความ และแสดงชื่อและรหัสผ่าน Wi-Fi ใหม่ที่ได้รับใน Serial Monitor ดังรูปที่ 3.23

```
void callback(char *topic, byte *payload, unsigned int length) {
  Serial.printf("\nMessage arrived in topic: ");
  Serial.print(topic);
  String message;
  String wfn;
  String wfp;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  if (message.startsWith("wfn")) {
    for (int i = 0; i < EEPROM_SIZE; i++) {
      EEPROM.write(i, 0);
    }
    EEPROM.commit();
    Serial.println("\n\n==========  [ EEPROM Clearing Old data... ]  ==========\n");

    char buffer[100];
    message.toCharArray(buffer, 100);  // Convert String to char array
    char *pair = strtok(buffer, ",");
    char key[50], value[50];
    while (pair != NULL) {
      sscanf(pair, "%49[^:]:%49s", key, value);
      if (strcmp(key, "wfn") == 0) {
        wfn = value;
      } else if (strcmp(key, "wfp") == 0) {
        wfp = value;
      }
      pair = strtok(NULL, ",");
    }
    Serial.printf("[Payload Detail] >> ");
    Serial.print(message);
    Serial.printf("\n New Wi-fi Name : ");
    Serial.print(wfn);
    Serial.printf("\n New Wi-fi Password : ");
    Serial.print(wfp);
```

รูปที่ 3.23 ฟังก์ชันลบข้อมูลเก่าจาก EEPROM และแยกข้อมูล Wi-Fi

9) คำสั่งโปรแกรมนี้จะทำการเขียนข้อมูลชื่อและรหัสผ่าน Wi-Fi ลงใน EEPROM ของ ESP32 หากเขียนข้อมูลสำเร็จแล้ว โดยจะแสดงข้อความใน Serial Monitor ว่าข้อมูลถูกเขียนลง EEPROM สำเร็จ ดังรูปที่ 3.24

```
  for (int i = 0; i < wfn.length(); i++) {
    EEPROM.write(i, wfn[i]);
  }
  EEPROM.write(wfn.length(), '\0');

  // Write wfp to EEPROM starting from address after username
  int pwdStart = wfn.length() + 1;
  for (int i = 0; i < wfp.length(); i++) {
    EEPROM.write(pwdStart + i, wfp[i]);
  }
  EEPROM.write(pwdStart + wfp.length(), '\0');
  EEPROM.commit();

  digitalWrite(statusconnect, LOW);
  for (int i = 0; i < 2; i++) {
    digitalWrite(statusnewwifi, HIGH);
    delay(1000);
    digitalWrite(statusnewwifi, LOW);
    delay(1000);
  }
  digitalWrite(statusconnect, LOW);
  Serial.println("\n\n==========  [ Data written to EEPROM Success!! ]  ==========\n");
}
```

รูปที่ 3.24 คำสั่งโปรแกรมนี้จะทำการเขียนข้อมูล Wi-Fi ใหม่ลงใน EEPROM

10) คำสั่งโปรแกรมนี้จะทำการลบข้อมูลที่เก็บไว้ใน EEPROM เมื่อได้รับคำสั่ง "defaultwifi" ผ่าน MQTT และแสดงข้อความใน Serial Monitor ว่าข้อมูลใน EEPROM ถูกลบสำเร็จแล้ว ดังรูปที่ 3.25

```
if (message.startsWith("defaultwifi")) {
    for (int i = 0; i < EEPROM_SIZE; i++) {
        EEPROM.write(i, 0);
    }
    EEPROM.commit();
    digitalWrite(statusconnect, LOW);
    for (int i = 0; i < 4; i++) {
        digitalWrite(statusconnect, LOW);
        digitalWrite(statusnewwifi, HIGH);
        delay(300);
        digitalWrite(statusconnect, LOW);
        digitalWrite(statusnewwifi, LOW);
        delay(300);
    }
    Serial.println("\n\n==========  [ EEPROM Cleared... ]  ==========\n");
```

รูปที่ 3.25 คำสั่งโปรแกรมนี้จะทำการลบข้อมูลที่เก็บไว้ใน EEPROM เมื่อได้รับคำสั่ง "defaultwifi"

11) คำสั่งโปรแกรมเช็คข้อความที่ได้รับจาก MQTT Broker เริ่มต้นด้วย "ctrl/" หรือไม่ ถ้าใช่จะเข้าสู่ฟังก์ชันในการควบคุมอุปกรณ์ตามฟังก์ชันการทำงานที่ผู้ใช้งานได้สร้างไว้ ดังรูปที่ 3.26

```
if (message.startsWith("ctrl/")) {
    Serial.println("function control...");

    String command = message;

  if (message == "ctrl/red_on") {
      controlRelayAndLight(true, true, false, false);
  }
  else if (message == "ctrl/green_on") {
      controlRelayAndLight(true, false, true, false);
  }
  else if (message == "ctrl/blue_on") {
      controlRelayAndLight(true, false, false, true);
  }
```

รูปที่ 3.26 คำสั่งโปรแกรมเช็คข้อความที่ได้รับจาก MQTT Broker เริ่มต้นด้วย "ctrl/"

12) คำสั่งโปรแกรมนี้จะส่งข้อความ "online" ไปยัง MQTT broker ทุก ๆ 2 วินาที เพื่อแจ้งว่า ESP32 ยังคงเชื่อมต่อและทำงานได้อยู่ โดยใช้เวลาจากฟังก์ชัน millis() เพื่อคำนวณระยะเวลาและควบคุมการส่งข้อความ ดังรูปที่ 3.27

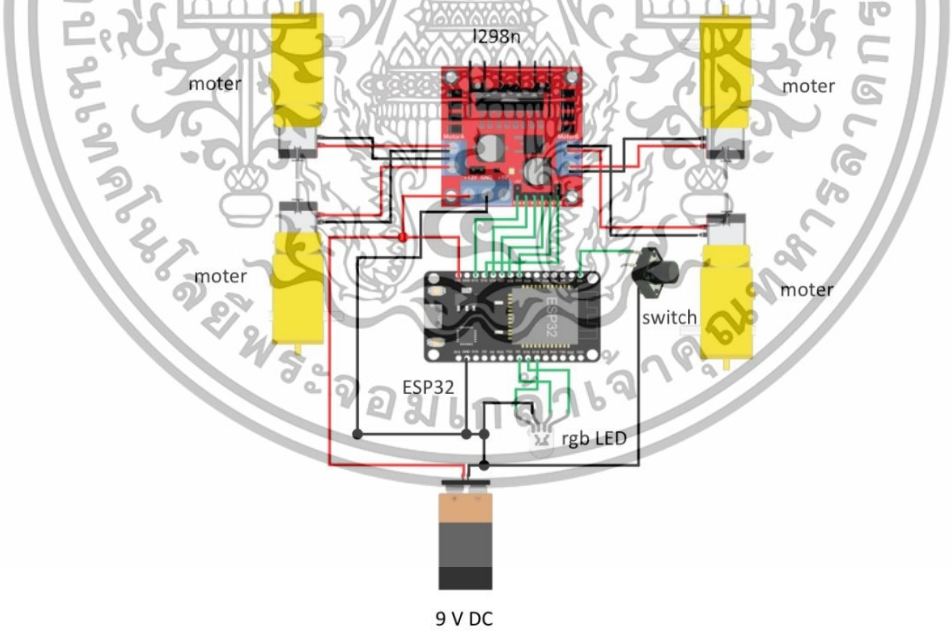
```
void loop()
{

  //Sending Online Status
  static unsigned long lastHeartbeat = 0;
  unsigned long now = millis();
  if (now - lastHeartbeat > 2000) {
    client.publish(topic, "online");
    lastHeartbeat = now;
  }
```

รูปที่ 3.27 คำสั่งโปรแกรมจะส่งข้อความ "online" ไปยัง MQTT broker

ส่วนของอุปกรณ์ตัวอย่างที่ออกแบบมาเพื่อแสดงเป็นตัวอย่างในการทำงานและการเชื่อมต่อกับแพลตฟอร์ม ประกอบไปด้วย รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ โดยมีตัวอย่างการออกแบบ ดังนี้

3.1.4.3 ตัวอย่างการออกแบบรถยนต์บังคับ

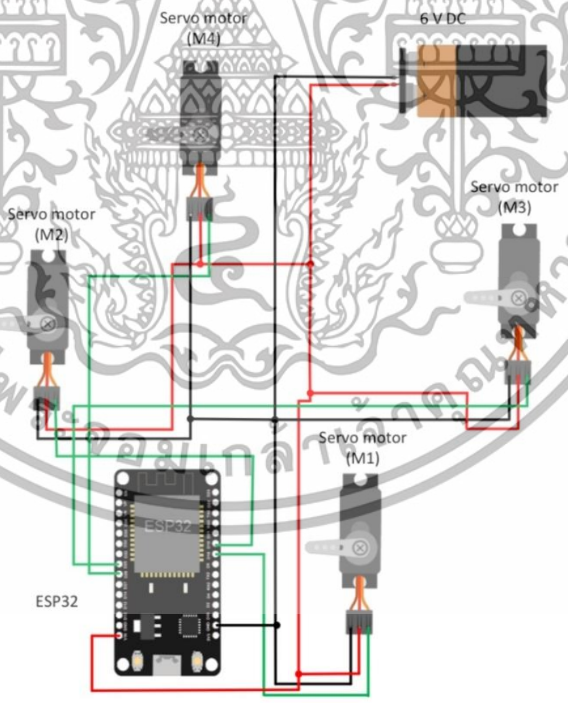
ออกแบบการทำงานของรถบังคับ โดยใช้บอร์ดไมโครคอนโทรลเลอร์ ESP32 ในการควบคุมการทำงานของโมดูล L298N เพื่อควบคุมมอเตอร์ DC สำหรับการเคลื่อนที่ของรถบังคับ โดยสามารถควบคุมทิศทางและความเร็วของมอเตอร์ได้ทั้งสี่ตัว มอเตอร์ทั้งสองตัวสามารถขับได้ในสองทิศทาง โดยการเปลี่ยนทิศทางจะทำผ่านการควบคุมสัญญาณไฟฟ้าผ่านขา IN1 IN2 และ IN3 IN4 ของโมดูล L298N นอกจากนี้ ยังสามารถควบคุมความเร็วของมอเตอร์ได้โดยใช้ PWM (Pulse Width Modulation) ผ่านขา ENA และ ENB ของ L298N และจ่ายไฟจากถ่านอัลคาไลน์ 9 โวลต์ จ่ายไฟให้กับวงจร มี LED RGB ในการแสดงผลสีไฟตามสถานะการเชื่อมต่อ Wi-Fi และมีสวิตช์ต่อเข้ากับขา EN ของบอร์ดไมโครคอนโทรลเลอร์ ESP32 เพื่อรีเซ็ตค่าการทำงานตามการสั่งงานต่อไป โดยวงจรการทำงานของรถยนต์บังคับมีการเชื่อมต่อ ดังรูปที่ 3.28



รูปที่ 3.28 การจำลองวงจรรถยนต์บังคับ

3.1.4.4 ตัวอย่างการออกแบบแขนกล

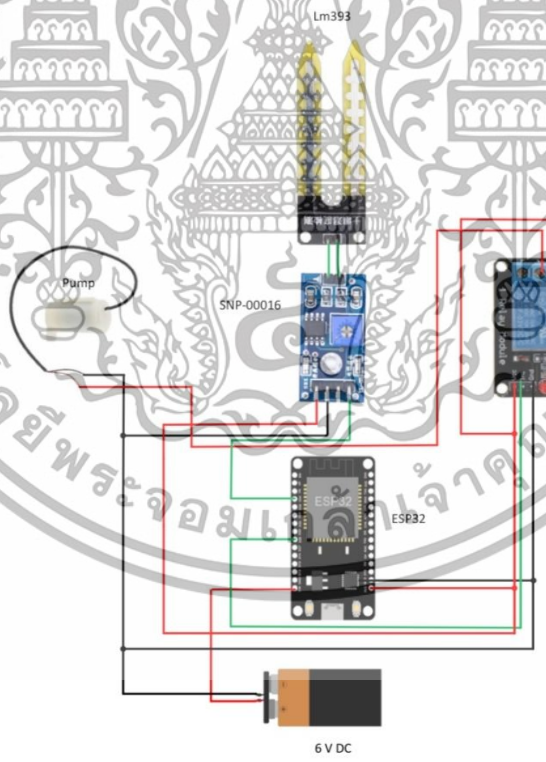
การออกแบบการทำงานของแขนกลที่ควบคุมด้วยบอร์ไมโครคอนโทรลเลอร์ ESP32 เซอร์โวมอเตอร์ 180 องศา 1 ตัวและเซอร์โวมอเตอร์ 360 องศา 3 ตัว สามารถแบ่งออกเป็นสองส่วนหลัก ๆ คือการควบคุมการเคลื่อนไหวของแขนและมือจับ รวมถึงการหมุนฐานของแขนกล ESP32 ทำหน้าที่ควบคุมเซอร์โวมอเตอร์ทั้ง 4 ตัว เซอร์โวมอเตอร์ตัวแรกถูกใช้เป็นมือจับในการหยิบจับวัตถุ โดยสามารถหมุนได้ในช่วง 0-180 องศา เพื่อเปิดและปิดมือจับ เซอร์โวมอเตอร์อีก 2 ตัว หมุนได้ในช่วง 0-360 องศา ควบคุมการเคลื่อนไหวของแขนทั้งสองข้าง ทำให้แขนกลสามารถยกขึ้นและลงได้เพื่อปรับทิศทางของแขน เซอร์โวมอเตอร์ทั้งสี่ตัวนี้จะทำงานร่วมกัน เพื่อให้แขนกลสามารถหยิบจับวัตถุและเคลื่อนย้ายวัตถุได้ สำหรับฐานของแขนกลจะใช้เซอร์โวมอเตอร์ 360 องศา ซึ่งสามารถหมุนได้รอบทิศทางเต็ม 360 องศา และภายในวงจรจะมีจ่ายไฟจากถ่านอัลคาไลน์ 9 โวลต์ จ่ายไฟให้กับวงจร มี LED RGB ในการแสดงผลสีไฟตามสถานะการเชื่อมต่อ Wi-Fi และมีสวิตช์ต่อเข้ากับขา EN ของบอร์ดไมโครคอนโทรลเลอร์ ESP32 เพื่อรีเซ็ตค่าการทำงานตามการสั่งงานต่อไปโดยวงจรการทำงานของแขนกลมีการเชื่อมต่อ ดังรูปที่ 3.29



รูปที่ 3.29 การจำลองวงจรแขนกล

3.1.4.5 ตัวอย่างการออกแบบอุปกรณ์รดน้ำต้นไม้

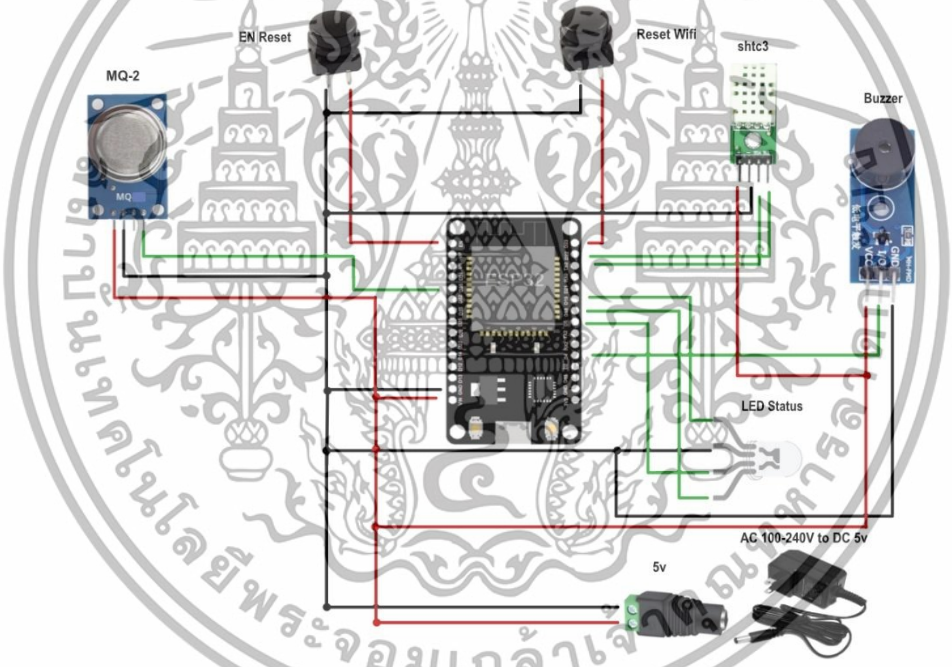
ออกแบบการทำงานของอุปกรณ์รดน้ำต้นไม้โดยใช้บอร์ดไมโครคอนโทรลเลอร์ ESP32 ในการควบคุมการทำงานรีเลย์ 3.3V เพื่อควบคุมปั๊มน้ำขนาดเล็ก ในการรดน้ำต้นไม้ โดยจะต่อเซนเซอร์วัดความชื้นในดินเข้ากับบอร์ดไมโครคอนโทรลเลอร์ ESP32 จะอ่านค่าความชื้นจากเซนเซอร์ในรูปแบบแอนะล็อก และทำการแมพค่าให้เป็นเปอร์เซ็นต์ โดยค่าความชื้นในดินจะอยู่ในช่วง 0% (ดินแห้งสนิท) ถึง 100% (ดินชื้นเต็มที่) โดยมีเงื่อนไขเมื่อค่าความชื้นในดินต่ำกว่า 30% ESP32 จะสั่งให้รีเลย์ทำงานและเปิดปั๊มน้ำเพื่อรดน้ำต้นไม้ เมื่อน้ำถูกปั๊มเข้าสู่ดินจนค่าความชื้นสูงกว่า 30% ระบบจะสั่งให้รีเลย์ปิดปั๊มน้ำเพื่อหยุดการรด และสามารถสั่งงานผ่านเว็บแอปพลิเคชันจากผู้ใช้ และภายในวงจรจะมีจ่ายไฟจากถ่านอัลคาไลน์ 9 โวลต์จ่ายไฟให้กับวงจร มี LED RGB ในการแสดงผลสีไฟตามสถานะการเชื่อมต่อ Wi-Fi และมีสวิตช์ต่อเข้ากับขา EN ของบอร์ดไมโครคอนโทรลเลอร์ ESP32 เพื่อรีเซ็ตค่าการทำงานตามการสั่งงานต่อไป โดยวงจรการทำงานของอุปกรณ์รดน้ำต้นไม้มีการเชื่อมต่อ ดังรูปที่ 3.30



รูปที่ 3.30 การจำลองวงจรอุปกรณ์รดน้ำต้นไม้

3.1.4.6 ตัวอย่างการออกแบบอุปกรณ์ตรวจจับควันไฟ

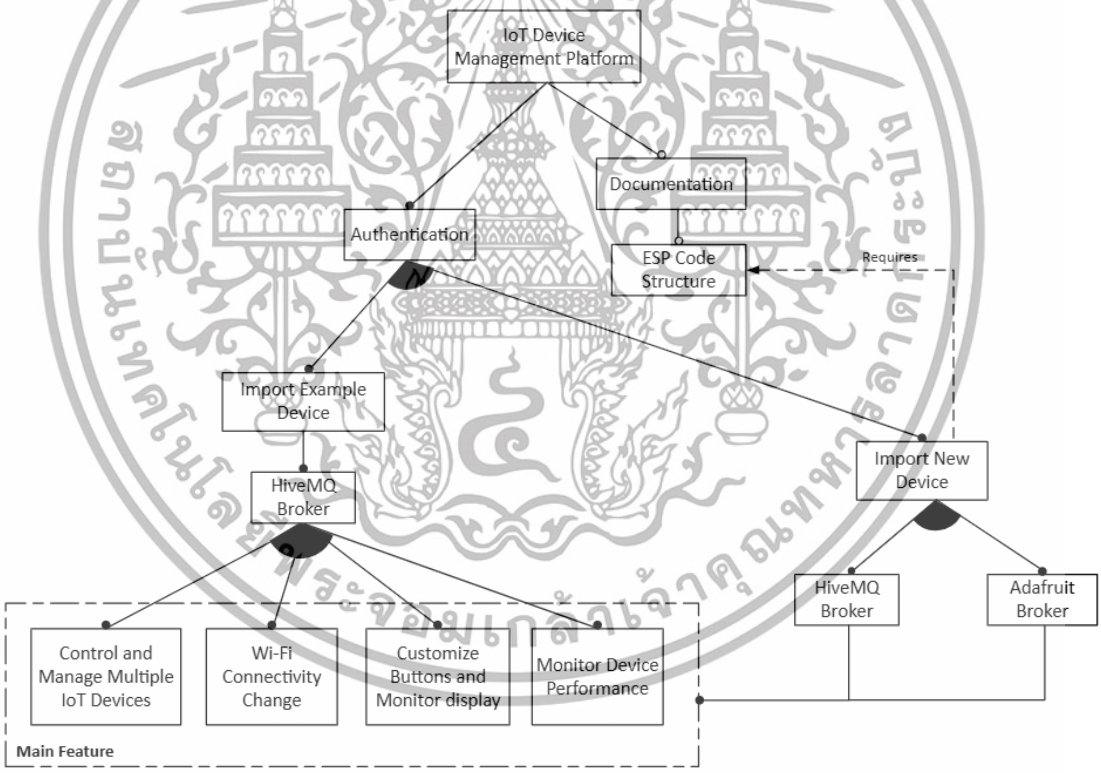
ออกแบบการทำงานของอุปกรณ์ตรวจจับควันไฟโดยใช้บอร์ดไมโครคอนโทรลเลอร์ ESP32 โดยมีการรับค่าการทำงานของเซนเซอร์ SHTC3 ซึ่งเป็นเซนเซอร์ในการวัดอุณหภูมิและความชื้น และมีเซนเซอร์ MQ-2 ในการตรวจจับควันไฟที่เกิดจากการเผาไหม้ เมื่อตรวจจับควันไฟ Buzzer ที่อยู่ภานในอุปกรณ์จะดังขึ้นเพื่อแจ้งเตือนให้กับผู้ใช้ ภายในวงจรจะมีจ่ายไฟผ่านตัวแปลงไฟจากไฟ 220 โวลต์ เป็นไฟ 9 โวลต์ เพื่อจ่ายเข้าสู่วงจร มี LED RGB ในการแสดงผลสีไฟตามสถานะการเชื่อมต่อ Wi-Fi และมีสวิตช์ต่อเข้ากับขา EN ของบอร์ดไมโครคอนโทรลเลอร์ ESP32 เพื่อรีเซ็ตค่าการทำงานตามการสั่งงานต่อไป โดยวงจรการทำงานของอุปกรณ์ตรวจจับควันไฟมีการเชื่อมต่อ ดังรูปที่ 3.31



รูปที่ 3.31 การจำลองวงจรอุปกรณ์ตรวจจับควันไฟ

3.2 การออกแบบแพลตฟอร์มตัวกลางควบคุมและจัดการอุปกรณ์ IoT

แพลตฟอร์มตัวกลางสำหรับการควบคุมและจัดการอุปกรณ์ Internet of Things (IoT) มีแนวคิดในการออกแบบมาเพื่อเป็นตัวกลางในการบริหารจัดการอุปกรณ์ IoT อย่างมีประสิทธิภาพ ลดความซับซ้อนในการใช้งาน และอำนวยความสะดวกให้กับผู้ใช้ผ่านแพลตฟอร์มเดียว แทนที่จะต้องใช้งานแอปพลิเคชันหรือเครื่องมือหลายตัว แพลตฟอร์มนี้ช่วยให้สามารถเชื่อมต่อและควบคุมอุปกรณ์ IoT ได้จากทุกที่ผ่านเครือข่ายอินเทอร์เน็ต ด้วยการรองรับการทำงานร่วมกับระบบ Cloud ซึ่งช่วยให้สามารถจัดการอุปกรณ์ที่อยู่ในเครือข่ายเฉพาะที่ (Local Network) ได้อย่างสะดวกยิ่งขึ้น โดยมีคุณสมบัติการทำงานภายในโครงสร้างของแพลตฟอร์ม ดังรูปที่ 3.32 ซึ่งแสดงให้เห็นถึงฟังก์ชันหลักที่แพลตฟอร์มรองรับ รวมถึงการเชื่อมโยงระหว่างองค์ประกอบต่าง ๆ เพื่อให้ผู้ใช้งานสามารถเข้าถึงและจัดการอุปกรณ์ได้อย่างเป็นระบบ

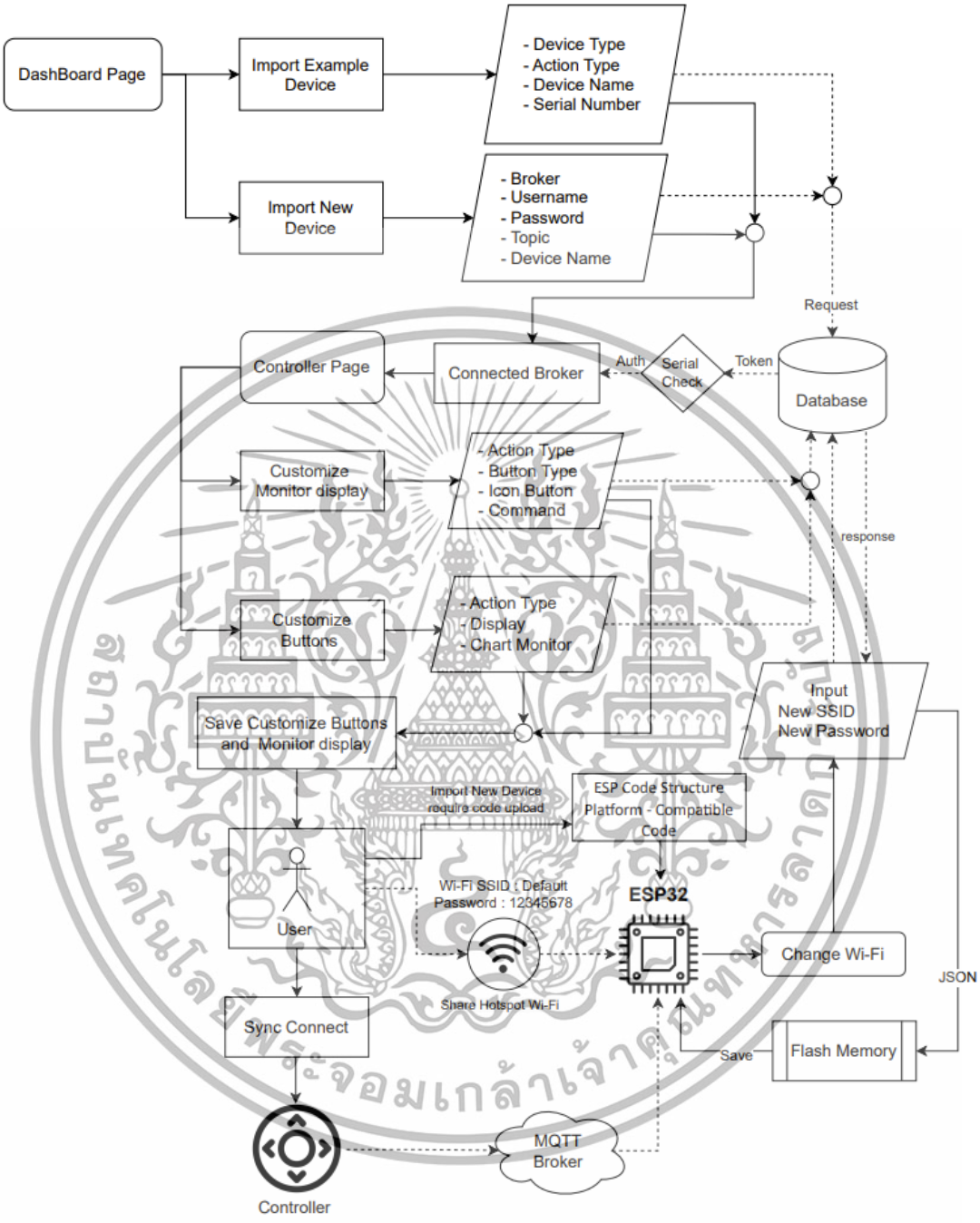


รูปที่ 3.32 แผนภาพคุณสมบัติภายในเว็บแอปพลิเคชัน

ฟังก์ชันหลักของแพลตฟอร์ม ประกอบด้วย Authentication ซึ่งเป็นกระบวนการยืนยันตัวตนของผู้ใช้ก่อนเข้าสู่ระบบ เพื่อเพิ่มความปลอดภัยและป้องกันการเข้าถึงอุปกรณ์โดยไม่ได้รับอนุญาต นอกจากนี้ยังมี Documentation ที่ให้ข้อมูลและแนวทางการใช้งานแพลตฟอร์ม เพื่อช่วยให้ผู้ใช้สามารถเรียนรู้และใช้งานระบบได้ง่ายขึ้น หนึ่งในฟังก์ชันที่สำคัญ คือ Import Example Device ซึ่งเป็นฟังก์ชันในการควบคุมอุปกรณ์ภายในแพลตฟอร์ม ได้แก่ รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ โดยผู้ใช้สามารถเพิ่มอุปกรณ์ IoT และกำหนดค่าการใช้งานได้ ฟังก์ชันนี้รองรับความสามารถต่าง ๆ เช่น การควบคุมและจัดการอุปกรณ์ IoT หลายตัวพร้อมกัน (Control and Manage Multiple IoT Devices) การเปลี่ยนแปลงการเชื่อมต่อ Wi-Fi (Wi-Fi Connectivity Change) ในกรณีที่ผู้ใช้งานต้องการเปลี่ยนเครือข่ายในการเชื่อมต่อเครือข่าย Wi-Fi สามารถเปลี่ยนการเชื่อมต่อได้ภายในเว็บแอปพลิเคชัน การปรับแต่งปุ่มควบคุมและหน้าจอแสดงผล (Customize Buttons and Monitor Display) เป็นฟังก์ชันการทำงานที่ออกแบบมาให้ผู้ใช้งานสามารถปรับเปลี่ยนส่วนควบคุมและการแสดงผลตามอุปกรณ์ได้ตามความต้องการของผู้ใช้ และการตรวจสอบการทำงานของอุปกรณ์ (Monitor Device Performance) โดยฟังก์ชันทั้งหมดเหล่านี้ช่วยให้ผู้ใช้สามารถจัดการอุปกรณ์ได้อย่างมีประสิทธิภาพ ลดความยุ่งยากในการตั้งค่า และเพิ่มความสะดวกในการควบคุมอุปกรณ์จากตัวกลาง นอกจากนี้ยังมีฟังก์ชันที่ช่วยให้แพลตฟอร์มมีความยืดหยุ่นมากขึ้น คือ การนำเข้าอุปกรณ์จากภายนอก (Import New Device) ในกรณีที่ผู้ใช้ต้องการใช้งานอุปกรณ์ IoT ที่ตนเองพัฒนา ซึ่งต้องรองรับการเชื่อมต่อผ่านโปรโตคอล MQTT โดยใช้ HiveMQ Broker หรือ Adafruit Broker และต้องทำการอัปโหลดโค้ดโปรแกรมที่ใช้งานร่วมกันกับแพลตฟอร์มได้ ทำให้ระบบสามารถขยายขอบเขตการใช้งานได้โดยไม่ถูกจำกัดเฉพาะอุปกรณ์ที่เชื่อมต่อโดยตรงกับแพลตฟอร์ม และยังสามารถใช้งานคุณสมบัติต่าง ๆ ภายในแพลตฟอร์มได้เหมือนกับอุปกรณ์ภายในแพลตฟอร์มทุกประการ

จึงได้ออกแบบระบบการทำงานโดยประกอบด้วย 3 ส่วน คือ อุปกรณ์ IoT ระบบฐานข้อมูล และเว็บแอปพลิเคชัน เพื่อให้ผู้ใช้สามารถควบคุมการทำงานและสั่งการอุปกรณ์ IoT ได้ทันทีโดยผ่านเว็บแอปพลิเคชัน จึงได้ออกแบบระบบการทำงาน โดยมีขั้นตอนการทำงานของระบบตามแผนภาพเริ่มจากหน้าแดชบอร์ดที่ผู้ใช้เลือกเพิ่มอุปกรณ์ (Import Example Device) หรือเชื่อมต่ออุปกรณ์จากภายนอก (Import New Device) ผ่าน MQTT Broker คือ HiveMQ หรือ Adafruit ผู้ใช้ต้องป้อนข้อมูลสำคัญ เช่น ประเภทอุปกรณ์ (Device Type) ประเภทของการทำงาน (Action Type) ชื่ออุปกรณ์ (Device Name) และหมายเลขประจำอุปกรณ์ (Serial ID) เพื่อให้ระบบสามารถลงทะเบียนอุปกรณ์ในฐานข้อมูลได้อย่างถูกต้อง โดยในการเพิ่มอุปกรณ์ต้องกรอก

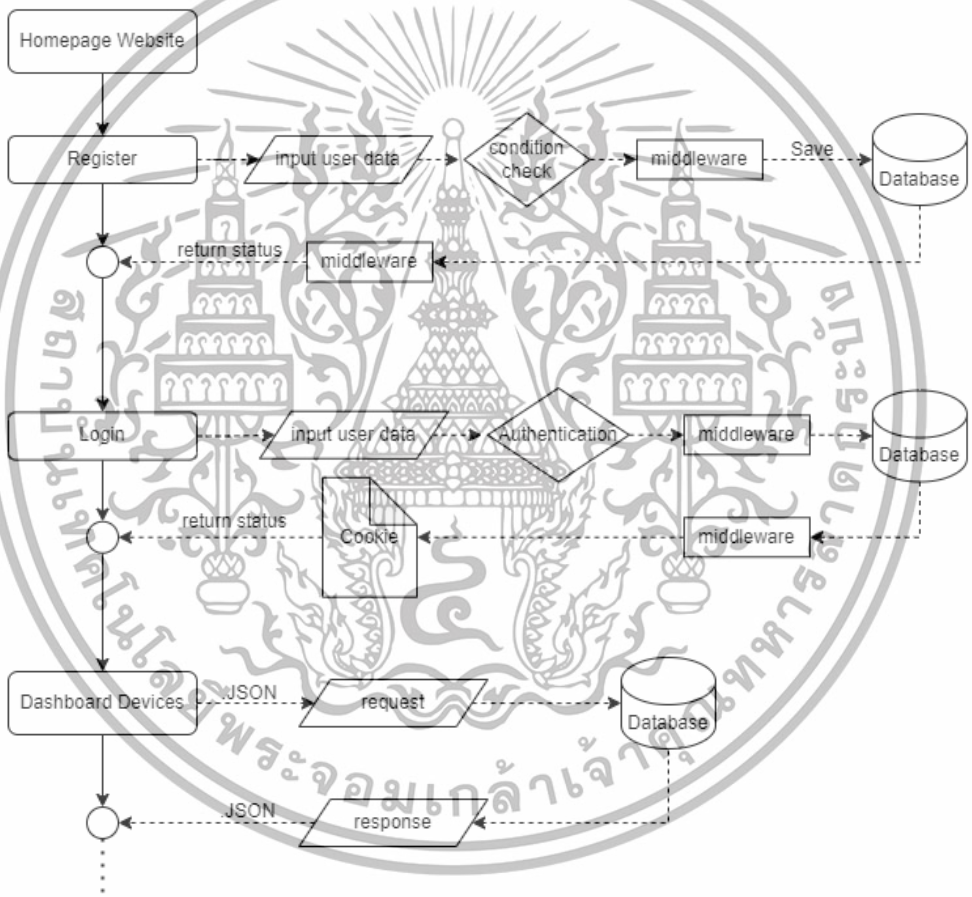
ข้อมูลประเภท หมายเลขเครื่อง และชื่ออุปกรณ์ในขั้นตอนนี้ หลังจากกรอกข้อมูลเสร็จแล้ว ระบบจะส่งคำขอไปยังฐานข้อมูลเพื่อตรวจสอบหมายเลขเครื่องที่ลงทะเบียนไว้ หากการตรวจสอบสำเร็จ ระบบจะส่งข้อมูลการเชื่อมต่อให้กับผู้ใช้งาน ผู้ใช้สามารถเข้าสู่ควบคุมอุปกรณ์ (Controller Page) เพื่อกำหนดค่าการควบคุมอุปกรณ์ IoT ได้อย่างละเอียด โดยสามารถปรับแต่งหน้าแสดงผลการทำงาน (Monitor Display) เพื่อกำหนดประเภทของการแสดงผล ไอคอนปุ่ม และคำสั่งต่าง ๆ รวมถึงสามารถปรับแต่งปุ่มควบคุม (Customize Buttons) ให้เหมาะสมกับการใช้งาน โดยสามารถกำหนดประเภทของการทำงานของอุปกรณ์ IoT ชนิดนั้น ๆ รูปแบบการแสดงผล และการใช้แผนภูมิติดตามข้อมูล เมื่อการตั้งค่าทั้งหมดเสร็จสมบูรณ์ ระบบจะบันทึกการเปลี่ยนแปลงลงฐานข้อมูลและอัปเดตข้อมูลไปยังอุปกรณ์ที่เชื่อมต่อ โดยค่าเริ่มต้นของ Wi-Fi ในอุปกรณ์จะถูกตั้งค่าไว้เป็นชื่อ “Default” และรหัสผ่าน “12345678” เพื่อความสะดวกในการเชื่อมต่อครั้งแรก จากนั้นผู้ใช้ต้องแชร์สัญญาณ Wi-Fi Hotspot โดยตั้งชื่อและรหัสผ่านให้ตรงกับที่อุปกรณ์ตั้งไว้ เพื่อให้การเชื่อมต่อเบื้องต้นสำเร็จ เมื่อผู้ใช้เชื่อมต่อกับอุปกรณ์ได้แล้ว จะสามารถทำการผูกอุปกรณ์เข้ากับบัญชีในเว็บแอปพลิเคชันได้ เมื่อขั้นตอนนี้เสร็จสิ้น ผู้ใช้สามารถแก้ไขและตั้งค่าชื่อ Wi-Fi และรหัสผ่านใหม่ได้ตามต้องการผ่านหน้าเว็บแอปพลิเคชัน ข้อมูลการตั้งค่านี้จะถูกบันทึกลงใน EEPROM ของอุปกรณ์ เมื่อผู้ใช้บันทึกการเปลี่ยนแปลงเสร็จแล้ว จะต้องกดปุ่มรีเซ็ตที่ตัวอุปกรณ์เพื่อเริ่มการเชื่อมต่อกับ Wi-Fi ใหม่ที่เพิ่งตั้งค่า ในกรณีที่ผู้ใช้ต้องการรีเซ็ตค่า Wi-Fi กลับไปเป็นค่าเริ่มต้น สามารถกดปุ่ม Reset Wi-Fi ที่ตัวอุปกรณ์หรือเลือกตัวเลือกรีเซ็ตผ่านหน้าเว็บแอปพลิเคชันได้ การรีเซ็ตนี้จะทำให้ชื่อและรหัสผ่าน Wi-Fi กลับไปเป็นค่าเริ่มต้นตามที่กำหนดไว้ ซึ่งการออกแบบขั้นตอนเหล่านี้พิจารณาจากปัญหาที่อาจเกิดขึ้นเมื่อผู้ใช้นำอุปกรณ์ไปใช้งานในเครือข่ายที่มีชื่อและรหัสผ่าน Wi-Fi แตกต่างกัน ขณะเดียวกัน อุปกรณ์จำเป็นต้องมีการเชื่อมต่อกับ Wi-Fi เพื่อใช้เป็นช่องทางหลักในการเชื่อมต่อกับ Cloud Broker โดยอุปกรณ์นี้จะเชื่อมต่อกับ MQTT Broker ซึ่งทำหน้าที่เป็นตัวกลางในการรับส่งข้อมูลระหว่างอุปกรณ์และเว็บแอปพลิเคชัน ในการควบคุมอุปกรณ์ผ่านเว็บแอปพลิเคชัน ผู้ใช้สามารถสั่งงานอุปกรณ์ผ่านหน้า Controller Page ของเว็บแอปพลิเคชัน ซึ่งสามารถส่งคำสั่งไปยัง MQTT Broker ในรูปแบบข้อความ JSON เมื่ออุปกรณ์ ESP32 ที่เชื่อมต่อกับ Broker ได้รับคำสั่ง จะประมวลผลคำสั่งและทำงานตามที่ระบุ เช่น การควบคุมทิศทางหรือการเปลี่ยนค่าอุปกรณ์ต่าง ๆ การสื่อสารระหว่างเว็บแอปพลิเคชันและ ESP32 ผ่าน MQTT นี้ ช่วยให้การควบคุมอุปกรณ์เป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ โดยทำให้ผู้ใช้สามารถจัดการและควบคุมอุปกรณ์ได้จากระยะไกลผ่านอินเทอร์เน็ต โดยมีการใช้งานภายในระบบ ดังรูปที่ 3.33



รูปที่ 3.33 ขั้นตอนการใช้งานภายในระบบ

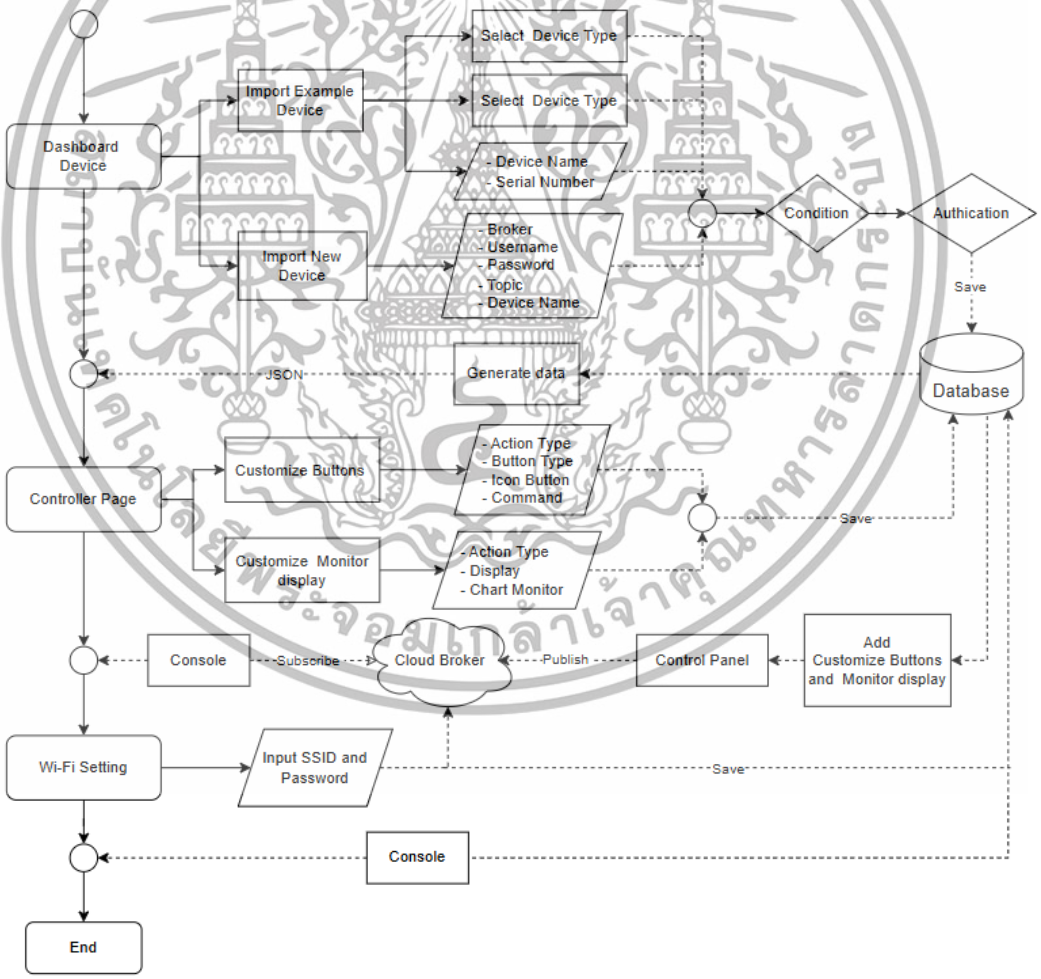
3.2.1 การทำงานและฟังก์ชันการใช้งานเว็บแอปพลิเคชัน

การออกแบบการทำงานของเว็บแอปพลิเคชันสำหรับการจัดการอุปกรณ์ IoT เริ่มจากการทำให้ผู้ใช้สามารถลงทะเบียน เข้าสู่ระบบ และใช้งานได้ โดยผู้ใช้จะต้องลงทะเบียนเพื่อสร้างบัญชีผู้ใช้งาน กรอกข้อมูล และเข้าสู่ระบบเพื่อเข้าถึงฟีเจอร์ต่าง ๆ เช่น การควบคุมอุปกรณ์ การเพิ่มอุปกรณ์ใหม่ และการตั้งค่าเครือข่ายของอุปกรณ์ ทุกขั้นตอนจะถูกประมวลผลร่วมกับฐานข้อมูลและใช้โปรโตคอลการสื่อสารอย่าง MQTT ในการควบคุมอุปกรณ์ผ่าน Cloud Broker โดยมีกระบวนการทำงานดังรูปที่ 3.34



รูปที่ 3.34 กระบวนการทำงานของเว็บแอปพลิเคชัน

เริ่มต้นด้วยการที่ผู้ใช้งานเข้าสู่เว็บไซต์และทำการลงทะเบียน (Sign up) โดยผู้ใช้ต้องกรอกข้อมูลส่วนตัว จากนั้นระบบจะทำการตรวจสอบความถูกต้องของข้อมูลที่ได้รับผ่านเงื่อนไขต่างๆ ถ้าผ่านการตรวจสอบ ข้อมูลของผู้ใช้จะถูกส่งผ่าน middleware เพื่อบันทึกลงในฐานข้อมูล หลังจากนั้นระบบจะส่งสถานะการลงทะเบียนกลับไปยังผู้ใช้ผ่าน middleware ขั้นตอนต่อไปคือการเข้าสู่ระบบ (Login) ผู้ใช้จะต้องกรอกข้อมูลเข้าสู่ระบบ ซึ่งระบบจะทำการตรวจสอบสิทธิ์ผ่านกระบวนการ Authentication และใช้ cookie สำหรับการจัดการเซสชัน ถ้าการตรวจสอบผ่านข้อมูลผู้ใช้จะถูกส่งไปยัง middleware และสถานะการเข้าสู่ระบบจะถูกส่งกลับมายังผู้ใช้ เมื่อเข้าสู่ระบบสำเร็จ ผู้ใช้สามารถเข้าถึงหน้าควบคุมอุปกรณ์ (Dashboard Devices) ซึ่งการสื่อสารระหว่างหน้าควบคุมอุปกรณ์กับฐานข้อมูลจะใช้ข้อมูลในรูปแบบ JSON โดยการร้องขอ (request) และตอบกลับ (response) ข้อมูลกับฐานข้อมูล



รูปที่ 3.34 กระบวนการทำงานของเว็บแอปพลิเคชัน (ต่อ)

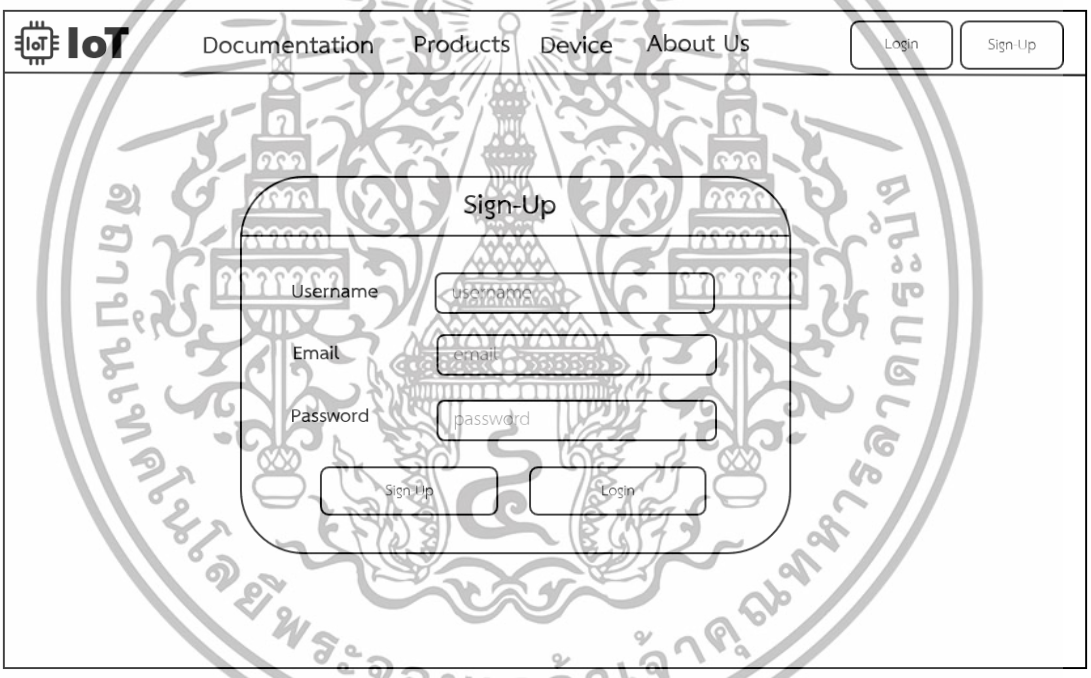
หลังจากเข้าสู่หน้า Dashboard Device หน้าหลักของระบบสำหรับการจัดการอุปกรณ์ ซึ่งมีสองทางเลือกหลักคือ การเพิ่มอุปกรณ์ใหม่ (Import Example Device) และการนำเข้าอุปกรณ์จากภายนอก (Import New Device) หากผู้ใช้เลือกเพิ่มอุปกรณ์ใหม่ (Import Example Device) ผู้ใช้ต้องเลือกประเภทของอุปกรณ์ที่ต้องการเพิ่มป้อนข้อมูลอุปกรณ์ ได้แก่ ชื่ออุปกรณ์ (Device Name) และหมายเลขประจำอุปกรณ์ (Serial ID) และเลือกประเภทอุปกรณ์ และการใช้งานตามประเภทอุปกรณ์ หลังจากนั้นระบบตรวจสอบเงื่อนไข (Condition) หากข้อมูลถูกต้อง ระบบจะดำเนินกระบวนการรับรองสิทธิ์ (Authentication) เมื่อผ่านการตรวจสอบแล้ว ระบบจะบันทึกข้อมูลอุปกรณ์ลงในฐานข้อมูล หากเลือกการนำเข้าอุปกรณ์จากภายนอก (Import New Device) ผู้ใช้ต้องเลือกประเภทของอุปกรณ์ และป้อนข้อมูลการเชื่อมต่อ ได้แก่ ที่อยู่ของ Broker ชื่อผู้ใช้ (Username) และรหัสผ่าน (Password) หัวข้อสำหรับส่งข้อมูล (Topic) ชื่ออุปกรณ์ จากนั้นระบบทำการสร้างข้อมูลในรูปแบบ JSON และส่งไปยังฐานข้อมูล เมื่อผู้ใช้เข้าสู่หน้าควบคุมอุปกรณ์ (Controller Page) ผู้ใช้สามารถกำหนดค่าปุ่มควบคุม (Customize Buttons) โดยกำหนดประเภทของปุ่ม (Button Type) ไอคอนของปุ่ม (Icon Button) และคำสั่งที่ต้องการให้ปุ่มทำงาน (Command) ผู้ใช้สามารถกำหนดค่าการแสดงผล (Customize Monitor Display) โดยเลือกประเภทของข้อมูลที่จะแสดง (Action Type) ประเภทของจอแสดงผล (Display) และรูปแบบของกราฟที่ใช้แสดงผล (Chart Monitor) ข้อมูลที่ตั้งค่าไว้จะถูกบันทึกลงฐานข้อมูล หลักจากนั้นจะแสดงผลข้อมูลจากอุปกรณ์ และแสดงปุ่มควบคุมที่ได้รับการตั้งค่าจากผู้ใช้ เมื่อมีการเพิ่มปุ่มควบคุมหรือจอแสดงผลใหม่ ระบบจะทำการบันทึกค่าเหล่านี้ลงในฐานข้อมูลและในการสื่อสารเพื่อให้สามารถควบคุมอุปกรณ์ได้จะต้องสื่อสารกันผ่าน Cloud Broker ทำหน้าที่เป็นศูนย์กลางสำหรับส่งและรับข้อมูลระหว่างเว็บแอปพลิเคชันและอุปกรณ์ IoT ผู้ใช้สามารถดูข้อมูลที่ได้รับผ่าน Console ซึ่งจะแสดงค่าที่อุปกรณ์ส่งมา โดยการ Subscribe จากนั้นระบบสามารถส่งคำสั่งไปยังอุปกรณ์ผ่าน Cloud Broker และทำการ Publish ผ่าน Control Panel ผ่านหน้าต่างควบคุมอุปกรณ์ เมื่อผู้ใช้ต้องการเปลี่ยนการเชื่อมต่อโดยการตั้งค่า Wi-Fi (Wi-Fi Setting) ผู้ใช้สามารถป้อน SSID และรหัสผ่านของเครือข่าย Wi-Fi ระบบจะส่งข้อมูลนี้ไปยังอุปกรณ์เพื่อให้สามารถเชื่อมต่ออินเทอร์เน็ต ผู้ใช้สามารถตรวจสอบสถานะการเชื่อมต่อผ่าน Console เมื่อผู้ใช้ตั้งค่าและควบคุมอุปกรณ์เสร็จแล้ว เมื่อผู้ใช้ดำเนินการทั้งหมดเสร็จสิ้น ระบบจะอยู่ในสถานะพร้อมทำงาน โดยสามารถควบคุมและติดตามสถานะอุปกรณ์ผ่านเว็บแอปพลิเคชันได้ทันที

3.2.1.1 การออกแบบหน้าเว็บแอปพลิเคชัน (Front-End)

หน้าเว็บแอปพลิเคชันเป็นส่วนที่ใกล้ชิดกับผู้ใช้งานมากที่สุด โดยเป็นส่วนหน้าที่รับคำสั่งมาจากผู้ใช้งาน และสั่งงานมาที่ระบบจัดการเว็บแอปพลิเคชันต่อไป จึงมีการออกแบบหน้าเว็บแอปพลิเคชัน ดังนี้

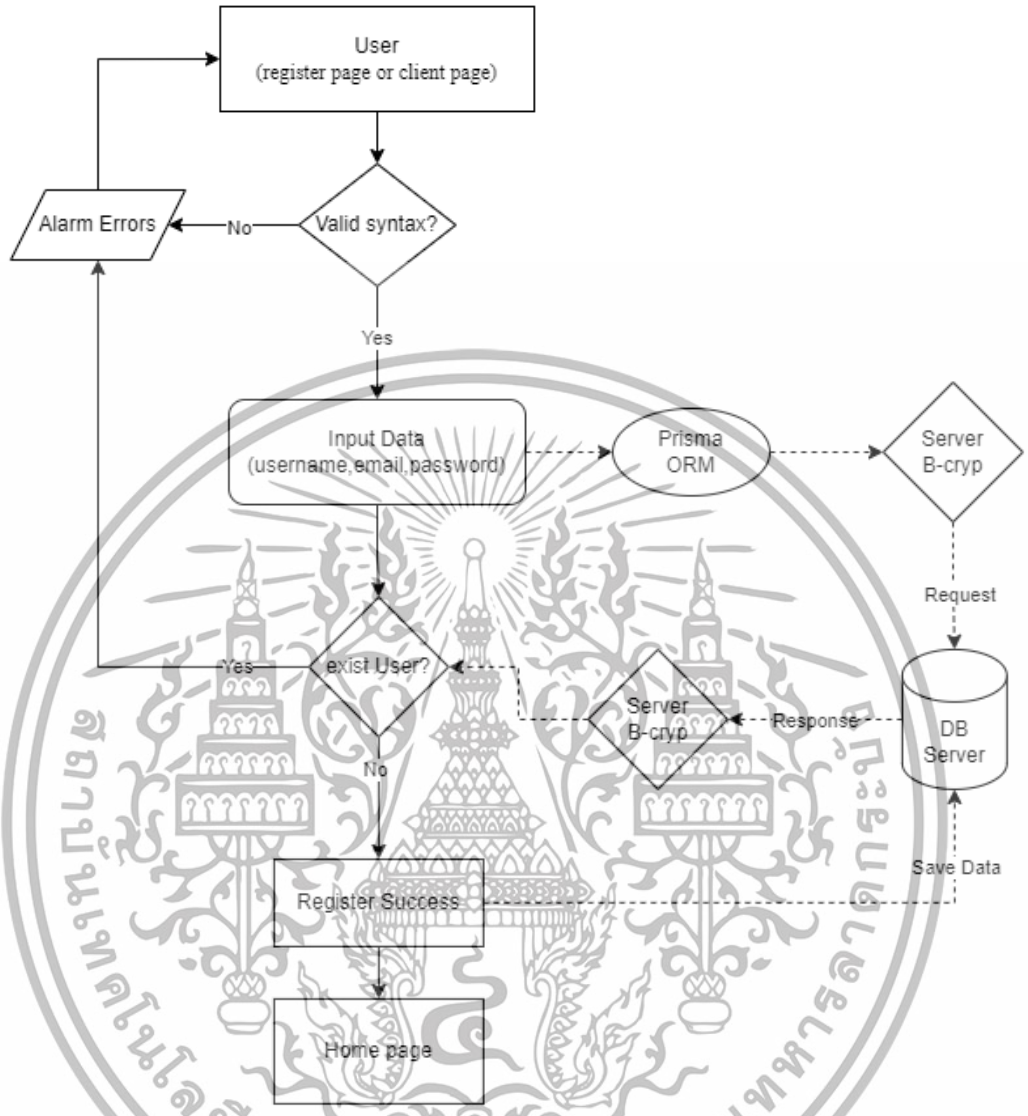
1) เว็บแอปพลิเคชันหน้า Sign up

ออกแบบเว็บแอปพลิเคชันหน้า Sign up โดยมีหน้าต่างให้ผู้ใช้งานต้องกรอกข้อมูลในช่อง Username Email และ Password ดังรูปที่ 3.35 ซึ่งเป็นข้อมูลจำเป็นที่ต้องกรอกจึงจะสามารถใช้งานภายในเว็บแอปพลิเคชันได้ และเมื่อกดปุ่ม Sign up จะมีการส่งข้อมูลเข้าฐานข้อมูลที่สร้างขึ้นมาเพื่อจัดเก็บข้อมูลบัญชีของผู้ใช้งาน



รูปที่ 3.35 โครงร่างเว็บแอปพลิเคชันหน้า Sign up

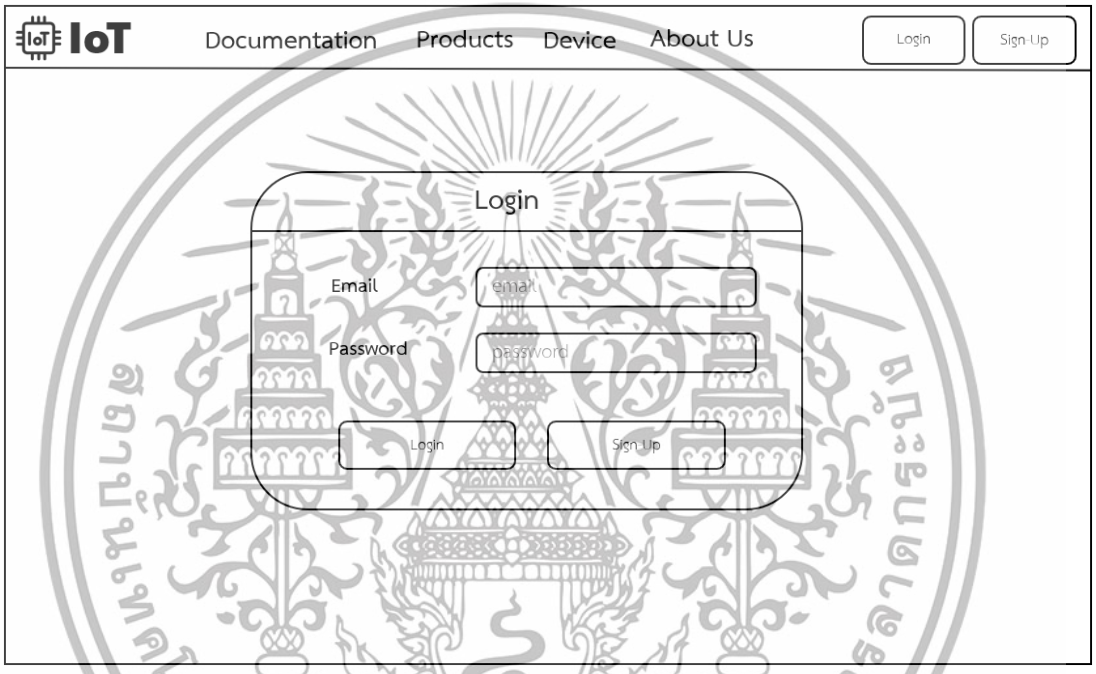
เมื่อผู้ใช้มีการกดปุ่ม Sign up ระบบจะมีการส่งข้อมูลผู้ใช้ที่ทำการสมัครบัญชีผู้ใช้งานเข้าสู่ฐานข้อมูล โดยมีกระบวนการทำงานในการสมัครบัญชีผู้ใช้งาน ดังรูปที่ 3.36



รูปที่ 3.36 กระบวนการทำงานของเว็บแอปพลิเคชันหน้า Sign up

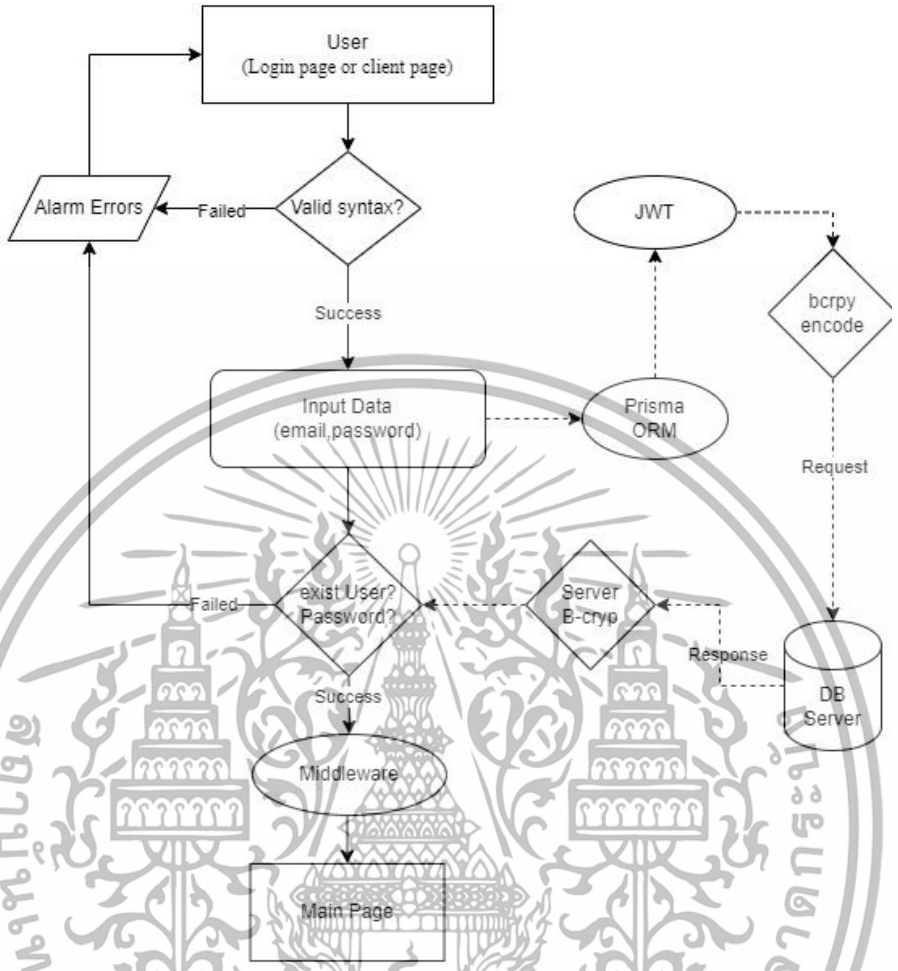
2) เว็บแอปพลิเคชันหน้า Login

ออกแบบเว็บแอปพลิเคชันหน้า Login โดยมีหน้าต่างให้ผู้ใช้งานต้องกรอกข้อมูลในช่อง Email และ Password ดังรูปที่ 3.37 โดยที่ข้อมูล Email และ Password ที่นำมากรอกในช่องหน้า Login ต้องเป็นข้อมูลที่ได้สมัครใช้งานจากหน้า Sign up มาแล้วจึงจะสามารถ Login ใช้งานภายในเว็บแอปพลิเคชันได้ และเมื่อกดปุ่ม Login จะมีการทำงานตรวจข้อมูลภายในฐานข้อมูลให้ตรงกับข้อมูลที่ผู้ใช้เข้าใช้งาน User



รูปที่ 3.37 โครงร่างเว็บแอปพลิเคชันหน้า Login

เมื่อผู้ใช้มีการกดปุ่ม Login ระบบจะตรวจข้อมูลภายในฐานข้อมูลให้ตรงกับข้อมูลที่ผู้ใช้เข้าใช้งาน โดยมีกระบวนการทำงานในการสมัครบัญชีผู้ใช้งาน ดังรูปที่ 3.38



รูปที่ 3.38 กระบวนการทำงานของเว็บแอปพลิเคชันหน้า Login

3) เว็บแอปพลิเคชันหน้าหลัก

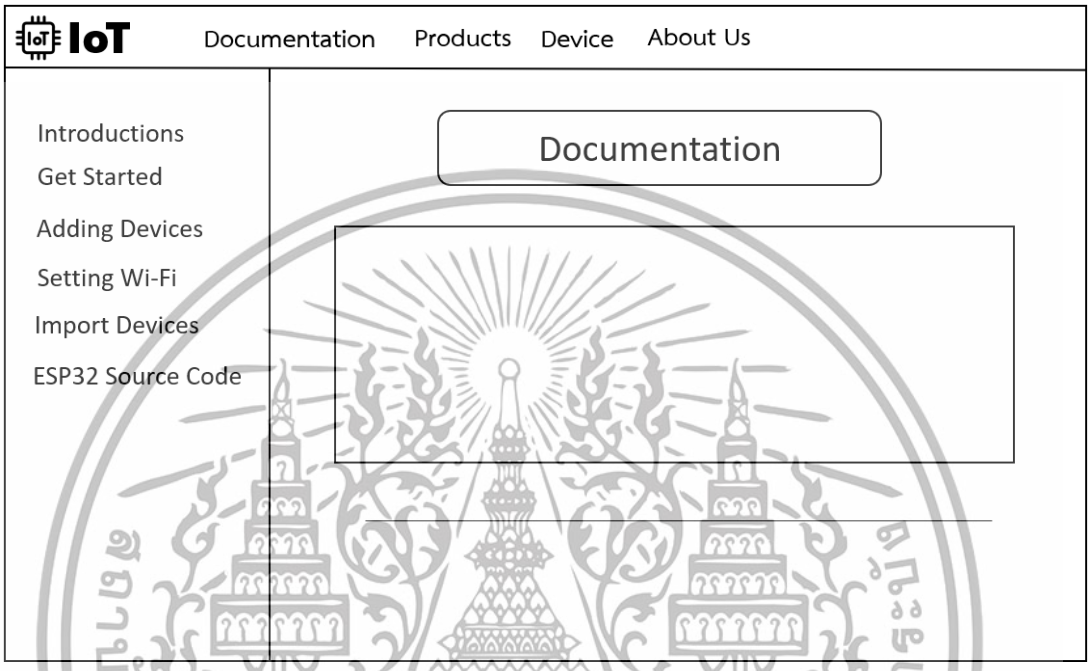
ออกแบบเว็บแอปพลิเคชันหน้าหลัก โดยเมื่อผู้ใช้งานมีการ Login เข้าใช้งานเข้ามาภายในเว็บแอปพลิเคชัน จะปรากฏหน้าหลักของเว็บแอปพลิเคชัน ดังรูปที่ 3.39 โดยมีแถบนำทาง (Navigation Bar) ซึ่งเป็นส่วนสำคัญในการเชื่อมต่อหน้าให้แก่ผู้ใช้งานมีส่วนประกอบ คือ Document Device About Us และข้อมูลผู้ใช้งาน ดังรูปที่ 3.40

รูปที่ 3.39 โครงร่างเว็บแอปพลิเคชันหน้าหลัก

รูปที่ 3.40 แถบนำทาง (Navigation Bar)

4) เว็บแอปพลิเคชันหน้า Document

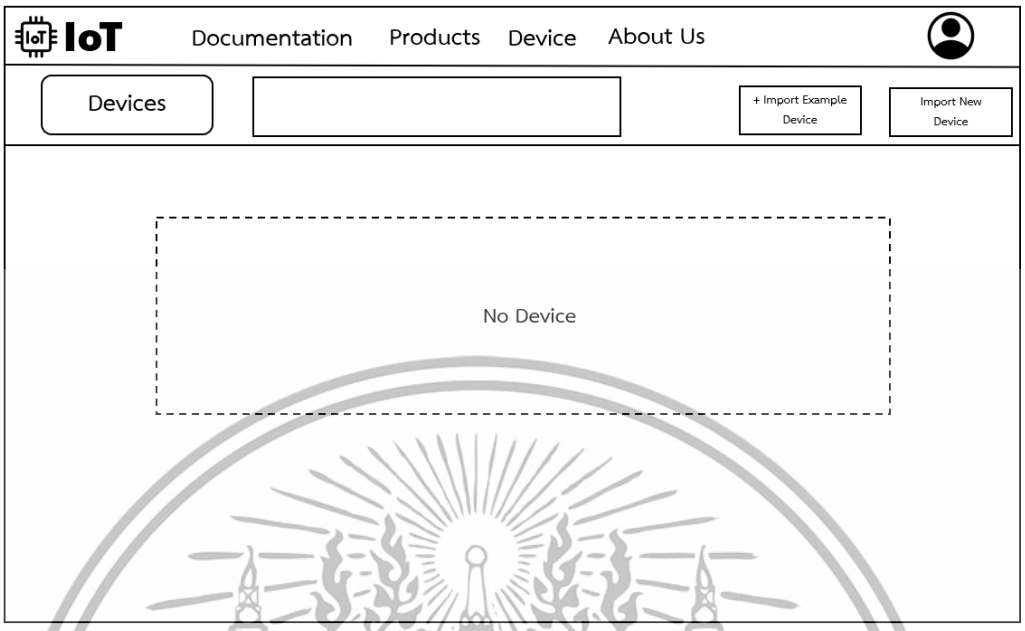
ออกแบบเว็บแอปพลิเคชันหน้า Document โดยมีส่วนการแสดงผลเป็นการสอนการใช้งานเบื้องต้นของเว็บแอปพลิเคชัน ดังรูปที่ 3.41



รูปที่ 3.41 โครงร่างเว็บแอปพลิเคชันหน้า Document

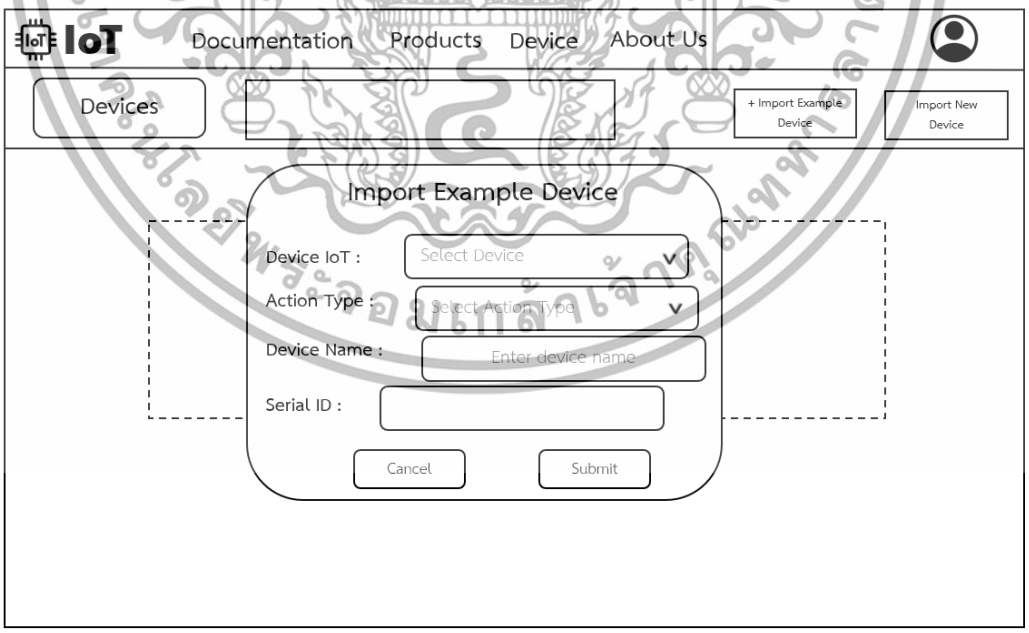
5) เว็บแอปพลิเคชันหน้า Device

ออกแบบเว็บแอปพลิเคชันหน้า Device โดยเมื่อผู้ใช้งานมีการกดเข้าใช้งานหน้า Device เข้ามาภายในเว็บแอปพลิเคชัน ด้านบนถัดจากแถบนำทางจะแสดงข้อมูลชื่อผู้ใช้งาน อีเมล จำนวนอุปกรณ์ภายในบัญชี สถานการณ์ใช้งาน และที่อยู่ IP ของผู้ใช้งานในขณะนั้น เมื่อยังไม่มีการเพิ่มอุปกรณ์ในการใช้งานจะปรากฏหน้า Device ของเว็บแอปพลิเคชัน ดังรูปที่ 3.42



รูปที่ 3.42 โครงร่างเว็บแอปพลิเคชันหน้า Device

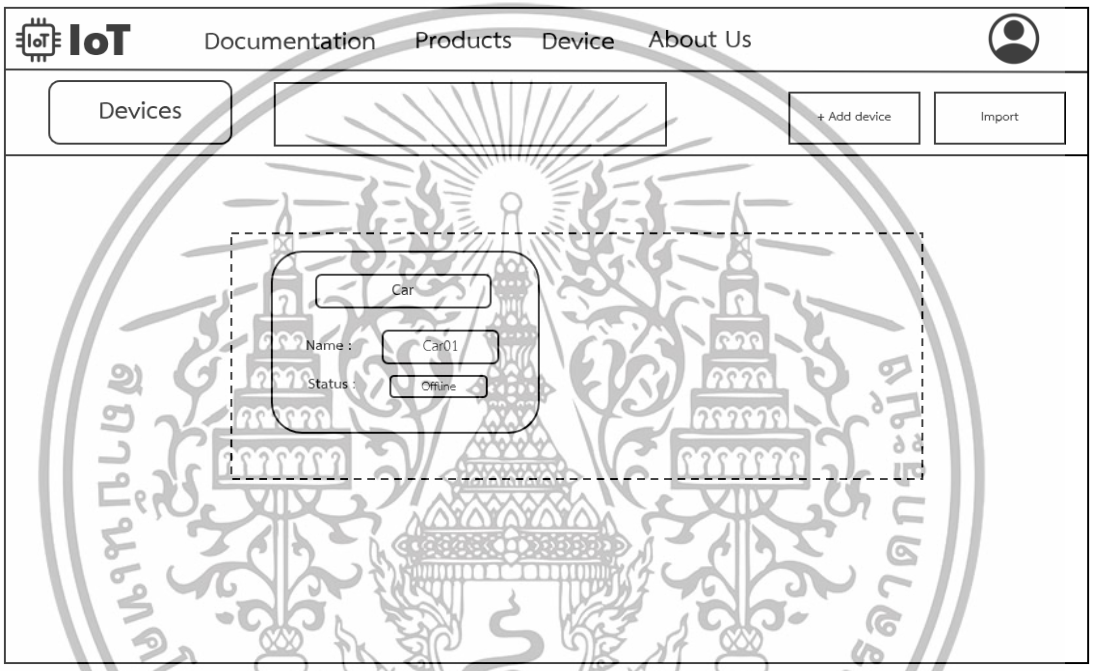
เมื่อมีการกดเพิ่มอุปกรณ์เข้ามาจากปุ่ม Import Example Device จะปรากฏหน้าต่างรายละเอียดในการเพิ่มอุปกรณ์ตามความต้องการของผู้ใช้งาน ดังรูปที่ 3.43



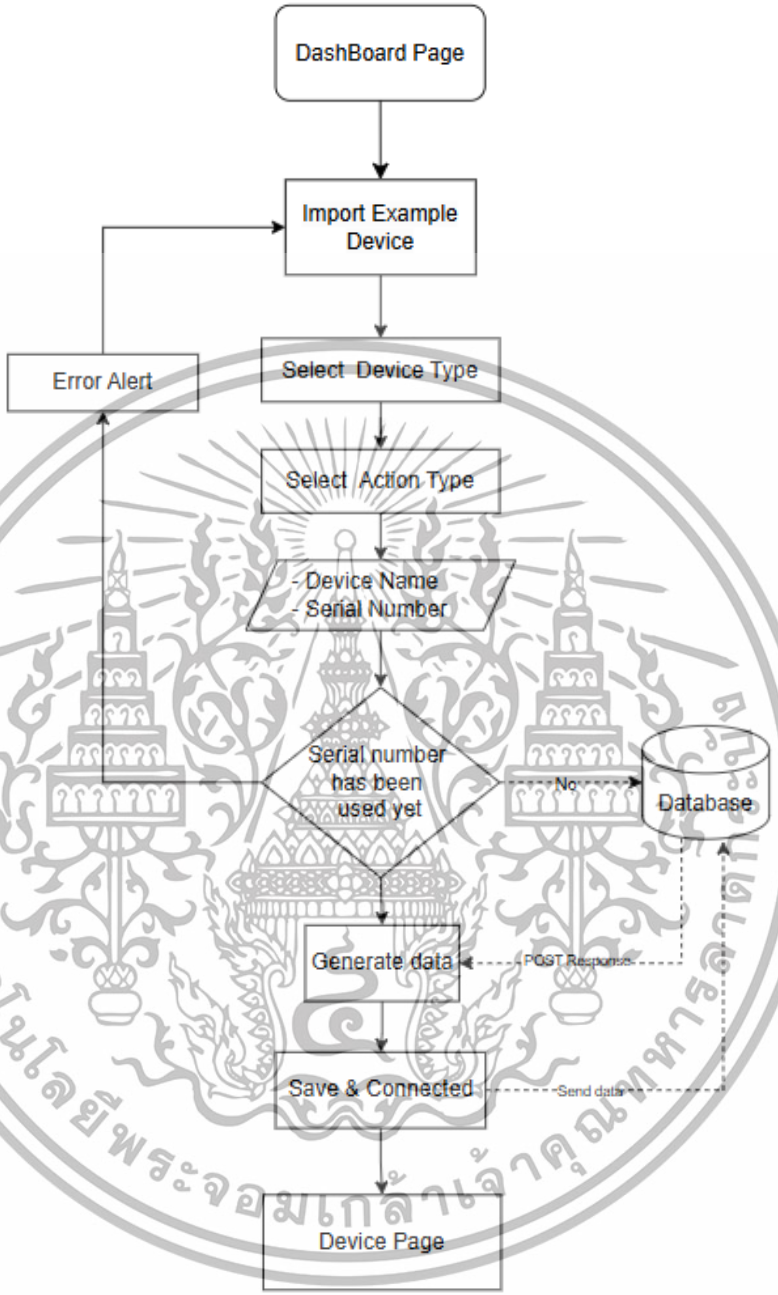
รูปที่ 3.43 โครงร่างเว็บแอปพลิเคชัน Import Example Device

6) เว็บแอปพลิเคชันหน้า Import Example Device

หลังจากกดปุ่ม Submit และเมื่อมีการนำเข้าอุปกรณ์ตัวอย่างจากการ Import Example Device เข้ามาแล้ว หน้า Import Example Device จะมีการแสดงการนำเข้าอุปกรณ์ตัวอย่างเข้ามา ดังรูปที่ 3.44 และระบบจะมีการเก็บข้อมูลเข้าฐานข้อมูล โดยจะเชื่อมโยง User เข้ากับ Device ในฐานข้อมูลของระบบ โดยมีกระบวนการทำงานในการนำเข้าอุปกรณ์ตัวอย่าง ดังรูปที่ 3.45



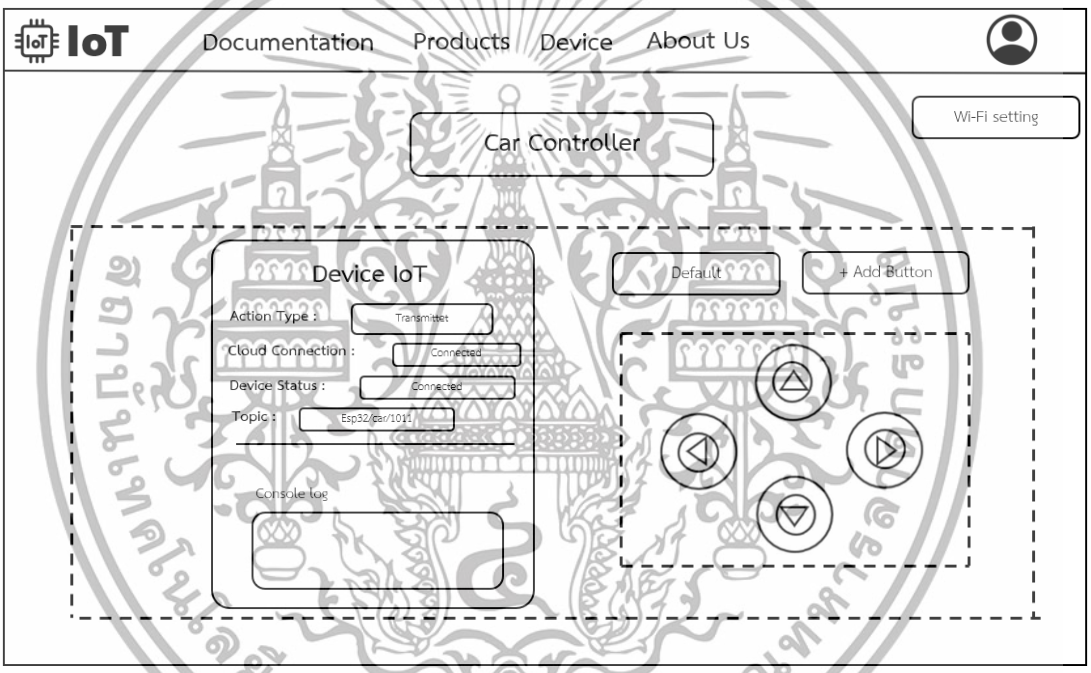
รูปที่ 3.44 โครงร่างเว็บแอปพลิเคชันเมื่อนำเข้าอุปกรณ์ตัวอย่าง



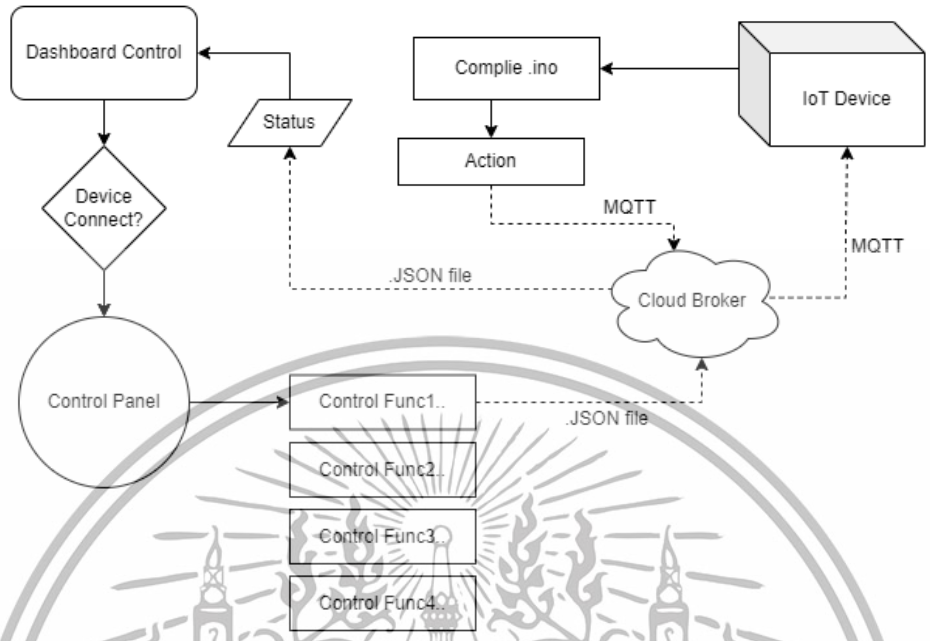
รูปที่ 3.45 กระบวนการทำงานในการนำเข้าอุปกรณ์ตัวอย่าง

7) เว็บแอปพลิเคชันหน้าควบคุมอุปกรณ์

ออกแบบเว็บแอปพลิเคชันหน้าควบคุมอุปกรณ์ โดยเมื่อผู้ใช้งานมีการกดเข้ามาภายในเว็บแอปพลิเคชัน โดยหน้าเว็บแอปพลิเคชันควบคุมอุปกรณ์จะแสดงข้อมูล Device Name (ชื่ออุปกรณ์ตามที่ผู้ใช้ตั้ง) Action Type (ประเภทการทำงานของอุปกรณ์) Cloud Connection (สถานะเชื่อมต่อกับ Cloud Broker) Device Status (สถานะการเชื่อมต่ออุปกรณ์กับเว็บแอปพลิเคชัน) Topic path ที่ใช้สื่อสาร Console log แสดงการทำงานของอุปกรณ์ และมีส่วนการควบคุมอุปกรณ์ทางด้านซ้ายของหน้าควบคุม ดังรูปที่ 3.46 และมีกระบวนในการทำงานในหน้าควบคุม ดังรูปที่ 3.47



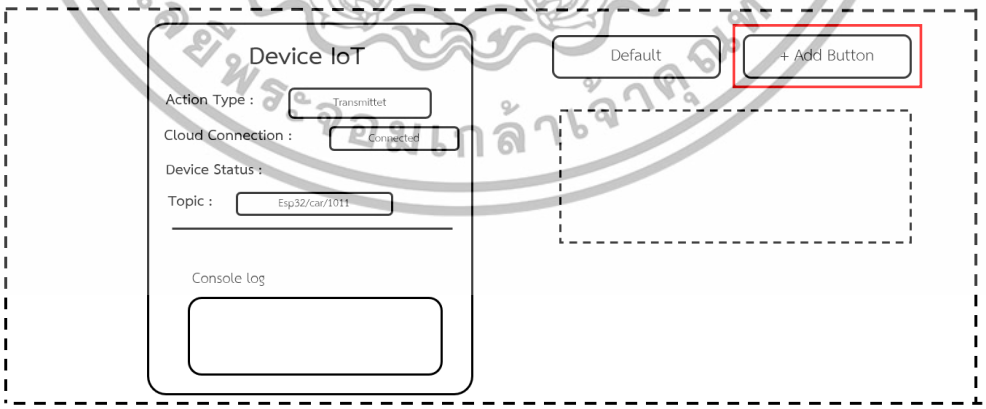
รูปที่ 3.46 โครงร่างเว็บแอปพลิเคชันหน้าควบคุมอุปกรณ์



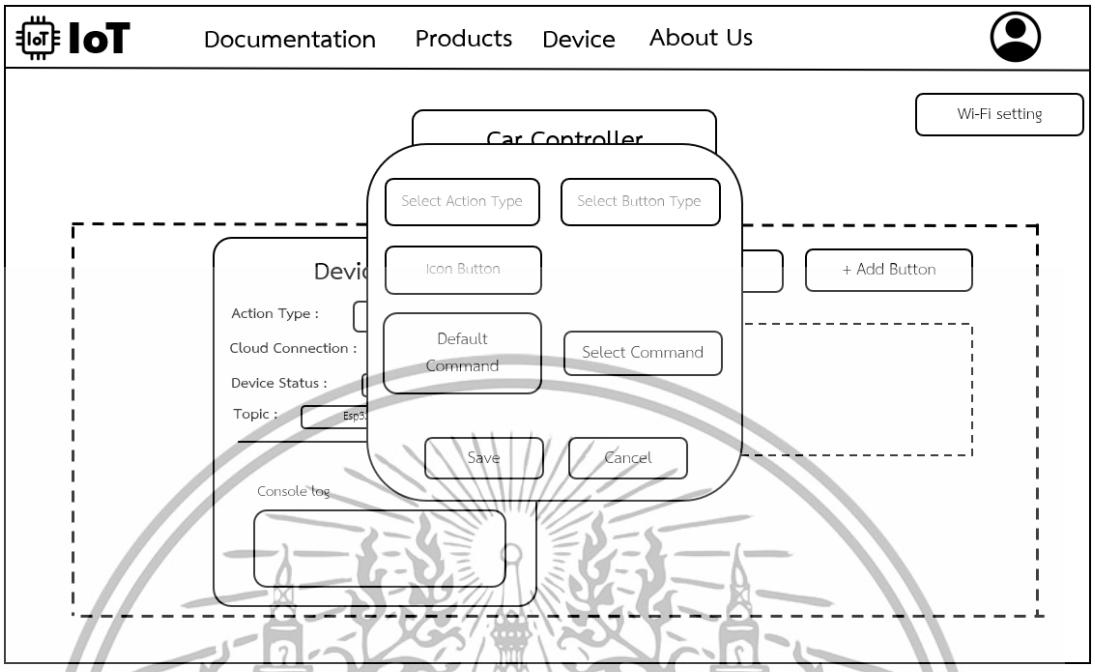
รูปที่ 3.47 กระบวนการทำงานในการควบคุมอุปกรณ์

8) เว็บแอปพลิเคชันการ Custom Button Feature

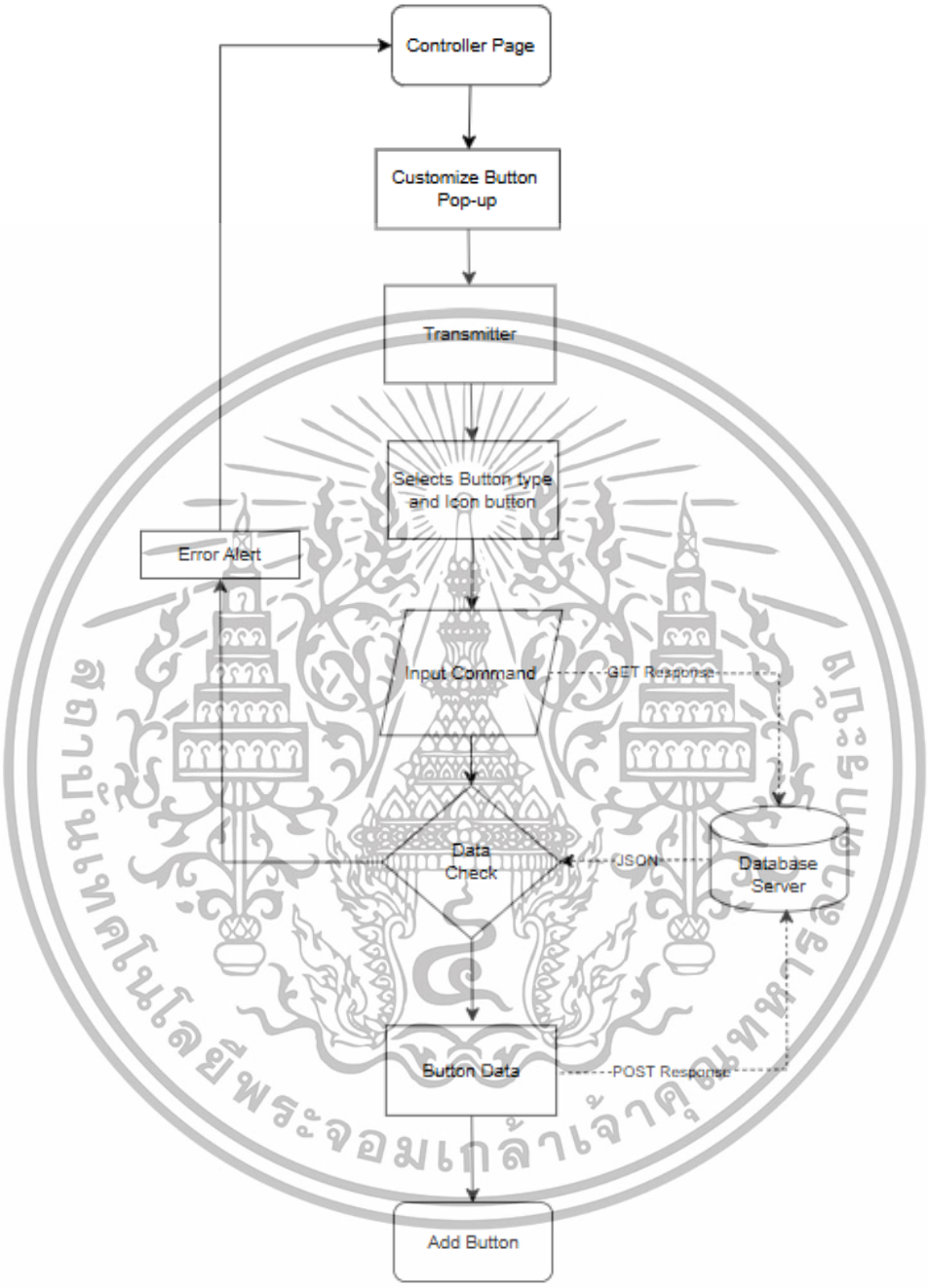
ออกแบบเว็บแอปพลิเคชันการ Custom Button Feature จะแสดงก็ต่อเมื่อผู้ใช้งานเลือกกดปุ่ม Add Button ดังรูปที่ 3.48 ในหน้าเริ่มต้นของอุปกรณ์ โดยผู้ใช้งานต้องเลือกและกรอกคำสั่งที่ต้องการตามความต้องการของตนเองใส่ภายในหน้าต่างรับค่าการสร้างปุ่มการสั่งงานดังรูปที่ 3.49 และมีกระบวนการทำงานดังรูปที่ 3.50



รูปที่ 3.48 โครงส่วนหน้าเริ่มต้นของอุปกรณ์



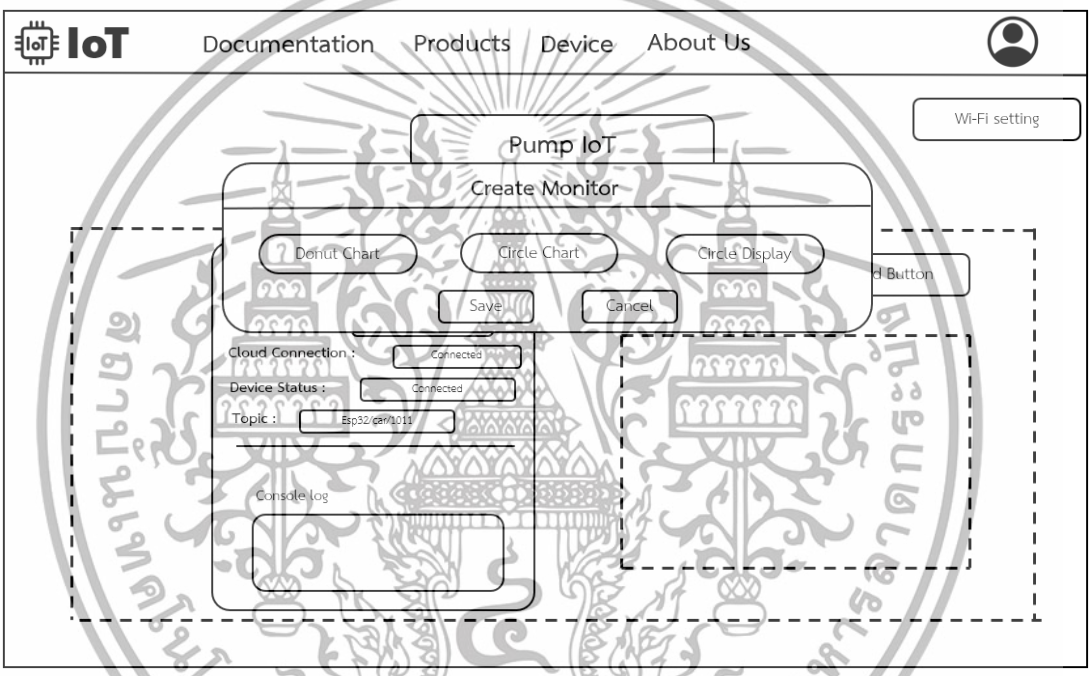
รูปที่ 3.49 โครงร่างเว็บแอปพลิเคชันการ Custom Button Feature



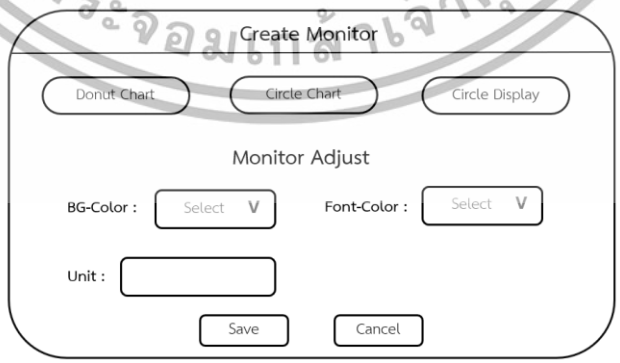
รูปที่ 3.50 กระบวนการทำงานในการ Custom Button Feature

9) เว็บแอปพลิเคชันการ Custom Monitor Display Feature

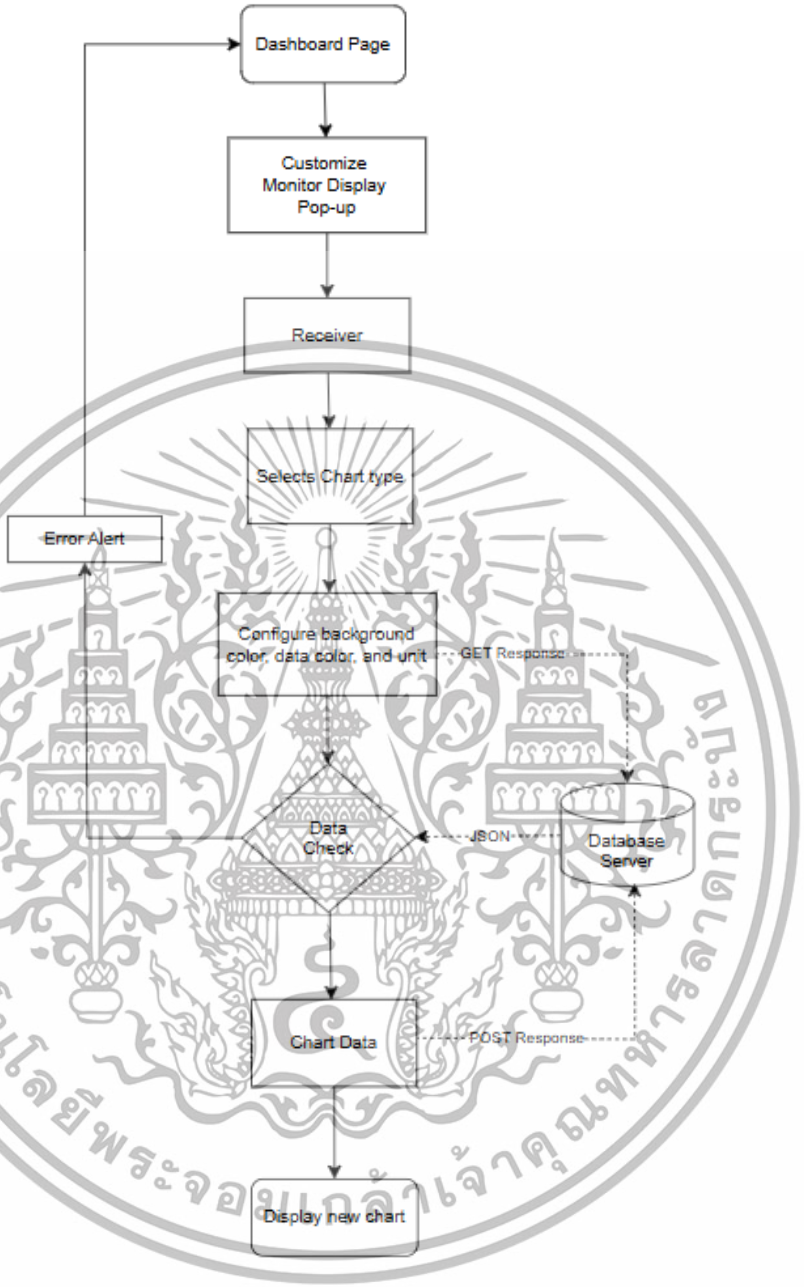
ออกแบบเว็บแอปพลิเคชันการ Custom Monitor Display Feature จะแสดงก็ต่อเมื่อผู้ใช้งานเลือกกดปุ่ม Add Button ดังรูปที่ ในหน้าเริ่มต้นของอุปกรณ์ที่มีการทำงานโดยการรับค่าการทำงานของอุปกรณ์จากเซนเซอร์ โดยผู้ใช้งานต้องเลือกและกรอกหน่วยของข้อมูลที่ต้องการแสดงตามความตั้งการของตนเองใส่ภายในหน้าต่างรับค่าการสร้างหน้าปัดแสดงผลและแผนภูมิ ดังรูปที่ 3.51 หากผู้ใช้งานเลือกการแสดงผลแบบหน้าปัดวงกลมจะต้องเลือกรูปแบบการแสดงผลของหน้าปัด ดังรูปที่ 3.52 และมีกระบวนการทำงานดังรูปที่ 3.53



รูปที่ 3.51 โครงร่างเว็บแอปพลิเคชันการ Custom Button Feature



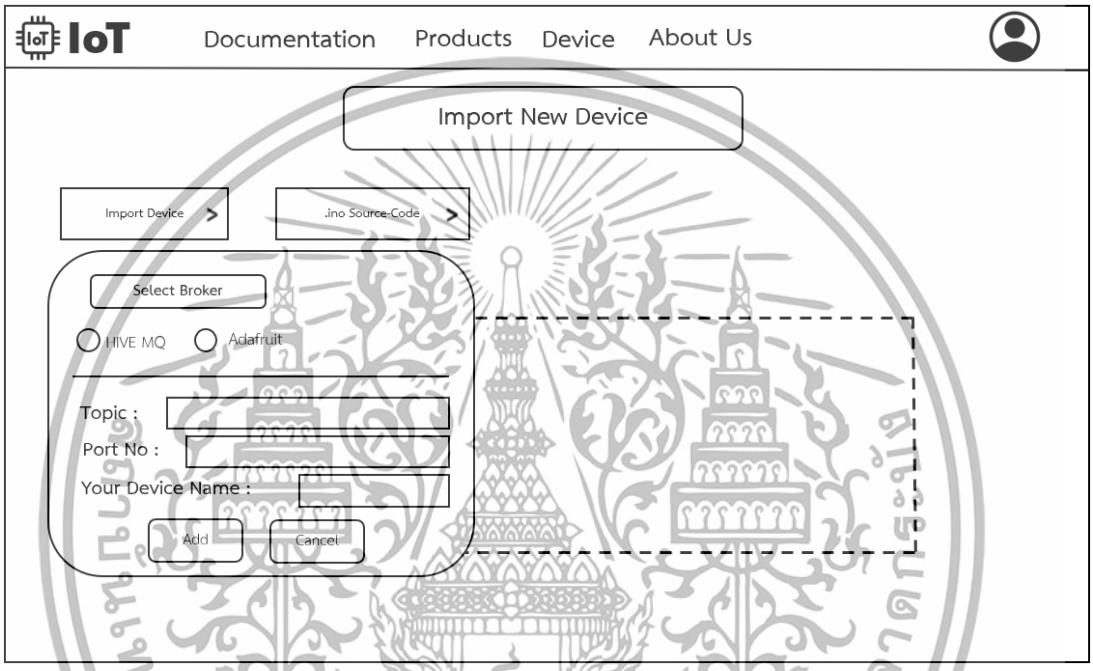
รูปที่ 3.52 โครงร่างหน้าต่างปรับแต่งหน้าปัดแสดงผลวงกลม



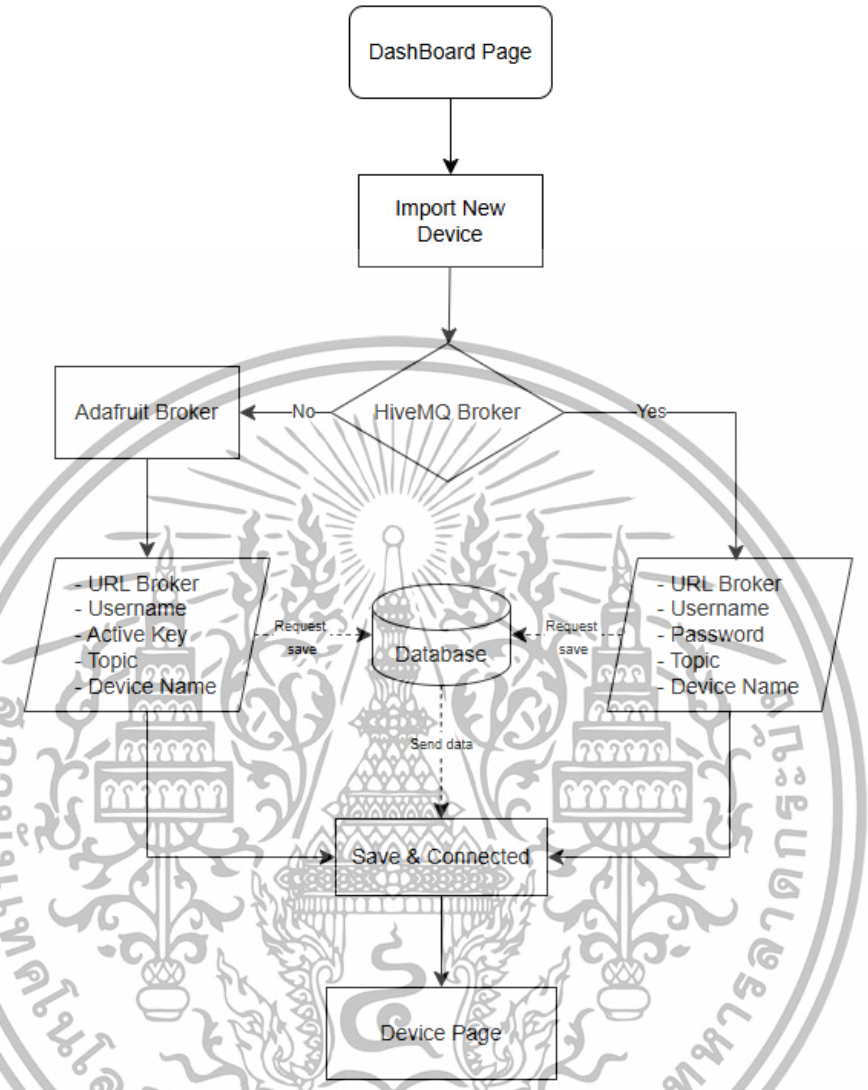
รูปที่ 3.53 กระบวนการทำงานในการ Custom Monitor Display Feature

10) เว็บแอปพลิเคชันหน้า Import New Device

ออกแบบเว็บแอปพลิเคชันหน้า Import New Device โดยเมื่อผู้ใช้งานมีการกดเข้าสู่หน้าการนำเข้าอุปกรณ์จากภายนอกมาใช้งานร่วงกับเว็บแอปพลิเคชัน ผู้ใช้ต้องเลือก Broker และกรอกข้อมูลที่ใช้ในการเชื่อมต่ออุปกรณ์ให้ครบถ้วนภายในกล่องรับข้อมูล ดังรูปที่ 3.54 และมีกระบวนการทำงานดังรูปที่ 3.55



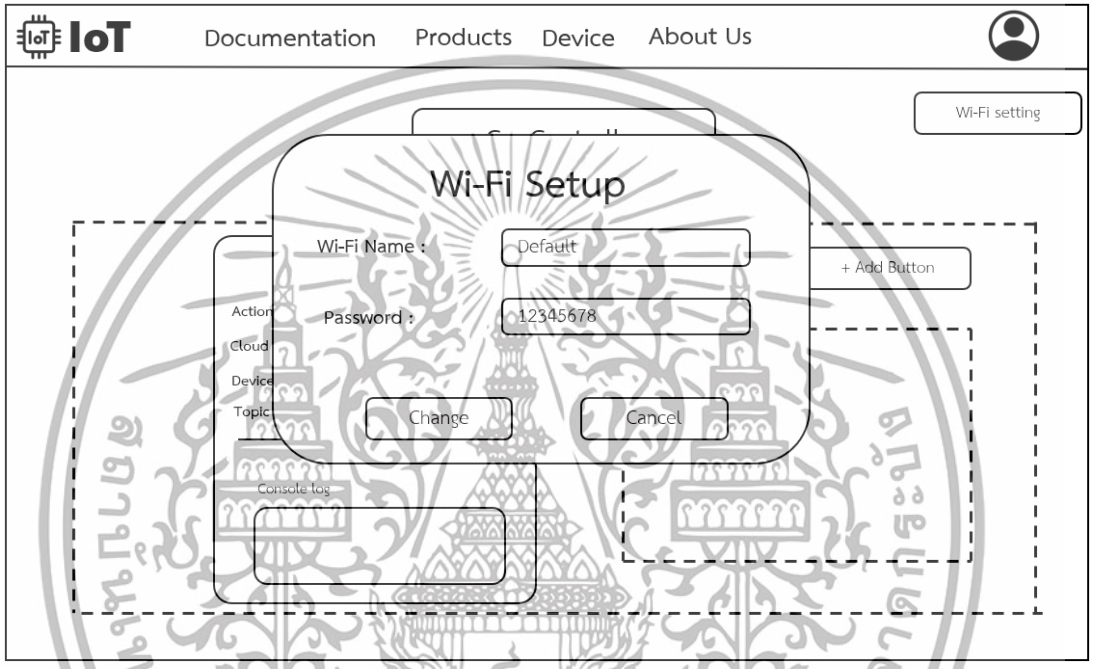
รูปที่ 3.54 โครงร่างเว็บแอปพลิเคชันหน้า Import New Device



รูปที่ 3.55 กระบวนการทำงานหน้า Import New Device

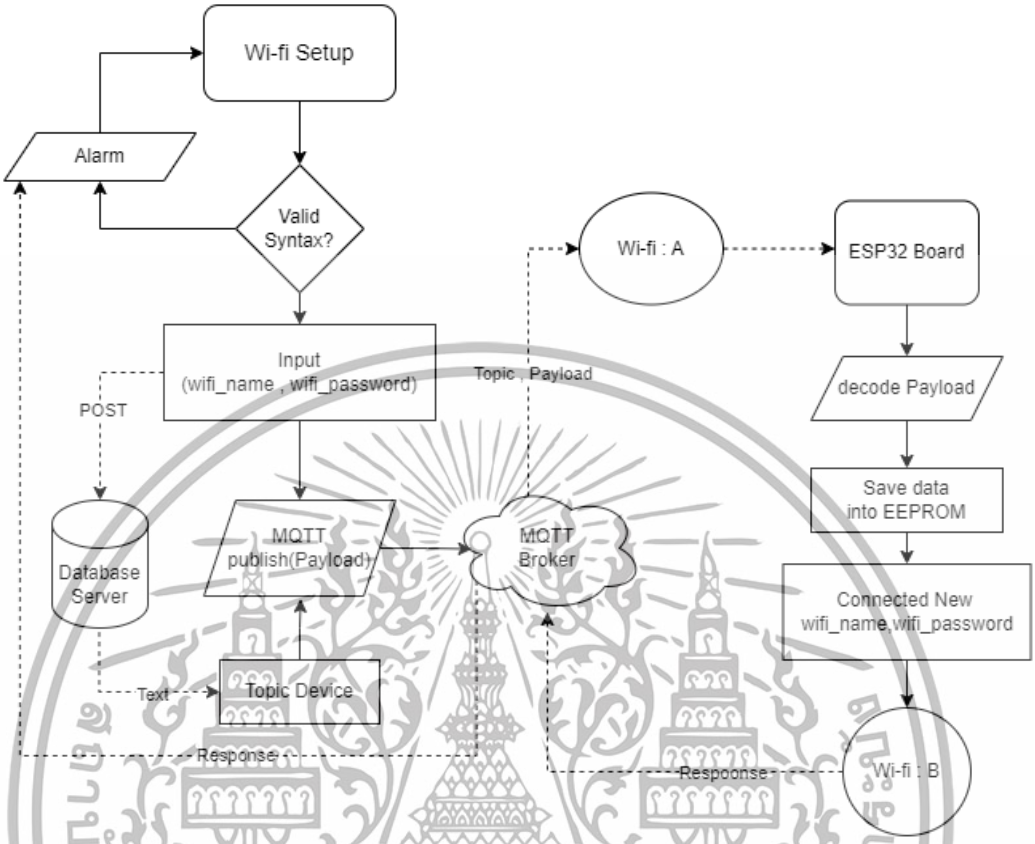
11) เว็บแอปพลิเคชันหน้าตั้งค่า Wi-Fi

ออกแบบเว็บแอปพลิเคชันหน้าตั้งค่า Wi-Fi โดยเมื่อผู้ใช้งานกดเข้าใช้งานตั้งค่า Wi-Fi เข้ามาภายในเว็บแอปพลิเคชันจากหน้า Device โดยหน้าเว็บแอปพลิเคชันจะแสดงข้อมูล Username และ Password เดิมของอุปกรณ์ และมีช่องรับค่า Username และ Password ใหม่ของอุปกรณ์ที่ผู้ใช้งานต้องกรอกค่าตาม Wi-Fi ของตนเอง ดังรูปที่ 3.56



รูปที่ 3.56 โครงร่างเว็บแอปพลิเคชันหน้าตั้งค่า Wi-Fi

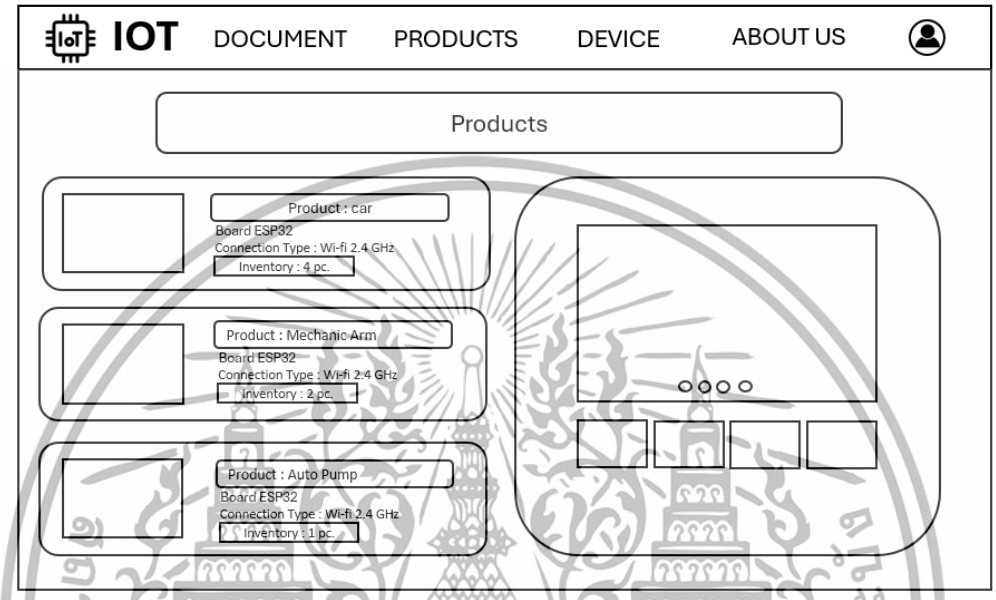
เมื่อผู้ใช้มีการกดปุ่ม Submit ระบบจะส่งค่า Username และ Password ใหม่ที่มีการตั้งค่าโดยผู้ใช้งานเข้าฐานข้อมูล และส่งไปให้อุปกรณ์เพื่อใช้ในการเชื่อมต่อ Wi-Fi ต่อไป โดยมีกระบวนการทำงานในการตั้งค่า Wi-Fi ดังรูปที่ 3.57



รูปที่ 3.57 กระบวนการทำงานในการตั้งค่า Wi-Fi

12) เว็บแอปพลิเคชันหน้า Product

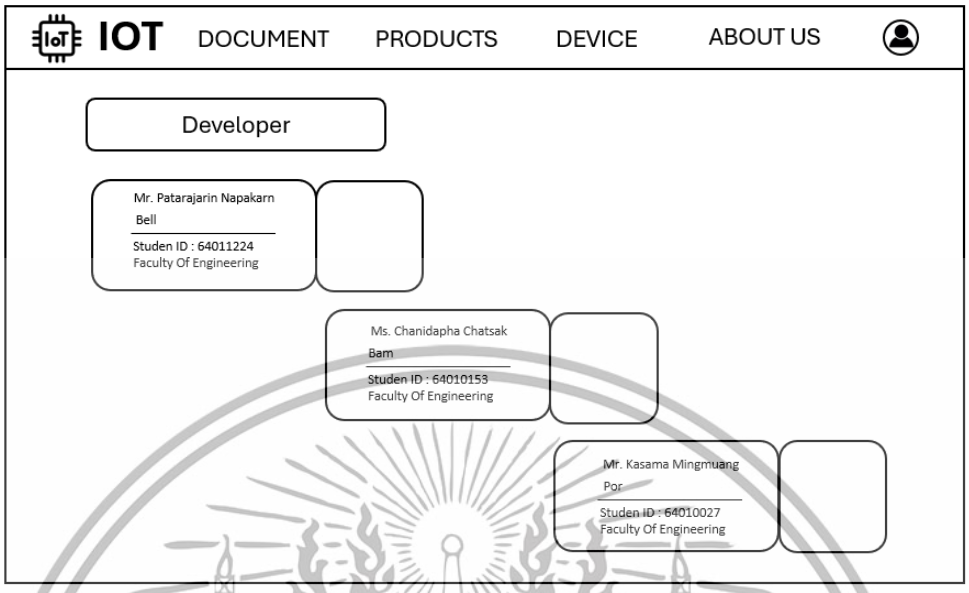
ออกแบบเว็บแอปพลิเคชันหน้า Product เป็นหน้าแสดงอุปกรณ์ทั้งหมดภายในเว็บแอปพลิเคชัน โดยหน้าเว็บแอปพลิเคชันจะแสดงข้อมูลอุปกรณ์ ดังรูปที่ 3.58



รูปที่ 3.58 โครงร่างเว็บแอปพลิเคชันหน้า Product

13) เว็บแอปพลิเคชันหน้า About Us

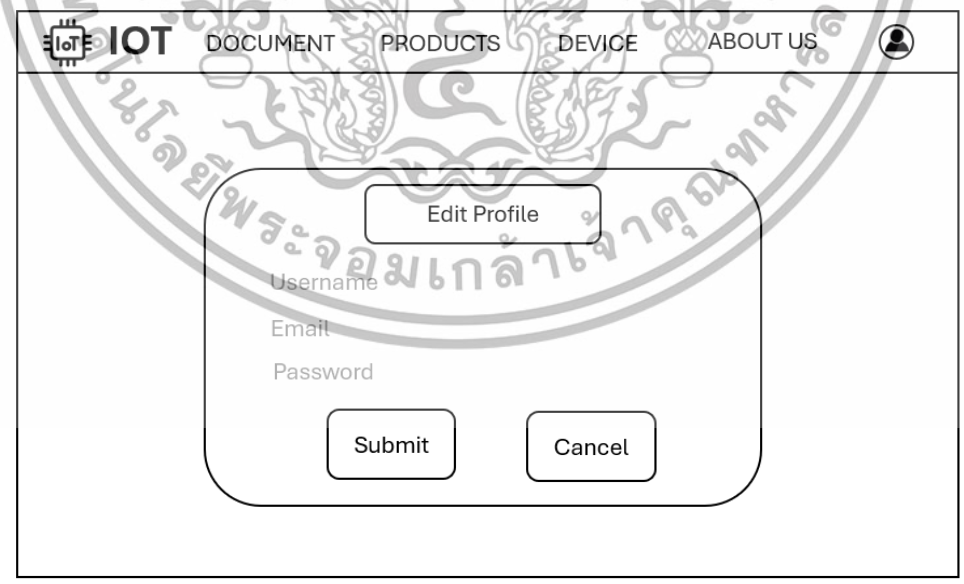
ออกแบบเว็บแอปพลิเคชันหน้า About Us โดยจะมีการแสดงข้อมูลสำคัญของวิทยานิพนธ์ ชื่อผู้จัดทำ และอาจารย์ที่ปรึกษา ดังรูปที่ 3.59



รูปที่ 3.59 โครงร่างเว็บแอปพลิเคชันหน้า About Us

14) เว็บแอปพลิเคชันหน้าข้อมูลผู้ใช้งาน

ออกแบบเว็บแอปพลิเคชันหน้าข้อมูลผู้ใช้งาน โดยจะมีการแสดงข้อมูลพื้นฐานที่ผู้ใช้งานสมัครเข้าใช้งานเว็บแอปพลิเคชัน และสามารถแก้ไขข้อมูลผู้ใช้งานได้ผ่านหน้านี้ ดังรูปที่ 3.60



รูปที่ 3.60 โครงร่างเว็บแอปพลิเคชันหน้าข้อมูลผู้ใช้งาน

3.2.1.2 ระบบจัดการเว็บแอปพลิเคชัน (Back-End)

ระบบจัดการเว็บแอปพลิเคชัน (Back-End) โดยการออกแบบ API ภายในเว็บแอปพลิเคชัน เพื่อใช้ในการจัดการข้อมูลภายในเว็ปพลิเคชัน ดังนี้

1) การออกแบบ API สำหรับการจัดการอุปกรณ์ (Devices) จะถูกแบ่งออกเป็น 4 ส่วน ได้แก่ getDevice POST PUT และ getDeviceById โดยทุกตัวจะทำการดึงข้อมูลจาก API มาแสดงผล (Fetch) ซึ่งแต่ละส่วนมีหน้าที่เฉพาะในการทำงานกับฐานข้อมูล MongoDB ซึ่งเป็นฐานข้อมูลประเภท Document ที่มีตารางชื่อ devices ดังนี้

- getDevice ใช้ Method GET ในการดึงข้อมูลทั้งหมดที่มีในฐานข้อมูลออกมาแสดงผลในหน้าเว็บและหน้าผู้ดูแลระบบ เพื่อทำการตรวจสอบข้อมูลอุปกรณ์ที่ถูกเก็บไว้ได้ การดึงข้อมูลนี้จะช่วยให้ผู้ดูแลสามารถเห็นสถานะและรายละเอียดของอุปกรณ์ทุกชิ้นที่มีอยู่ในระบบ รวมถึงสามารถตรวจสอบปัญหาหรือความผิดปกติได้
- POST โดยใช้ Method POST ในการส่งข้อมูลที่ผู้ใช้ทำการ Import Example Device จากหน้าเว็บเข้าไปสู่ฐานข้อมูล เพื่อสร้างและเก็บบันทึกข้อมูลของอุปกรณ์ที่ผู้ใช้ได้ทำการเก็บไว้ การใช้งานนี้จะช่วยให้ผู้ใช้สามารถเพิ่มอุปกรณ์ที่ต้องการติดตาม ซึ่งข้อมูลที่ส่งไปจะต้องอยู่ในรูปแบบที่กำหนดไว้ เช่น ชื่ออุปกรณ์ ประเภท และสถานการณ์เชื่อมต่อ
- PUT โดยใช้ Method PUT ในการอัปเดตข้อมูลและการกระทำต่าง ๆ ที่มีผลทำให้ข้อมูลของตัวอุปกรณ์เปลี่ยนแปลง เป็นการเข้าไปแก้ไขข้อมูลบางจุดในฐานข้อมูลให้เป็นข้อมูลล่าสุด เช่น สถานการณ์เชื่อมต่อหรือข้อมูลอื่น ๆ ที่เกี่ยวข้องกับอุปกรณ์นั้น การอัปเดตข้อมูลนี้จะช่วยให้ฐานข้อมูลมีข้อมูลที่เป็นปัจจุบันและถูกต้องตลอดเวลา
- getDeviceById โดยใช้ Method GET/{params} เพื่อดึงข้อมูลแบบเฉพาะเจาะจงด้วยไอดีของผู้ใช้งาน ในการนำเพียงแค่ข้อมูลอุปกรณ์ของผู้ใช้นั้นมาแสดงไว้บนหน้าเว็บแอปพลิเคชัน

2) การออกแบบ API สำหรับการจัดการผลิตภัณฑ์ (Products) จะถูกแบ่งออกเป็น 3 ส่วน ได้แก่ GET POST และ PUT โดยทุกตัวจะทำการ fetch ซึ่งแต่ละส่วนมีหน้าที่เฉพาะในการทำงานกับฐานข้อมูล MongoDB ซึ่งเป็นฐานข้อมูลประเภท Document ที่มีตารางชื่อ products ดังนี้

- GET โดยใช้ Method GET ในการดึงข้อมูล products ทุกตัวมาแสดงหรือนำมาตรวจหาข้อมูล products ที่ตรงกัน มีไว้เพื่อแสดงอุปกรณ์ภายในหน้าเว็บแอปพลิเคชัน
- POST โดยใช้ Method POST ในการสร้าง products ส่วนนี้จะอนุญาตเฉพาะผู้ดูแลของระบบเท่านั้น ซึ่งจะมีการตรวจสอบสิทธิ์ก่อนที่จะทำการเพิ่มข้อมูล เพื่อป้องกันการแก้ไขหรือเพิ่มข้อมูลที่ไม่เหมาะสม
- PUT โดยใช้ Method PUT ในการอัปเดตข้อมูลและการกระทำต่าง ๆ ที่มีผลทำให้ข้อมูลของ products เปลี่ยนแปลง เป็นการเข้าไปแก้ไขข้อมูลบางจุดในฐานข้อมูลให้เป็นข้อมูลล่าสุด เช่น สถานะการเป็นเจ้าของ ใน product นั้น ๆ

3) การออกแบบ API สำหรับระบบ Authentication แบ่งออกเป็น 3 ส่วน ได้แก่ Sign up Login และ Logout ซึ่งมีการใช้งาน API และฐานข้อมูล MongoDB ในการจัดการข้อมูลผู้ใช้ ดังนี้

- Sign up โดยใช้ Method POST ในการทำงานเมื่อผู้ใช้ทำการสมัครสมาชิก ข้อมูลที่ผู้ใช้ป้อนเข้ามาจะถูกตรวจสอบเพื่อป้องกันการซ้ำซ้อนของข้อมูลผู้ใช้ เช่น ตรวจสอบว่ามีอีเมลซ้ำกันหรือไม่ ถ้าข้อมูลผ่านการตรวจสอบโปรแกรมจะทำการเข้ารหัสรหัสผ่านของผู้ใช้ด้วย bcrypt เพื่อความปลอดภัย และทำการแปลงข้อมูลให้เป็นรูปแบบ base64 หลังจากนั้นข้อมูลผู้ใช้ที่ผ่านการเข้ารหัสจะถูกบันทึกลงในฐานข้อมูล MongoDB เพื่อให้ผู้ใช้สามารถเข้าสู่ระบบได้ในครั้งถัดไป
- Login โดยใช้ Method GET ในการทำงานเมื่อผู้ใช้ทำการเข้าสู่ระบบ โปรแกรมจะทำการเข้ารหัสข้อมูลของผู้ใช้ ได้แก่ อีเมลและรหัสผ่านเป็นรูปแบบ base64 ก่อนที่จะนำไปตรวจสอบกับข้อมูลที่มีอยู่ในฐานข้อมูล

หากข้อมูลที่ป้อนตรงกับข้อมูลที่มีอยู่ (ทั้งอีเมลและรหัสผ่าน) โปรแกรมจะทำการสร้าง Token สำหรับการยืนยันตัวตนของผู้ใช้ Token นี้จะถูกเก็บไว้ใน Session ของเว็บไซต์ ซึ่งข้อมูลดังกล่าวจะถูกบันทึกลงใน Cookies เพื่อให้ระบบสามารถตรวจสอบและเก็บสถานการณ์เข้าสู่ระบบของผู้ใช้ได้

- Logout จะไม่มีการใช้งาน Method กับ API โดยตรง แต่จะทำงานเมื่อผู้ใช้ทำการออกจากระบบ ระบบจะทำการดึงข้อมูลจาก Session Cookies เพื่อตรวจสอบเวลาที่ผู้ใช้ทำการออกจากระบบ

4) การออกแบบ API สำหรับการจัดการ Wi-Fi ถูกแบ่งออกเป็น 4 ส่วนหลัก ได้แก่ getWifi POST PUT และ getWifiById ซึ่งแต่ละส่วนมีหน้าที่เฉพาะในการทำงานกับฐานข้อมูล MongoDB ซึ่งเป็นฐานข้อมูลแบบ Document โดยใช้ Mongoose ในการจัดการข้อมูล และตารางที่ใช้เก็บข้อมูลมีชื่อว่า wifi โดยแต่ละฟังก์ชันใน API นี้ทำหน้าที่แตกต่างกันไป ดังนี้

- getWifi ใช้ Method GET ในการดึงข้อมูล Wi-Fi ทั้งหมด ที่มีอยู่ในฐานข้อมูลมาแสดงผลบนหน้าเว็บและหน้าผู้ดูแลระบบ เพื่อให้สามารถตรวจสอบรายการ Wi-Fi ที่มีอยู่ทั้งหมดได้ ข้อมูลที่ดึงมาจะประกอบไปด้วย SSID (wifiName) รหัสผ่าน (wifiPassword) สถานะ (status) และข้อมูลที่เกี่ยวข้อง
- POST ใช้ Method POST เพื่อทำการเพิ่มข้อมูล Wi-Fi ใหม่ เข้าไปในฐานข้อมูล ซึ่งข้อมูลที่ผู้ใช้กรอกเข้ามาจากหน้าเว็บจะถูกบันทึกลงในฐานข้อมูล โดยมีข้อมูลที่จำเป็น
- PUT ใช้ Method PUT เพื่ออัปเดตข้อมูลของ Wi-Fi ในฐานข้อมูล โดยใช้ wifiId เป็นตัวกำหนดข้อมูลที่ต้องการเปลี่ยนแปลง
- getWifiById ใช้ Method GET/{params} เพื่อนำข้อมูล Wi-Fi เฉพาะตัวที่ต้องการ ออกมาแสดง โดยใช้ wifiId เป็นตัวระบุเพื่อดึงข้อมูลเฉพาะ Wi-Fi ที่ต้องการ

5) การออกแบบ API สำหรับการจัดการปุ่ม แบ่งออกเป็น 4 ส่วน ได้แก่ GET POST DELETE และ GET/{id} โดย API นี้ทำงานร่วมกับฐานข้อมูล MongoDB ผ่าน Mongoose เพื่อจัดเก็บข้อมูลของปุ่มควบคุม ซึ่งเชื่อมโยงกับอุปกรณ์แต่ละตัวผ่าน deviceId โดยมีการเชื่อมต่อฐานข้อมูลผ่าน connect() ก่อนดำเนินการ และแต่ละฟังก์ชันใน API นี้ทำหน้าที่แตกต่างกันไป ดังนี้

- GET ใช้ Method GET เพื่อดึงข้อมูล ปุ่มทั้งหมดที่มีอยู่ในฐานข้อมูลมาแสดง API นี้ไม่มีการรับพารามิเตอร์ และจะดึงข้อมูลทั้งหมดจากคอลเลกชัน Button เพื่อนำไปแสดงผลในหน้าเว็บหรือแดชบอร์ดของระบบ ข้อมูลที่ได้จะประกอบด้วย id type category label command และ deviceId การเรียกใช้ API นี้ช่วยให้สามารถดูปุ่มที่มีอยู่ทั้งหมด ซึ่งเป็นประโยชน์สำหรับผู้ดูแลระบบที่ต้องการตรวจสอบรายการปุ่มที่มีการสร้างไว้แล้ว
- GET/{id} ใช้ Method GET พร้อมพารามิเตอร์ id เพื่อดึงข้อมูลปุ่มทั้งหมดที่เชื่อมโยงกับอุปกรณ์ที่กำหนด โดยจะใช้ deviceId เป็นตัวกรองเพื่อค้นหาปุ่มทั้งหมดที่เกี่ยวข้องกับอุปกรณ์นั้น
- POST ใช้ Method POST เพื่อเพิ่มข้อมูลปุ่มใหม่ลงในฐานข้อมูล ผู้ใช้จะต้องส่งข้อมูลในรูปแบบ JSON ซึ่งประกอบไปด้วย id type category label, command และ deviceId โดยระบบจะใช้ค่าที่ได้รับเพื่อสร้างเอกสารใหม่ในคอลเลกชัน Button เมื่อเพิ่มข้อมูลสำเร็จ ระบบจะส่งข้อความ "Button Created" กลับมาเพื่อยืนยันว่าการสร้างปุ่มเสร็จสมบูรณ์
- DELETE/{id} ใช้ Method DELETE เพื่อทำการลบปุ่มที่ต้องการออกจากฐานข้อมูล โดยจะใช้ id ของปุ่มเป็นตัวกำหนดเป้าหมายในการลบ หากปุ่มถูกลบสำเร็จ ระบบจะส่งข้อความ "Delete Button Success" กลับมาเพื่อแจ้งว่าการลบเสร็จสิ้น

6) การออกแบบ API สำหรับการจัดการข้อมูลกราฟนี้ถูกออกแบบมาเพื่อรองรับการ เพิ่ม ดึงข้อมูล และลบกราฟ ที่เชื่อมโยงกับอุปกรณ์ IoT ผ่านฐานข้อมูล MongoDB โดยใช้ Mongoose เพื่อจัดเก็บข้อมูลลักษณะกราฟ เช่น ประเภทของกราฟ สีพื้นหลัง สีตัวอักษร หน่วยวัด และอุปกรณ์ที่เกี่ยวข้อง แต่ละฟังก์ชันใน API นี้ทำหน้าที่แตกต่างกันไป ดังนี้

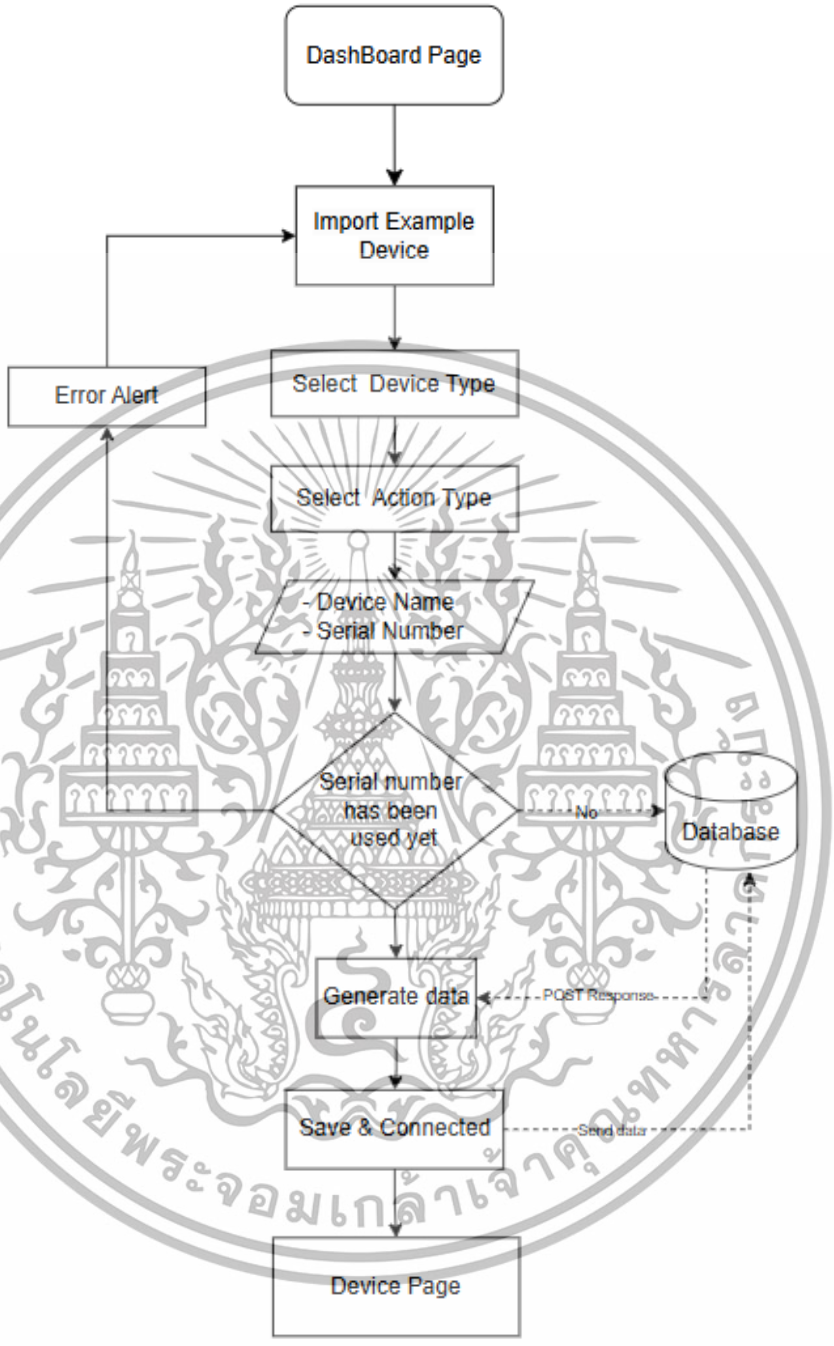
- POST ใช้ Method POST เพื่อเพิ่มข้อมูลของกราฟใหม่เข้าสู่ฐานข้อมูล โดยต้องส่งข้อมูลในรูปแบบ JSON ซึ่งประกอบด้วย id type label bgcolor fgcolor unit และ deviceId เมื่อ API ได้รับข้อมูลแล้ว ระบบจะใช้คำสั่ง Chart.create() เพื่อบันทึกข้อมูลกราฟลงในฐานข้อมูล จากนั้นส่งข้อความ "Chart Created" เพื่อยืนยันว่ากราฟถูกสร้างสำเร็จ
- GET ใช้ Method GET เพื่อดึงข้อมูลของกราฟทั้งหมดที่ถูกบันทึกไว้ในฐานข้อมูล API นี้ไม่มีพารามิเตอร์เพิ่มเติม และใช้คำสั่ง Chart.find() เพื่อค้นหาข้อมูลกราฟทั้งหมดที่มีอยู่ในคอลเลกชัน Chart จากนั้นส่งคืนข้อมูลทั้งหมดในรูปแบบ JSON การเรียก API นี้ช่วยให้สามารถตรวจสอบรายการกราฟทั้งหมดที่มีอยู่ เหมาะสำหรับผู้ดูแลระบบที่ต้องการดูข้อมูลกราฟที่สร้างไว้เพื่อใช้ในการแสดงผลข้อมูลจากอุปกรณ์ต่าง ๆ
- GET/{id} ใช้ Method GET พร้อม id ของอุปกรณ์เป็นพารามิเตอร์ (deviceId) เพื่อดึงข้อมูลกราฟที่เกี่ยวข้องกับอุปกรณ์ IoT นั้นโดยเฉพาะ API นี้ใช้คำสั่ง Chart.find({ deviceId: id }) เพื่อค้นหาข้อมูลของกราฟที่เชื่อมโยงกับอุปกรณ์นั้น หากไม่พบข้อมูล ระบบจะส่งข้อความ "Not Found"
- DELETE/{id} ใช้ Method DELETE พร้อม id เป็นพารามิเตอร์เพื่อระบุกราฟที่ต้องการลบ และใช้คำสั่ง Chart.deleteOne({ _id: id }) เพื่อลบข้อมูลกราฟที่มี id ตรงกับที่ได้รับมา

7) การออกแบบ API สำหรับการจัดการนำเข้าอุปกรณ์จากภายนอก (External Device) นี้ถูกออกแบบมาเพื่อรองรับการ GET POST DELETE ที่เชื่อมต่อกับระบบผ่าน MQTT Broker และ Wi-Fi โดยใช้ฐานข้อมูล MongoDB ผ่าน Mongoose ซึ่งแต่ละฟังก์ชันใน API นี้ทำหน้าที่แตกต่างกันไป ดังนี้

- POST ใช้ Method POST เพื่อนำเข้าข้อมูลของอุปกรณ์ใหม่เข้าสู่ฐานข้อมูล โดยต้องส่งข้อมูลในรูปแบบ JSON ซึ่งประกอบไปด้วย deviceId userId name topic broker connectPath username password status wifiId และ wifiConnect เมื่อ API ได้รับข้อมูลแล้ว ระบบจะใช้ฟังก์ชัน ExternalDevice.create() เพื่อบันทึกอุปกรณ์ลงในฐานข้อมูล เพื่อยืนยันการสร้างอุปกรณ์สำเร็จ
- GET ใช้ Method GET เพื่อดึงข้อมูลอุปกรณ์ภายนอกทั้งหมดที่ถูกบันทึกไว้ในฐานข้อมูล API นี้ไม่มีการรับพารามิเตอร์ และจะใช้คำสั่ง ExternalDevice.find() เพื่อค้นหาข้อมูลอุปกรณ์ทั้งหมดในคอลเลกชัน ExternalDevice จากนั้นส่งคืนข้อมูลที่พบในรูปแบบ JSON การเรียกใช้ API นี้ช่วยให้สามารถตรวจสอบรายการอุปกรณ์ทั้งหมดที่มีอยู่ในระบบ เหมาะสำหรับผู้ดูแลระบบที่ต้องการดูข้อมูลอุปกรณ์ทั้งหมดที่ลงทะเบียนไว้
- DELETE ใช้ Method DELETE เพื่อลบอุปกรณ์ที่ระบุโดยใช้ id เพื่อเจาะจงไปกับอุปกรณ์ซึ่งจะถูกส่งมาเป็นพารามิเตอร์ผ่าน query string และใช้คำสั่ง ExternalDevice.findByIdAndDelete(id) เพื่อลบข้อมูลอุปกรณ์ที่มี id ตรงกับที่ได้รับมา
- GET/{userId} ใช้ Method GET พร้อม userId เป็นพารามิเตอร์เพื่อดึงข้อมูลอุปกรณ์ทั้งหมดที่เป็นของผู้ใช้รายนั้น API นี้จะค้นหาข้อมูลโดยใช้คำสั่ง ExternalDevice.find({ userId: userId }) ซึ่งช่วยให้สามารถดึงเฉพาะอุปกรณ์ที่เชื่อมโยงกับผู้ใช้แต่ละรายได้ หากไม่พบข้อมูล ระบบจะส่งข้อความ "Not Found"

3.2.2 การนำเข้าอุปกรณ์ตัวอย่าง

ฟังก์ชันการนำเข้าอุปกรณ์ตัวอย่าง ได้แก่ รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ เริ่มต้นจากการเข้าถึงหน้าแดชบอร์ดของอุปกรณ์ หลังจากนั้นผู้ใช้จะต้องเลือกตัวเลือก "Import Example Device" เพื่อเพิ่มอุปกรณ์ใหม่ เมื่อเข้าสู่ขั้นตอนนี้ ระบบจะให้ผู้ใช้เลือกประเภทของอุปกรณ์ที่ต้องการนำเข้า เช่น รถยนต์บังคับ แขนกล หรืออุปกรณ์อื่นๆ ที่รองรับ หลังจากนั้นผู้ใช้ต้องเลือกประเภทของการทำงานของอุปกรณ์เป็นตัวส่งคำสั่งหรือตัวรับคำสั่ง ในขั้นตอนถัดไป ผู้ใช้จะต้องระบุชื่ออุปกรณ์และ Serial ID ของอุปกรณ์ที่ต้องการนำเข้า ระบบจะตรวจสอบว่า Serial ID ดังกล่าวเคยถูกใช้งานมาก่อนหรือไม่ หากหมายเลขซีเรียลซ้ำ ระบบจะแสดง "Error Alert" เพื่อแจ้งเตือนให้ผู้ใช้ทำการแก้ไขหมายเลขซีเรียลใหม่ แต่หากหมายเลขซีเรียลยังไม่เคยถูกใช้ ระบบจะดึงข้อมูลจากฐานข้อมูลและดำเนินการสร้างข้อมูลที่จำเป็นสำหรับอุปกรณ์นั้น กระบวนการนี้จะได้รับการตอบกลับจากฐานข้อมูล หลังจากที่ระบบสร้างข้อมูลเสร็จสิ้น ระบบจะดำเนินการบันทึกข้อมูลและเชื่อมต่ออุปกรณ์เข้าสู่ระบบ โดยจะส่งข้อมูลไปยังฐานข้อมูลเพื่อจัดเก็บและใช้งานต่อไป เมื่อทุกอย่างเสร็จสมบูรณ์อุปกรณ์จะพร้อมใช้งาน และผู้ใช้จะถูกนำไปยังหน้าจัดการอุปกรณ์ Device Page ซึ่งสามารถควบคุมหรือปรับแต่งการทำงานของอุปกรณ์ที่เพิ่มเข้ามาได้ โดยมีกระบวนการทำงานดังรูปที่ 3.61

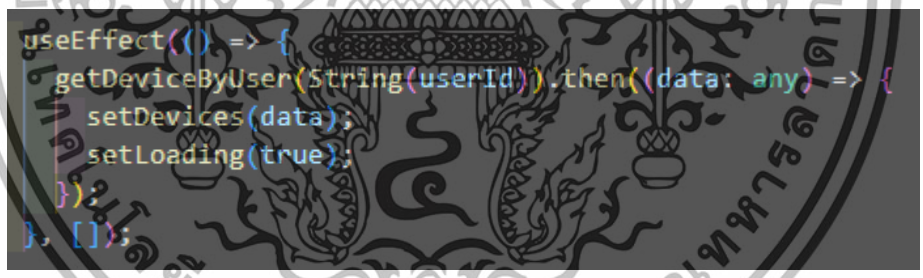


รูปที่ 3.61 กระบวนการนำเข้าอุปกรณ์ตัวอย่าง

ในการดำเนินการสร้างฟังก์ชันเพื่อนำเข้าอุปกรณ์ตัวอย่างจะเริ่มจากดึงข้อมูลอุปกรณ์ของผู้ใช้ผ่าน getDeviceByUser เมื่อล็อกอินเข้าแอป ระบบจะดึงข้อมูลอุปกรณ์ของผู้ใช้โดยใช้ฟังก์ชัน getDeviceByUser(userId) ซึ่งส่งคำขอ (fetch) ไปยัง API /api/deviceByUser/{id} เพื่อรับข้อมูลอุปกรณ์ทั้งหมดที่เป็นของผู้ใช้ ดังรูปที่ 3.62

รูปที่ 3.62 คำสั่งโปรแกรมดึงข้อมูลอุปกรณ์ของผู้ใช้ผ่าน getDeviceByUser

ใช้ useEffect เพื่อให้ฟังก์ชัน getDeviceByUser(userId) ทำงานเมื่อ component ถูกโหลด โดย userId จะถูกแปลงเป็น string แล้วส่งไปดึงข้อมูลอุปกรณ์ของผู้ใช้จาก API ซึ่งข้อมูลที่ได้จะถูกนำไปเซ็ตใน setDevices(data) และเปลี่ยนสถานะ setLoading(true); เพื่อระบุว่าการโหลดข้อมูลเสร็จสมบูรณ์แล้ว ดังรูปที่ 3.63



รูปที่ 3.63 คำสั่งโปรแกรมเพื่อระบุว่าการโหลดข้อมูลเสร็จสมบูรณ์แล้ว

ฟังก์ชันนี้ใช้สำหรับกำหนดค่าให้ device_name และ product_id เมื่อผู้ใช้ต้องการเพิ่มอุปกรณ์ใหม่ โดยจะเซ็ตค่า setDeviceName(name); และ setProductId(id); แล้วเปลี่ยนค่า setTriggerSubmit(true); เพื่อให้ useEffect ตรวจสอบและเรียกใช้งานฟังก์ชัน addDeviceSubmit() ดังรูปที่ 3.64

```
const getAddDevicePopUp = (name: string, id: string) => {
  setDeviceName(name);
  setProductId(id);
  setTriggerSubmit(true);
};
```

รูปที่ 3.64 คำสั่งโปรแกรมเมื่อผู้ใช้ต้องการเพิ่มอุปกรณ์ใหม่

ฟังก์ชันเพิ่มอุปกรณ์ (addDeviceSubmit) แบ่งการทำงานเป็น 3 ส่วน ตรวจสอบข้อมูลอุปกรณ์ คือ device_name และ product_id เมื่อเกิดข้อผิดพลาดจะแสดง toast.error() ดังรูปที่ 3.65

```
const addDeviceSubmit = async () => {
  setLoadingButton();
  if (!device_name) {
    toast.error("Please Set Device Name into Your Device.");
  }
  if (!product_id) {
    toast.error("not have product id.");
  }
```

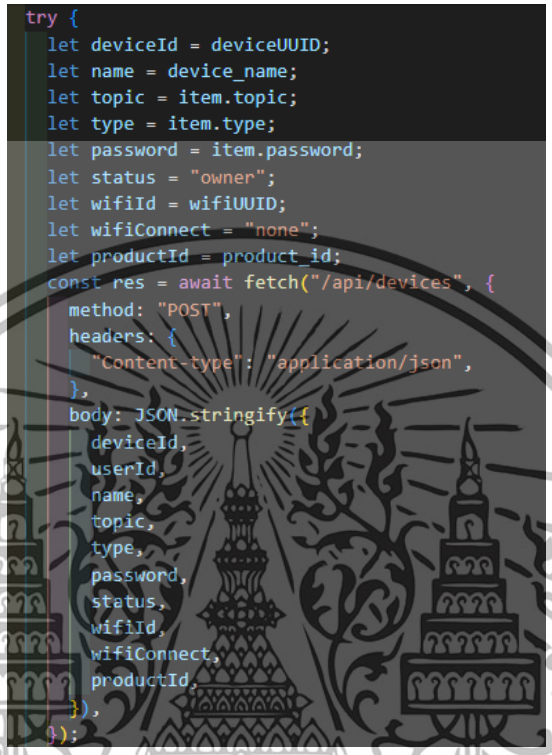
รูปที่ 3.65 คำสั่งโปรแกรมตรวจสอบข้อมูลอุปกรณ์ตรวจสอบ

ถ้า product_id มีค่า จะเรียก getProductId(product_id); เพื่อดึงข้อมูลของสินค้าจาก API และตรวจสอบว่าอุปกรณ์นี้มีเจ้าของแล้วหรือยัง โดยดูจากค่าของ ownerStatus ถ้ายังไม่มีเจ้าของ ownerStatus == false จะสามารถดำเนินการเพิ่มอุปกรณ์ได้ และสร้าง UUID สำหรับอุปกรณ์และ WiFi ดังรูปที่ 3.66

```
if (product_id) {
  try {
    const item = await getProductId(product_id);
    if (item?.ownerStatus == false) {
      const deviceUUID = uuidv4();
      const wifiUUID = uuidv4();
```

รูปที่ 3.66 คำสั่งโปรแกรมตรวจสอบสถานะอุปกรณ์และสร้าง UUID สำหรับอุปกรณ์และ WiFi

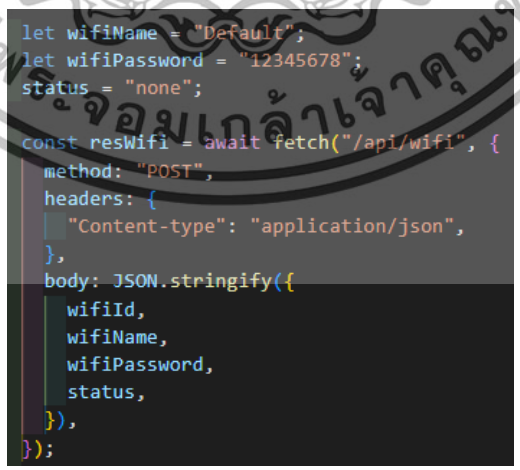
ฟังก์ชันนี้จะสร้าง fetch request ไปที่ API /api/devices ด้วย POST Method โดยจะส่งข้อมูลไป ดังรูปที่ 3.67



```
try {
  let deviceId = deviceUUID;
  let name = device_name;
  let topic = item.topic;
  let type = item.type;
  let password = item.password;
  let status = "owner";
  let wifiId = wifiUUID;
  let wifiConnect = "none";
  let productId = product_id;
  const res = await fetch("/api/devices", {
    method: "POST",
    headers: {
      "Content-type": "application/json",
    },
    body: JSON.stringify({
      deviceId,
      userId,
      name,
      topic,
      type,
      password,
      status,
      wifiId,
      wifiConnect,
      productId,
    }),
  });
```

รูปที่ 3.67 คำสั่งโปรแกรม POST Method โดยจะส่งข้อมูลเข้าฐานข้อมูล

และทำการเพิ่มข้อมูล WiFi ค่าเริ่มต้นใน /api/wifi เมื่ออุปกรณ์ถูกสร้างขึ้น จากนั้นใช้ fetch request POST เพื่อบันทึกข้อมูลไปยัง API /api/wifi ดังรูปที่ 3.68



```
let wifiName = "Default";
let wifiPassword = "12345678";
status = "none";

const resWifi = await fetch("/api/wifi", {
  method: "POST",
  headers: {
    "Content-type": "application/json",
  },
  body: JSON.stringify({
    wifiId,
    wifiName,
    wifiPassword,
    status,
  }),
});
```

รูปที่ 3.68 คำสั่งโปรแกรมเพื่อบันทึกข้อมูลไปยัง API

เมื่ออุปกรณ์ถูกเพิ่มเรียบร้อย จะต้องเปลี่ยนสถานะ ownerStatus ของสินค้าให้เป็น true ใช้ fetch ส่งคำขอ PUT ไปยัง /api/products/${product_id} เพื่ออัปเดตสถานะการเป็นเจ้าของ ดังรูปที่ 3.69

```
let newOwnerStatus = true;
const resProduct = await fetch(`/api/products/${product_id}`, {
  method: "PUT",
  headers: {
    "Content-type": "application/json",
  },
  body: JSON.stringify({
    newOwnerStatus,
  }),
});
```

รูปที่ 3.69 คำสั่งโปรแกรมส่งคำขอ PUT ไปยัง API

ตรวจสอบว่าการเพิ่มอุปกรณ์ตัวอย่างนั้นสำเร็จหรือไม่ ถ้า res.ok resWifi.ok และ resProduct.ok ทั้งหมดเป็น true หมายความว่าอุปกรณ์ถูกเพิ่มสำเร็จ Import Example Device success ดังรูปที่ 3.70

```
if (res.ok && resWifi.ok && resProduct.ok) {
  toast.success("Add Device success.");
  unsetLoadingButton();
  onClosePopUp();
} else {
  toast.error("Errors something went wrong.");
}
```

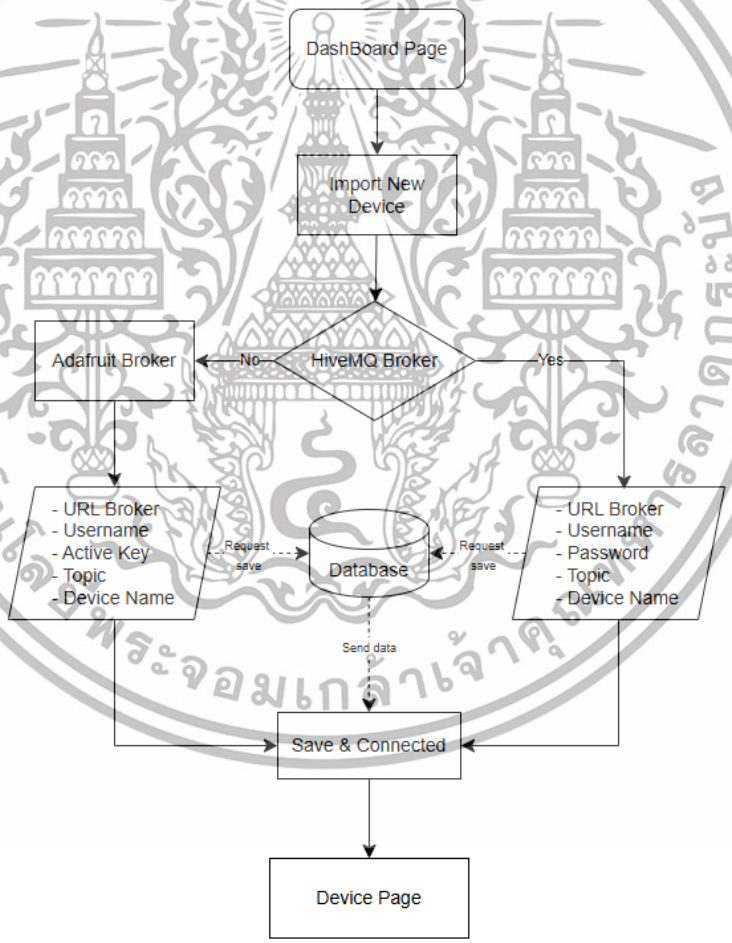
รูปที่ 3.70 คำสั่งโปรแกรมตรวจสอบว่าการเพิ่มอุปกรณ์สำเร็จ

และเมื่อได้ทำการเพิ่มอุปกรณ์เสร็จสิ้นสมบูรณ์แล้ว หน้ารวมของอุปกรณ์จะแสดงอุปกรณ์ตัวอย่างที่ได้นำเข้าจะปรากฏขึ้นทันทีที่นำเข้าสู่แพลตฟอร์มสำเร็จ

3.2.3 การนำเข้าอุปกรณ์จากภายนอก

ฟังก์ชันการนำเข้าอุปกรณ์จากภายนอกถูกออกแบบมาเพื่อรองรับความต้องการของผู้ใช้งานในกรณีที่ผู้ใช้ต้องการใช้อุปกรณ์ IoT ที่ตนเองพัฒนาร่วมกับเว็บแอปพลิเคชัน หรืออุปกรณ์ที่รองรับ MQTT มาใช้งานร่วมกับเว็บแอปพลิเคชันได้อย่างสะดวกโดยเริ่มต้นจากหน้า Dashboard Page ซึ่งเป็นหน้าหลักของระบบที่ผู้ใช้สามารถดำเนินการนำเข้าอุปกรณ์ภายนอกได้ เมื่อผู้ใช้เลือก Import New Device ระบบจะเข้าสู่หน้าเว็บแอปพลิเคชัน โดยผู้ใช้งานจะต้องเลือกว่าอุปกรณ์ที่นำเข้านั้นต้องการเชื่อมต่อกับ HiveMQ Broker หรือไม่ หากอุปกรณ์ต้องการใช้ HiveMQ Broker

ระบบจะทำการขอข้อมูลที่จำเป็น ได้แก่ ที่อยู่ของ Broker Username Password Topic ที่ใช้สื่อสาร และชื่อของอุปกรณ์ ซึ่งข้อมูลเหล่านี้จะถูกบันทึกลงใน Database ผ่านการส่งคำขอไปยังฐานข้อมูลเพื่อจัดเก็บข้อมูลไว้ ในกรณีที่ผู้ใช้งานเลือกใช้ Adafruit Broker แทน และต้องดำเนินการในลักษณะเดียวกัน โดยทำการขอข้อมูลที่จำเป็น ได้แก่ ที่อยู่ของ Broker Active Key Password Topic และชื่ออุปกรณ์ แล้วทำการบันทึกลงฐานข้อมูลผ่านการส่งคำขอเช่นเดียวกับกรณีของ HiveMQ เมื่อข้อมูลถูกบันทึกเรียบร้อยแล้ว ฐานข้อมูลจะส่งข้อมูลกลับไปยังระบบ และกระบวนการจะดำเนินต่อไป เพื่อยืนยันว่าการบันทึกข้อมูลและการเชื่อมต่อเสร็จสมบูรณ์ จากนั้นระบบจะนำผู้ใช้ไปยัง Device Page ซึ่งเป็นหน้าที่ใช้ในการจัดการและควบคุมอุปกรณ์ที่เชื่อมต่อแล้ว โดยมีกระบวนการทำงานดังรูปที่ 3.71



รูปที่ 3.71 กระบวนการนำเข้าอุปกรณ์จากภายนอก

ในการดำเนินการสร้างฟังก์ชันเพื่อนำเข้าอุปกรณ์จากภายนอกโดยใช้ useState เก็บค่าของ MQTT Broker และข้อมูลที่ใช้ในการเชื่อมต่อ คือ newBroker บันทึกประเภทของ Broker จากตัวเลือก HiveMQ Broker หรือ Adafruit Broker และบันทึกค่าที่ต้องใช้สำหรับการเชื่อมต่อ ได้แก่ newUsernameBroker เพื่อเก็บชื่อผู้ใช้ที่เชื่อมต่อกับ Broker newPasswordBroker เพื่อเก็บรหัสผ่านผู้ใช้ที่เชื่อมต่อกับ Broker newTopic เพื่อเก็บหัวข้อที่ใช้ในการสื่อสาร newEndPoint เพื่อเก็บที่อยู่ที่เชื่อมต่อกับ Broker และ newDeviceName เพื่อเก็บชื่ออุปกรณ์ ดังรูปที่ 3.72 จากนั้นทำการสร้างหน้าต่างให้ผู้ใช้เลือก Broker ดังรูปที่ 3.73

```
const [newBroker, setNewBroker] = useState<string>("");
const [newUsernameBroker, setNewUsernameBroker] = useState<string>("");
const [newPasswordBroker, setNewPasswordBroker] = useState<string>("");
const [newTopic, setNewTopic] = useState<string>("");
const [newEndPoint, setNewEndPoint] = useState<string>("");
const [newDeviceName, setNewDeviceName] = useState<string>("");
```

รูปที่ 3.72 คำสั่งโปรแกรมเก็บค่าการเชื่อมต่อกับ MQTT Broker

```
<div className="flex gap-4 ">
  <div className="flex gap-2 items-center">
    <label className="text-lg text-white ">HIVE MQ</label>
    <input
      type="radio"
      name="broker"
      onChange={(e) => setNewBroker(e.target.value)}
      value="hivemq"
      className="w-4 h-4 shadow-none"
    />
  </div>
  <div className="flex gap-2 items-center">
    <label className="text-lg text-white ">Ada-fruite</label>
    <input
      type="radio"
      name="broker"
      onChange={(e) => setNewBroker(e.target.value)}
      value="adafruite"
      defaultValue={newBroker || ""}
      className="w-4 h-4 shadow-none"
    />
  </div>
</div>
```

รูปที่ 3.73 คำสั่งโปรแกรมสร้างหน้าต่างเลือกการเชื่อมต่อ MQTT Broker

โดยในการเชื่อมต่อกับ Broker จะต้องสร้างช่องรับข้อมูลที่ผู้ใช้จะต้องกรอก ที่อยู่ของ Broker Username Password Topic และชื่ออุปกรณ์เพื่อใช้ในการสื่อสาร หากผู้ใช้งานเลือก HiveMQ Broker ระบบจะขอ MQTT End Point Username Password และ Topic ในการสื่อสาร ดังรูปที่ 3.74 แต่ถ้าหากเลือก Adafruit Broker ระบบจะขอ MQTT End Point Username Active Key และ Topic ดังรูปที่ 3.75

รูปที่ 3.74 คำสั่งโปรแกรมสร้างช่องรับข้อมูลในการเชื่อมต่อ HiveMQ Broker

```
    <hr className="w-full mt-2 text-white" />
  </div>
) : newBroker == "adafruite" ? (
  <div className="grid gap-2">
    <h1 className="font-semibold text-blue-500  text-lg">
      Ada-Fruite Connection
    </h1>
    <div className="grid lg:flex gap-2 text-white">
      <label className="w-[140px]">MQTT EndPiont :</label>
      <input
        type="text"
        className="rounded-sm bg-gray-300 text-sm lg:w-[300px] md:w-[300px]  text-gray-800 px-2"
        onChange={(e) => setNewEndPoint(e.target.value)}
        value={newEndPoint}
      />
    </div>
    <div className="grid lg:flex gap-2 text-white">
      <label className="w-[140px]">Username :</label>
      <input
        type="text"
        className="rounded-sm bg-gray-300 text-sm lg:w-[200px] md:w-[200px]  text-gray-800 px-2"
        onChange={(e) => setNewUsernameBroker(e.target.value)}
        value={newUsernameBroker}
      />
    </div>
    <div className="grid lg:flex gap-2 text-white">
      <label className="w-[140px]">Key :</label>
      <input
        type="text"
        className="rounded-sm bg-gray-300 text-sm lg:w-[200px] md:w-[200px]  text-gray-800 px-2"
        onChange={(e) => setNewPasswordBroker(e.target.value)}
        value={newPasswordBroker}
      />
    </div>
    <hr className="w-full mt-2 text-white" />
  </div>
```

รูปที่ 3.75 คำสั่งโปรแกรมสร้างช่องรับข้อมูลในการเชื่อมต่อ Adafruit Broker

เมื่อมีการเปิดหน้าอุปกรณ์ที่นำเข้ามาจะทำการดึงข้อมูลอุปกรณ์ที่นำเข้ามาตาม ID ของผู้ใช้งานในระบบจากฐานข้อมูลที่ผู้ใช้ได้ลงทะเบียนไว้ ดังรูปที่ 3.76 และใช้ useEffect เพื่อเรียกใช้งาน getExternalDevice(userId) และอัปเดต state devices ดังรูปที่ 3.77

```
const getExternalDevice = async (userid: string) => {
  try {
    const result = await fetch(`/api/importDevice/${userid}`);
    return await result.json();
  } catch (error) {
    console.log(error instanceof Error ? error.message : "Unknown error");
    return [];
  }
};
```

รูปที่ 3.76 คำสั่งโปรแกรมดึงข้อมูลอุปกรณ์ที่นำเข้ามาตาม ID ของผู้ใช้งาน

```
useEffect(() => {
  getExternalDevice(String(userId)).then((item: any) => {
    setDevices(item);
    setLoading(true);
  });
}, []);
```

รูปที่ 3.77 คำสั่งโปรแกรมเพื่อเรียก getExternalDevice(userId) และอัปเดต state devices

จากนั้นในการแสดงผลข้อมูลจะทำการตรวจสอบ topic และ deviceData.name มีค่าหรือไม่ ถ้ามีค่าทั้งคู่จะทำการแสดงคอมโพแนนต์ Panel ที่จะแสดงข้อมูลของอุปกรณ์ต่อไป มีคำสั่งการเขียนโปรมแกรม ดังรูปที่ 3.78

```
<div className="lg:flex md:flex justify-center    w-fit lg:w-fit lg:py-5">
  {topic && deviceData?.name && (
    <Panel
      isConnected={isConnected}
      client={client}
      topic={topic}
      isLoading={isLoading}
      device_id={deviceId}
      device_log={returnedLog}
      device_connect={deviceConnected}
      broker={broker}
      connectPath={connectPath}
      username={username}
      password={password}
      deviceName={deviceData?.name}
    />
  )}
</div>
```

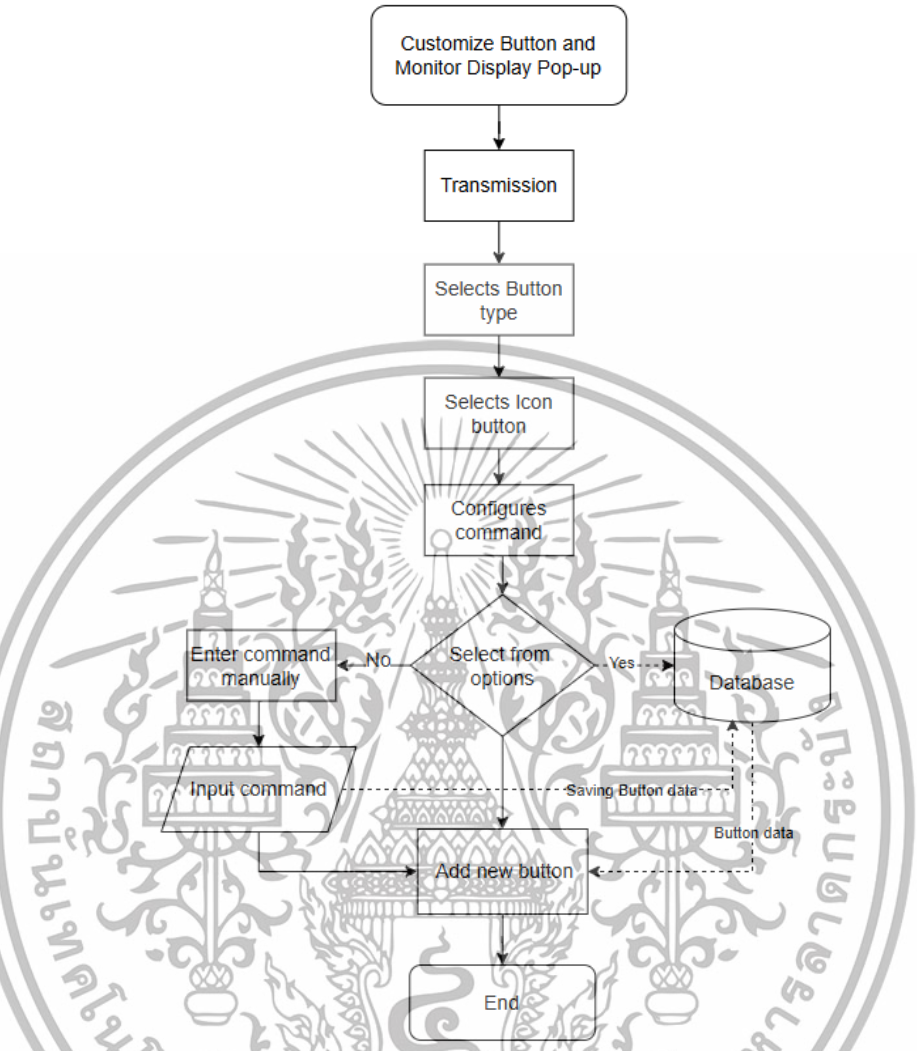
รูปที่ 3.78 คำสั่งโปรแกรมแสดงข้อมูลของอุปกรณ์นำเข้า

3.2.4 การปรับแต่งปุ่มสั่งการและหน้าแสดงผล

การออกแบบปรับแต่งปุ่มสั่งการและหน้าแสดงผลแบ่งส่วนในการออกแบบเป็น 2 ส่วน คือ การปรับแต่งปุ่มสั่งการและหน้าแสดงผลการทำงานของอุปกรณ์ IoT โดยมีรายละเอียด ดังนี้

3.2.4.1 การทำงานของฟังก์ชันปรับแต่งปุ่มสั่งการ

ในการออกแบบฟังก์ชันปรับแต่งปุ่มสั่งการมีการออกแบบการทำงาน โดยเมื่อผู้ใช้คลิกเพื่อเพิ่มปุ่มใหม่ ระบบจะแสดงหน้าต่างที่ประกอบด้วย dropdown menus หลายตัวเพื่อให้ผู้ใช้เลือกประเภทของปุ่ม (Type) หมวดหมู่ (Category) และรูปไอคอนปุ่ม (Lable) ของปุ่มนั้น ๆ ระบบจะให้ผู้ใช้เลือกคำสั่ง (Command) ที่เกี่ยวข้องกับปุ่มนั้น ๆ เช่น "On" "Off" "Up" "Down" เป็นต้น นอกจากนี้ ผู้ใช้ยังสามารถกำหนดคำสั่งเองได้โดยการคลิกที่ปุ่ม "Config Command" และกรอกคำสั่งที่ต้องการลงในช่องข้อความ โดยเมื่อผู้ใช้กรอกข้อมูลครบถ้วนและกดบันทึก ระบบจะทำการบันทึกข้อมูลปุ่มใหม่ลงในฐานข้อมูล โดยข้อมูลที่บันทึกจะประกอบด้วย ID ของปุ่ม ประเภท หมวดหมู่ รูปไอคอนปุ่ม คำสั่ง และ ID ของอุปกรณ์ที่ปุ่มนี้เกี่ยวข้อง หลังจากบันทึกข้อมูลเสร็จสิ้น ระบบจะแสดงข้อความแจ้งเตือนว่าการเพิ่มปุ่มสำเร็จและปุ่มใหม่จะถูกเพิ่มเข้าไปในรายการปุ่มที่มีอยู่ มีกระบวนการทำงาน ดังรูปที่ 3.79



รูปที่ 3.79 กระบวนการทำงานของฟังก์ชันปรับแต่งปุ่มสั่งการ

ในการดำเนินการสร้างฟังก์ชันปรับแต่งปุ่มสั่งการเริ่มโดยการการเก็บค่าที่ผู้ใช้เลือกผ่านหน้าต่างรับค่าหน้าเว็บแอปพลิเคชัน ซึ่งเป็นหน้าต่างที่ให้ผู้ใช้เลือกประเภทของปุ่ม (type) หมวดหมู่ (category) รูปไอคอนปุ่ม (label) และคำสั่ง (command) โดยที่ข้อมูลที่ผู้ใช้เลือกจะถูกเก็บใน State ของ React ผ่าน useState ดังรูปที่ 3.80

```
const PopUpAddButton = ({ setPopUpBtn, setButtons, deviceId }: PopUpBtnProps) => {
  const [category, setButtonCategory] = React.useState<string>("");
  const [label, setButtonLabel] = React.useState<string>("");
  const [command, setButtonCommand] = React.useState<string>("");
  const [type, setButtonType] = React.useState<string>("");
  const customizeBtn: CustomizeButtonModel = {
```

รูปที่ 3.80 คำสั่งโปรแกรมในการเก็บข้อมูลที่ผู้ใช้เลือก

เพื่อความสะดวกและความง่ายต่อการใช้งานของผู้ใช้จะใช้ select dropdown เพื่อให้ผู้ใช้เลือกประเภทและหมวดหมู่ของปุ่ม เมื่อผู้ใช้เลือก type ระบบจะอัปเดตค่าหมวดหมู่ (category) และรูปไอคอนปุ่ม (label) ตามที่กำหนดไว้ใน CustomizeButtonModel ดังรูปที่ 3.81 หากปุ่มเป็นประเภท transmitter และต้องการกำหนดคำสั่งเอง ระบบจะแสดง input field ให้ผู้ใช้พิมพ์คำสั่งแบบกำหนดเองได้ ดังรูปที่ 3.82

```
<select
  className="px-4 py-1 border rounded mb-2"
  value={selectedType || ""}
  onChange={handleTypeChange}
>
  <option value="">--Type--</option>
  {Object.keys(customizeBtn).map((type) => (
    <option key={type} value={type}>
      {type}
    </option>
  ))}
</select>
```

รูปที่ 3.81 ตัวอย่างคำสั่งโปรแกรมในการให้ผู้ใช้เลือกค่าต่าง ๆ

```
<label className="text-white">Command: </label>
<input
  type="text"
  className="py-1 lg:w-[120px] bg-black text-green-400 w-[120px] rounded-sm px-4"
  placeholder=">/"
  value={command}
  onChange={(e) => setButtonCommand(e.target.value)}
/>
```

รูปที่ 3.82 คำสั่งโปรแกรมในการให้ผู้ใช้กำหนดคำสั่งการ

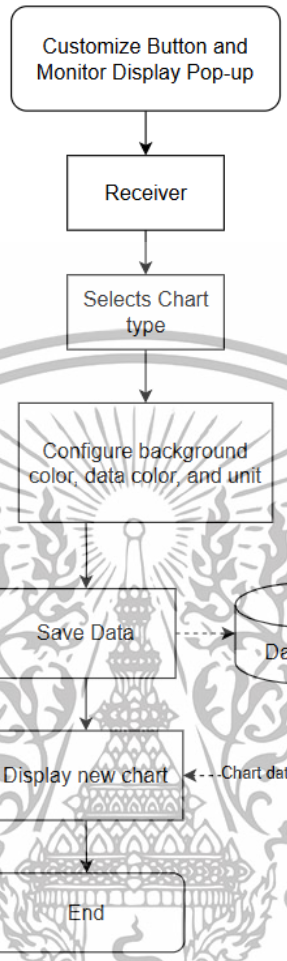
เมื่อผู้ใช้กดบันทึกหลังจากตั้งค่าปุ่มสั่งการเสร็จแล้ว โดยคำสั่งโปรแกรมนี้ทำหน้าที่ส่งข้อมูลของปุ่มใหม่ไปยังเซิร์ฟเวอร์ผ่าน API เพื่อบันทึกลงฐานข้อมูล หาก API ตอบกลับสำเร็จ ระบบจะแสดงข้อความแจ้งเตือน "Add button success" ด้วย toast.success() เพื่อให้ผู้ใช้ทราบว่าการเพิ่มปุ่มเสร็จสมบูรณ์ แต่ถ้าเกิดข้อผิดพลาด เช่น การเชื่อมต่อมีปัญหา หรือ API ไม่สามารถบันทึกข้อมูลได้ ระบบจะแสดงข้อความ "Failed" ด้วย toast.error() เพื่อแจ้งให้ผู้ใช้ทราบว่าการเพิ่มปุ่มล้มเหลว ดังรูปที่ 3.83

```
const resAddButton = await fetch("/api/button", {
  method: "POST",
  headers: {
    "Content-type": "application/json",
  },
  body: JSON.stringify({
    id,
    type,
    category,
    label,
    command,
    deviceId,
  }),
});
if (resAddButton.ok) {
  toast.success("Add buttom success");
} else {
  toast.error("Failed");
}
```

รูปที่ 3.83 คำสั่งโปรแกรมส่งข้อมูลของปุ่มใหม่ไปยังเซิร์ฟเวอร์ผ่าน API

3.2.4.2 การทำงานของฟังก์ชันปรับแต่งหน้าแสดงผล

ในการออกแบบฟังก์ชันปรับแต่งหน้าแสดงผลมีการออกแบบการทำงานดังรูปที่ 3.84 ระบบจะเริ่มต้นทำงานเมื่อผู้ใช้ต้องการเพิ่มหน้าจอแสดงผลใหม่ โดยเมื่อผู้ใช้กดปุ่มเพิ่ม ระบบจะแสดงหน้าต่างขึ้นมาเพื่อให้ผู้ใช้เลือกและกำหนดค่าต่าง ๆ ของหน้าจอแสดงผลที่ต้องการ ในหน้าต่างนี้จะมีตัวเลือกให้ผู้ใช้เลือกประเภทของการแสดงผล เช่น Donut Chart Circle Chart และ Circle Display โดยแต่ละประเภทจะมีรูปแบบการนำเสนอข้อมูลที่แตกต่างกัน เมื่อผู้ใช้เลือกประเภทของการแสดงผลแล้ว ระบบจะบันทึกค่าที่เลือกไว้เพื่อนำไปใช้ในการสร้างหน้าจอแสดงผลใหม่ หากผู้ใช้เลือกประเภทการแสดงผลแบบ Circle Display ระบบจะให้ผู้ใช้กำหนดค่าพื้นฐานเพิ่มเติม เช่น สีพื้นหลัง (Background Color) สีตัวอักษร (Font Color) และหน่วยของค่าที่แสดง (Unit) โดยผู้ใช้สามารถเลือกสีพื้นหลังจากสีที่มีให้ และเลือกสีตัวอักษรเป็นสีดำหรือสีขาว นอกจากนี้ ผู้ใช้สามารถกำหนดหน่วยของค่าที่ต้องการแสดง เช่น องศาเซลเซียส (°C) โวลต์ (V) แอมป์ (A) หรือ เฮิรตซ์ (Hz) ซึ่งค่าต่าง ๆ ที่ผู้ใช้เลือกจะถูกบันทึกลงในตัวแปรเพื่อใช้ในการสร้างและแสดงผลในระบบต่อไป เมื่อตั้งค่าต่าง ๆ เสร็จเรียบร้อยแล้ว หากผู้ใช้กด "Save" ระบบจะทำการสร้างข้อมูลของหน้าจอแสดงผลใหม่



รูปที่ 3.84 กระบวนการทำงานของฟังก์ชันปรับแต่งหน้าแสดงผล

ในการดำเนินการสร้างฟังก์ชันปรับแต่งหน้าแสดงผล เริ่มโดยการการเก็บค่าที่ผู้ใช้เลือกผ่านหน้าต่างกำหนดค่าเบื้องต้นหน้าเว็บแอปพลิเคชัน ซึ่งเป็นหน้าต่างที่ให้ผู้ใช้เลือกประเภทของการแสดงผล (customizeType) หน่วยของค่าที่แสดง (unit) สีพื้นหลัง (bgcolor) และสีตัวอักษร (fgcolor) ดังรูปที่ 3.85

```
const [customizeType, setCustomizeType] = useState<string | null>();
const [unit, setUnit] = useState<string>("");
const [bgcolor, setBGcolor] = useState<string>("");
const [fgcolor, setFGcolor] = useState<string>("");
```

รูปที่ 3.85 คำสั่งโปรแกรมกำหนดค่าเบื้องต้น

เมื่อผู้ใช้คลิกที่ปุ่ม Donut Chart Circle Chart หรือ Circle Display ค่าของ customizeType จะถูกตั้งค่าให้ตรงกับประเภทที่เลือก โดยใช้คำสั่งโปรแกรมดังรูปที่ 3.86 3.87 และ 3.88 ตามลำดับ

```
<button
  className={` ${
    customizeType == "donut" ? "bg-green-400" : "bg-gray-500"
  } px-4 py-1 text-white  rounded-2xl hover:opacity-75`}
  onClick={() => setCustomizeType("donut")}
>
  Donut Chart
</button>
```

รูปที่ 3.86 คำสั่งโปรแกรมเลือกแผนภูมิโดนัท

```
<button
  className={` ${
    customizeType == "circle" ? "bg-green-400" : "bg-gray-500"
  } px-4 py-1 text-white  rounded-2xl hover:opacity-75`}
  onClick={() => setCustomizeType("circle")}
>
  Circle Chart
</button>
```

รูปที่ 3.87 คำสั่งโปรแกรมเลือกแผนภูมิวงกลม

```
<button
  className={` ${
    customizeType == "monitorcircle"
      ? "bg-green-400"
      : "bg-gray-500"
  } px-4 py-1 text-white bg-blue-500 rounded-2xl hover:opacity-75 `}
  onClick={() => setCustomizeType("monitorcircle")}
>
  Circle Display
</button>
```

รูปที่ 3.88 คำสั่งโปรแกรมเลือกแสดงผลเป็นวงกลม

หลังจากผู้ใช้เลือกประเภทการแสดงผลแล้วสามารถเลือกตั้งค่าการแสดงผลเพิ่มเติม จะสามารปรับเปลี่ยนสีพื้นหลังของวงกลมที่แสดงผล ดังรูปที่ 3.89 สามารถเลือกสีตัวหนังสือและหน่วยในการแสดงผลได้ดังรูปที่ 3.90

```
<div className="text-lg font-semibold px-4 bg-gray-700 rounded-md text-center my-4">
  Monitor Adjust
</div>
<div className="grid lg:flex items-center  gap-2 ">
  <label>BG-Color :</label>
  <select
    className="px-2 text-white py-1 text-center rounded-md bg-gray-500"
    onChange={(e) => setBGcolor(e.target.value)}
  >
    <option className="bg-white text-black" value="">
      -select-
    </option>
    <option
      className="bg-[#00ccff] text-center font-semibold"
      value="bg-[#00ccff]"
    >
      sky
    </option>
    <option
      className="bg-[#ff0d0d] text-center font-semibold"
      value="bg-[#ff0d0d]"
```

รูปที่ 3.89 คำสั่งโปรแกรมตัวเลือกสีพื้นหลัง

```
  <label className="ml-4">Font-Color : </label>
  <select
    className="px-2 text-white py-1 text-center rounded-md bg-gray-500"
    onChange={(e) => setFGcolor(e.target.value)}
  >
    <option className="bg-white text-black" value="">
      -select-
    </option>
    <option className="bg-black text-white" value="text-white">
      white
    </option>
    <option className="bg-white text-black" value="text-black">
      black
    </option>
  </select>
</div>
<div className="flex items-center gap-2 my-2">
  <label className="text-white">Unit :</label>
  <input
    className="bg-gray-500 text-white w-[200px] rounded-md px-2 py-1 "
    type="text"
    onChange={(e) => setUnit(e.target.value)}
    value={unit}
  />
</div>
```

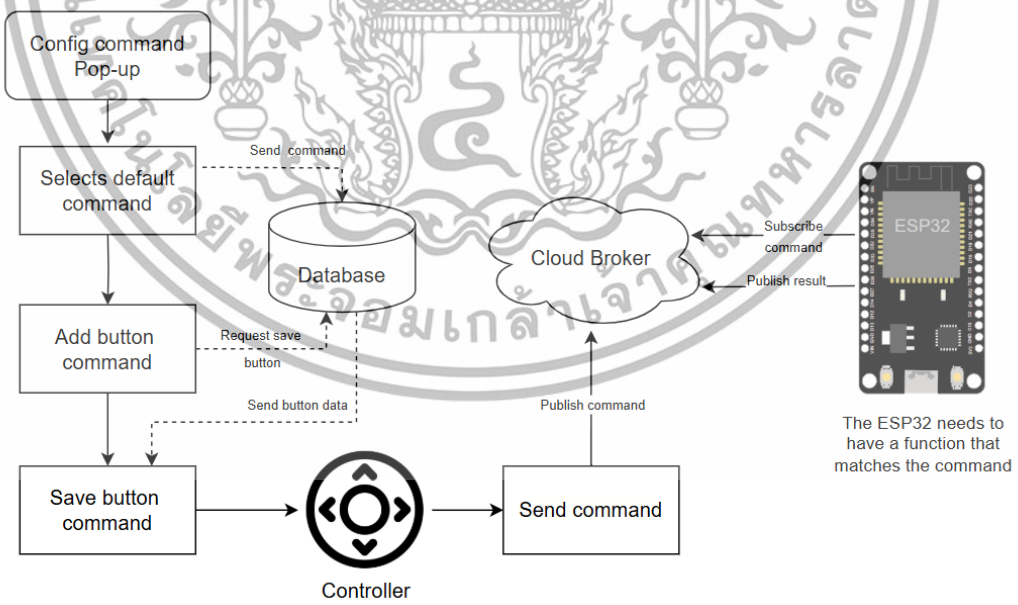
รูปที่ 3.90 คำสั่งโปรแกรมตัวเลือกสีตัวหนังสือและหน่วยแสดงผล

3.2.5 การกำหนดคำสั่งจากหน้าเว็บแอปพลิเคชัน

จากการปรับแต่งปุ่มสั่งการอุปกรณ์ IoT ผ่านหน้าเว็บแอปพลิเคชันที่สามารถกำหนดคำสั่งจากผู้ใช้ได้ผ่านหน้าต่างกำหนดคำสั่ง (Command) จึงเป็นส่งสำคัญในกรณีที่ผู้ใช้งานต้องการปรับเปลี่ยนคำสั่งสั่งการเพื่อสั่งงานให้อุปกรณ์ทำงานตามความต้องการ อีกทั้งทำให้การใช้งานภายในแพลตฟอร์มนั่นยืดหยุ่นขึ้นกว่าเดิม เหมาะแก่การรองรับอุปกรณ์เพิ่มเติมในอนาคต จึงมีการออกแบบและดำเนินการ ดังนี้

3.2.5.1 คำสั่งที่กำหนดไว้ภายในเว็บแอปพลิเคชัน

คำสั่งที่กำหนดไว้ภายในเว็บแอปพลิเคชันออกแบบเพื่อในกรณีที่ผู้ใช้งานไม่ต้องการปรับเปลี่ยนเองให้ยุ่งยาก แต่ต้องการใช้งานปุ่มตามความต้องการตนเองโดยไม่ต้องไปตั้งค่าเพิ่มเติมให้ยุ่งยาก มีขั้นตอนการทำงานเริ่มต้นจากผู้ใช้ที่กำหนดคำสั่งผ่านหน้าต่างหน้าเว็บแอปพลิเคชัน จากนั้นเมื่อบันทึกคำสั่งลงในฐานข้อมูล ระบบจะส่งคำขอเพื่อบันทึกข้อมูลคำสั่งลงในฐานข้อมูล และฐานข้อมูลจะจัดเก็บข้อมูลคำสั่งพร้อมส่งข้อมูลกลับไปยังระบบเพื่อยืนยันการบันทึก เมื่อผู้ใช้ต้องการส่งคำสั่งไปยัง ESP32 ระบบจะส่งคำสั่งไปยัง Cloud Broker ซึ่งทำหน้าที่เป็นตัวกลางในการส่งคำสั่งไปยัง ESP32 เมื่อได้รับคำสั่งแล้ว ESP32 จะประมวลผลตามฟังก์ชันที่กำหนดไว้ ดังรูปที่ 3.91



รูปที่ 3.91 กระบวนการทำงานของคำสั่งที่กำหนดไว้ภายในเว็บแอปพลิเคชัน

ในการดำเนินงานส่วนของคำสั่งที่ผู้ใช้เลือกจาก select dropdown มีหลายตัวเลือก เช่น "On" "Off" "Up" "Down" "Left" "Right" "Forward" และ "Backward" ซึ่งเป็นคำสั่งที่มีการตั้งค่าไว้เมื่อผู้ใช้เลือกคำสั่งจาก dropdown นี้ คำสั่งที่เลือกจะถูกส่งผ่านไปยังฟังก์ชัน setButtonCommand และเก็บค่าไว้ในตัวแปร buttonCommand ซึ่งเป็นตัวแปรที่ใช้เพื่อกำหนดคำสั่งที่ปุ่มจะทำงานให้กับ ESP32 เมื่อกดปุ่มที่สร้างขึ้นมาใน PressButton ฟังก์ชัน Control จะถูกเรียกใช้ และคำสั่งนี้จะถูกส่งผ่าน MQTT ไปยัง Cloud Broker เพื่อให้ ESP32 ประมวลผลและทำการสั่งงานตามคำสั่ง ดังรูปที่ 3.92

```
<select
  className="px-2 py-1 rounded-md bg-gray-500 text-white"
  onChange={(e) => setButtonCommand(e.target.value)}
>
  <option defaultValue={"None"}>select-command</option>
  <option value={"on"}>On</option>
  <option value={"off"}>Off</option>
  <option value={"up"}>Up</option>
  <option value={"down"}>Down</option>
  <option value={"left"}>Left</option>
  <option value={"right"}>Right</option>
  <option value={"forward"}>Forward</option>
  <option value={"backward"}>Backward</option>
</select>
```

รูปที่ 3.92 คำสั่งโปรแกรมสร้างหน้าต่างตัวเลือกคำสั่ง

เมื่อกดปุ่มบันทึก ฟังก์ชันนี้จะรับข้อมูลที่ผู้ใช้กรอกมาและบันทึกลงในฐานข้อมูล โดยสามารถใช้การเชื่อมต่อ API ที่เชื่อมต่อกับฐานข้อมูล ดังรูปที่ 3.93

```
const handleSave = async () => {
  const newButton: ButtonModel = {
    id,
    type,
    category,
    label,
    command,
    deviceId,
  };

  const resAddButton = await fetch("/api/button", {
    method: "POST",
    headers: {
      "Content-type": "application/json",
    },
    body: JSON.stringify({
      id,
      type,
      category,
      label,
      command,
      deviceId,
    }),
  });
```

รูปที่ 3.93 ฟังก์ชันบันทึกปุ่ม

เมื่อมีการคลิกปุ่มที่ได้ปรับแต่งปุ่มไว้ ฟังก์ชัน Control จะถูกเรียกใช้ ซึ่งในฟังก์ชันนี้จะทำการตรวจสอบว่า client และ isConnected เชื่อมต่ออยู่หรือไม่ หากเชื่อมต่อแล้วมันจะส่งคำสั่งผ่าน MQTT ไปยัง Cloud Broker ด้วยคำสั่ง client.publish ซึ่ง cmd คือค่าของคำสั่งที่ผู้ใช้เลือกหรือคำสั่งที่ผู้ใช้กำหนดเอง ดังรูปที่ 3.94

```
const Control = async (cmd :string) => {
  if (client && isConnected) {
    client.publish(topic, `ctrl/${cmd}`);
    console.log(`Press : ${cmd.toUpperCase()}`);
    onLogReturn(`Press : ${cmd.toUpperCase()}`);
  }
};
```

รูปที่ 3.94 ฟังก์ชันโปรแกรมส่งค่าคำสั่งขึ้น Broker

ที่ฝั่งจัดการเว็บแอปพลิเคชัน ข้อมูลที่ได้รับจะถูกจัดเก็บในฐานข้อมูล การทำงานนี้อาจจะเป็นการสร้างบันทึกใหม่ในตารางหรือคอลเล็กชันที่เกี่ยวข้องกับคำสั่งที่ผู้ใช้ได้ตั้งค่าไว้ และจะเรียกใช้เมื่อผู้ใช้งานเข้าสู่หน้าควบคุมอุปกรณ์ ดังรูปที่ 3.95 จะแสดงปุ่มที่และคำสั่งที่ผู้ใช้งานได้ปรับแต่งไว้ ดังรูปที่ 3.96

```
const fetchButton = async (id: string) => {
  const response = await fetch(`/api/button/${id}`);
  return response.json();
};
```

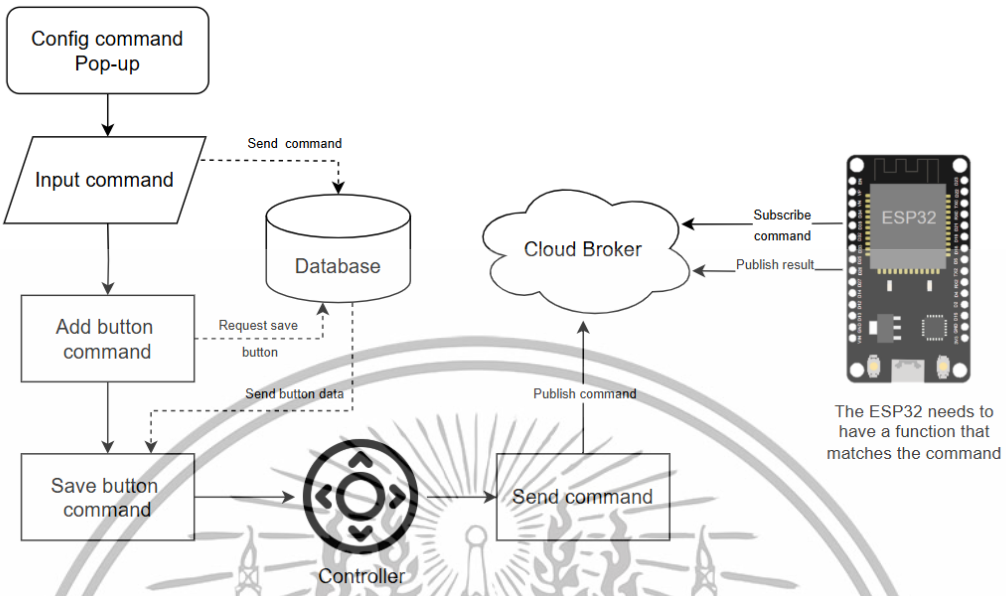
รูปที่ 3.95 คำสั่งโปรแกรมดึงข้อมูลปุ่มจาก API โดยใช้ ID ของปุ่มเป็นตัวระบุ

```
{item.type == "transmitter" ? (
  <div className="animate-fastFade grid place-items-center w-full">
    {item.category == "press" ? (
      <PressButton
        category={item.category}
        cmd={item.command}
        label={item.label}
        type={item.type}
        isConnected={isConnected}
        client={client}
        topic={topic}
        onLogReturn={getLogReturned}
      />
```

รูปที่ 3.96 คำสั่งโปรแกรมแสดงข้อมูลปุ่ม

3.2.5.2 คำสั่งที่ผู้ใช้กำหนดเอง

คำสั่งที่ผู้ใช้กำหนดเองออกแบบไว้เพื่อในกรณีที่ผู้ใช้งานต้องการกำหนดคำสั่งการใช้งานเอง หรือนำอุปกรณ์ที่ตนเองพัฒนามาใช้ร่วมกับแพลตฟอร์ม เพื่ออำนวยความสะดวกและให้เกิดความยืนหยุ่นในการทำงานของอุปกรณ์ IoT กับแพลตฟอร์มมากขึ้น มีขั้นตอนการทำงานเริ่มต้นจากการที่ผู้ใช้กรอกข้อมูลคำสั่งลงในหน้าต่างบนหน้าเว็บแอปพลิเคชัน เมื่อผู้ใช้พิมพ์คำสั่งตามต้องการ ระบบจะรับค่าเหล่านั้นและจัดเก็บไว้ในตัวแปรที่เตรียมไว้สำหรับจัดการคำสั่ง หลังจากนั้น เมื่อผู้ใช้กดปุ่มบันทึก ระบบจะทำการรวบรวมข้อมูลคำสั่งที่กรอกเข้ามาพร้อมกับข้อมูลอื่น ๆ ที่เกี่ยวข้อง เช่น ประเภทของปุ่มหรือหมวดหมู่ จากนั้นข้อมูลทั้งหมดนี้จะถูกส่งไปยังเซิร์ฟเวอร์ผ่าน API เพื่อบันทึกลงในฐานข้อมูล หลังจากที่ข้อมูลถูกบันทึกลงในฐานข้อมูลเรียบร้อยแล้ว เมื่อมีการเรียกดูข้อมูลในภายหลัง ระบบจะดึงข้อมูลคำสั่งที่ถูกบันทึกไว้มาแสดงผลให้กับผู้ใช้ โดยที่ปุ่มที่ปรับแต่งเองไว้จะแสดงออกมาตามข้อมูลที่ได้จากฐานข้อมูล เมื่อผู้ใช้ต้องการส่งคำสั่งไปยัง ESP32 ระบบจะส่งคำสั่งไปยัง Cloud Broker ซึ่งทำหน้าที่เป็นตัวกลางในการส่งคำสั่งไปยัง ESP32 เมื่อได้รับคำสั่งแล้ว ESP32 จะประมวลผลตามฟังก์ชันที่กำหนดไว้ มีกระบวนการทำงานดังรูปที่ 3.97



รูปที่ 3.97 กระบวนการทำงานของคำสั่งที่ผู้ใช้กำหนดเอง

ในการดำเนินงานสำหรับคำสั่งที่ผู้ใช้สามารถกำหนดเองได้ ผู้ใช้จะสามารถพิมพ์คำสั่งใน input field ที่อยู่ในส่วนของ input หากผู้ใช้ใส่ข้อความหรือคำสั่งที่ต้องการ คำสั่งเหล่านี้จะถูกเก็บไว้ในตัวแปร command ซึ่งจะถูกส่งไปยัง MQTT เมื่อผู้ใช้คลิกปุ่ม Save หรือทำการส่งคำสั่งออกไป ในโค้ดนี้จะเห็นว่าเมื่อผู้ใช้กรอกคำสั่งและกดปุ่มบันทึก ฟังก์ชัน handleSave จะถูกเรียกใช้ เพื่อบันทึกค่าของคำสั่งที่กรอก ดังรูปที่ 3.98

```
<label className="text-white">Command: </label>
<input
  type="text"
  className="py-1 lg:w-[120px] bg-black text-green-400 w-[120px] rounded-sm px-4"
  placeholder="/"
  value={command}
  onChange={(e) => setButtonCommand(e.target.value)}
/>
```

รูปที่ 3.98 คำสั่งโปรแกรมสร้าง Input field รับค่าคำสั่ง

เมื่อกดปุ่มบันทึก ฟังก์ชัน handleSave จะถูกเรียกใช้ เพื่อบันทึกค่าของคำสั่งที่กรอกมาและบันทึกลงในฐานข้อมูล โดยสามารถใช้การเชื่อมต่อ API ส่งข้อมูลไปยังฐานข้อมูล ดังรูปที่ 3.99

```
const handleSave = async () => {
  const newButton: ButtonModel = {
    id,
    type,
    category,
    label,
    command,
    deviceId,
  };

  const resAddButton = await fetch("/api/button", {
    method: "POST",
    headers: {
      "Content-type": "application/json",
    },
    body: JSON.stringify({
      id,
      type,
      category,
      label,
      command,
      deviceId,
    }),
  });
```

รูปที่ 3.99 ฟังก์ชันบันทึกปุ่ม

เมื่อมีการคลิกปุ่มที่ได้ปรับแต่งปุ่มไว้ ฟังก์ชัน Control จะถูกเรียกใช้ ซึ่งในฟังก์ชันนี้ จะทำการตรวจสอบว่า client และ isConnected เชื่อมต่ออยู่หรือไม่ หากเชื่อมต่อแล้วมันจะส่งคำสั่งผ่าน MQTT ไปยัง Cloud Broker ด้วยคำสั่ง client.publish ซึ่ง cmd คือค่าของคำสั่งที่ผู้ใช้เลือกหรือคำสั่งที่ผู้ใช้กำหนดเอง ดังรูปที่ 3.100

```
const Control = async (cmd :string) => {
  if (client && isConnected) {
    client.publish(topic, `ctrl/${cmd}`);
    console.log(`Press : ${cmd.toUpperCase()}`);
    onLogReturn(`Press : ${cmd.toUpperCase()}`);
  }
};
```

รูปที่ 3.100 ฟังก์ชันโปรแกรมส่งค่าคำสั่งขึ้น Broker

ที่ฝั่งจัดการเว็บแอปพลิเคชัน ข้อมูลที่ได้รับจะถูกจัดเก็บในฐานข้อมูล การทำงานนี้อาจจะเป็นการสร้างบันทึกใหม่ในตารางหรือคอลเล็กชันที่เกี่ยวข้องกับคำสั่งที่ผู้ใช้ได้ตั้งค่าไว้ และจะเรียกใช้เมื่อผู้ใช้งานเข้าสู่หน้าควบคุมอุปกรณ์ ดังรูปที่ 3.101 จะแสดงปุ่มที่และคำสั่งที่ผู้ใช้งานได้ปรับแต่งไว้ ดังรูปที่ 3.102

```
const fetchButton = async (id: string) => {
  const response = await fetch(`/api/button/${id}`);
  return response.json();
};
```

รูปที่ 3.101 คำสั่งโปรแกรมดึงข้อมูลปุ่มจาก API โดยใช้ ID ของปุ่มเป็นตัวระบุ

```
{item.type == "transmitter" ? (
  <div className="animate-fastFade grid place-items-center w-full">
    {item.category == "press" ? (
      <PressButton
        category={item.category}
        cmd={item.command}
        label={item.label}
        type={item.type}
        isConnected={isConnected}
        client={client}
        topic={topic}
        onLogReturn={getLogReturned}
      />
```

รูปที่ 3.102 คำสั่งโปรแกรมแสดงข้อมูลปุ่ม

3.2.6 การรับค่าการทำงานของอุปกรณ์ IoT

การออกแบบในการรับค่าการทำงานของอุปกรณ์ IoT ทำการออกแบบเพื่อให้แสดงค่าผ่านหน้าปัดหรือแผนภูมิให้แก่ผู้ใช้งาน ตามจำนวนค่าการทำงานของอุปกรณ์ โดยมีกระบวนการทำงานในการนำค่าที่ได้การทำงานของอุปกรณ์ IoT ไปแสดงผลบนหน้าเว็บแอปพลิเคชัน ดังรูปที่ 3.103

รูปที่ 3.103 กระบวนการทำงานของการรับค่าการทำงานของอุปกรณ์ IoT

มีการดำเนินการในการเขียนคำสั่งโปรแกรมจากกระบวนการทำงาน โดยเริ่มจากการเชื่อมต่อ MQTT Broker ซึ่งทำหน้าที่เป็นตัวกลางในการรับส่งข้อมูลระหว่างอุปกรณ์ IoT และเว็บแอปพลิเคชัน โดยใช้ไลบรารี mqtt ของ JavaScript การเชื่อมต่อระบุ host port username และ password เพื่อยืนยันตัวตน ดังรูปที่ 3.104

```
const client = mqtt.connect(
  "wss://4cff082ff4a746da91e5ff64e35e8674.s1.eu.hivemq.cloud:8884/mqtt",
  {
    username: "admin",
    password: "Bam1234!",
    protocol: "wss",
  }
);
```

รูปที่ 3.104 คำสั่งโปรแกรมเชื่อมต่อ MQTT Broker

จากนั้นทำการ Subscribe Topic หลังจากเชื่อมต่อกับ Broker สำเร็จ ดังรูปที่ 3.105 ใช้คำสั่งโปรแกรม Subscribe topic ที่อุปกรณ์ IoT ส่งข้อมูลเข้ามา Topic โดยค่าของ topic จะถูกดึงมาจากข้อมูลของอุปกรณ์ที่ได้มาจาก API การ Subscribe นี้อยู่ใน useEffect hook ที่สอง ซึ่งจะทำงานเมื่อค่า topic มีการเปลี่ยนแปลง

```
console.log("is connecting...");
client.on("connect", () => {
  if (topic != null) {
    client.subscribe(topic, (err) => {
      if (!err) {
        console.log("Subscribed to Connected Message");
      }
    });
  } else {
    console.log("none topic");
  }
});
```

รูปที่ 3.105 คำสั่งโปรแกรม Subscribe Topic

เมื่อมีข้อมูลถูกส่งเข้ามาใน topic ที่ Subscribe ไว้ Event message จะถูกเรียกและจะทำการ parse message ที่ได้รับ โดยข้อมูลที่ส่งมาจากอุปกรณ์ IoT จะอยู่ในรูปแบบ string โดยใช้ regular expression เพื่อดึงค่า value1 ถึง value5 ออกมา ดังรูปที่ 3.106 ค่าเหล่านี้จะถูกเก็บไว้ใน state variables (value1, value2, ..., value5) และจะถูกนำไปแสดงผลบนหน้าเว็บแอปพลิเคชัน

```
client.on("message", (topic, message) => {
  console.log(`Received message on ${topic}: ${message}`);

  const msgStr = message.toString();

  // Use regex to extract values by matching "valueX:number"
  const regex = /value1:(\d+(\.\d+)?),\s*value2:(\d+(\.\d+)?),\s*value3:(\d+(\.
  \d+)?),\s*value4:(\d+(\.\d+)?),\s*value5:(\d+(\.\d+)?)/;
  const match = msgStr.match(regex);

  if (match) {
    setValue1(match[1] || null);
    setValue2(match[3] || null);
    setValue3(match[5] || null);
    setValue4(match[7] || null);
    setValue5(match[9] || null);
  }
});
```

รูปที่ 3.106 คำสั่งโปรแกรมแยกข้อความและจับคู่ค่าที่ตรงกัน

และเมื่อทำการแยกข้อความและจับคู่ค่าที่ตรงกัน จะนำค่า value เพื่อนำไปแสดงผลในผ่าน Device log และหน้าปัดในการแสดงผลต่อไป ดังรูปที่ 3.107

```
<div className={`mb-6 ${isLoading ? "animate-fadeIn " : "opacity-0"}`}>
  <div className="lg:text-xl text-white my-1">Device log</div>
  <div className=" grid lg:text-xl rounded-sm gap-2 text-black font-semibold px-3 py-1 ">
    <p  className="py-1 bg-white rounded-3xl px-4 text-blue-700 line-clamp-1 w-[180px] align-middle">LPG : {value1}</p>
    <p  className="py-1 bg-white rounded-3xl px-4 text-blue-700 line-clamp-1 w-[180px] align-middle">CO : {value2}</p>
    <p  className="py-1 bg-white rounded-3xl px-4 text-blue-700 line-clamp-1 w-[180px] align-middle">Smoke : {value3}</p>
    <p  className="py-1 bg-white rounded-3xl px-4 text-blue-700 line-clamp-1 w-[180px] align-middle">Temperature : {value4}</p>
    <p  className="py-1 bg-white rounded-3xl px-4 text-blue-700 line-clamp-1 w-[180px] align-middle">Humidity : {value5}</p>
  </div>
</div>
```

รูปที่ 3.107 คำสั่งโปรแกรมแสดงผลในผ่าน Device log

3.2.7 ระบบฐานข้อมูลเพื่อเก็บข้อมูลของอุปกรณ์ IoT และผู้ใช้งาน

ในการออกแบบฐานข้อมูลสำหรับแอปพลิเคชันนี้ มีการใช้ MongoDB เป็นฐานข้อมูลในการจัดเก็บข้อมูลสำคัญต่าง ๆ ทั้งข้อมูลของผู้ใช้ อุปกรณ์ ข้อความ การเชื่อมต่อ Wi-Fi การส่งและรับข้อมูลจากไมโครคอนโทรลเลอร์ การสื่อสารกันเองภายในเว็บแอปพลิเคชัน หน้าปัดการแสดงผล การปรับแต่งปุ่ม รวมถึงการนำเข้าอุปกรณ์จากภายนอก โดยข้อมูลถูกจัดเก็บในลักษณะของเอกสารที่มีความยืดหยุ่นและสามารถเชื่อมโยงกันได้ โดยมีรายละเอียด ดังนี้

3.2.7.1 ตารางข้อมูล User

ตารางผู้ใช้งาน เก็บข้อมูลสำคัญของผู้ใช้งาน เช่น ชื่อผู้ใช้ อีเมล และรหัสผ่าน ซึ่งจำเป็นต่อการยืนยันตัวตนและการเข้าถึงระบบภายในเว็บแอปพลิเคชัน รายละเอียดดังตารางที่ 3.1

ตารางที่ 3.1 ตารางข้อมูล User

ประเภท Key	ประเภทข้อมูล	รายละเอียด
id	UUID	คีย์หลักที่ใช้ในการระบุผู้ใช้แต่ละรายอย่างชัดเจนไม่ซ้ำกัน
username	String	ชื่อผู้ใช้งาน
email	String	อีเมลผู้ใช้งาน
password	String	รหัสผ่าน

3.2.7.2 ตารางข้อมูล Product

ตารางอุปกรณ์ตัวอย่าง เก็บรวบรวมรายละเอียดของสินค้าแต่ละชนิด เช่น รหัสอุปกรณ์ ประเภทสินค้า topic ที่ใช้ในการสื่อสาร และสถานะความเป็นเจ้าของ ซึ่งเป็นข้อมูลสำคัญในการจัดการและติดตามอุปกรณ์ตัวอย่าง รายละเอียดดังตารางที่ 3.2

ตารางที่ 3.2 ตารางข้อมูล Product

ประเภท Key	ประเภทข้อมูล	รายละเอียด
id	UUID	คีย์หลักที่ใช้ระบุอุปกรณ์ตัวอย่างแต่ละรายการอย่างชัดเจนไม่ซ้ำกัน
productId	String	รหัสสินค้า
type	String	ประเภทของสินค้า
topic	String	Topic ของอุปกรณ์ที่ใช้ในการสื่อสาร MQTT
password	String	รหัสผ่านของสินค้า
ownerStatus	Boolean	สถานะความเป็นเจ้าของ

3.2.7.3 ตารางข้อมูล Device

ตารางอุปกรณ์ จัดเก็บข้อมูลของอุปกรณ์ IoT ที่เชื่อมต่อกับระบบ เช่น รหัสอุปกรณ์ ชื่อ ประเภท สถานะ และข้อมูลการเชื่อมต่อ Wi-Fi รวมถึง ID ของผู้ใช้งานที่ครอบครองอุปกรณ์ และ ID สินค้าที่อุปกรณ์นั้น ๆ สังกัด รายละเอียดดังตารางที่ 3.3

ตารางที่ 3.3 ตารางข้อมูล Device

Field	ประเภทข้อมูล	รายละเอียด
id	UUID	คีย์หลักที่ใช้ในการระบุอุปกรณ์แต่ละตัวภายในตารางอย่างชัดเจน ไม่มีซ้ำกัน
deviceId	UUID	รหัสอุปกรณ์
userId	String	ID ของผู้ใช้งานที่เป็นเจ้าของอุปกรณ์
name	String	ชื่ออุปกรณ์
topic	String	Topic ของอุปกรณ์ที่ใช้ในการสื่อสาร MQTT
type	String	ประเภทของอุปกรณ์
password	String	รหัสผ่านของอุปกรณ์
status	String	สถานะของอุปกรณ์
wifiId	String	ID ของ Wi-Fi ที่อุปกรณ์เชื่อมต่อ
wifiConnect	String	สถานะการเชื่อมต่อ Wi-Fi ของอุปกรณ์
productId	String	ID ของสินค้าที่อุปกรณ์นี้

3.2.7.4 ตารางข้อมูล ExternalDevice

ตารางอุปกรณ์ภายนอก มีโครงสร้างคล้ายกับตารางอุปกรณ์ แต่เน้นข้อมูลของอุปกรณ์จากภายนอกระบบ โดยเพิ่ม broker username password และ path ในการเชื่อมต่อเข้ามา รายละเอียดดังตารางที่ 3.4

ตารางที่ 3.4 ตารางข้อมูล ExternalDevice

ประเภท Key	ประเภทข้อมูล	รายละเอียด
id	UUID	คีย์หลักที่ใช้ในการระบุอุปกรณ์ภายนอกแต่ละตัวภายในตารางอย่างชัดเจน ไม่มีซ้ำกัน
deviceId	String	รหัสอุปกรณ์
userId	String	ID ของผู้ใช้งานที่เป็นเจ้าของอุปกรณ์
name	String	ชื่ออุปกรณ์
topic	String	Topic ของอุปกรณ์ที่ใช้ในการสื่อสาร MQTT
broker	String	Broker ที่ใช้ในการสื่อสาร MQTT
connectPath	String	Path สำหรับการเชื่อมต่อ
username	String	Username สำหรับการเชื่อมต่อ MQTT
password	String	Password สำหรับการเชื่อมต่อ MQTT
status	String	สถานะของอุปกรณ์
wifiId	String	ID ของ Wi-Fi ที่อุปกรณ์เชื่อมต่อ
wifiConnect	String	สถานะการเชื่อมต่อ Wi-Fi ของอุปกรณ์

3.2.7.5 ตารางข้อมูล Wi-Fi

ตาราง Wi-Fi เก็บรวบรวมข้อมูลของเครือข่าย Wi-Fi ที่อุปกรณ์ต่างๆ ใช้ในการเชื่อมต่อ เช่น ชื่อ รหัสผ่าน และสถานะ เพื่อให้ระบบสามารถจัดการการเชื่อมต่อของอุปกรณ์ได้อย่างถูกต้อง รายละเอียดดังตารางที่ 3.5

ตารางที่ 3.5 ตารางข้อมูล Wi-Fi

ประเภท Key	ประเภทข้อมูล	รายละเอียด
id	UUID	คีย์หลักที่ใช้ระบุเครือข่าย Wi-Fi ไม่ซ้ำกัน
wifiId	UUID	รหัส Wi-Fi
wifiName	String	ชื่อ Wi-Fi
wifiPassword	String	รหัสผ่านของ Wi-Fi สำหรับการเชื่อมต่อ
status	String	สถานะการทำงานของ Wi-Fi

3.2.7.6 ตารางข้อมูล Button

ตารางปุ่มควบคุมใช้เก็บข้อมูลการตั้งค่าของปุ่มควบคุมที่ใช้ในการสั่งการอุปกรณ์ เช่น ประเภท หมวดหมู่ รูปไอคอนของปุ่ม และคำสั่ง โดยเชื่อมโยงกับตารางอุปกรณ์เพื่อระบุว่าปุ่มควบคุมนี้ใช้สั่งการอุปกรณ์ใด รายละเอียดดังตารางที่ 3.6

ตารางที่ 3.6 ตารางข้อมูล Button

ประเภท Key	ประเภทข้อมูล	รายละเอียด
id	UUID	คีย์หลักที่ใช้ในการระบุปุ่มแต่ละตัวภายในตารางอย่างชัดเจน ไม่มีซ้ำกัน
deviceId	String	ID ของอุปกรณ์ที่เชื่อมโยงกับปุ่ม
type	String	หมวดหมู่ของปุ่ม
label	String	รูปไอคอนของปุ่ม
command	String	คำสั่งที่ส่งเมื่อกดปุ่ม

3.2.7.7 ตารางข้อมูล Chart

ตารางแผนภูมิใช้สำหรับเก็บข้อมูลการตั้งค่าของแผนภูมิที่ใช้ในการแสดงผลข้อมูลจากอุปกรณ์ เช่น ประเภท ชื่อของแผนภูมิ สี และหน่วยของข้อมูล โดยเชื่อมโยงกับตารางอุปกรณ์ เพื่อระบุว่าแผนภูมินี้ใช้แสดงข้อมูลจากอุปกรณ์ใด รายละเอียดดังตารางที่ 3.7

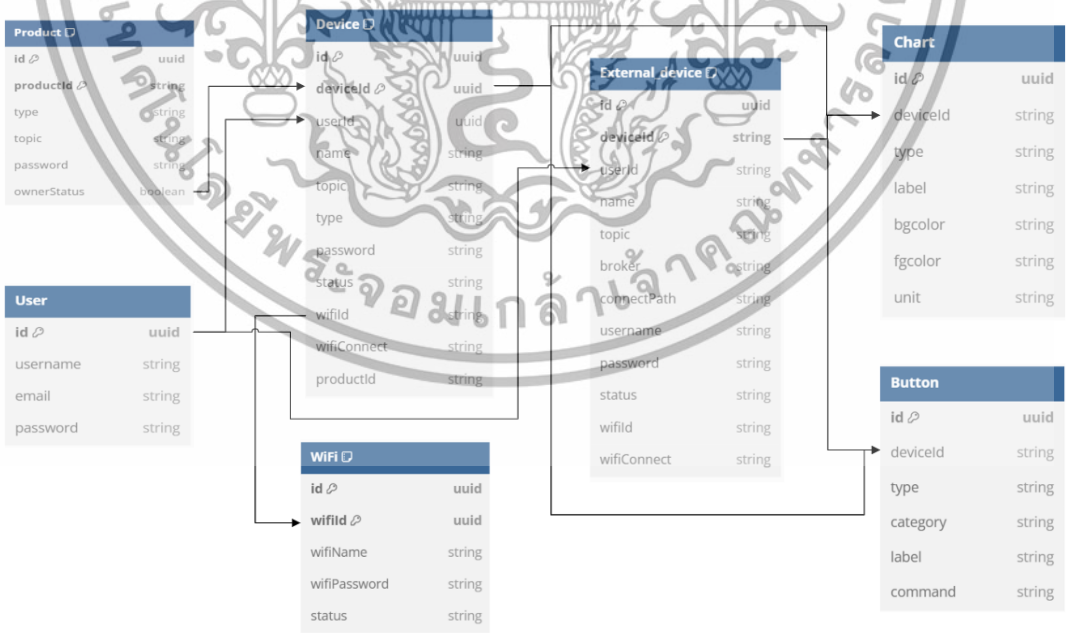
ตารางที่ 3.7 ตารางข้อมูล Button

ประเภท Key	ประเภทข้อมูล	รายละเอียด
id	UUID	คีย์หลักที่ใช้ในการระบุแผนภูมิภายในตารางอย่างชัดเจน ไม่มีซ้ำกัน
deviceId	String	ID ของอุปกรณ์ที่เชื่อมโยงกับปุ่ม
type	String	หมวดหมู่ของปุ่ม
label	String	ชื่อของแผนภูมิ
bgcolor	String	สีพื้นหลังของแผนภูมิ
fgcolor	String	สีตัวหนังสือของแผนภูมิ
unit	String	หน่วยของข้อมูลในแผนภูมิ

3.2.7.8 ความสัมพันธ์ระหว่างข้อมูล

การออกแบบตารางข้อมูลโดยให้มีความสัมพันธ์ระหว่างข้อมูลทั้ง 7 ตาราง จากตารางที่ 3.1 3.2 3.3 3.4 3.5 3.6 และ 3.7 ตามลำดับ โดยที่ทุกตารางในระบบนี้ถูกออกแบบมาให้ทำงานร่วมกันได้ เริ่มจากตาราง User เก็บข้อมูลของผู้ใช้แต่ละราย โดยใช้ id เป็นคีย์หลักและมีข้อมูล username email และ password สำหรับการตรวจสอบตัวตนของผู้ใช้ ตารางนี้เชื่อมโยงกับตาราง Device และ External_device ผ่านฟิลด์ userId เพื่อกำหนดว่าอุปกรณ์ใดเป็นของผู้ใช้รายใด ตาราง Product ใช้เก็บข้อมูลเกี่ยวกับประเภทของอุปกรณ์ (type) หัวข้อ (topic) รหัสผ่าน (password) และสถานะของเจ้าของ (ownerStatus) โดย id เป็นคีย์หลัก และมี productId เป็นตัวเชื่อมโยงกับ Device เพื่อกำหนดว่าอุปกรณ์แต่ละชิ้นมีผลิตภัณฑ์ตัวอย่างอะไรบ้าง ในส่วนตาราง Device เป็นศูนย์กลางของอุปกรณ์ IoT ที่ผู้ใช้เป็นเจ้าของ โดยแต่ละอุปกรณ์มี id และ deviceId เป็น UUID สำหรับระบุเอกลักษณ์ พร้อมกับข้อมูล userId ที่เชื่อมไปยังตาราง User เพื่อบ่งบอก

เจ้าของอุปกรณ์ นอกจากนี้ Device ยังมีฟิลด์ wifiId ที่เชื่อมโยงกับ WiFi เพื่อติดตามว่ามีการเชื่อมต่อกับเครือข่ายใด และ productId เพื่ออ้างอิงถึงประเภทผลิตภัณฑ์ที่อุปกรณ์นั้นใช้ ในส่วนการเชื่อมต่อกับเครือข่าย Wi-Fi ผ่านตาราง WiFi ใช้จัดเก็บข้อมูลเครือข่าย Wi-Fi ที่อุปกรณ์สามารถเชื่อมต่อได้ โดยใช้ id และ wifiId เป็น UUID เก็บข้อมูล wifiName wifiPassword และ status ตารางนี้เชื่อมโยงกับ Device ผ่าน wifiId เพื่อให้แต่ละอุปกรณ์สามารถระบุได้ว่ากำลังเชื่อมต่อกับเครือข่ายใด เมื่อผู้ใช้งานได้มีการนำเข้าอุปกรณ์จากภายนอกเข้ามาใชเงานกับแพลตฟอร์ม ตาราง External_device เป็นอีกประเภทหนึ่งของอุปกรณ์ที่เชื่อมต่อกับระบบ โดยมี id และ deviceId เป็นตัวระบุ และ userId เพื่อเชื่อมโยงกับผู้ใช้ นอกจากนี้ ยังมีข้อมูล topic broker connectPath, username password และ status เพื่อกำหนดค่าการเชื่อมต่อไปยัง MQTT broker และยังเชื่อมโยงกับ WiFi ผ่าน wifiId เช่นเดียวกับ Device ส่วนการปรับแต่งเพิ่มเติมจะใช้ตาราง Chart เก็บข้อมูลการแสดงผลกราฟที่เกี่ยวข้องกับอุปกรณ์ โดยแต่ละ Chart มี id เป็น UUID และ deviceId เพื่อเชื่อมโยงกับอุปกรณ์ใดอุปกรณ์หนึ่ง พร้อมกับข้อมูล type label bgcolor fgcolor และ unit เพื่อกำหนดค่าการแสดงผล ในกรณีที่ต้องการสร้างปุ่มคำสั่งในการสั่งงานอุปกรณ์ ตาราง Button ใช้สำหรับกำหนดปุ่มที่ใช้ควบคุมอุปกรณ์ โดยแต่ละ Button มี id เป็น UUID และ deviceId เพื่อเชื่อมโยงไปยังอุปกรณ์ที่เกี่ยวข้อง พร้อมกับข้อมูล type category label และ command เพื่อกำหนดการทำงานของปุ่ม โดยมีการเชื่อมโยง ดังรูปที่ 3.108



รูปที่ 3.108 ความสัมพันธ์กันระหว่างข้อมูลภายในเว็บแอปพลิเคชัน

3.3 เครื่องมือที่ใช้ในการทดสอบ

ปริญญานิพนธ์เล่มนี้มีการใช้งานและทำงานผ่านหน้าเว็บแอปพลิเคชันเป็นส่วนใหญ่ โดยมีอุปกรณ์และเครื่องมือที่ใช้ในการทดสอบ ดังนี้

3.3.1 คอมพิวเตอร์วางตัก

คอมพิวเตอร์วางตักที่ใช้ในการทำปริญญานิพนธ์เล่มนี้ โดยใช้ในการเปิดโปรแกรม Visual Studio Code โปรแกรม Arduino IDE และใช้ในการเปิดเว็บเบราว์เซอร์เพื่อทดสอบจากการเขียนโปรแกรมสร้างเว็บแอปพลิเคชัน โดยใช้งานผ่านคอมพิวเตอร์วางตักจำนวน 3 เครื่อง ดังนี้

3.3.1.1 ACER NITRO V 15 ANV15-51-574G

คอมพิวเตอร์วางตัก Acer Nitro V 15 ANV15-51-574G [65] ดังรูปที่ 3.109 โดยมีคุณสมบัติเฉพาะ ดังนี้

1) CPU : Intel Core i5-13420H (2.10 GHz 18 MB L3 Cache up to 4.60 GHz) 4 (P-Core) / 4 (E-Core)

2) GPU : NVIDIA GeForce RTX 4050 (6GB GDDR6) 75 Watt TGP

3) RAM : 16 GB DDR5 4800 MHz

4) STORAGE : 512 GB SSD PCIe M.2 Gen 4

5) DISPLAY : 15.6 inch (1920x1080) Full HD

รูปที่ 3.109 ACER NITRO V 15 ANV15-51-574G [65]

3.3.1.2 LENOVO IDEAPAD 330S-14IKB-81F401MKTA

คอมพิวเตอร์วางตัก LENOVO IDEAPAD 330S-14IKB [66] ดังรูปที่ 3.110 โดยมีคุณสมบัติเฉพาะ ดังนี้

1) CPU : Intel Core i5-8250U

2) GPU : AMD Radeon 535 2 GB GDDR5

3) RAM : 8 GB DDR4-2400 MHz

4) STORAGE : 256 GB PCIe/NVMe M.2 SSD

5) DISPLAY : 14.0 inch (1920x1080) Full HD IPS

รูปที่ 3.110 LENOVO IDEAPAD 330S-14IKB-81F401MKTA [66]

3.3.1.3 ASUS TUF GAMING F15 FX506LH-HN002T

คอมพิวเตอร์วางตัก ASUS TUF GAMING F15 FX506LH-HN002T [67] ดังรูปที่ 3.111 โดยมีคุณสมบัติเฉพาะ ดังนี้

1) CPU : Intel Core i5-10300H (2.50 GHz 8 MB L3 Cache up to 4.50 Ghz) 4 (P-Core)

2) GPU : NVIDIA GeForce GTX 1650 (4GB GDDR6)

3) RAM : 8 GB DDR4 2933Mhz

4) STORAGE : 512 GB SSD PCIe M.2

5) DISPLAY : 15.6 inch (1920x1080) Full HD

รูปที่ 3.111 ASUS TUF GAMING F15 FX506LH-HN002T [67]

3.3.2 Postman

Postman [68] เป็นแพลตฟอร์มที่ใช้ในการพัฒนาและทดสอบ API (Application Programming Interface) ซึ่งช่วยให้นักพัฒนาสามารถทดสอบ API ได้อย่างสะดวกผ่านการสร้างและส่งคำขอ (requests) ไปยัง API เพื่อดูการตอบกลับ (responses) และวิเคราะห์ผลลัพธ์จาก API ที่ได้ออกแบบและสร้างขึ้น

3.3.3 HiveMQ Cloud Web Client

HiveMQ Cloud Web Client [69] ใช้สำหรับเชื่อมต่อกับ HiveMQ Cloud Cluster เพื่อดูผลการทดสอบและทำการแก้ไขหรือทดสอบกรณีการใช้งาน MQTT ในเวลาจริง เพื่อดูการส่ง (publish) และสมัครรับ (subscribe) ข้อความ รวมถึงรับข้อความที่ส่งผ่านหัวข้อต่าง ๆ ภายใน Cluster

3.3.4 Adafurit

Adafurit ใช้สำหรับเป็น Broker ในการเชื่อมต่อ Cloud Feeds เพื่อดูผลการทดสอบและทำการแก้ไขหรือทดสอบกรณีการใช้งาน MQTT ในเวลาจริง เพื่อดูการส่ง (publish) และสมัครรับ (subscribe) ข้อความ รวมถึงรับข้อความที่ส่งผ่านหัวข้อต่าง ๆ ภายใน Feeds

3.3.5 Google Chrome

Google Chrome [70] คือเว็บเบราว์เซอร์ฟรีที่ใช้สำหรับการเข้าถึงอินเทอร์เน็ตและการใช้งานแอปพลิเคชัน โดยมีพื้นฐานมาจากโครงการเว็บเบราว์เซอร์ Open Source Chromium ใช้เพื่อทดสอบการทำงานและการแสดงผลของเว็บแอปพลิเคชัน

3.3.6 Arduino IDE

Arduino IDE (Integrated Development Environment) เป็นซอฟต์แวร์ Open source ที่ใช้สำหรับการพัฒนาโปรแกรมและอัปโหลดโปรแกรมลงในบอร์ดไมโครคอนโทรลเลอร์ทำให้ผู้ใช้สามารถเขียนโค้ดในภาษา C และ C++ รวมถึงการควบคุมการทำงานของฮาร์ดแวร์ผ่านไมโครคอนโทรลเลอร์ได้ง่าย ๆ ในการทดสอบจะใช้ Serial Monitor ของ Arduino IDE เพื่อดูผลการทำงานและการสั่งการของบอร์ดไมโครคอนโทรลเลอร์ ESP32

3.4 การจัดเก็บผลการทดลอง

3.4.1 การทดสอบการทำงานอุปกรณ์

ทดสอบการทำงานของอุปกรณ์เมื่อมีการสั่งงานผ่านเว็บแอปพลิเคชัน โดยเก็บผลหารทำงานของอุปกรณ์ ได้แก่ รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ รวมถึงทดสอบการทำงานของอุปกรณ์ตัวอย่างใด ๆ ที่นำเข้ามาใช้งานกับแพลตฟอร์ม

3.4.2 การทดสอบการเชื่อมต่อกับ Broker

ทดสอบโดยการเชื่อมต่อเว็บแอปพลิเคชันเข้ากับ HiveMQ Cloud Web Client และ Adafruit Broker รวมถึงการเชื่อมต่อบอร์ดไมโครคอนโทรลเลอร์ ESP32 เข้ากับ HiveMQ Cloud Web Client และ Adafruit Broker เพื่อดูการส่งและรับข้อมูลผ่าน Topic

3.4.3 การทำงานของเว็บแอปพลิเคชัน

ทดสอบโดยแบ่งการทำงานของเว็บแอปพลิเคชันเป็น 2 ส่วน คือ การทดสอบการทำงานหน้าเว็บแอปพลิเคชัน (Front-End) และทดสอบระบบจัดการเว็บไซต์ (Back-End)

3.4.4 การทดสอบการนำเข้าอุปกรณ์

ทดสอบโดยการจะแบบการทดสอบเป็น 2 ส่วน ได้แก่ การนำเข้าอุปกรณ์ตัวอย่าง และการนำเข้าอุปกรณ์ใด ๆ จากภายนอก โดยสร้างอุปกรณ์ที่ใช้โครงสร้างคำสั่งและโปรโตคอลในการสื่อสารตามที่แพลตฟอร์มได้กำหนด จากนั้นจะนำเข้าอุปกรณ์เข้ามาภายในระบบและทดสอบการทำงานของอุปกรณ์นั้น

3.4.5 การทดสอบการเปลี่ยนการเชื่อมต่อ Wi-Fi

ทดสอบโดยการเปลี่ยน SSID และ รหัสผ่านของ Wi-Fi ผ่านหน้าเว็บแอปพลิเคชันโดยสามารถเปลี่ยนได้ตามความต้องการของผู้ใช้ และยังสามารถล้างค่า Wi-Fi ในบอร์ด ESP32 ให้กลับไปเป็นค่าเริ่มต้น

3.4.6 การทำงานปรับแต่งปุ่มควบคุมและการแสดงข้อมูลอุปกรณ์ IoT

ทดสอบโดยการใช้รถยนต์บังคับ เริ่มจากการคาบคุมอุปกรณ์จากปุ่มเริ่มต้นที่มี และทำการปรับแต่งโดยใช้คำสั่งที่มีให้ภายในเว็บแอปพลิเคชัน จากนั้นให้ทำการปรับแต่งปุ่มแลคำสั่งในการควบคุมรถยนต์บังคับ ในส่วนการปรับแต่งหน้าจอแสดงผลจะใช้อุปกรณ์รดน้ำต้นไม้ โดยจะให้ปรับแต่งแผนภูมิแสดงและหน้าปัดในการแสดงข้อมูลของความชื้นในดิน

บทที่ 4
ผลการทดลอง

4.1 ผลการทำงานของอุปกรณ์

จากการออกแบบอุปกรณ์ IoT เพื่อแสดงการสั่งการและควบคุมมาจากหน้าเว็บแอปพลิเคชัน โดยทำการออกแบบและจัดทำอุปกรณ์ตามการออกแบบทั้งหมด 3 อุปกรณ์ ได้แก่ รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ อุปกรณ์ตรวจจับควันไฟ และอุปกรณ์ใด ๆ ซึ่งเป็นอุปกรณ์ตัวอย่างในการนำเข้าอุปกรณ์จากภายนอกมาใช้งานกับแพลตฟอร์ม มีผลดำเนินงานและผลการทำงานของอุปกรณ์ ดังนี้

4.1.1 รถยนต์บังคับ

รถยนต์บังคับมีการออกแบบอุปกรณ์ภายในโดยการใช้อุปกรณ์ประกอบไปด้วยส่วนควบคุม คือ DC Gear Motor โมดูลขับมอเตอร์ L298N และบอร์ดไมโครคอนโทรลเลอร์ ESP32 และมีส่วนประกอบที่ทำให้รถยนต์บังคับนั้นสมบูรณ์ คือ ล้อรถ ฐานรถอะคริลิค และโครงรถที่ใช้เทคโนโลยีการพิมพ์ 3 มิติ โดยเมื่อทำการประกอบและจัดทำรถยนต์บังคับออกมาจะได้ ดังรูปที่ 4.1

รูปที่ 4.1 รถยนต์บังคับ

4.1.1.1 ผลการสั่งการรถยนต์บังคับ

เมื่อทำการสั่งการและควบคุมรถยนต์บังคับจากหน้าเว็บแอปพลิเคชัน โดยส่งคำสั่งการสั่งการผ่าน Broker และบอร์ดไมโครคอนโทรลเลอร์ ESP32 จะทำการรับสั่งมาเพื่อสั่งงานอุปกรณ์ตามโปรแกรมที่ได้กำหนดขาควบคุมให้มอเตอร์มีการทำงาน ดังรูปที่ 4.2

และเมื่อมีคำสั่งเข้ามาจากเว็บแอปพลิเคชันผ่าน Broker และทำการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามา ดังรูปที่ 4.3

```
const int EN1 = 12;
const int EN2 = 14;


const int INP1 = 27;
const int INP2 = 26;
const int INP3 = 25;
const int INP4 = 33;
```

รูปที่ 4.2 คำสั่งโปรแกรมกำหนดขาการทำงาน

```
if (command == "ctrl/forward") {
    forward();
} else if (command == "ctrl/backward") {
    backward();
} else if (command == "ctrl/left") {
    left();
} else if (command == "ctrl/right") {
    right();
} else if (command == "ctrl/stop") {
    stop();
}
```

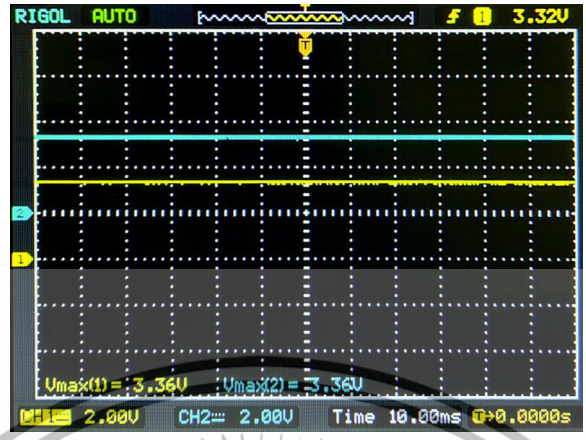
รูปที่ 4.3 คำสั่งโปแกรมตรวจสอบเงื่อนไขแต่ละคำสั่ง

1) เมื่อรับคำสั่ง forward

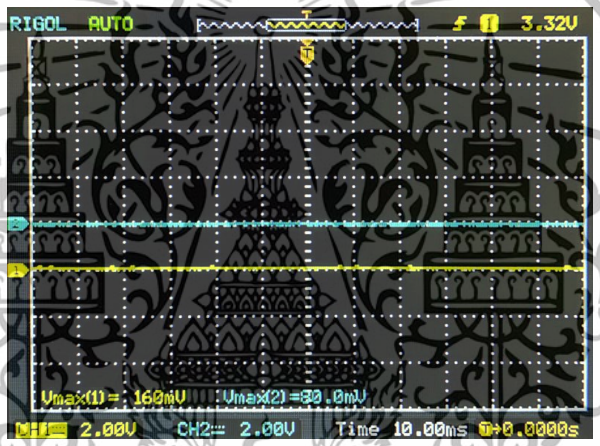
เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น “ctrl/forward” จะสั่งการทำงานตามคำสั่งโปรแกรม ดังรูปที่ 4.4 ฟังก์ชันนี้จะทำให้มอเตอร์หมุนไปในทิศทางที่ทำให้รถเคลื่อนที่ไปข้างหน้า ขา INP1 และ INP3 จะได้รับสัญญาณ HIGH มีผลการทำงานดังรูปที่ 4.5 ในขณะที่ INP2 และ INP4 จะได้รับสัญญาณ LOW มีผลการทำงานดังรูปที่ 4.6

```
void forward() {
    digitalWrite(INP1, HIGH);
    digitalWrite(INP2, LOW);
    digitalWrite(INP3, HIGH);
    digitalWrite(INP4, LOW);
    Serial.println("Moving forward");
}
```

รูปที่ 4.4 คำสั่งโปรแกรมฟังก์ชัน forward



รูปที่ 4.5 ผลการทำงานขา INP1 และ INP3 ได้รับสัญญาณ HIGH



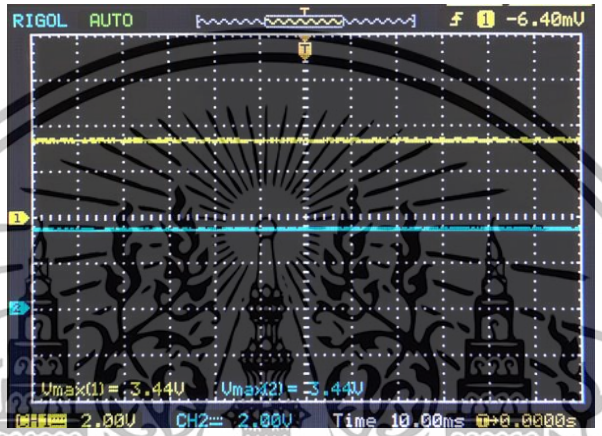
รูปที่ 4.6 ผลการทำงานขา INP2 และ INP4 ได้รับสัญญาณ LOW

2) เมื่อรับคำสั่ง backward

เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น “ctrl/ backward” ดังรูปที่ 4.7 จะสั่งการทำงานตามคำสั่งโปรแกรม โดยฟังก์ชันนี้ใช้สำหรับให้มอเตอร์เคลื่อนที่รถไปในทิศทางถอยหลัง จึงทำให้มอเตอร์หมุนในทิศทางตรงกันข้ามกับฟังก์ชัน forward() ซึ่งจะทำให้รถเคลื่อนที่ถอยหลัง โดยขา INP2 และ INP4 จะได้รับสัญญาณ HIGH มีผลการทำงานดังรูปที่ 4.8 ในขณะที่ INP1 และ INP3 จะได้รับสัญญาณ LOW มีผลการทำงานดังรูปที่ 4.9

```
void backward() {
    digitalWrite(INP1, LOW);
    digitalWrite(INP2, HIGH);
    digitalWrite(INP3, LOW);
    digitalWrite(INP4, HIGH);
    Serial.println("Moving backward");
}
```

รูปที่ 4.7 คำสั่งโปรแกรมฟังก์ชัน backward



รูปที่ 4.8 ผลการทำงานขา INP2 และ INP4 ได้รับสัญญาณ HIGH

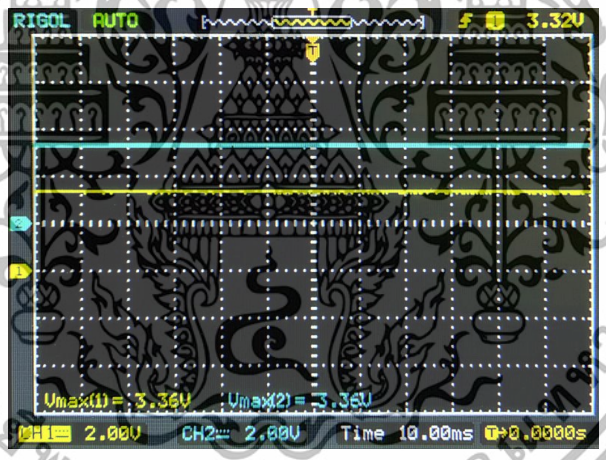
รูปที่ 4.9 ผลการทำงานขา INP1 และ INP3 ได้รับสัญญาณ LOW

3) เมื่อรับคำสั่ง left

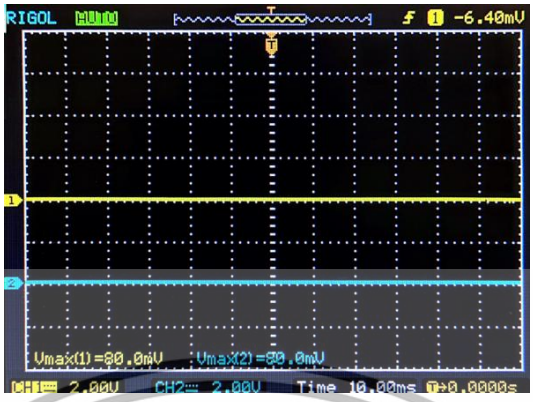
เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น “ctrl/left” จะสั่งการทำงานตามคำสั่งโปรแกรม ดังรูปที่ 4.10 ฟังก์ชันนี้จะทำให้มอเตอร์หมุนไปในทิศทางที่ทำให้รถเคลื่อนที่ไปทางซ้ายของรถ ขา INP1 และ INP4 จะได้รับสัญญาณ HIGH มีผลการทำงานดังรูปที่ 4.11 ในขณะที่ INP2 และ INP3 จะได้รับสัญญาณ LOW มีผลการทำงานดังรูปที่ 4.12

```
void left() {
    digitalWrite(INP1, LOW);
    digitalWrite(INP2, HIGH);
    digitalWrite(INP3, HIGH);
    digitalWrite(INP4, LOW);
    Serial.println("Turning left");
}
```

รูปที่ 4.10 คำสั่งโปรแกรมฟังก์ชัน left



รูปที่ 4.11 ผลการทำงานขา INP2 และ INP3 ได้รับสัญญาณ HIGH



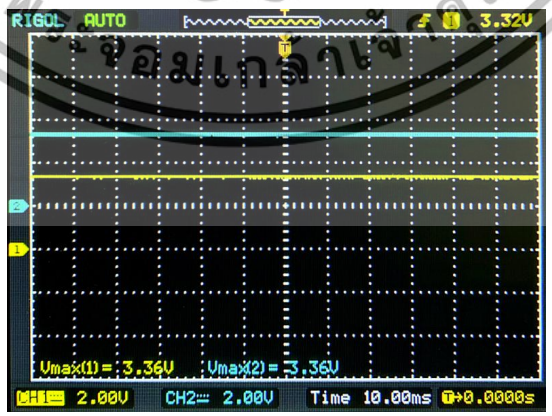
รูปที่ 4.12 ผลการทำงานขา INP1 และ INP4 ได้รับสัญญาณ LOW

4) เมื่อรับคำสั่ง right

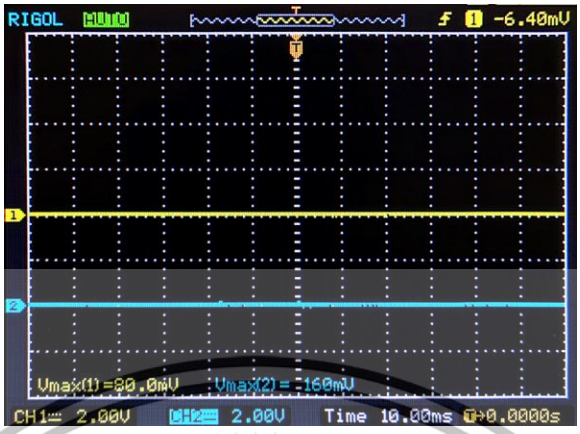
เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น “ctrl/right” จะสั่งการทำงานตามคำสั่งโปรแกรม ดังรูปที่ 4.13 โดยฟังก์ชันนี้ใช้สำหรับให้มอเตอร์เคลื่อนที่รถไปในทิศทางขวา จึงทำให้มอเตอร์หมุนในทิศทางตรงกันข้ามกับฟังก์ชัน left() ซึ่งจะทำให้รถเลี้ยวซ้าย โดยขา INP2 และ INP4 จะได้รับสัญญาณ HIGH มีผลการทำงานดังรูปที่ 4.14 ในขณะที่ INP1 และ INP3 จะได้รับสัญญาณ LOW มีผลการทำงานดังรูปที่ 4.15

```
void right() {
  digitalWrite(INP1, HIGH);
  digitalWrite(INP2, LOW);
  digitalWrite(INP3, LOW);
  digitalWrite(INP4, HIGH);
  Serial.println("Turning right");
}
```

รูปที่ 4.13 คำสั่งโปรแกรมฟังก์ชัน right



รูปที่ 4.14 ผลการทำงานขา INP1 และ INP4 ได้รับสัญญาณ HIGH



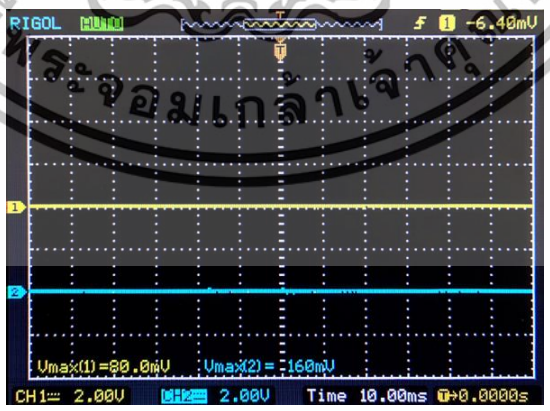
รูปที่ 4.15 ผลการทำงานขา INP2 และ INP3 ได้รับสัญญาณ LOW

5) เมื่อรับคำสั่ง stop

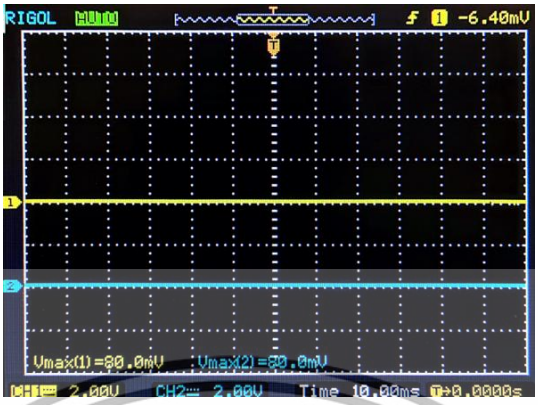
เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น “ctrl/stop” จะสั่งการทำงานตามคำสั่งโปรแกรม ดังรูปที่ 4.16 โดยฟังก์ชันนี้ใช้สำหรับให้มอเตอร์หยุดการเคลื่อนที่ของรถ ซึ่งจะทำให้รถหยุดนิ่ง โดยทำให้ขา INP1 INP2 ได้รับสัญญาณ LOW มีผลการทำงานดังรูปที่ 4.17 INP3 INP4 ได้รับสัญญาณ LOW มีผลการทำงานดังรูปที่ 4.18

```
void stop() {
    digitalWrite(INP1, LOW);
    digitalWrite(INP2, LOW);
    digitalWrite(INP3, LOW);
    digitalWrite(INP4, LOW);
    Serial.println("Stop");
}
```

รูปที่ 4.16 คำสั่งโปรแกรมฟังก์ชัน stop



รูปที่ 4.17 ผลการทำงานขา INP1 และ INP2 ได้รับสัญญาณ LOW



รูปที่ 4.18 ผลการทำงานขา INP3 และ INP4 ได้รับสัญญาณ LOW

4.1.2 แขนกล

แขนกลมีการออกแบบอุปกรณ์ภายในโดยการใช้อุปกรณ์ประกอบไปด้วยส่วนควบคุม คือ Servo motor MG90s และบอร์ดไมโครคอนโทรลเลอร์ ESP32 และมีส่วนประกอบที่ทำให้แขนกลนั้นสมบูรณ์ คือ โครงอะคริลิคประกอบเป็นแขนกล และกล่องใส่วงจรที่ใช้เทคโนโลยีการพิมพ์ 3 มิติ โดยเมื่อทำการประกอบและจัดทำออกมาจะได้ ดังรูปที่ 4.19

รูปที่ 4.19 แขนกล

4.1.2.1 ผลการสั่งการแขนกล

เมื่อทำการสั่งการและควบคุมแขนกลจากหน้าเว็บแอปพลิเคชัน โดยส่งคำสั่งการสั่งการผ่าน Broker และบอร์ดไมโครคอนโทรลเลอร์ ESP32 จะทำการรับสั่งมาเพื่อสั่งงานอุปกรณ์ตามโปรแกรมที่ได้กำหนดขาควบคุมให้มอเตอร์มีการทำงาน ดังรูปที่ 4.20

```
Serial.begin(115200);
myservo1.attach(18);  //M1 base
myservo2.attach(19);  //M2 forward backward
myservo3.attach(25);  //M3 Up-Down
myservo4.attach(26);  //M4 Hold
```

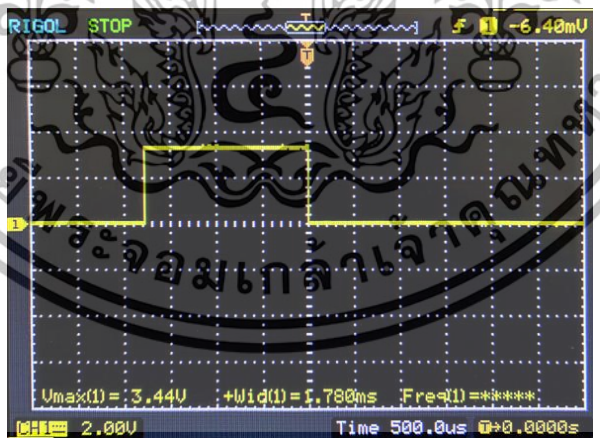
รูปที่ 4.20 คำสั่งโปรแกรมกำหนดขาการทำงาน

1) เมื่อรับคำสั่ง forward และ backward

จากการทดสอบส่งข้อความเพื่อควบคุมแขนกลโดยใช้คำสั่ง forward เมื่อมีการตรวจสอบเงื่อนไขแล้วว่าตรงกับเงื่อนไขที่ได้รับข้อความ forward ดังรูปที่ 4.21 มอเตอร์ที่ทำการควบคุมแขนกลให้ไปข้างหน้าจึงมีการขยับแขนส่วนนั้น ทำให้แขนกลมีการขยับไปในทิศทางข้างหน้า โดยเมื่อโปรแกรมได้รับคำสั่ง "ctrl/forward" มอเตอร์ M2 จะหมุนไปยังตำแหน่งมุม 120 องศา ซึ่งหมายความว่าแขนกลจะเคลื่อนที่ในทิศทางที่เป็นไปข้างหน้า และส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1780 ไมโครวินาที ดังรูปที่ 4.22

```
// M2
if (message == "ctrl/forward") {
  myservo2.write(120);
  delay(300);
  myservo2.writeMicroseconds(1500);
}
```

รูปที่ 4.21 คำสั่งโปรแกรม forward

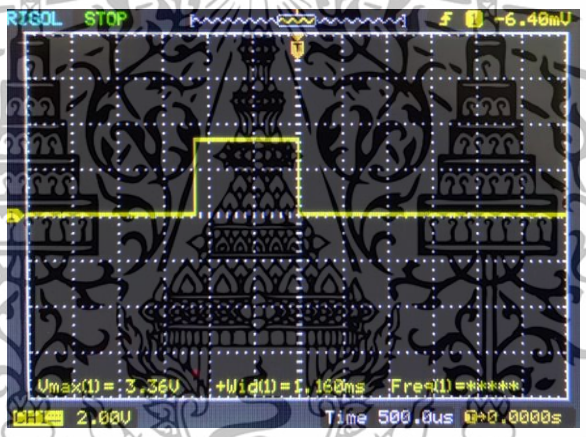


รูปที่ 4.22 ผลการทำงานมอเตอร์ M2 เมื่อได้รับข้อความ forward

เมื่อมีการตรวจสอบเงื่อนไขแล้วว่าตรงกับเงื่อนไขที่ได้รับข้อความ backward ดังรูปที่ 4.23 มอเตอร์ที่ทำการควบคุมแขนกลให้ไปข้างหลังจึงมีการขยับแขนส่วนนั้น ทำให้แขนกลมีการขยับไปในทิศทางกลับไปข้างหลัง โดยเมื่อโปรแกรมได้รับคำสั่ง "ctrl/ backward " มอเตอร์ M2 จะหมุนไปยังตำแหน่งมุม 120 องศา ซึ่งหมายความว่าแขนกลจะเคลื่อนที่ในทิศทางที่เป็นไปข้างหน้า และส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1160 ไมโครวินาที ดังรูปที่ 4.24

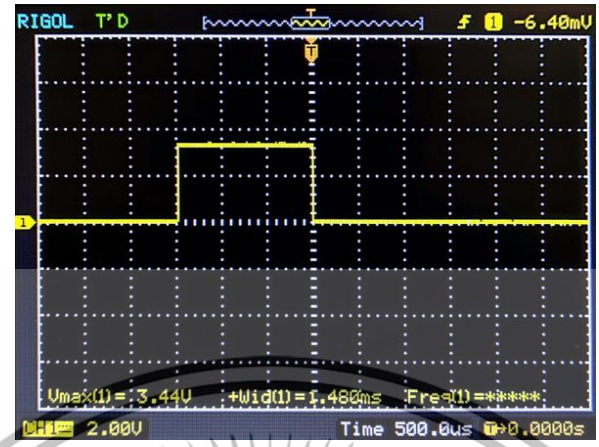
```
else if (message == "ctrl/backward") {
  myservo2.write(60);
  delay(300);
  myservo2.writeMicroseconds(1500);
}
```

รูปที่ 4.23 คำสั่งโปรแกรม backward



รูปที่ 4.24 ผลการทำงานมอเตอร์ M2 เมื่อได้รับข้อความ backward

โดยหลังมอเตอร์เคลื่อนที่ครบ 300 มิลลิวินาที มอเตอร์จะหยุดหมุนโดยตั้งค่าเป็นค่ากลาง โดยส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1500 ไมโครวินาที เมื่อทำการวัดสัญญาณดังกว่างจะได้ผลดังรูปที่ 4.25



รูปที่ 4.25 ผลการทำงานมอเตอร์ M2 เมื่อต้องหยุดหมุน

2) เมื่อรับคำสั่ง up และ down

จากการทดสอบส่งข้อความเพื่อควบคุมแขนกลโดยใช้คำสั่ง up เมื่อมีการตรวจสอบเงื่อนไขแล้วว่าตรงกับเงื่อนไขที่ได้รับข้อความ up ดังรูปที่ 4.26 มอเตอร์ที่ทำการควบคุมแขนกลให้ไปด้านบนจึงมีการขยับแขนส่วนนั้น ทำให้แขนกลมีการขยับไปในทิศทางด้านบน โดยเมื่อโปรแกรมได้รับคำสั่ง "ctrl/up" มอเตอร์ M3 จะหมุนตามเข็มนาฬิกา ซึ่งหมายความว่าแขนกลจะเคลื่อนที่ในทิศทางด้านบน และส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1780 ไมโครวินาที ดังรูปที่ 4.27

```
// M3
if (message == "ctrl/up") {
  myservo3.write(120);
  delay(300);
  myservo3.writeMicroseconds(1500);
```

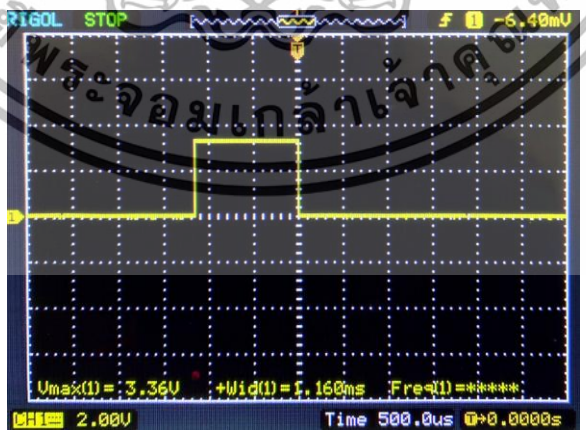
รูปที่ 4.26 คำสั่งโปรแกรม up

รูปที่ 4.27 ผลการทำงานมอเตอร์ M3 เมื่อได้รับข้อความ up

เมื่อมีการตรวจสอบเงื่อนไขแล้วว่าตรงกับเงื่อนไขที่ได้รับข้อความ down ดังรูปที่ 4.28 มอเตอร์ที่ทำการควบคุมแขนกลให้ไปด้านหลังจึงมีการขยับแขนส่วนนั้น ทำให้แขนกลมีการขยับไปในทิศทางกลับไปด้านหลัง โดยเมื่อโปรแกรมได้รับคำสั่ง "ctrl/ down" มอเตอร์ M3 จะหมุนทวนเข็มนาฬิกา ซึ่งหมายความว่าแขนกลจะเคลื่อนที่ในทิศทางด้านหลัง และส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1160 ไมโครวินาที ดังรูปที่ 4.29

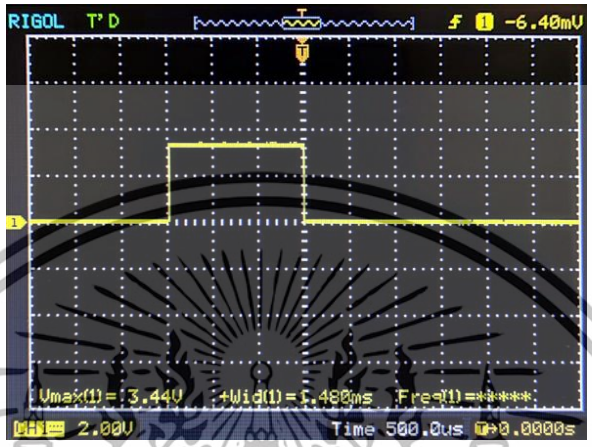
```
} else if (message == "ctrl/down") {
  myservo3.write(60);
  delay(300);
  myservo3.writeMicroseconds(1500);
}
```

รูปที่ 4.28 คำสั่งโปรแกรม down



รูปที่ 4.29 ผลการทำงานมอเตอร์ M3 เมื่อได้รับข้อความ down

โดยหลังมอเตอร์เคลื่อนที่ครบ 300 มิลลิวินาที มอเตอร์จะหยุดหมุนโดยตั้งค่าเป็นค่ากลาง โดยส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1500 ไมโครวินาที เมื่อทำการวัดสัญญาณดังกว่างจะได้ผลดังรูปที่ 4.30



รูปที่ 4.30 ผลการทำงานมอเตอร์ M3 เมื่อต้องหยุดหมุน

3) เมื่อรับคำสั่ง left และ right

จากการทดสอบส่งข้อความเพื่อควบคุมแขนกลโดยใช้คำสั่ง left เมื่อมีการตรวจสอบเงื่อนไขแล้วว่าตรงกับเงื่อนไขที่ได้รับข้อความ left ดังรูปที่ 4.31 มอเตอร์ที่ทำการควบคุมแขนกลให้ไปด้านบนจึงมีการขยับแขนส่วนนั้น ทำให้แขนกลมีการขยับไปในทิศทางด้านซ้ายโดยเมื่อโปรแกรมได้รับคำสั่ง "ctrl/left" มอเตอร์ M1 จะหมุนตามเข็มนาฬิกา ซึ่งหมายความว่าแขนกลจะเคลื่อนที่ในทิศทางด้านซ้าย และส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1780 ไมโครวินาที ดังรูปที่ 4.32

```
// M1
else if (message == "ctrl/left") {
  myservo1.write(120);
  delay(300);
  myservo1.writeMicroseconds(1500);
```

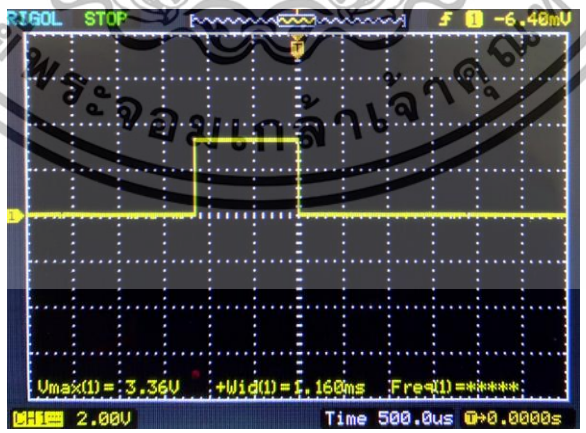
รูปที่ 4.31 คำสั่งโปรแกรม left

รูปที่ 4.32 ผลการทำงานมอเตอร์ M1 เมื่อได้รับข้อความ left

เมื่อมีการตรวจสอบเงื่อนไขแล้วว่าตรงกับเงื่อนไขที่ได้รับข้อความ right ดังรูปที่ 4.33 มอเตอร์ที่ทำการควบคุมแขนกลให้ไปด้านหลังจึงมีการขยับแขนส่วนนั้น ทำให้แขนกลมีการขยับไปในทิศทางกลับไปด้านหลัง โดยเมื่อโปรแกรมได้รับคำสั่ง "ctrl/right " มอเตอร์ M1 จะหมุนทวนเข็มนาฬิกา ซึ่งหมายความว่าแขนกลจะเคลื่อนที่ในทิศทางด้านหลัง และส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1160 ไมโครวินาที ดังรูปที่ 4.34

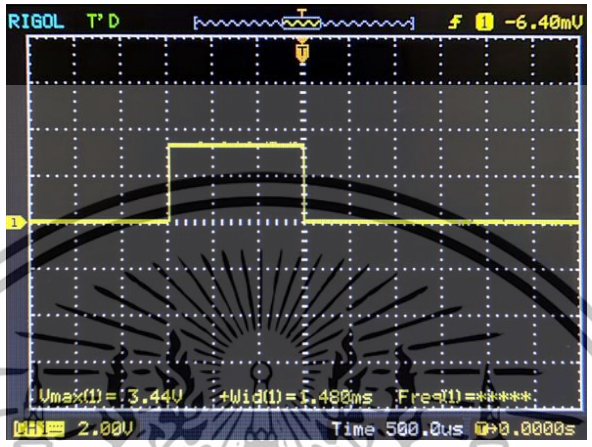
```
} else if (message == "ctrl/right") {
    myservo1.write(60);
    delay(300);
    myservo1.writeMicroseconds(1500);
}
```

รูปที่ 4.33 คำสั่งโปรแกรม right



รูปที่ 4.34 ผลการทำงานมอเตอร์ M3 เมื่อได้รับข้อความ right

โดยหลังมอเตอร์เคลื่อนที่ครบ 300 มิลลิวินาที มอเตอร์จะหยุดหมุนโดยตั้งค่าเป็นค่ากลาง โดยส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 1500 ไมโครวินาที เมื่อทำการวัดสัญญาณดังกว่างจะได้ผลดังรูปที่ 4.35



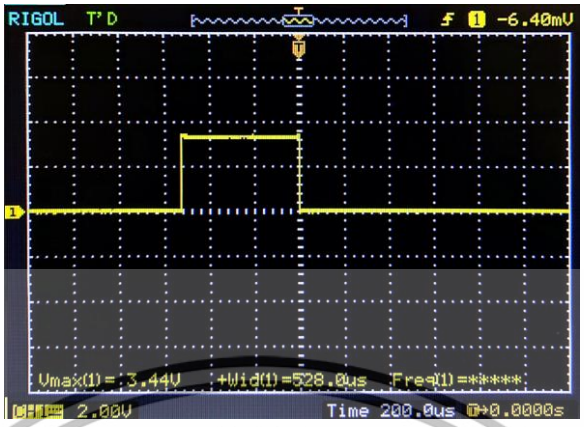
รูปที่ 4.35 ผลการทำงานมอเตอร์ M3 เมื่อต้องหยุดหมุน

4) เมื่อรับคำสั่ง holdON และ holdOFF

จากการทดสอบส่งข้อความเพื่อควบคุมแขนกลโดยใช้คำสั่ง holdON เมื่อมีการตรวจสอบเงื่อนไขแล้วว่าตรงกับเงื่อนไขที่ได้รับข้อความ holdON ดังรูปที่ 4.36 มอเตอร์ที่ทำการควบคุมแขนกลให้จับวัตถุที่ต้องการ โดยเมื่อโปรแกรมได้รับคำสั่ง "ctrl/holdON " มอเตอร์ M4 จะหมุนไปยังตำแหน่งมุม 0 องศา ซึ่งหมายความว่าแขนกลจะจับวัตถุ และส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 528 ไมโครวินาที ดังรูปที่ 4.37

```
//M4
else if (message == "ctrl/holdON") {
  myservo4.write(0);
```

รูปที่ 4.36 คำสั่งโปรแกรม holdON

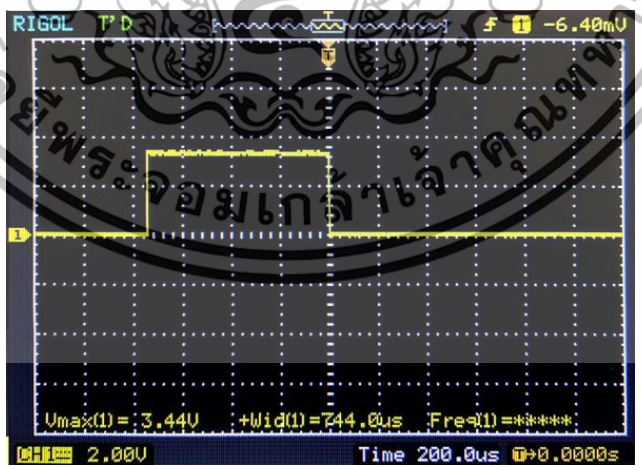


รูปที่ 4.37 ผลการทำงานมอเตอร์ M4 เมื่อได้รับข้อความ holdON

เมื่อมีการตรวจสอบเงื่อนไขแล้วว่าตรงกับเงื่อนไขที่ได้รับข้อความ holdOFF ดังรูปที่ 4.38 มอเตอร์ที่ทำการควบคุมแขนกลให้ปล่อยวัตถุที่ต้องการ โดยเมื่อโปรแกรมได้รับคำสั่ง "ctrl/holdOFF " มอเตอร์ M4 จะหมุนไปยังตำแหน่งมุม 20 องศา ซึ่งหมายความว่าแขนกลจะปล่อยวัตถุที่จับอยู่ และส่งสัญญาณ PWM ที่มีความกว้างของสัญญาณพัลส์ 744 ไมโครวินาที ดังรูปที่ 4.39

```
} else if (message == "ctrl/holdOFF") {
  myservo4.write(20);
}
```

รูปที่ 4.38 คำสั่งโปรแกรม holdOFF



รูปที่ 4.39 ผลการทำงานมอเตอร์ M4 เมื่อได้รับข้อความ holdOFF

4.1.3 อุปกรณ์รดน้ำต้นไม้

อุปกรณ์รดน้ำต้นไม้มีการออกแบบอุปกรณ์ภายในโดยการใช้อุปกรณ์ประกอบไปด้วยส่วนควบคุมและรับค่าการทำงาน คือ รีเลย์ ปั๊มน้ำขนาดเล็ก เซนเซอร์วัดความชื้นในดิน และบอร์ดไมโครคอนโทรลเลอร์ ESP32 และมีส่วนประกอบที่ทำให้อุปกรณ์รดน้ำต้นไม้นั้นสมบูรณ์ คือ กล่องกระถางต้นไม้ที่ออกแบบให้มีช่องในการใส่ต้นไม้ ช่องน้ำเพื่อรดต้นไม้ และช่องในวงจรที่ใช้เทคโนโลยีการพิมพ์ 3 มิติ โดยเมื่อทำการประกอบและจัดทำออกมาจะได้ดังรูปที่ 4.40

รูปที่ 4.40 อุปกรณ์รดน้ำต้นไม้

4.1.3.1 ผลการทำงานอุปกรณ์รดน้ำต้นไม้

เมื่อทำการสั่งการและควบคุมอุปกรณ์รดน้ำต้นไม้จากหน้าเว็บแอปพลิเคชันโดยส่งคำสั่งการสั่งการผ่าน Broker และบอร์ดไมโครคอนโทรลเลอร์ ESP32 จะทำการรับสั่งมาเพื่อสั่งงานอุปกรณ์ตามโปรแกรมที่ได้กำหนดขาควบคุมให้รีเลย์และเซนเซอร์วัดความชื้นในดิน ดังรูปที่ 4.41

```
#define relay 26
#define soilsensor 34
```

รูปที่ 4.41 คำสั่งโปรแกรมกำหนดขาการทำงาน

1) เมื่อรับคำสั่ง on

เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น “ctrl/on” ดังรูปที่ 4.42 จะสั่งการทำงานตามคำสั่งโปรแกรม ให้เรียกใช้ฟังก์ชัน startPump และเมื่อเสร็จสิ้นการทำงานในฟังก์ชัน จะเรียกใช้ฟังก์ชัน stopPump ดังรูปที่ 4.45

```
if (command == "ctrl/on") {
  startPump();
  stopPump();
```

รูปที่ 4.42 คำสั่งโปรแกรมตรวจสอบเงื่อนไขข้อความ

เนื่องจากการทำงานของอุปกรณ์รดน้ำต้นไม้มีการวัดความชื้นในดิน เพื่อนำมาแสดงผลความชื้นในดินผ่านหน้าเว็บแอปพลิเคชัน และมีการนำความชื้นในดินที่วัดได้สั่งงานการทำงานของรีเลย์ให้จ่ายไฟเพื่อให้ปั๊มน้ำขนาดเล็กทำงาน ดังคำสั่งโปรแกรมในรูปที่ 4.43 จึงได้มีการวัดแรงดันจากความชื้นที่วัดได้ในดินด้วยช่องสัญญาณที่ 2 และวัดแรงดันในการทำงานของรีเลย์ในช่องสัญญาณที่ 1

```
int soilMoistureValue = analogRead(SOIL_SENSOR_PIN);
soilMoistureValue = map(soilMoistureValue, 0, 4095, 100, 0);

unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
  previousMillis = currentMillis;

  String soilMoistureMessage ="value1:" + String(soilMoistureValue);
  client.publish(topic,soilMoistureMessage.c_str() );
  Serial.print("Soil moisture value : ");
  Serial.println(soilMoistureMessage);
}
```

รูปที่ 4.43 คำสั่งโปรแกรมที่ควบคุมรีเลย์และวัดความชื้นในดิน

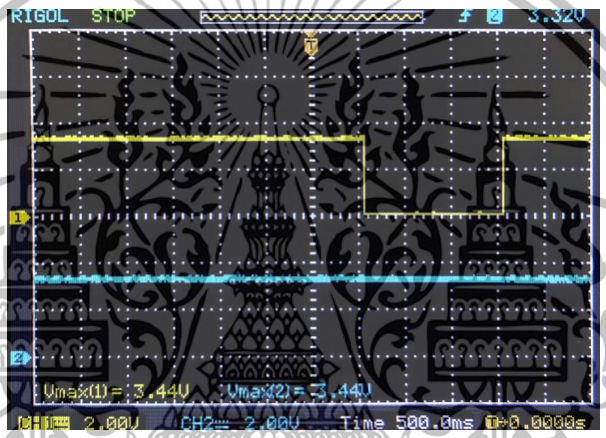
จากฟังก์ชัน startPump ที่มีการกำหนด relay ให้มีสถานะ LOW เมื่อมีการรับคำสั่งให้ on ทำงานเป็นเวลา 1500 ไมโครวินาที มีคำสั่งโปรแกรมดังรูปที่ 4.44 และมีความชื้นในดิน 0 เปอร์เซ็นต์ แรงดันที่ได้จากการวัดความชื้นจึงวัดได้ 3.44 โวลท์ ทำให้ปั๊มน้ำรดน้ำให้แก่ต้นไม้ มีผลการทำงานดังรูป 4.45 จากนั้นจะเรียกใช้ฟังก์ชัน stopPump ที่มีการกำหนด relay ให้มีสถานะ HIGH ทำงานเป็นเวลา 5000 ไมโครวินาที ทำให้ปั๊มน้ำหยุดการรดน้ำให้แก่ต้นไม้ มีผลการทำงานดังรูป 4.46

```
void startPump() {
  digitalWrite(relay, LOW);
  Serial.println("Pump and Relay ON");
  delay(1500);
}
```

รูปที่ 4.44 คำสั่งโปรแกรมฟังก์ชัน startPump

```
void stopPump() {
  digitalWrite(relay, HIGH);
  Serial.println("Pump and Relay OFF");
  delay(5000);
}
```

รูปที่ 4.45 คำสั่งโปรแกรมฟังก์ชัน stopPump



รูปที่ 4.46 ผลการทำงานของรีเลย์เมื่อรับคำสั่ง on และมีความชื้นในดิน 0 เปอร์เซ็นต์

2) เมื่อรับคำสั่ง off

เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น “ctrl/off” ดังรูปที่ 4.47 จะสั่งการทำงานตามคำสั่งโปรแกรม ให้เรียกใช้ฟังก์ชัน stopPump

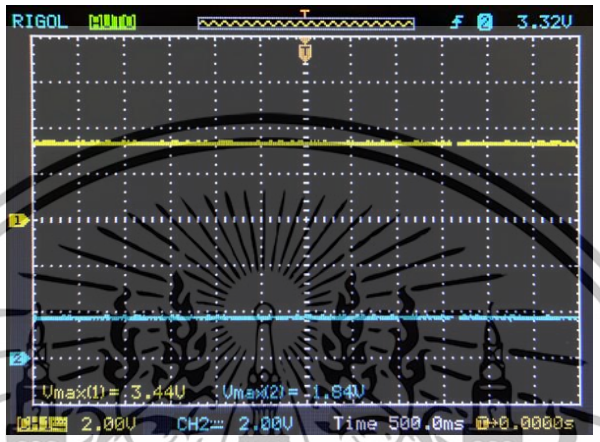
```
} else if (command == "ctrl/off") {
  stopPump();
}
```

รูปที่ 4.47 คำสั่งโปรแกรมตรวจสอบเงื่อนไขข้อความ

จากฟังก์ชัน stopPump ที่มีการกำหนด relay ให้มีสถานะ HIGH เมื่อมีการรับคำสั่งให้ off ทำงานเป็นเวลา 5000 มีคำสั่งโปรแกรมดังรูปที่ 4.48 และมีความชื้นในดิน 60 เปอร์เซ็นต์ แรงดันที่ได้จากการวัดความชื้นจึงวัดได้ 1.84 โวลท์ ทำให้ปั๊มน้ำหยุดการรดน้ำให้แก่ต้นไม้ มีผลการทำงานดังรูป 4.49

```
void stopPump() {
  digitalWrite(relay, HIGH);
  Serial.println("Pump and Relay OFF");
  delay(5000);
}
```

รูปที่ 4.48 คำสั่งโปรแกรมฟังก์ชัน stopPump

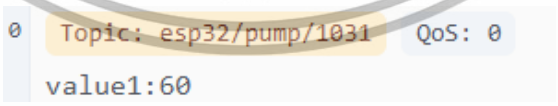


รูปที่ 4.49 ผลการทำงานของรีเลย์เมื่อรับคำสั่ง on และมีความชื้นในดิน 60 เปอร์เซ็นต์

จากนั้นค่าที่ได้จะนำส่งขึ้น Broker ผ่านคำสั่ง Publish โดยสั่งค่าในแต่ละค่าที่วัดด้วยรูปแบบ value1 ดังรูปที่ 4.50 โดยเมื่อค่าที่วัดได้ถูกส่งขึ้นมาถึง Broker ดังรูปที่ 4.51 หน้าเว็บแอปพลิเคชันจะทำการ Subscribe ค่าที่ส่งมาจากอุปกรณ์ตรวจจับความชื้นขึ้นมาแสดงบนหน้าเว็บแอปพลิเคชันดังรูปที่ 4.52

```
String soilMoistureMessage ="value1:" + String(soilMoistureValue);
client.publish(topic,soilMoistureMessage.c_str() );
Serial.print("Soil moisture value : ");
Serial.println(soilMoistureMessage);
```

รูปที่ 4.50 คำสั่งโปรแกรมในการส่งค่าให้อยู่ในรูปแบบ value

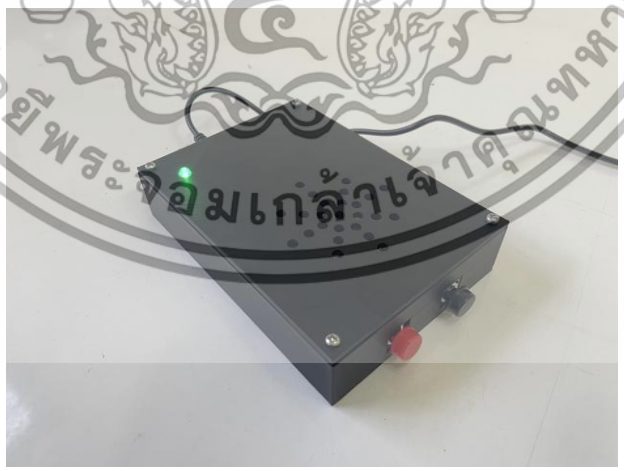


รูปที่ 4.51 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value

รูปที่ 4.52 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน

4.1.4 อุปกรณ์ตรวจจับควันไฟ

อุปกรณ์ตรวจจับควันไฟมีการออกแบบอุปกรณ์ภายในโดยการใช้อุปกรณ์ประกอบไปด้วยส่วนควบคุมและรับค่าการทำงาน คือ เซนเซอร์ MQ-2 เซนเซอร์ SHTC3 Buzzer และบอร์ดไมโครคอนโทรลเลอร์ ESP32 และมีส่วนประกอบที่ทำให้อุปกรณ์ตรวจจับควันไฟนั้นสมบูรณ์ คือ กล่องใส่วงจรของอุปกรณ์ที่ออกแบบใส่วงจรที่ใช้เทคโนโลยีการพิมพ์ 3 มิติ โดยเมื่อทำการประกอบและจัดทำออกมาจะได้ดังรูปที่ 4.53



รูปที่ 4.53 อุปกรณ์ตรวจจับควันไฟ

4.1.4.1 ผลการทำงานอุปกรณ์ตรวจจับควันไฟ

เมื่อทำการสั่งการและรับค่าการทำงานของอุปกรณ์ตรวจจับควันไฟจากหน้าเว็บแอปพลิเคชัน โดยส่งคำสั่งการสั่งการผ่าน Broker และบอร์ดไมโครคอนโทรลเลอร์ ESP32 จะทำการรับสั่งมา เพื่อสั่งงานอุปกรณ์ตามโปรแกรมที่ได้กำหนดขาควบคุมให้ Buzzer เซนเซอร์ MQ-2 และเซนเซอร์ SHTC3 โดยที่ SHTC3 จะใช้การสื่อสารผ่านโปรโตคอล I2C ใช้ขาเริ่มต้นของ I2C โดยอัตโนมัติ คือ GPIO 21 SDA (Data) และ GPIO 22SCL (Clock) ดังรูปที่ 4.54

```
Adafruit_SHTC3 shtc3;

#define MQ_PIN 34
#define BUZZER_PIN 4
```

รูปที่ 4.54 คำสั่งโปรแกรมกำหนดขาการทำงาน

1) รับค่าการทำงานของเซนเซอร์ SHTC3

จากการสร้างตัวแปร humidity และ temp ซึ่งเป็นข้อมูลที่ใช้เก็บค่าความชื้นและอุณหภูมิที่ได้จากเซนเซอร์ SHTC3 จากนั้นดึงค่าความชื้นและอุณหภูมิจากเซนเซอร์เมื่อได้ค่ามาแล้ว ค่าที่อ่านได้จะถูกเก็บไว้ในตัวแปร temperature และ humidityValue โดยที่ temperature เก็บค่าอุณหภูมิในหน่วยองศาเซลเซียส (°C) และ humidityValue เก็บค่าความชื้นสัมพัทธ์ (% RH) ดังรูปที่ 4.55 และมีผลการทำงานผ่าน Serial Monitor ดังรูปที่ 4.56

```
sensors_event_t humidity, temp;
shtc3.getEvent(&humidity, &temp);

float temperature = temp.temperature;
float humidityValue = humidity.relative_humidity;
```

รูปที่ 4.55 คำสั่งโปรแกรมอ่านข้อมูลอุณหภูมิและความชื้นจากเซนเซอร์ SHTC3

```
18:05:26.861 -> Temperature: 27.08 °C
18:05:26.861 -> Humidity: 86.39 % RH
```

รูปที่ 4.56 อุณหภูมิและความชื้นสัมพันธ์ผ่าน Serial Monitor

2) รับค่าการทำงานของเซนเซอร์ MQ-2

จากการทำงานด้วยการใช้ฟังก์ชัน calibrat() เพื่อใช้ใน setup() เพื่อทำการเปรียบเทียบเซนเซอร์ MQ-2 โดยฟังก์ชัน MQCalibration(int mq_pin) จะอ่านค่าความต้านทาน (Rs) ของเซนเซอร์ในอากาศบริสุทธิ์หลายครั้ง แล้วนำมาหาค่าเฉลี่ย จากนั้นนำค่าเฉลี่ยที่ได้มาหารด้วยค่าอากาศบริสุทธิ์ ซึ่งเป็นค่าคงที่ที่แสดงถึงอัตราส่วนของ Rs/Ro ในอากาศสะอาด เพื่อให้ได้ค่า Ro ที่ถูกต้อง ค่า Ro นี้จะถูกนำไปใช้ในการคำนวณความเข้มข้นของแก๊สต่อไป 4.57

```
void calibrat(){
  Serial.println("Calibrating...");
  Ro = MQCalibration(MQ_PIN);
  Serial.println("Calibration is done...");
  Serial.print("Ro = ");
  Serial.print(Ro);
  Serial.println(" kohm");
}

float MQCalibration(int mq_pin) {
    float val = 0;
    for (int i = 0; i < CALI_SAMPLE; i++) {
        val += MQRCal(analogRead(mq_pin));
        delay(CALI_SAMPLETIME);
    }
    val = val / CALI_SAMPLE;
    return val / RO_CLEAN_AIR;
}
```

รูปที่ 4.57 ฟังก์ชันเปรียบเทียบเซนเซอร์ MQ-2

จากนั้นจะคำนวณค่าความต้านทานของเซนเซอร์ (Rs) จากค่า Analog ที่อ่านได้จากขา MQ_PIN โดยใช้สูตรคำนวณที่แปลงค่า ADC เป็นค่าความต้านทานและฟังก์ชัน MQRead(int mq_pin) จะอ่านค่า Rs หลายครั้ง แล้วนำมาหาค่าเฉลี่ยเพื่อให้ได้ค่า Rs ที่แม่นยำยิ่งขึ้น ดังรูปที่ 4.58

```
float MQRCal(int raw_adc) {
    if (raw_adc == 0) return -1;
    float rs = (RL_VALUE * (4095.0 - raw_adc) / raw_adc);
    return constrain(rs, 0.1, 10000.0);
}

float MQRead(int mq_pin) {
    float rs = 0;
    for (int i = 0; i < READ_SAMPLE; i++) {
        rs += MQRCal(analogRead(mq_pin));
        delay(READ_SAMPLETIME);
    }
    return rs / READ_SAMPLE;
}
```

รูปที่ 4.58 ฟังก์ชันอ่านค่าความต้านทาน

และคำนวณความเข้มข้นของแก๊สแต่ละชนิด โดยรับค่าอัตราส่วน Rs/Ro และชนิดของแก๊ส จากนั้นเรียกใช้ฟังก์ชัน MQPercentage เพื่อคำนวณค่า ppm โดยใช้สมการ Log-Log scale ที่ปรับเทียบสำหรับแก๊สแต่ละชนิด ดังรูปที่ 4.59

```
int MQGasPercentage(float rs_ro_ratio, int gas_id) {
  if (rs_ro_ratio <= 0) return 0;

  switch (gas_id) {
    case GAS_LPG:
      return MQPercentage(rs_ro_ratio, LPGCurve);
    case GAS_CO:
      return MQPercentage(rs_ro_ratio, COCurve);
    case GAS_SMOKE:
      return MQPercentage(rs_ro_ratio, SmokeCurve);
    default:
      return 0;
  }
}

int MQPercentage(float rs_ro_ratio, float *pcurve) {
    rs_ro_ratio = constrain(rs_ro_ratio, 0.01, 10.0);
    return (int)pow(10, ((log10(rs_ro_ratio) - pcurve[1]) / pcurve[2]) + pcurve[0]);
}
```

รูปที่ 4.59 ฟังก์ชันคำนวณความเข้มข้นของแก๊ส

จากนั้นจะมีการเรียกใช้ฟังก์ชันเหล่านี้เพื่ออ่านค่าเซนเซอร์ MQ-2 และการตรวจสอบว่าค่าที่ได้เกินกว่าค่าที่กำหนดไว้หรือไม่ ถ้าเกินแสดงว่าตรวจพบการเผาไหม้และควันไฟ และจะสั่งให้ Buzzer ทำงาน เพื่อแจ้งเตือน ถ้าไม่เกิน Buzzer จะหยุดทำงาน ดังรูปที่ 4.60

```
int level = analogRead(MQ_PIN);

float ratio = MQRead(MQ_PIN) / Ro;

int lpg = MQGasPercentage(ratio, GAS_LPG);
int co = MQGasPercentage(ratio, GAS_CO);
int smoke = MQGasPercentage(ratio, GAS_SMOKE);


if (smoke > SMOKE_THRESHOLD || temperature > TEMP_THRESHOLD) {
  Serial.println("Warning: High smoke level or temperature detected!");
  digitalWrite(BUZZER_PIN, LOW);
} else {
  digitalWrite(BUZZER_PIN, HIGH);
}

Serial.println("-----------------------------------------------------");
delay(1000);
```

รูปที่ 4.60 คำสั่งโปรแกรมอ่านค่าเซนเซอร์ MQ-2 และตรวจสอบเงื่อนไข

เมื่ออยู่ในสภาพอากาศปกติไม่มีควันไฟหรือการเผาไหม้เกิดขึ้น จะมีผลการทำงานผ่าน Serial Monitor ดังรูปที่ 4.61 และเมื่อเกิดการเผาไหม้เกิดขึ้นจนสามารถตรวจจับควันไฟได้ จะมีผลการทำงานอุปกรณ์ตรวจจับควันไฟผ่าน Serial Monitor ดังรูปที่ 4.62

```
18:05:26.861 -> LPG: 3 ppm    CO: 33 ppm    SMOKE: 17 ppm
18:05:26.861 -> Temperature: 27.08 °C
18:05:26.861 -> Humidity: 86.39 % RH
18:05:26.861 -> -----------------------------------------------------
```

รูปที่ 4.61 ค่าที่วัดได้สภาพอากาศปกติไม่มีควันไฟหรือการเผาไหม้

```
18:06:41.885 -> LPG: 168 ppm    CO: 8243 ppm    SMOKE: 1845 ppm
18:06:41.885 -> Temperature: 27.26 °C
18:06:41.919 -> Humidity: 86.43 % RH
18:06:41.919 -> Warning: High smoke level or temperature detected!
18:06:41.919 -> -----------------------------------------------------
```

รูปที่ 4.62 ค่าที่วัดได้เมื่อเกิดการเผาไหม้เกิดขึ้น

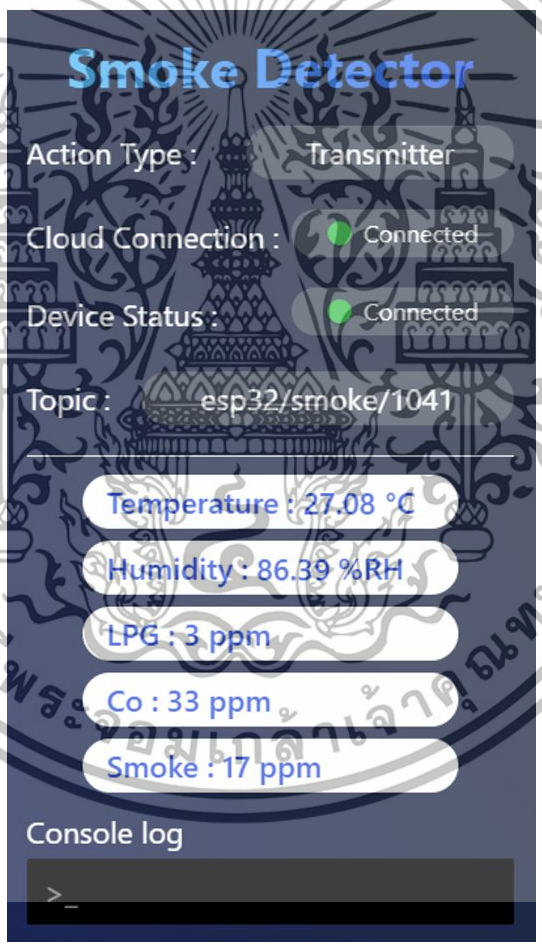
จากนั้นค่าที่ได้จะนำส่งขึ้น Broker ผ่านคำสั่ง Publish โดยสั่งค่าในแต่ละค่าที่วัดด้วยรูปแบบ value1 value2 value3 value4 และ value5 ดังรูปที่ 4.63 โดยเมื่อค่าที่วัดได้ถูกส่งขึ้นมาถึง Broker ดังรูปที่ 4.64 หน้าเว็บแอปพลิเคชันจะทำการ Subscribe ค่าที่ส่งมาจากอุปกรณ์ตรวจจับควันขึ้นมาแสดงบนหน้าเว็บแอปพลิเคชันดังรูปที่ 4.65

```
char payload[200];
snprintf(payload, sizeof(payload),
    "value1:%d, value2:%d, value3:%d, value4:%.2f, value5:%.2f",
    lpg, co, smoke, temperature, humidityValue);
client.publish(topic, payload);
```

รูปที่ 4.63 คำสั่งโปรแกรมส่งค่าจากอุปกรณ์ตรวจจับควันขึ้น Broker

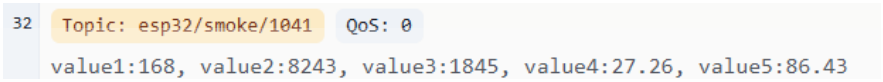
```
16  Topic: esp32/smoke/1041   QoS: 0
    value1:3, value2:33, value3:17, value4:27.08, value5:86.39
```

รูปที่ 4.64 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value

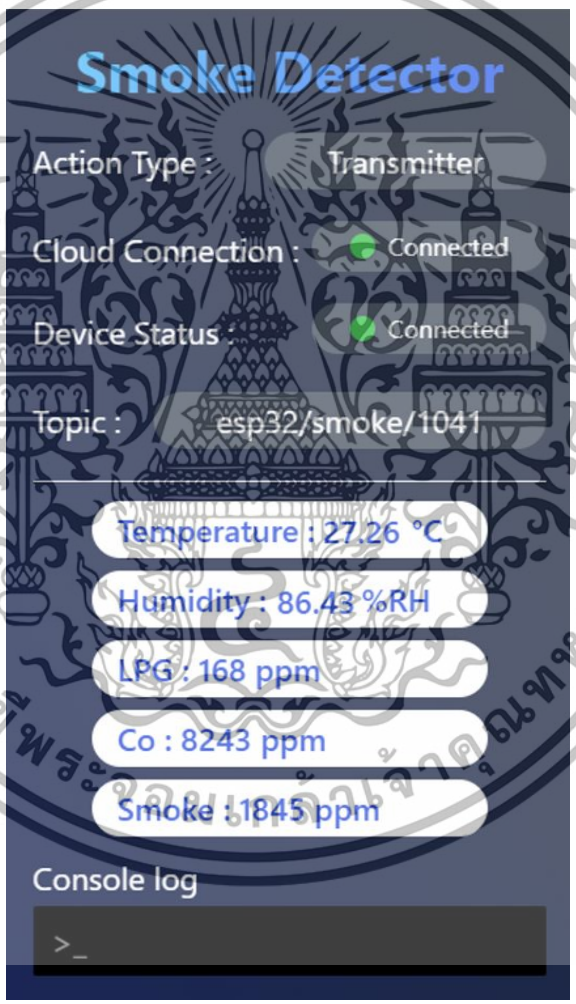


รูปที่ 4.65 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน

เช่นเดียวกันกับเมื่ออุปกรณ์ตรวจจับควันไฟได้จะทำการนำส่งขึ้น Broker ผ่านคำสั่ง Publish โดยสั่งค่าในแต่ละค่าที่วัดด้วยรูปแบบ value1 value2 value3 value4 โดยเมื่อค่าที่วัดได้ถูกส่งขึ้นมาถึง Broker ดังรูปที่ 4.66 หน้าเว็บแอปพลิเคชันจะทำการ Subscribe ค่าที่ส่งมาจากอุปกรณ์ตรวจจับควันขึ้นมาแสดงบนหน้าเว็บแอปพลิเคชันดังรูปที่ 4.67



รูปที่ 4.66 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value



รูปที่ 4.67 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน

4.1.5 ตัวอย่างอุปกรณ์ควบคุมใด ๆ

อุปกรณ์นำเข้าจากภายนอกมีการออกแบบอุปกรณ์ภายในโดยการใช้อุปกรณ์ประกอบไปด้วยส่วนควบคุมและรับค่าการทำงาน คือ รีเลย์ และ RGB LED และบอร์ดไมโครคอนโทรลเลอร์ ESP32 และมีส่วนประกอบที่ทำให้อุปกรณ์นำเข้าจากภายนอก คือ กรอบที่ออกแบบให้มีช่องในการใส่พัดลม LED และวงจร โดยใช้เทคโนโลยีการพิมพ์ 3 มิติ โดยเมื่อทำการประกอบและจัดทำออกมาจะได้ดังรูปที่ 4.68

รูปที่ 4.68 ตัวอย่างอุปกรณ์ควบคุมใด ๆ

4.1.5.1 ผลการสั่งการตัวอย่างอุปกรณ์ควบคุมใด ๆ

เมื่อทำการสั่งการและอุปกรณ์นำเข้าตัวอย่างจากภายนอกผ่านหน้าเว็บแอปพลิเคชัน โดยส่งคำสั่งการสั่งการผ่าน Broker และบอร์ดไมโครคอนโทรลเลอร์ ESP32 จะทำการรับสั่งมา เพื่อสั่งงานอุปกรณ์ตามโปรแกรมที่ได้กำหนดขาควบคุมให้รีเลย์และ LED มีการทำงาน ดังรูปที่ 4.69 และได้กำหนดการทำงานเริ่มต้นไว้ ดังรูปที่ 4.70

```
#define RELAY_PIN 33
#define BLUE_PIN 5
#define GREEN_PIN 18
#define RED_PIN 19
```

รูปที่ 4.69 คำสั่งโปรแกรมกำหนดขาการทำงาน

```
void setup() {
    Serial.begin(115200);
    pinMode(RELAY_PIN, OUTPUT);
    pinMode(RED_PIN, OUTPUT);
    pinMode(GREEN_PIN, OUTPUT);
    pinMode(BLUE_PIN, OUTPUT);
    pinMode(resetWifi_btn, INPUT_PULLUP);
    pinMode(statusconnect, OUTPUT);
    pinMode(statusnewwifi, OUTPUT);
    controlRelayAndLight(false, false, false, false);
```

รูปที่ 4.70 คำสั่งโปรแกรมกำหนดการทำงานเริ่มต้น

จากนั้นสร้างฟังก์ชัน controlRelayAndLight() เป็นฟังก์ชันที่ใช้ควบคุมรีเลย์และไฟ RGB LED โดยรับค่าพารามิเตอร์เป็นตัวแปรแบบ Boolean 4 ค่า ได้แก่ relayState red green และ blue เพื่อตัดสินใจว่าจะเปิด (true) หรือปิด (false) อุปกรณ์แต่ละตัว โดยภายในฟังก์ชันค่าของ relayState ถูกใช้กำหนดสถานะของรีเลย์ ซึ่งสลับค่ากับปกติ โดยที่ LOW หมายถึงเปิดรีเลย์ และ HIGH หมายถึงปิดรีเลย์ ส่วนไฟ LED ควบคุมโดยตรงผ่านขา RED_PIN GREEN_PIN และ BLUE_PIN ซึ่งใช้ค่า Boolean เมื่อ true จะส่งสัญญาณ HIGH เพื่อเปิดไฟ และ false จะส่ง LOW เพื่อปิดไฟ ดังรูปที่ 4.71

```
void controlRelayAndLight(bool relayState, bool red, bool green, bool blue) {
    digitalWrite(RELAY_PIN, relayState ? LOW : HIGH);
    Serial.printf("Relay %s\n", relayState ? "ON" : "OFF");

    digitalWrite(RED_PIN, red ? HIGH : LOW);
    digitalWrite(GREEN_PIN, green ? HIGH : LOW);
    digitalWrite(BLUE_PIN, blue ? HIGH : LOW);
```

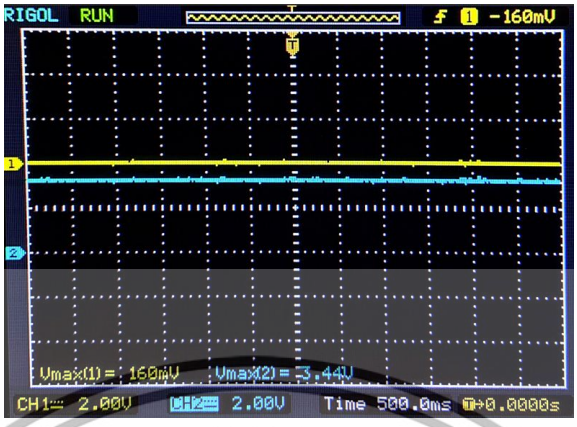
รูปที่ 4.71 ฟังก์ชันที่ใช้ควบคุมรีเลย์และไฟ RGB LED

1) เมื่อรับคำสั่ง red_on

เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น red_on ดังรูปที่ 4.72 เป็นคำสั่งให้เปิดรีเลย์ และเปิดไฟ LED สีแดง โดยการควบคุมการฟังก์ชันจะเปิดรีเลย์และเปิดไฟ LED สีแดงเท่านั้น มีผลการทำงานดังรูป 4.73 โดยที่ช่องสัญญาณที่ 1 เป็นผลการทำงานของรีเลย์และช่องสัญญาณที่ 2 เป็นผลการทำงานของไฟ LED สีแดง

```
} else if (message == "ctrl/red_on") {
    controlRelayAndLight(true, true, false, false);
```

รูปที่ 4.72 คำสั่งโปรแกรม red_on



รูปที่ 4.73 ผลการทำงานของรีเลย์เมื่อ true และไฟ LED สีแดง true

2) เมื่อรับคำสั่ง green_on

เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น green_on ดังรูปที่ 4.74 เป็นคำสั่งให้เปิดรีเลย์ และเปิดไฟ LED สีเขียว โดยการควบคุมการฟังก์ชันจะเปิดรีเลย์และเปิดไฟ LED สีเขียวเท่านั้น มีผลการทำงานดังรูป 4.75 โดยที่ช่องสัญญาณที่ 1 เป็นผลการทำงานของรีเลย์และช่องสัญญาณที่ 2 เป็นผลการทำงานของไฟ LED สีเขียว

```
} else if (message == "ctrl/green_on") {
    controlRelayAndLight(true, false, true, false);
```

รูปที่ 4.74 คำสั่งโปรแกรม green_on

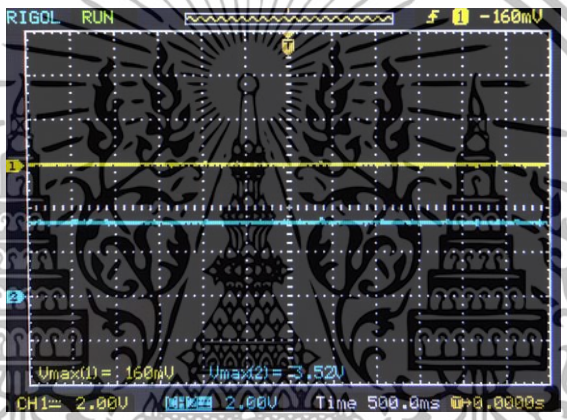
รูปที่ 4.75 ผลการทำงานของรีเลย์เมื่อ true และไฟ LED สีแดง true

3) เมื่อรับคำสั่ง blue_on

เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น blue_on ดังรูปที่ 4.76 เป็นคำสั่งให้เปิดรีเลย์ และเปิดไฟ LED สีน้ำเงิน โดยการควบคุมการฟังก์ชันจะเปิดรีเลย์และเปิดไฟ LED สีน้ำเงินเท่านั้น มีผลการทำงานดังรูป 4.77 โดยที่ช่องสัญญาณที่ 1 เป็นผลการทำงานของรีเลย์และช่องสัญญาณที่ 2 เป็นผลการทำงานของไฟ LED สีน้ำเงิน

```
} else if (message == "ctrl/blue_on") {
    controlRelayAndLight(true, false, false, true);
```

รูปที่ 4.76 คำสั่งโปรแกรม blue_on



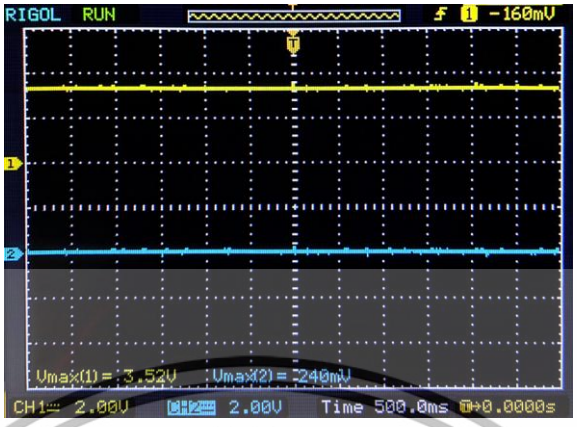
รูปที่ 4.77 ผลการทำงานของรีเลย์เมื่อ true และไฟ LED สีน้ำเงิน true

4) เมื่อรับคำสั่ง stop

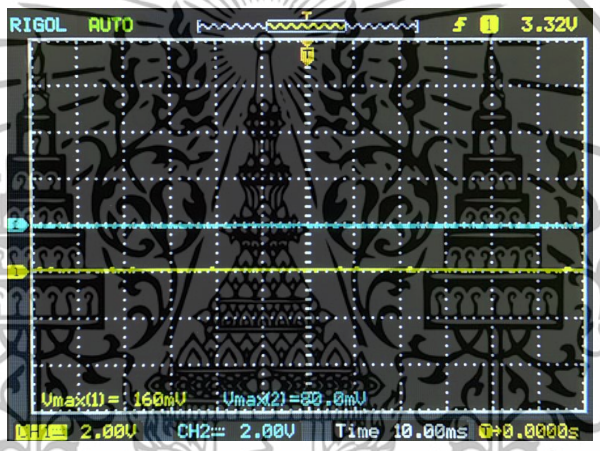
เมื่อทำการการตรวจสอบเงื่อนไขแต่ละคำสั่งที่เข้ามาเป็น stop ดังรูปที่ 4.78 เป็นคำสั่งปิดรีเลย์และปิดไฟ LED ทุกดวง มีผลการทำงานดังรูป 4.79 โดยที่ช่องสัญญาณที่ 1 เป็นผลการทำงานของรีเลย์และช่องสัญญาณที่ 2 เป็นผลการทำงานของไฟ LED สีแดง และรูปที่ 4.80 โดยที่ช่องสัญญาณที่ 1 เป็นผลการทำงานของไฟ LED สีเขียว และช่องสัญญาณที่ 2 เป็นผลการทำงานของไฟ LED สีน้ำเงิน

```
} else if (message == "ctrl/stop") {
    controlRelayAndLight(false, false, false, false);
}
```

รูปที่ 4.78 คำสั่งโปรแกรม stop



รูปที่ 4.79 ผลการทำงานของรีเลย์เมื่อ false และไฟ LED สีแดง false



รูปที่ 4.80 ผลการทำงานของไฟ LED สีเขียวและไฟ LED สีน้ำเงิน false

4.1.6 ตัวอย่างอุปกรณ์รับข้อมูลใด ๆ

อุปกรณ์รับข้อมูลใด ๆ มีการออกแบบอุปกรณ์ภายในโดยการใช้อุปกรณ์ประกอบไปด้วยส่วนควบคุมและรับค่าการทำงาน คือ เซนเซอร์ MQ-6 Buzzer และบอร์ดไมโครคอนโทรลเลอร์ ESP32 และมีส่วนประกอบที่ทำให้อุปกรณ์ตรวจจับก๊าซปิโตรเลียมเหลวนั้นสมบูรณ์ คือ กล่องใส่วงจรของอุปกรณ์ที่ออกแบบใส่วงจรที่ใช้เทคโนโลยีการพิมพ์ 3 มิติ โดยเมื่อทำการประกอบและจัดทำออกมาจะได้ดังรูปที่ 4.81

รูปที่ 4.81 ตัวอย่างอุปกรณ์รับข้อมูลใด ๆ

4.1.6.1 ผลการทำงานตัวอย่างอุปกรณ์รับข้อมูลใด ๆ

ทำการสั่งการและรับค่าการทำงานของอุปกรณ์ตรวจจับก๊าซปิโตรเลียมเหลว จากหน้าเว็บแอปพลิเคชัน โดยส่งคำสั่งการสั่งการผ่าน Broker และบอร์ดไมโครคอนโทรลเลอร์ ESP32 จะทำการรับสั่งมา เพื่อสั่งงานอุปกรณ์ตามโปรแกรมที่ได้กำหนดขาควบคุมให้ Buzzer เซนเซอร์ MQ-6 ดังรูปที่ 4.82

```
#define EEPROM_SIZE 512
#define BUZZER_PIN 4
const byte MQ6_Pin = 32;
```

รูปที่ 4.82 คำสั่งโปรแกรมกำหนดขาการทำงาน

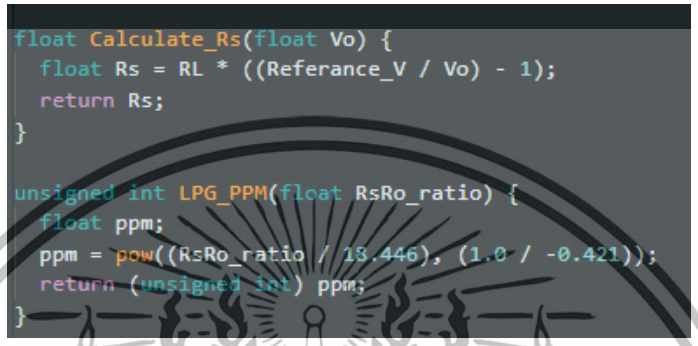
1) รับค่าการทำงานของเซนเซอร์ MQ-6

การอ่านค่าจากเซนเซอร์ MQ-6 จะฟังก์ชัน ดังรูปที่ 4.83 ในการอ่านค่าแรงดันไฟฟ้าจากขา Analog ของเซนเซอร์ MQ-6 โดยใช้ฟังก์ชัน analogRead() ค่าที่อ่านได้จะถูกแปลงค่าเป็นแรงดันไฟฟ้าที่เปลี่ยนแปลงตามความเข้มข้นของแก๊ส การอ่านค่านี้จะทำหลายครั้ง เพื่อนำมาหาค่าเฉลี่ย ลดสัญญาณรบกวน และให้ค่าที่แม่นยำยิ่งขึ้น

```
float Get_mVolt(byte AnalogPin) {
  int ADC_Value = analogRead(AnalogPin);
  delay(1);
  float mVolt = ADC_Value * (Referance_V / 4096.0);
  return mVolt;
}
```

รูปที่ 4.83 คำสั่งโปรแกรมอ่านค่าจากเซนเซอร์ MQ-6

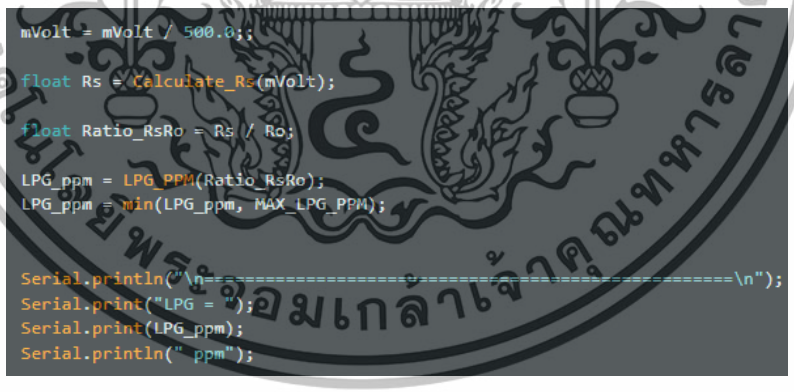
จากนั้นจะนำค่าที่ได้ไปคำนวณค่าความต้านทานของเซนเซอร์ ตอนเริ่มต้นโปรแกรมมีการวัดค่า Ro แล้วนำค่า Rs ที่คำนวณได้ในแต่ละรอบการทำงาน มาหารด้วย Ro เพื่อหาอัตราส่วนความต้านทาน (Rs/Ro) อัตราส่วนนี้จะถูกนำไปใช้ในการคำนวณความเข้มข้นของแก๊ส ดังรูปที่ 4.84



```
float Calculate_Rs(float Vo) {
  float Rs = RL * ((Referance_V / Vo) - 1);
  return Rs;
}

unsigned int LPG_PPM(float RsRo_ratio) {
  float ppm;
  ppm = pow((RsRo_ratio / 18.446), (1.0 / -0.421));
  return (unsigned int) ppm;
}
```

รูปที่ 4.84 คำสั่งโปรแกรมคำนวณค่าความต้านทานของเซนเซอร์

จากนั้นคำนวณความเข้มข้นของก๊าซปิโตรเลียมเหลวในหน่วย ppm (parts per million) โดยใช้อัตราส่วนความต้านทาน (Rs/Ro) ที่คำนวณได้ และมีความสัมพันธ์แบบ logarithmic ระหว่างอัตราส่วนความต้านทานและความเข้มข้นของก๊าซ ดังรูปที่ 4.85



```
mVolt = mVolt / 500.0;;

float Rs = Calculate_Rs(mVolt);

float Ratio_RsRo = Rs / Ro;

LPG_ppm = LPG_PPM(Ratio_RsRo);
LPG_ppm = min(LPG_ppm, MAX_LPG_PPM);


Serial.println("\n==========================================\n");
Serial.print("LPG = ");
Serial.print(LPG_ppm);
Serial.println(" ppm");
```

รูปที่ 4.85 คำสั่งโปรแกรมคำนวณความเข้มข้นของก๊าซปิโตรเลียมเหลวในหน่วย ppm

จะมีการตรวจสอบว่าค่าที่ได้เกินกว่าค่าที่กำหนดไว้หรือไม่ ถ้าเกินแสดงว่าตรวจพบก๊าซปิโตรเลียมเหลว และจะสั่งให้ Buzzer ทำงาน เพื่อแจ้งเตือน ถ้าไม่เกิน Buzzer จะหยุดทำงาน ดังรูปที่ 4.86

```
if (LPG_ppm > LPG_THRESHOLD) {
    Serial.println("-------- LPG Detected! --------------");
    digitalWrite(BUZZER_PIN, LOW);
    delay(500);
} else {
    digitalWrite(BUZZER_PIN, HIGH);
}
Serial.println("\n=================================================\n");
```

รูปที่ 4.86 เงื่อนไขในการตรวจสอบก๊าซปิโตรเลียมเหลวเกินค่าที่กำหนด

เมื่ออยู่ในสภาพอากาศปกติไม่มีก๊าซปิโตรเลียมเหลวรั่วไหลเกิดขึ้น จะมีผลการทำงานผ่าน Serial Monitor ดังรูปที่ 4.87 และเมื่อเกิดการรั่วไหลของก๊าซจนสามารถตรวจจับได้ จะมีผลการทำงานอุปกรณ์ตรวจจับก๊าซปิโตรเลียมเหลวผ่าน Serial Monitor ดังรูปที่ 4.88

```
22:23:11.780 ->
22:23:11.780 -> LPG = 3 ppm
22:23:11.826 ->
22:23:11.826 -> =================================================
```

รูปที่ 4.87 ค่าที่วัดได้สภาพอากาศปกติไม่มีแก๊สรั่วไหล

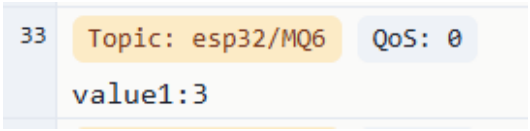
```
22:23:46.434 ->
22:23:46.434 -> LPG = 334 ppm
22:23:46.434 -> -------- LPG Detected! --------------
22:23:46.951 ->
22:23:46.951 -> =================================================
```

รูปที่ 4.88 ค่าที่วัดได้เมื่อเกิดแก๊สรั่วไหล

จากนั้นค่าที่ได้จะนำส่งขึ้น Broker ผ่านคำสั่ง Publish โดยสั่งค่าในแต่ละค่าที่วัดด้วยรูปแบบ value1 ดังรูปที่ 4.89 โดยเมื่อค่าที่วัดได้ถูกส่งขึ้นมาถึง Broker ดังรูปที่ 4.90 หน้าเว็บแอปพลิเคชันจะทำการ Subscribe ค่าที่ส่งมาจากอุปกรณ์ตรวจจับการรั่วไหลของก๊าซขึ้นมาแสดงบนหน้าเว็บแอปพลิเคชันดังรูปที่ 4.91

```
char payload[20];
sprintf(payload, "value1:%d", LPG_ppm);
client.publish(topic, payload);
```

รูปที่ 4.89 คำสั่งโปรแกรมส่งค่าจากอุปกรณ์ตรวจจับก๊าซปิโตรเลียมเหลว



รูปที่ 4.90 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value

รูปที่ 4.91 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน

เช่นเดียวกันกับเมื่ออุปกรณ์ตรวจจับการรั่วไหลของก๊าซจะทำการนำส่งขึ้น Broker ผ่านคำสั่ง Publish โดยสั่งค่าในแต่ละค่าที่วัดด้วยรูปแบบ value1 โดยเมื่อค่าที่วัดได้ถูกส่งขึ้นมาถึง Broker ดังรูปที่ 4.92 หน้าเว็บแอปพลิเคชันจะทำการ Subscribe ค่าที่ส่งมาจากอุปกรณ์ตรวจจับก๊าซปิโตรเลียมเหลวขึ้นมาแสดงบนหน้าเว็บแอปพลิเคชันดังรูปที่ 4.93

รูปที่ 4.92 ค่าการทำงานของอุปกรณ์ที่อยู่ในรูปแบบ value

รูปที่ 4.93 ค่าที่วัดได้จากอุปกรณ์นำมาแสดงหน้าเว็บแอปพลิเคชัน

4.2 การทดสอบการเชื่อมต่อกับ MQTT Broker

4.2.1 ผลการเชื่อมต่อ HiveMQ Broker กับเว็บแอปพลิเคชัน

ทดสอบการเชื่อมต่อ HiveMQ Broker กับเว็บแอปพลิเคชัน โดยจะเป็นขั้นตอนที่ทำให้เว็บแอปสามารถสื่อสารกับอุปกรณ์ IoT หรือระบบอื่น ๆ ผ่านโปรโตคอล MQTT ซึ่งเป็นโปรโตคอลที่ถูกออกแบบมาเพื่อการสื่อสารระหว่างอุปกรณ์ที่มีข้อจำกัดด้านพลังงานและเครือข่าย ในการเชื่อมต่อกับ HiveMQ Broker จะต้องมีข้อมูลที่จำเป็น คือ URL ของเซิร์ฟเวอร์ ชื่อผู้ใช้ และรหัสผ่าน ดังรูปที่ 4.94 เป็นข้อมูลสำคัญที่ได้สมัครไว้กับ HiveMQ Cloud ไว้เรียบร้อยแล้ว

รูปที่ 4.94 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker

เมื่อมีการสร้างการเชื่อมต่อกับ HiveMQ Broker ใช้ mqtt.connect() ซึ่งเป็นฟังก์ชันที่ทำหน้าที่เชื่อมต่อกับเซิร์ฟเวอร์ MQTT ผ่าน URL ของ broker ที่กำหนดไว้แล้ว จัดการการเชื่อมต่อ (connection) หลังจากที่เริ่มการเชื่อมต่อแล้ว จะมีการตั้งค่าการทำงานเมื่อการเชื่อมต่อ

สำเร็จ (on connect) และเมื่อเกิดข้อผิดพลาด (on error) มีคำสั่งโปรแกรมดังรูปที่ 4.95 เมื่อเชื่อมต่อ HiveMQ Broker สำเร็จไม่มีข้อผิดพลาดจะแสดงผลผ่าน Console เบราว์เซอร์ ดังรูปที่ 4.96

```
client.on("connect", () => {
  setIsConnected(true);
  console.log("Connected to HiveMQ broker over WebSocket");
});

client.on("error", (err) => {
  console.error("Connection error: ", err.message);
  console.error("Details: ", err);
  client.end();
});
```

รูปที่ 4.95 คำสั่งโปรแกรมการเชื่อมต่อ

รูปที่ 4.96 ผลการเชื่อมต่อผ่าน Console เบราว์เซอร์

4.2.2 ผลการเชื่อมต่อ HiveMQ Broker กับบอร์ดไมโครคอนโทรลเลอร์ ESP32

ทดสอบการเชื่อมต่อ HiveMQ Broker กับบอร์ดไมโครคอนโทรลเลอร์ ESP32 โดยจะเป็นขั้นตอนที่ทำให้บอร์ดไมโครคอนโทรลเลอร์ ESP32 สามารถรับคำสั่งการควบคุมและส่งผลจากอุปกรณ์ ผ่านโปรโตคอล MQTT ในการเชื่อมต่อกับ HiveMQ Broker จะต้องมีข้อมูลที่จำเป็น คือ URL ของเซิร์ฟเวอร์ ชื่อผู้ใช้ รหัสผ่าน และพอร์ตในการสื่อสาร ดังรูปที่ 4.97 เป็นข้อมูลสำคัญที่ได้สมัครไว้กับ HiveMQ Cloud ไว้เรียบร้อยแล้ว

```
// MQTT Broker
const char *mqtt_broker = "4cff082ff4a746da91e[redacted]hivemq.cloud";
const char *topic = "esp32/car/1012";
const char *mqtt_username = "admin";
const char *mqtt_password = "Bam[redacted]
const int mqtt_port = 8883;
```

รูปที่ 4.97 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker

จากคำสั่งโปรแกรมดังรูปที่ 4.98 client.setServer() จะถูกใช้เพื่อกำหนดค่าเซิร์ฟเวอร์ MQTT Broker โดยระบุ URL และพอร์ตที่ใช้งาน รวมถึงตั้งค่า callback function (callback) เพื่อจัดการกับข้อความที่ได้รับจาก Topic ที่ได้ Subscribe ไว้ ในขั้นตอนการเชื่อมต่อกับ MQTT Broker มีการใช้ while loop เพื่อตรวจสอบว่าการเชื่อมต่อสำเร็จหรือไม่ โดยจะสร้าง client_id โดยรวม MAC address ของ ESP32 เพื่อให้มีความเฉพาะตัว และพยายามทำการเชื่อมต่อจนกว่าจะเชื่อมต่อได้สำเร็จ เมื่อการเชื่อมต่อสำเร็จจะแสดงข้อความใน Serial Monitor ดังรูปที่ 4.99 เพื่อบอกสถานะการเชื่อมต่อ หากการเชื่อมต่อสำเร็จจะมีการแสดงข้อความว่า "Connected" และทำการส่งข้อความ "connected" ขึ้น HiveMQ Broker

```
client.setServer(mqtt_broker, mqtt_port);
client.setCallback(callback);
while (!client.connected()) {
  String client_id = "esp32-client-";
  client_id += String(WiFi.macAddress());
  Serial.printf("The client %s connects to the public mqtt broker\n", client_id.c_str());
  if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
    Serial.println(" ========== HiveMQ MQTT broker [ Connected ] ==========");
    client.publish(topic, "connected");
  } else {
    Serial.print("failed with state ");
    Serial.print(client.state());
    Serial.print(" NOT CONNECT !!!!!!!");
    delay(2000);
  }
}
```

รูปที่ 4.98 คำสั่งโปรแกรมเชื่อมต่อบอร์ดไมโครคอนโทรลเลอร์ ESP32 กับ HiveMQ Broker

```
21:37:51.805 -> -------- Connected to the WiFi network --------
21:37:51.843 -> The client esp32-client-D0:EF:76:58:B6:60 connects to the public mqtt broker
21:37:53.850 ->  ========== HiveMQ MQTT broker [ Connected ] ==========
21:40:10.220 ->
```

รูปที่ 4.99 ผลการเชื่อมต่อผ่าน Serial Monitor

4.2.3 ผลการเชื่อมต่อ Adafruit Broker กับเว็บแอปพลิเคชัน

ทดสอบการเชื่อมต่อ Adafruit Broker กับเว็บแอปพลิเคชัน โดยจะเป็นขั้นตอนที่ทำให้เว็บแอปสามารถสื่อสารกับอุปกรณ์ IoT หรือระบบอื่น ๆ ผ่านโปรโตคอล MQTT ซึ่งเป็นโปรโตคอลที่ถูกออกแบบมาเพื่อการสื่อสารระหว่างอุปกรณ์ที่มีข้อจำกัดด้านพลังงานและเครือข่าย ในการเชื่อมต่อกับ Adafruit Broker จะต้องมีข้อมูลที่จำเป็น คือ URL ของเซิร์ฟเวอร์ ชื่อผู้ใช้ และ รหัสผ่าน ดังรูปที่ 4.100 เป็นข้อมูลสำคัญที่ได้สมัครไว้กับ Adafruit Broker ไว้เรียบร้อยแล้ว

```
const MQTT_BROKER = "wss://io.adafruit.com/mqtt/";
const TOPIC = "iotp[redacted]feeds/testweb";

const client = mqtt.connect(MQTT_BROKER, {
  username: "iot[redacted]",
  password: [redacted]KeXQu6YYyjZ81b"
});
```

รูปที่ 4.100 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker

เมื่อมีการสร้างการเชื่อมต่อกับ Adafruit Broker ใช้ mqtt.connect() ซึ่งเป็นฟังก์ชันที่ทำหน้าที่เชื่อมต่อกับเซิร์ฟเวอร์ MQTT ผ่าน URL ของ broker ที่กำหนดไว้แล้ว จัดการการเชื่อมต่อ (connection) หลังจากที่เริ่มการเชื่อมต่อแล้ว จะมีการตั้งค่าการทำงานเมื่อการเชื่อมต่อสำเร็จ (on connect) และเมื่อเกิดข้อผิดพลาด (on error) มีคำสั่งโปรแกรมดังรูปที่ 4.101 เมื่อเชื่อมต่อ Adafruit Broker สำเร็จไม่มีข้อผิดพลาดจะแสดงผลผ่าน Console เบราว์เซอร์ดังรูปที่ 4.102

```
client.on("connect", () => {
  setStatus("Connected");
  console.log("Connected to Adafruit Broker");
});

client.on("error", (err) => {
  console.error("MQTT Error:", err);
  setStatus("Error");
});
```

รูปที่ 4.101 คำสั่งโปรแกรมการเชื่อมต่อ

รูปที่ 4.102 ผลการเชื่อมต่อผ่าน Console เบราว์เซอร์

4.2.4 ผลการเชื่อมต่อ Adafruit Broker กับบอร์ดไมโครคอนโทรลเลอร์ ESP32

ทดสอบการเชื่อมต่อ Adafruit Broker กับบอร์ดไมโครคอนโทรลเลอร์ ESP32 โดยจะเป็นขั้นตอนที่ทำให้บอร์ดไมโครคอนโทรลเลอร์ ESP32 สามารถรับคำสั่งการควบคุมและส่งผลจากอุปกรณ์ ผ่านโปรโตคอล MQTT ในการเชื่อมต่อกับ Adafruit Broker จะต้องมีข้อมูลที่จำเป็น คือ URL ของเซิร์ฟเวอร์ ชื่อผู้ใช้ รหัสผ่าน และพอร์ตในการสื่อสาร ดังรูปที่ 4.103 เป็นข้อมูลสำคัญที่ได้สมัครไว้กับ Adafruit Broker ไว้เรียบร้อยแล้ว

```
// MQTT Broker
const char *mqttServer = "io.adafruit.com";
const int mqttPort = 1883;
const char *mqttUser = "iot
const char *mqttPassword = "ai                    XQu6YYyjZ81b";
const char *Topic = "iotp                s/testweb";
```

รูปที่ 4.103 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker

จากคำสั่งโปรแกรมดังรูปที่ 4.104 client.setServer() จะถูกใช้เพื่อกำหนดค่าเซิร์ฟเวอร์ MQTT Broker โดยระบุ URL และพอร์ตที่ใช้งาน รวมถึงตั้งค่า callback function (callback) เพื่อจัดการกับข้อความที่ได้รับจาก Topic ที่ได้ Subscribe ไว้ ในขั้นตอนการเชื่อมต่อกับ MQTT Broker มีการใช้ while loop เพื่อตรวจสอบว่าการเชื่อมต่อสำเร็จหรือไม่ และพยายามทำการเชื่อมต่อจนกว่าจะเชื่อมต่อได้สำเร็จ เมื่อการเชื่อมต่อสำเร็จจะแสดงข้อความใน Serial Monitor ดังรูปที่ 4.105 เพื่อบอกสถานะการเชื่อมต่อ หากการเชื่อมต่อสำเร็จจะมีการแสดงข้อความว่า "Adafruit Broker MQTT Connected"

```
client.setServer(mqttServer, mqttPort);
client.setCallback(callback);
Serial.println("Connecting to MQTT...");
while (!client.connected()) {
    String clientId = "ESP32Client-" + String(random(0xffff), HEX);
    if (client.connect(clientId.c_str(), mqttUser, mqttPassword)) {

        Serial.println("Adafruit Broker MQTT Connected");
    } else {
        Serial.print("Failed, State: ");
        Serial.println(client.state());
        delay(2000);
    }
}
```

รูปที่ 4.104 คำสั่งโปรแกรมเชื่อมต่อบอร์ดไมโครคอนโทรลเลอร์ ESP32 กับ HiveMQ Broker

```
17:10:38.002 -> Connecting to WiFi.............
17:10:44.530 -> WiFi connected
17:10:44.530 -> Connecting to MQTT...
17:10:45.901 -> Adrafruit Broker MQTT Connected
```

รูปที่ 4.105 ผลการเชื่อมต่อผ่าน Serial Monitor

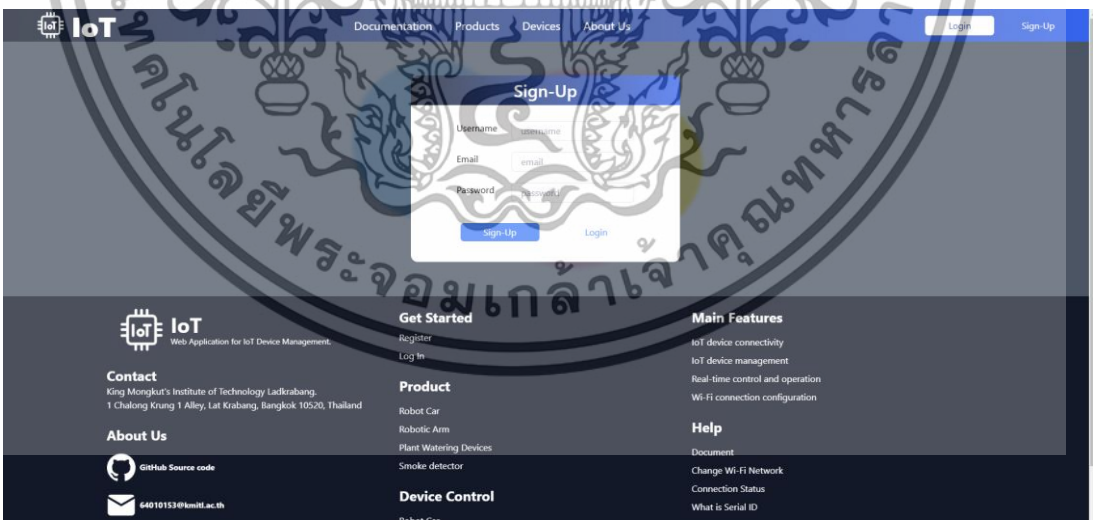
4.3 ผลการทำงานของเว็บแอปพลิเคชัน

ทดสอบโดยแบ่งการทำงานของเว็บแอปพลิเคชันเป็น 2 ส่วน คือ การทดสอบการทำงานหน้าเว็บแอปพลิเคชัน (Front-End) และทดสอบระบบจัดการเว็บไซต์ (Back-End) โดยมีกระบวนการทำงานในมุมของผู้ใช้ดังรูปที่

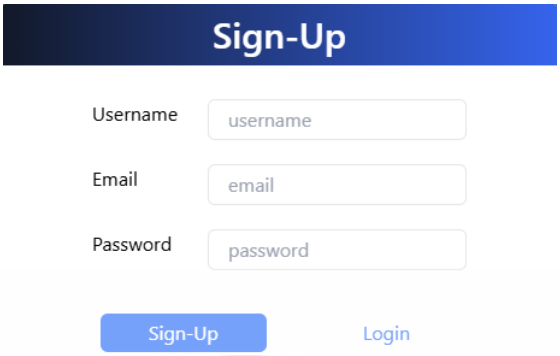
4.3.1 หน้าเว็บแอปพลิเคชัน (Front-End)

4.3.1.1 เว็บแอปพลิเคชันหน้า Sign up

เมื่อทดสอบให้ผู้ใช้สมัครบัญชีเพื่อใช้งานเว็บแอปพลิเคชัน เพื่อจะได้ลงชื่อเข้าใช้งานเว็บแอปพลิเคชัน ดังรูปที่ 4.106 โดยที่หน้า Sign up จะมีกล่องข้อมูลให้ผู้ใช้กรอกข้อมูลเพื่อใช้ในการสมัครบัญชีผู้ใช้ที่จำเป็น คือ Username Email และ Password ดังรูปที่ 4.107 เมื่อผู้ใช้กรอกข้อมูลเรียบร้อยแล้วกดปุ่ม Sign up และมีกล่องโต้ตอบขึ้นมาดังรูปที่ 4.108 เป็นการยืนยันว่าผู้ใช้ได้สมัครบัญชีผู้ใช้เรียบร้อยแล้ว



รูปที่ 4.106 เว็บแอปพลิเคชันหน้า Sign up

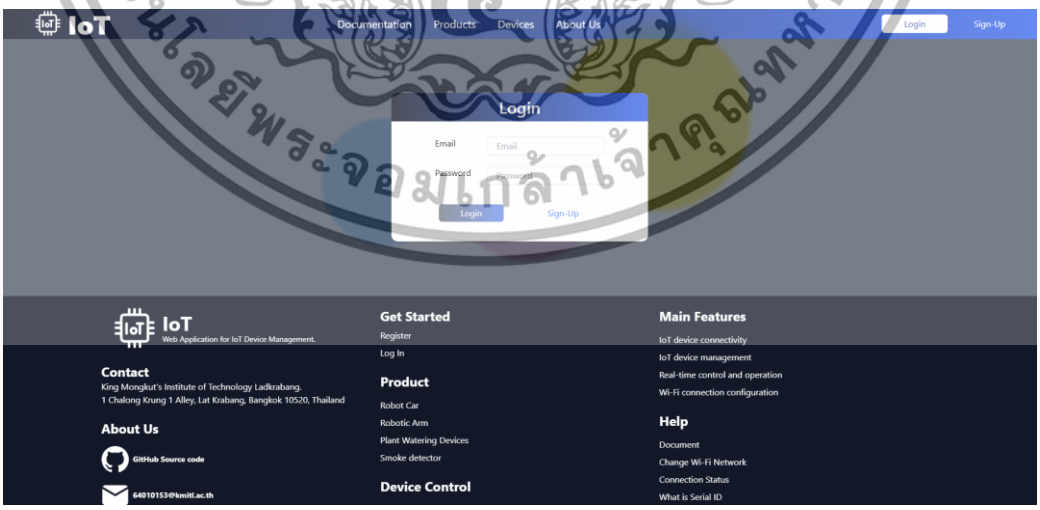


รูปที่ 4.107 กล่องข้อมูล Sign up

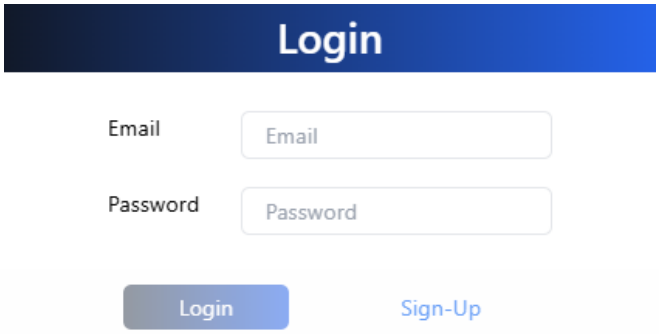
รูปที่ 4.108 กล่องโต้ตอบเมื่อสมัครบัญชีผู้ใช้

4.3.1.2 เว็บแอปพลิเคชันหน้า Login

เมื่อทดสอบลงชื่อเข้าใช้งานเว็บแอปพลิเคชัน จากที่ได้ทำการสมัครบัญชีผู้ใช้เรียบร้อยแล้วให้ผู้ใช้เข้าเว็บแอปพลิเคชันหน้า Login ดังรูปที่ 4.109 จะมีกล่องข้อมูลให้ผู้ใช้กรอกข้อมูลเพื่อลงชื่อใช้งานเว็บแอปพลิเคชัน คือ Email และ Password ดังรูปที่ 4.110 เมื่อผู้ใช้กรอกข้อมูลเรียบร้อยแล้วกดปุ่ม Login และมีกล่องโต้ตอบขึ้นมาดังรูปที่ 4.111 เป็นการยืนยันว่าผู้ใช้ได้ลงชื่อเข้าใช้เว็บแอปพลิเคชันเรียบร้อยแล้ว



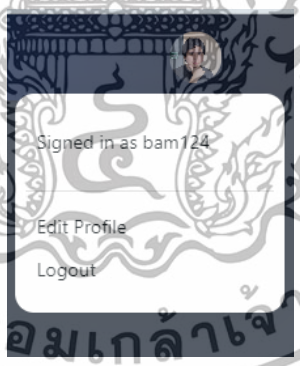
รูปที่ 4.109 เว็บแอปพลิเคชันหน้า Login



รูปที่ 4.110 กล่องข้อมูล Login

รูปที่ 4.111 กล่องโต้ตอบเมื่อลงชื่อเข้าใช้

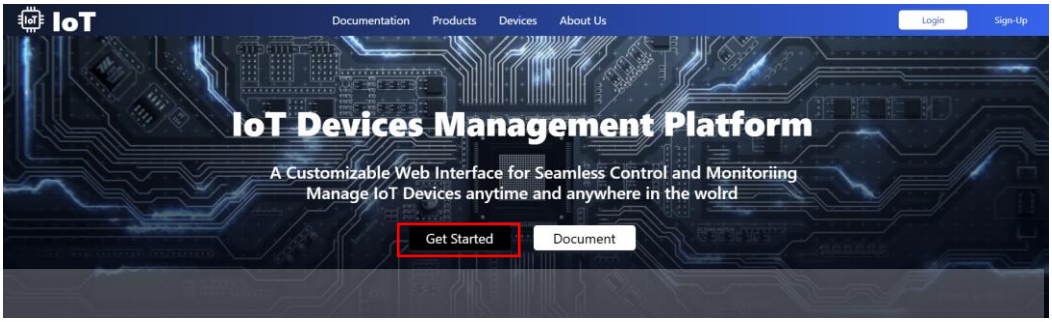
เมื่อลงชื่อเข้าใช้เว็บแอปพลิเคชันเรียบร้อยแล้ว เว็บแอปพลิเคชันจะเข้าสู่หน้าหลักของเว็บแอปพลิเคชันและจะลงชื่อเข้าใช้ด้วยบัญชีที่ผู้ใช้งานลงชื่อเข้าใช้ ดังรูปที่ 4.112



รูปที่ 4.112 ข้อมูลบัญชีผู้ใช้ที่ได้ลงชื่อเข้าใช้งานเว็บแอปพลิเคชัน

4.3.1.3 เว็บแอปพลิเคชันหน้าหลัก

เมื่อเปิดเว็บแอปพลิเคชันขึ้นมาผ่านเบราว์เซอร์ จะเจอหน้าหลักของเว็บแอปพลิเคชัน ดังรูปที่ 4.113 โดยมีปุ่ม GetStart ในการเริ่มการทำงาน โดยผู้ใช้งานจะต้องทำการลงชื่อเข้าใช้ผ่านหน้าเว็บแอปพลิเคชัน Login ก่อนจึงจะสามารถใช้งานภายในเว็บแอปพลิเคชัน

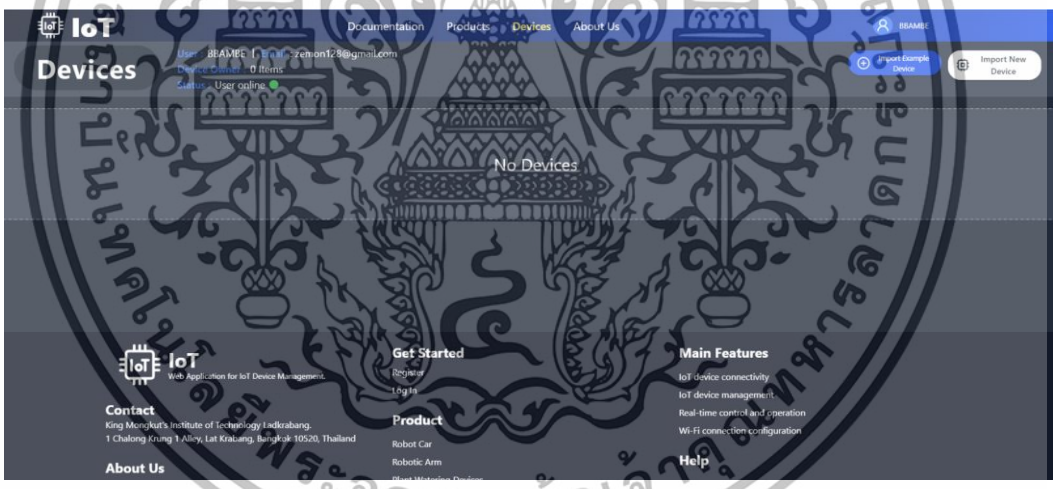


รูปที่ 4.113 เว็บแอปพลิเคชันหน้าหลัก

4.3.1.4 เว็บแอปพลิเคชันหน้า Device

1) เว็บแอปพลิเคชันหน้า Device เมื่อไม่มีการนำเข้าอุปกรณ์ตัวอย่าง

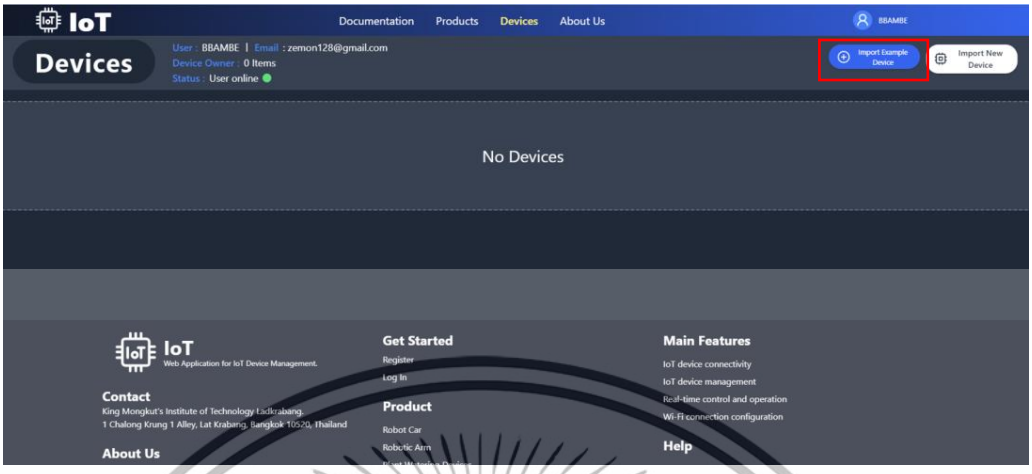
เมื่อทดสอบเข้าใช้งานหน้าเว็บแอปพลิเคชันหน้า Device หากผู้ใช้งานไม่มีการนำเข้าอุปกรณ์ตัวอย่าง หน้าเว็บแอปพลิเคชันจะแสดงว่าบัญชีของผู้ใช้งานนี้ยังไม่มีอุปกรณ์อยู่ในฐานข้อมูล ดังรูปที่ 4.114



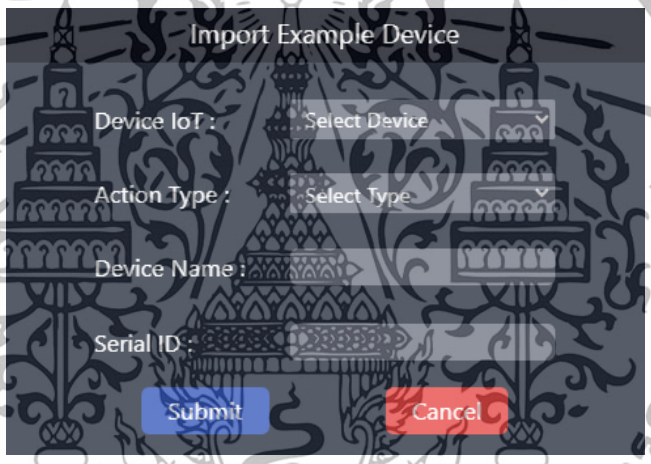
รูปที่ 4.114 เว็บแอปพลิเคชันหน้า Device เมื่อไม่มีการนำเข้าอุปกรณ์ตัวอย่าง

2) เว็บแอปพลิเคชันหน้า Device เมื่อนำเข้าอุปกรณ์ตัวอย่าง

เมื่อทดสอบเข้าใช้งานหน้าเว็บแอปพลิเคชันหน้า Device โดยที่ผู้ใช้งานนำเข้าอุปกรณ์ตัวอย่าง ผ่านการกดปุ่ม Import Example Device ดังรูปที่ 4.115 จะปรากฏกล่องข้อความเพื่อรับข้อมูลของอุปกรณ์ คือ ประเภทของอุปกรณ์ ได้แก่ Robot Car Robotic Arm Pump และ Smoke Detector ดังรูปที่ 4.115 ชื่ออุปกรณ์ (Device Name) และ Serial ID ของอุปกรณ์

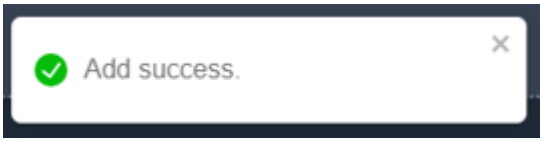


รูปที่ 4.115 ปุ่ม Import Example Device ในเว็บแอปพลิเคชันหน้า Device

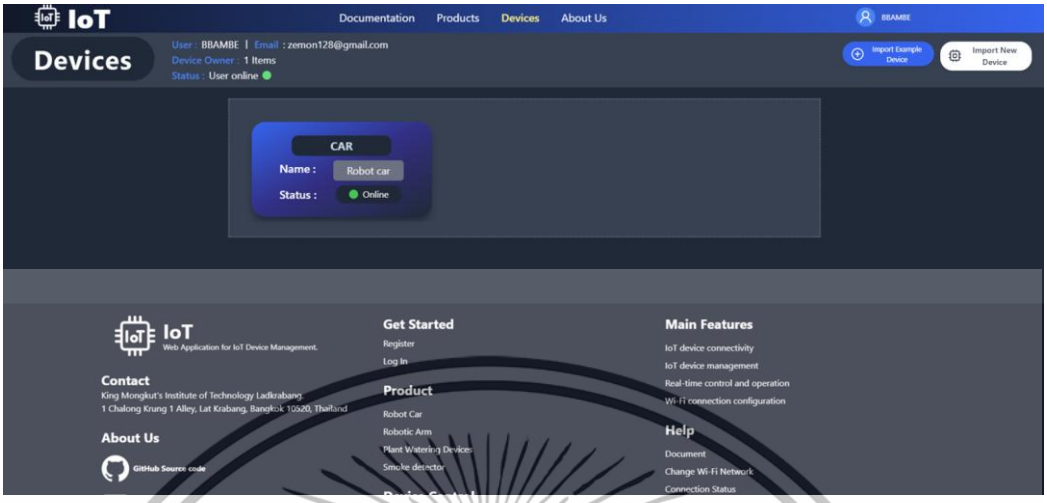


รูปที่ 4.116 กล่องข้อความในการนำเข้าอุปกรณ์ตัวอย่าง

เมื่อกรอกข้อมูลใสกล่องข้อมูลนำเข้าอุปกรณ์ตัวอย่างแล้วกดปุ่ม Submit เพื่อทำการเพิ่มอุปกรณ์ตามที่ผู้ใช้งานกรอก กล่องโต้ตอบจะปรากฏขึ้นมาเป็นการยืนยันนำเข้าอุปกรณ์ตัวอย่างสำเร็จ ดังรูปที่ 4.117 และเมื่อระบบได้รับข้อมูลการเพิ่มอุปกรณ์แล้ว ในส่วนเว็บแอปพลิเคชันหน้า Device เมื่อเพิ่มอุปกรณ์จะแสดงข้อมูลอุปกรณ์ และจำนวนอุปกรณ์ของผู้ใช้ ดังรูปที่ 4.118



รูปที่ 4.117 กล่องโต้ตอบเมื่อเพิ่มอุปกรณ์



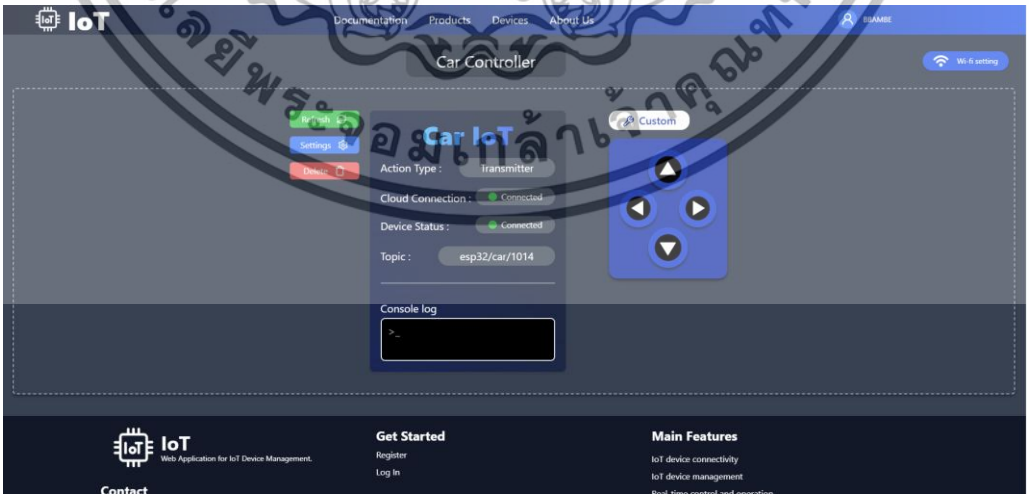
รูปที่ 4.118 เว็บแอปพลิเคชันหน้า Device เมื่อนำเข้าอุปกรณ์ตัวอย่าง

4.3.1.5 เว็บแอปพลิเคชันหน้า Import Example Device

จากการออกแบบตามอุปกรณ์ในระบบ โดยแสดงผลหน้าในการควบคุมอุปกรณ์และการทำงานของอุปกรณ์ มีอุปกรณ์ทั้งหมด 3 ชนิด ได้แก่

1) เว็บแอปพลิเคชันหน้า Robot Car

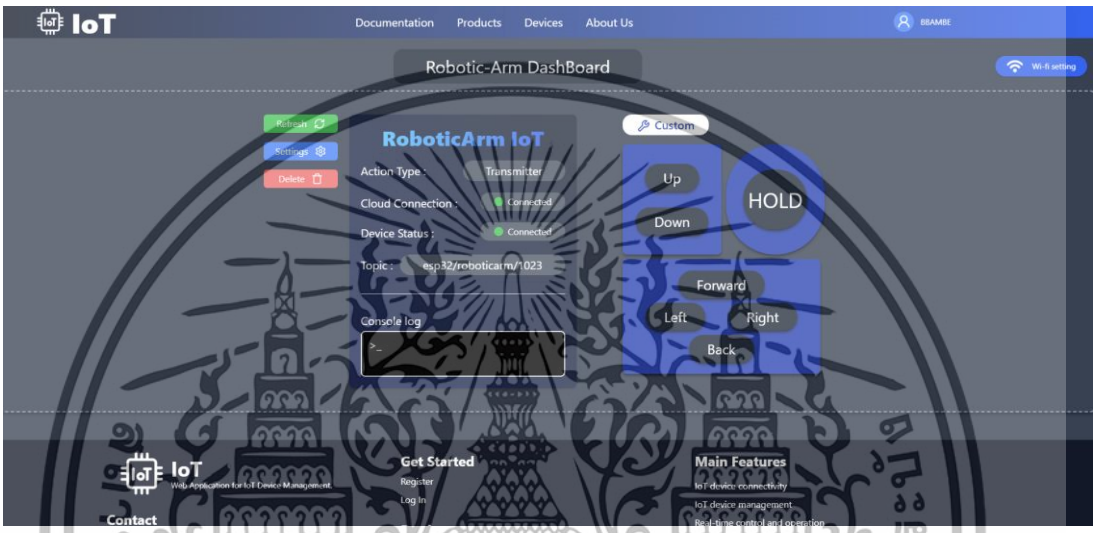
เว็บแอปพลิเคชันหน้า Car เป็นหน้าแสดงการควบคุมรถยนต์บังคับให้ไปในทิศทางที่ต้องการ แสดงสถานะเชื่อมต่อกับ Cloud Broker สถานะการเชื่อมต่อระหว่างรถยนต์บังคับกับเว็บแอปพลิเคชัน Topic ที่ใช้สื่อสาร และบันทึกตามการทำงานของอุปกรณ์ ดังรูปที่ 4.119



รูปที่ 4.119 เว็บแอปพลิเคชันหน้า Car

2) เว็บแอปพลิเคชันหน้า Robotic Arm

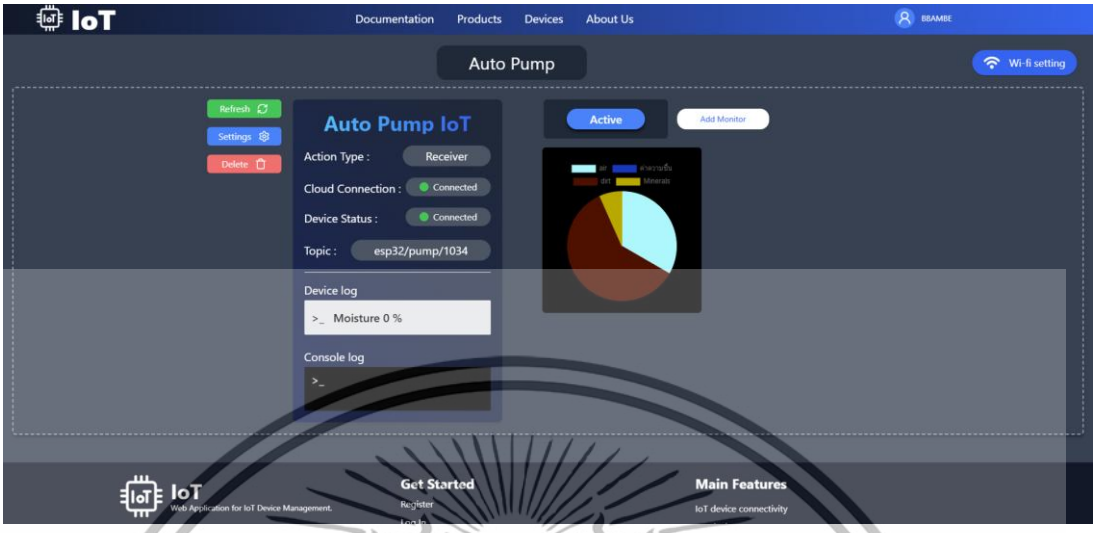
เว็บแอปพลิเคชันหน้า Robotic Arm เป็นหน้าแสดงการควบคุมแขนกลให้ไปในทิศทางที่ต้องการ แสดงสถานะเชื่อมต่อกับ Cloud Broker สถานะการเชื่อมต่อระหว่างแขนกลกับเว็บแอปพลิเคชัน Topic ที่ใช้สื่อสาร และบันทึกตามการทำงานของอุปกรณ์ ดังรูปที่ 4.120



รูปที่ 4.120 เว็บแอปพลิเคชันหน้า Robotic Arm

3) เว็บแอปพลิเคชันหน้า Pump

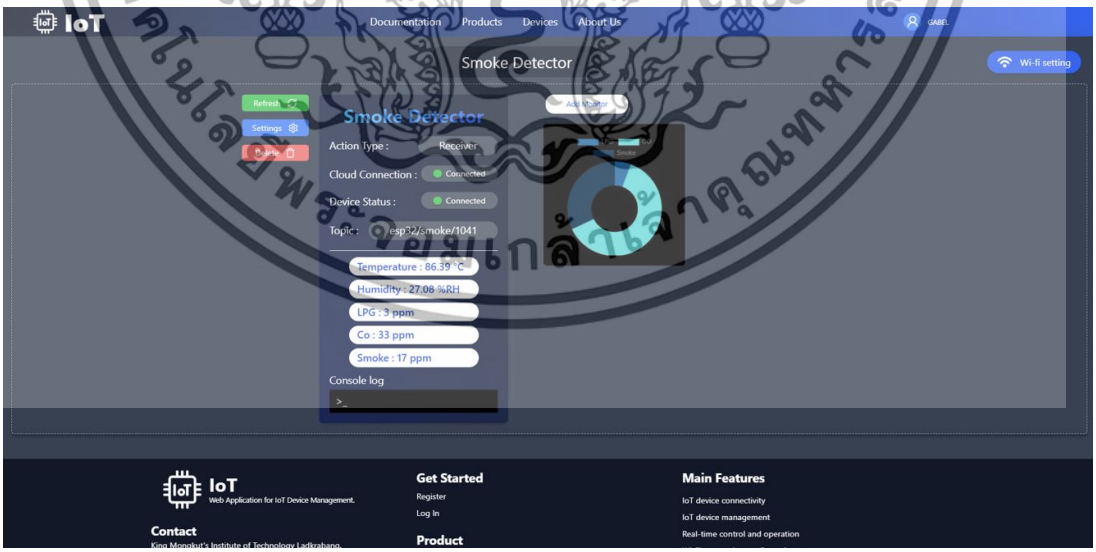
เว็บแอปพลิเคชันหน้า Pump เป็นหน้าแสดงการควบคุมอุปกรณ์รดน้ำต้นไม้ให้ไปในรดน้ำต้นไม้ตามที่ต้องการ แสดงสถานะเชื่อมต่อกับ Cloud Broker สถานะการเชื่อมต่อระหว่างอุปกรณ์รดน้ำต้นไม้กับเว็บแอปพลิเคชัน Topic ที่ใช้สื่อสาร และบันทึกตามการทำงานของอุปกรณ์ ดังรูปที่ 4.121



รูปที่ 4.121 เว็บแอปพลิเคชันหน้า Pump

4) เว็บแอปพลิเคชันหน้า Smoke Detector

เว็บแอปพลิเคชันหน้า Smoke Detector เป็นหน้าแสดงการผลการทำงานของอุปกรณ์ได้ตรวจจับอุณหภูมิ ความชื้นสัมพัทธ์ และการตรวจจับแก๊ส ได้แก่ แก๊ส LPG คาร์บอนไดออกไซด์ และค่าควันไฟที่เกิดจากการเผาไหม้ โดยแสดงสถานะเชื่อมต่อกับ Cloud Broker สถานะการเชื่อมต่อระหว่างอุปกรณ์ตรวจจับควันไฟกับเว็บแอปพลิเคชัน Topic ที่ใช้สื่อสารและบันทึกตามการทำงานของอุปกรณ์ ดังรูปที่ 4.122

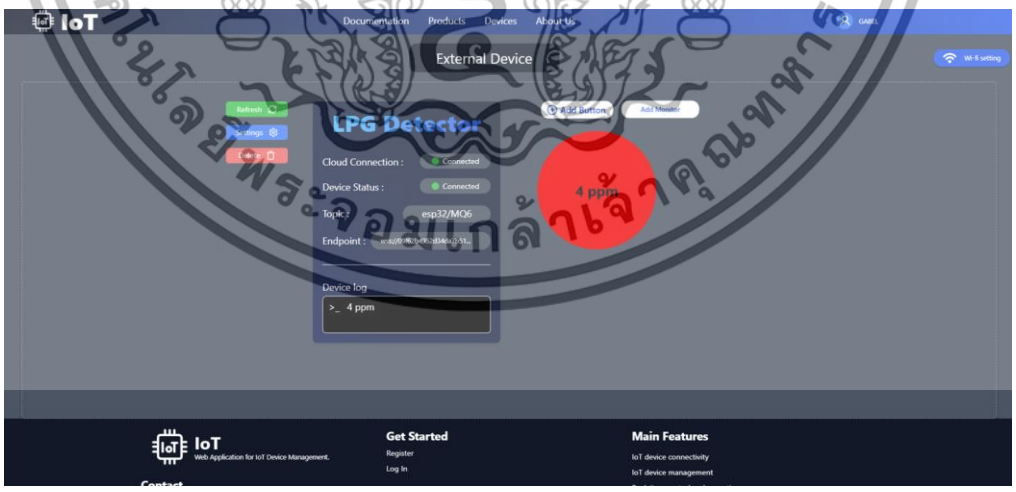


รูปที่ 4.122 เว็บแอปพลิเคชันหน้า Smoke Detector

4.3.1.6 เว็บแอปพลิเคชันหน้า Import New Device

เมื่อทดสอบนำเข้าอุปกรณ์ภายนอกตัวอย่างเข้ามาภายในแพลตฟอร์ม เมื่อกดเข้าดูหน้าอุปกรณ์ จะปรากฏข้อมูลอุปกรณ์ ประกอบไปด้วยสถานะเชื่อมต่อกับ Cloud Broker สถานการณ์เชื่อมต่อระหว่าเว็บแอปพลิเคชันกับอุปกรณ์ IoT Topic ในการสื่อสาร ที่อยู่ Broker และ Console log การทำงานของอุปกรณ์ นอกจากนี้ยังสามารถปรับแต่งปุ่มสั่งการและหน้าแสดงผลการทำงานของอุปกรณ์ได้เหมือนกับอุปกรณ์ตัวอย่างภายในแพลตฟอร์มทุกประการ ดังรูปที่ 4.123 และอุปกรณ์รับข้อมูลตัวอย่าง ดังรูปที่ 4.124

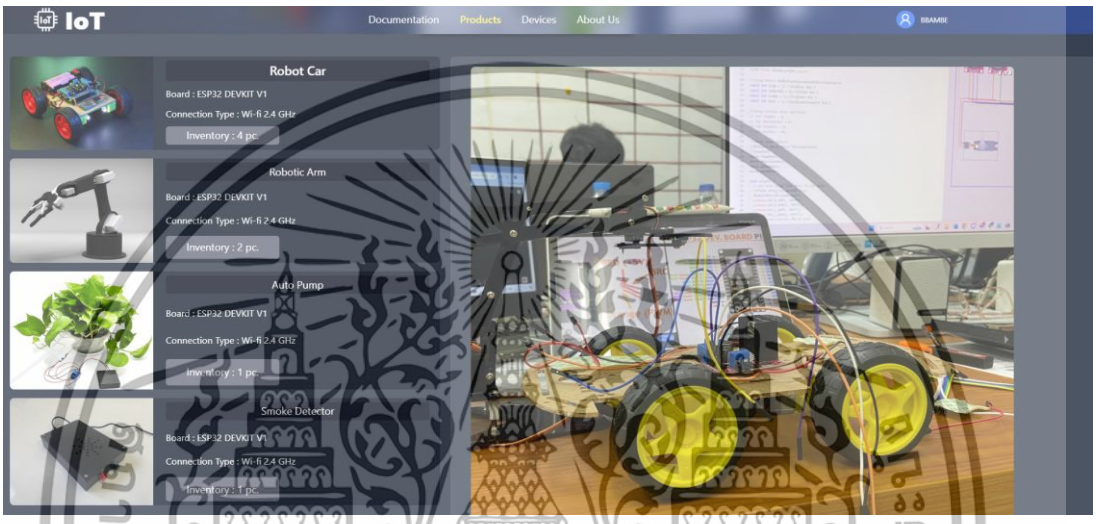
รูปที่ 4.123 เว็บแอปพลิเคชันหน้า Import New Device ของตัวอย่างอุปกรณ์ควบคุมใด ๆ



รูปที่ 4.124 เว็บแอปพลิเคชันหน้า Import New Device ของตัวอย่างอุปกรณ์รับข้อมูลใด ๆ

4.3.1.7 เว็บแอปพลิเคชันหน้า Products

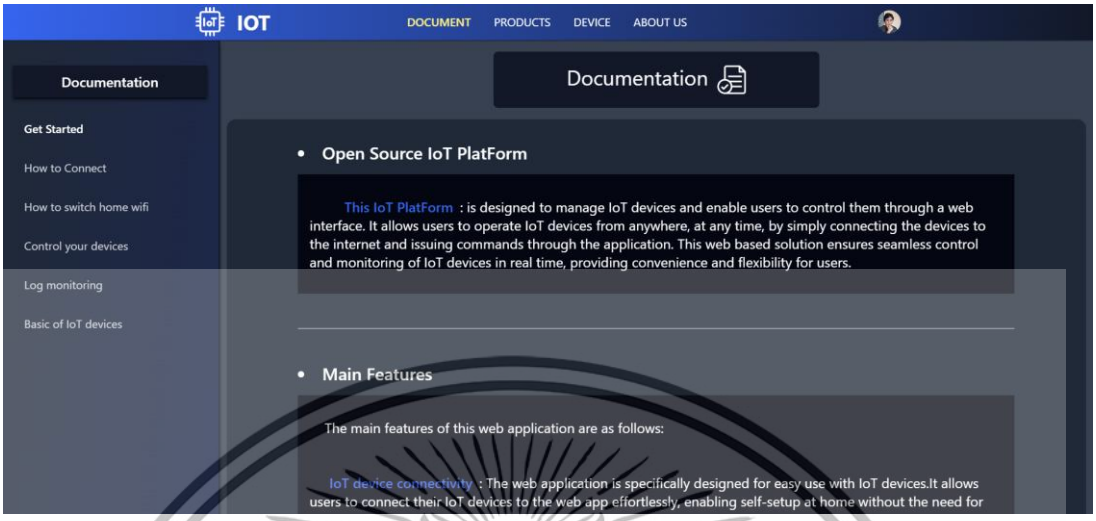
เมื่อทดสอบเข้าใช้งานหน้าเว็บแอปพลิเคชันหน้า Products โดยจะแสดงข้อมูลของอุปกรณ์ทั้งหมดที่มีในระบบบานข้อมูล และจะนำมาแสดงผลจำนวนอุปกรณ์ตามประเภทอุปกรณ์ที่มีในระบบ คือ ประเภทของอุปกรณ์ ได้แก่ Robot Car Robotic Arm Pump และ Smoke Detector ดังรูปที่ 4.125



รูปที่ 4.125 เว็บแอปพลิเคชันหน้า Product

4.3.1.8 เว็บแอปพลิเคชันหน้า Documentation

เมื่อทดสอบเข้าใช้งานหน้าเว็บแอปพลิเคชันหน้า Documentation หรือเป็นหน้าคู่มือการใช้งานของเว็บแอปพลิเคชันร่วมกับอุปกรณ์ โดยจะมีเนื้อการเริ่มต้นใช้งาน วิธีการเชื่อมต่ออุปกรณ์เข้ากับเว็บแอปพลิเคชัน การเปลี่ยนการเชื่อมต่อ Wi-Fi การแสดงผลที่ได้จากอุปกรณ์ การนำเข้าอุปกรณ์จากภายนอก และคำสั่งโปรแกรมที่จะทำให้อุปกรณ์ใช้งานร่วมกันกับแพลตฟอร์มได้ รูปที่ 4.126



รูปที่ 4.126 เว็บแอปพลิเคชันหน้า Documentation

4.3.2 ระบบจัดการเว็บแอปพลิเคชัน (Back-End)

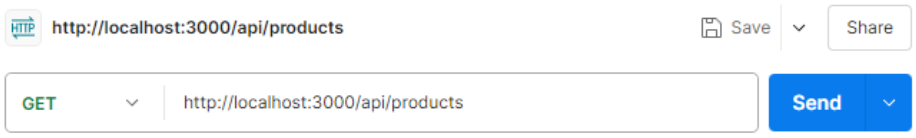
ผลการทดสอบระบบจัดการเว็บแอปพลิเคชันโดยการดึงข้อมูลจาก API ที่ออกแบบภายในเว็บแอปพลิเคชันในระบบการทำงานต่าง ๆ ดังนี้

4.3.2.1 API ชื่อ Products

API สำหรับการจัดการผลิตภัณฑ์ (Products) จะถูกแบ่งออกเป็น 3 ส่วน ได้แก่ GET POST และ PUT

1) GET

ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลสินค้ามาแสดงบนหน้าเว็บแอปพลิเคชัน ดังรูป 4.127 ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ถูกดึงออกมานั้นตรงตามที่จัดเก็บไว้ในฐานข้อมูล โดยข้อมูลดังกล่าวจะถูกจัดเก็บในรูปแบบ JSON ซึ่งเป็นรูปแบบข้อมูลที่อ่านและทำความเข้าใจได้ง่าย เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของสินค้าที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman ดังรูป 4.128

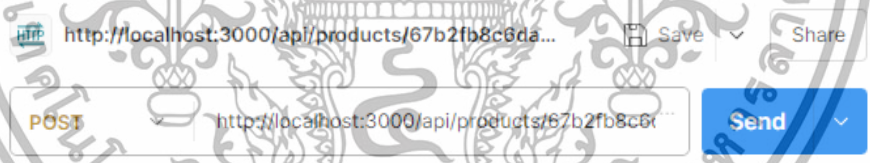


รูปที่ 4.127 ทดสอบ Method GET ผ่าน Postman

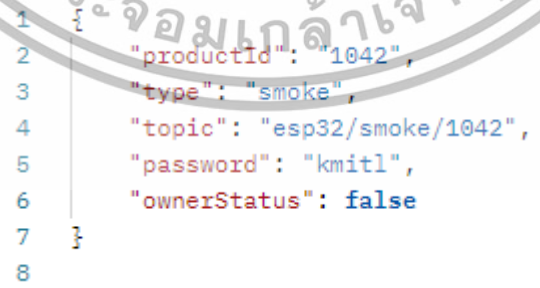
รูปที่ 4.128 ผลการทดสอบเมื่อใช้ Method GET

2) POST

ผลจากการทดสอบด้วยการใช้ Method POST ดังรูป 4.129 เพื่อส่งข้อมูลของสินค้าใหม่ (Product) ไปเก็บในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่ส่งผ่านไปในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังรูป 4.130



รูปที่ 4.129 ทดสอบ Method POST ผ่าน Postman

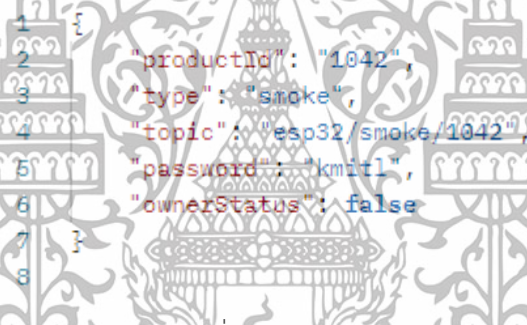


รูปที่ 4.130 ผลการทดสอบเมื่อใช้ Method POST

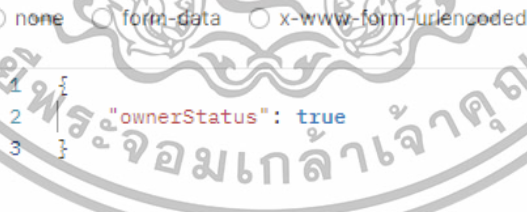
3) PUT

ผลจากการทดสอบด้วยการใช้ Method PUT ดังรูป 4.131 เพื่ออัปเดตข้อมูลที่มีอยู่แล้วใน API ของสินค้า (Product) ในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่อัปเดตในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) บอกเซิร์ฟเวอร์ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังรูป 4.134 มีข้อมูลเดิมจากรูปที่ 4.132 โดยใช้ข้อมูล PUT เข้าฐานข้อมูล ดังรูปที่ 4.133

รูปที่ 4.131 ทดสอบ Method PUT ผ่าน Postman



รูปที่ 4.132 ข้อมูลเดิม



รูปที่ 4.133 ข้อความอัปเดตเพื่อ PUT เข้าฐานข้อมูล

```
{
    "_id": "67b2fb8c6da497a450d18dc9",
    "productId": "1042",
    "type": "smoke",
    "topic": "esp32/smoke/1042",
    "password": "kmitl",
    "ownerStatus": true,
    "createdAt": "2025-02-17T09:04:12.124Z",
    "updatedAt": "2025-02-17T09:14:00.508Z",
    "__v": 0
},
```


รูปที่ 4.134 ผลการทดสอบเมื่อใช้ Method POST

4.3.2.2 API ชื่อ Device

API สำหรับการจัดการอุปกรณ์ (Device) จะถูกแบ่งออกเป็น 4 ส่วน ได้แก่ GET POST PUT และ GET{id}

1) GET

ผลจากการทดสอบด้วยการใช้ Method GET ดังรูปที่ 4.135 เพื่อดึงข้อมูล อุปกรณ์มาแสดงบนหน้าเว็บแอปพลิเคชัน ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ถูกดึงออกมานั้นตรงตามที่จัดเก็บไว้ในฐานข้อมูล โดยข้อมูลดังกล่าวจะถูกจัดเก็บในรูปแบบ JSON (JavaScript Object Notation) ซึ่งเป็นรูปแบบข้อมูลที่อ่านและทำความเข้าใจได้ง่าย เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของอุปกรณ์ที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman ดังตัวอย่างในรูปที่ 4.136

รูปที่ 4.135 ทดสอบ Method GET ด้วย ID ผ่าน Postman

```
{
    "_id": "67b2fdd86da497a450d18dd5",
    "deviceId": "872619f1-83d8-4751-8443-6d3c4f054d08",
    "userId": "67a6fdbeb36635e08406f7a5",
    "name": "Smoke detector",
    "topic": "esp32/smoke/1042",
    "type": "smoke",
    "password": "kmitl",
    "status": "owner",
    "wifiId": "cad9fcfc-9a9c-4f84-a863-4bc16440cb1f",
    "wifiConnect": "none",
    "productId": "1042",
    "createdAt": "2025-02-17T09:14:00.348Z",
    "updatedAt": "2025-02-17T09:14:00.348Z",
    "__v": 0
}
```


รูปที่ 4.136 ผลการทดสอบเมื่อใช้ Method GET

2) POST

ผลจากการทดสอบด้วยการใช้ Method POST ดังรูป 4.137 เพื่อส่งข้อมูลของอุปกรณ์ใหม่ (Device) ไปเก็บในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่ส่งผ่านไปในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) ดังรูปที่ 4.138 ซึ่งมีข้อมูลที่ต้องการเพิ่มลงในตารางหรือเอกสารของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.139

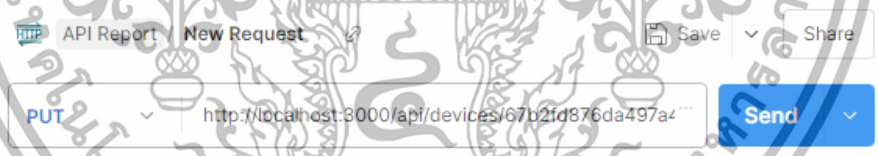
รูปที่ 4.137 ทดสอบ Method POST ผ่าน Postman

รูปที่ 4.138 ข้อมูล JSON ที่ POST

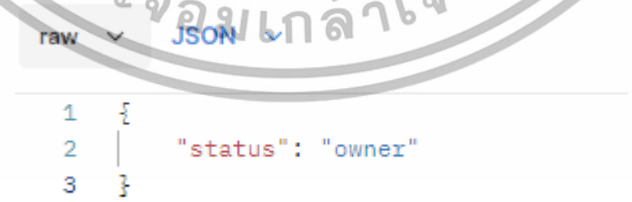
รูปที่ 4.139 ผลการทดสอบเมื่อใช้ Method POST

3) PUT

ผลจากการทดสอบด้วยการใช้ Method PUT ดังรูป 4.140 เพื่ออัปเดตข้อมูลที่มีอยู่แล้วใน API ของอุปกรณ์ (Device) ในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่อัปเดตในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) บอกเซิร์ฟเวอร์ ดังรูปที่ 4.141 ซึ่งมีข้อมูลที่ต้องการอัปเดตลงในตารางหรือเอกสารของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.143 จากข้อมูลเดิมดังรูปที่ 4.142



รูปที่ 4.140 ทดสอบ Method PUT ผ่าน Postman



รูปที่ 4.141 ข้อมูล JSON ที่ PUT

```
{
    "_id": "67b2fd876da497a450d18dcf",
    "deviceId": "4c8863d8-0cf9-401c-ba1d-7ae71f852505",
    "userId": "67a6fdbeb36635e08406f7a5",
    "name": "Robotic arm",
    "topic": "esp32/roboticarm/1023",
    "type": "roboticarm",
    "password": "kmitl",
    "status": "none",
    "wifiId": "25835431-c341-4c03-9d5b-ab8f42ca559c",
    "wifiConnect": "none",
    "productId": "1023",
    "createdAt": "2025-02-17T09:12:39.959Z",
    "updatedAt": "2025-02-17T09:12:39.959Z",
    "__v": 0
}
```

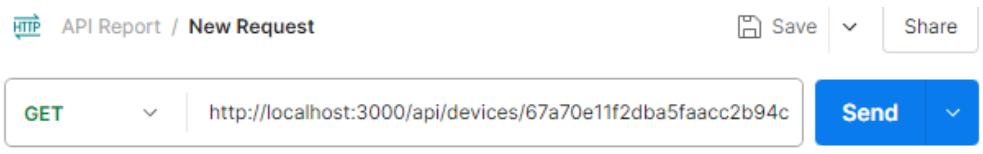

รูปที่ 4.142 ข้อมูลเดิม


```
{
    "_id": "67b2fd876da497a450d18dcf",
    "deviceId": "4c8863d8-0cf9-401c-ba1d-7ae71f852505",
    "userId": "67a6fdbeb36635e08406f7a5",
    "name": "Robotic arm",
    "topic": "esp32/roboticarm/1023",
    "type": "roboticarm",
    "password": "kmitl",
    "status": "owner",
    "wifiId": "25835431-c341-4c03-9d5b-ab8f42ca559c",
    "wifiConnect": "none",
    "productId": "1023",
    "createdAt": "2025-02-17T09:12:39.959Z",
    "updatedAt": "2025-02-17T09:15:39.959Z",
    "__v": 0
}
```


รูปที่ 4.143 ผลการทดสอบเมื่อใช้ Method PUT

4) GET{id}

ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลอุปกรณ์เฉพาะเจาะจง ดังรูป 4.144 เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของอุปกรณ์ที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.145



รูปที่ 4.144 ทดสอบ Method GET ด้วย ID ผ่าน Postman

```
{
    "_id": "67a70e11f2dba5faacc2b94c",
    "deviceId": "d098ba56-b654-4eca-82f1-8db607618d8b",
    "userId": "67a6fdbeb36635e08406f7a5",
    "name": "watering pot",
    "topic": "esp32/pump/1033",
    "type": "pump",
    "password": "kmitl",
    "status": "owner",
    "wifiId": "bf9661d8-ed65-4fed-899c-b43a8d4183f1",
    "wifiConnect": "none",
    "productId": "1033",
    "createdAt": "2025-02-08T07:56:01.699Z",
    "updatedAt": "2025-02-08T07:56:01.699Z",
    "__v": 0
}
```

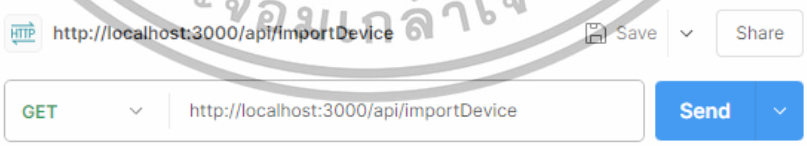

รูปที่ 4.145 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID

4.3.2.3 API ชื่อ Import device

API สำหรับการจัดการอุปกรณ์จากภายนอก (Import New Device) จะถูกแบ่งออกเป็น 4 ส่วน ได้แก่ GET POST DELETE และ GET{userId}

1) GET

ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลอุปกรณ์จากภายนอกแสดงบนหน้าเว็บแอปพลิเคชัน ดังรูป 4.146 ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ถูกดึงออกมานั้นตรงตามที่จัดเก็บไว้ในฐานข้อมูล โดยข้อมูลดังกล่าวจะถูกจัดเก็บในรูปแบบ JSON ซึ่งเป็นรูปแบบข้อมูลที่อ่านและทำความเข้าใจได้ง่าย เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของอุปกรณ์จากภายนอกที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman โดยมีข้อมูลต่าง ๆ ที่ได้ออกแบบไว้ในตาราง Products ดังรูป 4.147



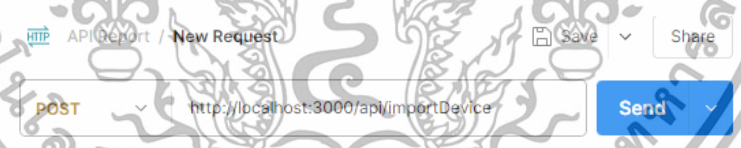
รูปที่ 4.146 ทดสอบ Method GET ผ่าน Postman

```
{
    "_id": "67ad93a6efe2a598694b6231",
    "deviceId": "098cddf7-5b77-4515-ac7f-a51c9e400545",
    "userId": "[object Object]",
    "name": "LED Control",
    "topic": "customled",
    "broker": "hivemq",
    "connectPath": '                                    e5ee186.
        s1.eu.hivemq
    "username": "adm
    "password": "Iot
    "status": "",
    "wifiId": "dba00b49-334f-48e9-a31d-dc3f82b09fb7",
    "wifiConnect": "none",
    "createdAt": "2025-02-13T06:39:34.172Z",
    "updatedAt": "2025-02-13T06:39:34.172Z",
    "__v": 0
}
```


รูปที่ 4.147 ผลการทดสอบเมื่อใช้ Method GET

2) POST

ผลจากการทดสอบด้วยการใช้ Method POST ดังรูป 4.148 เพื่อส่งข้อมูลของอุปกรณ์ภายนอกใหม่ไปเก็บในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่ส่งผ่านไปในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) ดังรูปที่ 4.149 ซึ่งมีข้อมูลที่ต้องการเพิ่มลงในตารางหรือเอกสารของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.150



รูปที่ 4.148 ทดสอบ Method POST ผ่าน Postman

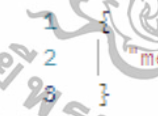
รูปที่ 4.149 ข้อมูล JSON ที่ POST

รูปที่ 4.150 ผลการทดสอบเมื่อใช้ Method POST

3) DELETE

ผลจากการทดสอบด้วยการใช้ Method DELETE ดังรูป 4.151 เพื่อส่งคำสั่งลบข้อมูลของอุปกรณ์ภายนอกไปยังฐานข้อมูลที่เชื่อมต่อกับระบบ โดยจะทำการส่งคำขอเพื่อลบข้อมูลของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.152 เมื่อข้อมูลถูกลบสำเร็จ

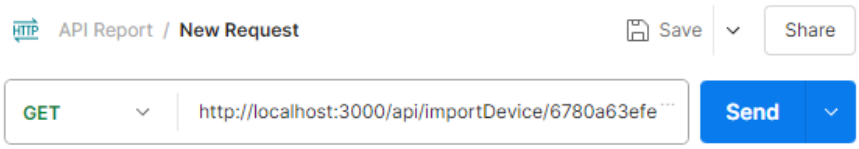
รูปที่ 4.151 ทดสอบ Method DELETE ผ่าน Postman



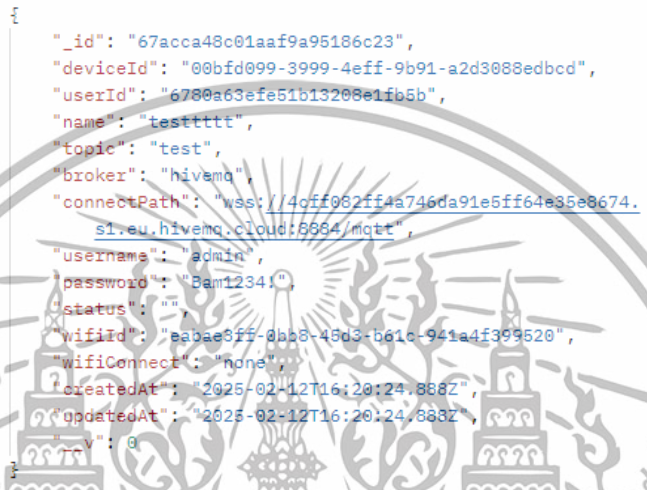
รูปที่ 4.152 ผลการทดสอบเมื่อใช้ Method DELETE

4) GET{id}

ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลอุปกรณ์เฉพาะเจาะจง ดังรูป 4.153 เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบ ข้อมูลทั้งหมดของอุปกรณ์ที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.154



รูปที่ 4.153 ทดสอบ Method GET ด้วย ID ผ่าน Postman



รูปที่ 4.154 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID

4.3.2.4 API ชื่อ Wi-Fi

API สำหรับการจัดการเครือข่าย Wi-Fi (Wi-Fi) จะถูกแบ่งออกเป็น 4 ส่วน ได้แก่ GET POST PUT และ GET{id}

1) GET

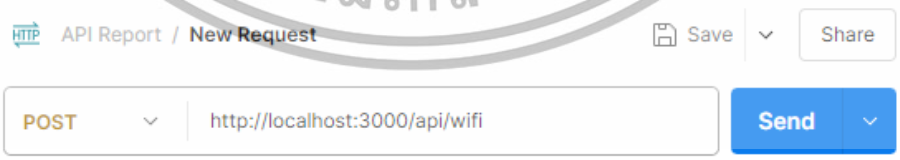
ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลเครือข่าย Wi-Fi ของอุปกรณ์ แสดงบนหน้าเว็บแอปพลิเคชัน ดังรูป 4.155 ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ถูกดึงออกมานั้นตรงตามที่จัดเก็บไว้ในฐานข้อมูล โดยข้อมูลดังกล่าวจะถูกจัดเก็บในรูปแบบ JSON ซึ่งเป็นรูปแบบข้อมูลที่อ่านและทำความเข้าใจได้ง่าย เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของเครือข่าย Wi-Fi ของอุปกรณ์ที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman โดยมีข้อมูลต่าง ๆ ที่ได้ออกแบบไว้ในตาราง Products ดังรูป 4.156

รูปที่ 4.155 ทดสอบ Method GET ผ่าน Postman

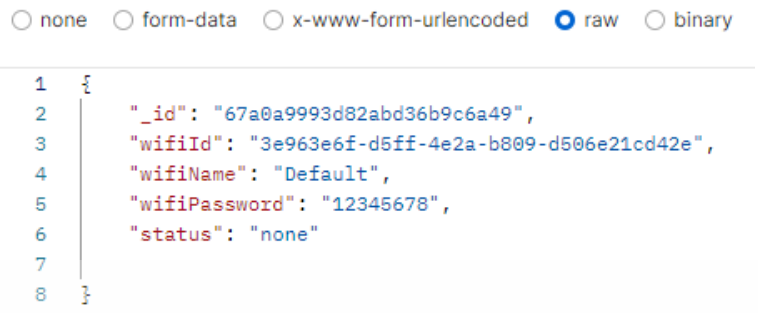
รูปที่ 4.156 ผลการทดสอบเมื่อใช้ Method GET

2) POST

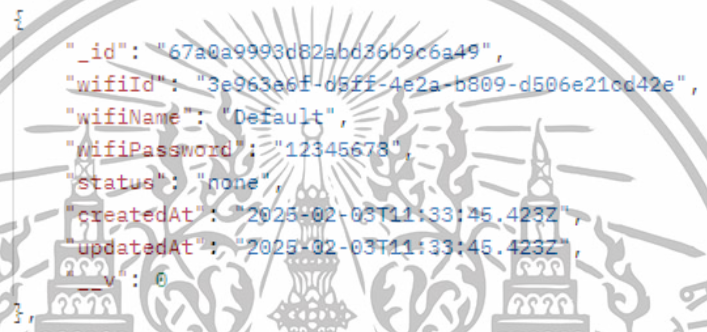
ผลจากการทดสอบด้วยการใช้ Method POST ดังรูป 4.157 เพื่อส่งข้อมูลของเครือข่าย Wi-Fi ใหม่ ไปเก็บในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่ส่งผ่านไปในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) ดังรูปที่ 4.158 ซึ่งมีข้อมูลที่ต้องการเพิ่มลงในตารางหรือเอกสารของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.159



รูปที่ 4.157 ทดสอบ Method POST ผ่าน Postman



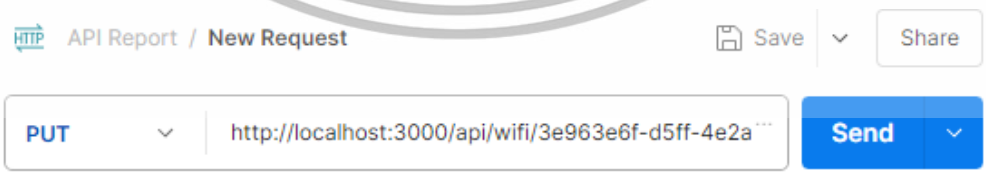
รูปที่ 4.158 ข้อมูล JSON ที่ POST



รูปที่ 4.159 ผลการทดสอบเมื่อใช้ Method POST

3) PUT

ผลจากการทดสอบด้วยการใช้ Method PUT ดังรูป 4.160 เพื่ออัปเดตข้อมูลที่มีอยู่แล้วใน API ของเครือข่าย Wi-Fi ในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่อัปเดตในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) บอกเซิร์ฟเวอร์ ดังรูปที่ 4.161 ซึ่งมีข้อมูลที่ต้องการอัปเดตลงในตารางหรือเอกสารของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.162



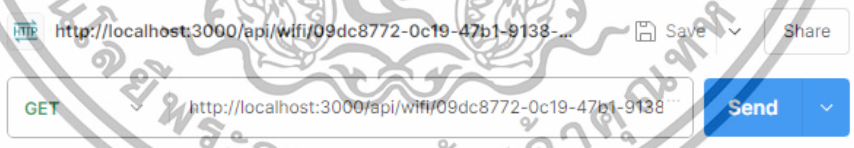
รูปที่ 4.160 ทดสอบ Method PUT ผ่าน Postman

รูปที่ 4.161 ข้อมูล JSON ที่ POST

รูปที่ 4.162 ผลการทดสอบเมื่อใช้ Method PUT

4) GET{id}

ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลอุปกรณ์เฉพาะเจาะจง ดังรูป 4.163 เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบ ข้อมูลทั้งหมดของอุปกรณ์ที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.164



รูปที่ 4.163 ทดสอบ Method GET ด้วย ID ผ่าน Postman

รูปที่ 4.164 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID

4.3.2.5 API ชื่อ Button

API สำหรับการจัดการปรับแต่งปุ่มสั่งการ (Button) จะถูกแบ่งออกเป็น 4 ส่วน ได้แก่ GET POST DELETE และ GET{deviceId}

1) GET

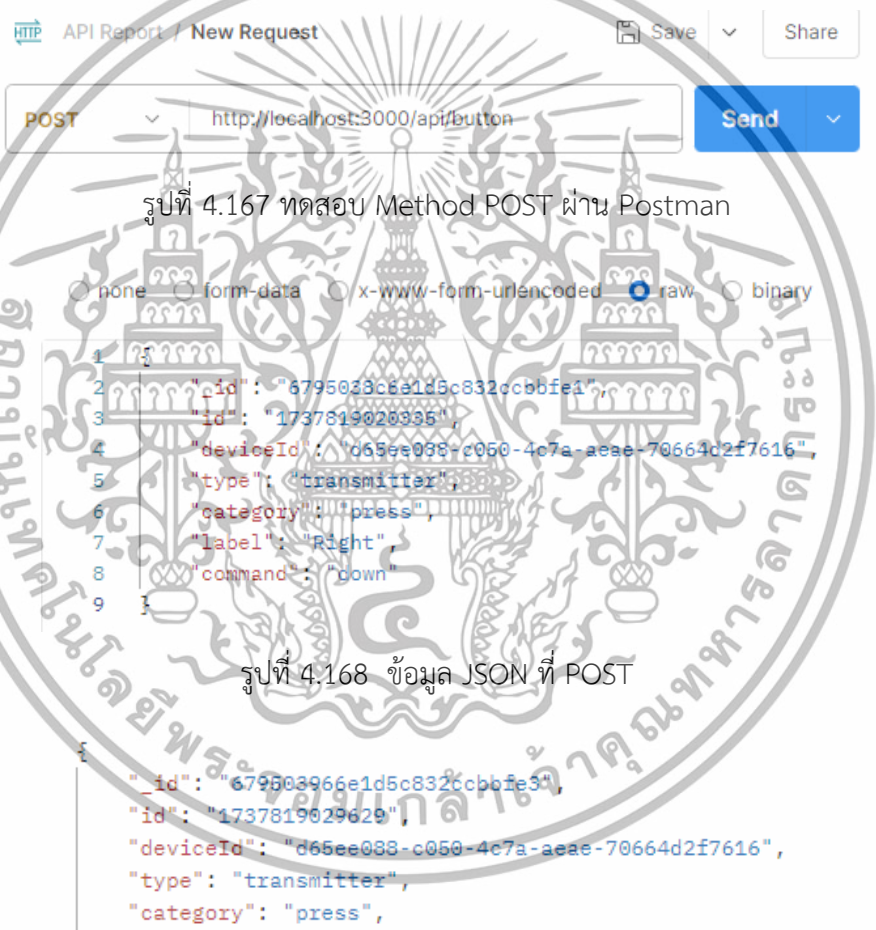
ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลการปรับแต่งปุ่มสั่งการแสดงบนหน้าเว็บแอปพลิเคชัน ดังรูป 4.165 ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ถูกดึงออกมานั้นตรงตามที่จัดเก็บไว้ในฐานข้อมูล โดยข้อมูลดังกล่าวจะถูกจัดเก็บในรูปแบบ JSON ซึ่งเป็นรูปแบบข้อมูลที่อ่านและทำความเข้าใจได้ง่าย เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของการปรับแต่งปุ่มสั่งการที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman โดยมีข้อมูลที่ดึงมาแสดง ดังรูป 4.166

รูปที่ 4.165 ทดสอบ Method GET ด้วย deviceId ผ่าน Postman

รูปที่ 4.166 ผลการทดสอบเมื่อใช้ Method GET

2) POST

ผลจากการทดสอบด้วยการใช้ Method POST ดังรูป 4.167 เพื่อส่งข้อมูลการปรับแต่งปุ่มสั่งการไปเก็บในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่ส่งผ่านไปในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) ดังรูปที่ 4.168 ซึ่งมีข้อมูลที่ต้องการเพิ่มลงในตารางหรือเอกสารของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.169



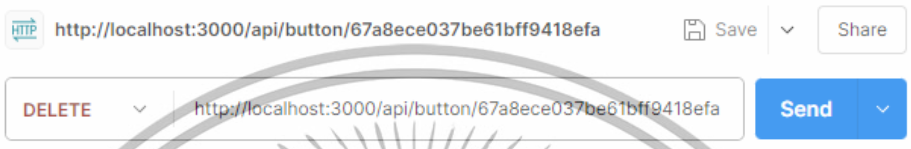
รูปที่ 4.167 ทดสอบ Method POST ผ่าน Postman

รูปที่ 4.168 ข้อมูล JSON ที่ POST

รูปที่ 4.169 ผลการทดสอบเมื่อใช้ Method POST

3) DELETE

ผลจากการทดสอบด้วยการใช้ Method DELETE ดังรูป 4.170 เพื่อส่งคำสั่งลบข้อมูลการปรับแต่งปุ่มสั่งการไปยังฐานข้อมูลที่เชื่อมต่อกับระบบ โดยจะทำการส่งคำขอเพื่อลบข้อมูลของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.171 เมื่อข้อมูลถูกลบสำเร็จ

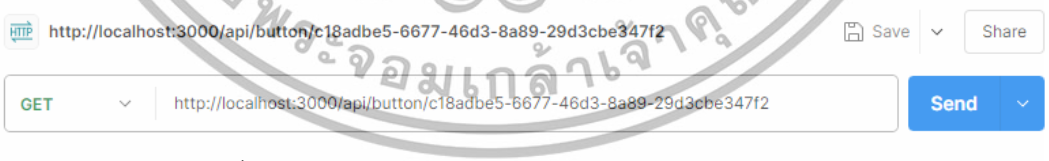


รูปที่ 4.170 ทดสอบ Method DELETE ผ่าน Postman

รูปที่ 4.171 ผลการทดสอบเมื่อใช้ Method DELETE

4) GET{id}

ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลการปรับแต่งปุ่มเฉพาะเจาะจง ดังรูป 4.172 เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของอุปกรณ์ที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.173



รูปที่ 4.172 ทดสอบ Method GET ด้วย ID ผ่าน Postman

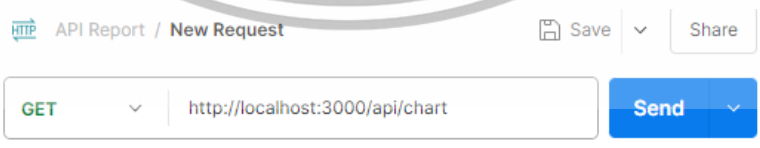
รูปที่ 4.173 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID

4.3.2.6 API ชื่อ Chart

API สำหรับการจัดหน้าปัดการแสดงผล (Chart) จะถูกแบ่งออกเป็น 4 ส่วน ได้แก่ GET POST DELETE และ GET{deviceId}

1) GET

ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลหน้าปัดการแสดงผลแสดงบนหน้าเว็บแอปพลิเคชัน ดังรูป 4.174 ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ถูกดึงออกมานั้นตรงตามที่จัดเก็บไว้ในฐานข้อมูล โดยข้อมูลดังกล่าวจะถูกจัดเก็บในรูปแบบ JSON ซึ่งเป็นรูปแบบข้อมูลที่อ่านและทำความเข้าใจได้ง่าย เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของหน้าปัดการแสดงผลที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman โดยมีข้อมูลต่าง ๆ ที่ได้ออกแบบไว้ในตาราง Products ดังรูป 4.175



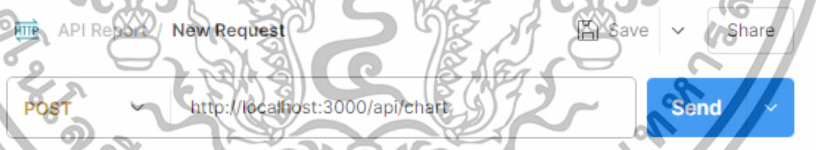
รูปที่ 4.174 ทดสอบ Method GET ผ่าน Postman

```
{
    "_id": "67b2f4e742ffa5f2ec3ceeed",
    "id": "1739781351392",
    "deviceId": "db621f8a-a36c-4978-af0f-4b8c0d4c8dac",
    "type": "monitorcircle",
    "label": "",
    "bgcolor": "bg-[#00ccff]",
    "fgcolor": "text-black",
    "unit": "%",
    "createdAt": "2025-02-17T08:35:51.470Z",
    "updatedAt": "2025-02-17T08:35:51.470Z",
    "__v": 0
},
```

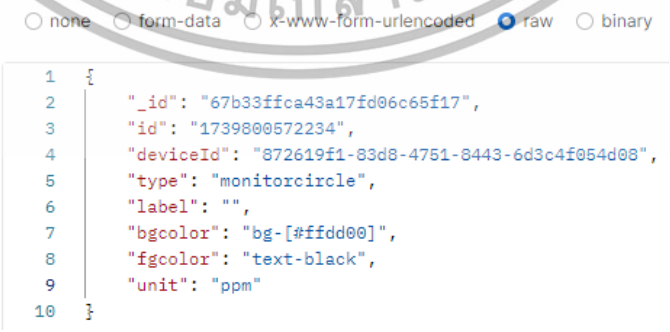

รูปที่ 4.175 ผลการทดสอบเมื่อใช้ Method GET

2) POST

ผลจากการทดสอบด้วยการใช้ Method POST ดังรูป 4.176 เพื่อส่งข้อมูลข้อมูลหน้าปัดการแสดงผลไปเก็บในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยในที่นี้ข้อมูลที่ส่งผ่านไปในรูปแบบ JSON ถูกจัดอยู่ใน Body ของคำขอ (Request Body) ดังรูปที่ 4.177 ซึ่งมีข้อมูลที่ต้องการเพิ่มลงในตารางหรือเอกสารของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman สามารถตรวจสอบได้ว่าข้อมูลที่ส่งออกมาเก็บไว้ในฐานข้อมูล ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.178



รูปที่ 4.176 ทดสอบ Method POST ผ่าน Postman



รูปที่ 4.177 ข้อมูล JSON ที่ POST

```
{
    "_id": "67b33ffca43a17fd06c65f17",
    "id": "1739800572234",
    "deviceId": "872619f1-83d8-4751-8443-6d3c4f054d08",
    "type": "monitorcircle",
    "label": "",
    "bgcolor": "bg-[#ffdd00]",
    "fgcolor": "text-black",
    "unit": "ppm",
    "createdAt": "2025-02-17T13:56:12.285Z",
    "updatedAt": "2025-02-17T13:56:12.285Z",
    "__v": 0
}
```


รูปที่ 4.178 ผลการทดสอบเมื่อใช้ Method POST

3) DELETE

ผลจากการทดสอบด้วยการใช้ Method DELETE ดังรูป 4.179 เพื่อส่งคำสั่งลบข้อมูลหน้าปัดการแสดงผลไปในฐานข้อมูลที่เชื่อมต่อกับระบบ โดยจะทำการส่งคำขอเพื่อลบข้อมูลของฐานข้อมูล MongoDB ผ่านเครื่องมือ Postman ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.180 เมื่อข้อมูลถูกลบสำเร็จ

รูปที่ 4.179 ทดสอบ Method DELETE ผ่าน Postman


```
{
  "message": "Delete Chart Success"
}
```


รูปที่ 4.180 ผลการทดสอบเมื่อใช้ Method DELETE

4) GET{deviceId}

ผลจากการทดสอบด้วยการใช้ Method GET เพื่อดึงข้อมูลหน้าปัดการแสดงผลเฉพาะเจาะจง (GET by ID) ดังรูป 4.181 เมื่อทำการเรียกใช้งาน Method GET ไปยัง API Endpoint ที่ออกแบบไว้ ข้อมูลทั้งหมดของอุปกรณ์ที่จัดเก็บในฐานข้อมูล MongoDB จะถูกดึงออกมาและแสดงผลในรูปแบบ JSON ผ่านทาง Postman ผลทดสอบจะแสดงผลในรูปแบบ JSON ดังในรูปที่ 4.182

รูปที่ 4.181 ทดสอบ Method GET ด้วย ID ผ่าน Postman

รูปที่ 4.182 ผลการทดสอบเมื่อใช้ Method GET ด้วย ID

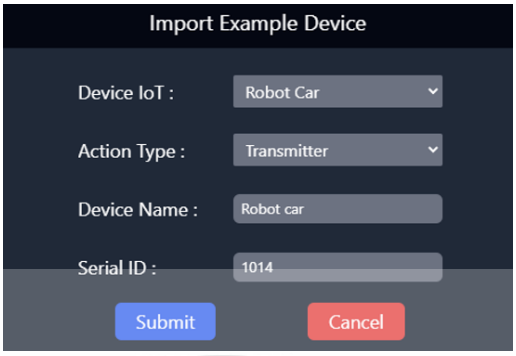
4.4 การนำเข้าอุปกรณ์

ทำการทดสอบโดยการจะแบ่งการทดสอบออกเป็น 2 ส่วน ดังนี้

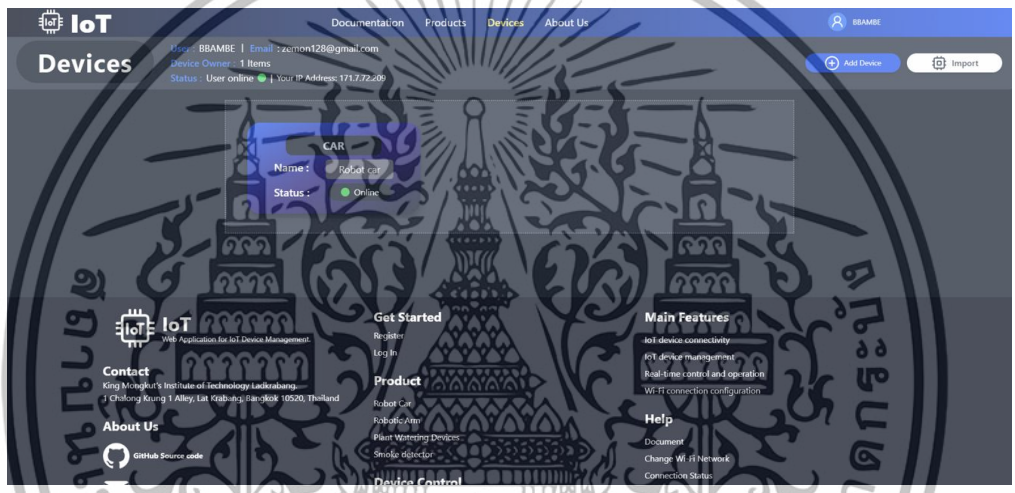
4.4.1 ผลทดสอบการนำเข้าอุปกรณ์ตัวอย่าง

ทดสอบการทำงานโดยการสร้างชุดข้อมูลอุปกรณ์ตัวอย่างภายในบัญชีผู้ใช้ 4 อุปกรณ์ ตามชนิดของอุปกรณ์ตัวอย่าง ได้แก่ รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ โดยในการนำเข้าอุปกรณ์ตัวอย่างจะต้องกดปุ่ม Import Example Device จากหน้ารวมอุปกรณ์มีขั้นตอนการนำเข้าอุปกรณ์ตัวอย่างดังนี้

เมื่อกดปุ่ม Import Example Device จะปรากฏหน้าต่างเพื่อให้ผู้ใช้นำเข้าอุปกรณ์ตัวอย่างที่ตนเองมี ในการนำเข้ารถยนต์บังคับทำได้โดยการกรอกข้อมูล ดังรูปที่ 4.183 โดยอุปกรณ์แต่ละชิ้นจะมี Serial ID ไม่เหมือนกัน ผู้ใช้ต้องกรอกข้อมูลให้ถูกต้องจึงจะนำเข้าอุปกรณ์ตัวอย่างได้สำเร็จ ดังรูปที่ 4.184

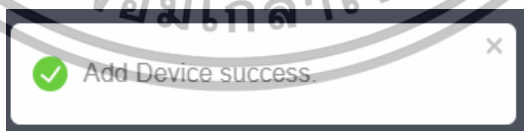


รูปที่ 4.183 หน้าต่างรับข้อมูลรถยนต์บังคับ



รูปที่ 4.184 หน้ารวมแสดงข้อมูลอุปกรณ์ที่ได้เพิ่มภายในบัญชี

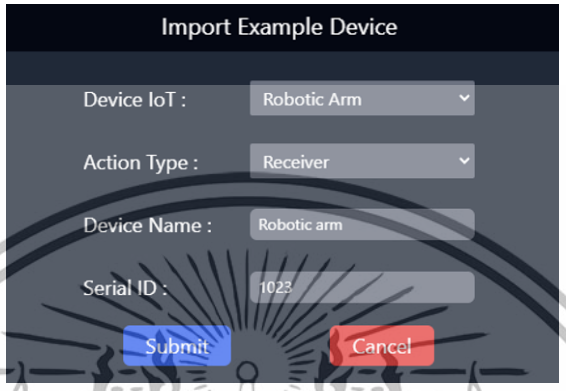
หากข้อมูลที่ผู้ใช้งานกรอกถูกต้องทุกประการ กล่องตอบโต้จะแสดงการทำงานนำเข้าอุปกรณ์ตัวอย่างนั้นสำเร็จ ดังรูปที่ 4.185 แต่ถ้าหากข้อมูลที่กรอกนั้นผิดพลาด หรือ Serial ID นั้นได้ถูกใช้งานไปแล้ว กล่องตอบโต้จะแสดงข้อความ ดังรูปที่ 4.186



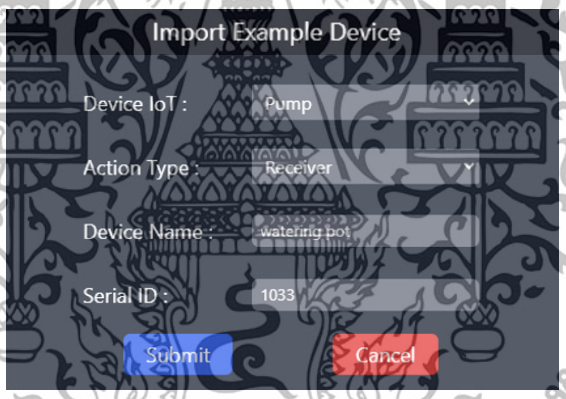
รูปที่ 4.185 กล่องตอบโต้เมื่อเพิ่มอุปกรณ์สำเร็จ

รูปที่ 4.186 กล่องตอบโต้เมื่อเพิ่มอุปกรณ์ไม่สำเร็จ

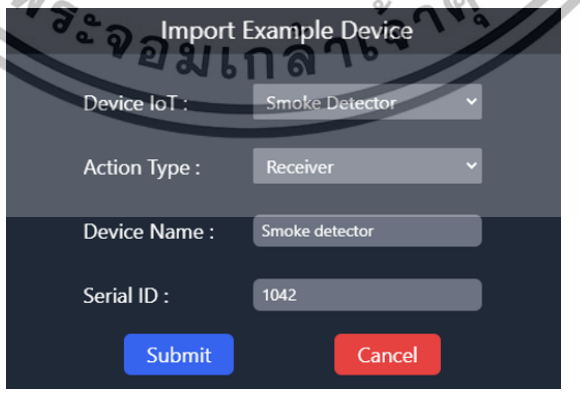
หากเป็นการนำเข้าแขนกล ผู้ใช้งานจะต้องทำการกรอกข้อมูลให้ถูกต้อง ดังรูปที่ 4.187 เช่นเดียวกับการนำเข้าอุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ ผู้ใช้งานจะต้องทำการกรอกข้อมูลให้ถูกต้อง ดังรูปที่ 4.188 และ 4.189 ตามลำดับ



รูปที่ 4.187 หน้าต่างรับข้อมูลแขนกล

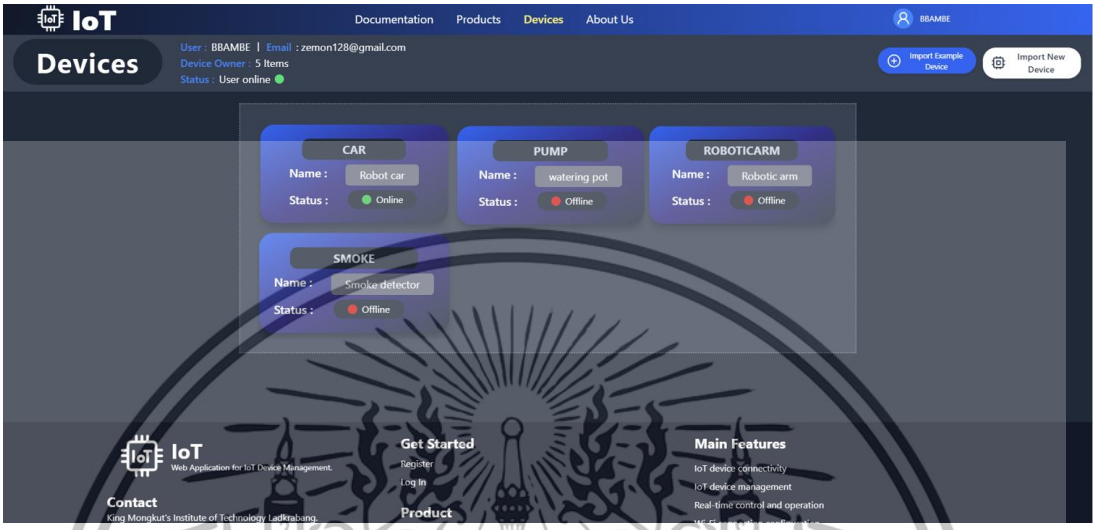


รูปที่ 4.188 หน้าต่างรับข้อมูลอุปกรณ์รดน้ำต้นไม้



รูปที่ 4.189 หน้าต่างรับข้อมูลอุปกรณ์ตรวจจับควันไฟ

เมื่อได้นำเข้าอุปกรณ์ตัวอย่างจากการนำเข้าอุปกรณ์ตัวอย่างเข้ามาครบทุก 4 ชนิด หน้ารวมจะแสดงข้อมูลของอุปกรณ์ตามที่ผู้ใช้งานได้เพิ่มข้อมูลไว้ ดังรูปที่ 4.190

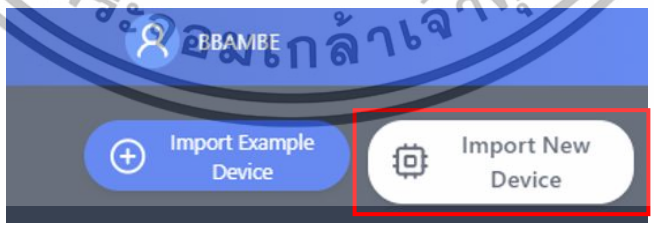


รูปที่ 4.190 หน้ารวมอุปกรณ์

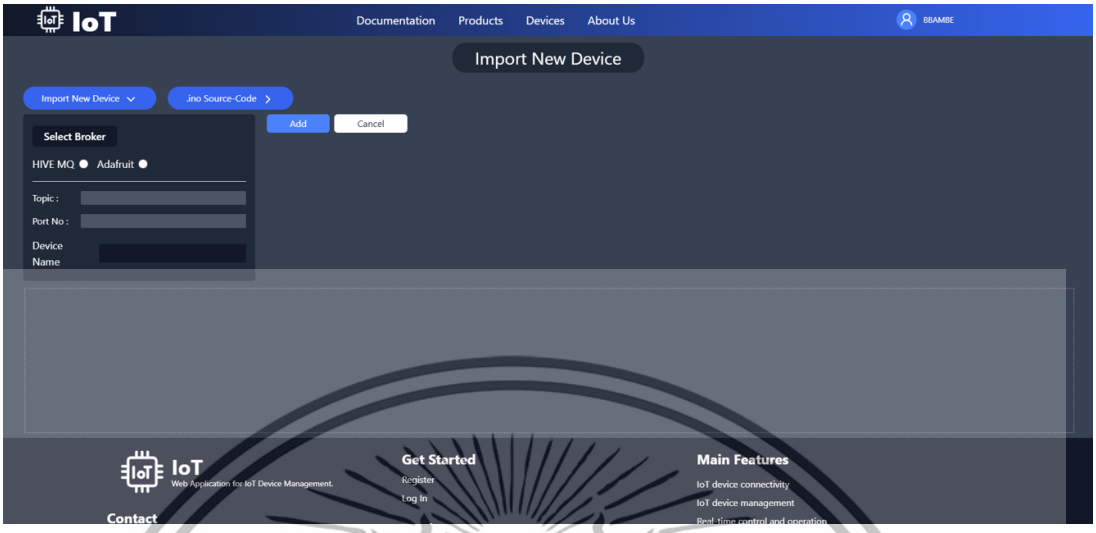
4.4.2 ผลทดสอบการนำเข้าอุปกรณ์ใด ๆ

การทดสอบในการนำเข้าอุปกรณ์ใด ๆ จะทำการสร้างอุปกรณ์ต้นแบบอย่างง่าย คือ การสั่งการเปิด-ปิดพัดลม และไฟ LED เพื่อให้แสดงการทำงานและการใช้งานในคุณสมบัติของการนำเข้าอุปกรณ์ใด ๆ มีขั้นตอนการใช้งาน ดังนี้

เริ่มต้นโดยการกดปุ่ม Import ดังรูปที่ 4.191 เมื่อผู้ใช้กดปุ่มแล้วระบบจะนำเข้าสู่หน้า Import New Device ซึ่งเป็นหน้าในการนำเข้าอุปกรณ์จากภายนอกเข้ามาร่วมใช้งานกับแพลตฟอร์ม ดังรูปที่ 4.192

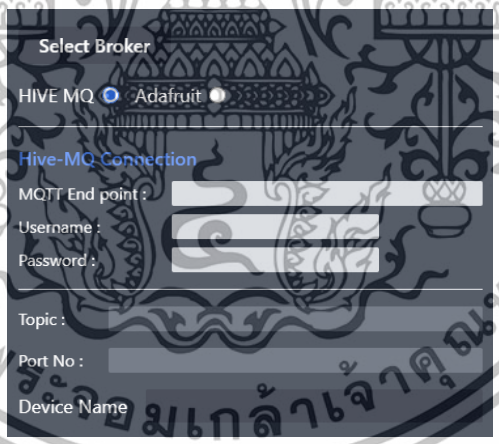


รูปที่ 4.191 ปุ่ม Import New Device

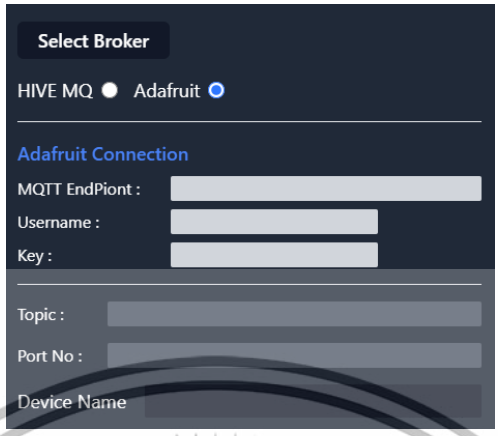


รูปที่ 4.192 หน้าเว็บแอปพลิเคชัน Import New Device

เมื่อเข้าสู่หน้านำเข้าอุปกรณ์จากภายนอก จะปรากฏหน้าต่างรับค่าเพื่อใช้เชื่อมต่อกับ MQTT Broker โดยผู้ใช้ต้องเลือกใช้ HiveMQ Broker ดังรูปที่ 4.193 หรือ Adafruit Broker ดังรูปที่ 4.194 ตามความต้องการของผู้ใช้

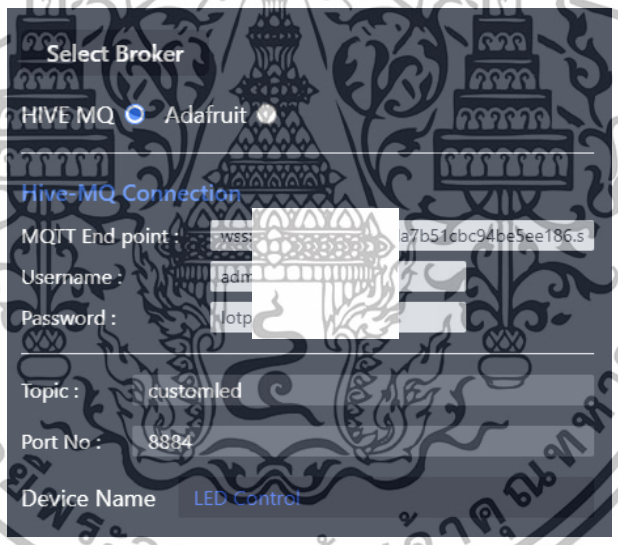


รูปที่ 4.193 หน้าต่างรับค่าเพื่อเชื่อมต่อกับ HiveMQ Broker



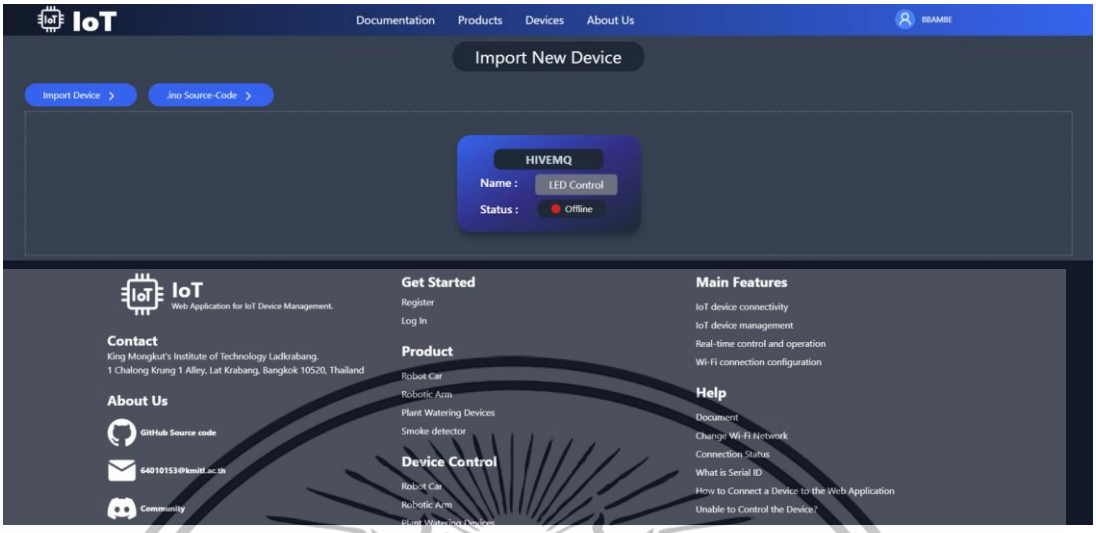
รูปที่ 4.194 หน้าต่างรับค่าเพื่อเชื่อมต่อกับ Adafruit Broker

ในการทดสอบนี้จะใช้ HiveMQ Broker ในการเชื่อมต่อระหว่างเว็บแอปพลิเคชันกับอุปกรณ์ตัวอย่างภายนอก โดยมีรายละเอียดดังรูปที่ 4.195



รูปที่ 4.195 ข้อมูลที่ใช้เชื่อมต่อกับ HiveMQ Broker

และเมื่อกดปุ่ม Add เพื่อนำเข้าอุปกรณ์จากภายนอกสำเร็จจะปรากฏข้อมูลอุปกรณ์ที่ได้เพิ่มเข้ามาตามบัญชีของผู้ใช้ ดังรูปที่ 4.196



รูปที่ 4.196 หน้ารวมอุปกรณ์นำเข้าจากภายนอก

4.5 การเปลี่ยนการเชื่อมต่อ Wi-Fi

จากการออกแบบให้สามารถเปลี่ยนเครือข่ายการเชื่อมต่อ Wi-Fi จากค่าเริ่มต้นไปเป็นเครือข่าย Wi-Fi ของผู้ใช้จากหน้าเว็บแอปพลิเคชัน โดยสั่งการเปลี่ยนเครือข่าย Wi-Fi ผ่านหน้าเว็บแอปพลิเคชัน โดยมี Broker เป็นตัวกลางในการสื่อสารระหว่างหน้าเว็บแอปพลิเคชันกับอุปกรณ์ ESP32

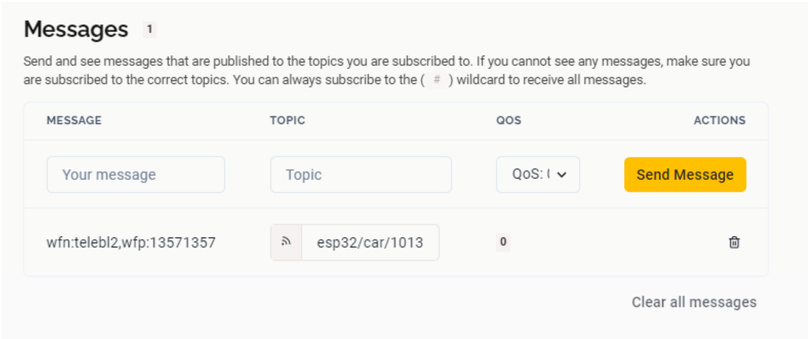
4.5.1 ผลทดสอบการเปลี่ยนเครือข่าย Wi-Fi ใหม่ผ่านหน้าเว็บแอปพลิเคชัน

ทดสอบโดยการเข้าหน้าเว็บแอปพลิเคชันโดยในรถยนต์บังคับเป็นอุปกรณ์ในการเปลี่ยนการเชื่อมต่อเครือข่าย Wi-Fi โดยจากเดิม Wi-Fi ที่ใช้ในการเชื่อมต่อของอุปกรณ์จะเป็น Wi-Fi เริ่มต้นที่มาพร้อมกับอุปกรณ์ ดังรูปที่ 4.197

รูปที่ 4.197 ค่า Wi-Fi เริ่มต้นที่แสดงผลในหน้าเว็บแอปพลิเคชัน

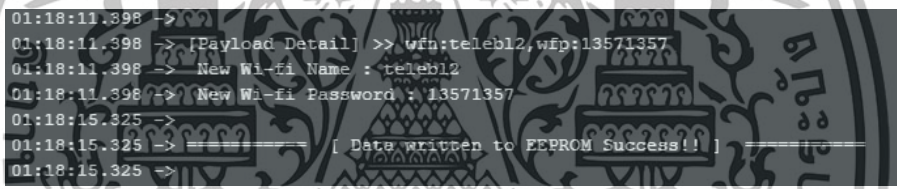
เมื่อมีกรอก Wi-Fi ของผู้ใช้งานที่ต้องการเปลี่ยนเครือข่าย Wi-Fi ผ่านหน้าเว็บแอปพลิชัน ดังรูปที่ 4.198 และกดปุ่ม Change Wi-Fi ที่ผู้ใช้งานได้เปลี่ยนใหม่ล่าสุดจะถูกส่งไปให้อุปกรณ์เพื่อเปลี่ยนการเชื่อมต่อ Wi-Fi ตามผู้ใช้ได้เปลี่ยนแปลง ผ่าน Broker โดยหน้าเว็บแอปพลิเคชันจะทำการ Publish ข้อความของเครือข่าย Wi-Fi ขึ้น Cloud Broker ดังรูปที่ 4.199

รูปที่ 4.198 Wi-Fi เมื่อผู้ใช้ทำการเปลี่ยนผ่านหน้าเว็บแอปพลิเคชัน



รูปที่ 4.199 SSID และ Password ที่ถูก Publish จากหน้าเว็บแอปพลิเคชัน

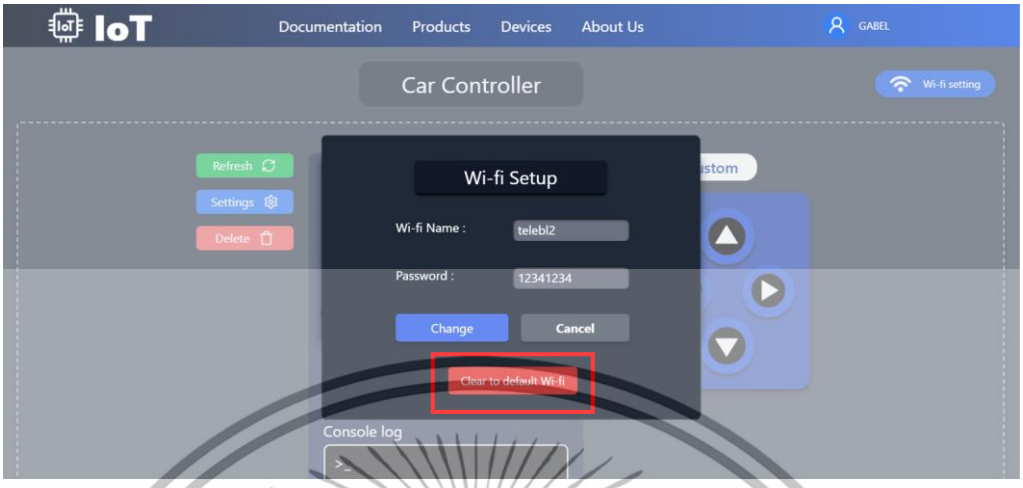
หลังจากนั้นรถยนต์บังคับจะรับค่า Wi-Fi ล่าสุดมาทำการเปลี่ยน SSID และ Password ตามที่ผู้ใช้งานกรอก โดยการ Subscribe แปลงค่าข้อความที่เข้ามาและทำการบันทึกลงใน EEPORM เพื่อให้จดจำ Wi-Fi นี้ไปตลอดแม้จะไม่มีการจ่ายไฟเข้าอุปกรณ์ก็ตาม และทำการเชื่อมต่อ Wi-Fi นั้นต่อไป ดังรูปที่ 4.200



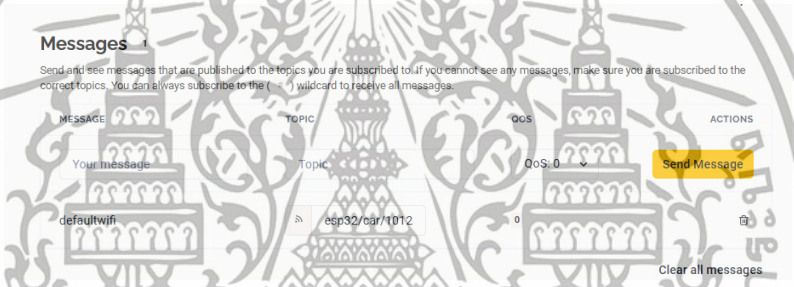
รูปที่ 4.200 บันทึก SSID และ Password ลงใน EEPROM

4.5.2 ผลการเปลี่ยนเครือข่าย Wi-Fi กลับไปค่าเริ่มต้นผ่านหน้าเว็บแอปพลิเคชัน

เมื่อต้องการเปลี่ยน Wi-Fi กลับไปเป็นค่าเริ่มต้น ทำได้โดยการเข้าหน้าเว็บแอปพลิเคชันอุปกรณ์นั้น ๆ ในการทดสอบนี้จะใช้รถยนต์บังคับในการทดสอบ โดยเมื่อมีการเปลี่ยน Wi-Fi ตามความต้องการของผู้ใช้ไปแล้ว ผู้ใช้จะสามารถเปลี่ยนกลับไปเป็นค่าเริ่มต้นได้ ดังรูปที่ 4.201 เมื่อกดปุ่ม Clear to default Wi-Fi หน้าเว็บแอปพลิเคชันจะ Publish ข้อความตามรูปที่ 4.202 เข้า Broker เพื่อรอให้อุปกรณ์ Subscribe ต่อไป

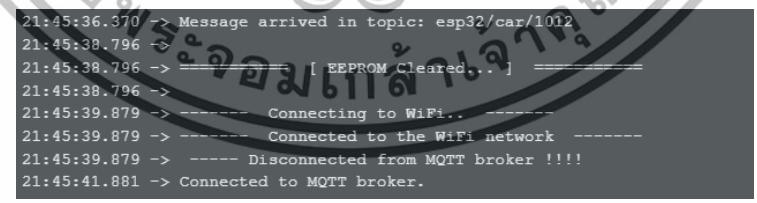


รูปที่ 4.201 หน้าเว็บแอปพลิเคชันเพื่อเปลี่ยนค่ากลับไปเป็น Wi-Fi เริ่มต้น

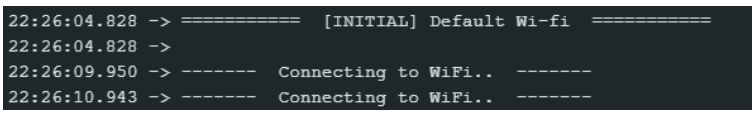


รูปที่ 4.202 ข้อความที่ถูก Publish จากหน้าเว็บแอปพลิเคชัน

หลังจากนั้นรถยนต์บังคับจะทำการ Subscribe ข้อความจาก Broker และทำการลบข้อมูลใน EEPORM ดังรูปที่ 4.203 เพื่อให้กลับไปเป็นค่า SSID และ Password เริ่มต้นของอุปกรณ์ และทำการเชื่อมต่อ Wi-Fi นั้นต่อไป ดังรูปที่ 4.204



รูปที่ 4.203 ลบข้อมูล SSID และ Password ออกจาก EEPROM



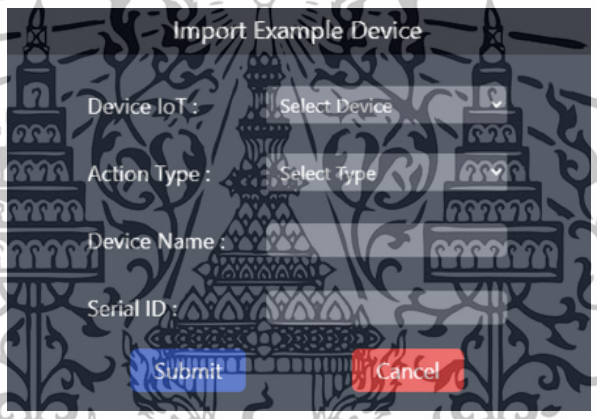
รูปที่ 4.204 ข้อความเมื่อ Wi-Fi ของอุปกรณ์กลับมาเป็นค่าเริ่มต้น

4.6 การปรับแต่งปุ่มควบคุมและการแสดงข้อมูลอุปกรณ์ IoT

ทำการทดสอบโดยแบ่งการปรับแต่งปุ่มเป็น 2 แบบตามการใช้งาน คือ การทดสอบปรับแต่งปุ่มในการสั่งการ และการทดสอบการปรับแต่งปุ่มแสดงข้อมูลอุปกรณ์ IoT

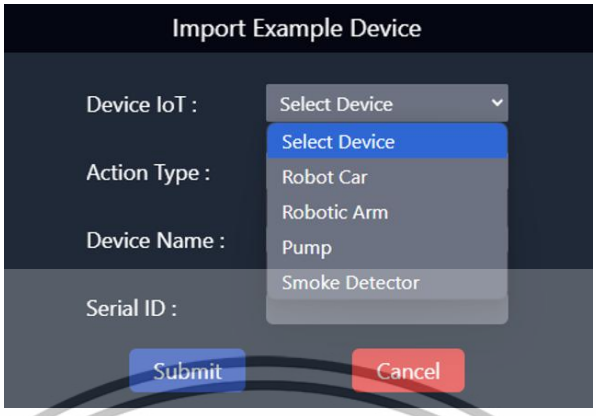
4.6.1 ผลการทดสอบการปรับแต่งปุ่มในการสั่งการ

เริ่มต้นจากการเพิ่มอุปกรณ์ IoT เพื่อใช้ในการสั่งงาน โดยผู้ใช้งานต้องทราบข้อมูลของอุปกรณ์เพื่อให้ในการลงทะเบียนอุปกรณ์ใช้งานร่วมกับเว็บแอปพลิเคชัน ได้แก่ ประเภทของอุปกรณ์ IoT (Device IoT) ประเภทการทำงานของอุปกรณ์ (Action Type) ชื่ออุปกรณ์ (Device Name) และหมายเลขประจำตัวของอุปกรณ์ (Serial ID) โดยผู้ใช้ต้องกรอกข้อมูลข้างต้นใส่หน้าต่างเพิ่มอุปกรณ์ ดังรูปที่ 4.205



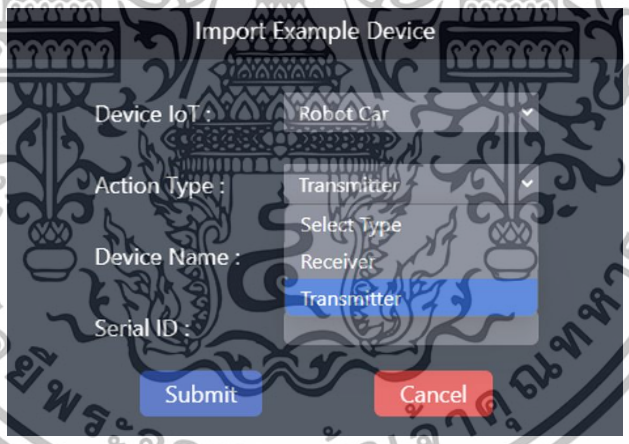
รูปที่ 4.205 หน้าต่างเพิ่มอุปกรณ์

เมื่อทราบประเภทของอุปกรณ์ IoT (Device IoT) ได้แก่ Robot Car Robotic Arm Watering Pot และ Smoke Detector ผู้ใช้งานต้องเลือกประเภทของอุปกรณ์ IoT ที่ตนมีให้ถูกต้อง ในการทดสอบนี้เพิ่มอุปกรณ์ประเภทเครื่องส่งคำสั่ง เลือกอุปกรณ์รถยนต์บังคับ โดยต้องเลือกประเภทอุปกรณ์จากตัวเลือกจากหน้าต่างที่แสดงดังรูป 4.206

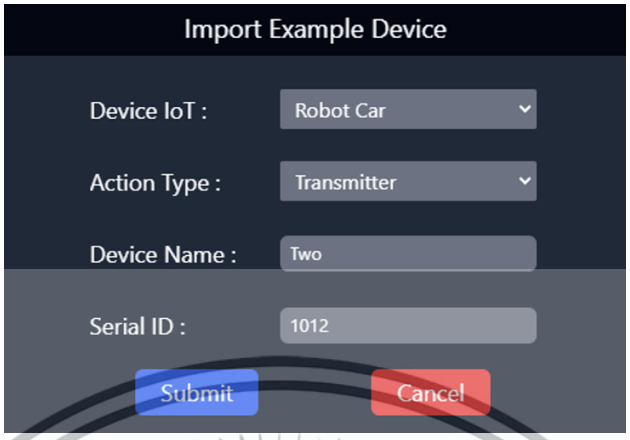


รูปที่ 4.206 หน้าต่างตัวเลือกประเภทอุปกรณ์

หลังจากนั้นผู้ใช้ต้องทราบถึงประเภทการทำงานของอุปกรณ์ (Action Type) เป็นอุปกรณ์ชนิดส่งคำสั่งสั่งการหรือเป็นอุปกรณ์ชนิดรับรับสั่งการ โดยเลือกจากหน้าต่างตัวเลือกในรูปที่ 4.207 และต้องตั้งชื่ออุปกรณ์ (Device Name) ทุกครั้ง พร้อมทั้งต้องทราบหมายเลขประจำตัวของอุปกรณ์ (Serial ID) ดังรูปที่ 4.208

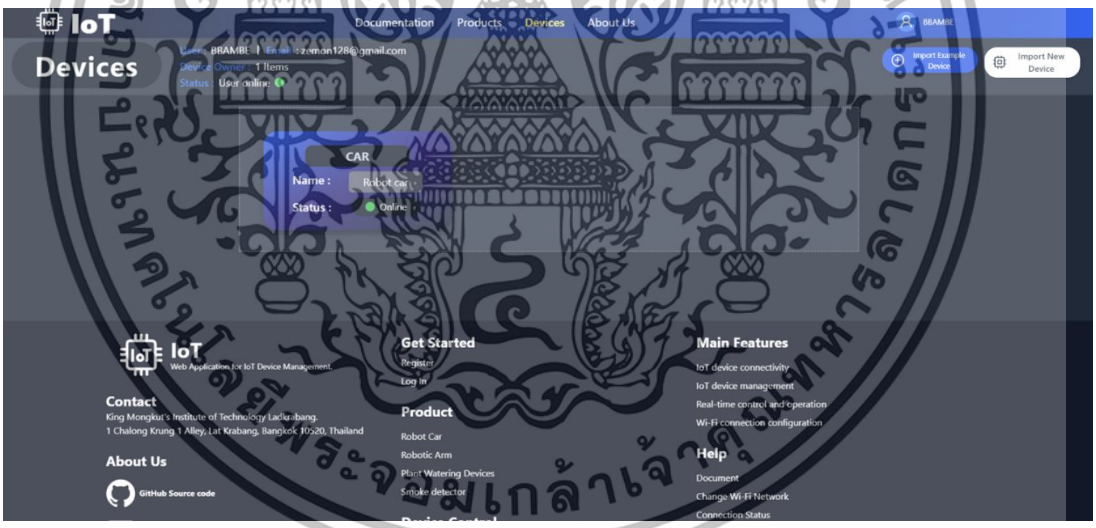


รูปที่ 4.207 หน้าต่างตัวเลือกประเภทการทำงานของอุปกรณ์



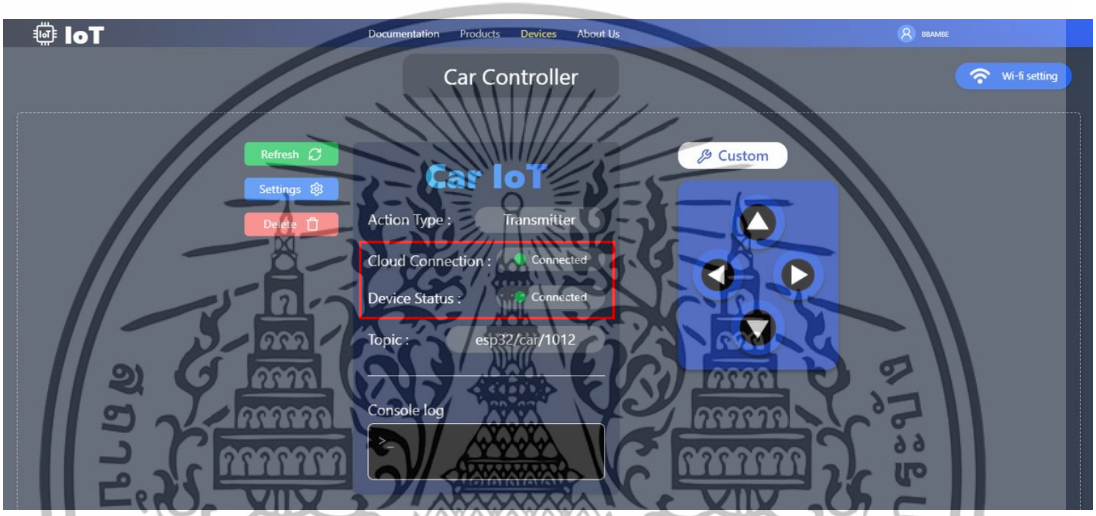
รูปที่ 4.208 หน้าต่างการลงทะเบียนเพิ่มอุปกรณ์

เมื่อกรอกข้อมูลครบเรียบร้อย กดปุ่ม Submit เพื่อบันทึกข้อมูลของอุปกรณ์ จะปรากฏข้อมูลอุปกรณ์ที่ได้ทำการเพิ่มเข้าไป เมื่ออุปกรณ์ IoT ชนิดนั้นได้ทำการเชื่อมต่ออินเทอร์เน็ตผ่านเครือข่าย Wi-Fi จะแสดงสถานะของอุปกรณ์ Online ดังรูปที่ 4.209



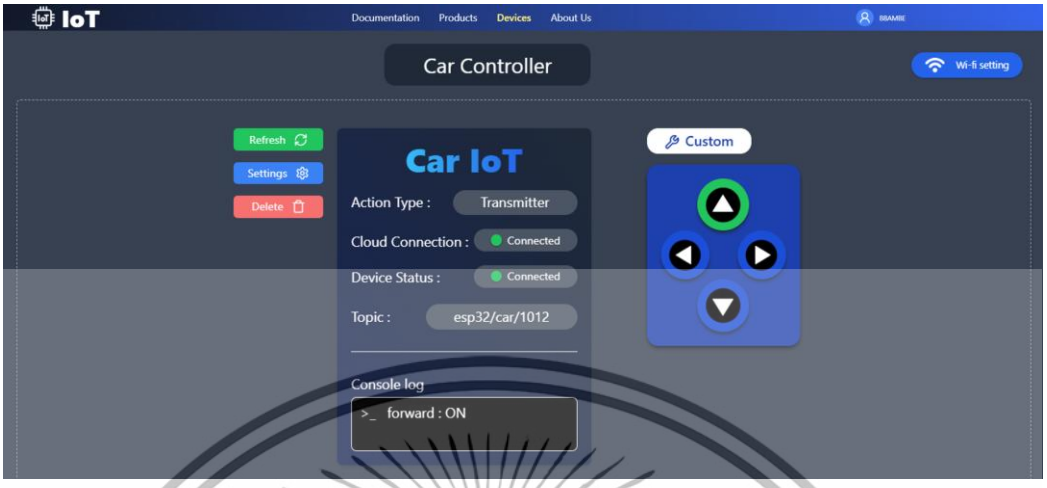
รูปที่ 4.209 หน้าแสดงข้อมูลอุปกรณ์ของผู้ใช้

เชื่อมต่อเว็บแอปพลิเคชันกับรถยนต์บังคับที่จะทำหน้าที่เป็นเครื่องส่งคำสั่งจากหน้าเว็บแอปพลิเคชันเข้าไปควบคุมการทำงานของรถยนต์บังคับให้มอเตอร์ขับเคลื่อนได้ เมื่อสถานะแสดง “Connected” ทั้ง Cloud และอุปกรณ์ จะสามารถสั่งการอุปกรณ์และปรับแต่งปุ่มได้ โดยในตอนเริ่มต้นของหน้าควบคุมอุปกรณ์จะแสดงปุ่มเพื่อให้ผู้ใช้เลือกจะปุ่ม Default ซึ่งเป็นค่าเริ่มต้นของอุปกรณ์ทุกตัวตามประเภทของอุปกรณ์ หรือปุ่ม Custom ซึ่งเป็นปุ่มที่ผู้ใช้สามารถปรับแต่งเองได้ ดังรูปที่ 4.210



รูปที่ 4.210 หน้าเริ่มต้นของอุปกรณ์

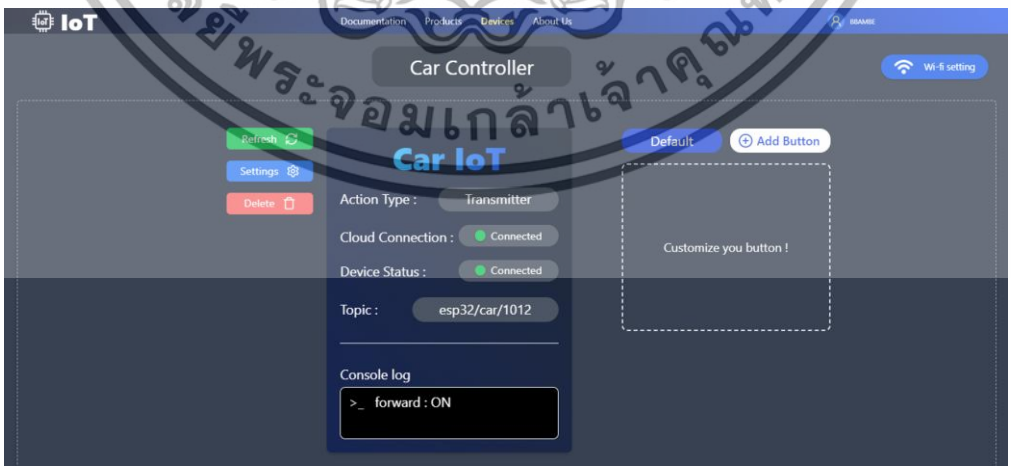
อุปกรณ์ที่สามารถเลือกกดปุ่ม Default ได้จะมีเฉพาะแค่อุปกรณ์ 4 ชนิด คือ Robot Car Robotic Arm Watering Pot และ Smoke Detector ซึ่งเป็นอุปกรณ์ IoT ต้นแบบในการใช้งานร่วมกับเว็บแอปพลิเคชัน หากผู้ใช้เลือกที่จะกดปุ่ม Default ซึ่งเป็นปุ่มควบคุมเริ่มต้นของอุปกรณ์ หน้าเว็บแอปพลิเคชันจะแสดงผลดังรูป 4.211 เมื่อลองกดสั่งคำสั่งเพื่อควบคุมการทำงานจากปุ่มค่าเริ่มต้น คำสั่งจะถูกส่งเข้า Cloud Broker ซึ่งเป็นตัวกลางในการสื่อสารระหว่างอุปกรณ์กับเว็บแอปพลิเคชัน เมื่อกดปุ่มสั่งงานให้เดินหน้า จะฝังคำสั่งไว้ในปุ่ม “forward” และมีคำสั่งเข้า Cloud Broker ดังรูปที่ 4.212



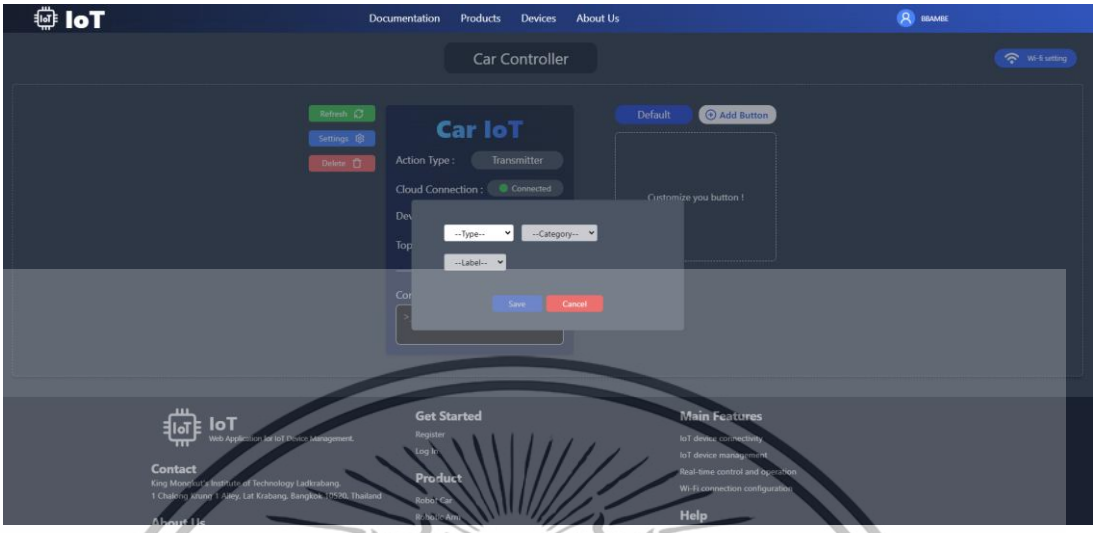
รูปที่ 4.211 ทดสอบกดปุ่มสั่งงานให้เดินหน้า

รูปที่ 4.212 คำสั่งเข้า Cloud Broker

เมื่อผู้ใช้งานต้องการปรับแต่งปุ่ม โดยกดปุ่ม Custom Button และกดปุ่ม Add Button ดังรูปที่ 4.213 จะแสดงหน้าต่างตัวเลือกเพื่อปรับแต่งปุ่ม ดังรูปที่ 4.214

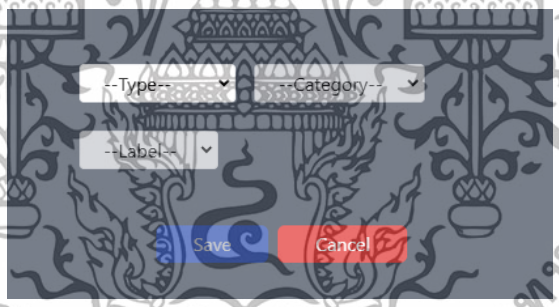


รูปที่ 4.213 หน้าแสดงผลเมื่อกดปุ่ม Custom Button



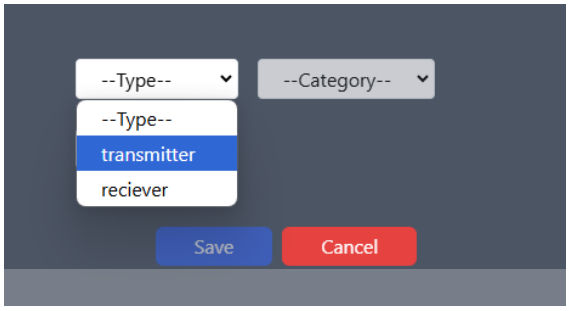
รูปที่ 4.214 หน้าแสดงผลเมื่อกดปุ่ม Add Button

หากอุปกรณ์เป็นประเภทเครื่องส่ง โดยเลือกประเภทการทำงาน และต้องทำการเลือกประเภทปุ่มการสั่งงานตามการใช้งานของอุปกรณ์ จากหน้าต่างตัวเลือกในการปรับแต่งปุ่ม ดังรูปที่ 4.215

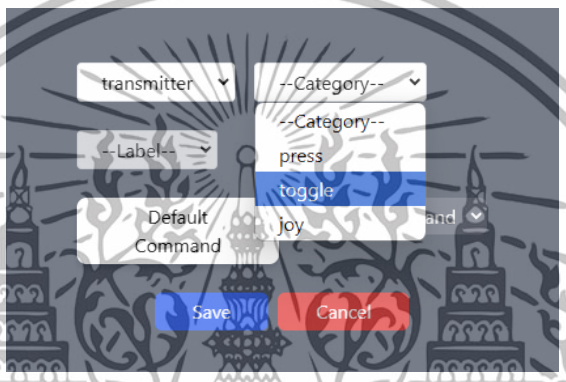


รูปที่ 4.215 หน้าต่างตัวเลือกในการปรับแต่งปุ่ม

โดยต้องเลือกประเภทการทำงานของอุปกรณ์ เพื่อให้ตรงกับการทำงานของอุปกรณ์แต่ละชนิด ดังรูปที่ 4.216 และต้องเลือกประเภทการทำงานงานของปุ่มเพื่อให้ในการส่งคำสั่งจากหน้าเว็บแอปพลิเคชันเพื่อเข้าไปควบคุมอุปกรณ์ ดังรูปที่ 4.217 หลังจากนั้นต้องเลือกรูปแบบปุ่มเพื่อความสวยงามและการเข้าใจง่าย

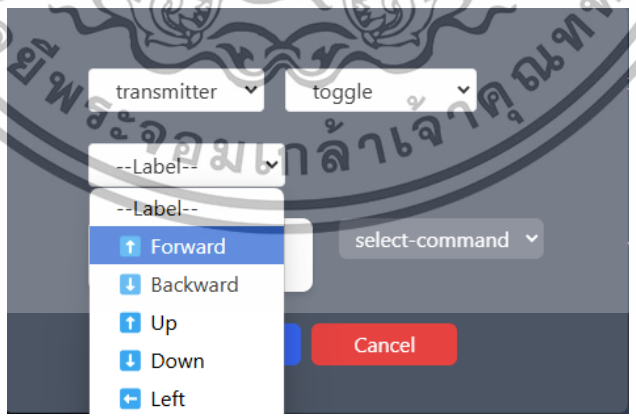


รูปที่ 4.216 หน้าต่างตัวเลือกชนิดการทำงานของอุปกรณ์



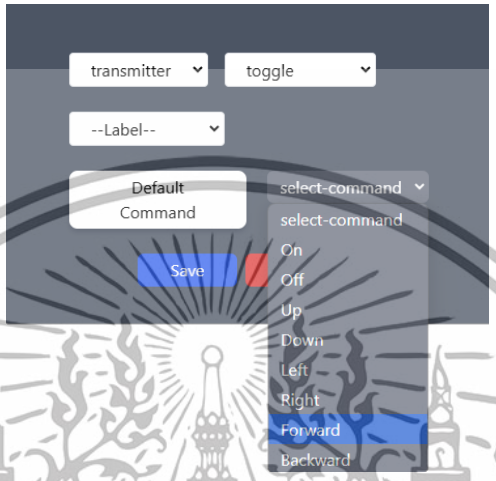
รูปที่ 4.217 หน้าต่างตัวเลือกประเภทการทำงานงานของปุ่ม

หลังจากนั้นต้องเลือกรูปแบบปุ่ม โดยจะเพิ่มปุ่มในรูปลักษณ์ที่ผู้ใช้ต้องเลือกให้ถูกกับความต้องการและการใช้งานของตนเองเพื่อความสวยงามและการเข้าใจง่าย ในการทดสอบนี้จะเลือกปุ่มลูกศรเดินหน้าในการควบคุมรถยนต์บังคับ ดังรูปที่ 4.218



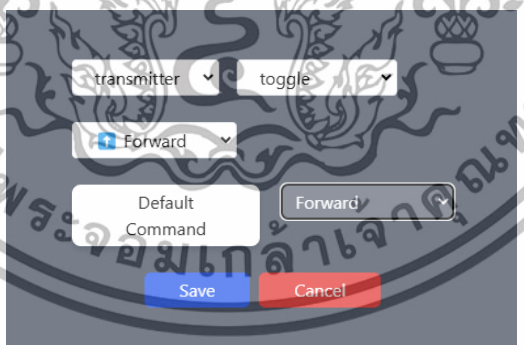
รูปที่ 4.218 หน้าต่างตัวเลือกรูปแบบปุ่ม

ในส่วนสุดท้ายผู้ใช้งานต้องมีการเลือกส่งคำสั่ง (Command) เพื่อส่งคำสั่งไปควบคุมอุปกรณ์ตามที่เขียนโปรแกรม Arduino ที่ได้อัปโหลดลงไมโครคอนโทรลเลอร์ หากผู้ใช้ไม่ต้องการส่งคำสั่งใหม่ สามารถเลือกคำสั่งค่าเริ่มต้นได้ตามประเภทของอุปกรณ์ ดังรูปที่ 4.219

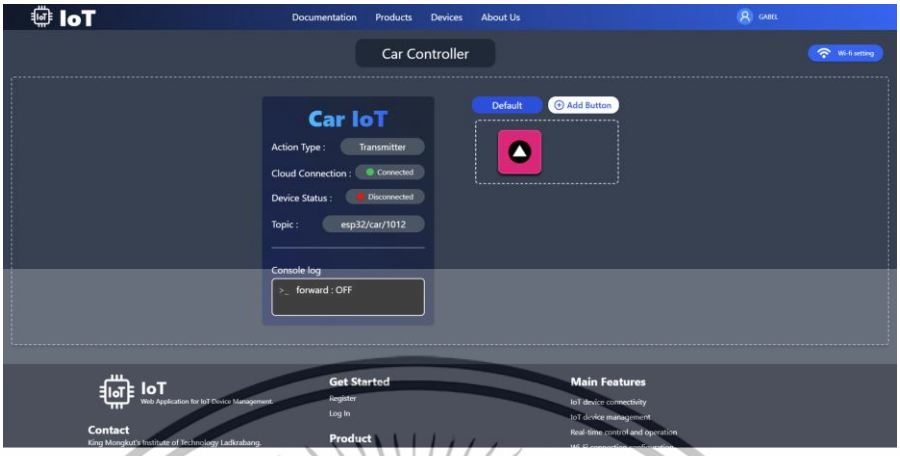


รูปที่ 4.219 หน้าต่างเลือกส่งคำสั่งค่าเริ่มต้น

เมื่อเสร็จสิ้นแล้วให้กดปุ่ม Save เพื่อบันทึกปุ่มที่ได้ทำการปุ่มแต่ง ดังรูปที่ 4.220 และทดสอบกดปุ่มที่ปรับแต่งขึ้นมา ดังรูปที่ 4.221 คำสั่งที่ได้เลือกไว้จาก Command จะถูกสั่งเข้า Cloud Broker เพื่อสั่งงานอุปกรณ์ต่อไป ดังรูปที่ 4.222



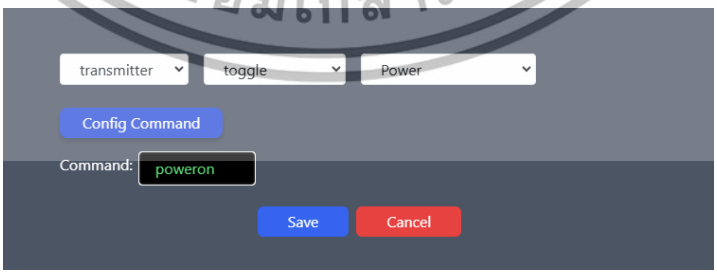
รูปที่ 4.220 หน้าต่างเมื่อปรับแต่งปุ่มเสร็จสิ้น



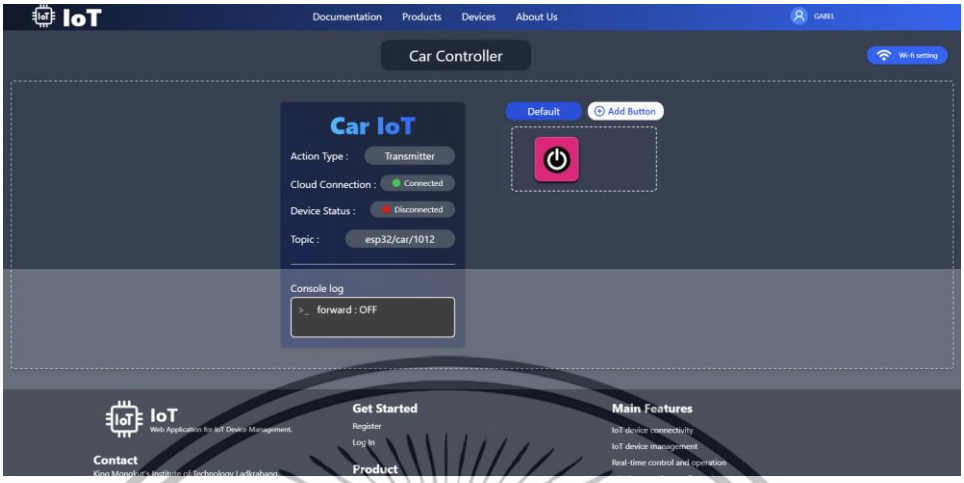
รูปที่ 4.221 หน้าควบคุมอุปกรณ์เมื่อปรับแต่งปุ่มเสร็จสิ้น

รูปที่ 4.222 คำสั่งเข้า Cloud Broker

หากผู้ใช้งานต้องการส่งคำสั่งใหม่ นอกเหนือจากคำสั่งเริ่มต้นสามารถทำได้โดยการกดปุ่ม Config Command และกรอกคำสั่งใหม่ที่ต้องการ จากนั้นกดปุ่ม Save ดังรูปที่ 4.223 และทำการทดสอบกดปุ่มสั่งงาน ดังรูปที่ 4.224 คำสั่งใหม่ที่บันทึกของ Command จะถูกสั่งเข้า Cloud Broker เพื่อสั่งงานอุปกรณ์ต่อไป ดังรูปที่ 4.225



รูปที่ 4.223 หน้าต่างเลือกส่งคำสั่งใหม่

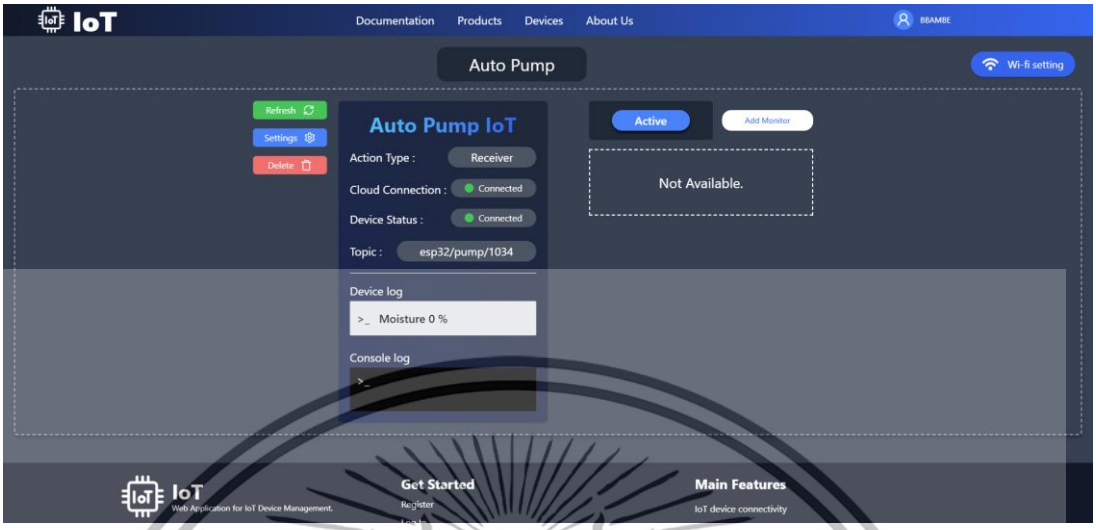


รูปที่ 4.224 ทดสอบกดปุ่มสั่งงาน poweron

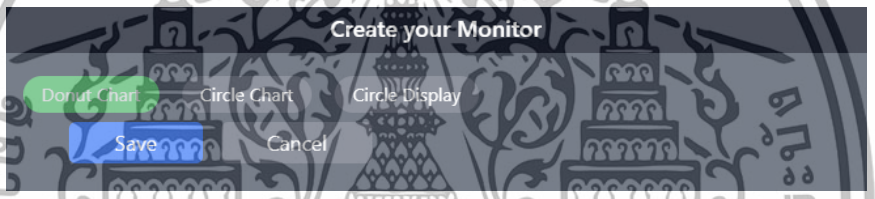
รูปที่ 4.225 คำสั่งเข้า Cloud Broker

4.6.2 ผลการทดสอบการปรับแต่งหน้าจอแสดงผลข้อมูลอุปกรณ์ IoT

ทำการทดสอบโดยใช้อุปกรณ์ประเภทเครื่องรับ คือ อุปกรณ์รดน้ำต้นไม้ ดังรูปที่ 4.226 จากนั้นผู้ใช้ต้องกดปุ่ม Add Monitor จะปรากฏหน้าต่างเพื่อเลือกประเภทการแสดงผลการทำงานของอุปกรณ์เป็นแผนภูมิหรือหน้าปัดวงกลม ดังรูปที่ 4.227

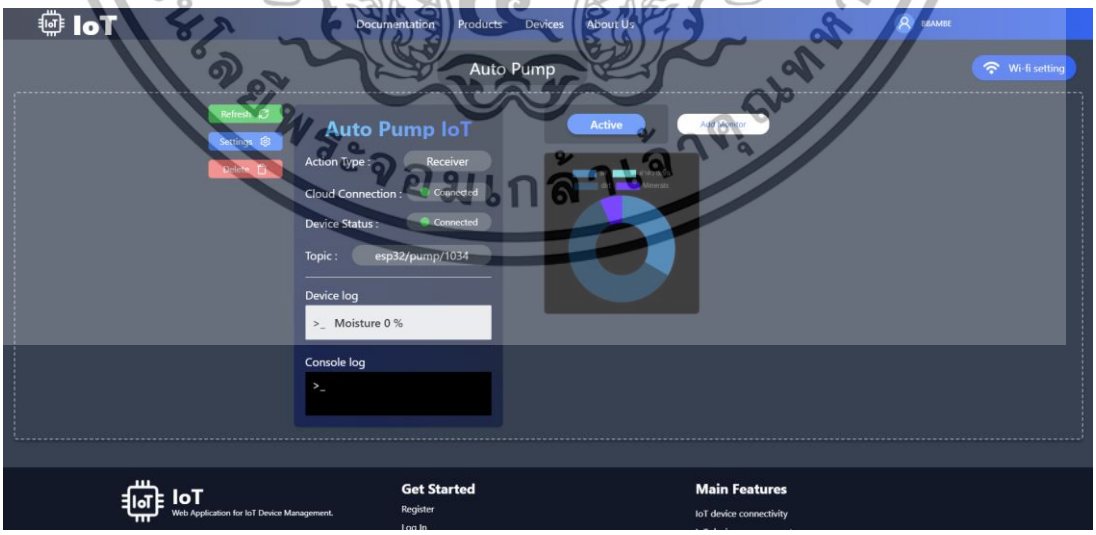


รูปที่ 4.226 หน้าเว็บแอปพลิเคชันอุปกรณ์รดน้ำต้นไม้



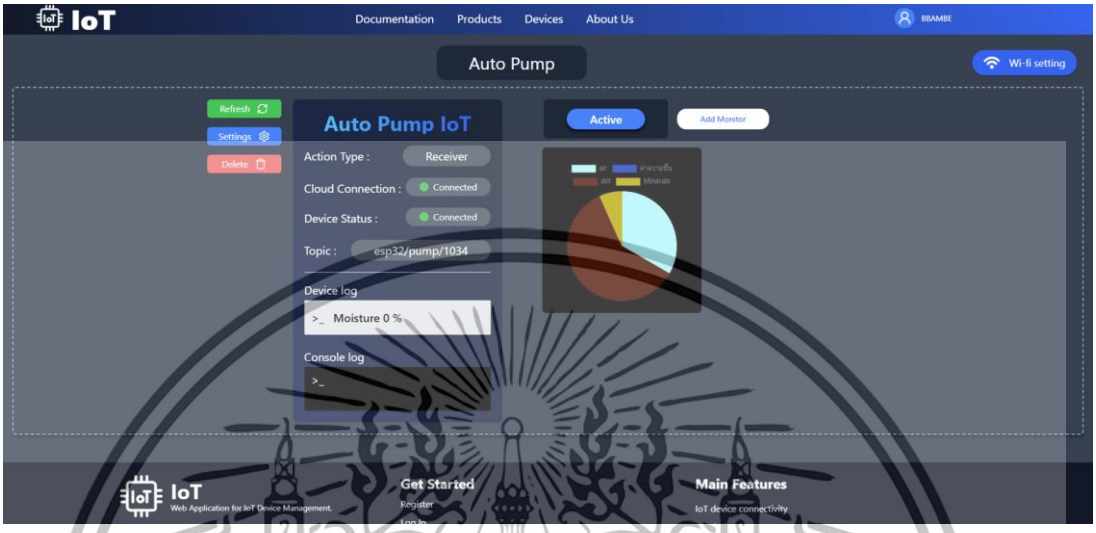
รูปที่ 4.227 หน้าต่างรับค่าเลือกประเภทการแสดงผลการทำงานของอุปกรณ์

หากผู้ใช้เลือก Donut Chart จะแสดงแผนภูมิโดนัทที่แสดงผลข้อมูลการทำงานของอุปกรณ์ ดังรูปที่ 4.228



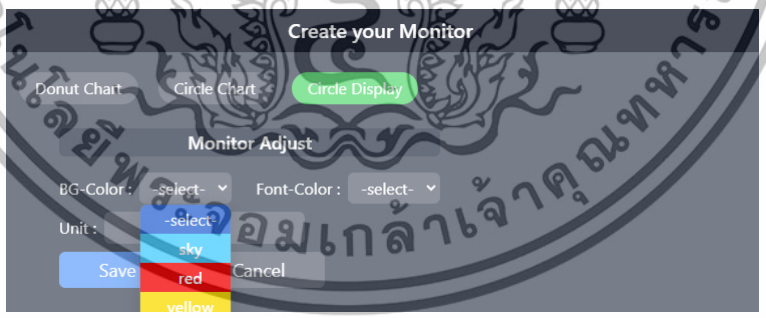
รูปที่ 4.228 แผนภูมิโดนัทที่แสดงผลข้อมูลการทำงานของอุปกรณ์

หากผู้ใช้เลือก Circle Chart จะแสดงแผนภูมิวงกลมที่แสดงผลข้อมูลการทำงานของอุปกรณ์ ดังรูปที่ 4.229

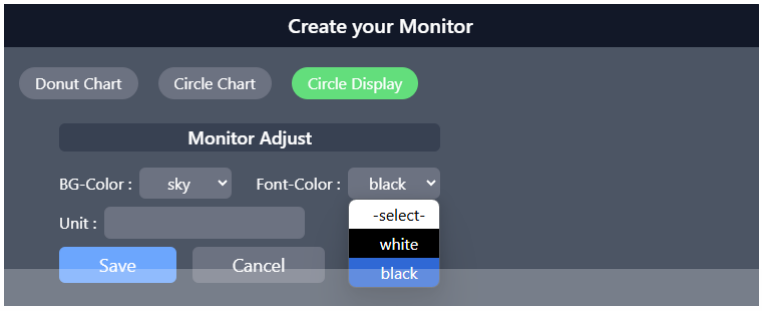


รูปที่ 4.229 แผนภูมิวงกลมที่แสดงผลข้อมูลการทำงานของอุปกรณ์

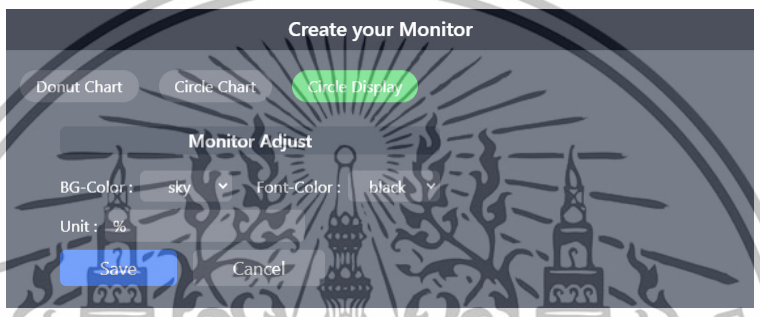
แต่ถ้าหากผู้ใช้เลือก Circle Display จะแสดงหน้าปัดวงกลมที่แสดงผลข้อมูลการทำงานของอุปกรณ์ โดยผู้ใช้งานต้องเลือกสีพื้นหลัง ดังรูปที่ 4.230 สีตัวหนังสือ ดังรูปที่ 4.231 และหน่วยในการแสดงผล ดังรูปที่ 4.232 เมื่อผู้ใช้เลือกเสร็จแล้วจะปรากฏหน้าปัดวงกลม ดังรูปที่ 4.233



รูปที่ 4.230 หน้าต่างเลือกสีพื้นหลังของหน้าปัด



รูปที่ 4.231 หน้าต่างเลือกสีตัวหนังสือของหน้าปัด



รูปที่ 4.232 หน้าต่างรับหน่วยแสดงผลข้อมูล

รูปที่ 4.233 หน้าปัดวงกลมที่แสดงผลข้อมูลการทำงานของอุปกรณ์

บทที่ 5
สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

ปริญญานิพนธ์นี้ประสบความสำเร็จในการศึกษา ออกแบบ และพัฒนาและทดสอบแพลตฟอร์มตัวกลางสำหรับควบคุมและจัดการอุปกรณ์ IoT เนื่องจากแพลตฟอร์มสามารถทำงานได้อย่างมีประสิทธิภาพและตอบสนองความต้องการของผู้ใช้งาน อีกทั้งระบบสามารถรองรับการเพิ่มอุปกรณ์ IoT ได้หลายประเภท เช่น รถยนต์บังคับ แขนกล อุปกรณ์รดน้ำต้นไม้ และอุปกรณ์ตรวจจับควันไฟ โดยสามารถลงทะเบียนอุปกรณ์ใหม่และนำเข้าอุปกรณ์ IoT จากภายนอกเข้ามาใช้งานร่วมกับแพลตฟอร์มผ่าน MQTT Broker ได้สำเร็จ นอกจากนี้ แพลตฟอร์มยังช่วยให้ผู้ใช้สามารถจัดการและควบคุมอุปกรณ์ได้จากระยะไกลผ่านเว็บแอปพลิเคชัน โดยมีฟังก์ชันสำคัญ ได้แก่ การปรับแต่งปุ่มควบคุมและหน้าแสดงข้อมูลการทำงานของอุปกรณ์ การแสดงผลสถานะอุปกรณ์ และการเปลี่ยนแปลงการเชื่อมต่อ Wi-Fi ได้โดยตรงจากหน้าเว็บแอปพลิเคชัน ร่วมถึงการเป็นตัวกลางในการเชื่อมต่อระหว่างอุปกรณ์ IoT ใด ๆ ผ่านโปรโตคอล MQTT ตามมาตรฐานที่แพลตฟอร์มกำหนดขึ้น

การทดสอบระบบพบว่าผู้ใช้สามารถดำเนินการตั้งค่าอุปกรณ์ผ่านเว็บแอปพลิเคชันได้อย่างสะดวก โดยเมื่อทำการเพิ่มอุปกรณ์ ระบบสามารถตรวจสอบและลงทะเบียนหมายเลขเครื่องในฐานข้อมูลได้อย่างถูกต้อง สามารถสั่งงานผ่านหน้าควบคุมอุปกรณ์ไปยังอุปกรณ์ ESP32 ผ่าน MQTT Broker นอกจากนี้ ระบบยังสามารถบันทึกค่าการตั้งค่า Wi-Fi ลงใน EEPROM ของอุปกรณ์และดำเนินการรีเซ็ตค่ากลับไปเป็นค่าเริ่มต้นได้โดยที่ไม่ต้องอัปโหลดโปรแกรมลงอุปกรณ์ใหม่ ผลการทดสอบยังแสดงให้เห็นว่าแพลตฟอร์มสามารถรองรับการทำงานร่วมกับอุปกรณ์หลายตัวพร้อมกัน และสามารถจัดการข้อมูลของอุปกรณ์ได้อย่างเป็นระบบ การออกแบบให้สามารถใช้งานผ่านอินเทอร์เน็ตช่วยให้แพลตฟอร์มมีความยืดหยุ่นสูงขึ้น ทำให้ผู้ใช้สามารถควบคุมอุปกรณ์จากทุกที่ นอกจากนี้ การนำโปรโตคอล MQTT มาใช้เป็นตัวกลางในการสื่อสารระหว่างอุปกรณ์กับเว็บแอปพลิเคชันช่วยเพิ่มความสามารถของระบบการสื่อสารระหว่างแพลตฟอร์มกับอุปกรณ์ IoT จึงสรุปได้ว่าแพลตฟอร์มที่พัฒนาขึ้นสามารถตอบโจทย์การบริหารจัดการอุปกรณ์ IoT ได้อย่างมีประสิทธิภาพ มีความสะดวกในการใช้งาน รองรับอุปกรณ์ได้หลากหลายเหมาะสำหรับเป็นทางเลือกระบบ IoT ในอนาคต

5.2 ข้อเสนอแนะ

1) ระบบสามารถพัฒนาต่อยอดให้รองรับการเพิ่มอุปกรณ์ IoT ได้มากขึ้นในอนาคต รวมถึงการปรับปรุงระบบให้มีความเสถียรในการเชื่อมต่อกับบอร์ด ESP32 หลายตัวพร้อมกัน เพื่อรองรับการใช้งานในขนาดที่ใหญ่ขึ้น

2) ควรเพิ่มฟีเจอร์การแจ้งเตือนเมื่อเกิดปัญหาในการเชื่อมต่อกับอุปกรณ์ IoT หรือเมื่ออุปกรณ์ทำงานผิดปกติ เพื่อให้ผู้ใช้งานสามารถดำเนินการแก้ไขได้ทันที

3) การเพิ่มฟังก์ชันการบันทึกประวัติการสั่งงานและการทำงานของอุปกรณ์จะช่วยให้ผู้ใช้งานสามารถติดตามผลการใช้งานย้อนหลัง และนำข้อมูลเหล่านั้นไปปรับปรุงการทำงานของอุปกรณ์ในอนาคตได้

บรรณานุกรม

[1] กรุงเทพมหานคร. “Internet of Things.”. [ออนไลน์].เข้าถึงได้จากhttps://apps.bangkok.go.th/info_gidsedbkk/bmainfo/data_DDS/document/internet-of-things.pdf. (วันที่ค้นข้อมูล 5 ตุลาคม 2567)

[2] Sethi, Pallavi, and Smruti R. Sarangi. Internet of Things: Architectures, Protocols, and Applications. Department of Computer Science, IIT Delhi: Journal of Electrical and Computer Engineering, 2017.เข้าถึงได้จากhttps://onlinelibrary.wiley.com/doi/10.1155/2017/9324035. (วันที่ค้นข้อมูล 21 สิงหาคม2567)

[3] Usmani, Mohammad Faiz. MQTT Protocol for the IoT - Review Paper. Hessen, Germany: Frankfurt University of Applied Sciences, 2021. เข้าถึงได้จาก https://www.researchgate.net/publication/373640610_MQTT_Protocol_for_the_IoT_-_Review_Paper. (วันที่ค้นข้อมูล 25 สิงหาคม 2567)

[4] IBM. “Transmission Control Protocol/Internet Protocol.”, 2024. https://www.ibm.com/docs/en/aix/7.2?topic=management-transmission-control-protocol internet-protocol. (วันที่ค้นข้อมูล 21 ตุลาคม 2567)

[5] HiveMQ. “MQTT Essentials Part 3: Client and Broker Connection Establishment.” [ออนไลน์]. เข้าถึงได้จาก https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/. (วันที่ค้นข้อมูล 27 สิงหาคม 2567)

[6] Ndutared. “Understanding MQTT with Eclipse Paho Python.” Dev.to. [ออนไลน์]. เข้าถึงได้จาก https://dev.to/ndutared/understanding-mqtt-with-eclipse-paho-python-664. (วันที่ค้นข้อมูล 5 ตุลาคม 2567)

[7] NETPIE. “MQTT API Documentation.” [ออนไลน์]. เข้าถึงได้จาก https://docs.netpie.io/mqtt-api.html#message-api-topic. (วันที่ค้นข้อมูล 5 ตุลาคม 2567)

บรรณานุกรม (ต่อ)

[8] HiveMQ. “MQTT Topics and Wildcards Best Practices - Part 5.” [ออนไลน์]. เข้าถึงได้จาก https://dev.to/hivemq_/mqtt-topics-wildcards-best-practices-part-5-87g. (วันที่ค้นข้อมูล 3 ตุลาคม 2567)

[9] Mishra, Biswajeeban. “TMCAS: An MQTT Based Collision Avoidance System for Railway Networks.” Conference Paper, University of Szeged, July 2018.เข้าถึงได้จาก https://www.researchgate.net/figure/TCP-MQTT-PacketFormat_fig1_327123494. (วันที่ค้นข้อมูล 15 ตุลาคม 2567)

[10] Assetwolf. “MQTT QoS: Understanding Quality of Service.” [ออนไลน์]. เข้าถึงได้จาก https://assetwolf.com/learn/mqtt-qos-understanding-quality-of-service. (วันที่ค้นข้อมูล 11 ตุลาคม 2567)

[11] Predict. “An Era of IoT: M2M Communication Protocols - MQTT.” Medium. [ออนไลน์]. เข้าถึงได้จาก https://medium.com/predict/an-era-of-iot-m2m-communication-protocols-mqtt-e68d81b93613. (วันที่ค้นข้อมูล 1 ตุลาคม 2567)

[12] Gcore. “What is WebSocket?” [ออนไลน์]. เข้าถึงได้จาก https://gcore.com/learning/what-is-websocket/. (วันที่ค้นข้อมูล 9 ตุลาคม 2567)

[13] HiveMQ. “HiveMQ Cloud Documentation.” [ออนไลน์]. เข้าถึงได้จาก https://docs.hivemq.com/hivemq-cloud/index.html (วันที่ค้นข้อมูล 6 สิงหาคม 2567)

[14] HiveMQ. “Send MQTT Messages Using HiveMQ Cloud Web Client.” [ออนไลน์]. เข้าถึงได้จาก https://www.hivemq.com/blog/send-mqtt-messages-using-hivemq-cloud-web-client/. (วันที่ค้นข้อมูล 6 สิงหาคม 2567)

[15] Mustafa Tamer. “Part 3: Adafruit IO (MQTT & Webhooks).” [ออนไลน์]. เข้าถึงได้จาก https://hackmd.io/@lnuiot/r1yEtcs55?utm_source=previewmode&utm_medium=rec. (วันที่ค้นข้อมูล 27 มกราคม 2568)

บรรณานุกรม (ต่อ)

[16] DeepSeaDev. “ESP32 Chip Explained and Advantages.” [ออนไลน์]. เข้าถึงได้จาก https://www.deepseadev.com/en/blog/esp32-chip-explained-and-advantages/. (วันที่ค้นข้อมูล 10 สิงหาคม 2567)

[17] Arduino. “Arduino IDE v1 Basics.” [ออนไลน์]. เข้าถึงได้จาก https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics/. (วันที่ค้นข้อมูล 12 สิงหาคม 2567)

[18] Mischianti. “DOIT ESP32 Dev Kit V1 High Resolution Pinout and Specs.”[ออนไลน์]. เข้าถึงได้จาก https://mischianti.org/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/. (วันที่ค้นข้อมูล 15 สิงหาคม 2567)

[19] IMI Component. “ESP32 NodeMCU.” [ออนไลน์]. เข้าถึงได้จาก https://www.imiconsystem.com/product/esp32-nodemcu/. (วันที่ค้นข้อมูล 22 สิงหาคม 2567)

[20] Espressif Systems. “ESP32 Datasheet.” [ออนไลน์]. เข้าถึงได้จาก https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. (วันที่ค้นข้อมูล 10 ตุลาคม 2567)

[21] Random Nerd Tutorials. “ESP32 Useful Wi-Fi Functions.” [ออนไลน์]. เข้าถึงได้จาก https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/. (วันที่ค้นข้อมูล 10 ตุลาคม 2567)

[22] Arduino4Pro. “DC Gear Motor 3-6 V.” [ออนไลน์]. เข้าถึงได้จาก https://www.arduino4pro.com/product/1154/dc-gear-motor-3-6-v-1220 (วันที่ค้นข้อมูล 12 ตุลาคม 2567)

[23] Amazon. “Electron Single Science Electronic Projects.” [ออนไลน์]. เข้าถึงได้จาก https://www.amazon.in/Electron-Single-Science-Elecronic-projects/dp/B0CQ22TZCT. (วันที่ค้นข้อมูล 10 ตุลาคม 2567)

บรรณานุกรม (ต่อ)

[24] Random Nerd Tutorials. “ESP32 DC Motor Control with L298N Motor Driver.” [ออนไลน์]. เข้าถึงได้จาก https://randomnerdtutorials.com/esp32-dc-motor-l298n-motor-driver-control-speed-direction/. (วันที่ค้นข้อมูล 19 ตุลาคม 2567)

[25] Giribabu Dandabathula. “Illustration of L298N Dual H-Bridge Motor Driver.” ResearchGate. [ออนไลน์]. เข้าถึงได้จาก https://www.researchgate.net/figure/Illustration-of-L298N-Dual-H-Bridge-Motor-Driver_fig2_335517346. (วันที่ค้นข้อมูล 14 ตุลาคม 2567)

[26] Meccanismo Complesso. “Arduino Servo Motors: How They Work and How to Control Them.” [ออนไลน์]. เข้าถึงได้จาก https://www.meccanismocomplesso.org/en/arduino-servo-motors-how-they-work-and-how-to-control-them/. (วันที่ค้นข้อมูล 8 ตุลาคม 2567)

[27] AZ-Delivery. “MG90S Micro Servo Motor.” [ออนไลน์]. เข้าถึงได้จาก https://www.az-delivery.de/en/products/mg90s-micro-servomotor. (วันที่ค้นข้อมูล 8 ตุลาคม 2567)

[28] Wokwi. “Learn Servo Motor Using Wokwi Logic Analyzer.” [ออนไลน์]. เข้าถึงได้จาก https://blog.wokwi.com/learn-servo-motor-using-wokwi-logic-analyzer/. (วันที่ค้นข้อมูล 8 ตุลาคม 2567)

[29] Components101. “Soil Moisture Sensor Module.” [ออนไลน์]. เข้าถึงได้จาก https://components101.com/modules/soil-moisture-sensor-module. (วันที่ค้นข้อมูล 19 ตุลาคม 2567)

[30] DatasheetHub. “Soil Moisture Meter for Arduino.” [ออนไลน์]. เข้าถึงได้จาก https://www.datasheethub.com/soil-moisture-meter-for-arduino/#google_vignette. (วันที่ค้นข้อมูล 19 ตุลาคม 2567)

บรรณานุกรม (ต่อ)

[31] Eleparts. “DC Motor 3V little water pump.” [ออนไลน์]. เข้าถึงได้จากhttps://m.eleparts.co.kr/data/goods_attach/202207/good-pdf-11902681-1.pdf. (วันที่ค้นข้อมูล 19 ตุลาคม 2567)

[32] TidoTech. “Relay Module 3V 1 Channel.” [ออนไลน์]. เข้าถึงได้จากhttps://www.tido.tech/index.php/product/relay-module-3v-1-channel/. (วันที่ค้นข้อมูล 14 ตุลาคม 2567)

[33] Pololu Robotics and Electronics. “MQ-2 Semiconductor Sensor for Combustible Gas.” [ออนไลน์]. เข้าถึงได้จาก https://www.pololu.com/file/0J309/mq2.pdf. (วันที่ค้นข้อมูล 27 มกราคม 2568)

[34] ESPboards. “ESP32 SHTC3 Temperature and Humidity Sensor.” [ออนไลน์]. เข้าถึงได้จาก https://www.espboards.dev/sensors/shtc3/. (วันที่ค้นข้อมูล 27 มกราคม 2568)

[35] Myduino. “MQ2 Gas/Smoke Sensor.” [ออนไลน์]. เข้าถึงได้จาก https://myduino.com/product/jhs-289/. (วันที่ค้นข้อมูล 27 มกราคม 2568)

[36] Autodesk. “How to Interface With Active Buzzer Sensor Module.” [ออนไลน์]. เข้าถึงได้จาก https://www.instructables.com/How-to-Interface-With-Active-Buzzer-Sensor-Module/. (วันที่ค้นข้อมูล 29 มกราคม 2568)

[37] Amazon. “Jopto 5 Pack DC 3.3-5V Low Level Alarm Sound Module Compatible with Arduino.” [ออนไลน์]. เข้าถึงได้จาก https://www.amazon.com.be/-/en/Jopto-3-3-5V-Module-Compatible-Arduino/dp/B096ZWVL43. (วันที่ค้นข้อมูล 29 มกราคม 2568)

[38] TechTarget. “Web Application (Web App) Definition.” [ออนไลน์]. เข้าถึงได้จาก https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app (วันที่ค้นข้อมูล 21 ตุลาคม 2567)

บรรณานุกรม (ต่อ)

[39] vFunction. “3-Tier Application.” [ออนไลน์]. เข้าถึงได้จากhttps://vfunction.com /blog/3-tier-application/. (วันที่ค้นข้อมูล 1 ตุลาคม 2567)

[40] Zealous System. “Web Application Architecture.” [ออนไลน์]. เข้าถึงได้จาก https://www.zealousys.com/blog/web-application-architecture/. (วันที่ค้นข้อมูล 1 ตุลาคม 2567)

[41] Mozilla. “What Is a Web Browser?” Firefox Browsers. เข้าถึงได้จาก https://www. mozilla.org/en-US/firefox/browsers/what-is-a-browser/. (วันที่ค้นข้อมูล 17 ตุลาคม 2567)

[42] Cloudflare. “Why is HTTP Not Secure?” [ออนไลน์]. เข้าถึงได้จาก https://www.clo udflare.com/learning/ssl/why-is-http-not-secure/. (วันที่ค้นข้อมูล 19 ตุลาคม 2567)

[43] Javatpoint. “Web Server Definition.” [ออนไลน์]. เข้าถึงได้จาก https://www.javat point.com/web-server-definition. (วันที่ค้นข้อมูล 2 ตุลาคม 2567)

[44] Amazon Web Services. “What is Database?” [ออนไลน์]. เข้าถึงได้จาก https://aws. amazon.com/th/what-is/database/. (วันที่ค้นข้อมูล 3 ตุลาคม 2567)

[45] Amazon Web Services. “The Difference Between HTTPS and HTTP.” [ออนไลน์]. เข้าถึงได้จาก https://aws.amazon.com/th/compare/the-difference-between-https-and-http/. (วันที่ค้นข้อมูล 4 ตุลาคม 2567)

[46] IBM. “What is an API?” [ออนไลน์]. เข้าถึงได้จาก https://www.ibm.com/topics/api. (วันที่ค้นข้อมูล 9 ตุลาคม 2567)

[47] Khan, Asad. “How Do APIs Work?” LinkedIn. [ออนไลน์]. เข้าถึงได้จาก https://www .linkedin.com/pulse/how-do-apis-work-asad-khan-sgsxe/. (วันที่ค้นข้อมูล : 17 ตุลาคม 2567)

[48] Jittagorn P. “What is WebSocket?” [ออนไลน์]. เข้าถึงได้จาก https://www.jittagornp .me/blog/what-is-websocket/. (วันที่ค้นข้อมูล : 19 ตุลาคม 2567)

บรรณานุกรม (ต่อ)

[49] Gcore. “What is WebSocket?” [ออนไลน์]. เข้าถึงได้จาก https://gcore.com/learning/what-is-websocket/. (วันที่ค้นข้อมูล 7 ตุลาคม 2567)

[50] Thai Programmer. “Web Guideline: Frontend - Next.js.” [ออนไลน์]. เข้าถึงได้จาก https://roadmap.thaiprogrammer.org/paths/web-guideline/frontend/nextjs. (วันที่ค้นข้อมูล 16 ตุลาคม 2567)

[51] Morphosis. “Tailwind CSS: A Framework that Makes Dev Work Easier.” [ออนไลน์]. เข้าถึงได้จาก https://morphos.is/th/blog/tailwind-css-a-framework-that-makes-dev-work-easier. (วันที่ค้นข้อมูล 16 ตุลาคม 2567)

[52] DevHub. “What is JSON?” [ออนไลน์]. เข้าถึงได้จาก https://devhub.in.th/blog/what-is-json (วันที่ค้นข้อมูล 17 ตุลาคม 2567)

[53] MDN Web Docs. “HTTP Cookies.” [ออนไลน์]. เข้าถึงได้จาก https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. (วันที่ค้นข้อมูล 17 ตุลาคม 2567)

[54] คณาจารย์ประจำสาขาวิชาการจัดการเทคโนโลยีสารสนเทศ. “โปรแกรมคอมพิวเตอร์เบื้องต้น” Bookdown.org. [ออนไลน์]. เข้าถึงได้จากhttps://bookdown.org/somsak_c/_946-141__introduction_to_computer_programming/ch1/ch1.html. (วันที่ค้นข้อมูล 18 มกราคม 2568)

[55] มหาวิทยาลัยเทคโนโลยีราชมงคลธัญบุรี. “Introduction to C Programming.” [ออนไลน์]. เข้าถึงได้จาก https://www.compro.rmutt.ac.th/wp-content/uploads/2020/08/CH3_Introduction_to_C-1.pdf .(วันที่ค้นข้อมูล 12 ตุลาคม 2567)

[56] WebDodee. “What is JavaScript?” [ออนไลน์]. เข้าถึงได้จากhttps://webdodee.com/what-is-javascript/. (วันที่ค้นข้อมูล 5 ตุลาคม 2567)

[57] Thai Programmer. “Web Guideline: Backend - TypeScript.” [ออนไลน์]. เข้าถึงได้จาก https://roadmap.thaiprogrammer.org/paths/webguideline/backend/typescript. (วันที่ค้นข้อมูล 7 ตุลาคม 2567)

บรรณานุกรม (ต่อ)

[58] มหาวิทยาลัยนเรศวร. “บทที่ 2 เอกสารและงานวิจัยที่เกี่ยวข้อง.” [ออนไลน์]. เข้าถึงได้จาก https://nuir.lib.nu.ac.th/dspace/bitstream/123456789/3462/5/chapter2.pdf .(วันที่ค้นข้อมูล 11 ตุลาคม 2567)

[59] Apipong. “การบรรยายเรื่องรูปแบบการออกแบบฐานข้อมูล (DB Model).” [ออนไลน์]. เข้าถึงได้จากhttps://apipong.weebly.com/uploads/5/3/5/8/53586393/bs312_lect_03_db_model.pdf. (วันที่ค้นข้อมูล 7 ตุลาคม 2567)

[60] Witsawa Chanton, Monster. “Non-relational database.” [ออนไลน์]. เข้าถึงได้จาก https://monsterconnect.co.th/what-is-non-relational-database/. (วันที่ค้นข้อมูล 20 มกราคม 2568)

[61] Amazon DocumentDB. “What Is a Document Database?.” [ออนไลน์]. เข้าถึงได้จาก https://aws.amazon.com/th/nosql/document/. (วันที่ค้นข้อมูล 20 มกราคม 2568)

[62] MongoDB. “What is MongoDB?” [ออนไลน์]. เข้าถึงได้จาก https://www.mongodb.com/company/what-is-mongodb. (วันที่ค้นข้อมูล 20 ตุลาคม 2567)

[63] Danilo Beuche และ Mark Dalgarno. “Software Product Line Engineering with Feature Models.” [ออนไลน์]. เข้าถึงได้จาก https://www.pure-systems.com/fileadmin/downloads/purevariants/tutorials/SPLWithFeatureModelling.pdf. (วันที่ค้นข้อมูล 30 มกราคม2568)

[64] Dusan Rodina, Software Ideas. “Feature Model Diagram” [ออนไลน์]. เข้าถึงได้จาก https://www.softwareideas.net/feature-model-diagram. (วันที่ค้นข้อมูล 31 มกราคม 2568)

[65] NotebookSpec. “Acer Nitro V 15 ANV15-51-574G.” [ออนไลน์]. เข้าถึงได้จาก https://notebookspec.com/notebook/13092-acer-nitro-v-15-anv15-51-574g.html. (วันที่ค้นข้อมูล 21 ตุลาคม 2567)

บรรณานุกรม (ต่อ)

[66] NotebookSpec. “Lenovo Ideapad 330S (81F400SQTA).” [ออนไลน์]. เข้าถึงได้จาก https://notebookspec.com/notebook/8877-lenovo-ideapad-330s-81f400sqta.html. (วันที่ค้นข้อมูล 29 ตุลาคม 2567)

[67] NotebookSpec. “ASUS TUF Gaming F15 FX506LH-HN002T.” [ออนไลน์]. เข้าถึงได้จาก https://notebookspec.com/notebook/10945-asus-tuf-gaming-f15-fx506lh-hn002t.html. (วันที่ค้นข้อมูล 29 ตุลาคม 2567)

[68] Postman. “API Testing Platform.” [ออนไลน์]. เข้าถึงได้จาก https://www.postman.com/api-platform/api-testing/. (วันที่ค้นข้อมูล 29 ตุลาคม 2567)

[69] HiveMQ. “Send MQTT Messages Using HiveMQ Cloud Web Client.” [ออนไลน์]. เข้าถึงได้จาก https://www.hivemq.com/blog/send-mqtt-messages-using-hivemq-cloud-web-client/. (วันที่ค้นข้อมูล 2 ตุลาคม 2567)

[70] TechTarget. “Google Chrome.” [ออนไลน์]. เข้าถึงได้จาก https://www.techtarget.com/searchmobilecomputing/definition/Google-Chrome. (วันที่ค้นข้อมูล 28 ตุลาคม 2567)