

ไอโอทีแพลตฟอร์มในโรงเพาะเห็ด
IoT Platform for Mushroom Nursery

ขันติชัย รุจิตรการโชติกุล
ชยุต สุรกุล

รายงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2560

ปริญญานิพนธ์ปีการศึกษา 2560

ภาควิชาวิศวกรรมคอมพิวเตอร์คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ไอโอทีแพลตฟอร์มในโรงเพาะเห็ด

IOT PLATFORM FOR MUSHROOM NURSERY

ผู้จัดทำ

- | | | |
|---------------|-------------------|-----------------------|
| 1. นายจันตชัย | รุจิระการ โชติกุล | รหัสนักศึกษา 57010127 |
| 2. นายชยุต | สุรกุล | รหัสนักศึกษา 57010266 |

อาจารย์ที่ปรึกษา
(อาจารย์เกียรติณรงค์ ทองประเสริฐ)

อาจารย์ที่ปรึกษาร่วม
(อาจารย์สรยุทธ กลมกล่อม)

ไอโอทีแพลตฟอร์มในโรงเพาะเห็ด

นายขันติชัย	รุจิระการ โชติกุล	57010127
นายชยุต	สุรกุล	57010266
อาจารย์เกียรติณรงค์ ทองประเสริฐ	อาจารย์ที่ปรึกษา	
อาจารย์สรยุทธ	กลมกล่อม	อาจารย์ที่ปรึกษาร่วม
ปีการศึกษา 2560		

บทคัดย่อ

โครงการนี้จัดทำขึ้นเพื่อศึกษา ออกแบบ และสร้างไอโอทีแพลตฟอร์มสำหรับใช้ในโรงเพาะเห็ด โดยการใช้ Raspberry Pi 3 Model B ซึ่งเป็น Microcontroller มาใช้วัดค่าอุณหภูมิและความชื้นจากเซนเซอร์ภายในโรงเรือน จากนั้นจึงส่งข้อมูลอุณหภูมิและความชื้นที่วัดได้ไปยัง Platform และเก็บข้อมูลอุณหภูมิและความชื้นลงบนฐานข้อมูลโดยใช้ MySQL เป็นทั้งฐานข้อมูลและตัวจัดการฐานข้อมูล นอกจากนั้นยังมีระบบ Web Application เพื่อใช้แสดงผลค่าอุณหภูมิและความชื้นที่ถูกส่งมาจาก Raspberry Pi หรือประวัติของค่าอุณหภูมิและความชื้นที่เก็บไว้ในฐานข้อมูล

โดย Protocol ที่เป็นตัวกลางในการรับส่งข้อมูลระหว่าง Raspberry Pi, Platform และ Web Application คือ MQTT Protocol และใช้ร่วมกับ NodeJS ซึ่งเป็น Service หลักที่ใช้ภายใน Platform

IoT Platform for Mushroom Nursery

Mr. Kantichai	Rujitrakarnchotikul	57010127
Mr. Chayoot	Surakul	57010266
Mr. Kiatnarong	Tongprasert	Advisor
Mr. Sorayut	Glomglome	Co-Advisor

Academic Year 2560

ABSTRACT

This project aims to study, design and build the IoT platform for the mushroom nursery by using Raspberry Pi 3 Model B+, a microcontroller to measure the temperature and humidity from sensors in the mushroom nursery. Then sending the measured temperature and humidity data to the platform and stores them in the database using MySQL. In addition, there is a web application to display real-time temperature and humidity data sent from Raspberry Pi or historical data stored in the database.

The protocol used to transmit data among Raspberry Pi, the platform and web application is MQTT protocol, and use NodeJS as the main service in the platform.

กิตติกรรมประกาศ

โครงการไอโอทีแพลตฟอร์มในโรงพยาบาลเห็นผลสำเร็จลุล่วงไปได้ด้วยดีด้วยความกรุณาและความช่วยเหลืออย่างสูงจากอาจารย์เกียรติคุณรงค์ ทองประเสริฐ ซึ่งเป็นอาจารย์ที่ปรึกษาของโครงการนี้ และอาจารย์สรยุทธ กลมกล่อม ซึ่งเป็นอาจารย์ที่ปรึกษาร่วมที่ได้ให้คำปรึกษาโครงการและการวางแผนการดำเนินงานตลอดจนให้คำแนะนำและแนะแนวทางปฏิบัติ รวมทั้งเสียสละเวลาอันมีค่าติดตามความคืบหน้าของโครงการอย่างสม่ำเสมอ และแก้ไขข้อบกพร่องของเนื้อหาและสำนวนของภาษาด้วยความเอาใจใส่อย่างยิ่ง

ขอขอบคุณอาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ได้ให้ความรู้ด้านทฤษฎีและการปฏิบัติในรายวิชาต่างๆที่เกี่ยวข้องกับการทำโครงการนี้ ทั้งความรู้ด้านฮาร์ดแวร์ ซอฟต์แวร์ และระบบเครือข่าย ตลอดจนการเรียนรู้การทำงานเป็นกลุ่ม และสามารถทำงานร่วมกับผู้อื่นได้เป็นอย่างดี

ขอกราบขอบพระคุณบิดามารดาที่ให้อำนาจใจและสนับสนุนในทุกๆด้าน รวมทั้งให้ความรู้และคำแนะนำในการทำงาน เพื่อส่งเสริมให้ผลลัพธ์ของโครงการนี้ให้เป็นไปตามเป้าหมายที่กำหนดไว้

ขอขอบคุณห้อง Make Ducation และห้อง HCRL ที่เป็นแหล่งสนับสนุนด้านสถานที่และอุปกรณ์ในการพัฒนาโครงการได้อย่างสะดวก และขอบคุณสมาชิกภายในห้อง Make Ducation และห้อง HCRL ทุกคนที่มีส่วนร่วมให้โครงการนี้สำเร็จลุล่วงไปได้ด้วยดี

ขอบคุณรุ่นพี่ และเพื่อนๆตลอดจนผู้ที่เกี่ยวข้องทุกท่านที่ไม่ได้กล่าวนามไว้ ณ ที่นี้ที่ได้ให้คำปรึกษาและคำแนะนำ รวมทั้งแสดงความคิดเห็นต่อโครงการในด้านต่างๆ

สุดท้ายนี้ด้วยคุณค่าและประโยชน์อันพึงมีจากโครงการนี้ ขอมอบแด่ผู้มีพระคุณทุกท่านที่กล่าวมาข้างต้น และขอกราบขอบพระคุณทุกท่านมา ณ ที่นี้

ขันติชัย รุจิระการโชติกุล

ชยุต สุรกุล

สารบัญ

หน้า

ไอโอทีแพลตฟอร์มในโรงเพาะเห็ด	I
IoT Platform for Mushroom Nursery	II
สารบัญ	IV
สารบัญตาราง	IX
สารบัญรูป	X
บทที่ 1 บทนำ	1
1.1 ที่มาและความสำคัญของโครงการ	1
1.2 วัตถุประสงค์ของโครงการ	2
1.3 ประโยชน์ที่คาดว่าจะได้รับ	2
1.4 ขอบเขตของโครงการ	2
1.5 แผนการดำเนินการ	5
บทที่ 2 ทฤษฎีที่เกี่ยวข้อง	6
2.1 Microcontroller และอุปกรณ์ต่างๆภายในโรงเรือนเพาะเห็ด	6
2.1.1 Raspberry Pi 3 Model B	6
2.1.2 Ultrasonic Mist Maker Fogger	8
2.1.3 Humidity and Temperature Sensor – DHT22	9
2.1.4 LCD Display	10
2.1.5 Grow LED Strip	11
2.1.6 Waterproof Fan	12
2.2 Service ที่ใช้ใน Platform	13
2.2.1 NodeJS	13
2.2.2 MQTT	13
2.2.3 Mosca	14
2.2.4 MySQL	14
2.3 Web Application	15
2.3.1 HTML	15

สารบัญ(ต่อ)

หน้า

2.4 Literature Review	16
2.4.1 TEMPERATUER & HUMIDTY CONTROL SYSTEM FOR MUSHROOM. 16	
2.4.2 WONDERFUL MUSHROOM PLANT	17
2.4.3 Automatic Temperature and Humidity control system by using Fuzzy Logic Algorithm for Mushroom nursery	18
บทที่ 3 การวิเคราะห์ และออกแบบ	20
3.1 ความต้องการของระบบ	20
3.2 โครงสร้างของระบบ	20
3.3 Sequence Diagram	22
3.3.1 MQTT Open Connection (1).....	23
3.3.2 Web Browsing (2)	23
3.3.3 Subscribe (3)	23
3.3.3 Publish (topic: logging) (4)	23
3.3.3 Publish (topic: profile settings) (5)	23
3.4 Use Case Diagram	24
3.4.1 รายละเอียด Use Case	25
3.5 Database Schema	29
3.5.1 user	29
3.5.2 farm_owner	30
3.5.3 farm	30
3.5.3 device	30
3.5.4 log.....	30
3.6 MQTT Design.....	32
3.5.1 profile settings	32
3.5.1 logging.....	32
บทที่ 4 การทดลองและผลการทดลอง.....	33
4.1 สร้างตู้จำลองโรงเรือนเพาะเห็ด.....	33
4.1.1 วัตถุประสงค์.....	33

สารบัญ(ต่อ)

	หน้า
4.1.2 วิธีการทดลอง	33
4.1.3 ผลการทดลอง	33
4.2 การทดลองรับ-ส่งข้อมูลระหว่าง Raspberry Pi และ Platform โดยใช้ MQTT Protocol35	
4.2.1 วัตถุประสงค์.....	35
4.2.2 วิธีการทดลอง	35
4.2.3 ผลการทดลอง	36
4.3 การดึงค่าจาก Back-End ที่ติดต่อกับ MySQL มาแสดงผลบน Web Application	36
4.3.1 วัตถุประสงค์.....	36
4.3.2 วิธีการทดลอง	36
4.3.3 ผลการทดลอง	36
4.4 การควบคุมระบบต่างๆภายในโรงเรือนผ่าน Web Application.....	37
4.4.1 วัตถุประสงค์.....	37
4.5 การควบคุมระบบต่างๆภายในโรงเรือนผ่านจอ LCD หน้าโรงเรือน	37
4.5.1 วัตถุประสงค์.....	37
4.5.2 วิธีการทดลอง	37
4.5.1 ผลการทดลอง	40
4.4.2 วิธีการทดลอง	40
4.4.3 ผลการทดลอง	43
4.6 การทดลองเพาะเห็ดโดยใช้ Platform ในการควบคุมสภาพแวดล้อมภายในตู้จำลอง โรงเรือนเพาะเห็ด	44
4.6.1 วัตถุประสงค์.....	44
4.6.2 วิธีการทดลอง	44
4.6.3 ผลการทดลอง	47
บทที่ 5 สรุปผล	49
5.1 ผลสรุปของโครงการ	49
5.1.1 ส่วนของผู้จำลองโรงเพาะเห็ด	49
5.1.2 ส่วนของ Platform.....	49
5.1.3 ส่วนของ Web Application	49
5.2 ปัญหาและอุปสรรค.....	50

สารบัญ(ต่อ)

	หน้า
5.2.1 ส่วนของผู้จำลองโรงเพาะเห็ด	50
5.2.2 ส่วนของ Platform.....	50
5.1.3 ส่วนของ Web Application	50
5.3 แนวทางพัฒนาต่อ	51
5.3.1 ส่วนของผู้จำลองโรงเพาะเห็ด	51
5.3.2 ส่วนของ Platform.....	51
5.3.3 ส่วนของ Web Application	51
บรรณานุกรม	52
ภาคผนวก ก Source Code	53
Platform	53
main-server.js	53
mosca-server.js.....	62
mysql.js	63
web-backend.js.....	63
Raspberry Pi	64
main-client.js	64
mqtt-client.js.....	66
lcd-display.js	67
sensor.js	68
profile.json.....	73
docker-compose.yaml	74
ภาคผนวก ข Deployment	75
Technology Requirement	75
Hardware	75
Operating System	75
Engine.....	75
Language	75
Installation Method Command	76

สารบัญ(ต่อ)

	หน้า
Platform.....	76
Raspberry Pi	76
Create Database Command	77
ภาคผนวก ก Usage.....	78
Platform	78
Microcontroller	81
Web Application.....	83

สารบัญตาราง

ตาราง	หน้า
1.1 แผนการดำเนินงาน.....	5
2.1 หลักการทำงานของ Fuzzy Logic.....	18
3.1 Login Use Case.....	25
3.2 Logout Use Case	26
3.3 View Dashboard Use Case.....	26
3.4 View Summary Use Case	27
3.5 Set Profile Use Case.....	27
3.6 Manage User/Device Use Case.....	28
3.7 Set Profile Use Case.....	28
4.1 อัตราการเจริญเติบโตของเห็ด.....	45

สารบัญรูป

รูป	หน้า
1.1 IoT.....	1
1.2 Model ตู้จำลอง โรงเรือนเพาะเห็ด	3
1.3 Hardware I/O Diagram.....	4
1.4 System Diagram.....	4
2.1 Raspberry Pi 3 Model B.....	6
2.2 Ultrasonic Mist Maker Fogger.....	8
2.3 DHT22	9
2.4 LCD Display	10
2.5 Grow LED Strip	11
2. 6 Waterproof Fan	12
2.7 MQTT	13
2.8 Web Application	15
2.9 รายละเอียดของส่วนประกอบภายใน โรงเรือนจำลอง.....	16
2.10 โรงเรือนเพาะเห็ดจำลอง.....	16
2.11 โรงเรือนเพาะเห็ดจำลอง.....	17
3.1 System Diagram.....	21
3.2 Hardware I/O Diagram.....	21
3.3 Sequence Diagram	22
3.4 Use Case Diagram.....	24
3.5 Database Schema	29
4.1 ส่วนสำหรับเพาะเห็ด.....	34
4.2 ส่วนแผงควบคุม	34
4.3 Dashboard	37
4.4 หน้า Login.....	40
4.5 หน้า Dashboard.....	41
4.6 หน้า Profile	41
4.7 หน้า Edit User	42
4.8 หน้า Edit Device	42

สารบัญรูป(ต่อ)

รูป	หน้า
4.9 หน้า Edit Farm	43
4.10 alert message	43
4.11 Dashboard สำหรับจอ LCD(1)	38
4.12 Dashboard สำหรับจอ LCD(2)	39
4.13 Dashboard สำหรับจอ LCD(3)	39
4.14 เห็นทั้ง 4 ชนิดในตู้เพาะเห็ด	44
4.15 คำ Profile สำหรับการเพาะเห็ด	45
4.16 นางรมฮังการี	47
4.17 เห็นนางนวลสีชมพู	47
4.18 เห็นนางฟ้าภูฐาน	48
4.19 เห็นเป่าฮือ	48
ค.1 ไฟล์ CentOS 7 - Platform.vhdx	78
ค.2 เพิ่ม Virtual Machine	78
ค.3 Virtual Machine Interface.....	79
ค.4 Network Configuration Using a Text User Interface (nmtui)	79
ค.5 Bash in NodeJS Container	80
ค.6 Run Service on Bash in NodeJS Container	80
ค.7 ด้านซ้ายมุมซ้ายล่างของตู้จำลอง	81
ค.8 ด้านบนของตู้จำลอง	81
ค.9 Raspberry Pi Terminal Interface.....	82
ค.10 Network Configuration.....	82

บทที่ 1

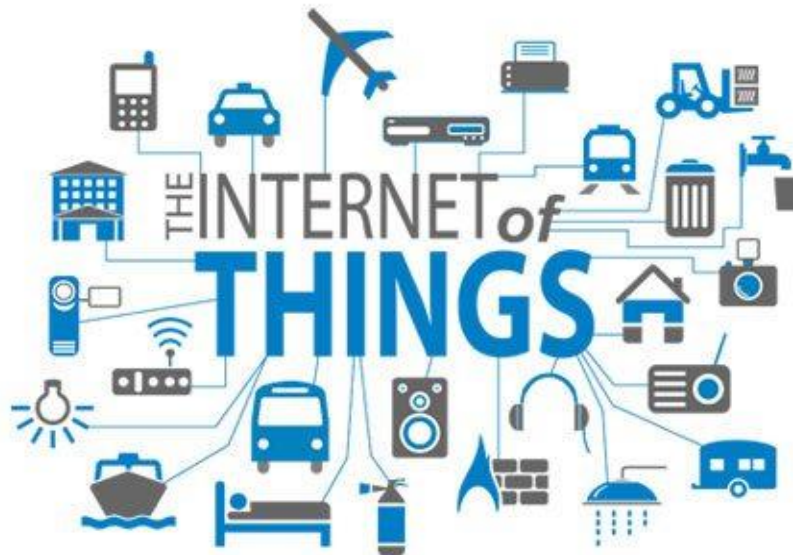
บทนำ

1.1 ที่มาและความสำคัญของโครงการ

ในปัจจุบันการเพาะเห็ดในโรงเรือนนั้นมีอุณหภูมิและความชื้นที่เหมาะสมในการออกดอกแตกต่างกันออกไปสำหรับเห็ดแต่ละประเภท การควบคุมระบบในโรงเรือนโดยมนุษย์นั้นอาจทำให้เกิดปัญหาเนื่องจากการดูแลที่ไม่ทั่วถึงได้ ทำให้เกิดผลกระทบกับคุณภาพของผลผลิตให้ไม่เป็นไปตามที่ต้องการได้

โครงการนี้จึงได้นำเอา Microcontroller มาช่วยในการควบคุมอุณหภูมิและความชื้นในโรงเรือนให้มีปริมาณที่เหมาะสมกับการเจริญเติบโตของเห็ด เพื่อให้เห็ดเจริญเติบโตในสภาพแวดล้อมที่ดีที่สุด ทำให้เพิ่มผลผลิตได้มากขึ้นและลดข้อผิดพลาดจากการควบคุมดูแล และใช้ควบคู่กับระบบ IoT Platform เพื่อที่สามารถแสดงข้อมูลต่างๆและจัดการระบบภายในโรงเรือนได้

แนวคิด Internet of Things ถูกคิดขึ้นในปี 1999 ต่อมาหลังจาก 2000 เป็นต้นไปได้มีอุปกรณ์อิเล็กทรอนิกส์ที่มีการใช้คำว่า Smart ออกมาจำนวนมาก เช่น smart device, smart home และ smart network ซึ่งทั้งหมดล้วนมีโครงสร้างพื้นฐานที่สามารถเชื่อมต่อกับโลกอินเทอร์เน็ตได้ตามรูป 1.1 ซึ่งทำให้อุปกรณ์เหล่านั้นสามารถสื่อสารกันเองได้ด้วยเช่นกัน



รูป 1.1 IoT

โดย Microcontroller ส่งข้อมูลต่างๆทั้งอุณหภูมิและความชื้นขึ้นไปเก็บไว้บนเซิร์ฟเวอร์ แล้วเรายังนำข้อมูลเหล่านั้นไปแสดงผลผ่านทาง IoT Platform โดยสามารถจัดการและควบคุมดูแลระบบต่างๆในโรงเรือนได้ผ่านทาง Web Application

1.2 วัตถุประสงค์ของโครงการ

- 1) เพื่อศึกษาและประยุกต์ใช้งาน Microcontroller
- 2) เพื่อจำลองระบบโรงเรือนที่มีการเพาะเลี้ยงเห็ดที่ควบคุมโดย Microcontroller
- 3) เพื่อออกแบบระบบควบคุมอุณหภูมิและความชื้นที่เหมาะสมกับเห็ดแต่ละชนิด
- 4) เพื่อศึกษาและประยุกต์ใช้งาน IoT Platform
- 5) เพื่อออกแบบระบบ Web Application สำหรับแสดงสถานะและควบคุมระบบภายในโรงเรือนเพาะเห็ด

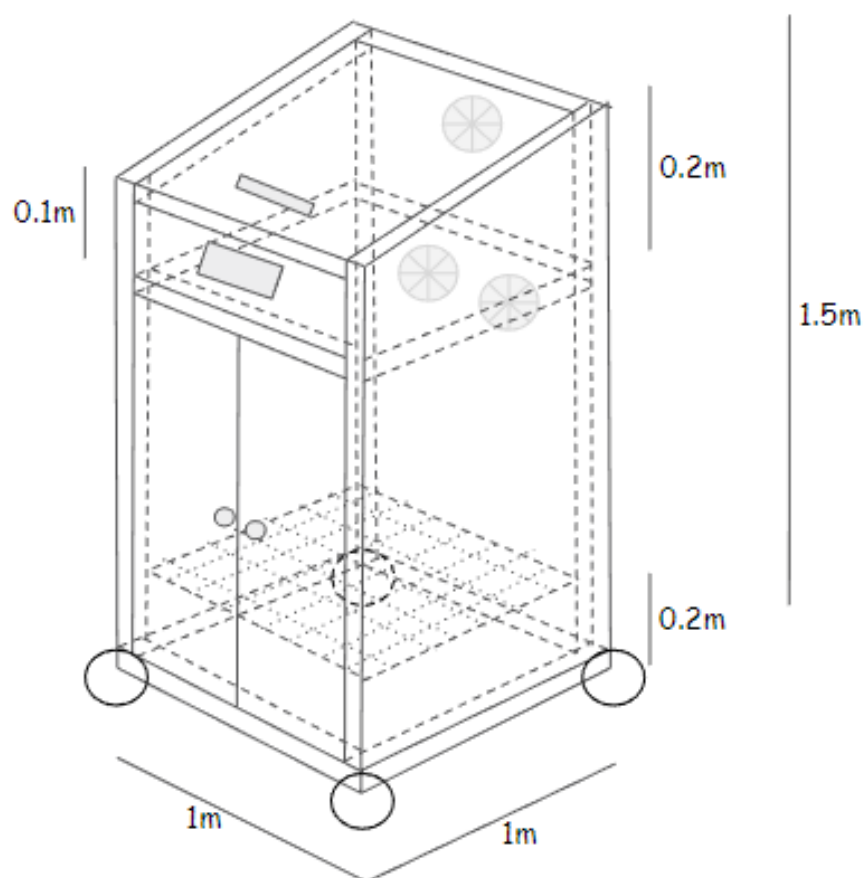
1.3 ประโยชน์ที่คาดว่าจะได้รับ

- 1) เข้าใจระบบการทำงานทั้งหมดตั้งแต่การเพาะเลี้ยงเห็ด, Microcontroller, IoT Platform และ Web Application
- 2) นำระบบควบคุมอุณหภูมิและความชื้นไปประยุกต์ใช้ในการปลูกพืชและเห็ดชนิดอื่นๆ

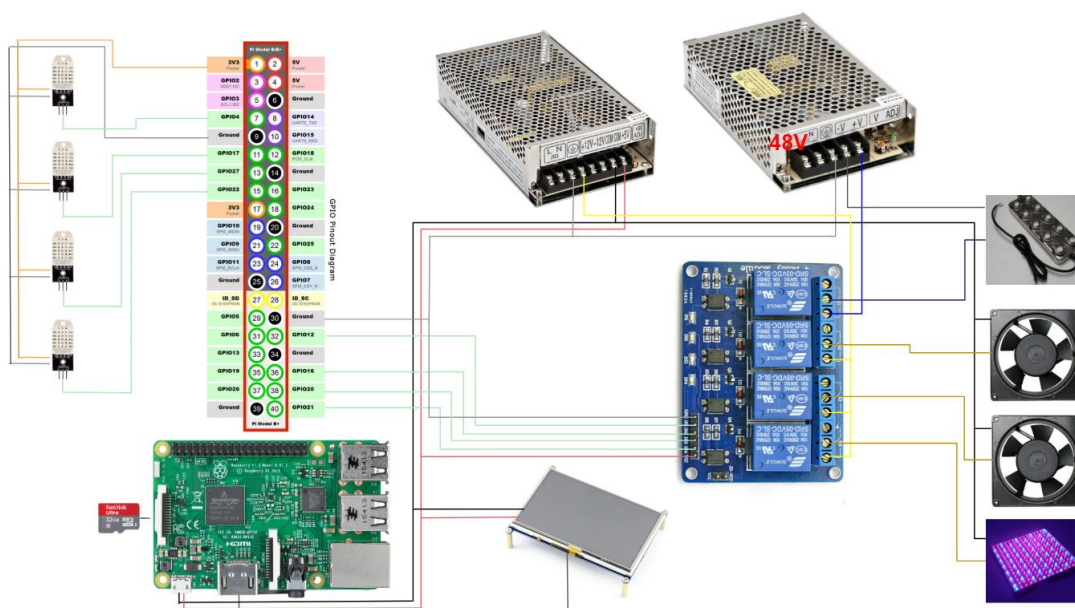
1.4 ขอบเขตของโครงการ

- 1) ออกแบบตู้เพาะเห็ดตามรูป 1.2 ที่สามารถควบคุมความชื้น อุณหภูมิ แสง ด้วย Profile ต่างๆของเห็ดแต่ละชนิด ด้วย Microcontroller โดยมี Diagram การเชื่อมต่อตามรูป 1.3 และสามารถส่งข้อมูลไปยังระบบ IoT Platform ที่ได้พัฒนาไว้
- 2) นำไปจัดแสดงในนิทรรศการและงานกิจกรรมต่างๆได้ โดยตู้เพาะเห็ดสามารถเคลื่อนย้ายได้ง่ายด้วยล้อเลื่อน เพื่อเป็นความรู้และแนวคิดให้กับผู้ชมในงาน ทั้งนี้สามารถเป็นชื่อเสียงให้กับสถาบันและภาควิชา
- 3) จากรูป 1.4 จะเห็นได้ว่าระบบมีการติดต่อกันระหว่าง ตู้เพาะเห็ด IoT Platform และผู้ใช้งานจาก Web Application โดยมีรายละเอียดดังนี้
 - a) ตู้เพาะเห็ดสามารถทำงานได้แบบ Standalone คือไม่ได้ติดต่อกับ Platform ก็สามารถทำงานตาม Profile ของเห็ดแต่ละชนิด และรับส่งข้อมูลไปยัง Platform ภายหลังที่สามารถเชื่อมต่อ Platform ได้
 - b) IoT Platform เป็นตัวสั่งการ และรับข้อมูลต่างๆจากตู้เพาะเห็ด ด้วย Protocol ที่กำหนดไว้

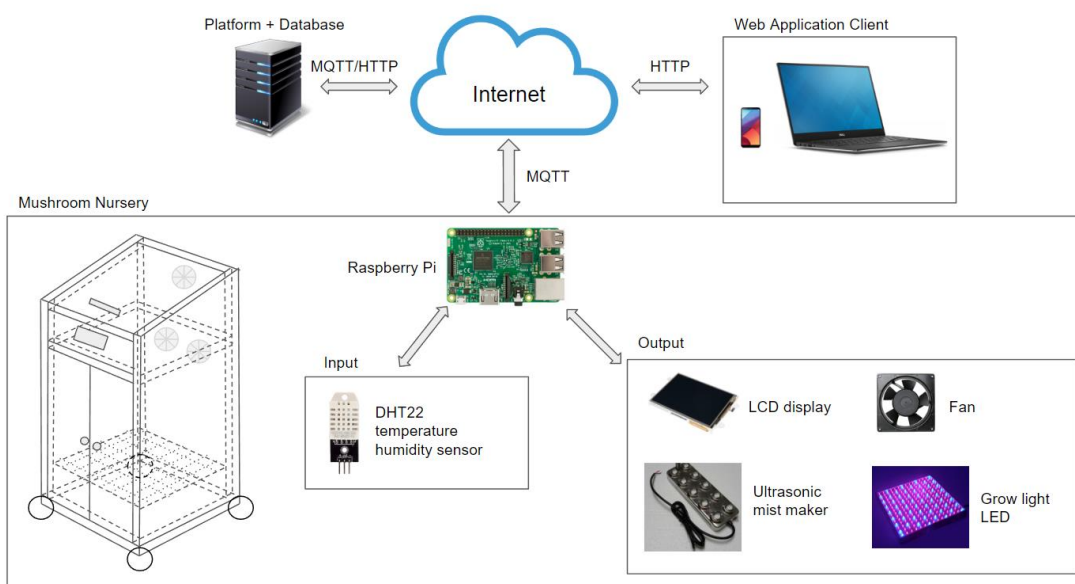
- c) มี Web Appilcation ที่ติดต่อกับผู้ใช้งานผ่าน HTTP เพื่อดูค่าสถานะและสั่งการได้



รูป 1.2 Model ตู้จำลองโรงเรือนเพาะเห็ด



រូប 1.3 Hardware I/O Diagram



រូប 1.4 System Diagram

บทที่ 2

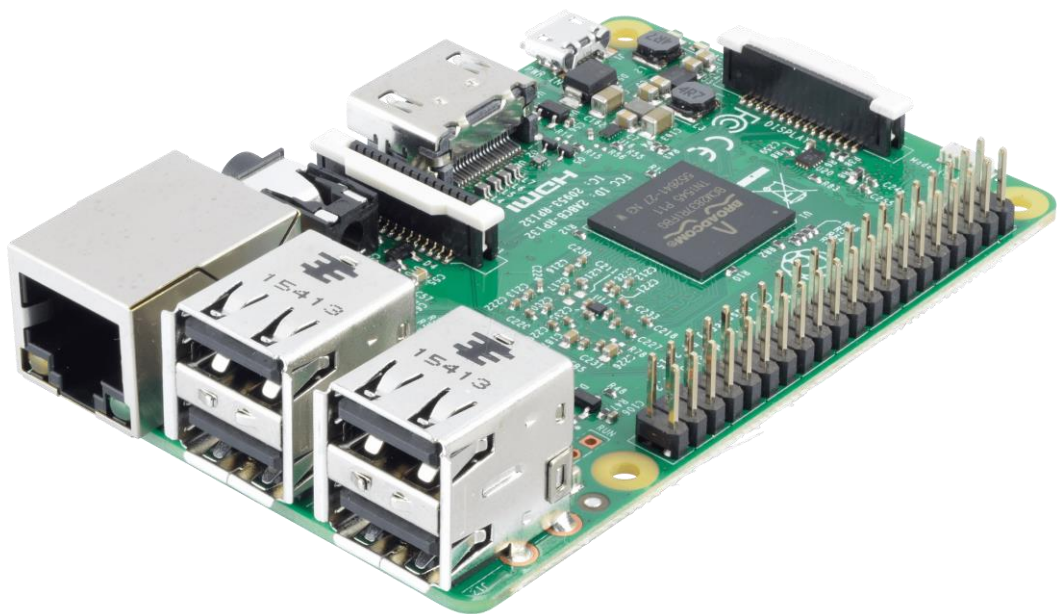
ทฤษฎีที่เกี่ยวข้อง

บทนี้กล่าวถึงทฤษฎีต่างๆ ที่ได้นำมาใช้ในโครงงานนี้ ซึ่งสามารถแบ่งได้ทั้งหมด 3 ส่วน คือ ส่วนของโรงเรือนจำลองและ Microcontroller ส่วนของ Platform และสุดท้ายคือส่วนของ Web Application แต่ละส่วนประกอบด้วยทฤษฎีต่างๆดังนี้

2.1 Microcontroller และอุปกรณ์ต่างๆภายในโรงเรือนเพาะเห็ด

Microcontroller ที่นำมาใช้ควบคุมอุปกรณ์ต่างๆภายในโรงเรือนเพาะเห็ดคือ Raspberry Pi 3 Model B ตามรูป 2.1 โดยอุปกรณ์ภายในโรงเรือนเพาะเห็ดประกอบด้วย เครื่องสร้างหมอกแบบ Ultrasonic ใช้สร้างกลุ่มหมอกเพื่อเพิ่มความชื้นภายในโรงเรือน เซนเซอร์ DHT 22 ใช้วัดความชื้นและอุณหภูมิ เซนเซอร์ ESEN171 ใช้วัดระดับน้ำ LCD Display สำหรับการแสดงผล Grow LED Strip ใช้เพื่อช่วยให้เห็ดเจริญเติบโตได้ดีขึ้น และพัฒนาระบบอากาศแบบกักน้ำ

2.1.1 Raspberry Pi 3 Model B



รูป 2.1 Raspberry Pi 3 Model B

Raspberry Pi ถูกออกแบบมาเพื่อให้เป็นคอมพิวเตอร์ขนาดเล็กที่สามารถเชื่อมต่อกับ จอ เม้าส์ และคีย์บอร์ดได้ สามารถทำงานได้เหมือนคอมพิวเตอร์ทั่วไป เช่น การใช้งาน Internet หรือเล่นไฟล์วีดีโอ

Raspberry Pi สามารถรองรับระบบปฏิบัติการ Linux ได้หลายระบบ เช่น Raspbian Pidora และ Arch Linux โดยติดตั้งลงบน SD Card โดย Raspberry Pi สามารถนำไปประยุกต์ใช้ทำสิ่งต่างๆได้มากมาย เช่น นำไปทำเป็นหุ่นยนต์ เครื่องเล่นดนตรี

Raspberry Pi 3 Model B เป็นรุ่นที่ 3 ของ Raspberry Pi ซึ่งมีขนาดของบอร์ดเท่ากับบัตรเครดิตเท่านั้น โดยที่มีความสามารถในการประมวลผลมากกว่า Raspberry Pi รุ่นแรกถึง 10 เท่า

คุณสมบัติของ Raspberry Pi 3 Model B มีดังนี้

- Broadcom BCM2837 chipset running at 1.2 GHz
- 64-bit quad-core ARM Cortex-A53
- 802.11 b/g/n Wireless LAN
- Bluetooth 4.1
- Dual core Videocore IV® Multimedia co-processor
- 1 GB LPDDR2 memory
- Supports all the latest ARM GNU/Linux distributions and Windows 10 IoT
- microUSB connector for 2.5 A power supply
- 1 x 10/100 Ethernet port
- 1 x HDMI video/audio connector
- 1 x RCA video/audio connector
- 4 x USB 2.0 ports
- 40 GPIO pins
- Chip antenna
- DSI display connector
- microSD card slot
- Size 85*56*17 mm

2.1.2 Ultrasonic Mist Maker Fogger



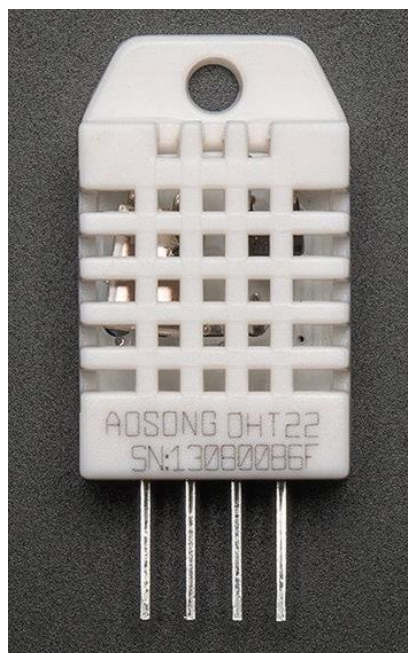
รูป 2.2 Ultrasonic Mist Maker Fogger

Ultrasonic Mist Maker Fogger จากรูป 2.2 ใช้ในการสร้างหมอกโดยอาศัยคลื่น Ultrasonic ในการทำให้น้ำเกิดการแตกตัวกลายเป็นกลุ่มหมอก ทำให้สามารถควบคุมความชื้นให้มีความเหมาะสมต่อการเจริญเติบโตของพืชได้

คุณสมบัติของ Ultrasonic Mist Maker Fogger มีดังนี้

- Power 250 w
- Electric Current 5 A
- Size 252*92*67 mm
- Water-depth : 60-80 mm
- Particle size : 0.1~0.5u
- Piezoceramics disc 1.7 MHZ
- Humidification amount : 4.5 kg/h
- Mist output tolerance $\pm 10\%$

2.1.3 Humidity and Temperature Sensor – DHT22



รูป 2.3 DHT22

DHT22 จากรูป 2.3 เป็นเซนเซอร์ที่ใช้วัดความชื้นและอุณหภูมิภายในตัวเดียว ถูกออกแบบมาให้ใช้งานง่ายและมีความแม่นยำสูง โดยค่าที่วัดได้เป็นรูปแบบ Digital โดยสามารถเชื่อมต่อกับ Raspberry Pi ด้วย 4 Pin connector

คุณสมบัติของ DHT22 มีดังนี้

- Low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 0-100% humidity readings with 2-5% accuracy
- Good for -40 to 80°C temperature readings $\pm 0.5^{\circ}\text{C}$ accuracy
- No more than 0.5 Hz sampling rate (once every 2 seconds)
- Body size 27mm x 59mm x 13.5mm (1.05" x 2.32" x 0.53")
- 4 pins, 0.1" spacing
- Weight (just the DHT22): 2.4g

2.1.4 LCD Display



รูป 2.4 LCD Display

จอ LCD Display จากรูป 2.4 เป็นจอสำหรับแสดงผลแบบสัมผัสขนาด 5 นิ้ว มีความละเอียด 800 * 480 โดยส่วนจอแสดงผลสามารถเชื่อมต่อกับ Raspberry Pi โดยผ่านพอร์ต HDMI และแผ่นทัชสกรีนแบบ Resistive เชื่อมต่อผ่านพอร์ต USB

ก่อนการเริ่มต้นใช้งานจอ LCD Display จำเป็นต้องเข้าไปปรับค่าความละเอียดของหน้าจอให้ตรงก่อนซึ่งต้องเพิ่มค่า Config ไว้ท้ายไฟล์ /boot/config.txt ใน SD Card ที่ติดตั้งระบบปฏิบัติการไว้ดังนี้

โปรแกรม 2.1 การตั้งค่า Config สำหรับจอ LCD Display

```
max_usb_current=1
hdmi_group=2
hdmi_mode=87
hdmi_cvt 800 480 60 6 0 0 0
hdmi_drive=1
```

2.1.5 Grow LED Strip



รูป 2.5 Grow LED Strip

ในแสงอาทิตย์ประกอบด้วย Color Spectrum ทุกสี โดยที่แสงแต่ละสีก็มีความยาวคลื่นที่ต่างกันไป และความยาวคลื่นของแสงที่จำเป็นในการเจริญเติบโตของพืชนั้นมีเพียงแสงสีแดงกับแสงสีน้ำเงินเท่านั้น โดยที่ความยาวคลื่นของแสงสีแดงอยู่ในช่วง 632nm~660nm และความยาวคลื่นของแสงสีน้ำเงินอยู่ในช่วง 440nm~460nm ซึ่งความยาวคลื่นทั้ง 2 ช่วงนี้สามารถช่วยให้พืชเจริญเติบโตได้ดีขึ้น ทำให้สามารถใช้ LED สีแดงผสมกับ LED สีน้ำเงิน ในอัตราส่วน สีแดง 80% ต่อสีน้ำเงิน 20% ตามรูป 2.5 จะทำให้เกิดเป็นแสงที่มีความยาวคลื่นตามที่พืชต้องการในการเจริญเติบโตได้ ทำให้สามารถปลูกพืชในที่ร่มได้

คุณสมบัติของ Grow LED Strip มีดังนี้

- Lamp Power 12 W
- Input Voltages 12 V / 24 V
- Lamp Luminous Flux 1080 - 1200 lm
- Lamp Luminous Efficiency 80 lm/w
- Working Temperature -20 - 60°C
- LED Wavelength Red:660nm, Blue:440nm

2.1.6 Waterproof Fan



รูป 2. 6 Waterproof Fan

พัดลมระบายอากาศแบบกันน้ำ ตามรูป 2.6 มีคุณสมบัติดังนี้

- Power 2.28 W
- Power Current 190 mA
- Input Voltages 12 V
- Air Volume 50.3 CFM
- Speed 4100 RPM
- Static Pressure 0.27 Inch-H₂O
- Noise 41.1 dB(A)
- Working Temperature -10 - 70°C

2.2 Service ที่ใช้ใน Platform

Platform นี้ใช้ NodeJS เป็น Service หลักในระบบ โดยมี MQTT Protocol library อยู่ภายใน NodeJS ใช้ในการติดต่อกันระหว่าง Microcontroller, Platform และ Web Application และมี MySQL ในการจัดเก็บและจัดการข้อมูล

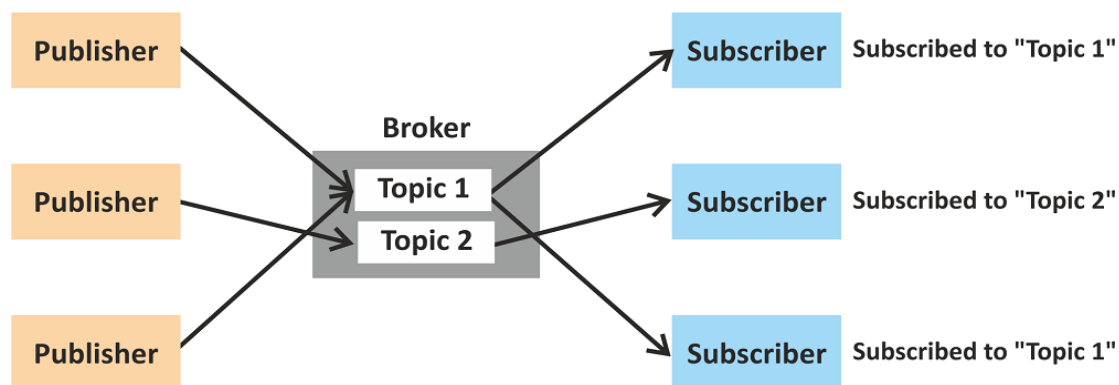
2.2.1 NodeJS

NodeJS คือ การเขียนโปรแกรมด้วยภาษา JavaScript ที่ฝั่ง Server และมีการรันด้วย Chrome's V8 JavaScript engine ซึ่งทำให้ NodeJS มีจุดเด่นอยู่ที่ความเร็วในการประมวลผล มีความสามารถ Scalable network applications และสามารถรันได้บนหลายระบบปฏิบัติการ

NodeJS มีการทำงานแบบ event-driven คือ การขับเคลื่อนด้วยเหตุการณ์ และ NodeJS ยังมาพร้อมกับเทคโนโลยี Non-Blocking I/O คือ ไม่จำเป็นต้องรอการทำงานของคำสั่งอื่นๆที่ใช้เวลานานให้เสร็จก่อน สามารถข้ามไปทำงานคำสั่งถัดไปได้เลย นอกจากนั้น NodeJS ยังมีความเป็น Real-time Application อีกด้วย

2.2.2 MQTT

MQTT เป็น Protocol ที่ถูกออกแบบมาให้ใช้ในการเชื่อมต่อระหว่างอุปกรณ์กับอุปกรณ์ เพื่อสนับสนุนเทคโนโลยี IoT โดยใช้หลักการ Client / Server มี Topology แบบ hub and spoke โดยฝั่ง Platform ถูกเรียกว่า Broker มีหน้าที่จัดการการรับ-ส่งของข้อมูลระหว่าง Client และใช้ Topic เป็นตัวอ้างอิง ฝั่ง Client เป็นได้ทั้ง Publisher และ Subscriber ตามรูป 2.8



รูป 2.7 MQTT

หน้าที่ของส่วนประกอบต่างๆของ MQTT คือ

- 1) Broker ทำหน้าที่เป็นตัวกลางคอยจัดการกับข้อความที่ได้จาก Publisher แล้วส่งไปให้ Subscriber โดยอิงจาก Topic ที่ Subscriber ได้ Subscribe ไว้
- 2) Publisher และ Subscriber ทำการเชื่อมต่อไปยัง Broker ซึ่งเชื่อมต่อได้ 2 วิธี คือ Persistent ที่สร้าง Session ค้างไว้ตลอดเวลาเพื่อติดต่อกับ Broker อีกวิธีคือ Transient ที่ทำให้ Broker ไม่สามารถติดตามสถานะได้
- 3) Topic เปรียบเสมือน Address บน Broker ที่ Client ทำการเชื่อมต่อเพื่อรับส่งข้อความต่างๆระหว่างกัน

2.2.3 Mosca

Mosca เป็น MQTT Broker ที่พัฒนาโดยใช้ NodeJS ทำให้ Mosca มีความเร็วในการอ่าน-เขียนข้อมูลได้อย่างรวดเร็ว โดยสามารถใช้งานแบบ Standalone หรือใช้งาน Embedded ใน NodeJS Application อื่นๆก็ได้

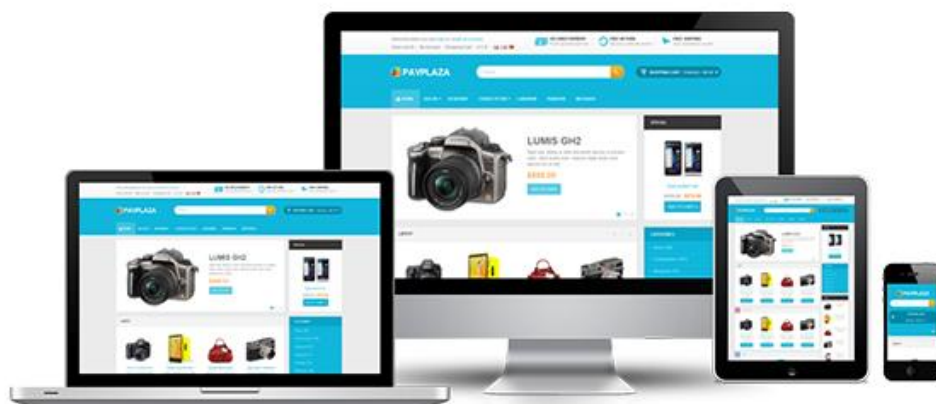
2.2.4 MySQL

ฐานข้อมูลมีลักษณะเป็นโครงสร้างของการเก็บรวบรวมข้อมูล การที่เข้าถึงหรือเปลี่ยนแปลงข้อมูลในฐานข้อมูลจึงจำเป็นต้องอาศัยระบบจัดการฐานข้อมูลเป็นตัวกลางในการจัดการข้อมูล และรองรับการทำงานของ Application อื่นๆที่ต้องการใช้งานข้อมูลนั้นๆ เพื่อความสะดวกในการจัดการกับข้อมูลจำนวนมาก MySQL จึงทำหน้าที่เป็นทั้งตัวฐานข้อมูลและระบบจัดการฐานข้อมูล

MySQL เป็นระบบจัดการฐานข้อมูลแบบ Relational ซึ่งทำการเก็บข้อมูลทั้งหมดในแบบของตาราง ทำให้ทำงานได้อย่างรวดเร็วและมีความยืดหยุ่น นอกจากนั้นตารางแต่ละตารางสามารถเชื่อมโยงเข้าหากัน ทำให้สามารถรวบรวมหรือจัดกลุ่มของข้อมูลได้ตามที่ต้องการโดยอาศัยภาษา SQL

จุดเด่นของ MySQL คือ สามารถทำงานได้อย่างรวดเร็ว น่าเชื่อถือ ใช้งานได้ง่าย และเป็น Open Source ซึ่งการพัฒนา ยังคงดำเนินอยู่อย่างต่อเนื่อง ทำให้มีฟังก์ชันการทำงานใหม่ๆ ที่อำนวยความสะดวกแก่ผู้ใช้งานเพิ่มขึ้น รวมไปถึงการปรับปรุงประสิทธิภาพ ความเร็วในการทำงาน และความปลอดภัย

2.3 Web Application



รูป 2.8 Web Application

Web Application คือ Application ที่ถูกเขียนขึ้นมาเพื่อเป็น Browser เพื่อใช้งาน Webpage ต่างๆ ตัวโปรแกรมของ Web Application ถูกติดตั้งไว้ที่ Platform เพื่อให้บริการกับ Client ซึ่งฝั่ง Client สามารถเป็น Device ได้หลากหลายตามรูป 2.9 และฝั่ง Client ไม่จำเป็นต้องติดตั้งโปรแกรมเพิ่มเติมนอกจาก Web Browser และข้อมูลที่ส่งหากันระหว่าง Client กับ Server บนมาตรฐาน HTTP ทั่วไป

2.3.1 HTML

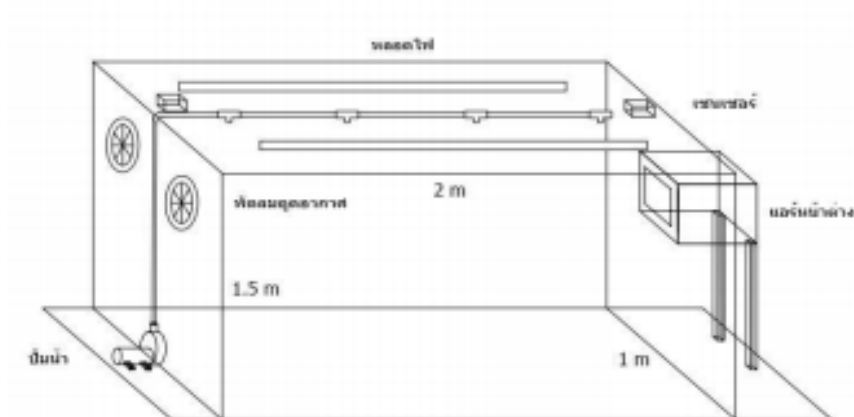
HTML ย่อมาจาก Hyper Text Markup Language เป็นภาษาหลักที่ใช้ในการเขียน Webpage HTML ถูกพัฒนามาจากภาษา SGML และได้มีการพัฒนาต่อมาเรื่อยๆ และกำหนดมาตรฐานโดย W3C มีโครงสร้างการเขียนโดยอาศัย Tag มาเพื่อควบคุมการแสดงผลของสิ่งต่างๆ ที่แสดงผลอยู่บนหน้า Webpage และแต่ละ Tag สามารถมี Attribute อื่นๆ เพิ่มเติมได้

2.4 Literature Review

2.4.1 TEMPERATUER & HUMIDTY CONTROL SYSTEM FOR MUSHROOM

งานวิจัยนี้ได้สร้างโรงเรือนเพาะเห็ดจำลองและทำระบบควบคุมอุณหภูมิและความชื้นที่ควบคุมด้วย Arduino MEGA 2560 R3 โดยใช้เซนเซอร์ DHT 22 และนำค่าที่วัดได้ไปแสดงผลผ่านจอ LCD Display

ภายในตู้เพาะเห็ดใช้เครื่องพ่นหมอกแบบสเปรย์จำนวน 3 ตัว ในการให้ความชื้น ใช้แอร์หน้าต่างในการทำมาเย็น และพัดลมสำหรับระบายอากาศจำนวน 2 ตัว ตามรูป 2.10 และรูป 2.11



รูป 2.9 รายละเอียดของส่วนประกอบภายในโรงเรือนจำลอง



รูป 2.10 โรงเรือนเพาะเห็ดจำลอง

หลักการทำงานของอุปกรณ์ต่างๆ คือ เมื่อทำการเปิดเครื่อง เซนเซอร์ตรวจจับอุณหภูมิและความชื้น จากนั้นนำไปแสดงผลผ่านจอ LCD Display และทำการปรับค่าอุณหภูมิและความชื้นตามต้องการ จากนั้น Arduino ควบคุมการทำงานของแอร์และเครื่องพ่นหมอกให้ได้ค่าสถานะต่างๆ ใกล้เคียงตามค่าที่ตั้งไว้

2.4.2 WONDERFUL MUSHROOM PLANT

งานวิจัยนี้ได้ทำระบบควบคุมอุณหภูมิและความชื้นที่ควบคุมด้วย Arduino UNO R3 โดยใช้เซนเซอร์ AMT1001 ซึ่งสามารถกำหนดค่าอุณหภูมิและความชื้นตามที่ต้องการได้และนำไปแสดงผลผ่านจอ LCD

ภายในตู้เพาะเห็ดได้ใช้เครื่องพ่นหมอกแบบ Ultrasonic ในการให้ความชื้นภายในตู้เพาะเห็ด ตามรูปที่ 2.12



รูป 2.11 โรงเรือนเพาะเห็ดจำลอง

2.4.3 Automatic Temperature and Humidity control system by using Fuzzy Logic

Algorithm for Mushroom nursery

งานวิจัยนี้ได้ทำระบบควบคุมอุณหภูมิและความชื้นที่ควบคุมด้วย Arduino UNO R3 โดยใช้เซนเซอร์ DHT 11 และใช้หลัก Fuzzy logic ในการควบคุมเครื่องพ่นหมอก พัดลม และหลอดไฟ

ภายในโรงเพาะเห็ดใช้เครื่องพ่นหมอกแบบสเปรย์จำนวน 1 ตัวในการให้ความชื้น พัดลมสำหรับระบายอากาศจำนวน 2 ตัว และหลอดไฟจำนวน 3 หลอดในการให้ความร้อนและแสงสว่าง

หลักการทำงานคือ เริ่มต้นทำการวัดค่าอุณหภูมิและความชื้น จากนั้น Arduino ส่งอุปกรณ์ทั้ง 3 ตัว คือ เครื่องพ่นหมอก พัดลม และหลอดไฟ ให้ทำงานดังตารางที่ 2.1

ตาราง 2.1 หลักการทำงานของ Fuzzy Logic

Input		Output		
อุณหภูมิ	ความชื้น	พัดลม	เครื่องพ่นหมอก	หลอดไฟ
Low	Low	Off	Medium	High
Low	Medium	Off	Low	High
Low	High	Off	Off	Medium
Medium	Low	Off	Off	High
Medium	Medium	Off	Off	Off
Medium	High	Low	Off	Low
High	Low	High	Low	Off
High	Medium	High	Off	Off
High	High	High	Off	Off

จากตารางที่ 2.1 ค่าของ Input เกิดจากสมการทั้ง 3 สมการต่อไปนี้

$$Medium(x) = \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right) \quad (1)$$

$$High(x) = \exp\left(-\frac{(x-m)^2}{2\sigma_1^2}\right) \quad (2) \quad \text{เมื่อ } \sigma_1 = m - \sigma$$

$$Low(x) = \exp\left(-\frac{(x-m)^2}{2\sigma_2^2}\right) \quad (3) \quad \text{เมื่อ } \sigma_2 = m - \sigma_1$$

และค่าของ Output เกิดจากการใช้ Gaussian Membership Function ดังต่อไปนี้

- 1) พัฒนาระบายอากาศมี 2 Fuzzy Sets คือ
 - a) Low ทำการเปิดพัดลมเพียง 1 ตัว
 - b) High ทำการเปิดพัดลมพร้อมกัน 2 ตัว
- 2) เครื่องพ่นหมอกมี 3 Fuzzy Sets คือ
 - a) Low ทำการเปิดหัวพ่นหมอกทั้งหมด 4 หัว
 - b) Medium ทำการเปิดหัวพ่นหมอกทั้งหมด 6 หัว
 - c) High ทำการเปิดหัวพ่นหมอกทั้งหมด 8 หัว
- 3) หลอดไส้มี 3 Fuzzy Sets คือ
 - a) Low ทำการเปิดหลอดไส้เพียง 1 ตัว
 - b) Medium ทำการเปิดหลอดไส้พร้อมกัน 2 ตัว
 - c) High ทำการเปิดหลอดไส้พร้อมกัน 3 ตัว

บทที่ 3

การวิเคราะห์ และออกแบบ

บทนี้อธิบายถึงการวิเคราะห์และการออกแบบระบบ ซึ่งแบ่งออกเป็นผังโรงงานจำลอง ผัง Platform และผัง Microcontroller โดยอธิบายเกี่ยวกับความต้องการของระบบ โครงสร้างของระบบ และ Diagram ต่างๆที่เกี่ยวข้อง

3.1 ความต้องการของระบบ

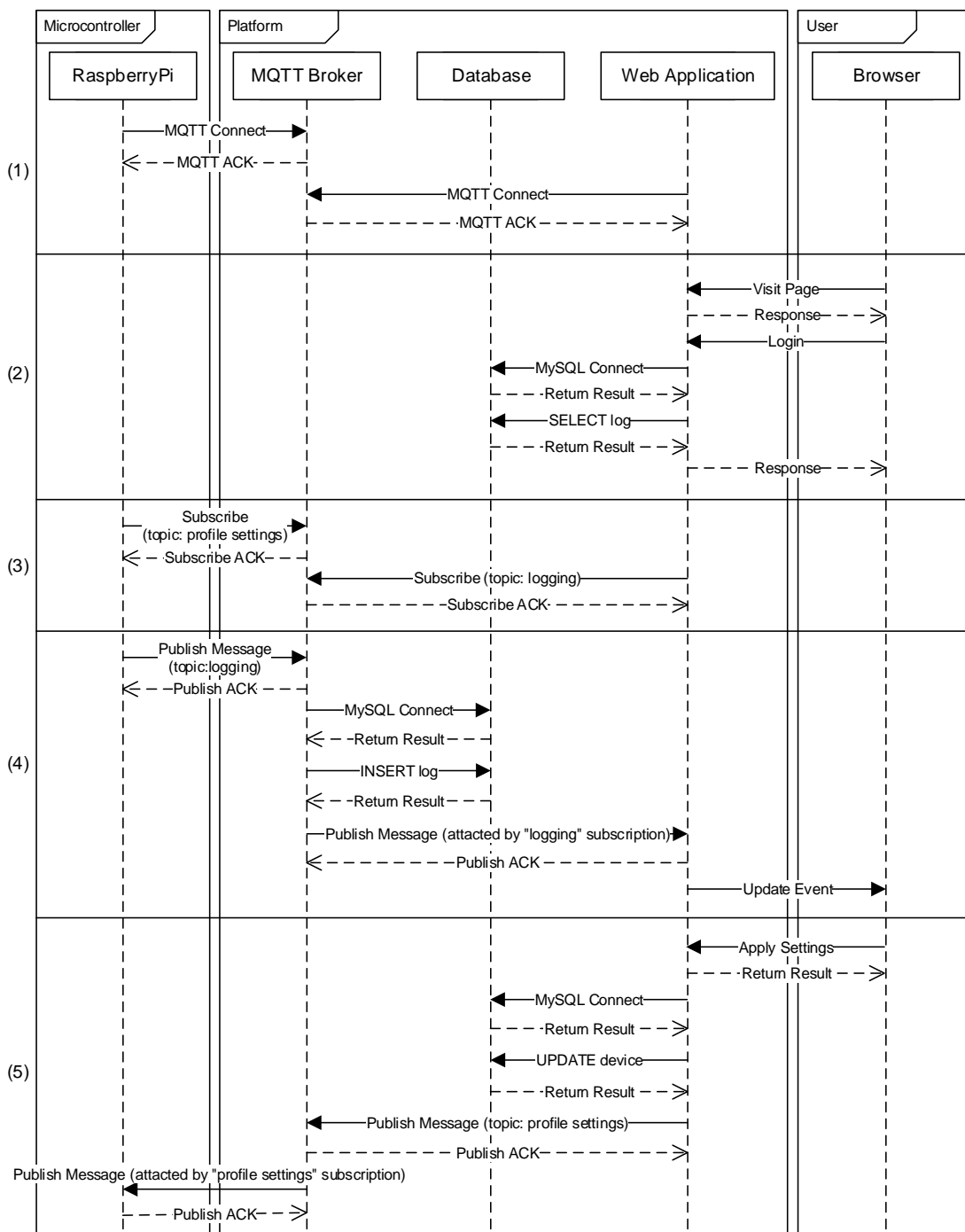
- 1) สามารถตั้งค่าชุดข้อมูลตามความเหมาะสมของเห็ดแต่ละชนิด
- 2) สามารถใช้เซนเซอร์วัดอุณหภูมิ ความชื้น และระดับน้ำภายในถังน้ำ
- 3) สามารถผลิตหมอกไอน้ำ แสง และระบายอากาศได้ตามความเหมาะสม
- 4) สามารถอ่านเขียนข้อมูลลงใน SD Card และส่งข้อมูลไปยัง Platform ผ่าน MQTT Protocol
- 5) มีจอ LCD ขนาด 5 นิ้ว สำหรับแสดงสถานะภายในตู้เห็ด
- 6) มี Web Application ให้ Login สำหรับจัดการและแสดงผลระบบทั้งหมด

3.2 โครงสร้างของระบบ

โครงสร้างของระบบจะแบ่งออกเป็น 2 ส่วน คือ ส่วนของ System Diagram ตามรูป 3.1 และส่วนของ Hardware I/O Diagram ตามรูป 3.2

3.3 Sequence Diagram

Sequence Diagram จะมีรายละเอียดดังรูป 3.3



รูป 3.3 Sequence Diagram

ระบบมีการทำงานแบบ Concurrency และ Event-driven ดังนั้นการเขียนโครงสร้างแบบ Sequence Diagram ทำให้เข้าใจและมีประสิทธิภาพมากกว่า Diagram ชนิดอื่น จากรูปภาพด้านบน แบ่งออกเป็น 5 สถานการณ์ ดังนี้

3.3.1 MQTT Open Connection (1)

เมื่อเริ่มต้น MQTT Broker Service ที่ฝั่ง Platform แล้ว ให้ Microcontroller และ Web Application Service ทำการติดต่อเข้ามาที่ Broker เพื่อใช้ในการรับ-ส่งข้อมูลต่อไป

3.3.2 Web Browsing (2)

ผู้ใช้งานติดต่อเข้ามาผ่าน Browser ตัว Web Application Service ติดต่อกับฐานข้อมูลเพื่อนำข้อมูลมาใช้งาน เมื่อเข้ามาแล้วแสดงในหน้าเข้าสู่ระบบ เมื่อเข้าสู่ระบบแล้วแสดงหน้า Dashboard สำหรับจัดการระบบต่อไป

3.3.3 Subscribe (3)

Microcontroller ทำการ Subscribe ไปที่ Platform ในหัวข้อ profile settings และ Web Application Service ทำการ Subscribe ไปที่ Platform ในหัวข้อ logging โดยเมื่อมีข้อมูล Publish เข้ามาในหัวข้อที่ได้ Subscribe ไว้ ทำให้ผู้ที่ Subscribe ไว้ที่รับข้อมูลด้วย

3.3.3 Publish (topic: logging) (4)

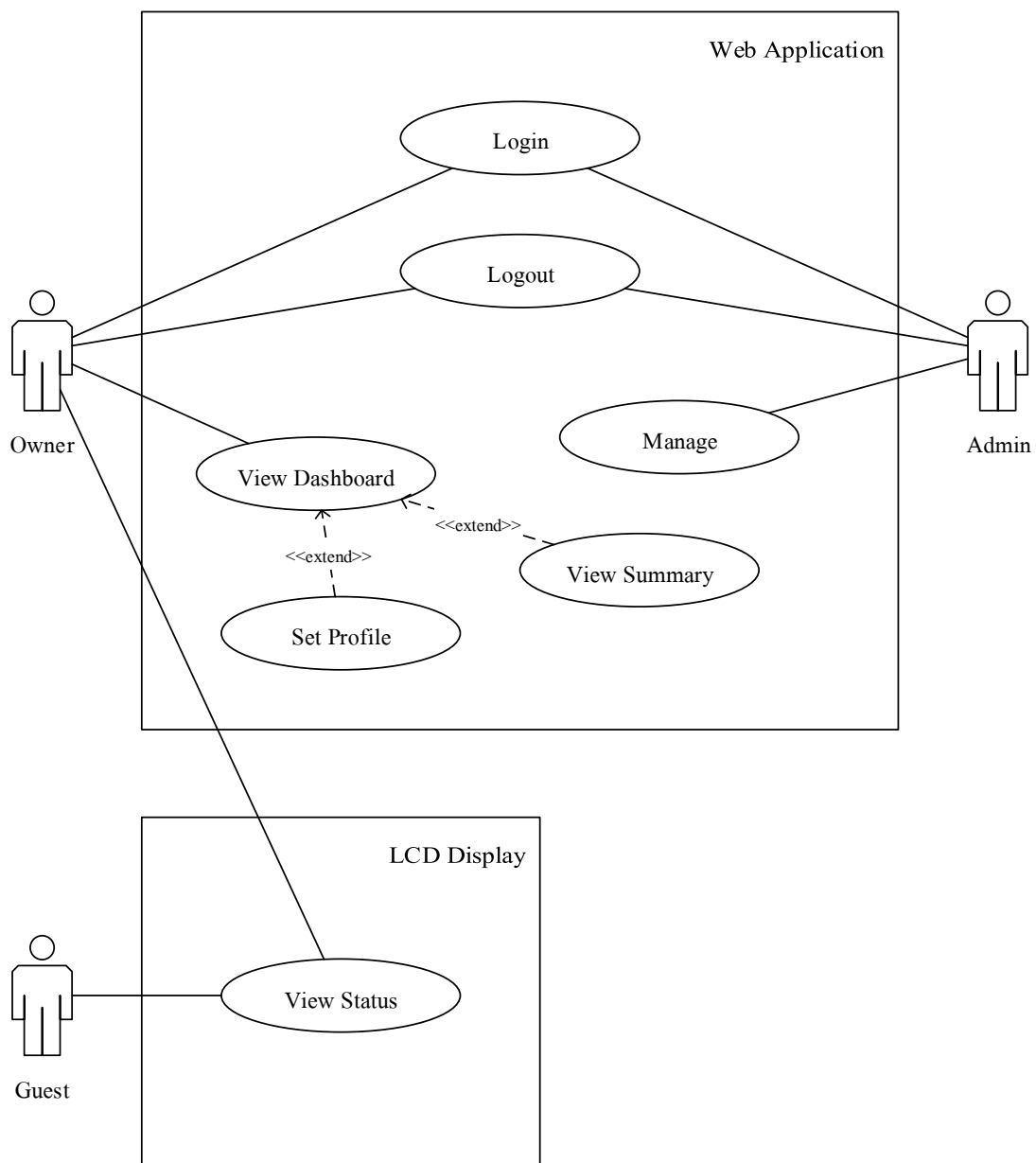
Microcontroller ส่งข้อมูลให้ Platform ทำให้มีเหตุการณ์เกิดขึ้น 3 อย่างคือ เก็บข้อมูลที่รับมาลงฐานข้อมูล, ส่งข้อมูลต่อไปที่ Web Application Service และทำการอัปเดตหน้าเว็บ

3.3.3 Publish (topic: profile settings) (5)

ผู้ใช้งานส่งข้อมูลให้ Platform ทำให้ Web Application ทำงานและมีเหตุการณ์เกิดขึ้น 3 อย่างคือ แก้ไขข้อมูลที่ได้รับมาลงฐานข้อมูล, ส่งข้อมูลต่อไปที่ Broker และ Broker ส่งข้อมูลต่อไปที่ Microcontroller

3.4 Use Case Diagram

Use Case Diagram จะมีรายละเอียดดังรูป 3.4



รูป 3.4 Use Case Diagram

3.4.1 รายละเอียด Use Case

ตาราง 3.1 ถึงตาราง 3.5 ด้านล่างต่อไปนี้แสดงถึงรายละเอียดการใช้งานของแต่ละ Use Case ตามรูปแบบของ UML Use Case Description

ตาราง 3.1 Login Use Case

Use Case Name	Login
Actors	Owner, Admin
Use Case Purpose	เพื่อใช้ในการเข้าสู่ระบบ
Pre-conditions	-
Post-conditions	เข้าสู่ระบบเพื่อเข้าหน้าจัดการข้อมูลต่อไป
Main Course	1.กรอก Username และ Password 2.กดปุ่ม Login หรือ Enter 3.เข้าสู่ระบบเสร็จสิ้น
Exceptions	1.Username หรือ Password ไม่ถูกต้อง 2.ไม่สามารถติดต่อกับฐานข้อมูลได้

ตาราง 3.2 Logout Use Case

Use Case Name	Logout
Actors	Owner, Admin
Use Case Purpose	เพื่อใช้ในการออกจากระบบ
Pre-conditions	เข้าสู่ระบบมาแล้วก่อนหน้านี้
Post-conditions	ออกจากระบบโดยสมบูรณ์
Main Course	1.กดปุ่ม Logout 2.ออกจากระบบเสร็จสิ้น
Exceptions	1.ระบบภายในผิดพลาด

ตาราง 3.3 View Dashboard Use Case

Use Case Name	View Dashboard
Actors	Owner
Use Case Purpose	ใช้ในการแสดงและจัดการระบบ
Pre-conditions	เข้าสู่ระบบมาแล้วก่อนหน้านี้
Post-conditions	-
Main Course	1.แสดงผลประวัติและกราฟต่างๆ 2.แก้ไข Profile ต่างๆ
Exceptions	1.ระบบภายในผิดพลาด 2.ใส่ข้อมูลไม่ถูกต้อง

ตาราง 3.4 View Summary Use Case

Use Case Name	View Summary
Actors	Owner
Use Case Purpose	ใช้ในการแสดงผลสรุปในรูปแบบต่างๆ
Pre-conditions	ใช้ร่วมกับ View Dashboard
Post-conditions	-
Main Course	-
Exceptions	1.ระบบภายในผิดพลาด

ตาราง 3.5 Set Profile Use Case

Use Case Name	Set Profile
Actors	Owner
Use Case Purpose	ใช้ในการตั้งค่า Profile
Pre-conditions	ใช้ร่วมกับ View Dashboard
Post-conditions	-
Main Course	-
Exceptions	1.ระบบภายในผิดพลาด

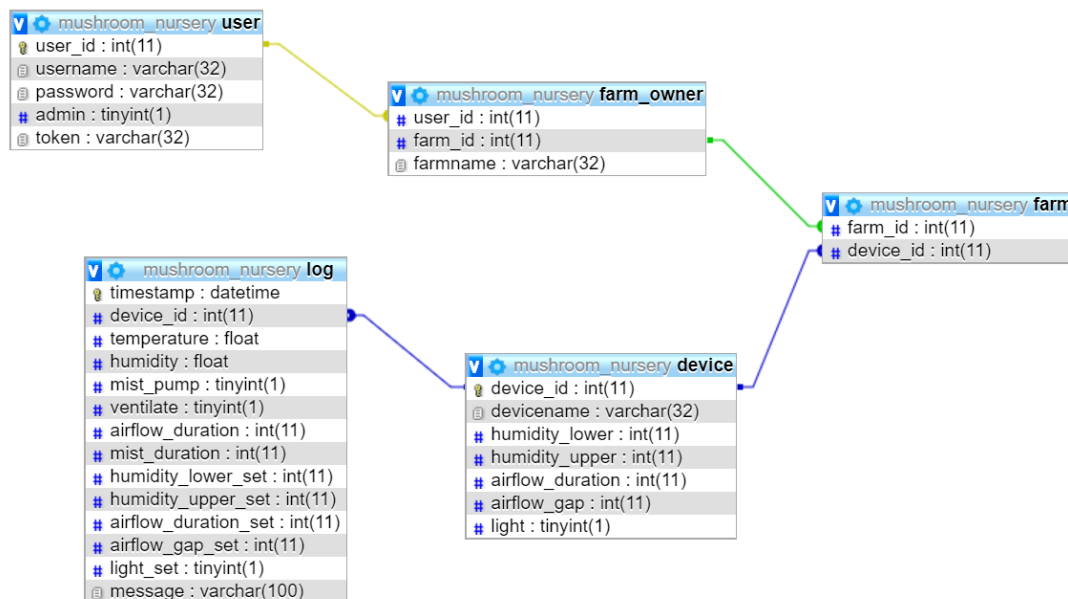
ตาราง 3.6 Manage User/Device Use Case

Use Case Name	Manage
Actors	Admin
Use Case Purpose	ใช้ในการจัดการระบบ
Pre-conditions	เข้าสู่ระบบมาแล้วก่อนหน้าและมีสถานะเป็นผู้ดูแลระบบ
Post-conditions	-
Main Course	1.จัดการ User, Farm และ Device
Exceptions	1.ระบบภายในผิดพลาด 2.ใส่ข้อมูลไม่ถูกต้อง

ตาราง 3.7 Set Profile Use Case

Use Case Name	View Status
Actors	Owner, Guest
Use Case Purpose	ใช้ในการแสดงสถานะบนจอ LCD Display บนตู้เห็ดจำลอง
Pre-conditions	-
Post-conditions	-
Main Course	1.นิ้วกดบนจอสัมผัสเพื่อเลือกแสดงส่วนต่างๆ
Exceptions	1.ระบบภายในผิดพลาด

3.5 Database Schema



รูป 3.5 Database Schema

จากรูป 3.5 จะแสดงโครงสร้างฐานข้อมูล โดยแบ่งออกเป็น 5 ตาราง มีรายละเอียดคำอธิบายแต่ละ Attribute ดังนี้

3.5.1 user

ใช้เก็บข้อมูลของผู้ใช้งาน

- 1) user_id ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บเลขไอดีของผู้ใช้งาน เป็น Primary Key
- 2) username ข้อมูลประเภทตัวอักษรความยาวไม่เกิน 32 ตัว ใช้เก็บชื่อผู้ใช้งาน
- 3) password ข้อมูลประเภทตัวอักษรความยาวไม่เกิน 32 ตัว ใช้เก็บรหัสผ่าน
- 4) admin ข้อมูลประเภทบูลีน ใช้เก็บสถานะผู้ดูแลระบบ
- 5) token ข้อมูลประเภทตัวอักษรความยาวไม่เกิน 32 ตัว ใช้เก็บ token สำหรับเข้าถึง Web Application ของ Platform

3.5.2 farm_owner

ใช้เก็บข้อมูลรายการฟาร์มของผู้ใช้งาน

- 1) user_id ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บเลขไอดีของผู้ใช้งาน เป็น Unique Key
- 2) farm_id ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บเลขไอดีของฟาร์ม เป็น Unique Key
- 3) farmname ข้อมูลประเภทตัวอักษรความยาวไม่เกิน 32 ตัว ใช้เก็บชื่อฟาร์ม

3.5.3 farm

ใช้เก็บข้อมูลรายการอุปกรณ์ของฟาร์ม

- 1) farm_id ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บเลขไอดีของฟาร์ม เป็น Unique Key
- 2) device_id ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บเลขไอดีของอุปกรณ์ เป็น Unique Key

3.5.3 device

ใช้เก็บข้อมูล Profile ของ Microcontroller

- 1) device_id ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บเลขอุปกรณ์ เป็น Primary Key
- 2) devicename ข้อมูลประเภทตัวอักษรความยาวไม่เกิน 32 ตัว ใช้เก็บชื่ออุปกรณ์
- 3) humidity_upper ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าช่วงความชื้นสูงสุด
- 4) humidity_lower ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าช่วงความชื้นต่ำสุด
- 5) airflow_duration ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าระยะเวลาที่ระบายอากาศ หน่วยเป็นนาฬิกา
- 6) airflow_gap ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าระยะเวลาที่พักระบายอากาศ หน่วยเป็นนาฬิกา
- 7) light ข้อมูลประเภทบูลีน ใช้เก็บค่าการในแสงในโรงเรือน

3.5.4 log

ใช้เก็บประวัติของค่าที่ต่างๆในช่วงเวลาต่างๆ

- 1) timestamp ข้อมูลประเภทวันเวลา ใช้เก็บเวลา Unix Timestamp เป็น Primary Key
- 2) device_id ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บเลขอุปกรณ์ เป็น Unique Key
temperature ข้อมูลประเภทเลขทศนิยมลอยตัว ใช้เก็บค่าอุณหภูมิ ณ ช่วงเวลานั้น
- 3) humidity ข้อมูลประเภทเลขทศนิยมลอยตัว ใช้เก็บค่าความชื้น ณ ช่วงเวลานั้น
- 4) mist_pump ข้อมูลประเภทบูลีน ใช้เก็บค่าสถานะกำลังใช้งานเครื่องทำหมอก
- 5) ventilate ข้อมูลประเภทบูลีน ใช้เก็บค่าสถานะกำลังใช้งานระบายอากาศ

- 6) airflow_duration ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าระยะเวลาเครื่องทำหมอกทำงานตั้งแต่เปิดเครื่อง
- 7) mist_duration ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าระยะเวลาระบายอากาศตั้งแต่เปิดเครื่อง
- 8) humidity_upper_set ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าช่วงความชื้นสูงสุด
- 9) humidity_lower_set ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าช่วงความชื้นต่ำสุด
- 10) airflow_duration_set ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าระยะเวลาที่ระบายอากาศหน่วยเป็นนาที
- 11) airflow_gap_set ข้อมูลประเภทเลขจำนวนเต็ม ใช้เก็บค่าระยะเวลาที่พักระบายอากาศหน่วยเป็นนาที
- 12) light ข้อมูลประเภทบูลีน ใช้เก็บค่าการในแสงในโรงเรือน
- 13) message ข้อมูลประเภทตัวอักษรความยาวไม่เกิน 100 ตัว ใช้เก็บข้อความที่ต้องการ เช่น error, note, comment, information เป็นต้น

3.6 MQTT Design

มีระบบ Authenticate เพื่อยืนยันอุปกรณ์ ข้อมูลที่ใช้สำหรับติดต่อสื่อสารเป็นรูปแบบ JSON แล้วให้ JavaScript แปลงเป็น Object เพื่อใช้งานต่อไป โดยแบ่ง Topic เพื่อใช้ในงานด้านต่างๆดังนี้

3.5.1 profile settings

ใช้สำหรับตั้งค่าโปรไฟล์ของอุปกรณ์ โดยส่งข้อมูลจาก Platform ไปยังอุปกรณ์ โดยมีรายละเอียดดังนี้

- 1) device_id หมายเลขไอดีของอุปกรณ์
- 2) humidity_lower ค่าการตั้งค่าความชื้นต่ำสุด
- 3) humidity_upper ค่าการตั้งค่าความชื้นสูงสุด
- 4) airflow_duration ค่าการตั้งค่าระยะเวลาระบายอากาศ
- 5) airflow_gap ค่าการตั้งค่าระยะเวลาพักระบายอากาศ
- 6) light ค่าการตั้งค่าการใช้แสง

3.5.1 logging

ใช้สำหรับส่งประวัติ โปรไฟล์ และสถานะต่างๆของอุปกรณ์ โดยส่งข้อมูลจากอุปกรณ์ไปยัง Platform โดยมีรายละเอียดดังนี้

- 1) device_id หมายเลขไอดีของอุปกรณ์
- 2) temperature ค่าอุณหภูมิ ณ เวลานั้น
- 3) humidity ค่าความชื้น ณ เวลานั้น
- 4) mist_pump ค่าการทำงานของเครื่องทำหมอก
- 5) ventilate ค่าการทำงานของระบายอากาศ
- 6) airflow_duration ค่าระยะเวลาระบายอากาศทั้งหมด ตั้งแต่เปิดเครื่อง
- 7) mist_duration ค่าระยะเวลาทำงานของเครื่องทำหมอกทั้งหมด ตั้งแต่เปิดเครื่อง
- 8) humidity_lower_set ค่าการตั้งค่าความชื้นต่ำสุด
- 9) humidity_upper_set ค่าการตั้งค่าความชื้นสูงสุด
- 10) airflow_duration_set ค่าการตั้งค่าระยะเวลาระบายอากาศ
- 11) airflow_gap_set ค่าการตั้งค่าระยะเวลาพักระบายอากาศ
- 12) light_set ค่าการตั้งค่าการใช้แสง
- 13) message ข้อความจากอุปกรณ์

บทที่ 4

การทดลองและผลการทดลอง

บทนี้เป็นการทดลองต่างๆ ทั้งส่วนโรงเรือนเพาะเห็ด ส่วน Platform และส่วน Web Application และการเชื่อมต่อทั้ง 3 ส่วนเข้าด้วยกันผ่าน MQTT Protocol โดยได้ทำการทดลองดังนี้

1. สร้างตู้จำลองโรงเรือนเพาะเห็ด
2. การทดลองรับ-ส่งข้อมูลระหว่าง Raspberry Pi และ Platform โดยใช้ MQTT Protocol
3. การดึงค่าจาก Back-End ที่ติดต่อกับ MySQL มาแสดงผลบน Web Application
4. การควบคุมระบบต่างๆภายในโรงเรือนผ่าน Web Application
5. การควบคุมระบบต่างๆภายในโรงเรือนผ่านจอ LCD หน้าโรงเรือน
6. การทดลองเพาะเห็ดโดยใช้ Platform ในการควบคุมสภาพแวดล้อมภายในตู้จำลองโรงเรือนเพาะเห็ด

4.1 สร้างตู้จำลองโรงเรือนเพาะเห็ด

4.1.1 วัตถุประสงค์

เพื่อจำลองระบบโรงเรือนที่มีการเพาะเลี้ยงเห็ดที่สามารถควบคุมความชื้นและอุณหภูมิได้โดยใช้ Microcontroller

4.1.2 วิธีการทดลอง

สร้างตู้จำลองโรงเรือนเพาะเห็ดจำลองขึ้นจาก Plastwood โดยโรงเรือนมีขนาด 1x1x1.5 เมตร และได้มีการแบ่งส่วนสำหรับเพาะเห็ดและส่วนแผงควบคุมออกจากกันดังนี้

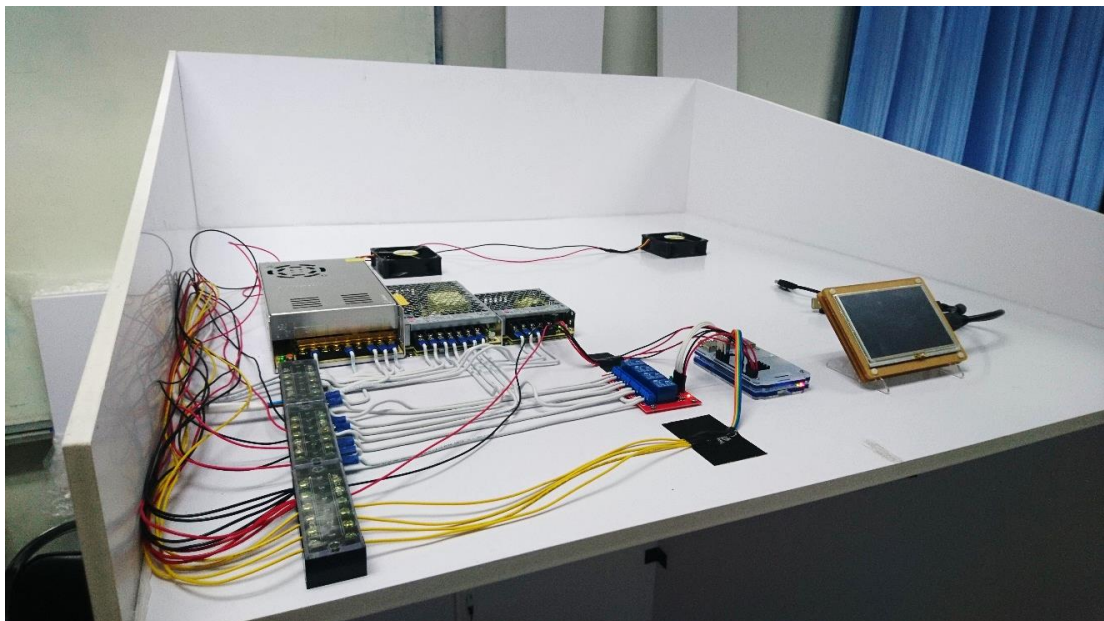
- 1) ส่วนสำหรับเพาะเห็ด ประกอบไปด้วยชั้นก้อนเห็ด มีเซนเซอร์วัดอุณหภูมิและความชื้น เซนเซอร์วัดระดับน้ำ เครื่องทำหมอก พัดลมระบายอากาศ ไฟ LED
- 2) ส่วนแผงควบคุม มี Raspberry Pi ที่คอยรับค่าต่างๆจากเซนเซอร์ และควบคุมการทำงานของอุปกรณ์ภายในโรงเรือน และจอ LCD เพื่อแสดงสถานะต่างๆ

4.1.3 ผลการทดลอง

สร้างตู้จำลองโรงเรือนเพาะเห็ดโดยมีส่วนของตู้แยกออกเป็น 2 ส่วน คือภายในตู้เป็นส่วนสำหรับเพาะเห็ดตามรูป 4.1 และด้านบนตู้จะเป็นส่วนแผงควบคุมตามรูป 4.2



รูป 4.1 ส่วนสำหรับเพาะเห็ด



รูป 4.2 ส่วนแผงควบคุม

4.2 การทดลองรับ-ส่งข้อมูลระหว่าง Raspberry Pi และ Platform โดยใช้ MQTT Protocol

4.2.1 วัตถุประสงค์

เพื่อส่งข้อมูลอุณหภูมิและความชื้นที่วัดได้จาก Raspberry Pi ไปแสดงบน Platform

4.2.2 วิธีการทดลอง

- 1) ติดตั้งระบบปฏิบัติการ NOOBS (New Out Of Box Software) บน Raspberry Pi
- 2) ติดตั้งระบบปฏิบัติการ Ubuntu บน Platform
- 3) เขียนโปรแกรม Microservice บน Platform โดยใช้ NodeJS และ Mosca ตามโปรแกรม 4.1

โปรแกรม 4.1 การตั้งค่าโปรแกรมเพื่อทดลอง Create MQTT Broker

```
var mosca = require('mosca');

var ascoltatore = {
  //using ascoltatore
  type: 'mongo',
  url: 'mongodb://localhost:27017/mqtt',
  pubsubCollection: 'ascoltatori',
  mongo: {}
};

var settings = {
  port: 1883
};

var server = new mosca.Server(settings);

server.on('clientConnected', function(client) {
  console.log('client connected', client.id);
});

//fired when a message is received
server.on('published', function(packet, client) {
  console.log('Published', packet.payload);
});

server.on('ready', setup);

//fired when the mqtt server is ready
function setup() {
  console.log('Mosca server is up and running');
}
```


4. เขียนโปรแกรม Microservice บน Raspberry Pi โดยใช้ NodeJS และ MQTT ตามโปรแกรม 4.2

โปรแกรม 4.2 การตั้งค่าโปรแกรมเพื่อทดลอง Open Connection, Subscribe และ Publish

```
var mqtt = require('mqtt')
var client = mqtt.connect('mqtt://test.mosquitto.org')

client.on('connect', function () {
  client.subscribe('presence')
  client.publish('presence', 'Hello mqtt')
})

client.on('message', function (topic, message) {
  //message is Buffer
  console.log(message.toString())
  client.end()
})
```

4.2.3 ผลการทดลอง

Platform สามารถรับข้อมูลจาก Raspberry Pi ผ่าน MQTT Protocol บนมาตรฐาน IP โดยการปรับแต่งให้ Microcontroller และ Platform สามารถเข้ากันได้

4.3 การดึงค่าจาก Back-End ที่ติดต่อกับ MySQL มาแสดงผลบน Web Application

4.3.1 วัตถุประสงค์

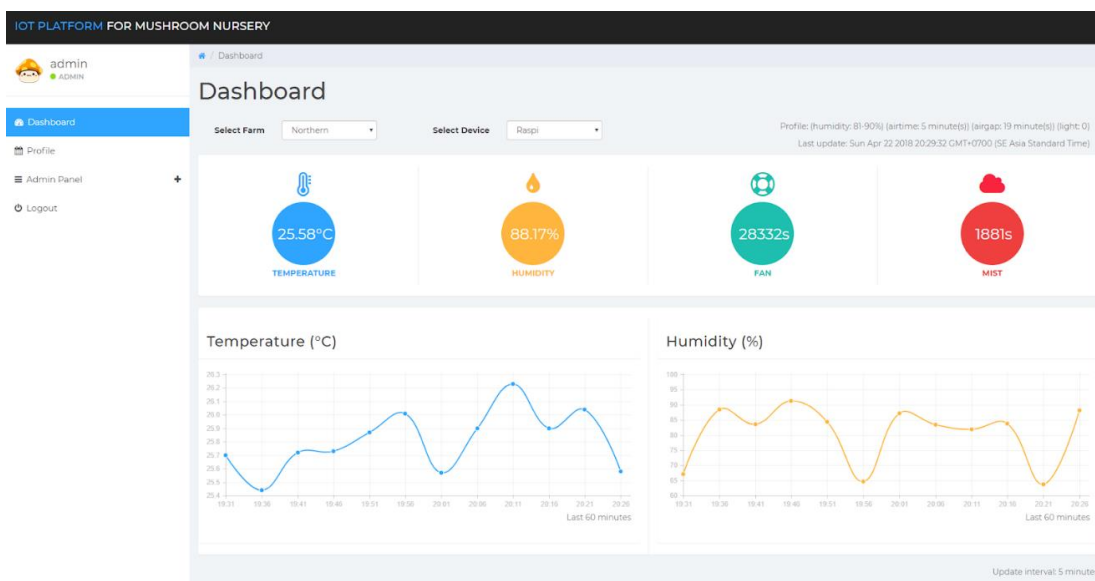
เพื่อดึงค่าอุณหภูมิ ความชื้น และค่า Profile ต่างๆที่ได้ตั้งค่าและถูกเก็บไว้ใน MySQL มาแสดงผลบนหน้า Dashboard ใน Web Application

4.3.2 วิธีการทดลอง

เขียน Code SQL เพื่อ query ข้อมูลส่วนที่ต้องการออกมา แล้วนำมาเก็บในรูปแบบ Object ของ JavaScript เพื่อส่งข้อมูลผ่าน Socket.io ไปยัง Front-End

4.3.3 ผลการทดลอง

หน้า Dashboard ตามรูป 4.3 จะสามารถแสดงผลค่าอุณหภูมิ และความชื้นปัจจุบันภายในโรงเรือนได้ และสามารถแสดงการฟ้อนหลังได้ทั้งอุณหภูมิ และความชื้น นอกจากนั้นยังสามารถแสดงระยะเวลาการทำงานทั้งหมดของพัดลมระบายอากาศกับเครื่องทำหมอก



รูป 4.3 Dashboard

4.4 การควบคุมระบบต่างๆภายในโรงเรือนผ่าน Web Application

4.4.1 วัตถุประสงค์

เพื่อออกแบบระบบ Web Application สำหรับแสดงสถานะและควบคุมระบบภายในโรงเรือนเพาะเห็ด

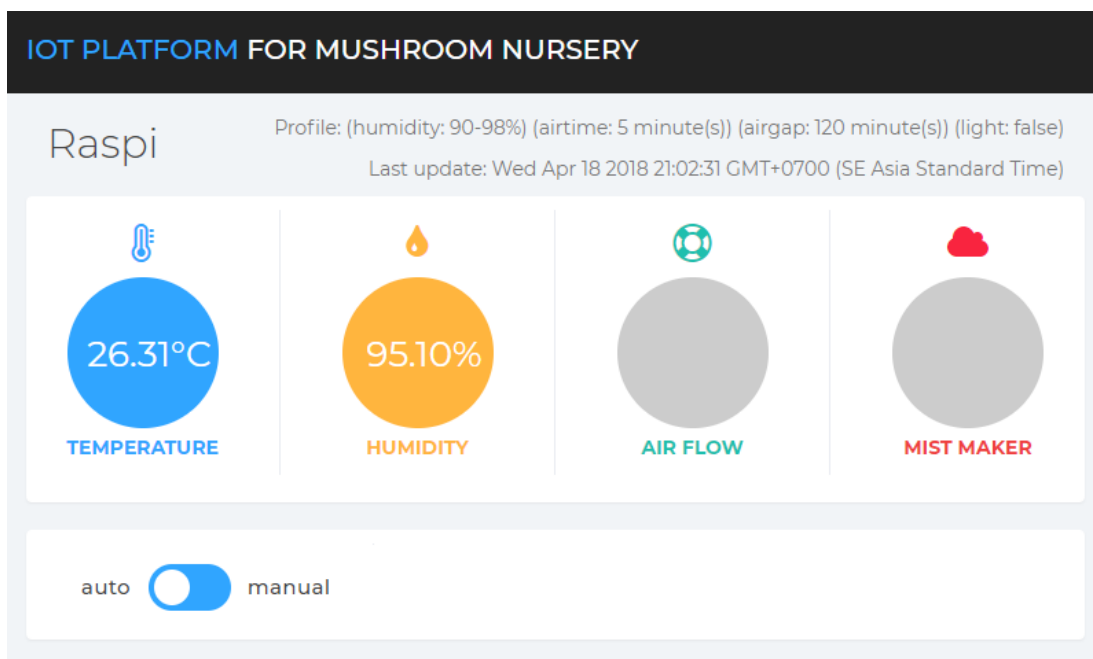
4.5 การควบคุมระบบต่างๆภายในโรงเรือนผ่านจอ LCD หน้าโรงเรือน

4.5.1 วัตถุประสงค์

เพื่อออกแบบ Dashboard สำหรับจอ LCD หน้าโรงเรือน เพื่อใช้แสดงสถานะและควบคุมระบบภายในโรงเรือนเพาะเห็ด

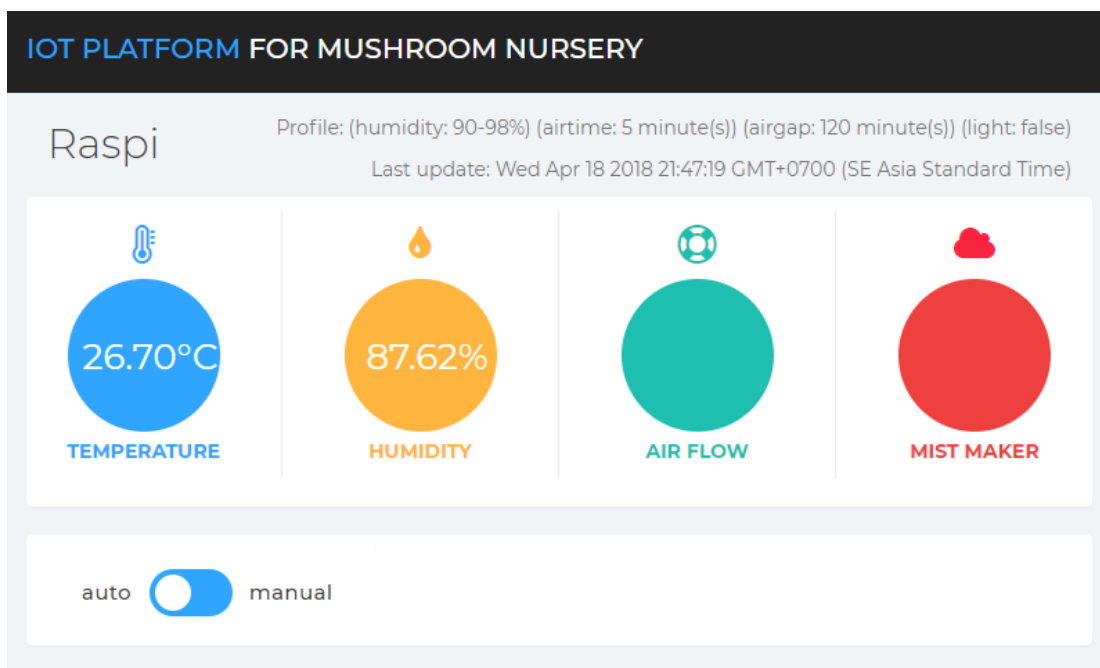
4.5.2 วิธีการทดลอง

ออกแบบหน้า Dashboard ตามรูป 4.11 สำหรับใช้แสดงสถานะและควบคุมระบบต่างๆภายในโรงเรือน โดยจอ LCD มีขนาด 800x480 โดยหน้า Dashboard จะมีรายละเอียดหลักๆดังนี้



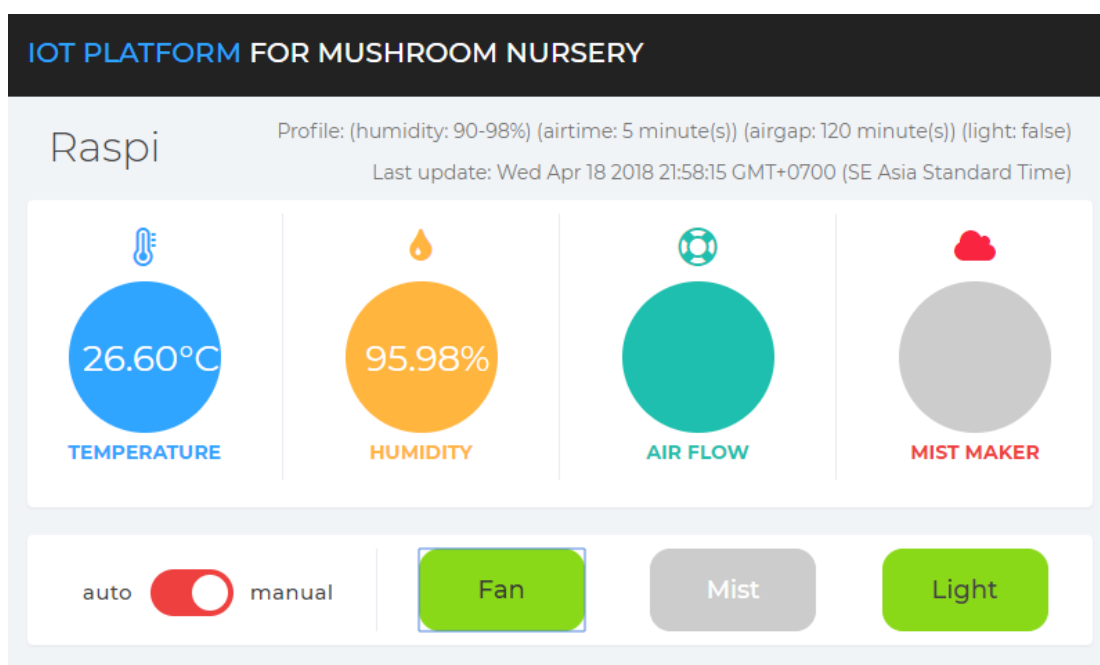
รูป 4.4 Dashboard สำหรับจอ LCD(1)

โดยด้านบนจะแสดงชื่อ Device, Profile ที่ได้ตั้งค่าไว้ และเวลาล่าสุดที่แสดง ส่วนกลางจะเป็นรายละเอียดหลักทั้ง 4 อย่าง ประกอบไปด้วย อุณหภูมิ ความชื้น พัฒมระบายอากาศ และเครื่องทำหมอก โดยส่วนอุณหภูมิ ความชื้นนั้นจะแสดงเป็นค่าปัจจุบันตามที่วัดค่าได้ภายในตู้เพาะเห็ด และส่วนพัฒมระบายอากาศ และเครื่องทำหมอกเมื่อทำการเปิดการทำงานจะมีไฟแสดงสถานะปรากฏขึ้น โดยจะเปิดการทำงานก็ต่อเมื่อค่าความชื้นอยู่ต่ำกว่าค่าที่ตั้งค่าไว้ใน Profile ตามรูป 4.12



รูป 4.5 Dashboard สำหรับจอ LCD(2)

ในส่วนล่างส่วนสุดท้ายจะสามารถสลับการทำงานจากอัตโนมัติมาเป็นการบังคับด้วยมือได้ โดยสามารถบังคับเปิด-ปิดการทำงานของพัดลมระบายอากาศ และเครื่องทำหมอกได้นอกจากนั้นยังสามารถเปิดไฟภายในตู้จำลองโรงเรือนเพาะเห็ดได้ ตามรูป 4.13



รูป 4.6 Dashboard สำหรับจอ LCD(3)

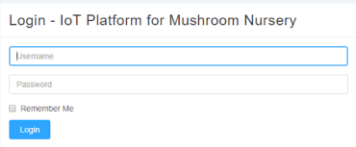
4.5.1 ผลการทดลอง

หน้า Dashboard สามารถแสดงค่าอุณหภูมิและความชื้นปัจจุบันภายในโรงเรือนได้ และเมื่อความชื้นต่ำกว่าที่กำหนดไว้พัฒนาระบบอากาศ และเครื่องทำหมอกก็จะทำงาน ทำให้ไฟแสดงสถานะติด และเมื่อสลับการควบคุมจากอัตโนมัติเป็นการควบคุมด้วยมือแล้วสามารถบังคับเปิด-ปิด พัฒนาระบบอากาศ, เครื่องทำหมอก และไฟภายในโรงเรือนได้

4.4.2 วิธีการทดลอง

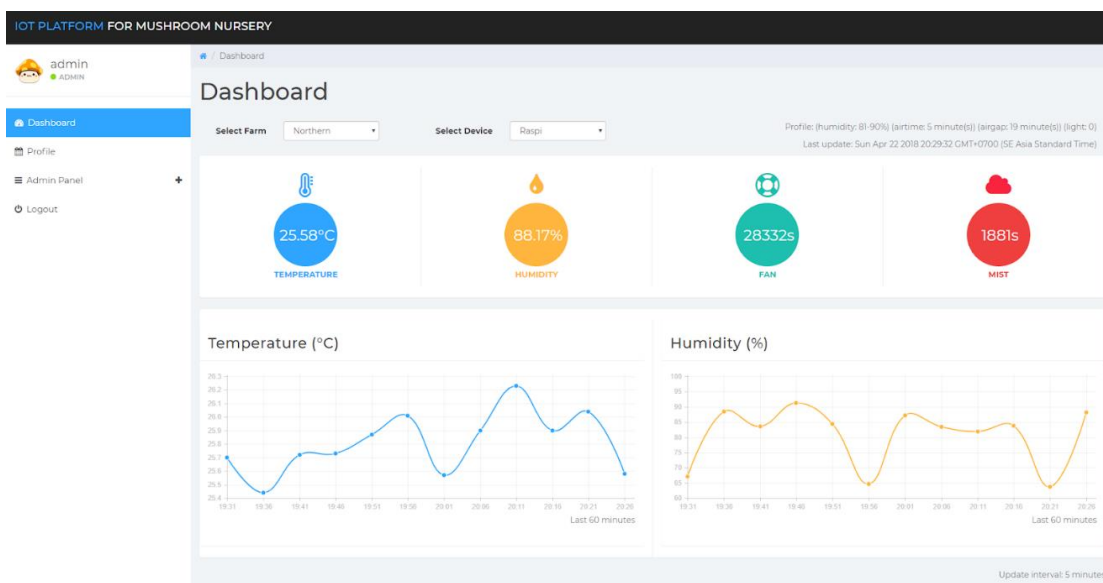
ออกแบบ Web Application สำหรับใช้แสดงสถานะและควบคุมระบบต่างๆภายในโรงเรือน โดยจะแบ่งออกเป็น 4 ส่วน คือ หน้า Login, หน้า Dashboard, หน้า Profile และหน้า Admin Panel

หน้า Login ตามรูป 4.4 มีรายละเอียดคือ ช่องสำหรับใส่ Username และ Password เพื่อเข้าใช้งาน



รูป 4.7 หน้า Login

หน้า Dashboard ตามรูป 4.5 จะสามารถเลือกฟาร์มและ Device เพื่อแสดงอุณหภูมิและความชื้นจากค่าที่วัดได้ภายในโรงเรือน พัฒนาระบบอากาศ และเครื่องทำหมอกจะแสดงระยะเวลาทำงานทั้งหมดเป็นวินาที และกราฟแสดงรายละเอียดย้อนหลังสำหรับอุณหภูมิและความชื้น



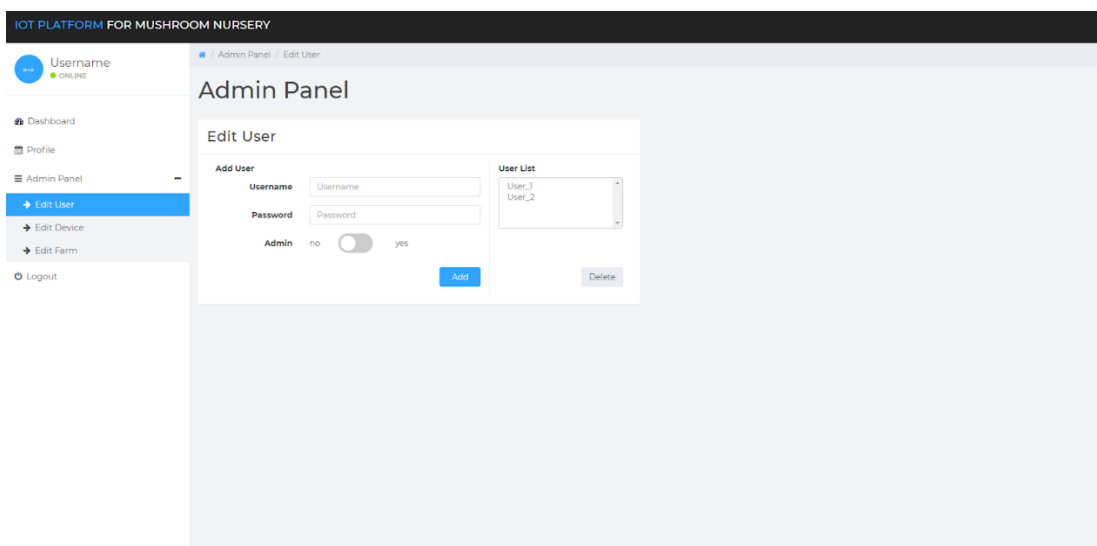
รูป 4.8 หน้า Dashboard

หน้า Profile ตามรูป 4.6 มีไว้สำหรับตั้งค่า Profile สำหรับใช้ในการควบคุมระบบต่างๆภายในโรงเรือน โดยทำการเลือกฟาร์ม และ Device จากนั้นทำการตั้งค่า ช่วงของความชื้น และระยะเวลาการเวลาในการเปิดพัดลมระบายอากาศต่อวัน

The screenshot displays the 'Profile' page of the 'IOT PLATFORM FOR MUSHROOM NURSERY'. The top navigation bar includes a sidebar with 'Dashboard', 'Profile', 'Admin Panel', and 'Logout'. The main content area shows a 'Profile' section with a 'Settings' form. The form includes the following fields: 'Farm' (dropdown), 'Device' (dropdown), 'Humidity range' (dropdown with a unit of '°C'), 'Air flow duration' (dropdown with a unit of 'minute(s) per day'), and a 'Light' toggle switch. At the bottom right of the form are 'Reset' and 'Save' buttons. The top header shows the user is 'Username' and the system is 'online'.

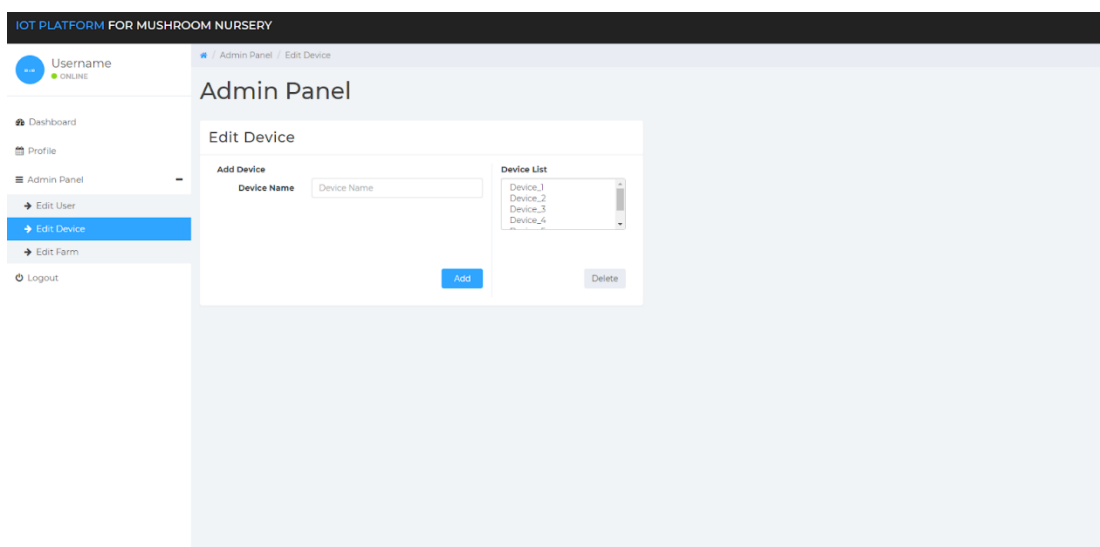
รูป 4.9 หน้า Profile

หน้า Admin Panel จะแบ่งเป็น 3 หน้าโดยในหน้าแรกจะเป็นหน้าสำหรับจัดการผู้ใช้งาน ตามรูป 4.7 โดยสามารถเพิ่มหรือลบผู้ใช้งานออกจากระบบได้ และในการเพิ่มผู้ใช้งานสามารถกำหนดได้ว่าจะเป็นผู้ใช้งานธรรมดาหรือหรือผู้ดูแลระบบ



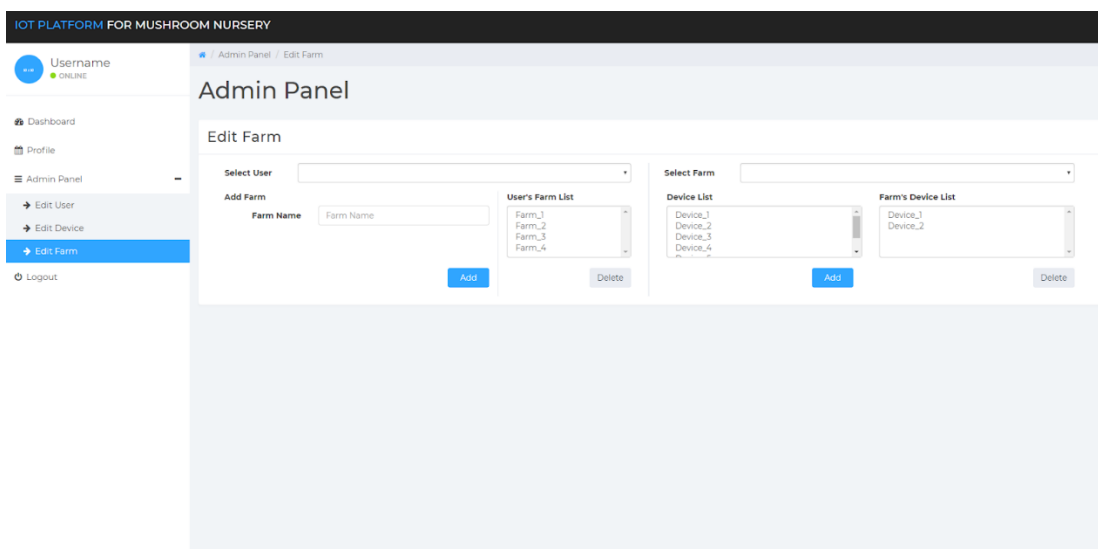
รูป 4.10 หน้า Edit User

หน้า Admin Panel หน้าที่สอง จะเป็นหน้าสำหรับจัดการ Device ตามรูป 4.8 โดยสามารถเพิ่มหรือลบ Device ออกจากระบบได้



รูป 4.11 หน้า Edit Device

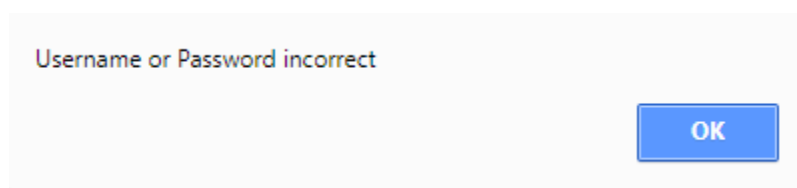
หน้า Admin Panel หน้าสุดท้ายจะเป็นหน้าสำหรับจัดการฟาร์ม ตามรูป 4.9 โดยการเลือกผู้ใช้งานที่มีอยู่ในระบบและจะสามารถเพิ่มหรือลบฟาร์มสำหรับผู้ใช้งานคนนั้นได้ หรือทำการเลือกฟาร์มที่มีอยู่ในระบบและสามารถเพิ่มหรือลบ Device จากฟาร์มนั้นๆได้



รูป 4.12 หน้า Edit Farm

4.4.3 ผลการทดลอง

หน้า Login เมื่อใส่ Username และ Password ถูกจะสามารถเข้าระบบได้ แต่ถ้าใส่ผิดจะไม่สามารถเข้าสู่ระบบได้ ตามรูป 4.10



รูป 4.13 alert message

หน้า Dashboard สามารถเลือกฟาร์มและ Device ที่จะแสดงรายละเอียดได้ และรายละเอียดต่างๆ ที่แสดงมีความถูกต้องสมบูรณ์

หน้า Profile สามารถเลือกฟาร์มและ Device เพื่อตั้งค่า Profile ให้ Device นั้นๆ ได้ และเมื่อทำการตั้งค่า Profile เสร็จจะอัปเดต Profile นั้นเข้าสู่ระบบ

หน้า Admin Panel สามารถจัดการผู้ใช้งาน Device และฟาร์มได้ โดยสามารถเพิ่มหรือลบผู้ใช้งานและ Device จากระบบได้ และสามารถเพิ่มหรือลบฟาร์มให้ผู้ใช้งานที่มีอยู่ในระบบได้ และสามารถเพิ่มหรือลบ Device ที่มีอยู่ในระบบให้ฟาร์มที่ต้องการได้

4.6 การทดลองเพาะเห็ดโดยใช้ Platform ในการควบคุมสภาพแวดล้อมภายในตู้จำลองโรงเรือนเพาะเห็ด

4.6.1 วัตถุประสงค์

เพื่อทดสอบประสิทธิภาพของ Platform ว่าสามารถช่วยในการเพาะเห็ดภายในโรงเรือนภายใต้สภาพแวดล้อมที่กำหนดได้

4.6.2 วิธีการทดลอง

ทำการเพาะเห็ดจำนวน 4 ชนิดในตู้จำลองโรงเรือนเพาะเห็ด ตามรูป 4.14 โดยมีเห็ดดังนี้

1. เห็ดนางรมฮังการี อยู่ด้านบนซ้ายของรูป
2. เห็ดนางนวลสีชมพู อยู่ด้านบนขวาของรูป
3. เห็ดนางฟ้าภูฐาน อยู่ด้านล่างซ้ายของรูป
4. เห็ดเป๋าฮื้อ อยู่ด้านล่างขวาของรูป



รูป 4.14 เห็ดทั้ง 4 ชนิดในตู้เพาะเห็ด




โดยกำหนดค่า Profile ดังรูป 4.15

Profile: (humidity: 90-98%) (airtime: 5 minute(s)) (airgap: 120 minute(s)) (light: false)

รูป 4.15 ค่า Profile สำหรับการเพาะเห็ด

จากนั้นทำการติดตามการเจริญเติบโตของเห็ดเป็นระยะเวลา 2 สัปดาห์ โดยทำการเก็บข้อมูลเป็นถ່ายรูปบันทึกดังตาราง 4.1

ตาราง 4.1 อัตราการเจริญเติบโตของเห็ด

วันที่	รูป	
1 - 2		
3 - 4		
5 - 6		

7 - 8			
9 - 10			
11 - 12			
13 - 14			

4.6.3 ผลการทดลอง

เห็ดทั้ง 4 ชนิด ภายในตู้สามารถเจริญเติบโตได้เป็นอย่างดีตามรูป 4.16 ถึงรูป 4.19 ภายใต้ค่า Profile ที่ตั้งไว้ได้



รูป 4.16 นางรมฮังการี



รูป 4.17 เห็ดนางนวลสีชมพู



รูป 4.18 เห็ดนางฟ้าภูฐาน



รูป 4.19 เห็ดเป๋าฮื้อ

บทที่ 5

สรุปผล

5.1 ผลสรุปของโครงการ

5.1.1 ส่วนของผู้จำลองโรงเพาะเห็ด

สั่งซื้อวัสดุ เช่น แผ่นไม้ Plastwood, ล้อรถเข็น, เหล็กฉาก, สายไฟฟ้า, ตะปู, กาวเชื่อม, ถังน้ำ, ท่อน้ำ และอุปกรณ์อิเล็กทรอนิกส์ เป็นต้น และเครื่องมือ เช่น สว่านไฟฟ้า, เลื่อย Jigsaw, ไขควง, คีมปากแหลม และคีมตัด เป็นต้น

ลงมือตัดแผ่น Plastwood ตามแบบตู้ที่ได้ออกแบบเอาไว้ จากนั้นนำมาประกอบตู้ด้วยวัสดุที่ได้กล่าวไปในข้างต้น ตามด้วยติดตั้งอุปกรณ์อิเล็กทรอนิกส์ตาม Hardware I/O Diagram ที่ได้ออกแบบเอาไว้

ทดสอบความแข็งแรงของตู้โดยการเคลื่อนย้าย วางของที่มีน้ำหนัก โดยตู้ต้องไม่บวม งอ หรือเบี้ยว ทดสอบระบบอิเล็กทรอนิกส์โดยจ่ายไฟเข้า Power Supply แล้วสั่งใช้งาน พัดลมเครื่องทำหมอก หลอดไฟLED อ่านค่า Sensor จาก DHT22 ด้วย Raspberry Pi ให้ถูกต้องตามที่กำหนด โดยที่ให้ค่าที่ไม่เพี้ยน ไม่ร้อนเกินค่าปกติ และไม่ให้ทองแดงในสายไฟฟ้าโดนความชื้นโดยตรง

5.1.2 ส่วนของ Platform

ติดตั้งระบบปฏิบัติการ CentOS ลงบน Server และ NOOBS บน Raspberry Pi แล้วตั้งค่า Network และค่าพื้นฐานที่จำเป็น เช่น รหัสผ่าน, Time Zone ให้เรียบร้อย

ติดตั้ง Service ที่ใช้ใน Platform ตามหัวข้อ Deployment ในภาคผนวก ข

ทดสอบระบบโดยการใช้งานทุกส่วนของระบบ เช่น ติดต่อกับฐานข้อมูลด้วยคำสั่ง Select, Insert, Update และ Delete, ใช้งาน Function ที่เขียนบน Platform และใช้งาน Web Application ด้วย Test Case ที่ครอบคลุมการใช้งานทั้งหมด

ทดสอบใช้งานจริง 24 ชั่วโมงต่อวัน เก็บค่า Log เพื่อนำมาประเมินความถูกต้องของระบบ และแก้ไขปัญหาตามความผิดพลาดที่พบ

5.1.3 ส่วนของ Web Application

เลือกใช้ Dashboard Template ที่เหมาะสมกับความต้องการของระบบ เลือกใช้ ณ ที่นี้คือ Lumino - Dashboard โดยใช้ HTML, CSS และ JavaScript ในการพัฒนา Web Application

ปรับแต่งส่วนของ User Interface ตามความต้องการที่ได้ออกแบบไว้ พัฒนา Front-end และ Back-end ด้วยภาษา JavaScript

ทดสอบ Web Application ทุกส่วนด้วย Test Case ที่ครอบคลุมการใช้งานทั้งหมด แบ่งเป็น Login, Logout, View Dashboard และ Admin Panel โดยต้องให้ค่าที่ถูกต้องตามที่กำหนด และแก้ไขตามความผิดพลาดที่พบ

5.2 ปัญหาและอุปสรรค

5.2.1 ส่วนของผู้จำลองโรงเพาะเห็ด

การสร้างผู้จำลองโรงเพาะเห็ดต้องตัดแผ่น Plastwood เองด้วย Jigsaw ทำให้เกิดความผิดพลาดได้ง่าย ตัดแล้วขนาดและรูปทรงไม่ได้ตามที่ต้องการ ต้องใช้เครื่องมือนำคือ ไม้บรรทัดเหล็กฟุตยาว และเส้นเหล็กแบน

ช่องทางการระบายอากาศไม่เพียงพอและไม่เหมาะสม ทำให้ดอกเห็ดออกมาเหี่ยว ไม่สวยตามที่คาดหวัง ต้องเพิ่มขนาดของช่องระบายอากาศ และเพิ่มเวลาที่ระบายอากาศในระบบ

5.2.2 ส่วนของ Platform

ในส่วนของ NodeJS และ MQTT ไม่เข้าใจการทำงานของเทคโนโลยีโดยละเอียด ทำให้เกิดความล่าช้าในการพัฒนา และ Code ที่พัฒนาไม่ได้ประสิทธิภาพ ต้องศึกษาและทำความเข้าใจ เพื่อให้ใช้งานได้อย่างมีประสิทธิภาพ ตัวอย่างเช่น NodeJS ได้ศึกษา Syntax มาตรฐาน, Arrow Function, Callback Function, Synchronous และ Asynchronous เป็นต้น ส่วนของ MQTT ได้ศึกษาการ Authenticate, Subscribe และ Publish เป็นต้น

5.1.3 ส่วนของ Web Application

จาก Dashboard Template ที่ได้เลือกมา ต้องใช้เวลาศึกษาโครงสร้างทั้งหมดจากผู้พัฒนา Lumino - Dashboard

เนื่องจากระบบมีหลาย Module ทำให้ต้องใช้เวลาและการจัดการสูง ต้องวางแผนและเขียนบันทึกให้ครอบคลุม แบ่ง Module การทำงานออกเป็นส่วนๆ ไม่ให้เกิดการ Bind กันของ Function

5.3 แนวทางพัฒนาต่อ

5.3.1 ส่วนของผู้จำลองโรงเพาะเห็ด

ลงมือติดตั้งและใช้งานในฟาร์มเห็ดจริง ให้สามารถ Scaling จำนวนของฟาร์มเห็ดจำนวนมากได้ จากหมู่บ้าน ชุมชน ของผู้เป็นเจ้าของฟาร์มเห็ด

5.3.2 ส่วนของ Platform

ทดลองรองรับ ผู้ใช้งาน, ฟาร์ม และอุปกรณ์ จำนวนมาก เชื่อมต่อเข้ามาใน Platform จากนั้นวิเคราะห์ว่ารองรับปริมาณได้เท่าไรในขนาด Platform หนึ่ง

เมื่อการใช้งานในระบบเดิมไม่เพียงพอ สามารถวางแผนออกแบบและพัฒนาทั้งแบบ Horizontal Scaling และ Vertical Scaling ได้

5.3.3 ส่วนของ Web Application

เพิ่ม Feature ตามความต้องการของเจ้าของฟาร์มเห็ดเช่น ควบคุมความชื้นแบบเปลี่ยนแปลงได้ในแต่ละช่วงเวลาที่กำหนด เป็นต้น

เพิ่มระบบ Sorting และ Searching เพิ่มรองรับปริมาณผู้ใช้งาน, ฟาร์ม และอุปกรณ์จำนวนมากที่เพิ่มขึ้น

บรรณานุกรม

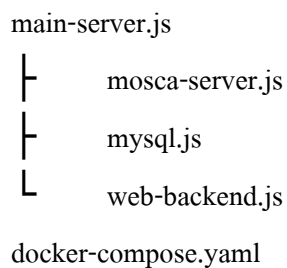
- Oran Chieochan and Anukit Saokaew. 2017. "IoT for Smart Farm: A case study of the Lingzhi Mushroom Farm at Maejo University." Bachelor's Thesis, Chiang Mai University, Thailand.
- Theeramet Kaewwiset and Paitoon Yodkhad. 2013. "Automatic Temperature and Humidity control system by using Fuzzy Logic Algorithm for Mushroom nursery." Bachelor's Thesis, Chiangrai College, Thailand.
- Adison Chuenthaisong. 2016. "Temperatuer and Humidiy Control System for Mushroom." Bachelor's Thesis, Kasetsart University Chalermphrakiat Sakonnakhon Province Campus, Thailand.
- Nithikorn Sabaengbal. 2016. "Wonderful Mushroom Plant." Bachelor's Thesis, Kasetsart University Chalermphrakiat Sakonnakhon Province Campus, Thailand.
- Hyemin Lee, Dongig Sin, Eunsoo Park, Injung Hwang, Gyeonghwan Hong, and Dongkun Shin. 2016. "Open Software Platform for Companion IoT Devices." Bachelor's Thesis, Sungkyunkwan University, Korea.
- กฤษณะ จ้อย . 2018. กฤษณะฟาร์มเห็ด . [Online] . Available : <http://www.kritsanamushroom.com/th>.
- Arpapon Chankaew. 2018. factors affect mushroom growth. [Online]. Available : <https://hughed.blogspot.com/2015/07/factors-affect-mushroom-growth.html>.

ภาคผนวก ก

Source Code

Platform

เขียนด้วยภาษา JavaScript ใช้งานด้วย NodeJS โดยแบ่งเป็น 4 Module ดังนี้



main-server.js

```
//require
var mosca_server = require('./mosca-server.js')
var mysql = require('./mysql.js')
var web_backend = require('./web-backend.js')

//function
function makeToken (length){
    var text = ''
    var possible =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'
    for(var i=0; i<length; i++)
        text += possible.charAt(Math.floor(Math.random()* possible.length))
    return text
}

function checkToken (token){
    var check_result = false
    var sql = 'SELECT user_id FROM user WHERE token =?'
    return mysql.con.query(sql, [token], function(err, result){
        if((result[0]===undefined))
            return true
        else
            return false
    })
}

function checkAdmin (token){
    var check_result = false
    var sql = 'SELECT admin FROM user WHERE token =?'
    return mysql.con.query(sql, [token], function(err, result){
        if((result[0]===undefined))
```

```

        return true
    } else {
        return false
    }
})

}

//object
//log:device -> mqtt -> mosca -> nodejs -> mysql
mosca_server.server.on('published', function(message, detail){
    if(message.topic === 'logging'){
        console.log('MQTT log received:', message.payload.toString())
        var sql = 'INSERT INTO log SET ?'
        mysql.con.query(sql, JSON.parse(message.payload), function(err, result){
            if(err)
                throw err
            console.log('MySQL 1 log record inserted\n')
        })
    }
})

//socket.io for web backend
web_backend.io.on('connection', function(socket){
    socket.on('login', function(message){
        var sql = 'SELECT password FROM user WHERE username = ?'
        mysql.con.query(sql, [message.username], function(err, result){
            if(err)
                throw err
            if((result[0] === undefined) && (message.password ===
result[0].password)){
                var token = makeToken(32)
                var sql = 'UPDATE user SET token = ? WHERE
username = ?'
                mysql.con.query(sql, [token, message.username],
function(err, result){
                    if(err)
                        throw err
                    socket.emit('login-result', token)
                    console.log('Socket.io user logged in:',
message.username)
                })
            } else {
                socket.emit('login-result', false)
            }
        })
    })

    socket.on('get-user-info', function(message){
        if(!checkToken(message))
            socket.emit('get-user-info-result', false)

        var sql = 'SELECT user_id, username, admin FROM user WHERE token = ?'
        mysql.con.query(sql, [message], function(err, result){

```

```

        if(err)
            throw err
        if((result[0]==undefined)){
            socket.emit('get-user-info-result', result[0])
        }
        else {
            socket.emit('get-user-info-result', false)
        }
    })
})

socket.on('update-dashboard', function(message){
    if(!checkToken(message.token))
        socket.emit('update-dashboard-result', false)

    var sql = 'SELECT log.*FROM user, log WHERE token =? AND device_id =?
AND timestamp >=(SELECT CURRENT_TIMESTAMP -(1000*5*2))'
    mysql.con.query(sql, [message.token, message.device_id], function(err,
result){
        if(err)
            throw err
        if((result == undefined)){
            socket.emit('update-dashboard-result', result)
        }
        else {
            socket.emit('update-dashboard-result', false)
        }
        console.log('Socket.io update dashboard')
    })
})

socket.on('update-profile-settings', function(message){
    if(!checkToken(message.token))
        socket.emit('update-profile-settings-result', false)

    var sql = 'UPDATE device SET humidity_lower=?, humidity_upper=?,
airflow_duration=?, airflow_gap=?, light=? WHERE device_id =?'
    mysql.con.query(sql, [message.humidity_lower, message.humidity_upper,
message.fan_duration, message.fan_gap, message.light, message.device_id], function(err,
result){
        if(err)
            throw err
        if((result == undefined)){
            socket.emit('update-profile-settings-result',
result)
        }
        else {
            socket.emit('update-profile-settings-result', false)
        }
        console.log('Socket.io update profile settings')
    })

    var sql = 'SELECT devicename FROM device WHERE device_id =?'

```

```

mysql.con.query(sql, [message.device_id], function(err, result){
    if(err)
        throw err
    var mqtt_message = {
        device_id: message.device_id,
        devicename: result[0].devicename,
        humidity_lower: message.humidity_lower,
        humidity_upper: message.humidity_upper,
        airflow_duration: message.fan_duration,
        airflow_gap: message.fan_gap,
        light: message.light,
    }
    mosca_server.server.publish({topic: 'profile_settings',
payload: JSON.stringify(mqtt_message)}, function(){
        console.log('Mosca sent:', 'profile settings')
    })
})

socket.on('get-user', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('get-user-result', false)

    var sql = 'SELECT username, user_id FROM user'
    mysql.con.query(sql, function(err, result){
        if(err)
            throw err
        if((result === undefined)){
            socket.emit('get-user-result', result)
        }
        else {
            socket.emit('get-user-result', false)
        }
        console.log('Socket.io get user')
    })
})

socket.on('add-user', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('add-user-result', false)

    var sql = 'SELECT username FROM user WHERE username =?'
    mysql.con.query(sql, [message.username], function(err, result){
        if(result[0] === undefined){
            var sql = 'INSERT INTO user SET username =?, password =?,
admin =?'

            mysql.con.query(sql, [message.username,
message.password, message.admin], function(err, result){
                if(err)
                    throw err
                if((result === undefined)){

```

```

socket.emit('add-user-result',
'add')
    }
    else {
        socket.emit('add-user-result',
false)
    }
    console.log('Socket.io add user')
})
}
else {
    var sql = 'UPDATE user SET password =?, admin =?
WHERE username =?'
    mysql.con.query(sql, [message.password,
message.admin, message.username], function(err, result){
        if(err)
            throw err
        if((result === undefined)){
            socket.emit('add-user-result',
'update')
        }
        else {
            socket.emit('add-user-result',
false)
        }
        console.log('Socket.io update user')
    })
}
})

socket.on('delete-user', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('delete-user-result', false)

    var sql = 'DELETE FROM user WHERE user_id =?'
    mysql.con.query(sql, [message.user_id], function(err, result){
        if(err)
            throw err
        if((result === undefined)){
            socket.emit('delete-user-result', result)
        }
        else {
            socket.emit('delete-user-result', false)
        }
        console.log('Socket.io delete user')
    })
})

socket.on('get-device', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('get-device-result', false)

```

```

var sql = 'SELECT *FROM device WHERE ?'
mysql.con.query(sql, [message.where], function(err, result){
    if(err)
        throw err
    if((result === undefined)){
        socket.emit('get-device-result', result)
    }
    else {
        socket.emit('get-device-result', false)
    }
    console.log('Socket.io get device')
})

socket.on('add-device', function(message){
    if((checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('add-device-result', false)

    var sql = 'SELECT devicename FROM device WHERE devicename =?'
    mysql.con.query(sql, [message.devicename], function(err, result){
        if(result[0]===undefined){
            var sql = 'INSERT INTO device SET devicename=?,
humidity_lower =81, humidity_upper =90, airflow_duration =5, airflow_gap =35, light =0'
            mysql.con.query(sql, [message.devicename], function
(err, result){

                if(err)
                    throw err
                if((result ===undefined)){
                    socket.emit('add-device-
result', result)
                }
                else {
                    socket.emit('add-device-
result', false)
                }
                console.log('Socket.io add device')
            })
        }
        else
            socket.emit('add-device-result', 'exist')
    })
})

socket.on('delete-device', function(message){
    if((checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('delete-device-result', false)

    var sql = 'DELETE FROM device WHERE device_id =?'
    mysql.con.query(sql, [message.device_id], function(err, result){
        if(err)
            throw err
        if((result ===undefined)){

```

```

        socket.emit('delete-device-result', result)
    }
    else {
        socket.emit('delete-device-result', false)
    }
    console.log('Socket.io delete device')
})

socket.on('get-farm', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('get-farm-result', false)

    var sql = 'SELECT *FROM farm_owner WHERE ?'
    mysql.con.query(sql, [message.where], function(err, result){
        if(err)
            throw err
        if((result === undefined)){
            socket.emit('get-farm-result', result)
        }
        else {
            socket.emit('get-farm-result', false)
        }
        console.log('Socket.io get farm')
    })
})

socket.on('get-user-farm', function(message){
    if(!checkToken(message.token))
        socket.emit('get-user-farm-result', false)

    var sql = 'SELECT *FROM farm_owner WHERE user_id = ?'
    mysql.con.query(sql, [message.user_id], function(err, result){
        if(err)
            throw err
        if((result === undefined)){
            socket.emit('get-user-farm-result', result)
        }
        else {
            socket.emit('get-user-farm-result', false)
        }
        console.log('Socket.io get user farm')
    })
})

socket.on('add-user-farm', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('add-user-farm-result', false)

    var sql = 'SELECT farmname FROM farm_owner WHERE farmname = ?'
    mysql.con.query(sql, [message.farmname], function(err, result){
        if(result[0] === undefined){

```



```

var sql = 'INSERT INTO farm_owner SET user_id=?,
farmname =?'

mysql.con.query(sql, [message.user_id,
message.farmname], function(err, result){

    if(err)
        throw err
    if((result === undefined)){
        socket.emit('add-user-farm-
result', result)

    }
    else {
        socket.emit('add-user-farm-
result', false)

    }
    console.log('Socket.io add user farm')
})

}
else
    socket.emit('add-user-farm-result', 'exist')
})

})

socket.on('delete-user-farm', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('delete-user-farm-result', false)

    var sql = 'DELETE FROM farm_owner WHERE user_id=? AND farm_id=?'
    mysql.con.query(sql, [message.user_id, message.farm_id], function(err,
result){

        if(err)
            throw err
        if((result === undefined)){
            socket.emit('delete-user-farm-result', result)
        }
        else {
            socket.emit('delete-user-farm-result', false)
        }
        console.log('Socket.io delete user farm')
    })

})

socket.on('get-farm-device', function(message){
    if(!checkToken(message.token))
        socket.emit('get-farm-device-result', false)

    var sql = 'SELECT device.*FROM farm, device WHERE farm_id=? AND
farm.device_id=device.device_id'
    console.log('Socket.io get farm device')
    mysql.con.query(sql, [message.farm_id], function(err, result){
        if(err)
            throw err
        if((result === undefined)){

```

```

        socket.emit('get-farm-device-result', result)
    }
    else {
        socket.emit('get-farm-device-result', false)
    }
}
})

socket.on('add-farm-device', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('add-farm-device-result', false)

    var sql = 'SELECT device_id FROM farm WHERE farm_id =? AND device_id
=?'

    mysql.con.query(sql, [message.farm_id, message.device_id], function(err,
result){
        if(result[0]==undefined){
            var sql = 'INSERT INTO farm SET farm_id =?,
device_id =?'

            mysql.con.query(sql, [message.farm_id,
message.device_id], function(err, result){
                if(err)
                    throw err
                if((result == undefined)){
                    socket.emit('add-farm-device-
result', result)
                }
                else {
                    socket.emit('add-farm-device-
result', false)
                }
                console.log('Socket.io add farm device')
            })
        }
        else
            socket.emit('add-farm-device-result', 'exist')
    })
})

socket.on('delete-farm-device', function(message){
    if(!checkToken(message.token) || !checkAdmin(message.token))
        socket.emit('delete-farm-device-result', false)

    var sql = 'DELETE FROM farm WHERE farm_id =? AND device_id =?'
    console.log('Socket.io delete farm device')
    mysql.con.query(sql, [message.farm_id, message.device_id], function(err,
result){
        if(err)
            throw err
        if((result == undefined)){
            socket.emit('delete-farm-device-result', result)
        }
    })
})

```

```

                                else {
                                    socket.emit('delete-farm-device-result', false)
                                }
                            })
                        })
                    })
                })
            })
        })
    })
}

```

mosca-server.js

```

//require
var mosca = require('mosca')

//variable
var settings = {
    port:1883,
}

var server = new mosca.Server(settings)

//Accepts the connection if the username and password are valid
var authenticate = function(client, username, password, callback){
    var authorized =(username.toString()=='mqtt' && password.toString()=='naja')
    if(authorized){
        client.user = username
        callback(null, authorized)
    }
}

//function
function setup(){
    server.authenticate = authenticate
    console.log('Mosca server is up and running on 0.0.0.0:'+settings.port)
}

//object
server.on('clientConnected', function(client){
    console.log('Mosca client connected:', client.id)
})

server.on('clientDisconnected', function(client){
    console.log('Mosca client disconnected:', client.id)
})

server.on('published', function(packet, client){
    //console.log('Received data', '\n')
})

server.on('ready', setup)

//exports
exports.server = server

```

mysql.js

```
//require
var mysql = require('mysql')

//variable
var settings = {
  host:"mysql",
  port:"3306",
  user:"root",
  password:"eiei",
  database : 'mushroom_nursery'
}

var con = mysql.createConnection(settings)

//exports
exports.con = con

console.log('MySQL connected to mysql:'+settings.port)
```

web-backend.js

```
//require
var io = require('socket.io')(9000)

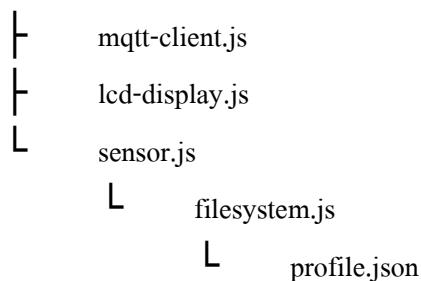
//exports
exports.io = io

console.log('Socket.io listening on 0.0.0.0:9000')
```

Raspberry Pi

เขียนด้วยภาษา JavaScript ใช้งานด้วย NodeJS โดยแบ่งเป็น 4 Module และ 1 JSON file
ดังนี้

main-client.js



main-client.js

```

//require
var mqtt_client = require('./mqtt-client.js')
var sensor = require('./sensor.js')
var filesystem = require('./filesystem.js')
var lcd_display = require('./lcd-display.js')

//variable
var profile = sensor.profile

//function
function sendLog() {
  var message = {
    device_id: profile.device_id,
    temperature: sensor.status.temperature,
    humidity: sensor.status.humidity,
    mist_pump: sensor.status.mist_pump,
    ventilate: sensor.status.ventilate,
    airflow_duration: sensor.status.airflow_duration,
    mist_duration: sensor.status.mist_duration,
    humidity_lower_set: profile.humidity_lower,
    humidity_upper_set: profile.humidity_upper,
    airflow_duration_set: profile.airflow_duration,
    airflow_gap_set: profile.airflow_gap,
    light_set: profile.light,
    message: 'OK'
  }

  mqtt_client.client.publish('logging', JSON.stringify(message), function() {
    console.log('MQTT sent:', 'log')
  })
}

setInterval(sendLog, 300000)

//object

```

```

mqtt_client.client.on('message', function(topic, message){
    if(topic === 'profile_settings' && JSON.parse(message).device_id ===
profile.device_id){
        console.log('MQTT get profile settings')
        profile = sensor.profile = JSON.parse(message)
        sensor.updateExportsProfile()
    }
})

lcd_display.io.on('connection', function(socket){
    console.log('Socket.io connected')
    socket.on('update-request', function(){
        sensor.updateExportsStatus()
        var message = {
            profile: profile,
            temperature: sensor.status.temperature,
            humidity: sensor.status.humidity,
            mist_pump: sensor.status.mist_pump,
            ventilate: sensor.status.ventilate
        }
        socket.emit('update-value', message)//Emit on the opened socket.
    })
    socket.on('control', function(message){
        sensor.control = message
        sensor.updateExportsControl()
        console.log(message)
    })
})

```

mqtt-client.js

```
//require
var mqtt = require('mqtt')

//variable
var settings = {
  clientId: 'raspi_01',
  username: 'mqtt',
  password: 'naja',
  reconnectPeriod: 1000 * 1
}

var client = mqtt.connect('mqtt://161.246.11.102', settings)

//object
client.on('connect', function() {
  console.log('MQTT connected to the server')
  client.subscribe('profile_settings', { qos: 2 })
  console.log('MQTT subscribed: profile_settings')
})

client.on('message', function(topic, message) {
  //console.log('MQTT (topic:', topic, ') received:', message.toString(), '\n')
})

client.on('error', function(error) {
  console.log('MQTT ERROR:', error)
})

client.on('offline', function() {
  console.log('MQTT offline')
})

client.on('reconnect', function() {
  console.log('MQTT reconnect')
})

//exports
exports.client = client
```

lcd-display.js

```
//require
var express = require('express')
var io = require('socket.io')(9000)

//variable
var settings = {
  port:80,
  env: '0.0.0.0'
}

//object
app = express()

app.use(express.static(__dirname + '/views'))

app.listen(settings.port, settings.env, function(){
  console.log('Express listening on', settings.env+':'+settings.port)
})

//exports
exports.app = app
exports.io = io
```


sensor.js

```

//require
var nodeDhtSensor = require('node-dht-sensor')
var gpio = require('onoff').Gpio
var filesystem = require('./filesystem.js')

//variable
var sensors =[
{name: 'DHT22-1', type:22, pin:6},
{name: 'DHT22-1', type:22, pin:13},
{name: 'DHT22-3', type:22, pin:19},
{name: 'DHT22-4', type:22, pin:26}
]

var relays =[
new gpio(12, 'out'),
new gpio(16, 'out'),
new gpio(20, 'out'),
new gpio(21, 'out')
]

var profile = filesystem.readFile()

var temperature =[0, 0, 0, 0]
var humidity =[0, 0, 0, 0]
var mist_pump =false
var ventilate =false
var airflow_duration =0
var mist_duration =0

var temperature_average
var humidity_average
var triggered =[0, 0]
var airflow =false
var airflow_duration_counter =0
var airflow_gap_counter =0

var control = {
    manual:false,
    fan:false,
    mist:false,
    light:false
}

var interval = {
    _1second:new Date(),
    _5second:new Date(),
    _60second:new Date()
}

//function

```

```

function readSensors () {
    for(var i in sensors){
        var read = nodeDhtSensor.read(sensors[i].type, sensors[i].pin)
        temperature[i]= read.temperature.toFixed(2)
        humidity[i]= read.humidity.toFixed(2)
    }
    temperature_average = (+temperature[0]*0.1++temperature[1]*0.3++temperature[2]*0.3+
+temperature[3]*0.3).toFixed(2)
    humidity_average = (+humidity[0]*0.3++humidity[1]*0.3++humidity[2]*0.1+
+humidity[3]*0.3).toFixed(2)
    writeRelays()
}

function writeRelays () {
    if(control.manual)
        return

    if(airflow){
        relays[1].writeSync(1)
        relays[2].writeSync(0)
        relays[3].writeSync(0)
        if(!ventilate)
            triggered[1]++
        mist_pump = false
        ventilate = true
    }
    elseif(humidity_average < profile.humidity_lower){
        relays[1].writeSync(1)
        relays[2].writeSync(1)
        relays[3].writeSync(1)
        if(!mist_pump)
            triggered[0]++
        if(!ventilate)
            triggered[1]++
        mist_pump = true
        ventilate = true
    }
    elseif(humidity_average > profile.humidity_upper){
        relays[1].writeSync(1)
        relays[2].writeSync(0)
        relays[3].writeSync(0)
        if(!ventilate)
            triggered[1]++
        mist_pump = false
        ventilate = true
    }
    else {
        relays[1].writeSync(0)
        relays[2].writeSync(0)
        relays[3].writeSync(0)
        mist_pump = false
    }
}

```

```

        ventilate = false
    }

    if(profile.light)
        relays[0].writeSync(1)
    else
        relays[0].writeSync(0)
}

function manualWriteRelays () {
    if(!control.manual)
        return

    if(control.fan) {
        relays[1].writeSync(1)
        if(!ventilate)
            triggered[1]++
        ventilate = true
    }
    else {
        relays[1].writeSync(0)
        ventilate = false
    }

    if(control.mist) {
        relays[2].writeSync(1)
        relays[3].writeSync(1)
        if(!mist_pump)
            triggered[0]++
        mist_pump = true
    }
    else {
        relays[2].writeSync(0)
        relays[3].writeSync(0)
        mist_pump = false
    }

    if(control.light)
        relays[0].writeSync(1)
    else
        relays[0].writeSync(0)
}

function consoleOutput () {
    console.log('\n', Date())
    console.log('Profile:', JSON.stringify(profile))
    console.log('DHT22-1\tTemperature:', temperature[0]+'°C', 'Humidity:',
humidity[0]+'%')
    console.log('DHT22-2\tTemperature:', temperature[1]+'°C', 'Humidity:',
humidity[1]+'%')
    console.log('DHT22-3\tTemperature:', temperature[2]+'°C', 'Humidity:',
humidity[2]+'%')
}

```

```

        console.log('DHT22-4\tTemperature:', temperature[3]+'°C', 'Humidity:',
humidity[3]+'%')
        console.log('Humidity:', humidity_average+'%', '    Triggered:', triggered)
        console.log('MistPump:', mist_pump, '    Ventilate:', ventilate)
        console.log('MistDuration:', mist_duration, '    AirflowDuration:',
airflow_duration, '\n')
    }

    function timer () {
        var time = new Date()
        if (time - interval_1second >= 1000) {
            interval_1second = +interval_1second + 1000
            if (mist_pump)
                mist_duration++
            if (ventilate)
                airflow_duration++
        }

        if (time - interval_5second >= 5000) {
            interval_5second = +interval_5second + 5000
            readSensors()
            consoleOutput()
        }

        if (time - interval_60second >= 60000) {
            interval_60second = +interval_60second + 60000
            if (airflow_duration == 1) {
                airflow = true
                console.log('Sensor start airflow')
            }
            else if (airflow_duration == 0) {
                airflow = false
                console.log('Sensor stop airflow')
            }
            else if (airflow) {
                airflow_duration_counter++
                if (airflow_duration_counter >= profile.airflow_duration) {
                    airflow_duration_counter = 0
                    airflow = false
                    console.log('Sensor stop airflow')
                }
            }
            else {
                airflow_gap_counter++
                if (airflow_gap_counter >= profile.airflow_gap) {
                    airflow_gap_counter = 0
                    airflow = true
                    console.log('Sensor start airflow')
                }
            }
        }
    }
}

```

```

setInterval(timer, 100)

function unexportOnClose (){
    for(var i in relays){
        relays[i].writeSync(0)
        console.log('Sensor unexport relay:', i)
    }
    process.exit()
}
process.on('SIGINT', unexportOnClose)

//exports
exports.profile = profile
exports.status = {
    temperature: temperature_average,
    humidity: humidity_average,
    mist_pump: mist_pump,
    ventilate: ventilate,
    airflow_duration: airflow_duration,
    mist_duration: mist_duration
}
exports.control = control

exports.updateExportsProfile =function(){
    profile = exports.profile
    filesystem.writeFile(profile)
    console.log('Sensor profile settings updated')
    if(profile.airflow_duration ==1){
        airflow =true
        console.log('Sensor start airflow')
    }
    elseif(profile.airflow_duration ==0){
        airflow =false
        console.log('Sensor stop airflow')
    }
}

exports.updateExportsStatus =function(){
    exports.status = {
        temperature: temperature_average,
        humidity: humidity_average,
        mist_pump: mist_pump,
        ventilate: ventilate,
        airflow_duration: airflow_duration,
        mist_duration: mist_duration
    }
}

exports.updateExportsControl =function(){
    control =exports.control
    manualWriteRelays()
}

console.log('Sensor activated')

```

filesystem.js

```
//require
var fs = require('fs')

//variable
var filename = 'profile.json'

//function
function readFile () {
    console.log('Filesystem read:', filename)
    return JSON.parse(fs.readFileSync(filename))
}

function writeFile (data) {
    fs.writeFile(filename, JSON.stringify(data), function(err) {
        if(err) throw err
        console.log('Filesystem write:', filename)
    })
}

//exports
exports.readFile = readFile
exports.writeFile = writeFile
```

profile.json

```
{
    "device_id":0,
    "devicename":"Raspi",
    "humidity_lower":81,
    "humidity_upper":90,
    "airflow_duration":5,
    "airflow_gap":25,
    "light":false
}
```

docker-compose.yaml

```
version: '3'
services:
  nginx-php:
    image: richarvey/nginx-php-fpm
    ports:
      - "80:80"
    volumes:
      - "/root/platform/nginx-php/var/www/html:Z"
    restart: always
    stdin_open: true
    tty: true
  mysql:
    image: mysql
    ports:
      - "3306:3306"
    volumes:
      - "/root/platform/mysql/var/lib/mysql:Z"
    environment:
      MYSQL_ROOT_PASSWORD: "eiei"
      MYSQL_ALLOW_EMPTY_PASSWORD: "no"
    restart: always
    stdin_open: true
    tty: true
  node:
    image: node
    ports:
      - "1883:1883"
      - "9000:9000"
    volumes:
      - "/root/platform/node/root/node:Z"
    restart: always
    stdin_open: true
    tty: true
```

ภาคผนวก ข

Deployment

Technology Requirement

Hardware

Server

Raspberry Pi Model B+

Operating System

CentOS

NOOBS

Engine

Docker

Docker-compose

NodeJS

MySQL

MQTT

Nginx

Language

Bash

JavaScript

HTML

CSS

SQL

Installation Method Command

Platform

```

yum update -y
yum install screen docker -y

dpkg-reconfigure tzdata
SET GLOBAL time_zone = 'Asia/Bangkok';
date +%T -s "0:00:00"

#upload phpmyadmin, create phpmyadmin database, generate blowfish passphrase

curl "https://bootstrap.pypa.io/get-pip.py"-o "get-pip.py"
python get-pip.py
pip install docker-compose

#add docker-compose.yaml+necessary files and change to that directory

docker-compose create
docker-compose start

screen -md -S node docker exec -it platform_node_1 bash
cd /root/node
npm init
npm install
node main-server.js

#ctrl+A then ctrl+D for detach and run on background

```

Raspberry Pi

```

apt-get update -y
apt-get install screen vim nodejs-y

dpkg-reconfigure tzdata
SET GLOBAL time_zone = 'Asia/Bangkok';
date +%T -s "0:00:00"

curl -o bcm2835-1.55.tar.gz http://www.airspayce.com/mikem/bcm2835/bcm2835-1.55.tar.gz
tar zxvf bcm2835-1.55.tar.gz
cd bcm2835-1.55
./configure
make && make check
make install

#add necessary files and change to that directory

npm init
npm install
screen -md -S client node main-client.js
screen -md -S browser chromium-browser --start-fullscreen

```

Create Database Command

```

CREATE DATABASE mushroom_nursery;

USE mushroom_nursery;

CREATE TABLE user(
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(32),
    password VARCHAR(32),
    admin BOOLEAN,
    token VARCHAR(32)
);

CREATE TABLE farm_owner (
    user_id INT UNIQUE KEY,
    farm_id INT UNIQUE KEY PRIMARY KEY AUTO_INCREMENT,
    farmname VARCHAR(32)
);

CREATE TABLE farm (
    farm_id INT UNIQUE KEY,
    device_id INT UNIQUE KEY
);

CREATE TABLE device (
    device_id INT PRIMARY KEY AUTO_INCREMENT,
    devicename VARCHAR(32),
    humidity_lower FLOAT,
    humidity_upper FLOAT,
    airflow_duration INT,
    airflow_gap INT,
    light BOOLEAN
);

CREATE TABLE log (
    timestamp TIMESTAMP PRIMARY KEY,
    device_id INT,
    temperature FLOAT,
    humidity FLOAT,
    mist_pump BOOLEAN,
    ventilate BOOLEAN,
    airflow_duration INT,
    mist_duration INT,
    humidity_lower_set FLOAT,
    humidity_upper_set FLOAT,
    airflow_duration_set INT,
    airflow_gap_set INT,
    light_set BOOLEAN,
    message VARCHAR(100)
);

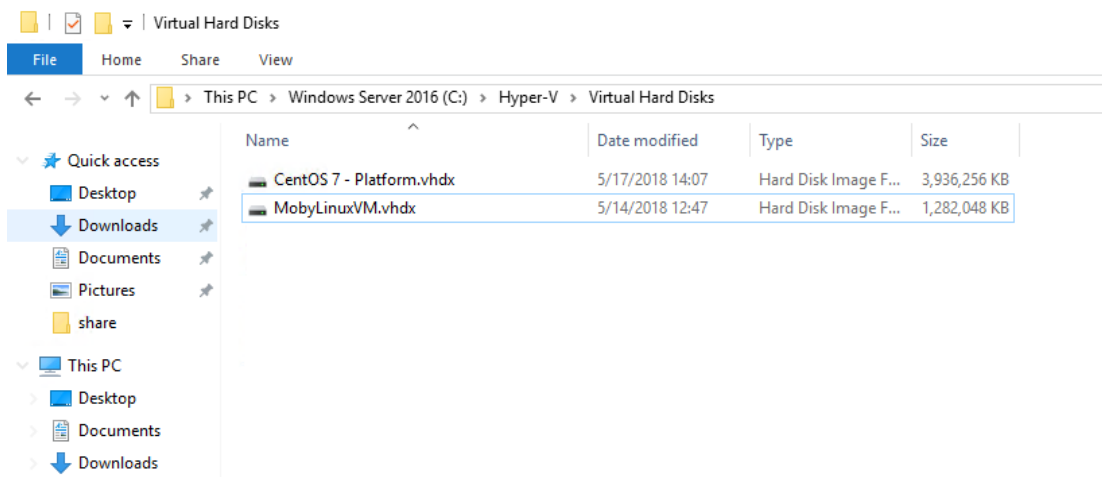
```

ภาคผนวก ค

Usage

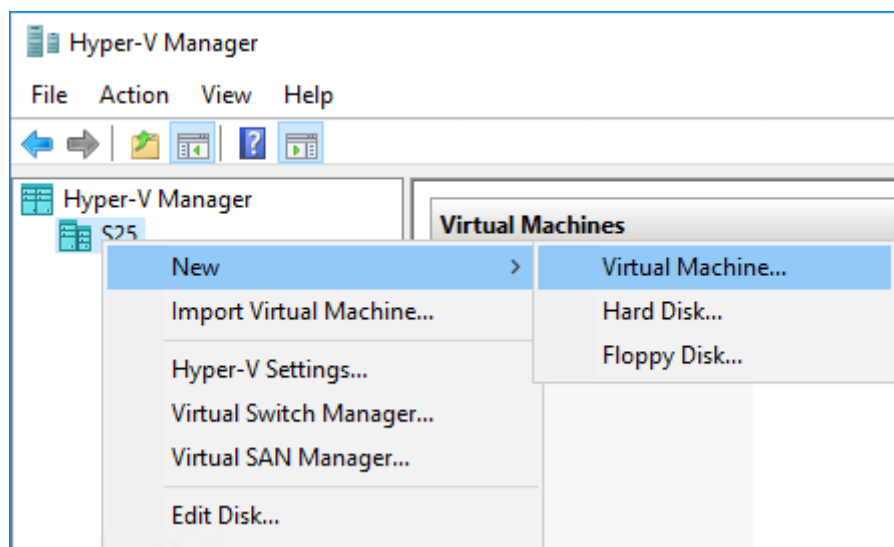
Platform

1. เตรียม HyperV และไฟล์ CentOS 7 - Platform.vhdx



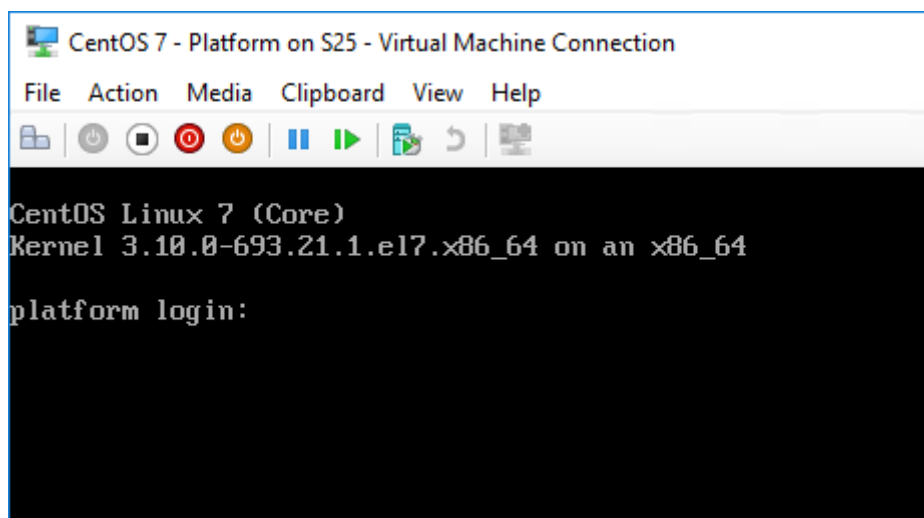
รูปที่ ค.1 ไฟล์ CentOS 7 - Platform.vhdx

2. เข้าไปที่ Hyper-V Manager คลิกขวาที่ Server -> New -> Virtual Machine... ตั้งค่าและเพิ่ม Virtual Disk ให้เรียบร้อย



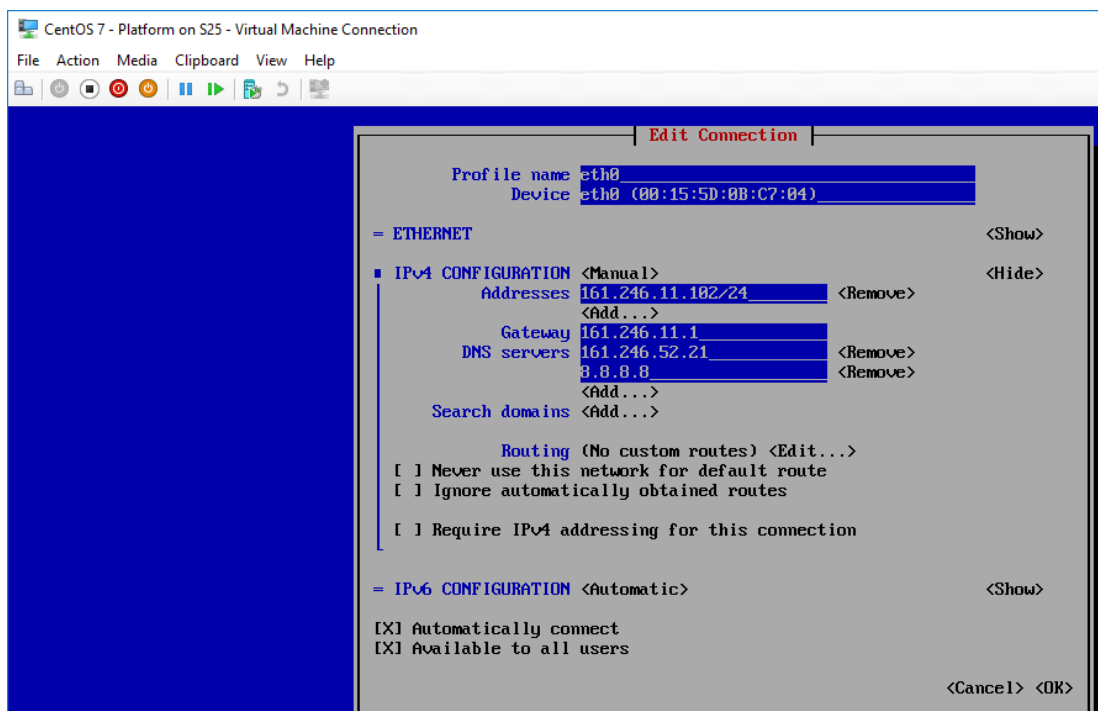
รูปที่ ค.2 เพิ่ม Virtual Machine

3. จากนั้น Connect และ Start Virtual Machine ใส่ Username คือ root และ Password คือ eiei



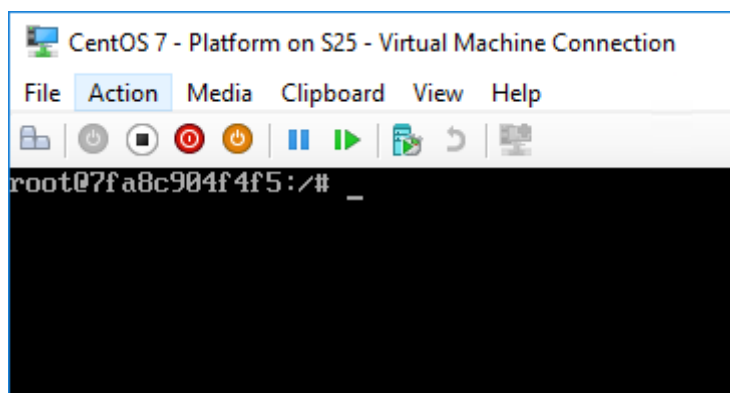
รูปที่ ค.3 Virtual Machine Interface

4. ใช้คำสั่ง `nmtui` จากนั้น Edit a Connection แก้ไข IP, Subnet และ Gateway บน Network Interface ตามต้องการ



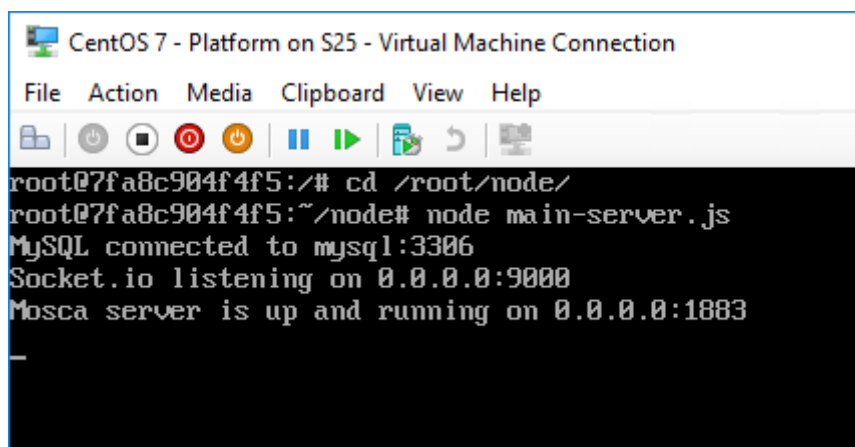
รูปที่ ค.4 Network Configuration Using a Text User Interface (nmtui)

5. ใช้คำสั่ง `screen -S node docker exec -it platform_node_1 bash` เพื่อเข้ามาจัดการ NodeJS



รูปที่ ค.5 Bash in NodeJS Container

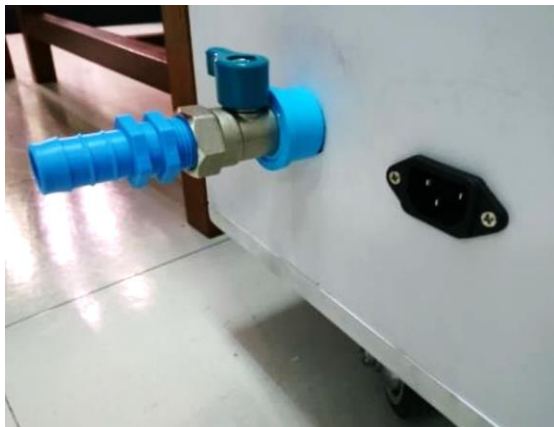
6. ใช้คำสั่ง `cd /root/node` และ `node main-server.js` เพื่อ run service (กด Ctrl+A แล้ว Ctrl+D เพื่อ detach และ run in background)
(กด Ctrl+C เพื่อ terminate)
(ใช้คำสั่ง `screen -r node` เพื่อกลับเข้ามาจัดการ NodeJS)



รูปที่ ค.6 Run Service on Bash in NodeJS Container

Microcontroller

1. เสียบปลั๊กไฟฟ้า และเสียบสายยางหรือเติมน้ำในถัง 5 ลิตรขึ้นไป ให้เรียบร้อย



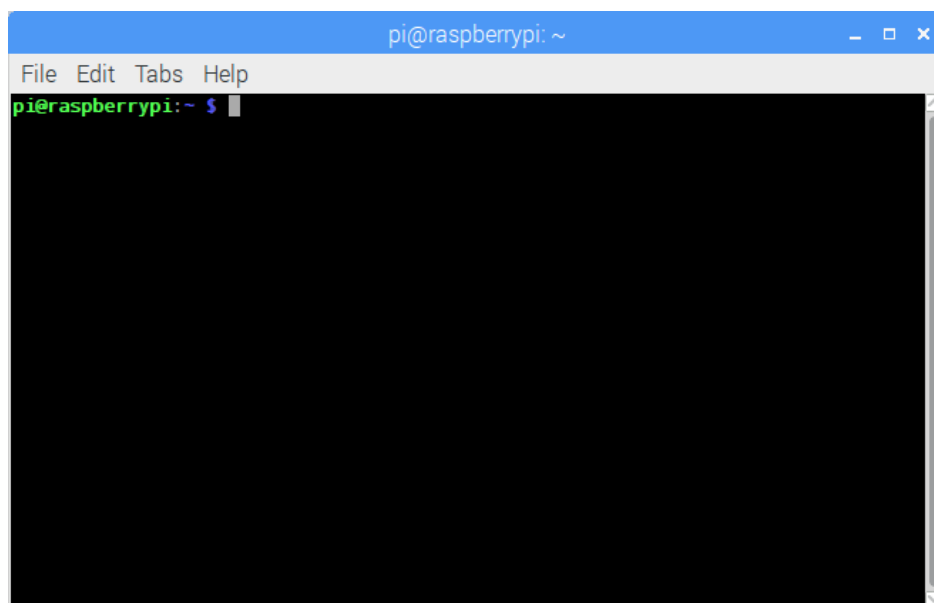
รูปที่ ค.7 ด้านซ้ายมุมซ้ายล่างของตู้จำลอง

2. ระบบจะทำงานอัตโนมัติ และขึ้นภาพดังรูป



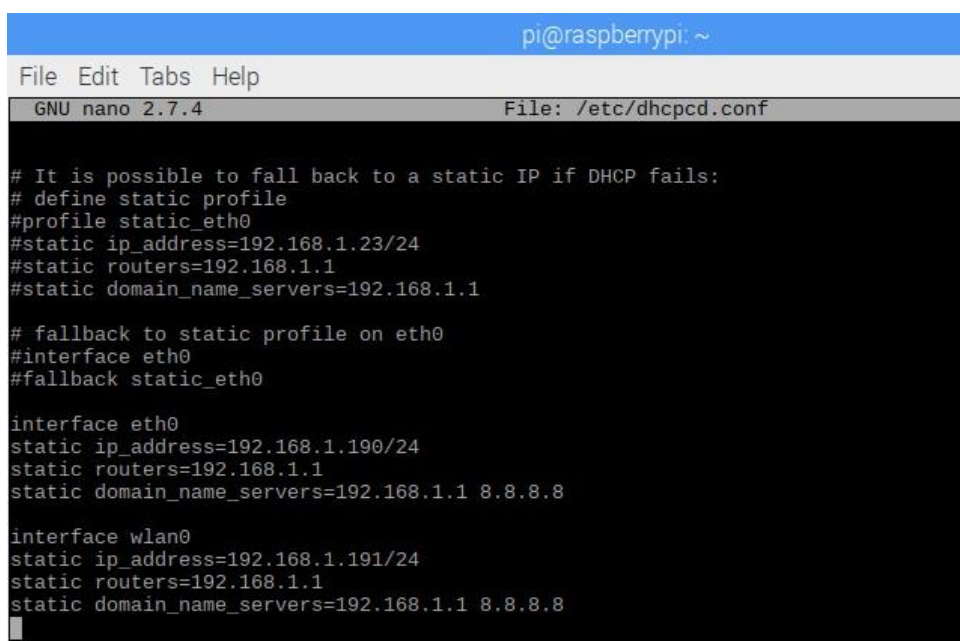
รูปที่ ค.8 ด้านบนของตู้จำลอง

3. ต่อคีย์บอร์ด เข้า terminal โดยกด Ctrl + Alt + T ให้ Username คือ pi และ Password คือ eiei



รูปที่ ค.9 Raspberry Pi Terminal Interface

4. ใช้คำสั่ง `nano /etc/dhcpd.conf` จากนั้น แก้ไข IP, Subnet และ Gateway บน Network Interface ตามต้องการ และใช้คำสั่ง `systemctl restart networking`



รูปที่ ค.10 Network Configuration

5. ใช้คำสั่ง `cd/client` แล้วใช้คำสั่ง `vim` แก้ไขทุก Module ที่มี IP ของ Platform ให้ตรงกับ IP ของ Platform ที่ใช้อยู่ปัจจุบัน

6. ใช้คำสั่ง `screen -S node node main-client.js` เพื่อ run service
(กด Ctrl+A แล้ว Ctrl+D เพื่อ detach และ run in background)
(กด Ctrl+C เพื่อ terminate)
(ใช้คำสั่ง `screen -r node` เพื่อกลับเข้ามาจัดการ NodeJS)

Web Application

Web Application สำหรับใช้แสดงสถานะและควบคุมระบบต่างๆภายในโรงเรือน โดยจะแบ่งออกเป็น 4 ส่วน คือ หน้า Login, หน้า Dashboard, หน้า Profile และหน้า Admin Panel

หน้า Login สำหรับใส่ Username และ Password เพื่อเข้าใช้งาน เมื่อ login เข้ามาแล้วจะเจอหน้า Dashboard ที่จะแสดงอุณหภูมิ และความชื้นจากค่าที่วัดได้ภายในโรงเรือน พัดลมระบายอากาศ และเครื่องทำหมอกจะแสดงระยะเวลาทำงานทั้งหมดเป็นวินาที และกราฟแสดงรายละเอียดย้อนหลังสำหรับอุณหภูมิและความชื้น โดยสามารถเลือก Farm และ Device ได้ว่าจะแสดงรายละเอียดต่างๆจาก Device ไหน

หน้า Profile มีไว้สำหรับตั้งค่า Profile สำหรับใช้ในการควบคุมระบบต่างๆภายในโรงเรือน โดยทำการเลือก Farm และ Device จากนั้นทำการตั้งค่า ช่วงของความชื้น ระยะเวลาการเวลาในการเปิดพัดลมระบายอากาศต่อวัน และสามารถเลือกได้ว่าจะเปิดไฟหรือไม่

หน้า Admin Panel จะสามารถใช้ได้เฉพาะผู้ใช้งานที่มีสถานะผู้ดูแลระบบเท่านั้น โดยจะแบ่งเป็น 3 หน้า ในหน้าแรกจะเป็นหน้าสำหรับการจัดการผู้ใช้งาน โดยสามารถเพิ่มหรือลบผู้ใช้งานออกจากระบบได้ และในการเพิ่มผู้ใช้งานสามารถกำหนดได้ว่าจะเป็นผู้ใช้งานธรรมดาหรือหรือผู้ดูแลระบบหน้าต่อไปจะเป็นหน้าสำหรับการจัดการ Device โดยสามารถเพิ่มหรือลบ Device ออกจากระบบได้ หน้าสุดท้ายจะเป็นหน้าสำหรับการจัดการฟาร์ม โดยการเลือกผู้ใช้งานที่มีอยู่ในระบบและจะสามารถเพิ่มหรือลบฟาร์มสำหรับผู้ใช้งานคนนั้นได้ หรือทำการเลือกฟาร์มที่มีอยู่ในระบบและสามารถเพิ่มหรือลบ