



คู่มือการใช้งาน NETPIE

An Official Guide to NETPIE



เอกสารฉบับนี้เป็นลิขสิทธิ์ของศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

เอกสารฉบับนี้เผยแพร่ภายใต้สัญญาอนุญาตแบบ

Creative Commons Attribution 4.0 International License



อนุญาตให้นำผลงานไปใช้ได้ แก้ไขได้ ดัดแปลงเนื้อหาได้ ใช้ในเชิงพาณิชย์ได้ โดยอ้างอิงที่มา
คร่วยละเวียดสัญญาอนุญาตที่ <https://creativecommons.org/licenses/by/4.0/>

เอกสารเวอร์ชัน 2017-02-02

สามารถดาวน์โหลดไฟล์เอกสารนี้ได้ที่ <https://netpie.io/tutorials>

สารบัญ

บทนำ.....	6
1. แนะนำ NETPIE	7
1.1. รู้จัก NETPIE.....	7
1.2. Microgear.....	8
1.3. MQTT (MQ Telemetry Transport).....	11
แบบฝึกหัด: Local MQTT Chat บน Raspberry Pi.....	19
1.4. Authorization และ Authentication ใน NETPIE	21
2. Getting Started.....	25
2.1 การสมัครสมาชิก.....	25
2.2 การสร้างแอปพลิเคชัน	26
2.3 การติดตั้งเครื่องมือพัฒนา	30
3. HTML 5 Microgear.....	36
Lab 3.1: Hello NETPIE.....	36
Lab 3.2: HTMLChat.....	38
Lab 3.3: Room Chat.....	40
4. ESP8266 Microgear	46
4.1 รู้จัก ESP8266/NodeMCU	46
Lab 4.1: NETPIE Blink	48
Lab 4.2: IoT Switch	52
5. Freeboard.....	57
Lab 5.1: การแสดงผลข้อมูลจากอุปกรณ์เซนเซอร์บน NETPIE Freeboard	62
Lab 5.2: การควบคุมอุปกรณ์ด้วย NETPIE Freeboard	68
6. NETPIE FEED	74
6.1 การสร้าง FEED	74

6.2	การกำหนดสิทธิ์ในการเข้าถึง feed	76
6.3	การเขียนข้อมูลลงใน Feed	78
6.4	การดึงข้อมูลจาก feed มาใช้งาน	80
6.5	ข้อจำกัดการใช้งาน.....	89
	Lab 6.1: การแสดงผลข้อมูลเซ็นเซอร์ด้วย NETPIE Feed	89
7.	NETPIE REST API	94
7.1	REST API.....	94
7.2	NETPIE REST API.....	94
	Lab 7.1: HTML → HTML.....	97
	Lab 7.2: HTML → NodeMCU.....	101
	Lab 7.3: KKMC IoT	105
8.	NETPIE Secure Connection	109
	Lab 8.1: สร้าง Secure Connection ระหว่าง NodeMCU และ HTML5	111
	ภาคผนวก NETPIE Microgear Reference Guide	118
1.	ESP8266-Arduino Microgear	118
1.1	ความเข้ากันได้.....	118
1.2	พอร์ตสื่อสาร	118
1.3	การติดตั้ง.....	118
1.4	ข้อจำกัดที่พบ	119
1.5	ตัวอย่างการเรียกใช้.....	119
1.6	การใช้งาน library.....	122
2.	Arduino-Ethernet Microgear	126
2.1	ความเข้ากันได้.....	126
2.2	ตัวอย่างการเรียกใช้.....	126
2.3	การใช้งาน library.....	128
3.	Node.js Microgear	132

3.1	พอร์ตสื่อสาร	132
3.2	การติดตั้ง	132
3.3	ตัวอย่างการเรียกใช้	132
3.4	การใช้งาน library	133
4.	HTML5 Microgear	138
4.1	การรองรับ	138
4.2	พอร์ตสื่อสาร	139
4.3	การติดตั้ง	139
4.4	ตัวอย่างการเรียกใช้	139
4.5	การใช้งาน library	140
5.	Python Microgear	144
5.1	การติดตั้ง	145
5.2	ตัวอย่างการเรียกใช้งาน	145
5.3	ตัวอย่างเพิ่มเติม	145
5.4	การใช้งาน library	145
6.	REST API	152
6.1	API Endpoint	152
6.2	Authentication	152
6.3	Resource Types	153
	ผู้เขียน	156

บทนำ

ในปัจจุบันเทคโนโลยี Internet of Things หรือ IoT เริ่มเข้ามามีบทบาทสำคัญในชีวิตประจำวัน IoT คือ สภาพแวดล้อมอันประกอบด้วยสิ่ง (Things) ที่สามารถสื่อสารและเชื่อมต่อกันได้ผ่านโทรศัพท์คอมพิวเตอร์ การสื่อสารทั้งแบบใช้สายและไร้สาย โดยสิ่งต่างๆ มีวิธีการระบุตัวตนได้ รับรู้ข้อมูลของสภาพแวดล้อมได้ และมีปฏิสัมพันธ์ได้ตอบและทำงานร่วมกันได้

ปัจจุบันเราสามารถนำเทคโนโลยี IoT มาประยุกต์ในด้านต่างๆ โดยเพิ่มขีดความสามารถของอุปกรณ์เครื่องใช้หรือบริการ ให้สามารถรับรู้ และเปลี่ยนข้อมูล แสดงผล ควบคุมหรือทำงานร่วมกันได้ โดย ก้าวข้ามขีดจำกัดในเรื่องของเวลาและสถานที่ เป็นการบูรณาการเทคโนโลยีต่างๆ เช่น การสื่อสารเครือข่ายคอมพิวเตอร์ อุปกรณ์ไมโครคอนโทรลเลอร์ อุปกรณ์ระบบสมองกลฝังตัว อุปกรณ์เซนเซอร์ และ ข้อมูล เข้าด้วยกัน และด้วยราคาของอุปกรณ์ต่างๆ ที่ถูกลงส่วนทางกับสมาร์ตโฟนที่เดี๋ยวนี้ ทำให้การพัฒนาต่อไป ยอดผลิตภัณฑ์หรือบริการที่เป็น IoT เป็นไปได้โดยง่าย

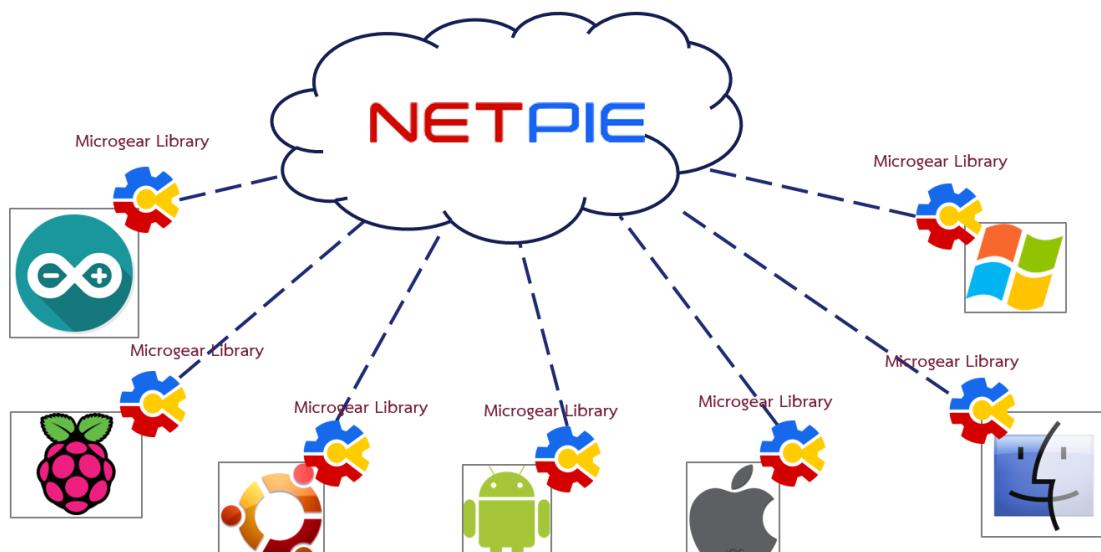
NETPIE เป็นแพลตฟอร์มให้บริการสำหรับ IoT เป็นระบบพื้นฐานที่อำนวยความสะดวกให้กับนักพัฒนา นักพัฒนาไฮาร์ดแวร์สามารถพัฒนาอุปกรณ์ โดยไม่ต้องกังวลเรื่องการติดตั้งดูแลระบบสื่อสารระบบเซิร์ฟเวอร์ หรือฐานข้อมูลใดๆ ในขณะเดียวกันแพลตฟอร์มนี้ยังช่วยให้นักพัฒนาซอฟต์แวร์เข้าถึง อุปกรณ์อิเล็กทรอนิกส์ต่างๆ ได้ง่ายขึ้น ผ่านไลบรารีสำเร็จรูปที่แพลตฟอร์มเตรียมไว้ให้ ดังนั้นบริการแพลตฟอร์ม NETPIE จึงเป็นเสมือนสะพานเชื่อมระหว่างนักพัฒนาไฮาร์ดแวร์และนักพัฒนาซอฟต์แวร์

เอกสารนี้อธิบายการประยุกต์ใช้ IoT เป็นต้นด้วย NETPIE คณจะได้พบความรู้ ความเข้าใจเกี่ยวกับเทคโนโลยี IoT เช่น จากระบบงานการพัฒนาผลิตภัณฑ์ บริการ และแอปพลิเคชันเบื้องต้น โดยใช้แพลตฟอร์มสื่อสาร NETPIE ในการเชื่อมต่อกับอุปกรณ์และแลกเปลี่ยนข้อมูลกันได้ในรูปแบบของ IoT เพื่อให้เทคโนโลยีที่เกี่ยวข้องกับ Internet of Things ได้รับการพัฒนาและต่อยอดเป็นนวัตกรรม เกิดการสร้างสรรค์พัฒนาผลิตภัณฑ์ บริการ และแอปพลิเคชันต่างๆ IoT โดยผู้ประกอบการไทย และเพิ่มขีดความสามารถในการแข่งขันกับนานาประเทศ

1. แนะนำ NETPIE

1.1. รู้จัก NETPIE

NETPIE เป็น IoT (Internet of Things) Cloud Platform ที่พัฒนาขึ้นโดยทีมงานวิจัยและเปิดให้บุคคลทั่วไปใช้งานโดยมี Web Portal ที่ให้สามารถลงทะเบียนและจัดการตัวตนและสิทธิ์ของแอปพลิเคชัน และอุปกรณ์ได้ที่เว็บไซต์ <https://netpie.io> ตั้งแต่เดือนกันยายน 2558 เป็นต้นมา NETPIE เป็น Middleware ที่มีหัวใจหลัก (นอกเหนือจากส่วนอื่นๆ) เป็น Distributed MQTT Brokers ซึ่งเป็นเสมือนจุดนัดพบให้สิ่งต่างๆ (Things) มาติดต่อสื่อสารและทำงานร่วมกันผ่านวิธีการส่งข้อความแบบ Publish/Subscribe NETPIE มีโครงสร้างสถาปัตยกรรมเป็นคลาวด์อย่างแท้จริงในทุกองค์ประกอบ ทำให้สามารถขยายตัวได้อย่างอัตโนมัติ (Auto-scale) สามารถดูแลและซ่อมแซมตัวเองได้อัตโนมัติเมื่อส่วนหนึ่งส่วนใดในระบบมีปัญหา (Self-healing, Self-recovery) โดยไม่ต้องพึ่งผู้ดูแลระบบ การบริหารจัดการระบบ เป็นแบบ Plug-and-Play ไม่ต้อง Configure หรือปรับแต่ง ในฝั่งคุปกรณ์ NETPIE มี Client Library หรือที่เรียกว่า Microgear ซึ่งทำหน้าที่สร้างและดูแลช่องทางสื่อสารระหว่างอุปกรณ์กับ NETPIE รวมไปถึงรักษาความปลอดภัยในการส่งข้อมูล Microgear เป็น Open Source และสามารถดาวน์โหลดได้จาก <https://github.com/netpieio> โดยณ ปัจจุบันมี Microgear สำหรับ OS และ Embedded Board หลักๆ ที่เป็นที่นิยมในหมู่นักพัฒนาเกือบทุกชนิด ไม่เดลาการสื่อสารของ NETPIE แสดงไว้ในรูปที่ 1.1



รูปที่ 1.1 วิธีการสื่อสารของสิ่งต่างๆ ผ่าน NETPIE

ประโยชน์ของ NETPIE

1. ช่วยลดการใช้ทรัพยากรของเครื่องคอมพิวเตอร์

NETPIE ช่วยให้อุปกรณ์สามารถสื่อสารกันได้โดยผู้ใช้ไม่ต้องกังวลว่า อุปกรณ์นั้นจะอยู่ที่ใด เพียงแค่ นำ Microgear Library ไปติดตั้งในอุปกรณ์ NETPIE จะรับหน้าที่ดูแลเชื่อมต่อให้ทั้งหมด ไม่ว่าอุปกรณ์นั้น จะอยู่ในเครือข่ายชนิดใด ลักษณะใด หรือแม้กระทั่งเคลื่อนย้ายไปอยู่ที่ใด ผู้ใช้สามารถตัดปัญหาในการเข้าถึงอุปกรณ์จากระยะไกล (Remote Access) ด้วยวิธีการแบบเดิมๆ เช่น การใช้ Fixed Public IP Address หรือการตั้ง Port Forwarding ในเราท์เตอร์และการต้องเปลี่ยนกับผู้ให้บริการ Dynamic DNS ซึ่งทั้งหมดล้วนมีความยุ่งยาก ลดความยืดหยุ่นของระบบ ไม่เพียงเท่านั้น NETPIE ยังช่วยให้การเริ่มต้นใช้งานเป็นไปโดยง่าย โดยออกแบบให้อุปกรณ์ถูกค้นพบและเข้าสู่บริการโดยอัตโนมัติ (Automatic Discovery, Plug-and-Play)

2. ช่วยลดภาระด้านความปลอดภัยของข้อมูล

NETPIE ถูกออกแบบให้มีระดับและสิทธิ์ในการเข้าถึงในระดับ Fine Grain กล่าวคือผู้ใช้สามารถออกแบบได้เองทั้งหมดว่า สิ่งใดมีสิทธิ์คุยกับสิ่งใด สิ่งใดมีสิทธิ์หรือไม่ - เพียงใดในการอ่านหรือเขียนข้อมูล และสิทธิ์เหล่านี้จะมีอยู่การใช้งานนานเท่าใด หรือจะถูกเพิกถอนภายใต้เงื่อนไขใด เป็นต้น

3. ยืดหยุ่นต่อการขยายระบบ

NETPIE มีสถาปัตยกรรมเป็นคลาวด์เซิร์ฟเวอร์อย่างแท้จริงในทุกองค์ประกอบของระบบ ทำให้เกิดความยืดหยุ่นและคล่องตัวสูงในการขยายตัว นอกเหนือไปจากความสามารถในการอ่านและเขียนข้อมูลต่างๆ ยังถูกออกแบบให้ทำงานแยกจากกัน เพื่อให้เกิดสภาวะ Loose Coupling และสื่อสารกันด้วยวิธี Asynchronous Messaging ช่วยให้แพลตฟอร์มมีความน่าเชื่อถือได้สูง นำไปใช้สำหรับพัฒนาต่อได้ง่าย ดังนั้นผู้พัฒนาไม่จำเป็นต้องกังวลกับการขยายตัวเพื่อรับโหลดที่เพิ่มขึ้นในระบบอีกด้วย

1.2. MICROGEAR

Microgear คือซอฟต์แวร์ labore ของ NETPIE ที่ติดตั้งอยู่บนอุปกรณ์ที่ต้องการเชื่อมต่อสื่อสารผ่านคลาวด์ของ NETPIE Microgear เปรียบเสมือนตัวกลางและผู้ช่วยในการสร้างและดูแลการเชื่อมต่อ ให้มีความเสถียร ปลอดภัย ให้การสื่อสารและเปลี่ยนข้อมูลระหว่างอุปกรณ์เป็นไปอย่างราบรื่น บทบาทหน้าที่ของ Microgear สามารถแบ่งออกเป็น 4 ด้านคือ

- ด้านการสื่อสาร (Communication) Microgear จะเป็นผู้ช่วยในการสร้างการเชื่อมต่อ (Connection) ไปยังคลาวด์ของ NETPIE และคอยตรวจสอบสถานะของการเชื่อมต่อ หากการเชื่อมต่อไม่ปัญหา Microgear สามารถช่วยเชื่อมต่อให้ใหม่เพื่อให้การสื่อสารเป็นไปได้อย่าง

จับรีน นอกจากนี้ Microgear ยังช่วยอำนวยความสะดวก ในการสร้างช่องทางการสื่อสารแบบเข้ารหัสในกรณีที่ผู้ใช้ต้องการ ส่วนการแลกเปลี่ยนข้อมูลระหว่าง Microgear และคลาวด์ของ NETPIE จะใช้proto協議 MQTT ใน การสื่อสาร

2. ด้านการยืนยันตัวตน (Authentication) ในขั้นตอนการสร้างการเชื่อมต่อ Microgear จะช่วยยืนยันตัวตนของคุปกรณ์บุคคลาดีของ NETPIE โดยการพิสูจน์ตัวตน (Identity) ของคุปกรณ์จะใช้ข้อมูลประกอบกันสามส่วนคือ AppID, App Key และ Token
3. ด้านการขออนุญาตสิทธิ์ (Authorization) การขออนุญาตสิทธิ์ในการสื่อสารจะเกิดขึ้นในขั้นตอนการสร้างการเชื่อมต่อ ควบคู่กับการยืนยันตัวตน คลาวด์ของ NETPIE จะเป็นผู้ออกใบอนุญาต (Token) ที่ระบุว่าคุปกรณ์ตัวนี้ สามารถสื่อสารได้กับคุปกรณ์ตัวใดบ้าง ในกรณีปกติคุปกรณ์ที่อยู่ภายใต้กลุ่ม AppID เดียวกันเท่านั้น จึงจะมีสิทธิ์สื่อสารกันได้ (ยกเว้นในกรณีการใช้ Freeboard Microgear ที่อนุญาตให้สื่อสารข้าม AppID ได้ ซึ่งจะอธิบายในบทที่ 5)
4. ด้านการประสานงาน (Coordination) Microgear มีฟังก์ชันที่ช่วยให้คุปกรณ์ต่างๆ ภายนอกกลุ่ม AppID เดียวกันทราบสถานะของกันและกัน เช่น ทราบว่ามีคุปกรณ์ใดออนไลน์เข้ามาใหม่ในกลุ่ม หรือมีคุปกรณ์ใดออกไปจากกลุ่ม รวมถึงทราบการเปลี่ยนแปลงสถานะของคุปกรณ์ที่สนใจ ติดตาม จากข้อมูลดังกล่าวผู้ใช้สามารถกำหนดบทบาทหน้าที่ให้คุปกรณ์ในกลุ่มตามสถานะของคุปกรณ์อื่นๆ ในกลุ่ม เช่น หากเป็นคุปกรณ์ตัวแรกในกลุ่มให้ทำหน้าที่เป็นหัวหน้ากลุ่ม เป็นต้น

Microgear ถูกพัฒนาขึ้นเพื่อให้ทำงานได้กับคุปกรณ์ที่หลากหลาย ในส่วนของซอฟต์แวร์มี Microgear ให้เลือกใช้กับ Programming Language ได้แก่ Node.js Python HTML5 Java Android C# สำหรับคุปกรณ์หารดแวร์ประเภทไมโครคอนโทรลเลอร์ Microgear เปรียบเสมือน Firmware ซึ่งมี Microgear ที่รองรับ Arduino with Ethernet Shield (ใช้ได้กับ Arduino Mega) และ Microgear สำหรับ WiFi ไมโครคอนโทรลเลอร์ ESP8266 Microgear ที่พัฒนาขึ้นทั้งหมดถูกควบรวมไว้ที่ <https://github.com/netpieio> โดยมีสัญญาอนุญาตให้ใช้สิทธิ์แบบเปิดประภาก ISC License ซึ่งอนุญาตให้ทำซ้ำ ดัดแปลง และ/หรือส่งต่อไปบาร์น์ได้ ทั้งในการใช้งานเชิงสาธารณะและเชิงพาณิชย์ รายละเอียดของสัญญาอนุญาตให้ใช้สิทธิ์แบบ ISC License มีดังนี้

ISC License

Copyright (c) 2015, NECTEC <@nectec.or.th>

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

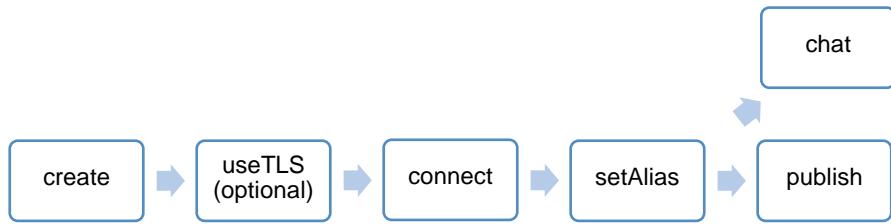
THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

ฟังก์ชันหลักของ Microgear

Microgear แต่ละชนิดอาจมีชื่อและชนิดของฟังก์ชันแตกต่างกันตามลักษณะของการเขียนโปรแกรมในภาษาต่างๆ ในที่นี้ขอยกตัวอย่างฟังก์ชันที่มีเหมือนกันอยู่ในหลาย Microgear โดยขออ้างอิงชื่อฟังก์ชันจาก HTML5 Microgear สำหรับรายละเอียดฟังก์ชันของแต่ละชนิด Microgear สามารถดูได้จากภาคผนวก หรือเอกสาร Readme ใน <https://github.com/netpie.io>

- create สร้าง Microgear เพื่อเริ่มต้นใช้งาน
- connect เชื่อมต่อ Microgear เข้ากับคลาวด์ของ NETPIE
- setAlias กำหนดชื่อเล่นของอุปกรณ์เพื่อใช้ระบุตัวตนของอุปกรณ์ภายใต้ NETPIE
- chat ส่งข้อความแบบเจาะจงผู้รับ
- publish ส่งข้อความแบบไม่เจาะจงผู้รับไปยังหัวข้อสนทนาที่กำหนด
- subscribe ระบุความสนใจในหัวข้อสนทนา บอกรับข้อความที่เกิดขึ้นบนหัวข้อนั้นๆ
- unsubscribe ยกเลิกการบอกรับข้อความในหัวข้อสนทนาที่เคย subscribe ไว้
- resetToken ยกเลิกใบอนุญาต (Token) และลบใบอนุญาตออกจาก cache บนอุปกรณ์
- useTLS ระบุว่าต้องการสร้างการเชื่อมต่อแบบเข้ารหัสระหว่าง Microgear กับคลาวด์ของ NETPIE
- on ตอบสนองต่อเหตุการณ์ที่สนใจผ่านการเรียก Callback Function

คำอธิบายในการเรียกฟังก์ชันพื้นฐานเพื่อเริ่มส่งข้อมูลเป็นไปตามแผนภาพในรูปที่ 1.2



รูปที่ 1.2 ลำดับการเรียกฟังก์ชันพื้นฐานของ Microgear

Events ของ Microgear

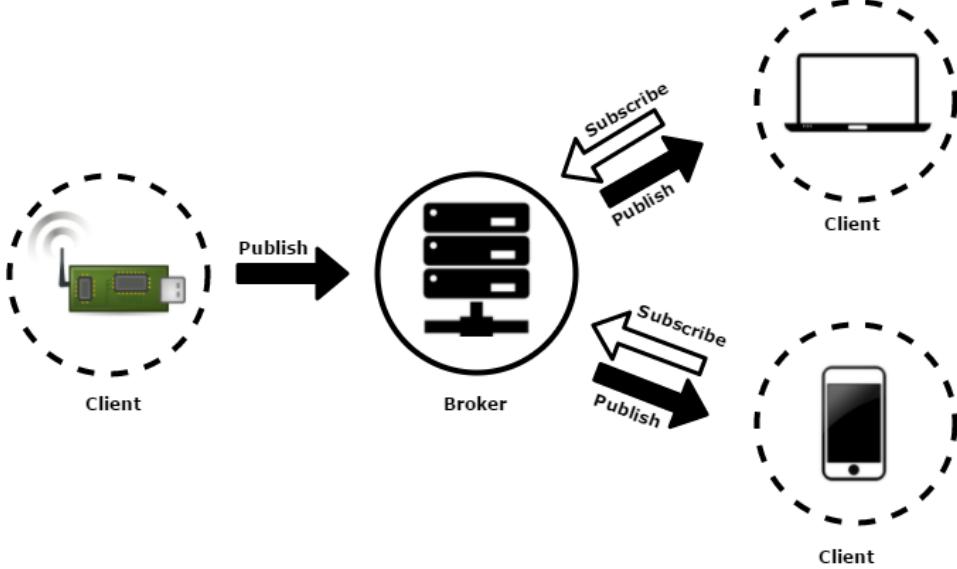
การทำงานของ Microgear เป็นแบบ Event-driven จึงต้องตอบสนองต่อเหตุการณ์ต่างๆ ด้วยการเรียก Callback Function ซึ่งชนิดของเหตุการณ์ที่สามารถเกิดขึ้น มีดังนี้

- connected เกิดขึ้นเมื่อ Microgear เชื่อมต่อกับ NETPIE สำเร็จ
- closed เกิดขึ้นเมื่อ Microgear ปิดการเชื่อมต่อกับ NETPIE
- error เกิดขึ้นเมื่อมีความผิดพลาดเกิดขึ้นกับ Microgear
- message เกิดขึ้นเมื่อมีข้อความเข้ามาที่อุปกรณ์
- present เกิดขึ้นเมื่อมีอุปกรณ์ใน AppID เดียวกันเชื่อมต่อเข้ามานบน NETPIE
- absent เกิดขึ้นเมื่อมีอุปกรณ์ใน AppID เดียวกันหายไปจากการเชื่อมต่อกับ NETPIE

1.3. MQTT (MQ TELEMETRY TRANSPORT)

MQTT เป็นโปรโตคอลสื่อสารชั้นแอปพลิเคชันที่รันบน TCP/IP ถูกพัฒนาขึ้นในปี 1999 โดย IBM และ Eurotech สำหรับการมอนิเตอร์สถานะท่อส่งน้ำมันส่วนที่วางแผนและตรวจสอบท่าเรือ ด้วยการออกแบบให้เป็นการรับส่งข้อมูลที่มีน้ำหนักเบามากและเป็นโปรโตคอลเปิด ทำให้ในปัจจุบัน MQTT ถูกนำมาใช้แพร่หลายในการสื่อสารแบบ M2M หรือ IoT เพราะเหมาะสมกับอุปกรณ์ปลายทางที่มีขนาดเล็ก/พลังงานจำกัด หรือในการสื่อสารระยะไกลที่ต้องการการใช้งานแบบดิจิทัลย่างมีประสิทธิภาพ

โมเดลการสื่อสารของproto콜แสดงดังรูปที่ 1.3 ประกอบด้วย 2 ส่วนคือ คลาวด์ และบราเซอร์



รูปที่ 1.3 โมเดลการสื่อสารของproto콜 MQTT

บราเซอร์ เป็นจุดศูนย์กลางในการรับส่งข้อมูลความเรหะระหว่าง คลาวด์ วิธีการกำหนดเส้นทาง (Routing) กระทำผ่านหัวข้อ (Topic) โดย คลาวด์ Subscribe ในหัวข้อที่ตนต้องการ จากนั้นบราเซอร์จะส่งข้อมูลความทั้งหมดที่ถูก Publish ในหัวข้อนั้นๆ ไปให้ ดังนั้น คลาวด์จะสื่อสารกันได้โดยไม่จำเป็นต้องรู้จักกัน ช่วยลดความเกี่ยวพันระหว่างผู้สร้างข้อมูลและผู้ใช้ข้อมูล ผลให้การขยายตัวของเครือข่ายทำได้ง่าย นอกเหนือจากนั้นที่ที่สำคัญอีกประการของบราเซอร์คือการรักษาความปลอดภัยของ คลาวด์ (Authorization, Authentication) ซึ่งในส่วนนี้สามารถขยายเพิ่มเติม หรือนำไปเชื่อมกับกลไกความปลอดภัยของระบบหลังบ้านที่มีอยู่แล้วได้ ช่วยให้นำบราเซอร์เข้าไปใช้งานเป็นส่วนหนึ่งของระบบอื่นๆ ได้ ส่วน Authorization ของ NETPIE ซึ่งจะได้กล่าวถึงในหัวข้อต่อไป ก็ถือเป็นตัวอย่างหนึ่งของการขยาย เพิ่มเติมการรักษาความปลอดภัยของบราเซอร์ใน MQTT ปัจจุบันมีบราเซอร์ MQTT ที่เปิดให้ดาวน์โหลด ไปใช้หรือติดตั้งอยู่หลายราย ได้แก่ Mosquitto, RabbitMQ, Erlang, VerneMQ ฯลฯ

คลาวด์ จะเป็นได้ทั้ง Publisher หรือ Subscriber หรือ Publisher/Subscriber พื้นที่ กัน และจะเป็นอุปกรณ์ใดๆ ก็ได้ที่สามารถรัน MQTT Client Library บน TCP/IP Stack การที่ MQTT ใช้โมเดล Publish/Subscribe ตรวจสอบในใหญ่จึงไม่ต้องมีบราเซอร์ ทำให้ Library มีขนาดเล็ก ติดตั้งได้ง่าย ใช้งานได้กับอุปกรณ์ที่มีทรัพยากรจำกัด คลาวด์จะสามารถต่ออุปกรณ์ต่อเนื่องที่ต่อไปนี้ได้ ไม่ว่าจะเป็น IoT Device, Camera, Sensor, Actuator ฯลฯ ทำให้สามารถเชื่อมต่ออุปกรณ์ต่อเนื่องได้โดยตรง ไม่ต้องผ่านบราเซอร์ ทำให้ลดต้นทุนและเพิ่มประสิทธิภาพการทำงาน

เมื่อเปรียบเทียบ MQTT กับ HTTP (REST) ที่มีสถาปัตยกรรมแบบ Request/Response จะพบว่า MQTT มีความได้เปรียบที่บราเซอร์สามารถผลัก (Push) ข้อมูลไปยัง คลาวด์ได้ตามเหตุการณ์

(Event-driven) ในขณะที่เมื่อใช้ HTTP ผู้ประกอบการต้องอยู่สู่มุมมอง (Poll) ข้อมูลเป็นระยะๆ และต้องตั้งค่าควบคุมการสุ่ม datum ไว้ก่อนล่วงหน้า โดยแต่ละครั้งต้องมีการสร้างการเข้ามายังตัวชี้ให้ใหม่และอาจจะไม่มีข้อมูลใหม่โดย ให้อัพเดท ดังนั้นหากต้องการให้ระบบทำงานแบบ Real Time หรือใกล้เคียง ย่อมหมายถึงต้องตั้งค่าควบคุมการสุ่ม datum ให้สั้น และเกิดความสิ้นเปลืองของการใช้ช่องสัญญาณที่ไม่จำเป็นที่ตามมา นี่จึงเป็นอีกเหตุผลสำคัญที่ทำให้ MQTT ได้รับความนิยมเนื่อง REST สำหรับการใช้งานแบบ M2M นอกจากนี้จากการมีน้ำหนักเบา

MQTT Topics

MQTT Topic เป็น UTF-8 String ในลักษณะเดียวกับ File Path คือสามารถจัดเป็นลำดับชั้นได้ด้วยการขั้นด้วย "/" ตัวอย่างเช่น `myhome/floor-one/room-c/temperature` คลอน์สามารถเลือก Publish หรือ Subscribe เฉพาะ Topic หรือ Subscribe หลาย Topic พร้อมๆ กันโดยใช้ Single-Level Wildcard (+) เช่น `myhome/floor-one/+/temperature` หมายถึงการขอเขียนหรือรับข้อมูลความ temperature จากทุกๆ ห้องของ `myhome/floor-one` หรือ Multi-Level Wildcard (#) เช่น `myhome/floor-one/#` หมายถึงการขอเขียนหรือรับข้อมูลทั้งหมดที่มี Topic ขึ้นต้นด้วย `myhome/floor-one` เป็นต้น

เราสามารถกำหนด Topic อย่างไรก็ได้ โดยมีข้อยกเว้นการขึ้นต้น Topic ด้วยเครื่องหมาย "\$" ซึ่งจะจำกัดไว้สำหรับการเก็บสถิติภายในของตัวบอกรეโกร์เท่านั้น ดังนั้นคลอน์จะไม่สามารถ Publish หรือ Subscribe ไปยัง Topic เหล่านี้ได้ โดยทั่วไป Topic เหล่านี้จะขึ้นต้นด้วย \$SYS

MQTT vs Message Queues

เรามักพบการสับสนระหว่าง MQTT กับ Message Queues โดยคนจำนวนไม่น้อยเชื่อว่า MQ ใน MQTT มาจากคำว่า Message Queue ในความเป็นจริงแล้ว MQ ในที่นี้มาจากชื่อรุ่นผลิตภัณฑ์ที่รองรับไฟร์วอลล์ MQTT ของ IBM ที่เรียกว่า MQ Series และ MQTT ไม่ได้ทำงานในลักษณะ Message Queue กล่าวคือ ข้อมูลใน MQTT จะถูกส่งให้กับคลอน์ทั้งหมดที่ Subscribe ใน Topic ในขณะที่ข้อมูลใน Message Queue สามารถถูกดึงออกได้ใช้งานโดยผู้ใช้เพียงรายเดียวเท่านั้น

MQTT Connections

แพ็กเกตควบคุม (Control Packets) ทั้งหมดที่ใช้ในการสื่อสารระหว่างบอกรีโกร์กับคลอน์ใน MQTT มีทั้งสิ้น 14 ชนิด แสดงไว้ดังตารางที่ 1.1

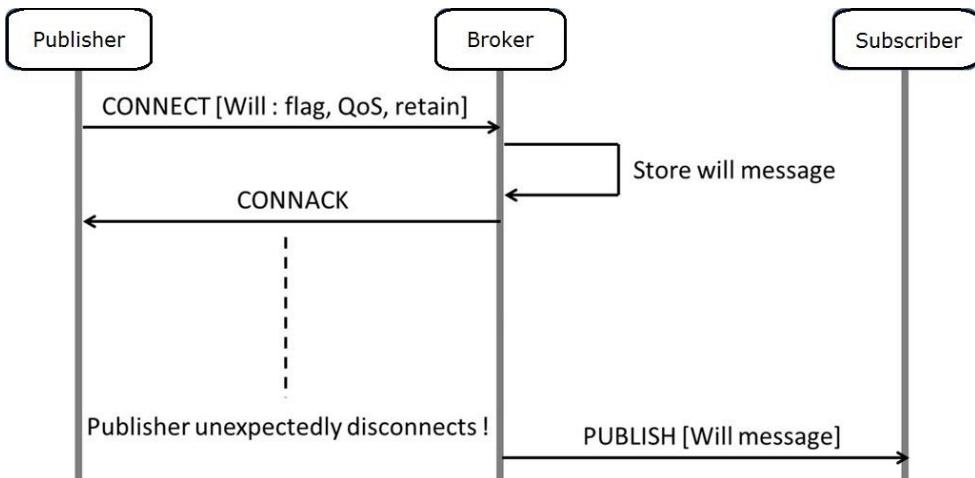
ตารางที่ 1.1 ชนิดของแพ็คเกตควบคุมในโพรโทคอล MQTT

แพ็คเกตควบคุม	ผู้ส่ง		คำอธิบาย
	บรากเกอร์	คลาเอนต์	
CONNECT	X		ขอเชื่อมต่อ
CONNACK	X		รับทราบการขอเชื่อมต่อ
PUBLISH	X	X	ข้อความที่จะขอ Publish
PUBACK	X	X	แจ้งว่าได้ Publish แล้ว (QoS Level 1)
PUBREC	X	X	แจ้งว่าได้ Publish แล้ว (QoS Level 2)
PUBREL	X	X	รับทราบว่าข้อความถูก Publish แล้ว และแจ้งให้อีกผู้รับลบค่าสถานะได้ (QoS Level 2)
PUBCOM	X	X	แจ้งว่า Publish เสร็จสิ้นและสถานะถูก [*] ลบ (QoS Level 2)
SUBSCRIBE		X	ขอ Subscribe
SUBACK	X		รับทราบการขอ Subscribe
UNSUBSCRIBE		X	ขอยกเลิก Subscribe
UNSUBACK	X		รับทราบการยกเลิก Subscribe
PINGREQ		X	PING Request
PINGRESP	X		PING Response
DISCONNECT		X	ขอยกเลิกการเชื่อมต่อ

การเชื่อมต่อ MQTT จะเริ่มต้นจากฝั่งไคลเอนต์ส่งแพ็คเกตควบคุม CONNECT ไปยังบอร์ดเครื่อง บอร์ดจะตอบรับด้วยแพ็คเกตควบคุม CONNACK วิธีนี้ช่วยแก้ปัญหาไคลเอนต์ที่ติดตั้งอยู่หลังเราท์เตอร์ หรือไม่มีเดลai โพรเซสการทำงาน เพราะบอร์ดมีเดลai โพรเซสการทำงานและจะรักษาการเชื่อมต่อสองทางไว้ตลอด หลังจากได้รับแพ็คเกตควบคุม CONNECT หากบอร์ดพบว่าแพ็คเกต CONNECT ที่ได้รับไม่ถูกต้อง หรือไคลเอนต์ใช้เวลานานเกินไปนับตั้งแต่เปิดซ็อกเก็ตจนกระทั่งเริ่มส่งแพ็คเกต บอร์ดจะปิดการเชื่อมต่อ เพื่อป้องกันไคลเอนต์ไม่ประสงค์ที่ต้องการต่อการทำงานของบอร์ด

ไคลเอนต์จะเป็นผู้ระบุค่าการเชื่อมต่อในแพ็คเกตควบคุม CONNECT ค่าเหล่านี้ได้แก่

- Client ID เพื่อให้บอร์ดใช้ระบุตัวตนของไคลเอนต์และเก็บค่าสถานะ เช่น ไว้ให้ ซึ่งได้แก่ Subscriptions และข้อความทั้งหมดที่ไคลเอนต์ยังไม่ได้รับ (ส่วนนี้เกี่ยวข้องกับการตั้งค่าระดับ QoS ซึ่งจะกล่าวถึงต่อไป) ดังนั้น Client ID จึงจำเป็นที่จะต้องไม่ซ้ำกัน แต่หากไม่ต้องการให้บอร์ดเก็บค่าสถานะ ไคลเอนต์สามารถได้รับส่วนนี้ว่างไว้ได้
- Clean Session มีค่า True หรือ False เพื่อระบุว่าไคลเอนต์ต้องการให้บอร์ดรักษาค่าสถานะของเขานั่นไว้ให้หรือไม่ หากต้องการ ให้ระบุค่าเป็น False หากไม่ ให้ระบุค่าเป็น True ดังนั้นหากไคลเอนต์ไม่ระบุค่า Client ID ในแพ็คเกต CONNECT จะต้องระบุ Clean Session เป็น True ด้วย มิฉะนั้นบอร์ดจะไม่รับการเชื่อมต่อ
- Username และ Password เพื่อให้บอร์ดใช้ในการ Authentication และ Authorization ซึ่งตามมาตรฐานปัจจุบัน (MQTT 3.1.1) proto ของ MQTT เองไม่มีการเข้ารหัสหรือency ในส่วนนี้ กล่าวคือ Username/Password จะถูกส่งเป็น Plaintext ดังนั้นจึงมีข้อควรระวังในการใช้ MQTT หากไม่มีชั้นความปลอดภัยเพิ่มเติม เช่น TLS ในชั้นทรายสปอร์ต
- Last Will Topic, Last Will QoS และ Last Will Message มีไว้ให้บอร์ด Publish ข้อความสั่งเสียสุดท้าย (Last Will Message) ไปยัง Topic ที่ระบุ (Last Will Topic) เพื่อให้ไคลเอนต์อื่นรับรู้ในกรณีการเชื่อมต่อของไคลเอนต์รายนี้ขาดลงแบบไม่เจตนา
- Keep Alive เป็นค่าตัวเลขควบเวลาที่ไคลเอนต์ตกลงกับบอร์ดว่าจะส่งแพ็คเกตควบคุม PINGREQ มาเป็นระยะเวลา ซึ่งบอร์ดจะตอบด้วยแพ็คเกต PINGRESP เพื่อให้ทั้งสองฝ่ายรับรู้ว่าการเชื่อมต่ออยู่คงอยู่



รูปที่ 1.4 ผังการรับส่งข้อความสั่งเสียสุดท้าย (Last Will Message)¹

โคลอนต์ Publish ข้อความ โดยบรรจุในส่วน Payload ของแพ็คเกตควบคุม PUBLISH ซึ่งจะต้องระบุ Packet ID, ชื่อ Topic, ระดับของ QoS, Duplicate Flag และ Retain Flag ไปรอกเอกสารจะตอบกลับด้วยแพ็คเกต PUBACK หรือ PUBREC ขึ้นอยู่กับระดับ QoS ที่ระบุในแพ็คเกต PUBLISH ในทางกลับกัน เมื่อต้องการรับข้อมูล โคลอนต์ส่งแพ็คเกตควบคุม SUBSCRIBE ไปยังบอร์ดโดยระบุรายชื่อ Topic ที่ต้องการ ซึ่งอาจมีมากกว่าหนึ่ง และสามารถเลือกตั้งค่าระดับ QoS ที่แตกต่างกันสำหรับแต่ละ Topic และบอร์ดจะตอบด้วยแพ็คเกต SUBACK โดยยืนยันค่าระดับ QoS ของแต่ละ Topic ที่โคลอนต์ขอ Subscribe กลับมาอีกครั้ง หาก Topic ใดที่ไม่อนุญาตหรือไม่ปรากฏในบอร์ด บอร์ดจะยืนยันการยกเลิกด้วยแพ็คเกต UNSUBACK

เมื่อโคลอนต์ต้องการยกเลิกการรับข้อมูล ทำได้โดยการส่งแพ็คเกตควบคุม UNSUBSCRIBE ไปยังบอร์ด โดยระบุรายชื่อ Topic ที่ต้องการยกเลิกในคราวเดียวกันได้มากกว่าหนึ่ง บอร์ดจะยืนยันการยกเลิกด้วยแพ็คเกต UNSUBACK

เมื่อโคลอนต์ต้องการเลิกการเชื่อมต่อ ทำได้โดยการส่งแพ็คเกตควบคุม DISCONNECT ไปยังบอร์ด หากโคลอนต์ CONNECT โดยตั้งค่า Clean Session เป็น True บอร์ดจะยกเลิก Subscription ทั้งหมดของโคลอนต์ให้เองโดยอัตโนมัติ ในทางกลับกันหาก Clean Session เป็น False บอร์ดจะยังคงเก็บค่าต่างๆ ของเซสชันไว้ เมื่อโคลอนต์เชื่อมต่อเข้ามาใหม่ด้วย Client ID เดิม จึงไม่จำเป็นต้องเริ่ม Subscribe ใหม่อีกครั้ง

¹<http://www.embedded101.com>

MQTT Quality of Service (QoS)

โคลเอนต์จะเป็นผู้กำหนดระดับของบริการส่งและรับข้อความหรือ QoS ที่ต้องการในแต่ละ Topic ในแพ็กเกต PUBLISH หรือ SUBSCRIBE และใบประกอบสนองด้วย QoS ระดับเดียวกันสำหรับ Topic นั้นๆ

QoS ใน MQTT แบ่งได้เป็น 3 ระดับ คือ

1. อ่าย่างมากหนึ่งครั้ง (At Most Once) แทนด้วยโค้ด 0

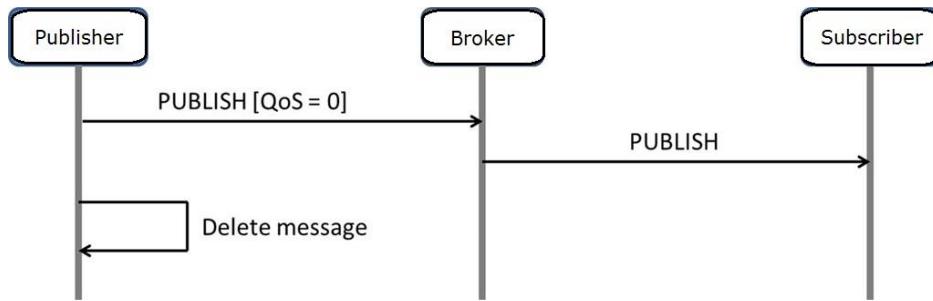
QoS 0 เป็นระดับบริการที่ต่ำที่สุด กล่าวคือไม่ว่าจะรับประกันว่าข้อความจะถูกส่งถึงผู้รับได้ เลยหรือไม่ หากโคลเอนต์ Publish ข้อความด้วย QoS 0 ใบประกอบจะไม่มีการตอบรับใดๆ ว่าได้ Publish ต่อไปให้ผู้รับรายอื่นหรือไม่ หากไม่มีผู้รับข้อความ ใบประกอบอาจเก็บข้อความไว้หรือลบทิ้งก็ได้ ขึ้นอยู่กับนโยบายของผู้ให้บริการเซิร์ฟเวอร์ ในทางกลับกันหากโคลเอนต์ที่เป็นผู้รับ Subscribe ไว้ด้วย QoS 0 เมื่อได้รับข้อความจากใบประกอบ ก็ไม่ต้องส่งข้อความตอบรับใดๆ กลับ ทำให้การส่งข้อความแบบนี้รวดเร็วที่สุด เพราะไม่มีโคลเอนต์เข้าในการตอบรับ ขณะเดียวกันหากข้อความถูกส่งไม่ถึง ก็ไม่มีทางทราบได้เช่นกัน

2. อ่ายางน้อยหนึ่งครั้ง (At Least Once) แทนด้วยโค้ด 1

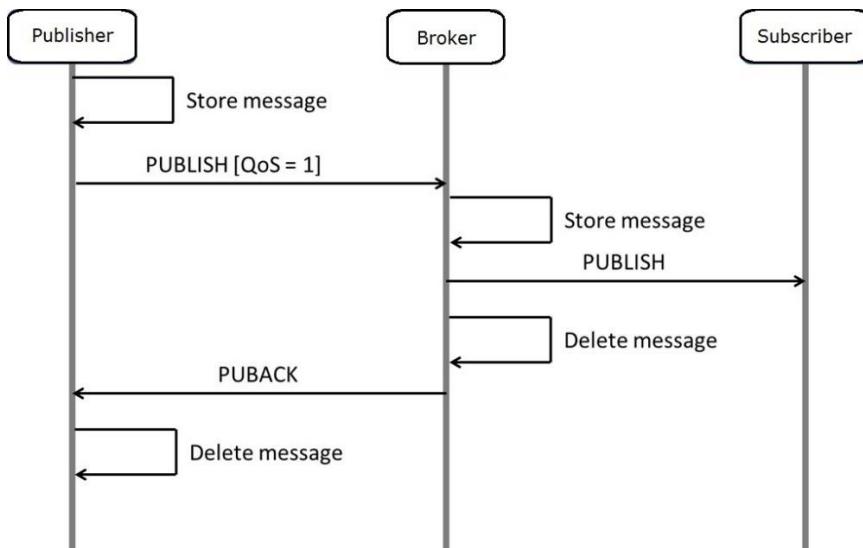
QoS 1 รับประกันว่าข้อความจะถูกส่งถึงผู้รับอย่างน้อยหนึ่งครั้ง การส่งลักษณะนี้ ผู้ส่งจะเก็บข้อความเอาไว้ จนกว่าจะได้รับแพ็กเกต PUBACK จากผู้รับ ในกรณีโคลเอนต์ที่ Publish ผู้รับข้อความซึ่งเป็นใบประกอบจะต้อง Publish ต่อไปยังโคลเอนต์ที่ Subscribe ไว้อย่างน้อยหนึ่งครั้ง จึงจะสามารถส่งแพ็กเกตตอบรับไปกลับยังผู้ส่ง ในกรณี Subscribe ผู้ส่งซึ่งก็คือใบประกอบจะต้องเก็บข้อความไว้จนกว่าโคลเอนต์ที่ตนส่งข้อความไปให้จะยืนยันตอบรับ ดังนั้นแพ็กเกต PUBACK จึงต้องมีหมายเลขไอเดียวกับแพ็กเกต PUBLISH เพื่อให้ผู้ส่งทราบว่าข้อความได้ถูกส่งถึงแล้วและสามารถลบออกได้

3. หนึ่งครั้งเท่านั้น (Exactly Once) แทนด้วยโค้ด 2

QoS 2 รับประกันว่าแต่ละข้อความจะถูกส่งถึงผู้รับเพียงหนึ่งครั้งเท่านั้น เป็นบริการที่ปลอดภัยที่สุด และซ้ำที่สุดของโพรโทคอล MQTT เนื่องจากผู้รับและผู้ส่งต้องส่งแพ็กเกตควบคุมไปกลับถึงสองรอบ เริ่มต้นด้วยผู้ส่งส่งข้อความไปในแพ็กเกต PUBLISH เมื่อผู้รับได้รับข้อความจะเก็บแพ็กเกตไว้และยืนยันกลับไปยังผู้ส่งด้วยแพ็กเกต PUBREC ผู้ส่งจึงสามารถลบข้อความนั้นออกจากหน่วยเก็บข้อมูลของตนได้ และส่งแพ็กเกต PUBREL ไปยังผู้รับเพื่อให้ผู้รับสามารถลบสถานะการส่งข้อความนี้ออกได้ หากผู้รับเป็นโคลเอนต์ปลายทางที่ Subscribe ข้อความเอาไว้ ผู้รับจะส่งแพ็กเกต PUBCOM เพื่อยืนยันว่าข้อความถูกส่งถึงแล้ว เรียบร้อยหนึ่งครั้ง หากผู้รับเป็นใบประกอบ ทันทีที่ได้ Publish ข้อความต่อไปยังโคลเอนต์ปลายทางหนึ่งครั้ง จึงจะลบข้อความนั้นออก และปิดเซสชันด้วยการส่งแพ็กเกต PUBCOM กลับไปยังผู้ส่ง



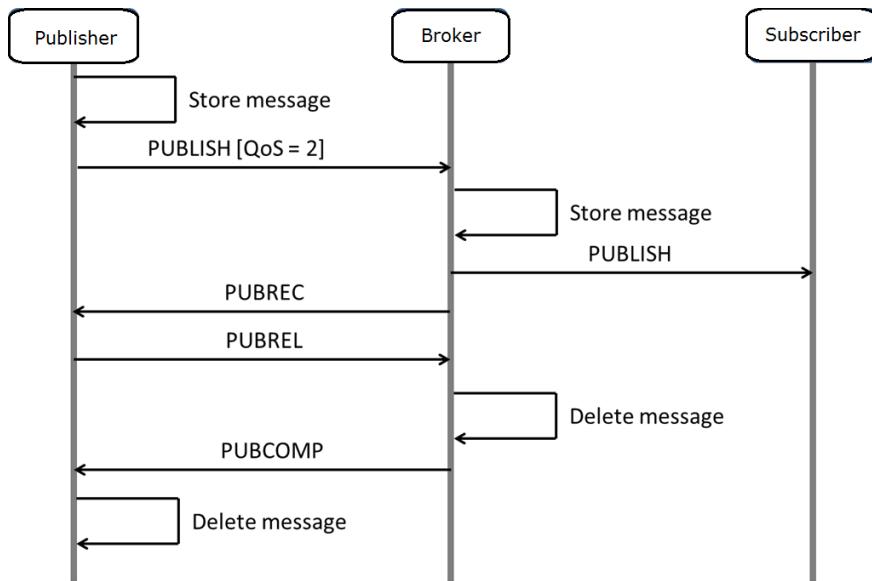
รูปที่ 1.5 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS 0²



รูปที่ 1.6 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS 1³

²<http://www.embedded101.com>

³<http://www.embedded101.com>



รูปที่ 1.7 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS⁴

Retained Messages

เมื่อไคลเอนต์ Publish ข้อความแบบ Retained Message โดยการตั้ง Retain Flag เป็น True จะทำให้ broker เก็บข้อความนั้นไว้ใน Topic ที่ระบุอย่างถาวร จนกว่าจะมี Retained Message อื่นที่ถูก Publish ภายหลังมาเก็บแทนที่ โดย broker จะเก็บ Retained Message ไว้ให้เพียง Topic ละหนึ่งข้อความ ดังนั้นทุกครั้งที่มีไคลเอนต์ Subscribe เข้ามาใหม่ ก็จะได้รับ Retained Message ทันที ไม่ต้องรอ จนกว่าจะมี Publication ใหม่เกิดขึ้น จึงมองได้ว่า Retained Message คือข้อมูลล่าสุดที่ Subscriber ทุกรายควรต้องทราบ ดังนั้น Retained Message จึงเป็นประโยชน์อย่างยิ่งกับแอปพลิเคชันที่เกี่ยวข้องกับการอัพเดทสถานะ หากไคลเอนต์ต้องการลบ Retained Message ใน Topic ใดๆ ก็สามารถทำได้ไม่ยาก ด้วยการส่งแพ็กเกต PUBLISH ที่ไม่มี Payload และตั้ง Retain Flag เป็น True ไปยัง Topic นั้นๆ หรือหากมีข้อมูลจะอัพเดท ก็ไม่มีความจำเป็นต้องลบ Retained Message เก่าออกก่อน แต่สามารถส่ง Retained Message เข้าไปเพิ่มนับได้เลย

แบบฝึกหัด: Local MQTT Chat บน Raspberry Pi

1. ติดตั้ง broker Mosquitto ลงบน Raspberry Pi โดยพิมพ์ชุดคำสั่งข้างล่าง และแทนที่ xxxx ด้วยชื่อรุ่นระบบปฏิบัติการ เช่น wheezy หรือ jessie

⁴ <http://www.embedded101.com>

```
curl -O http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key  
sudo apt-key add mosquitto-repo.gpg.key  
rm mosquitto-repo.gpg.key  
cd /etc/apt/sources.list.d/  
sudo curl -O http://repo.mosquitto.org/debian/mosquitto-xxxx.list  
sudo apt-get update
```

2. ติดตั้งโคลอนด์

```
sudo apt-get install mosquitto mosquitto-clients
```

3. ทดลองแชทโดยเปิดหน้าต่าง สร้างหัวข้อ “chat/test1” ลงบนบอร์ดเครื่องและ “Subscribe” โดยพิมพ์

```
mosquitto_sub -h localhost -t "chat/test1"
```

4. เปิดหน้าต่างใหม่โดยยังไม่เปิดหน้าต่างแรก พิมพ์คำสั่งข้างล่างเพื่อ Publish ข้อความ “hello” ไปยังหัวข้อ “chat/test1”

```
mosquitto_pub -h localhost -t "chat/test1" -m "hello"
```

ในหน้าต่างแรกที่ได้ Subscribe เอกำไร จะปรากฏข้อความ hello

5. ลองเกตผลของการตั้งระดับบริการรับส่งข้อมูล (QoS) ที่แตกต่างกัน พิมพ์คำสั่งข้างล่างในหน้าต่างแรก เพื่อ Subscribe ในหัวข้อ “chat/test2” ด้วย QoS 1

```
mosquitto_sub -h localhost -v -t "chat/#" -c -q 1 -i "Client ID"
```

6. ในหน้าต่างที่สอง พิมพ์คำสั่งข้างล่างเพื่อ Publish ข้อความ “You might get this message.” และ “You will get this message.” ด้วย QoS 0 และ QoS 1 ตามลำดับ

```
mosquitto_pub -h localhost -t "chat/test2" -m "You might get this message."  
mosquitto_pub -h localhost -t "chat/test2" -m "You will get this message."  
-q 1
```

7. ลองเกตผลลัพธ์ในหน้าต่างแรก
8. หยุดการทำงานของหน้าต่างแรกช้าๆโดยกด ctrl-c
9. ทำข้อ 6
10. ทำข้อ 5 และลองเกตผลลัพธ์

1.4. AUTHORIZATION และ AUTHENTICATION ใน NETPIE

เนื่องจาก NETPIE มีศูนย์กลางการสื่อสารเป็น Distributed MQTT Broker ความปลอดภัยจึงขึ้นตระกับการเข้าถึงบอร์ดของอุปกรณ์ ดังที่ได้กล่าวไปแล้วในหัวข้อที่ 1.3 ว่ามาตราฐาน MQTT ระบุให้ใช้เพียง Username และ Password ใน การพิสูจน์ตัวตน (Authentication) ของไคลเอนท์ โดยสามารถกำหนดสิทธิ์ การเข้าถึงบอร์ดของไคลเอนต์แต่ละรายด้วยการผูกเป็นรายการ (List) เข้ากับ Username บนบอร์ด ในขณะที่ NETPIE นั้น มีข้อมูลประจำตัว (Credentials) ของอุปกรณ์ที่จะมาขอเชื่อมต่อถ่องหนึ่ง คือ Key และ Token ซึ่งจะช่วยเพิ่มความยืดหยุ่นในการบริหารจัดการการเข้าถึงบอร์ดของอุปกรณ์ให้มีความละเอียด ยืดหยุ่นและมีประสิทธิภาพมากขึ้น ดังนั้น

อุปกรณ์จะได้มาซึ่งข้อมูลประจำตัวจากการขออนุญาตสิทธิ์ (Authorization) ซึ่งประกอบด้วย ผู้เกี่ยวข้อง 3 ฝ่ายคือ

1) อุปกรณ์ (Device) ที่จะมาเชื่อมต่อสื่อสารกันให้เกิดแอปพลิเคชัน ในที่นี้รวมถึงห้องอุปกรณ์ กายภาพและอุปกรณ์เสมือน (Object)

2) ผู้ใช้ (User) ซึ่งเป็นเจ้าของแอปพลิเคชัน และเป็นผู้กำหนดว่าอุปกรณ์ใดมีสิทธิ์หรือไม่ มาก่อนอย่างใดในแอปพลิเคชันของตน

3) NETPIE ทำหน้าที่เป็นตัวกลางในการขออนุญาตสิทธิ์ระหว่างผู้ใช้และอุปกรณ์

กระบวนการทั้งหมดมีขั้นตอนดัง

1) ผู้ใช้งานจะเปลี่ยนกับ NETPIE Web Portal และสร้างแอปพลิเคชัน

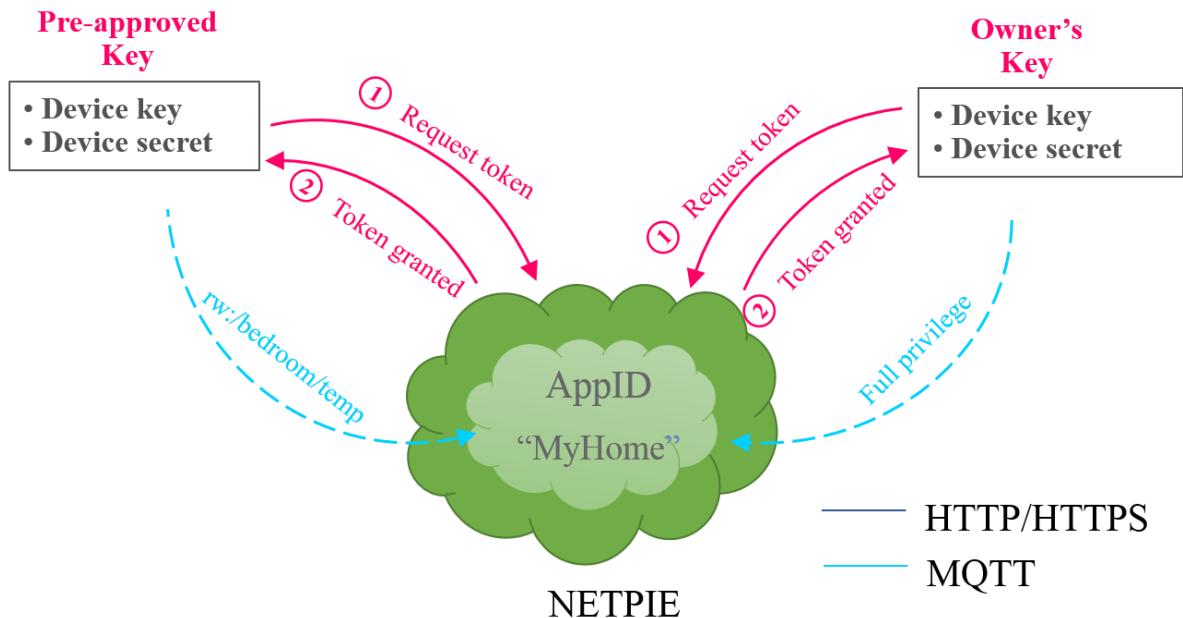
2) ผู้ใช้เพิ่มอุปกรณ์ภายใต้แอปพลิเคชันนั้นๆ และได้รับ Key และ Key Secret เพื่อนำมาใส่ลงในโค้ดของ Microgear ในอุปกรณ์

3) เมื่ออุปกรณ์ถูกเปิดใช้งาน Microgear จะเชื่อมต่อเข้ามายัง NETPIE โดยอัตโนมัติ และพิสูจน์ตัวตนด้วย Key และ Key Secret ที่ได้รับในการขอ Token เพื่อเชื่อมต่อและสื่อสารผ่าน NETPIE

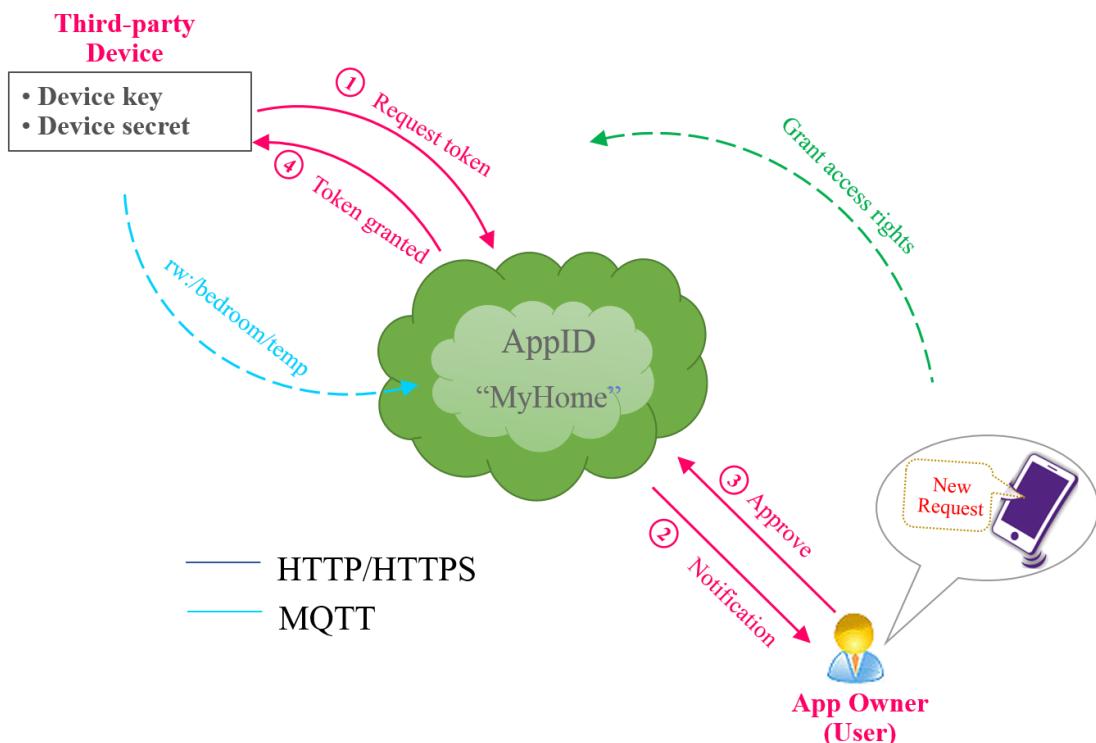
4) ในขั้นตอนนี้ NETPIE มีทางเลือกสองทางเพื่อความสะดวกของผู้ใช้

ทางเลือกที่ 1: Pre-approved Key กล่าวคือผู้ใช้สามารถกำหนดไว้ก่อนล่วงหน้าว่า Key ใดบ้างจะได้รับการอนุญาตสิทธิ์แบบอัตโนมัติ ดังนั้นทันทีที่อุปกรณ์ต่อเข้ามา จะได้รับ Token ทันที นอกจากนี้ผู้ใช้ยังสามารถกำหนดระดับของสิทธิ์ไว้ก่อนได้ เช่น กัน เข่น ให้สิทธิ์อุปกรณ์ของผู้ใช้เองแบบเต็ม ให้สิทธิ์อุปกรณ์ของคนอื่นบางส่วน เป็นต้น

ทางเลือกที่ 2: Third-Party Key ซึ่งหมายความว่าให้สิทธิ์อุปกรณ์ของบุคคลอื่นเข้ามาร่วมใช้แอปพลิเคชัน เมื่ออุปกรณ์ร้องขอ Token เข้ามา NETPIE AUTH Server จะติดต่อไปยังผู้ใช้ ให้เข้ามาอนุญาตสิทธิ์ โดยสามารถกำหนดระดับของสิทธิ์ และ NETPIE จะทำการออก Token ให้กับอุปกรณ์และบันทึกระดับของสิทธิ์ที่ได้รับอนุญาตเอาไว้



รูปที่ 1.8 NETPIE Authorization ในกรณี Pre-approved Key



รูปที่ 1.9 NETPIE Authorization ในกรณี Third-party Key

เมื่อคุณกรณ์ได้รับข้อมูลส่วนตัวครบแล้ว จะนำทั้งหมด (Key, Key Secret, Token, Token Secret) ไปสร้าง Client ID, Username และ Password ในการพิสูจน์ตัวตนเข้าใช้ NETPIE ตามprotoคอล MQTT โดยวิธีการสร้างจะทำให้ Username กับ Password มีการเปลี่ยนแปลงทุกครั้งในแต่ละเซสชัน และมีการแข็งแกร่งมากขึ้น ซึ่งป้องกันการดักฟังและการทำข้า



NETPIE รองรับการเชื่อมต่อ MQTT แบบ Persistent Connection เท่านั้น ดังนั้นจะมีการตั้งค่า Client ID ไว้เสมอ หากมีการเพิกถอนสิทธิ์ของอุปกรณ์ Client ID จะถูกลบ อุปกรณ์จะต้องต่อไปยัง NETPIE เพื่อขอ Token ใหม่ โดยไม่ต้องไปเริ่มต้นกระบวนการขอ Key ใหม่ตั้งแต่ต้น ในเดลของ NETPIE นี้เป็นการนำแนวคิดที่ใช้ในสังคมมนุษย์มาปรับใช้ในสังคมของสิ่งของ เช่น เมื่อคนทำผิดกฎหมาย ถูกยึดใบขับขี่ ทำให้ไม่มีสิทธิ์ในการขับขี่ยานพาหนะ แต่สามารถไปขอทำใบขับขี่ใหม่ได้ภายในหลัง เป็นต้น

ชนิดของ Key

NETPIE รองรับคุณกรณ์ที่หลากหลาย นอกจากคุณกรณ์ภาษาไทย เช่น บอร์ดไมโครคอนโทรลเลอร์ ต่างๆ แล้ว ยังรองรับคุณกรณ์สมือน้ำ เช่น เอปพลิเคชันต่างๆ ซึ่งรวมไปถึงแอปพลิเคชันที่เขียนด้วย HTML5 ที่สามารถรันได้บนเบราว์เซอร์ ในกรณีเช่นนี้ การเชื่อมต่อไม่สามารถเป็นแบบ Persistent ได้ เนื่องจากต้องมีการปิดเบราว์เซอร์ตลอดเวลา ดังนั้นเพื่อให้การจัดการ Key และ Token มีประสิทธิภาพ หลีกเลี่ยงการสร้าง Token ทึ่งไว้โดยไม่ได้ใช้งาน NETPIE จึงออกแบบให้มี Key อยู่ 2 ชนิด คือ

1. Device Key

เมื่อคุณกรณ์ได้รับการติดตั้ง Device Key และร้องขอ Token มาจาก NETPIE หากได้รับการอนุญาต สิทธิ์ อุปกรณ์จะได้รับ Token ที่ใช้ได้ตลอดไป ไม่มีการหมดอายุ ดังนั้นตราบใดที่ไม่มีการเพิกถอนสิทธิ์ อุปกรณ์สามารถใช้ Token เดิมไปได้ตลอดโดยไม่ต้องเข้าสู่กระบวนการ Authorization ใหม่ Device Key จึงเหมาะสมกับคุณกรณ์ภาษาไทยที่สามารถรักษาช่องการเชื่อมต่อ กับ NETPIE ได้โดยตลอด

2. Session Key

อุปกรณ์ที่ได้รับการติดตั้ง Session Key จะได้รับ Token ที่ใช้ได้เพียงครั้งเดียว (One-time Token) หากคุณกรณ์ยกเลิกการเชื่อมต่อไป และต่อกลับมาใหม่ จะต้องมีการนำ Session Key ไปขอ Token ใหม่ ทุกครั้ง Session Key จึงเหมาะสมกับการใช้งานแอปพลิเคชันที่รับบนเบราว์เซอร์ที่ต้องมีการปิดเบื้องตัวอย่างไร เพราะหากใช้ Device Key ทุกครั้งที่เบราว์เซอร์ถูกเปิดขึ้นจะมีการร้องขอ Token ใหม่ และเมื่อเบราว์เซอร์ถูกปิดไป Token ที่ได้จะถูกยกเลิกโดยไม่ได้ร้องขออีก ผลให้ภายใต้ Key เดียวกัน มี Token ทึ่งที่ใช้

งานและไม่ได้ใช้งานแล้วผูกอยู่เป็นจำนวนมาก สร้างความสับสนซับซ้อนที่ไม่จำเป็นให้กับผู้ใช้ จึงขอเน้นถึงเป็นอย่างยิ่งว่า ไม่ควรใช้ Device Key กับแอปพลิเคชันที่รับบนเบราว์เซอร์ (HTML5) โดยเด็ดขาด

ขอบเขตของสิทธิ์ (Scope) ในการเข้าถึง NETPIE MQTT Brokers

เมื่อผู้ใช้อุปกรณ์ให้กับคุปกรณ์ จะสามารถบุกเข้าไปในสิทธิ์ หรือที่เราระบุไว้ Scope ได้ว่า มากน้อยเพียงใด โดย Scope นี้จะผูกอยู่กับ Token ที่คุปกรณ์จะได้รับ และถูกเก็บไว้ที่ NETPIE รูปแบบของ Scope Identifier เกี่ยวพันโดยตรงกับโครงสร้างการสื่อสารแบบ Publish/Subscribe ไปยัง Topic บน broker MQTT มีได้ 3 ลักษณะคือ

1. rw:/topic เช่น rw:/home/bedroom/temperature หมายถึง มีสิทธิ์อ่าน (Subscribe) และเขียน (Publish) ใน home/bedroom/temperature
2. r:/topic เช่น r:/home/# หมายถึง มีสิทธิ์อ่าน (Subscribe) ทุก Topic ที่ขึ้นต้นด้วย home
3. w:/topic เช่น w:/home/+/temperature หมายถึง มีสิทธิ์เขียน (Publish) ไปยัง Topic ชื่อ temperature ของทุกๆ ห้องใน home

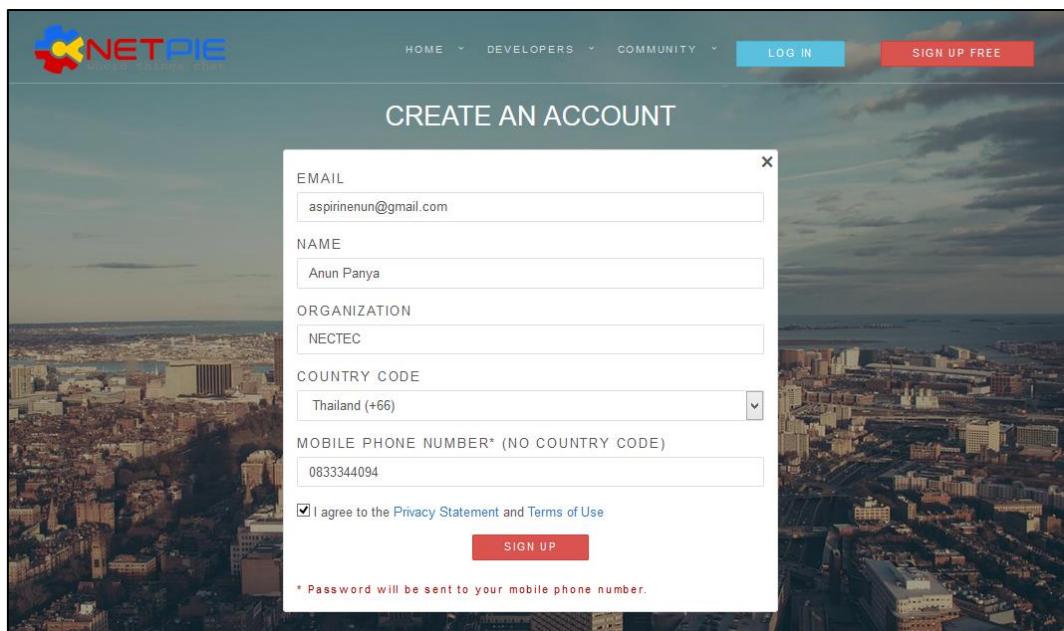
การกำหนด Scope ที่เกี่ยวข้องกับหลาย Topic นอกจากใช้ Wildcard ดังที่ได้แสดงไปแล้ว สามารถทำได้โดยการเรียง Scope Identifier เข้าด้วยกันแล้วคั่นด้วยเครื่องหมาย Bar "|" ตัวอย่างเช่น
rw:/home/bedroom/temperature|r:/home/#|w:/home/+/temperature

2. GETTING STARTED

2.1 การสมัครสมาชิก

ขั้นตอนการสมัครสมาชิกเพื่อใช้งาน NETPIE มีดังนี้

1. ไปที่เว็บไซต์ https://netpie.io/sign_up จะปรากฏหน้าเว็บดังรูป กรอกข้อมูลให้เรียบง่าย จากนั้นคลิกที่ปุ่ม SIGN UP เพื่อยืนยันการลงทะเบียน



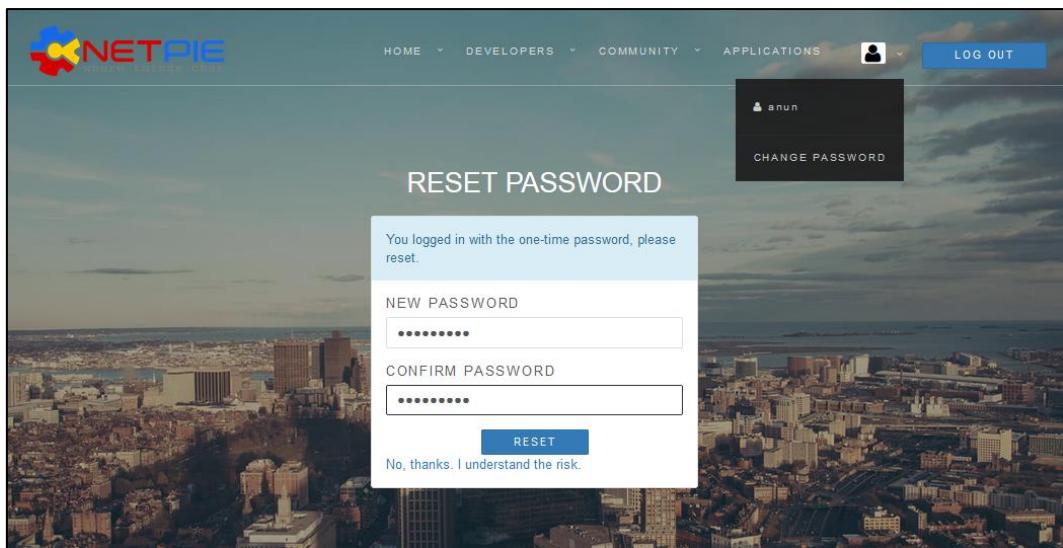
รูปที่ 2.1 หน้าลงทะเบียนผู้ใช้งาน NETPIE

2. รอรับ SMS จากทาง NETPIE ซึ่งส่งไปยังหมายเลขโทรศัพท์เคลื่อนที่ที่ลงทะเบียนไว้

ตัวอย่าง SMS:

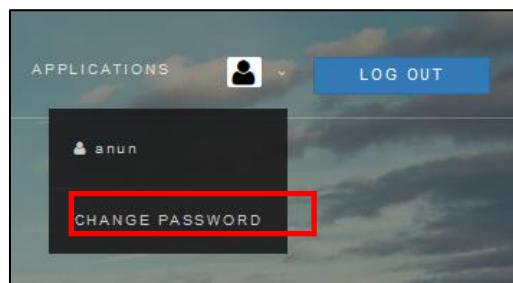
Your one-time password for NETPIE is 255906201550

3. คลิกที่เมนู LOG IN เพื่อเข้าสู่ระบบนำอีเมลที่ลงทะเบียนไว้ใส่ในช่อง USERNAME OR EMAIL ADDRESS และนำรหัสผ่านที่ได้รับจาก SMS ใส่ในช่อง PASSWORD แล้วคลิกปุ่ม LOG IN
4. ตั้งรหัสผ่านใหม่โดยใส่รหัสผ่านใหม่ในช่อง NEW PASSWORD และ CONFIRM PASSWORD



รูปที่ 2.2 หน้าตั้งรหัสผ่านใหม่สำหรับผู้ใช้ NETPIE

- สามารถเปลี่ยนรหัสผ่านได้ที่เมนู CHANGE PASSWORD



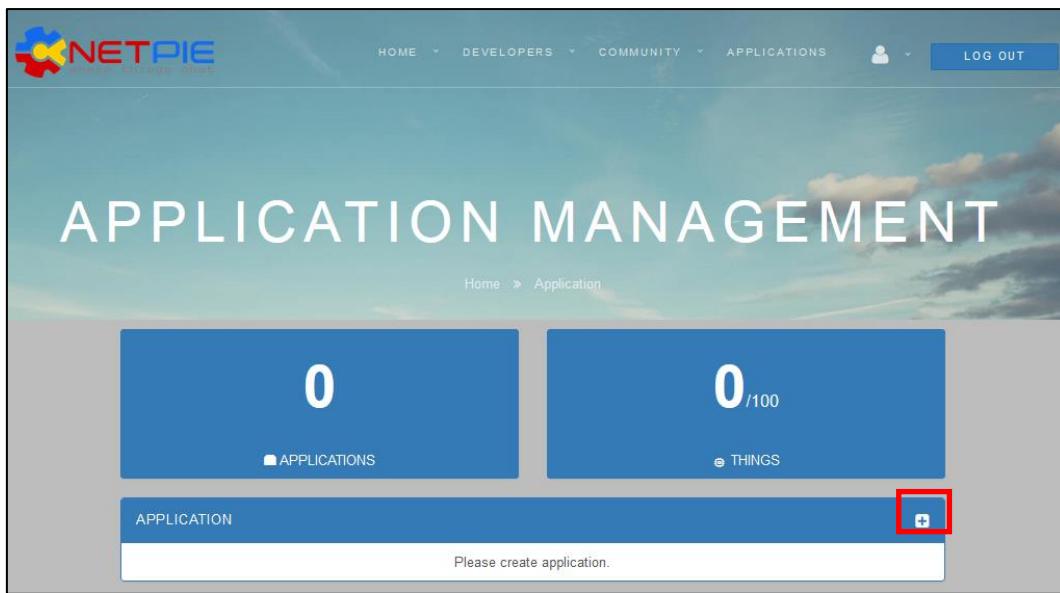
รูปที่ 2.3 เมนูในการเปลี่ยนรหัสผู้ใช้งาน NETPIE

2.2 การสร้างแอปพลิเคชัน

หลังจากสมัครสมาชิก และล็อกอินได้เรียบร้อยแล้ว ผู้ใช้สามารถสร้างแอปพลิเคชันได้โดยทำการคลิกที่ปุ่ม + ดังนั้นกดต่อไปนี้

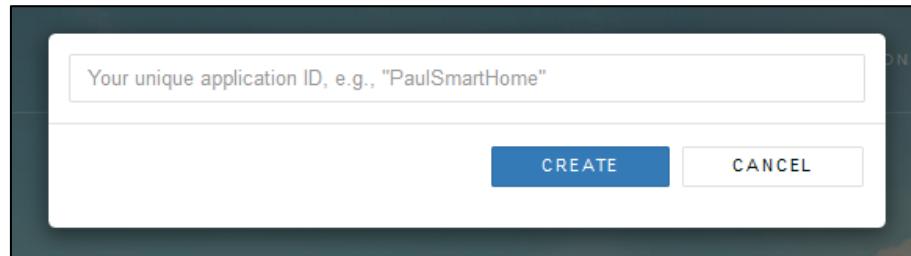
- เลือกที่เมนู APPLICATIONS เพื่อเข้าไปที่หน้า APPLICATION MANAGEMENT ซึ่งแสดงแอปพลิเคชันทั้งหมดที่ผู้ใช้งานมีอยู่ การสร้างแอปพลิเคชันใหม่ทำได้โดยการคลิกที่ปุ่ม + ดังแสดงในรูป





รูปที่ 2.4 หน้าจัดการแอปพลิเคชันสำหรับผู้ใช้งาน NETPIE

2. ในหน้าต่าง Pop-up ให้กำหนดชื่อแอปพลิเคชัน (Application ID หรือ AppID) ที่ต้องการ ในตัวอย่างนี้ใช้ชื่อว่า PaulSmartHome จากนั้นคลิกปุ่ม CREATE การตั้งชื่อ AppID จะต้องไม่ซ้ำกับผู้อื่น ดังนั้นควรเลือกชื่อที่มีความเฉพาะตัวเพื่อให้จำได้ เนื่องจาก AppID นี้จะถูกนำไปใช้ในการพัฒนาโปรแกรมต่อไป

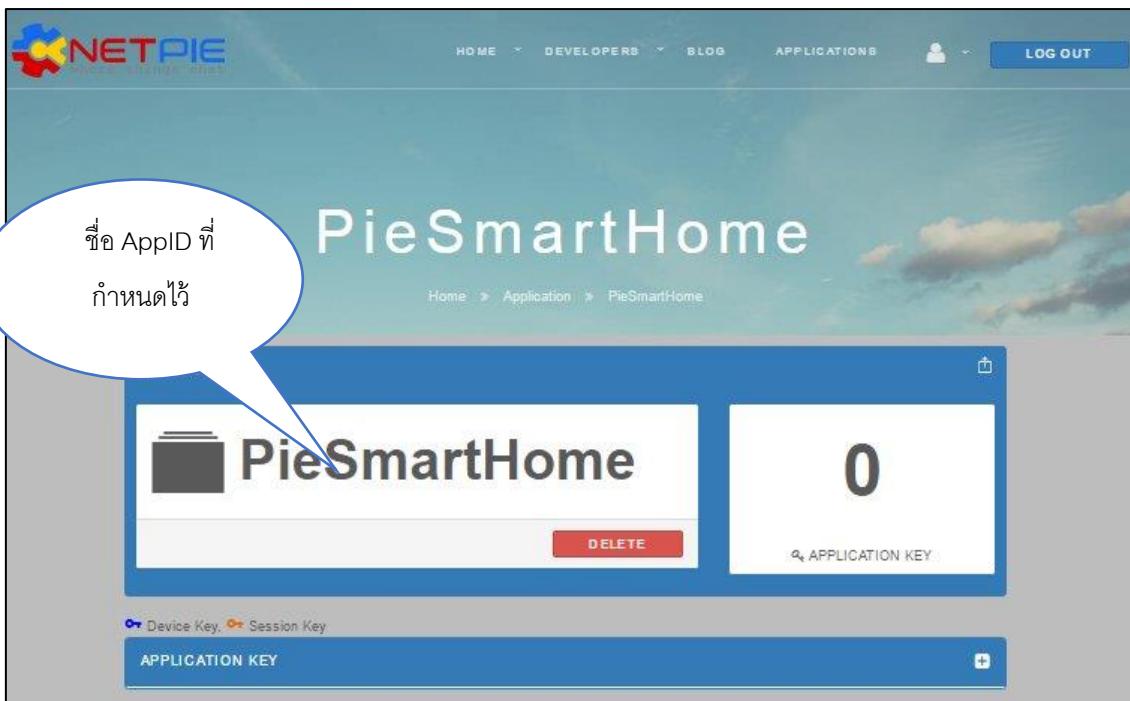


รูปที่ 2.5 กล่องให้ผู้ใช้ตั้งชื่อ AppID

3. หากสร้างแอปพลิเคชันใหม่ได้สำเร็จที่หน้าต่าง APPLICATION ในหน้า APPLICATION MANAGEMENT จะปรากฏช่องของชื่อ AppID ที่เรากำหนดไว้ หากชื่อซ้ำหรือมีข้อผิดพลาดจะมีข้อความแจ้งเตือนขึ้นมาดังนี้

This AppID is already in use. Please select another AppID ×

หลังจากสร้างสำเร็จจะเปลี่ยนไปที่หน้าของแอปพลิเคชันที่สร้างขึ้นใหม่โดยอัตโนมัติ

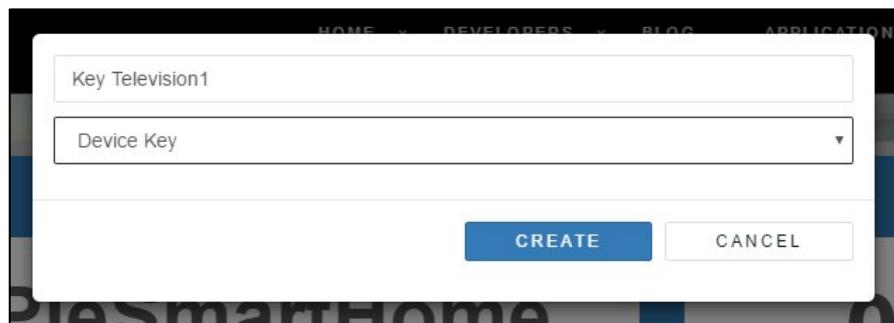


รูปที่ 2.6 ตัวอย่างหน้าแอปพลิเคชันนึงของ NETPIE

4. สร้าง Application Key โดยคลิกที่ปุ่ม +



5. กำหนดชื่อของ Application Key ตามต้องการ ในตัวอย่างใช้ชื่อความว่า **Key Television1** และเลือกชนิดของ Key ใน Drop-down box ให้เป็น Device Key



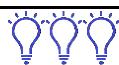
รูปที่ 2.7 การตั้งชื่อและเลือกชนิด Application Key

Application Key มี 2 ประเภท คือ

1. Device Key คือ Key ที่ใช้กับอุปกรณ์ประเภทกายภาพ โดย Device Key ของอุปกรณ์นั้นจะถูกจดจำไปตลอดถึงแม้อุปกรณ์จะไม่ได้เชื่อมต่อกับ NETPIE แล้ว แต่มีมีการเชื่อมต่อ

ใหม่ จะถูกมองว่าเป็นอุปกรณ์ตัวเดิม ประโยชน์คือเราจะสามารถจดจำอ้างอิงถึงอุปกรณ์ตัวนั้นได้

- Session Key คือ Key ที่ใช้กับอุปกรณ์ที่ทำการใช้งานไม่ทราบเข่น เบราว์เซอร์ เมื่อยกเลิกการเชื่อมต่อหรือปิดเบราว์เซอร์ไปตัวตนของอุปกรณ์นั้นจะถูกลบพิ้ง เมื่อเชื่อมต่อมาอีกรอบจะระบบจะสร้างบัตรผ่านใหม่จาก Key ตัวเดิมให้ โดย NETPIE จะไม่ถือว่าอุปกรณ์ที่ Online แต่ละครั้งมีตัวตนเดียวกัน แตกต่างจากการนี้ Device Key



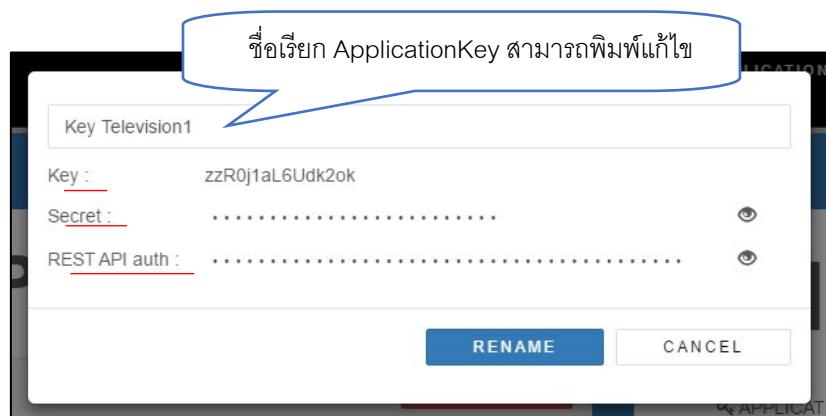
แนะนำให้ใช้ Device Key กับ Hardware และใช้ Session Key กับ Browser, HTML5 โดยความแตกต่างของ Key ทั้งสองประเภทจะแสดงในหน้าของ Application ด้วยรูปแบบและสีต่างกัน

เมื่อสร้างสำเร็จ จะมีชื่อ Key Television1 ปรากฏในหน้าต่าง APPLICATION KEY



- ตรวจสอบข้อมูลหรือดูรายละเอียด Key ด้วยการคลิกที่ Key Television1 จะปรากฏข้อมูลของ Application Key ซึ่งประกอบด้วย

- Application Key Name สามารถเปลี่ยนแปลงชื่อได้ตามที่ต้องการ มีไว้ให้ผู้ใช้สามารถระบุอุปกรณ์ของตนภายใต้ AppID หนึ่งเท่านั้น จึงสามารถตั้งชื่อกับชื่อผู้คนได้



รูปที่ 2.8 การเปลี่ยนชื่อ Application Key

- Key เป็นกุญแจที่อุปกรณ์ใช้สำหรับเชื่อมต่อ NETPIE

- Secret เป็นรหัสลับหรือรหัสผ่านที่อุปกรณ์ต้องใช้คู่กับ Key สำหรับให้อุปกรณ์เขื่อมต่อ NETPIE
- REST API auth เป็นกุญแจที่เรียงต่อ กับรหัสลับเพื่อใช้สำหรับการใช้งานด้วย REST API

2.3 การติดตั้งเครื่องมือพัฒนา

เครื่องมือสำหรับการพัฒนา ประกอบไปด้วย

- Arduino IDE และ SDK สำหรับ ESP8266
- NETPIE Microgear สำหรับ ESP8266
- CP2012 USB to UART driver
- เว็บเบราว์เซอร์ Chrome และ Firefox

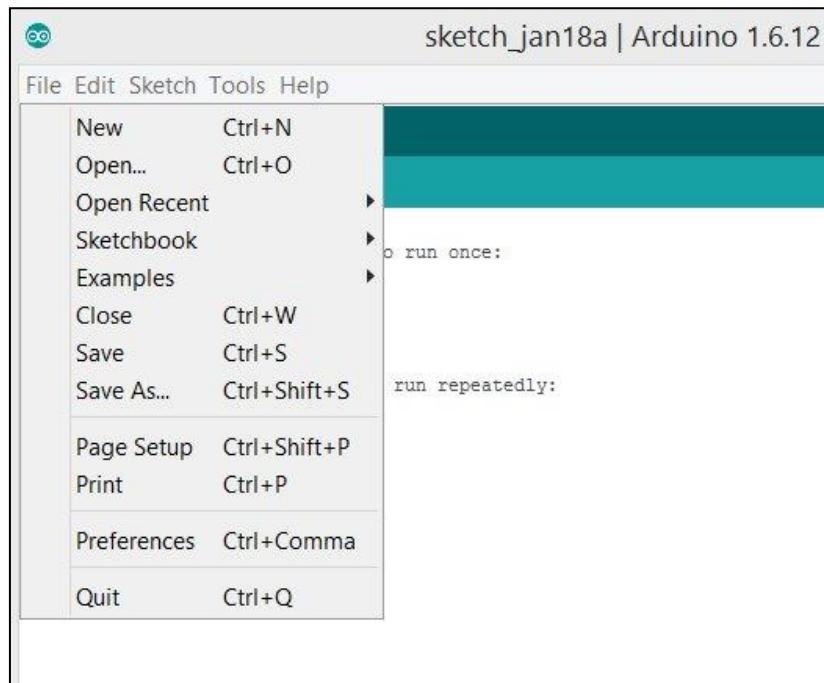
Arduino IDE

ดาวน์โหลด Arduino IDE ได้จาก <https://www.arduino.cc/en/Main/Software> ซึ่งสามารถเลือกไฟล์เพื่อติดตั้งในแพลตฟอร์มต่างๆ อาทิ Windows, Mac OS X, Linux หาก Arduino IDE ที่ต้องการติดตั้งไม่ได้เป็นเวอร์ชันล่าสุด ท่านสามารถไปดาวน์โหลดเวอร์ชันก่อนๆ ได้ที่ <https://www.arduino.cc/en/Main/OldSoftwareReleases>



เวอร์ชันของ Arduino IDE ที่ใช้ทดสอบกับ code ในเอกสารนี้คือเวอร์ชัน 1.6.12

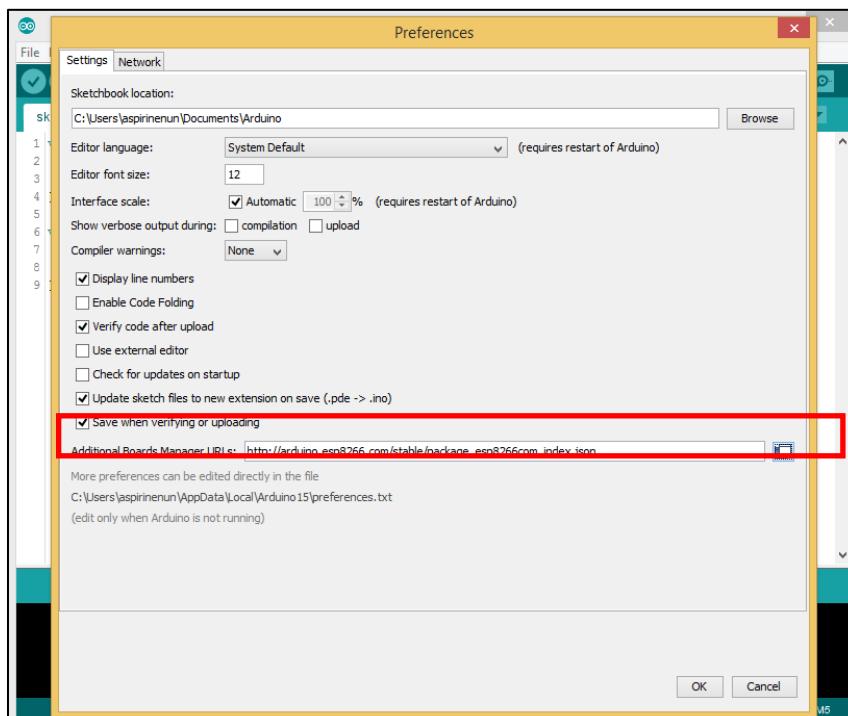
หลังจากติดตั้งเสร็จ ไปที่เมนู File --> Preferences



รูปที่ 2.9 การเลือกเมนู Preferences ใน Arduino IDE

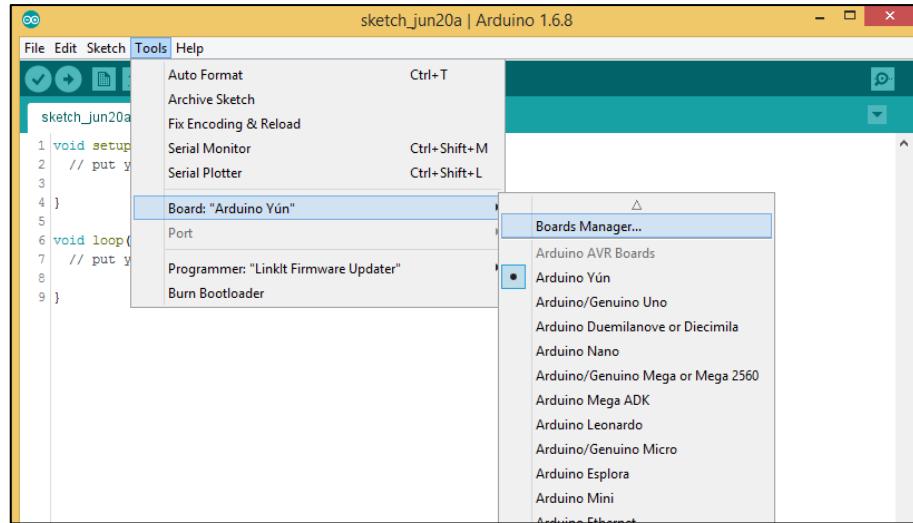
ใส่ข้อความต่อไปนี้ ลงในช่อง Additional Board Manager URLs

http://arduino.esp8266.com/stable/package_esp8266com_index.json



รูปที่ 2.10 การใส่ลิงค์เพื่อติดตั้ง ESP8266 Arduino Core

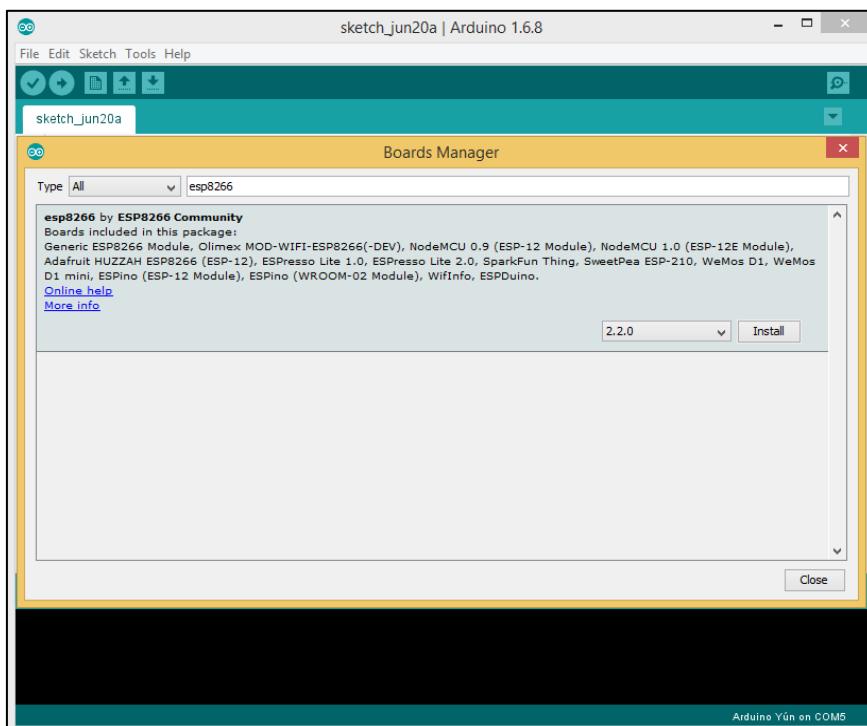
คลิกที่เมนู Tools --> Board: ?????? --> Board Manager...



รูปที่ 2.11 การเลือกเวอร์ชันของ ESP8266 สำหรับติดตั้ง

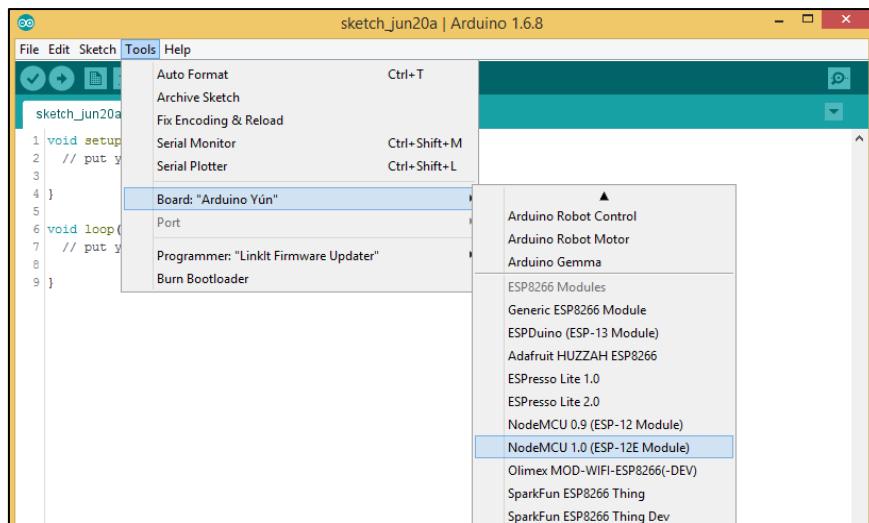
ค้นหาคำว่า ESP8266 เมื่อปรากฏผลลัพธ์ ให้เลือกเวอร์ชัน 2.2.0 ใน Drop-down Option และคลิก

Install



รูปที่ 2.12 การติดตั้ง ESP8266

ในเมนู Tools จะมีบอร์ด ESP8266 ขึ้นด้วยตัวๆ เพิ่มขึ้นมา เลือกให้ตรงกับชนิดของบอร์ดที่ใช้ ในที่นี่ให้เลือก NodeMCU 1.0 (ESP-12E Models)



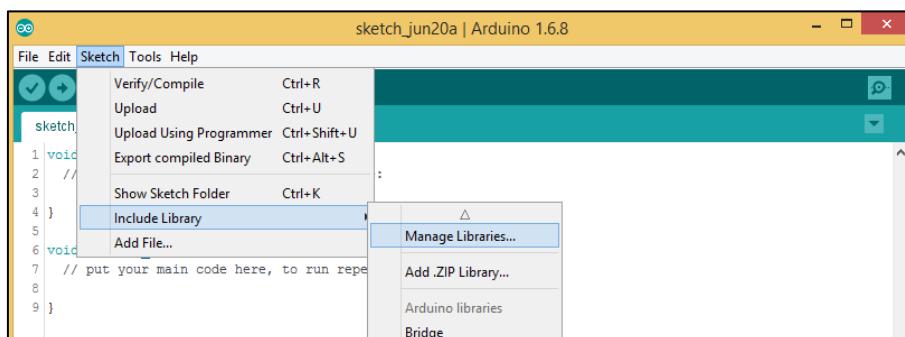
รูปที่ 2.13 การเลือกชนิดของบอร์ด ESP8266

NETPIE Microgear Library สำหรับ ESP8266

ดาวน์โหลด Microgear สำหรับ ESP8266 จาก GitHub ของ NETPIE หรือ Manage Libraries

กรณีที่ 1 ถ้าดาวน์โหลด Microgear จาก GitHub (<https://github.com/netpieio/microgear-esp8266-arduino>) ให้คลิกเมนู Sketch --> Include Library --> Add .ZIP Library... เลือก Zip File ที่ดาวน์โหลดมา คลิก Choose ถ้าติดตั้งสำเร็จ เข้าไปที่เมนู Sketch --> Include Library จะเห็นชื่อ ESP Microgear

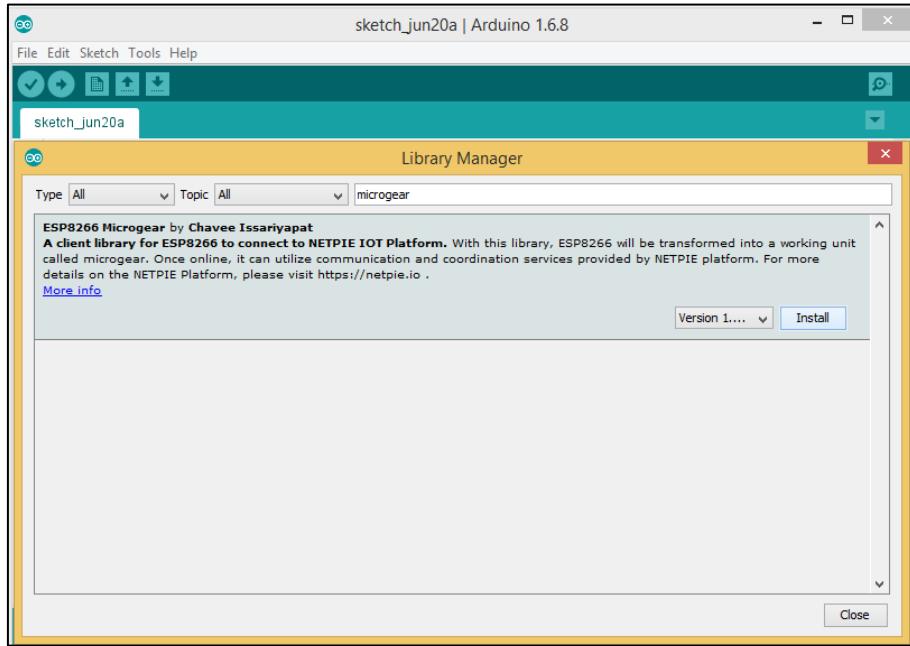
กรณีที่ 2 ถ้าดาวน์โหลด Microgear จากเมนู Manage Libraries... ให้คลิกเมนู Sketch --> Include Library --> Manage Libraries...



รูปที่ 2.14 การติดตั้ง Microgear Library จาก Arduino IDE

ค้นหาคำว่า Microgear ที่ดาวน์โหลดมา คลิก Choose

ถ้าติดตั้งสำเร็จ เมื่อเข้าไปที่เมนู Sketch --> Include Library--> Manage Libraries... จะเห็นชื่อ ESP8266 Microgear ถ้าลงเสร็จเรียบร้อยแล้วจะปรากฏข้อความว่า Installed แต่ถ้ายังไม่สำเร็จให้คลิกปุ่ม Install เพื่อติดตั้ง



รูปที่ 2.15 การติดตั้ง Microgear Library ลงบนบอร์ด

USB Driver

ใน Workshop เราจะใช้บอร์ด NodeMCU ซึ่งมี USB Port ที่ใช้ชิป CP2102 ดังนั้นเราจำเป็นที่จะต้องติดตั้ง Driver ให้ระบบปฏิบัติการรู้จัก USB Device ชนิดนี้ โดยดาวน์โหลด USB Driver ได้ที่

<https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx>

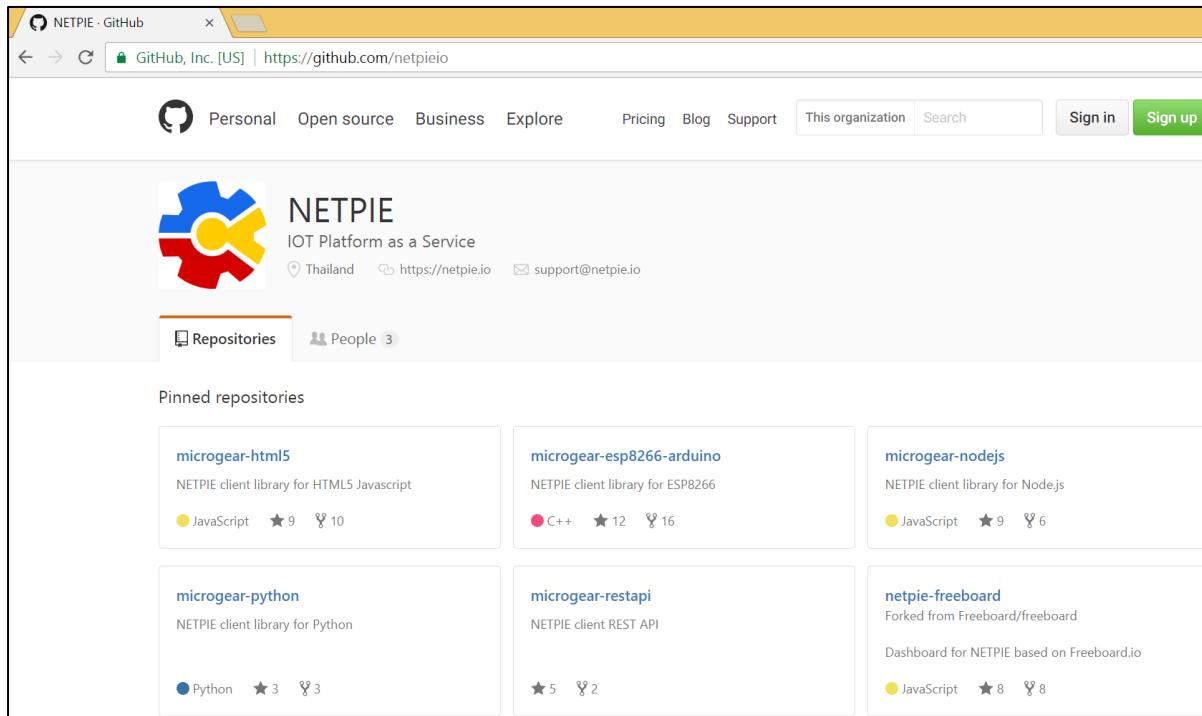
Web Browser

ขอแนะนำให้ใช้ Chrome หรือ Firefox หากยังไม่ได้ติดตั้ง สามารถดาวน์โหลดได้จาก

Chrome	https://www.google.com/chrome/browser/index.html
Firefox	https://www.mozilla.org/en-US/firefox/new/

การใช้งาน NETPIE กับอุปกรณ์ชนิดอื่นๆ

นอกจาก ESP8266 แล้ว NETPIE ยังมี Microgear สำหรับภาษาต่างๆ เช่น HTML5, Node.js, Python, Java, Android, C# และซอฟต์แวร์อื่นๆ โดยสามารถดาวน์โหลดพร้อมคู่มือการใช้งานได้ใน GitHub (<https://github.com/netpieio/>)



รูปที่ 2.16 หน้า GitHub ของ NETPIE

3. HTML 5 MICROGEAR

การใช้ NETPIE Client Library ในอีกรูปแบบหนึ่งที่ถูกนำมาประยุกต์ใช้อย่างแพร่หลายคือ HTML5 Microgear Library ซึ่งสามารถเปลี่ยน Web Browser ให้เป็น Thing (หรือ NETPIE device) เพื่อสื่อสารกับ Thing หรืออุปกรณ์อื่นๆ ที่เชื่อมต่อกันภายใต้แพลตฟอร์ม NETPIE ได้ HTML5 Microgear Library นี้ เขียนด้วยภาษา JavaScript นักพัฒนาสามารถนำไปต่ออยอดเป็น Web Application อื่นๆ โดยที่ไม่ต้องติดตั้งโปรแกรมใดเพิ่มเติมอีก และสามารถใช้ได้กับอุปกรณ์ทั่วไป เช่น คอมพิวเตอร์หรือสมาร์ทโฟนที่ใช้ Browser ต่างๆ ได้ในปัจจุบัน

Lab 3.1: Hello NETPIE

Lab นี้เป็นการทดลองใช้ HTML5 Microgear ด้วยวิธีง่ายๆ โดยการเขียนไฟล์ HTML ด้วย Javascript เพียงไฟล์เดียวเพื่อสร้าง Microgear ในการส่งข้อความผ่าน NETPIE กลับมายัง Browser ของตนเองเป็นระยะๆ Browser ซึ่งก็ถือว่าเป็น Thing ชนิดหนึ่งสามารถเชื่อมต่อกับ NETPIE ผ่านทาง Microgear Library ขั้นตอนการทดลอง มีดังนี้

1. พิมพ์โค้ดข้างล่างลงใน Text Editor โดยแก้ไข APPID, KEY, SECRET ตามที่ได้สร้างไว้บนเว็บ NETPIE โดยให้เลือกชนิด Application Key เป็น Session Key รวมทั้งตั้งชื่ออุปกรณ์ (ALIAS) เพื่อใช้งานจริงในการสื่อสาร จะเห็นได้ว่าการเรียกใช้ Microgear Library ถูก Define ไว้ในบรรทัดแรกที่อ้างถึง Source <https://cdn.netpie.io/microgear.js>

```
<script src="https://cdn.netpie.io/microgear.js"></script>

<script>
    const APPID = "YOUR_APPID";      // แทนที่ด้วย AppID
    const KEY = "YOUR_KEY";          // แทนที่ด้วย Key
    const SECRET = "YOUR_SECRET";    // แทนที่ด้วย Secret

    const ALIAS = "myhtml";           // ตั้งชื่อเรียกอุปกรณ์ (อย่างไรก็ได้ไม่จำกัด)

    var microgear = Microgear.create({
        key: KEY,
        secret: SECRET,
```

```

        alias : ALIAS
    });

    // สร้างฟังก์ชันที่จะตอบสนองต่อข้อความที่เข้ามา
    microgear.on('message', function(topic,msg) {
        //ในที่นี่เราจะเอาข้อความไปแทรกข้อความของ HTML element ชื่อ data
        document.getElementById("data").innerHTML = msg;
    });

    // สร้างฟังก์ชันที่จะถูกเรียกเมื่อเชื่อมต่อ NETPIE สำเร็จ
    microgear.on('connected', function() {
        microgear.setAlias(ALIAS); // ตั้งชื่อเรียกให้กับอุปกรณ์ตามตัวแปรที่ตั้งไว้ส่วนต้นของไฟล์

        // แสดงข้อความให้ทราบว่าเชื่อมต่อสำเร็จ
        document.getElementById("data").innerHTML = "Now I am connected with netpie...";

        // ตั้งค่า timer ให้ทำงานทุก 1 วินาที (ตัวเลข 1000 มีหน่วยเป็น ms หมายถึง 1000 ms)
        setInterval(function() {
            // ส่งข้อความไปยังอุปกรณ์ที่มีชื่อตามตัวแปร ALIAS ซึ่งก็คือตัวเอง
            microgear.chat(ALIAS,"Hello from myself at "+Date.now());
        },1000);
    });

    microgear.connect(APPID); // สร้างการเชื่อมต่อไปยัง NETPIE

```

</script>

<div id="data">_____</div>

Microgear ในตัวอย่างนี้คือ microgear-html5 ใน GitHub ซึ่งเป็น Client Library ของ NETPIE ที่จะเปลี่ยน Web Browser ให้เป็น Microgear เพื่อสื่อสารกับ Microgear อื่นๆ

2. Save เป็นไฟล์ชื่อ hellonetpie.html จากนั้นดับเบิลคลิกหรือคลิกขวาเพื่อเปิดไฟล์ด้วย Browser
3. บนหน้า Browser ที่เปิดขึ้นมา จะปรากฏข้อความ

Hello from myself at xxxxxxxxxxxx

โดย xxxxxxxxxxx เป็น Unix Time ที่จะเพิ่มขึ้นเรื่อยๆ ทุกๆ 1000 ms หรือ 1 วินาที

Lab 3.2: HTMLChat

ในการทดลองนี้ จะแสดงตัวอย่างการสื่อสารกันระหว่าง Web Browser 2 ตัว ผ่าน NETPIE โดยดัดแปลงมาจาก Lab 3.1 ดังนี้

- สร้าง Application Key ใหม่ 2 ค่า ภายใต้แอปพลิเคชัน AppID เดียวกัน
- สร้างไฟล์ HTML ขึ้นมาใหม่ 2 ไฟล์ (alice.html และ bob.html) โดยใช้โค้ดที่แสดงข้างล่าง สังเกตว่าความแตกต่างของห้องสองไฟล์อยู่ที่เพียงตัวแปร MYNAME และ YOURNAME เท่านั้น ที่สลับที่กัน

alice.html

```
<script src="https://cdn.netpie.io/microgear.js"></script>

<script>
    const APPID = "YOUR_APPID";
    const KEY = "KEY_1";
    const SECRET = "SECRET_1";

    const MYNAME = "alice";           // ชื่อของตัวเอง
    const YOURNAME = "bob";          // ชื่อของเพื่อนที่จะคุยกับ

    var microgear = Microgear.create({
        key: KEY,
        secret: SECRET,
    });

    microgear.on('message',function(topic,msg) {
        document.getElementById("data").innerHTML = msg;
    });

    microgear.on('connected', function() {
        microgear.setAlias(MYNAME);    // ตั้งชื่อตัวเอง
        document.getElementById("data").innerHTML = "Now I am connected
with netpie...";

        setInterval(function() {
            // ใช้งานชื่อ chat ในการส่งข้อความหาเพื่อน
        }, 1000);
    });
</script>
```

```

        microgear.chat(YOURNAME, "Hello from "+MYNAME+" at
"+Date.now());
        },1000);
    });

    microgear.on('disconnected', function() {
        document.getElementById("data").innerHTML = "Now I am
disconnected with netpie...";
    });

    microgear.connect(APPID);

```

</script>

<div id="data">_____</div>

bob.html

```

<script src="https://cdn.netpie.io/microgear.js"></script>

<script>
    const APPID  = "YOUR_APPID";
    const KEY    = "KEY_2";
    const SECRET = "SECRET_2";

    const MYNAME   = "bob";           // ตั้งชื่อของตัวเอง
    const YOURNAME = "alice";         // ตั้งชื่อของเพื่อนที่จะคุยด้วย

    var microgear = Microgear.create({
        key: KEY,
        secret: SECRET,
    });

    microgear.on('message',function(topic,msg) {
        document.getElementById("data").innerHTML = msg;
    });

    microgear.on('connected', function() {
        microgear.setAlias(MYNAME);      // ตั้งชื่อตัวเอง
        document.getElementById("data").innerHTML = "Now I am connected
with netpie...";
        setInterval(function() {

```

```

// chat หาเพื่อน
microgear.chat(YOURNAME, "Hello from "+MYNAME+" at
"+Date.now());
},1000);
});

microgear.on('disconnected', function() {
    document.getElementById("data").innerHTML = "Now I am
disconnected with netpie...";
});

microgear.connect(APPID);

```

<div id="data">_____</div>

3. ใส่ APPID (เหมือนกันในทั้งสองไฟล์) และ แต่ละคู่ KEY กับ SECRET ลงในแต่ละไฟล์
4. Save ไฟล์ทั้งสองไฟล์ แล้วดับเบิลคลิกหรือคิกขวาเพื่อเปิดไฟล์ทั้งสองด้วย Browser
5. ผลลัพธ์ที่ได้ในหน้าต่างที่เปิดขึ้นมาจาก alice.html จะเป็นดังนี้

Hello from bob at xxxxxxxxxxxx

และข้อความในหน้าต่างที่เปิดขึ้นมาจาก bob.html จะเป็นดังนี้

Hello from alice at yyyy/yyyyyy

โดย xxxxxxxxx และ yyyy/yyyyyy เป็น Unix Time ที่จะเพิ่มขึ้นเรื่อยๆ ทุก 1000 ms

Lab 3.3: Room Chat

ในการทดลองนี้แสดงตัวอย่างการสื่อสารกันระหว่าง Web Browsers ผ่าน NETPIE ที่มีการใช้งานที่ซับซ้อนกว่าเดิม เป็นการทบทวนการใช้ Function Call ต่างๆ ใน HTML5 Microgear Library (<https://github.com/netpieio/microgear-html5/blob/master/README.th.md>) เพื่อเปลี่ยน Web Browser ให้เป็น Microgear และทำการสื่อสารอัปเดทข้อมูลกัน มีขั้นตอนดังนี้

1. ศึกษาจากไฟล์ room_client.html และ room_server.html เพื่อทบทวน Function Call ต่างๆ ที่ระบุไว้ใน HTML5 Microgear Library
2. เปิดไฟล์ room_client.html ด้วย Text Editor และแก้ไขที่บรรทัด var slot = your_number; โดยให้เปลี่ยนเป็นหมายเลขที่แต่ละท่านได้รับ แล้วทำการ Save File

3. คลิกเพื่อเปิดไฟล์ room_client.html และ room_server.html ด้วย Browser ศึกษาการสื่อสารกันระหว่าง Browser ทั้งฝ่าย Client และ ฝ่าย Server และดูการแสดงผลดังแสดงในรูปที่ 3.1

room_client.html

```
<script src="https://cdn.netpie.io/microgear.js"></script><!-- Microgear library -->

<script>
    const APPID = "demobillboard"           //Application ID ของอุปกรณ์ที่เชื่อมต่อ NETPIE
    const KEY = "M0mqAyedJmlywVD";          // Key ของอุปกรณ์ที่เชื่อมต่อ NETPIE
    const SECRET = "moHzrlTv1Xu0k286LzxkoTiRn"; //Secret ของอุปกรณ์ที่เชื่อมต่อ NETPIE

    const ALIAS = "htmlgear";                // ชื่ออุปกรณ์ที่เชื่อมต่อ NETPIE
    var slot = 30;                          // slot number

    var microgear = Microgear.create({
        key: KEY,
        secret: SECRET,
        alias : ALIAS
    });

    function getColor(){
        return '#' + Math.random().toString(16).slice(2, 8).toUpperCase();
    }

    microgear.on('message',function(topic,msg) { //ตรวจสอบข้อความที่ส่งมา.yังอุปกรณ์
        //กำหนดข้อความให้แสดงผลใน tag id ชื่อ data
        document.getElementById("data").innerHTML = msg;
    });

    microgear.on('connected', function() { //ตรวจสอบเมื่อเชื่อมต่ออุปกรณ์กับ NETPIE
        microgear.setAlias(ALIAS); //กำหนดชื่ออุปกรณ์ที่เชื่อมต่อ กับ NETPIE
        document.getElementById("data").innerHTML = "Now I am connected with netpie..."; // แสดงข้อความเมื่อเชื่อมต่ออุปกรณ์กับ NETPIE

        setInterval(function() {           // กำหนดให้ทำงานตลอดเวลาทุกๆ 5000 ms
            var color = getColor();
        });
    });
</script>
```

```

// ส่งข้อความไปยังอุปกรณ์ชื่อ htmlgear ด้วย NETPIE
microgear.publish("/billboard/slot/" + slot, color);
document.getElementById("data").innerHTML = "Publish color : " + color;
document.body.style.backgroundColor = color;
}, 5000);
});

microgear.on('disconnected', function() { // ตรวจสอบเมื่อตัดการเชื่อมต่ออุปกรณ์กับ NETPIE
document.getElementById("data").innerHTML = "Now I am disconnected with netpie..."; // แสดงข้อความเมื่อตัดการเชื่อมต่ออุปกรณ์กับ NETPIE
});

// ตรวจสอบอุปกรณ์ที่เชื่อมต่อ NETPIE ด้วย APPID เดียวกัน
microgear.on('present', function(event) {
console.log(event);
});

// ตรวจสอบเหตุการณ์ที่อุปกรณ์เดียวกันเชื่อมต่อ NETPIE ด้วย APPID เดียวกันหายไป
microgear.on('absent', function(event) {
console.log(event);
});

microgear.connect(APPID); // เชื่อมอุปกรณ์กับ NETPIE
</script>

<div id="data">_____</div>

```

room_server.html

```

<script src="https://cdn.netpie.io/microgear.js"></script><!-- Microgear library -->

<style>
#rgb {
    width: 100%;
    font-size: 50px;
    vertical-align: middle;
}
.box {

```

```

        float: left;
        border: 1px solid black;
        margin-right: 50px;
        margin-bottom: 20px;
    }

    .boxcolor{
        width: 250px;
        padding: 10px;
        display: inline-block;
        color: #FFF;
        background-color: #FFF;
    }

    .right {
        text-align: center;
        width: 150px;
        padding: 10px;
        display: inline-block;
        color: #FFF;
        background-color: #000;
    }

}

</style>

<div id="data">_____</div>
<p/>
<div id="rgb"></div>

<script>
    var numberbox = 30;
    var htmlbox = "";

    for(var i=0; i<numberbox; i++){
        htmlbox+='div class="box"<div
class="right">' +(i+1) +'</div><div class="boxcolor"
id="'+(i+1) +'>' +i+'</div></div>';
    }

    document.getElementById("rgb").innerHTML = htmlbox;

    const APPID = "demobillboard"; // Application ID ของอุปกรณ์ที่เชื่อมต่อ NETPIE
    const KEY = "M0mqAyedJmlywVD"; // Key ของอุปกรณ์ที่เชื่อมต่อ NETPIE
    const SECRET = "moHzrlTv1Xu0k286LzxkoTiRn"; // Secret ของอุปกรณ์ที่เชื่อมต่อ
NETPIE

```

```

const ALIAS = "billboard";           // ชื่ออุปกรณ์ที่เชื่อมต่อ NETPIE

var microgear = Microgear.create({
    key: KEY,
    secret: SECRET,
    alias : ALIAS
});

microgear.on('message',function(topic,msg) { // ตรวจสอบข้อความที่ส่งมายังอุปกรณ์
    console.log(topic+" : "+msg);
    if(topic.indexOf("/demobillboard/billboard/slot")!==-1){
        if(msg.startsWith('#') || msg.startsWith('rgb(')){
            var m = topic.split("/demobillboard/billboard/slot/");
            document.getElementById(m[1]).style.color = msg;
            document.getElementById(m[1]).style.backgroundColor =msg;
        }
    }
});

microgear.on('connected', function() { // ตรวจสอบเมื่อเชื่อมต่ออุปกรณ์กับ NETPIE
    microgear.setAlias(ALIAS);          // กำหนดชื่ออุปกรณ์ที่เชื่อมต่อกับ NETPIE
    // แสดงข้อความเมื่อเชื่อมต่ออุปกรณ์กับ NETPIE
    document.getElementById("data").innerHTML = "Now I am connected with netpie...";
    microgear.subscribe("/billboard/slot/+");
});

microgear.on('disconnected', function() { // ตรวจสอบเมื่อตัดการเชื่อมต่ออุปกรณ์กับ NETPIE
    // แสดงข้อความเมื่อตัดการเชื่อมต่ออุปกรณ์กับ NETPIE
    document.getElementById("data").innerHTML = "Now I am disconnected with netpie...";
});

//ตรวจสอบอุปกรณ์ที่เชื่อมต่อกับ NETPIE ด้วย APPID เดียวกัน
microgear.on('present', function(event) {
    console.log(event);
});

// ตรวจสอบว่าอุปกรณ์ที่เคยเชื่อมต่อกับ NETPIE ด้วย APPID เดียวกันหายไป

```

```

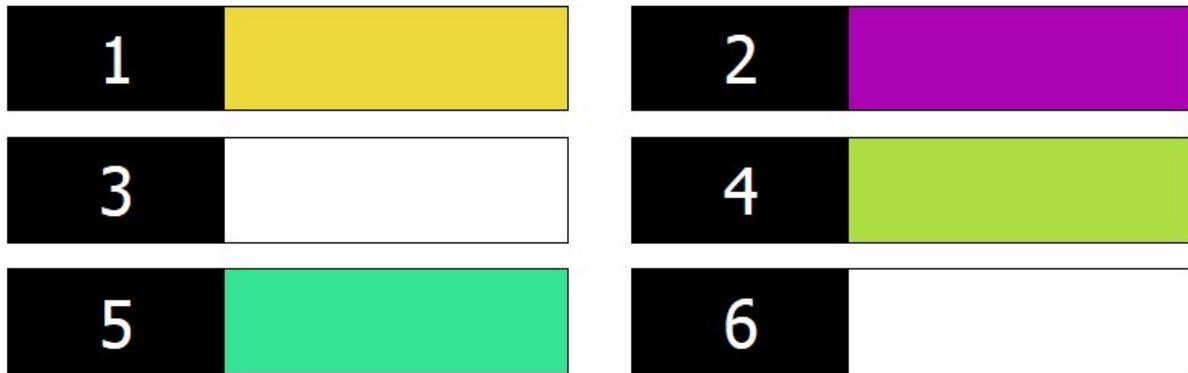
microgear.on('absent', function(event) {
    console.log(event);
});

microgear.connect(APPID); // เชื่อมอุปกรณ์กับ NETPIE

</script>

```

Now I am connected with netpie...



รูปที่ 3.1 การแสดงผลที่ Web Browser ที่เปิดไฟล์ room_server.html

แบบฝึกหัด

- ในไฟล์ room_client.html อธิบายความหมายของ Function Call นี้

```
microgear.publish("/billboard/slot/"+slot,color);
```

- ในไฟล์ room_server.html อธิบายความหมายของ Function Call นี้

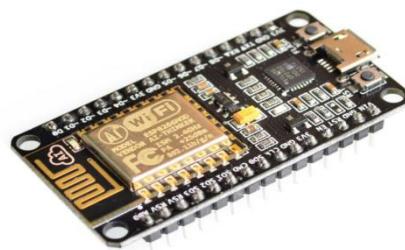
```
microgear.subscribe("/billboard/slot/+");
```

4. ESP8266 MICROGREAL

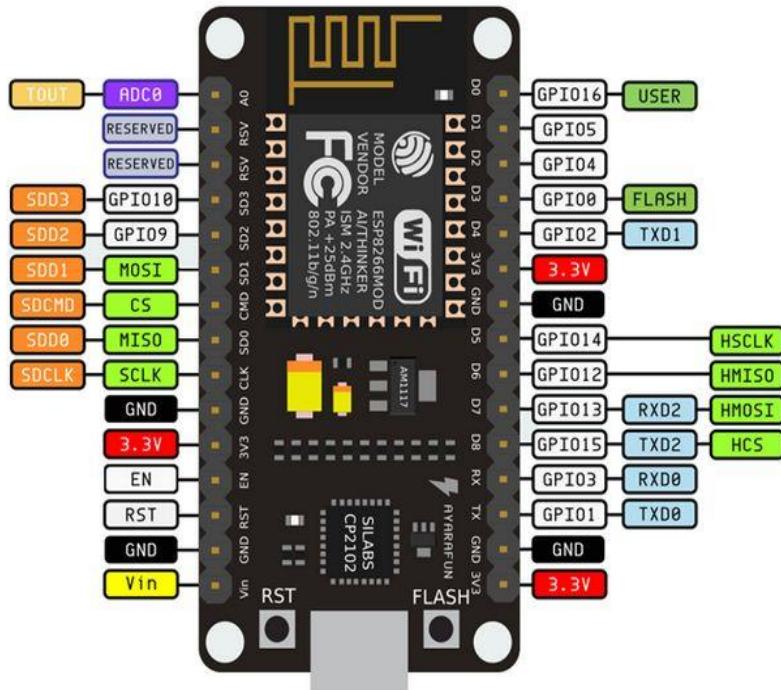
4.1 รู้จัก ESP8266/NODEMCU

บทนี้จะเกี่ยวข้องกับ Thing ประเภท Embedded Board เพื่อให้เห็นภาพว่าสามารถสร้างอุปกรณ์ยาร์ดแวร์ให้มาเข้ามาร่วมต่อกับ NETPIE ได้อย่างไร โดยในการทดลองศึกษา เราจะใช้ NodeMCU รุ่น 12E หรือ NodeMCU V2 หรือ NodeMCU Dev Kit v1.0 (<https://github.com/nodemcu/nodemcu-devkit-v1.0>) ซึ่งใช้โมดูล ESP8266-SoC (ESP-12E) ซึ่งเป็น 32-bit Microcontroller และมี Built-in WiFi

Node MCU-12E รวมทั้งผังคำอธิบายขาต่อแสดงไว้ดังรูปที่ 4.1



PIN DEFINITION



รูปที่ 4.1 Node MCU-12E และผังคำอธิบายขาต่อ

ในการเขียนโปรแกรมลงบบ NodeMCU เราจะใช้ Arduino IDE ในการ Compile และ Flash โปรแกรมลงไปผ่านทางสาย microB-USB ในส่วนของโครงสร้างโค้ด จะเหมือนกับการเขียนโปรแกรมบนบอร์ด Arduino โดยมีฟังก์ชันสองส่วนคือ ส่วนตั้งค่าและส่วนทำงานหลัก ดังแสดงข้างล่าง

```
void setup() {  
    // ส่วนนี้ทำครั้งเดียวตอนเริ่มต้นการทำงาน  
}  
  
void loop() {  
    // ส่วนนี้เมื่อส่วนหลัก ทำงานหาผลลัพธ์ไม่รู้จบ  
}
```

ท่านสามารถทดสอบการทำงานของบอร์ดเบื้องต้นดังนี้

1. เชื่อมต่อบอร์ดกับคอมพิวเตอร์ด้วยสาย microB-USB

2. ตรวจสอบ COM Port ที่ใช้งานให้ถูกต้อง

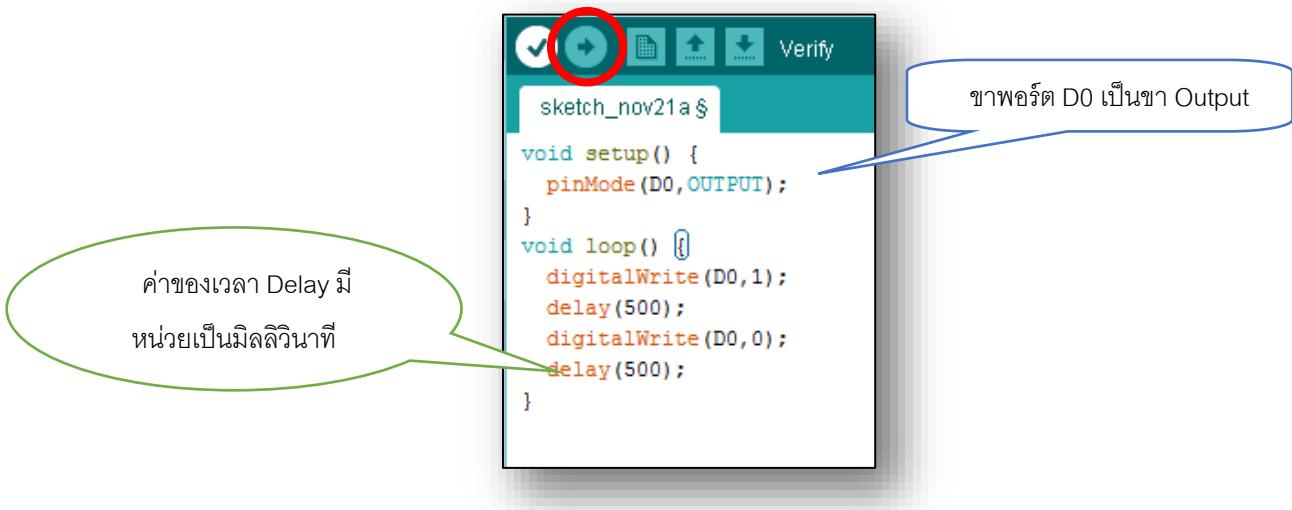
2.1 สำหรับ Windows ไปที่ Control panel --> System --> Device Manager และดูที่ Ports (COM & LPT) จะเห็นบรรทัด Silicon Labs CP210x USB to UART Bridge (COMx) เช่น เป็น COM3 เป็นต้น

2.2 เปิดโปรแกรม Arduino IDE และไปที่เมนู Tools -->Board --> NodeMCU 1.0 (ESP-12E Module) จากนั้นไปที่เมนู Tools > Port ถ้ามีหลาย Port แสดงอยู่ ให้ตรวจสอบว่าตรงกับ Port ใน 2.1

3. ใน Arduino IDE ไปที่ File --> New เพื่อเขียนโปรแกรมดังนี้

```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(D0, OUTPUT);  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
    digitalWrite(D0,1);  
    delay(500);  
    digitalWrite(D0,0);  
    delay(500);  
}
```

เมื่อเขียนโค้ดเรียบร้อยแล้ว ให้กดปุ่มลูกศรเพื่อ Compile และ Upload หากไม่มีข้อผิดพลาด โปรแกรมที่ Compile จะถูก Flash ลงใน NodeMCU และเริ่มต้นการทำงานอัตโนมัติ ตัวอย่างข้างล่าง เป็นโปรแกรมที่จะสั่งให้ไฟ LED บนบอร์ด กระพริบด้วยค่าเวลาเปิด 0.5 วินาที



รูปที่ 4.2 ตัวอย่างการเขียนโปรแกรมลงใน NodeMCU

Lab 4.1: NETPIE Blink

Lab 4.1 เป็นการโปรแกรม NodeMCU ให้ตอบสนองต่อข้อความ (Message) “1” และ “0” โดยถ้าได้รับข้อความ “1” NodeMCU จะเปิด LED แต่ถ้าเป็น “0” จะปิด LED

1. เปิดไฟล์ netpie_blink.ino ใน Folder เพื่อแก้ไข หรือสร้างไฟล์ขึ้นใหม่ตามโค้ดด้านล่าง
2. สร้าง AppID และสร้าง Application Key โดยเลือกชนิด Device Key
3. ใส่ APPID, KEY, และ SECRET ลงในไฟล์
4. Save และ Upload ไฟล์เข้า NodeMCU
5. เปิด Console โดยไปที่ Tools --> Serial Monitor หากโปรแกรมถูกต้องจะเห็นตามแสดงในหน้าต่าง Console ดังแสดงในรูปที่ 4.3 สำหรับการเปิดและปิดไฟเทียบกับบรรทัด Incoming message 0 หรือ 1

netpie_blink.ino

```
// #include  สองบรรทัดด้านล่างนี้ต้องมีเสมอ
#include <ESP8266WiFi.h>
#include <MicroGear.h>

const char* ssid      = "SSID";           // ใส่ชื่อ WiFi SSID แทน SSID
const char* password = "PASSWORD";        // ใส่รหัสผ่าน WiFi แทน PASSWORD

#define APPID    "YOUR_APPID"           // ให้ YOUR_APPID แทนที่ด้วย AppID
#define KEY      "YOUR_KEY"             // ให้ YOUR_KEY แทนที่ด้วย Key
#define SECRET   "YOUR_SECRET"          // ให้ YOUR_SECRET แทนที่ด้วย Secret

#define ALIAS    "pieblink"            // ตั้งชื่อเล่นให้ device นี้ เป็น pieblink

WiFiClient client;

int timer = 0;
char state = 0;

MicroGear microgear(client);           // ประกาศตัวแปร microgear

// สร้างฟังก์ชันที่จะถูกเรียกเมื่อมีข้อความเข้ามา
void onMsghandler(char *topic, uint8_t* msg, unsigned int msglen) {
    Serial.print("Incoming message -->");
    msg[msglen] = '\0';
    Serial.println((char *)msg);

// ถ้าข้อความที่เข้ามาเป็น 1 ให้เปิด LED ถ้าเป็น 0 ให้ปิด LED
    if(*(char *)msg == '1'){
        digitalWrite(LED_BUILTIN, LOW); // LED on
    }else{
        digitalWrite(LED_BUILTIN, HIGH); // LED off
    }
}

// สร้างฟังก์ชันที่จะถูกเรียกเมื่อ Microgear เชื่อมต่อ กับ NETPIE สำเร็จ
void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
```

```

// เราอาจจะใช้อุปกรณ์ดังชื่อหรือเปลี่ยนชื่อหรืออาจจะทำอะไรบางอย่าง เช่น subscribe
microgear.setAlias(ALIAS);

}

void setup() {
    // ประกาศให้เวลามีข้อความเข้ามาให้ไปทำฟังก์ชัน onMsgHandler()
    microgear.on(MESSAGE, onMsgHandler);

    // ประกาศให้เมื่อเชื่อมต่อสำเร็จให้ไปทำฟังก์ชัน onConnected()
    microgear.on(CONNECTED, onConnected);

    Serial.begin(115200);
    Serial.println("Starting...");

    pinMode(LED_BUILTIN, OUTPUT);

    // initiate Wifi
    if (WiFi.begin(ssid, password)) {
        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
        }
    }

    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    // initialize ตัวแปร microgear
    microgear.init(KEY, SECRET, ALIAS);

    // เชื่อมต่อไป NETPIE ไปยัง AppID ที่กำหนด
    microgear.connect(APPID); // ฟังก์ชันเชื่อมต่อ NETPIE
}

void loop() {
    // เช็คว่า Microgear ยังเชื่อมต่ออยู่หรือไม่
    if (microgear.connected()) { // ตรวจสอบการเชื่อมต่อ NETPIE
        Serial.println("connected"); // พิมพ์แจ้งการเชื่อมต่อ NETPIE สำเร็จ
    }
}

```



```
// เราต้องเรียก microgear.loop() เป็นระยะ เพื่อรักษาการเชื่อมต่อ
microgear.loop();

if (timer >= 1000) {
    Serial.println("Publish..."); // พิมพ์แจ้งการส่งข้อมูล NETPIE

    // chat หาตัวเองด้วย state ที่ตรงข้ามกัน
    if(state==0){
        microgear.chat(ALIAS,state);
        state=1;
    }else{
        microgear.chat(ALIAS,state);
        state=0;
    }
    timer = 0;
}

else timer += 100;

}
else {
    Serial.println("connection lost, reconnect...");

    if (timer >= 5000) {
        microgear.connect(APPID);
        timer = 0;
    }
    else timer += 100;
}

delay(100);
}
```

รูปที่ 4.3 Console ของโปรแกรม netpie_blink

กล่าวโดยสรุปคือ โปรแกรมนี้สั่งให้ NodeMCU ทำการ Chat หาตัวเองทุกๆ 1 วินาที ด้วยค่าที่ตรงข้ามกับสถานะปัจจุบัน (ถ้าเปิดอยู่ จะส่ง “0” แต่ถ้าปิดอยู่จะส่ง “1”) สลับกันไปเรื่อยๆ ผลที่เกิดขึ้นคือ LED จะกระพริบทุก 1 วินาที เป็นตัวอย่างของการเรียกใช้ Microgear Function เพื่อทำอะไรบางอย่าง เมื่อมีข้อความเข้ามา

Lab 4.2: IoT Switch

Lab 4.2 จะแสดงการสื่อสารแบบสองทางระหว่าง Thing 2 ชนิดคือทั้งอาร์ดเวย์และซูพ์ตัวเว็บ โดยจะให้ HTML5 ส่งค่า 0 และ 1 ไปยัง NodeMCU เมื่อ NodeMCU ได้รับข้อมูลจะกระทำการตามเงื่อนไขที่กำหนด และตอบกลับมาอย่าง HTML5 ขั้นตอนคือ

1. ฝัง NodeMCU เขียนไฟล์ pieled.ino ซึ่งดัดแปลงมาจากโค้ด netpie_blink.ino ใน Lab 4.1 (เพียงแต่ตัดส่วนตั้งเวลาและ Chat หาตัวเองออก) และสามารถใช้ AppID, Key และ Secret เดิมที่ใช้ใน Lab 4.1 ได้

2. ฝัง HTML5 Switch ให้เขียนโค้ด switch.html โดยใช้ AppID เดียวกับฝัง NodeMCU แต่ให้ไปสร้าง Application Key ใหม่ชนิด Session Key

3. ใส่ APPID, KEY, และ SECRET ลงในไฟล์ทั้งสอง

4. Save และ Upload ไฟล์ pieled.ino เข้าไปยัง NodeMCU

5. เปิด Console โดยไปที่ Tools → Serial Monitor และใช้ Browser เปิดไฟล์ switch.html

6. กดปุ่ม Switch บน Browser เพื่อทำการเปิดปิดไฟ

pieled.ino

```
#include <ESP8266WiFi.h>
#include <MicroGear.h>

const char* ssid      = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

#define APPID    "YOUR_APPID"
#define KEY     "YOUR_KEY"
#define SECRET   "YOUR_SECRET"

#define ALIAS    "pieled"

WiFiClient client;

int timer = 0;
MicroGear microgear(client);

void onMsghandler(char *topic, uint8_t* msg, unsigned int msglen) {
    Serial.print("Incoming message --> ");
    msg[msglen] = '\0';
    Serial.println((char *)msg);
    if(*(char *)msg == '1'){
        digitalWrite(LED_BUILTIN, LOW); // turn on the LED
        microgear.chat("switch","1");
    }
}
```

```

        }else{
            digitalWrite(LED_BUILTIN, HIGH); //turn off the LED
            microgear.chat("switch","0");
        }
    }

void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
    microgear.setName(ALIAS);
}

void setup() {
    microgear.on(MESSAGE,onMsghandler);
    microgear.on(CONNECTED,onConnected);

    Serial.begin(115200);
    Serial.println("Starting...");

    pinMode(LED_BUILTIN, OUTPUT);

    if (WiFi.begin(ssid, password)) {
        while (WiFi.status() != WL_CONNECTED) {
            delay(1000);
            Serial.print(".");
        }
    }

    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    microgear.init(KEY,SECRET,ALIAS);
    microgear.connect(APPID);
}

void loop() {
    if (microgear.connected()) {
        Serial.println("...");
        microgear.loop();
    }
}

```

```

        timer = 0;
    }
    else {
        Serial.println("connection lost, reconnect...") ;
        if (timer >= 5000) {
            microgear.connect(APPID) ;
            timer = 0;
        }
        else timer += 100;
    }
    delay(100);
}

```

switch.html

```

<script src="https://cdn.netpie.io/microgear.js"></script>

<script>
    const APPID = "YOUR_APPID";
    const KEY = "YOUR_KEY";
    const SECRET = "YOUR_SECRET";

    const ALIAS = "switch";

    var microgear = Microgear.create({
        key: KEY,
        secret: SECRET,
        alias : ALIAS
    });

    function toggle() {
        if(document.getElementById("button").innerText=="off"){
            microgear.chat('pieled','1');
        }
        else{
            microgear.chat('pieled','0');
        }
    }

    microgear.on('message',function(topic,msg) {

```

```

document.getElementById("data").innerHTML = msg;
if(msg=="1") {
    document.getElementById("button").innerText="on";
} else if(msg=="0") {
    document.getElementById("button").innerText="off";
}
});

microgear.on('connected', function() {
    microgear.setAlias(ALIAS);
    document.getElementById("data").innerHTML = "Now I am connected with
netpie...";
});

microgear.connect(APPID);
</script>

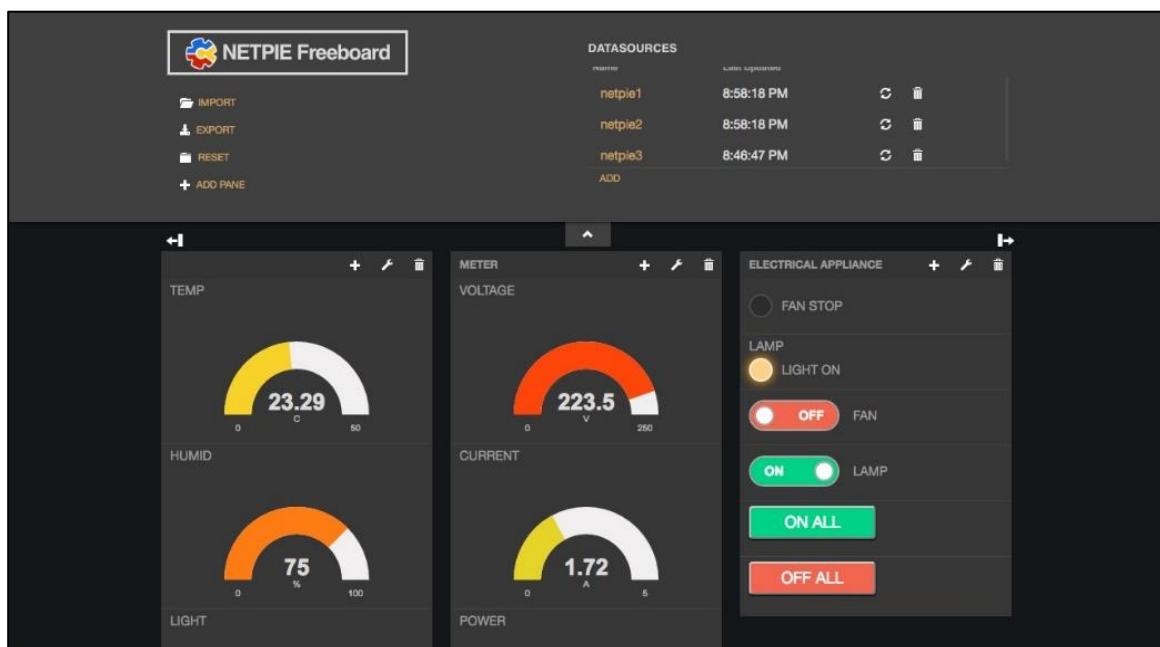
<div id="data">____</div>
<center>
<button onclick="toggle()" id="button">off</button>
</center>

```

5. FREEBOARD

Freeboard เป็น Web Application ที่สามารถสร้าง Dashboard เพื่อแสดงผลสำหรับ IoT แอปพลิเคชันโดยสามารถใช้เป็นกระดานส่วนตัว สามารถบันทึกผลลัพธ์ได้ สำหรับควบคุมอุปกรณ์ หรือ วางแผนหน้าบอร์ดเพื่อแสดงผลข้อมูลต่างๆ ที่ได้จากอุปกรณ์ เช่น เซนเซอร์ในระบบ IoT นอกจากนี้ยังสามารถแสดงผลเป็นกราฟได้ ส่วนหน้ากระดานหรือ Dashboard นั้น สามารถปรับแต่งได้โดยง่าย เพียงแค่ป้อนข้อมูลเข้าหรือกำหนดคำสั่งก็สามารถทำงานได้แล้ว โดยที่ผู้ใช้มีแค่เบื้องต้นของ HTML Web Page เอง และที่สำคัญคือข้อมูลนั้นมีการอัพเดทแบบ Real-time มีความเสถียรและเรียลไทม์ ได้ และเป็น Open-Source ซึ่งทำให้นักพัฒนาสามารถต่อยอดให้ดีขึ้นได้อีกด้วย

NETPIE Freeboard คือ Freeboard สำหรับการควบคุมและการแสดงผล (Visualization) ข้อมูลที่ได้มาจากการอุปกรณ์ที่ต่อ กับ NETPIE ที่มีงานได้พัฒนา Widget Plugins ขึ้นมาให้ผู้ใช้งานสามารถทำตามความต้องการได้หลากหลาย เช่น สามารถสร้างบูมควบคุมและใส่คำสั่ง Javascript สำหรับ Action ต่างๆ ได้ การใช้งาน NETPIE Freeboard นั้นสามารถใช้ Browser เปิดไฟล์ index.html ที่ได้จากการติดตั้ง NETPIE Freeboard หรือสามารถใช้ Freeboard ผ่านทางหน้าเว็บของ NETPIE



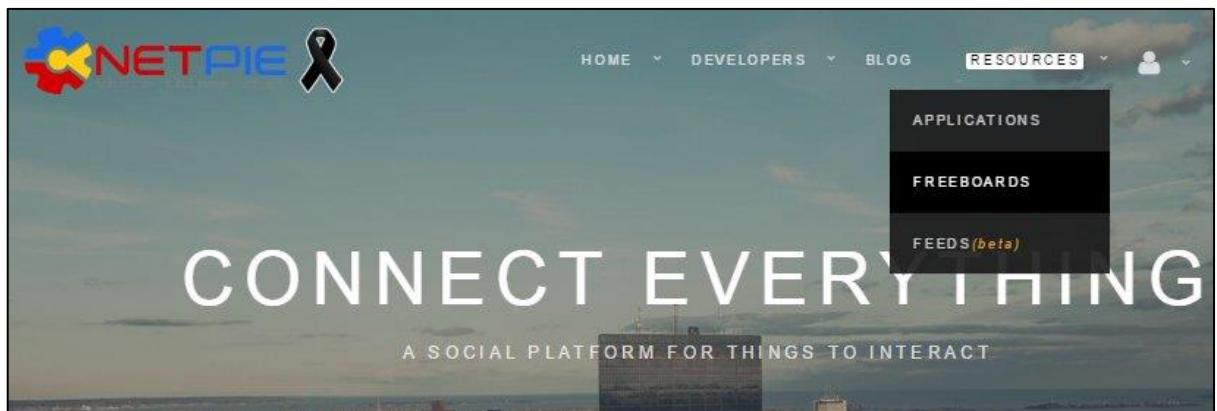
รูปที่ 5.1 Dashboard ของ NETPIE Freeboard

วิธีที่ 1 การใช้ NETPIE Freeboard ที่ติดตั้งในเครื่องแบบ Local

1. ดาวน์โหลดและติดตั้ง NETPIE Freeboard จาก Github (<https://github.com/netpieio/netpie-freeboard>) กดที่ปุ่ม Clone or download เพื่อเริ่มทำการดาวน์โหลดไฟล์แบบ Download ZIP
2. แตกไฟล์ใน ZIP Folder และให้ Browser เปิดไฟล์ที่ชื่อ index.html จะปรากฏหน้าสำหรับให้ใส่ข้อมูล
3. ปรับแต่งค่าใน NETPIE Freeboard ตามรายละเอียดที่จะอธิบายในลำดับถัดไป

วิธีที่ 2 การใช้ NETPIE Freeboard ผ่านทางหน้าเว็บ NETPIE

1. เข้าสู่ระบบ NETPIE Account และไปที่เมนู RESOURCES → FREEBOARDS



2. คลิกเครื่องหมาย + เพื่อสร้าง Freeboard ขึ้นมาใหม่
3. ตั้งชื่อ Freeboard และกดปุ่ม CREATE
4. ปรับแต่งค่าใน NETPIE Freeboard ตามรายละเอียดที่อธิบายในลำดับถัดไป

ตัวอย่างการปรับแต่งเบื้องต้น

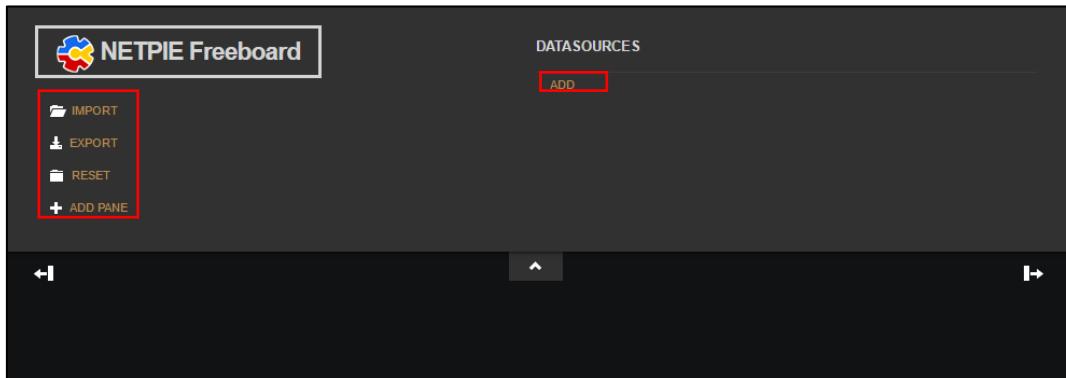
Main Menu

- IMPORT เป็นเมนูสำหรับอัพโหลดไฟล์ Configuration ของหน้า Freeboard ที่บันทึกเก็บไว้
- EXPORT เป็นเมนูสำหรับนำไฟล์ Configuration ออก (Export)
- RESET เป็นเมนูสำหรับล้าง Datasource และ Widget ที่สร้างไว้
- ADD PANE เป็นเมนูสำหรับเพิ่ม Panel ในการจัดวาง Widget

Datasources Menu

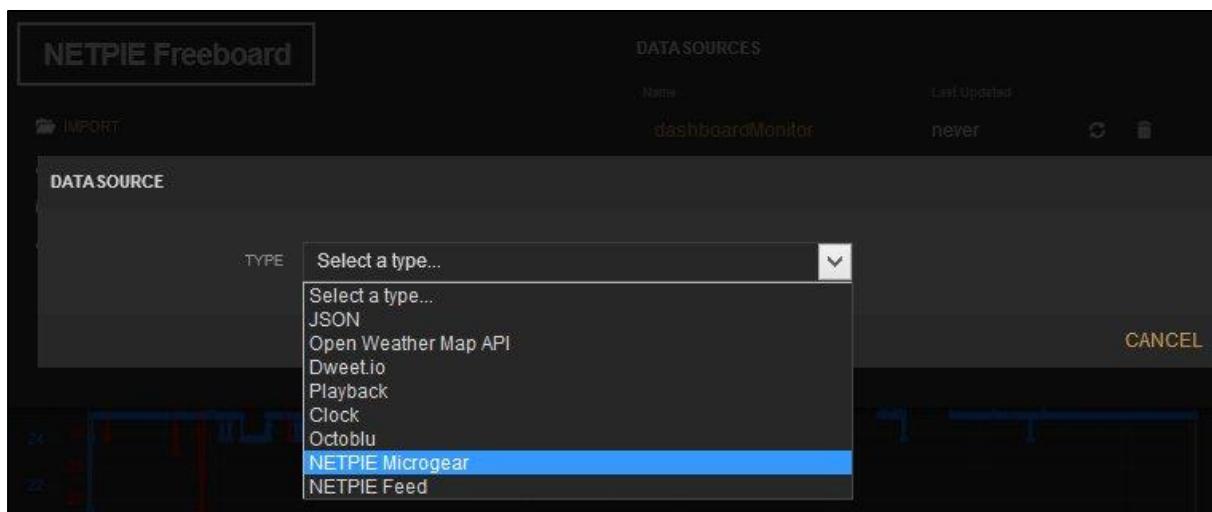


- ADD เป็นเมนูสำหรับเพิ่ม Datasource ที่เป็นแหล่งข้อมูลที่จะเข้ามายังต่อเพื่อดึงข้อมูลมาแสดง



รูปที่ 5.2 การเพิ่ม Datasource

1. ด้านล่าง DATA SOURCES คลิกที่ ADD จะปรากฏ Datasource Type ชนิดต่างๆ ให้เลือกเป็น NETPIE Microgear



รูปที่ 5.3 การเลือกประเภทของ Datasource

2. ใส่ข้อมูลสำหรับ Datasource ซึ่งประกอบด้วย

- NAME คือ ชื่อเรียก Datasource ที่ใช้ข้างใน (ไม่เกิน 16 ตัวอักษร) ในตัวอย่างในภาพ ด้านล่างคือ dashboardMonitor
- APP ID คือ App ID ที่ได้สร้างผ่านหน้าเดิบ <https://netpie.io/app> ในตัวอย่างในภาพ ด้านล่างคือ PieSmartHome
- KEY คือ Key ที่ได้จากการสร้าง App Key บนเว็บ NETPIE
- SECRET คือ Secret ของ Key บนเว็บ NETPIE

- SUBSCRIBED TOPIC คือ Topic ที่ใช้สำหรับการรับส่งข้อมูลที่อยู่ภายใต้ APPID นั้นๆ กรณีที่เป็น/# มีความหมายว่า รับข้อความจากทุก Topic

เมื่อกรอกข้อมูลเสร็จกด SAVE

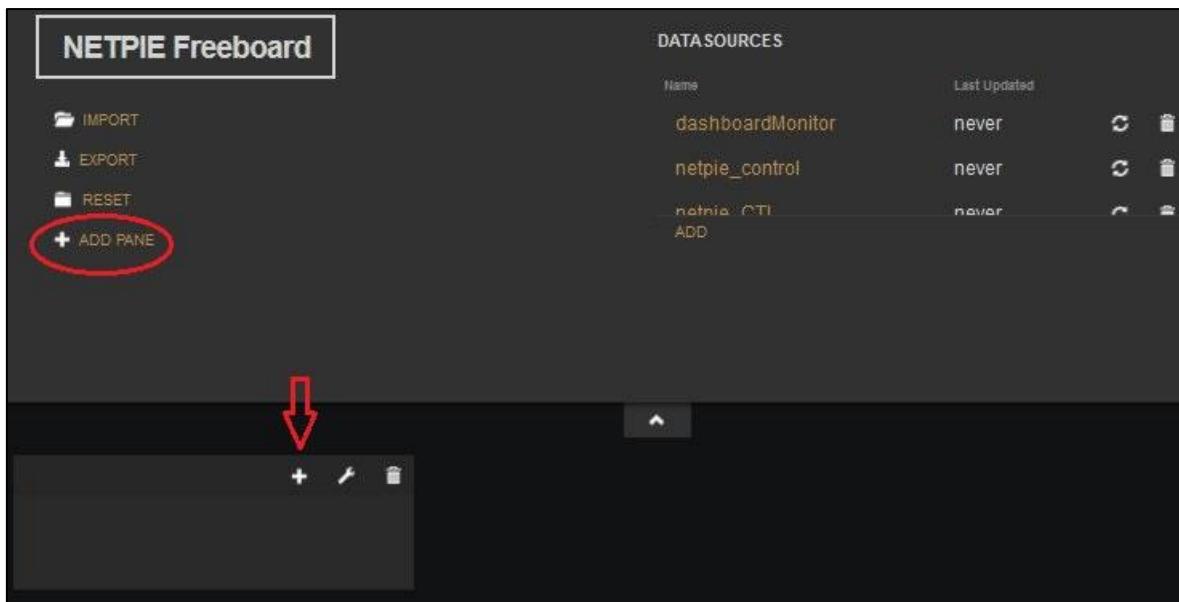
DATA SOURCE

Connect to NETPIE as a microgear to communicate real-time with other microgears in the same App ID. The microgear of this datasource is referenced by microgear[DATASOURCENAME]

TYPE	NETPIE Microgear
NAME	dashboardMonitor
APP ID	PieSmartHome
NETPIE App ID obtained from https://netpie.io/app	
KEY	vSM6154DYGXGFwec
Key	
SECRET	8e0a7c9990411a000000000000000000
Secret	
SUBSCRIBED TOPICS	#
Topics of the messages that this datasource will consume, the default is # which means all messages in this app ID.	
ONCONNECTED ACTION	
JS code to run after a microgear datasource is connected	
SAVE CANCEL	

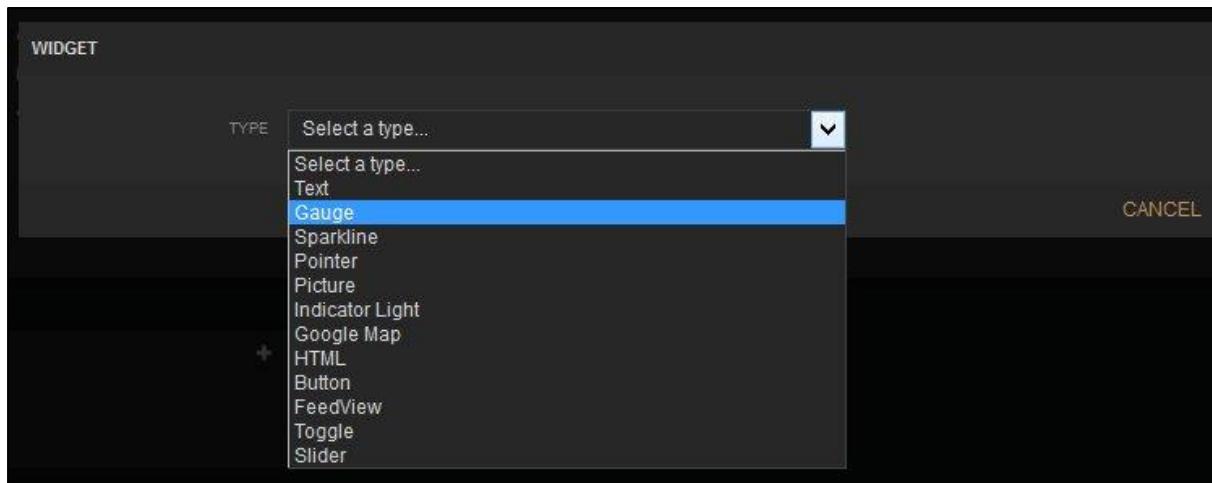
รูปที่ 5.4 การระบุข้อมูลของ Datasource

3. เพิ่ม Panel สำหรับสร้าง Widget ด้วยการคลิกที่ ADD PANE จะปรากฏ Panel เพิ่มขึ้นมาด้านล่าง



รูปที่ 5.5 การเพิ่ม Panel สำหรับสร้าง Widget

4. เพิ่ม Widget บน Panel ที่สร้างขึ้นใหม่โดย คลิกที่เครื่องหมาย + และเลือกชนิดของ Widget เช่น Gauge ตามตัวอย่างในรูป



รูปที่ 5.6 การเลือกประเภทของ Widget

5. กรอกข้อมูลง่ายๆแล้วกด Save

TITLE ตั้งชื่อให้ Widget นี้

VALUE ให้คลิกที่ + Datasource จะขึ้นรายชื่อของ Datasource ที่ได้สร้างไว้ และสามารถเลือกชื่อที่มีอยู่ และพิมพ์เพิ่มเติมตาม Format เป็นดังนี้

```
datasources["YourDataSourceName"]||"YourAppID/YourSubscribeTopic"]
```

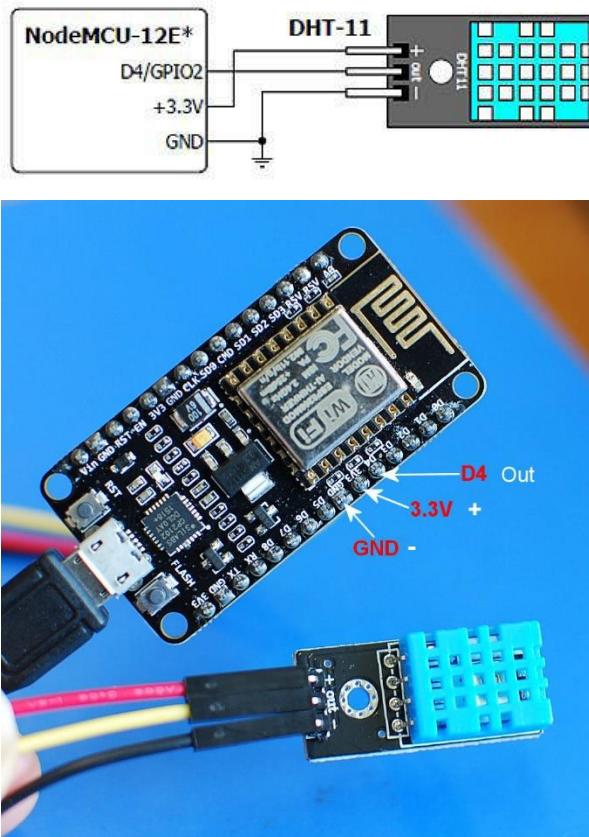
UNITS สามารถใส่หน่วยวัดที่ต้องการ หรือเว้นว่างไว้ได้

ขั้นตอนทั้งหมดคือ การสร้างโครงข่ายมาตรวัดหรือ Gauge เป็นต้น ที่ยังไม่ได้แสดงผล เนื่องจากยังไม่มีการระบุให้เชื่อมต่อกับอุปกรณ์หรือเซนเซอร์เพื่อรับข้อมูลเข้ามาเป็น Datasource ซึ่งจะเจาะลึกในรายละเอียดใน Lab 5.1 ต่อไป

Lab 5.1: การแสดงผลข้อมูลจากอุปกรณ์เซนเซอร์บน NETPIE Freeboard

Lab 5.1 เป็นการสร้างแอปพลิเคชันเซนเซอร์วัดอุณหภูมิและความชื้นด้วย NodeMCU และเซนเซอร์โมดูล DHT11 (อุปกรณ์ ZX-DHT11 ใน IoT Kit) ในขั้นแรกผู้ใช้ต้องติดตั้ง Library ที่เกี่ยวข้องกับเซนเซอร์ ก่อน ซึ่งในที่นี้คือ DHT.h ซึ่งสามารถดาวน์โหลดได้ที่ <https://github.com/adafruit/DHT-sensor-library> ให้ผู้ใช้งานคัดลอกไปวางไว้ใน Folder Arduino\libraries

ในภาพด้านล่างแสดงการเชื่อมต่อ เซนเซอร์ DHT11 กับ NodeMCU



รูปที่ 5.7 วิธีการเชื่อมต่อ DHT-11 เข้ากับ NodeMCU

หลังจากเชื่อมต่อเสร็จ ให้ทำการขั้นตอนดังนี้

1. ผัง NodeMCU สามารถเขียนโค้ดสร้างไฟล์ piedht.ino ตามข้างล่างและ Upload ไฟล์เข้า NodeMCU

piedht.ino

```
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <MicroGear.h>

const char* ssid      = "SSID";
const char* password = "PASSWORD";

#define APPID    "YOUR_APPID"
#define KEY      "YOUR_KEY"
#define SECRET   "YOUR_SECRET"

#define ALIAS    "piedht"

WiFiClient client;

int timer = 0;
char str[32];

#define DHTTYPE DHT11           // Define sensor type
#define DHTPIN D4                // Define sensor pin
DHT dht(DHTPIN, DHTTYPE, 15); //Initialize DHT sensor

int humid;
int temp;

MicroGear microgear(client);

void onMsgHandler(char *topic, uint8_t* msg, unsigned int msglen) {
    Serial.print("Incoming message -->");
    msg[msglen] = '\0';
    Serial.println((char *)msg);
}

void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
```

```

microgear.setAlias(ALIAS);

}

void setup() {
dht.begin();

microgear.on(MESSAGE,onMsgHandler);
microgear.on(CONNECTED,onConnected);

Serial.begin(115200);
Serial.println("Starting...");

if (WiFi.begin(ssid, password)) {
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
}

Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

microgear.init(KEY,SECRET,ALIAS);
microgear.connect(APPID);
}

void loop(){
if (microgear.connected()) {
    Serial.println("connected");
    microgear.loop();

    if (timer >= 1000) {
        humid = dht.readHumidity();
        temp = dht.readTemperature();
        sprintf(str,"%d,%d",humid,temp);
        Serial.println(str);

        Serial.print("Sending -->");
        microgear.publish("/dht",str);
    }
}
}

```

```

        timer = 0;
    }

else timer += 100;

}

else {

    Serial.println("connection lost, reconnect...");

    if (timer >= 5000) {

        microgear.connect(APPID);

        timer = 0;
    }

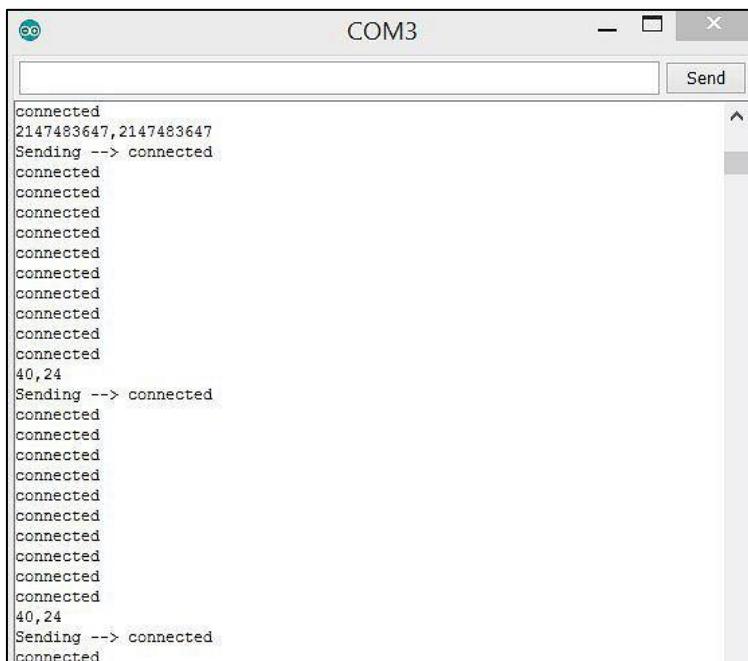
    else timer += 100;
}

delay(100);
}

```

ในตัวอย่างนี้ คำสั่ง microgear.publish("/dht",str) คือการ Publish ข้อมูล (ในที่นี้เป็น Message String str) ไปยัง Topic ที่ระบุคือ /dht ซึ่งข้อมูลนี้คือค่าของอุณหภูมิและความชื้นที่วัดได้จากเซนเซอร์ DHT

2. เปิด Console โดยไปที่ Tools ->Serial Monitor เพื่อตรวจสอบว่าเซนเซอร์ทำงานได้ปกติหรือไม่



รูปที่ 5.8 คอนโซลตรวจสอบการทำงานของเซนเซอร์

3. ผ่าน NETPIE Freeboard คลิกที่ Datasource ที่สร้างขึ้นก่อนหน้านี้ และแก้ไขในช่อง SUBSCRIBED TOPICS ให้เป็น /dht

4. กด + (ADD PANE) เพื่อสร้าง Widget ชนิด Gauge ใหม่ 2 Widget เพื่อแยกแสดงคุณภาพและความชื้น โดยกรอกตั้งค่าแต่ละ Widget ดังนี้

Widget 1:

- TITLE : Humidity
- VALUE : datasources["YourDatasourceName"]["/YourAppID/dht"].split(",")[0]
- UNIT : %
- MINIMUM : 0
- MAXIMUM : 100

Widget 2:

- TITLE : Temperature
- VALUE : datasources["YourDatasourceName"]["/YourAppID/dht"].split(",")[1]
- UNIT : C
- MINIMUM : 0
- MAXIMUM : 50

ภาพด้านล่างแสดงตัวอย่างการกรอกข้อมูลเพื่อสร้าง Widget และแสดงค่าความชื้น (Humidity)

WIDGET

TYPE
Gauge

TITLE
Humidity

VALUE
`datasources["dashboardMonitor"]["/PieSmartHome/dht"].split(" ")[0]`
+ DATASOURCE ✎ JS EDITOR

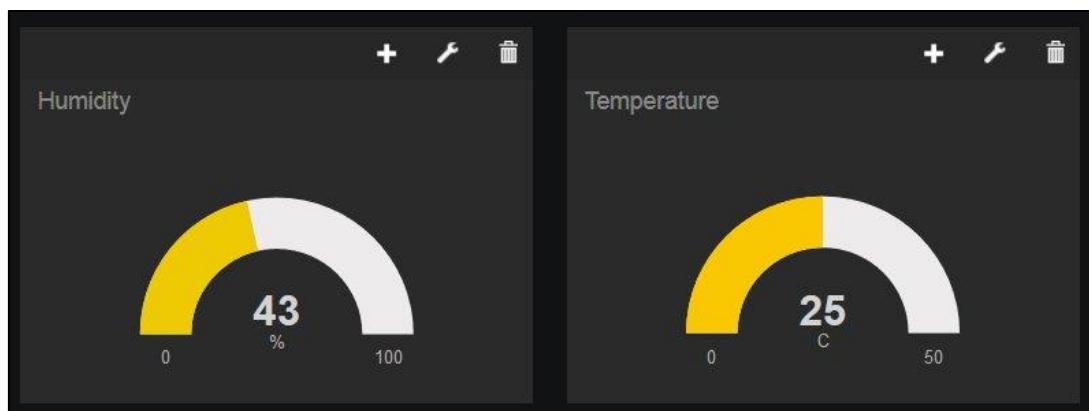
UNITS
%

MINIMUM
0

MAXIMUM
100

รูปที่ 5.9 ตัวอย่างหน้าจอดังค่า Widget แสดงค่าความชื้น

Widget แสดงผลอุณหภูมิและความชื้นที่สร้างขึ้นจะมีลักษณะตามแสดงในรูป



รูปที่ 5.10 Widget แสดงค่าอุณหภูมิและความชื้นที่วัดได้จากเซนเซอร์

คำอธิบายเพิ่มเติม

เนื่องจาก DHT ส่งค่ามาในรูปแบบ "Humidity, Temperature" เช่น "40,24" เวลาเจริบค่าเข้ามา จึงต้องทำการแยกออกเป็น Array โดยใช้เครื่องหมาย Comma "," เป็นตัวแบ่ง จากนั้นก็ถอดอิงถึงช่องใน Array ของ Message String ที่ Publish ใน Topic /dht เช่น Index [0] หมายถึงค่าแรก และ Index [1] หมายถึงค่าถัดมา

Lab 5.2: การควบคุมอุปกรณ์ด้วย NETPIE Freeboard

Lab 5.2 แสดงการประยุกต์ NETPIE Freeboard ในการควบคุมอุปกรณ์ โดยในเบื้องต้นนี้ เราจะควบคุมไฟ LED บนบอร์ด NodeMCU ซึ่งใช้หลักการทำงานบนพื้นฐานของการ Subscribe ข้อมูลจาก Topic หรือหัวข้อที่ระบุ และการกำหนดตัวรับของควบคุมทั้งในส่วนของ Datasource และส่วนของ Widget ที่ใช้ควบคุม โดยมีขั้นตอนดังต่อไปนี้

1. แก้ไขไฟล์ pieled2.ino โดยระบุข้อมูลการเข้าถึงเครือข่าย Wifi ข้อมูล APPID, KEY และ SECRET ตามโต๊ดข้างล่าง และทำการ Upload ไฟล์เข้า NodeMCU ให้เชื่อมต่อกับ NETPIE

pieled2.ino

```
#include <ESP8266WiFi.h>
#include <MicroGear.h>

const char* ssid      = "SSID";
const char* password = "PASSWORD";

#define APPID    "YOUR_APPID"
#define KEY     "YOUR_KEY"
#define SECRET  "YOUR_SECRET"

#define ALIAS   "pieled"

WiFiClient client;

char state = 0;
char stateOutdated = 0;
```



```

char buff[16];

MicroGear microgear(client);

void sendState(){
    if (state==0)
        microgear.publish("/pieled/state","0");
    else
        microgear.publish("/pieled/state","1");
    Serial.println("send state..");
    stateOutdated = 0;
}

void updateIO(){
    if (state >= 1) {
        digitalWrite(LED_BUILTIN, LOW);
    }
    else {
        state = 0;
        digitalWrite(LED_BUILTIN, HIGH);
    }
}

void onMsgHandler(char *topic, uint8_t* msg, unsigned int msglen) {
    char m = *(char *)msg;

    Serial.print("Incoming message -->");
    msg[msglen] = '\0';
    Serial.println((char *)msg);

    if (m == '0' || m == '1') {
        state = m=='0'?0:1;
        updateIO();
    }
    if (m == '0' || m == '1' || m == '?') stateOutdated = 1;
}

void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
    Microgear.setAlias(ALIAS);
}

```

```

stateOutdated = 1;
}

void setup(){
    Serial.begin(115200);
    Serial.println("Starting...");

    pinMode(LED_BUILTIN, OUTPUT);

    if (WiFi.begin(ssid, password)) {
        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
        }
    }

    microgear.on(MESSAGE,onMsghandler);
    microgear.on(CONNECTED,onConnected);
    microgear.init(KEY,SECRET,ALIAS);
    microgear.connect(APPID);
}

void loop(){
    if (microgear.connected()) {
        if (stateOutdated) sendState();
        microgear.loop();
    }
    else {
        Serial.println("connection lost, reconnect...");
        microgear.connect(APPID);
    }
}

```

2. ในหน้า NETPIE Freeboard คลิกเพิ่ม Datasource ที่สร้างขึ้นก่อนหน้านี้เพื่อแก้ไข ตั้งชื่อ Datasource ใส่ค่า KEY และ SECRET และในช่อง SUBSCRIBED TOPICS ให้ใส่ /pieler/state หรือ Topic ที่ท่านจะบุ้าร์สำหรับการ publish ในไฟล์ pieled2.ino ตั้งแสดงในภาพด้านล่าง และกด Save

DATASOURCE

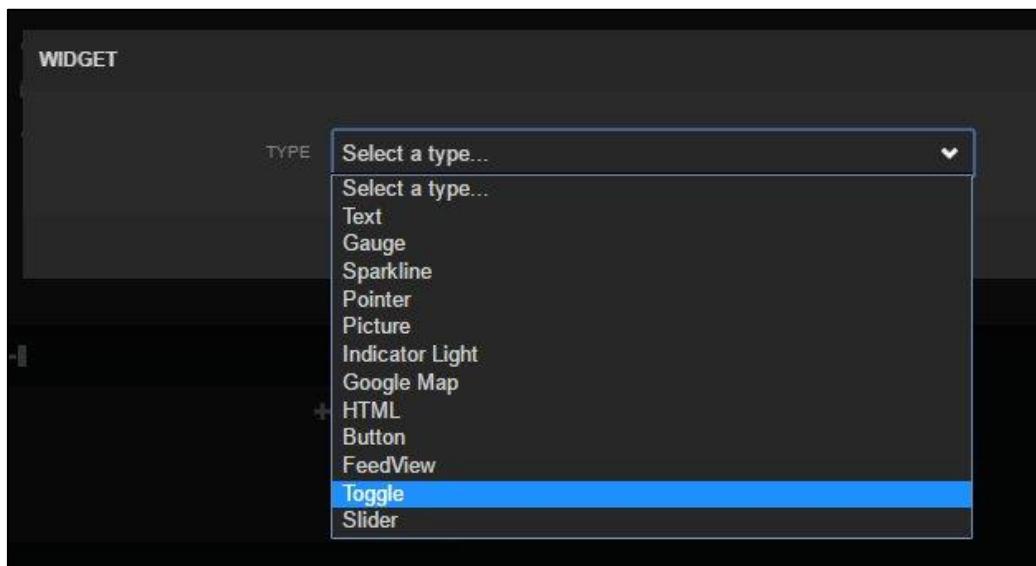
Connect to NETPIE as a microgear to communicate real-time with other microgears in the same App ID. The microgear of this datasource is referenced by microgear[DATASOURCENAME]

TYPE	NETPIE Microgear
NAME	netpie_control
APP ID	PieSmartHome
NETPIE App ID obtained from https://netpie.io/app	
KEY	com.netpie.PieSmartHome
Key	
SECRET	00000000000000000000000000000000
Secret	
SUBSCRIBED TOPICS	/pieled/state
Topics of the messages that this datasource will consume, the default is /# which means all messages in this app ID.	
ONCONNECTED ACTION	
JS code to run after a microgear datasource is connected	
SAVE CANCEL	

รูปที่ 5.11 หน้าต่างตั้งค่า Datasource ใน NETPIE Freeboard สำหรับควบคุม LED

3. สร้าง Widget ขึ้นมาใหม่โดยกด + (ADD PANE) และเลือกชนิดใน Drop Down Box เป็นแบบ

Toggle

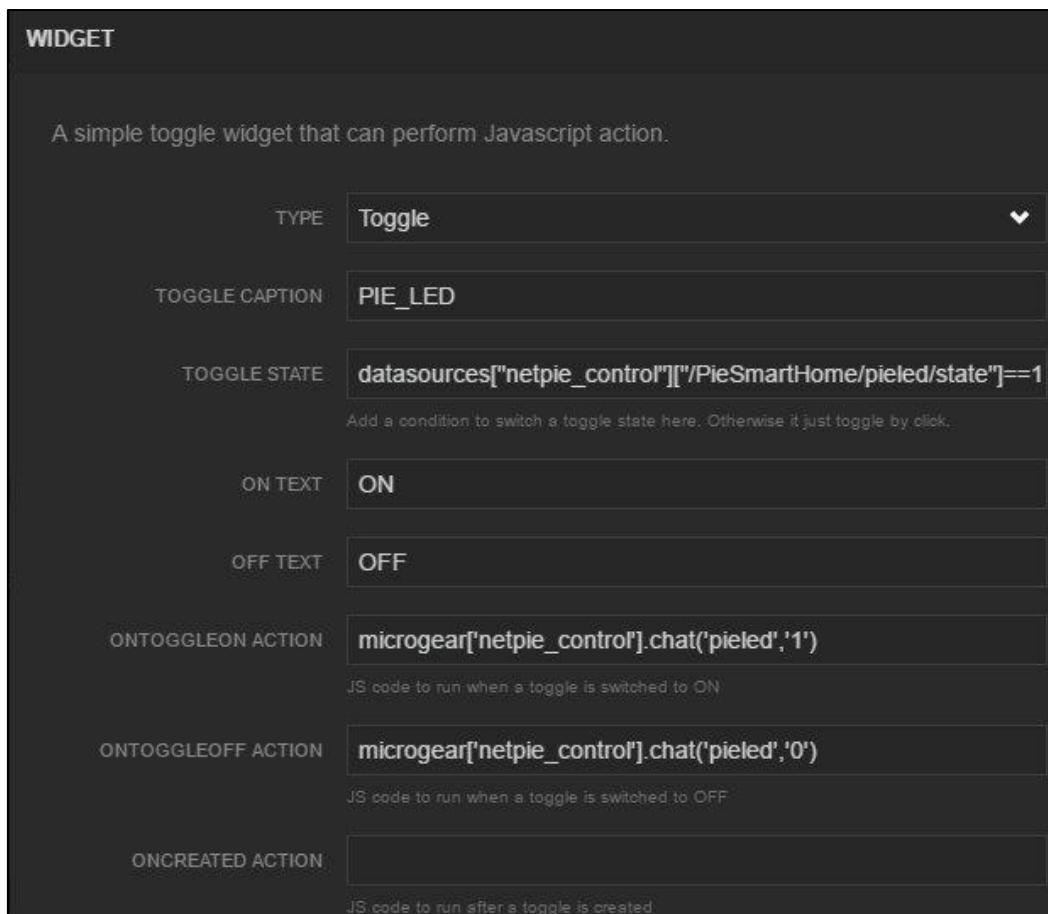


รูปที่ 5.12 หน้าจอเลือกชนิด Widget ให้เป็นแบบ Toggle

จากนั้นตั้งค่า Widget ดังนี้โดยหน้าจอการตั้งค่าแสดงดังรูป

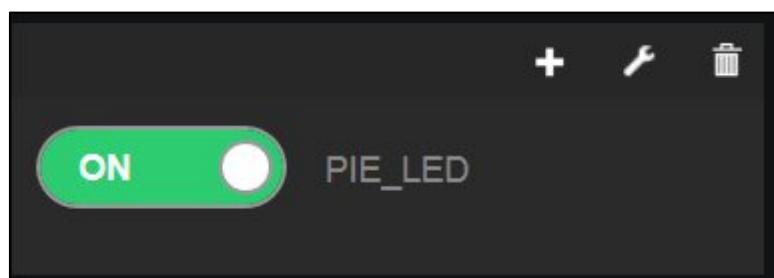
- TOGGLE CAPTION : ตั้งชื่อปุ่ม Toggle (ในตัวอย่างตั้งเป็น PIE_LED)

- TOGGLE STATE : ได้ข้อมูลตามชื่อของ Datasource และ Topic เช่น
`datasources["netpie_control"]["/PieSmartHome/pieled/state"]==1`
- ON TEXT : ON
- OFF TEXT : OFF
- ONTOGGLEON ACTION : `microgear['netpie_control'].chat('pieled','1')`
- ONTOGGLEOFF ACTION : `microgear['netpie_control'].chat('pieled','0')`



รูปที่ 5.13 หน้าจอการตั้งค่า Widget ชนิด Toggle

แล้วกด Save จะได้ Widget ที่มีปุ่มควบคุมดังแสดงในภาพ เพื่อทดสอบเปิดปิด LED บน NodeMCU



รูปที่ 5.14 Widget ใช้เปิดปิด LED บน NodeMCU

คำอธิบายเพิ่มเติม

TOGGLE STATE เป็นสถานะ On/Off ซึ่งสามารถผูกกับตระรากของ Datasource ในที่นี่เราตั้งค่าให้ Toggle เปลี่ยนสถานะตามค่าที่ส่งมาใน Topic ชื่อ /pieled/state

ONTOGGLEON และ ONTOGGLEOFF เป็นคำสั่งที่จะถูกเรียก เมื่อ Toggle เปลี่ยนสถานะไปเป็น ON และ OFF ตามลำดับ

ในหน้า Datasources ตรงช่อง SUBSCRIBED TOPICS นั้น นอกจากจะสามารถระบุค่าแบบเจาะจงเป็น topic /pieled/state แล้ว ยังสามารถระบุค่าเป็น /pieled/+ ก็ได้ โดยใช้เครื่องหมาย (+) ซึ่งเป็น Single-Level Wildcard เพื่อรับค่าของ State



เราสามารถใช้ Wildcard เพื่อช่วยในการ Subscribe Topic ต่างๆ เช่น หากต้องการ Subscribe Topic ตามที่ระบุแบบเจาะจงดังนี้: "/home/kitchen/temp", "/home/bedroom/temp", และ "/home/livingroom/temp" ก็สามารถยุบเหลือ 1 Topic คือ "/home/+temp"

นอกจาก + และยังใช้เครื่องหมาย # ได้ด้วย โดยที่เครื่องหมาย + จะแทนคำว่าไว้ก็ได้ ระดับชั้นเดียว ส่วน # จะแทนคำว่าไว้ก็ได้ในระดับชั้นกว่าต่อไปเท่าไหร่ก็ได้ เช่นจะให้ match 3 Topic ข้างต้น ก็อาจจะเขียนระบุเป็น Topic คือ "/home/#"

6. NETPIE FEED

ข้อมูลที่เกิดจากการอ่านค่าของอุปกรณ์เซนเซอร์ในบทที่แล้ว สามารถนำมาใช้ให้เกิดประโยชน์ได้อย่างหลากหลาย เช่น ใช้สำหรับการตรวจสอบ (Monitoring) หรือการแสดงผลของข้อมูล (Data Visualization) โดยเฉพาะอย่างยิ่งข้อมูลแบบทันเวลา (Real-time Data) ซึ่งจะต้องเก็บรวบรวมด้วยอัตราความถี่ที่เหมาะสมเพื่อให้การตรวจสอบหรือการแสดงผลตามความต้องการ ในปัจจุบันแพลตฟอร์ม NETPIE มีบริการที่สามารถเก็บข้อมูลและแสดงผลที่เรียกว่า Feed ซึ่งทำหน้าที่เสมือนถังเก็บข้อมูลประเภท Time-Series กล่าวคือ เป็นชุดข้อมูลหรือค่าตัวแปร ณ เวลาต่างๆ เช่น อุณหภูมิอากาศในช่วงเวลาต่างๆ ของวัน เป็นต้น ข้อมูลเหล่านี้จะถูกเก็บแบบต่อเนื่องสะสมกันไปตลอด และสามารถเรียกอุปกรณ์ในช่วงเวลาใดก็ได้ ในบทนี้จะอธิบายรายละเอียดการใช้งาน NETPIE Feed ตั้งแต่การตั้งค่าการเก็บข้อมูลจาก Datasource ไปจนถึงการแสดงผล

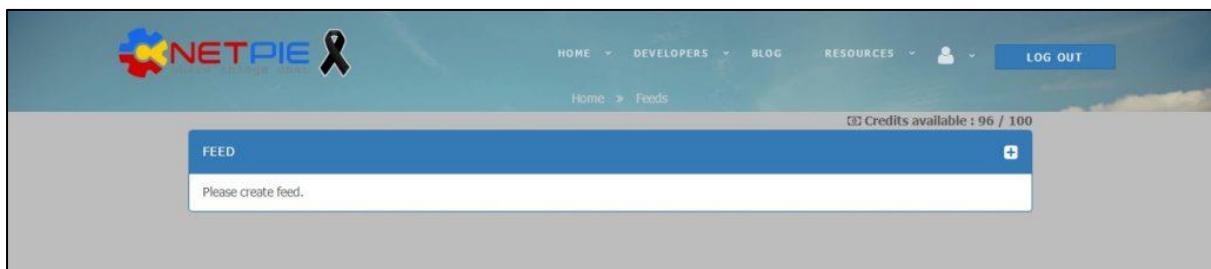
6.1 การสร้าง FEED

1. เลือก FEEDS จากเมนู RESOURCES



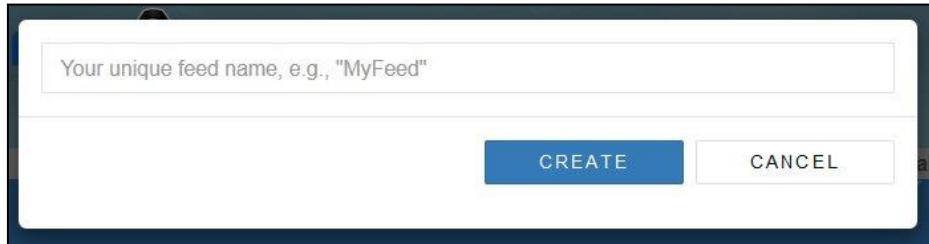
รูปที่ 6.1 การเข้าใช้งาน Feed จากเมนู Resources

2. สร้าง Feed ใหม่ หรือเพิ่ม Feed ด้วยการคลิกที่เครื่องหมาย + ที่มุมบนขวาของ Feed



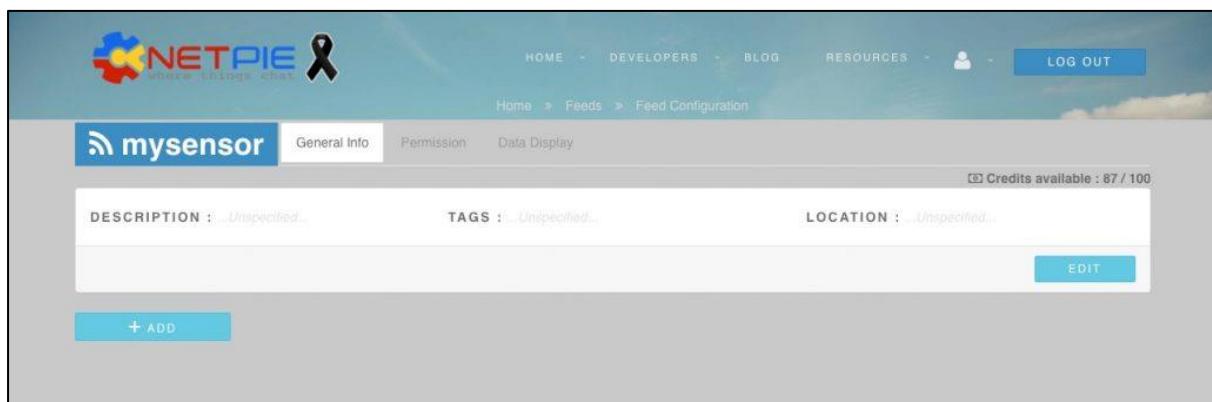
รูปที่ 6.2 การสร้างหรือเพิ่ม Feed ใหม่

3. ตั้งชื่อ Feed โดยที่ชื่อนี้จะต้องไม่ซ้ำกับที่เคยมีมาก่อน และห้ามซ้ำกับของผู้ใช้อื่น เมื่อตั้งชื่อแล้วให้คลิก CREATE เพื่อสร้าง Feed



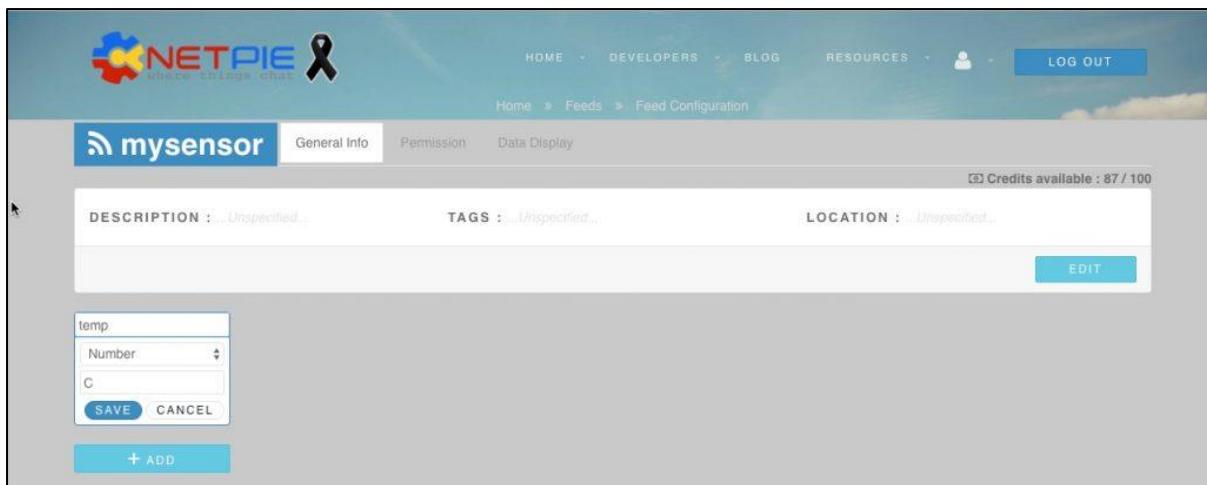
รูปที่ 6.3 การตั้งชื่อ Feed

4. เมื่อสร้าง Feed สำเร็จ จะเข้าสู่หน้าตั้งค่า Feed ตามภาพข้างล่าง ในตัวอย่างนี้ ได้สร้างสร้าง Feed ที่มีชื่อว่า “mysensor” ซึ่งจะเป็นรากฐานในหน้ารวม Feed เมื่อเราเข้ามาในครั้งต่อไปเราสามารถกลับมาที่หน้าตั้งค่าได้อีกจากหน้ารวมของ Feed โดยคลิกที่เครื่องหมายรูปประแจด้านขวาเมื่อหลังชื่อ Feed ที่ต้องการ



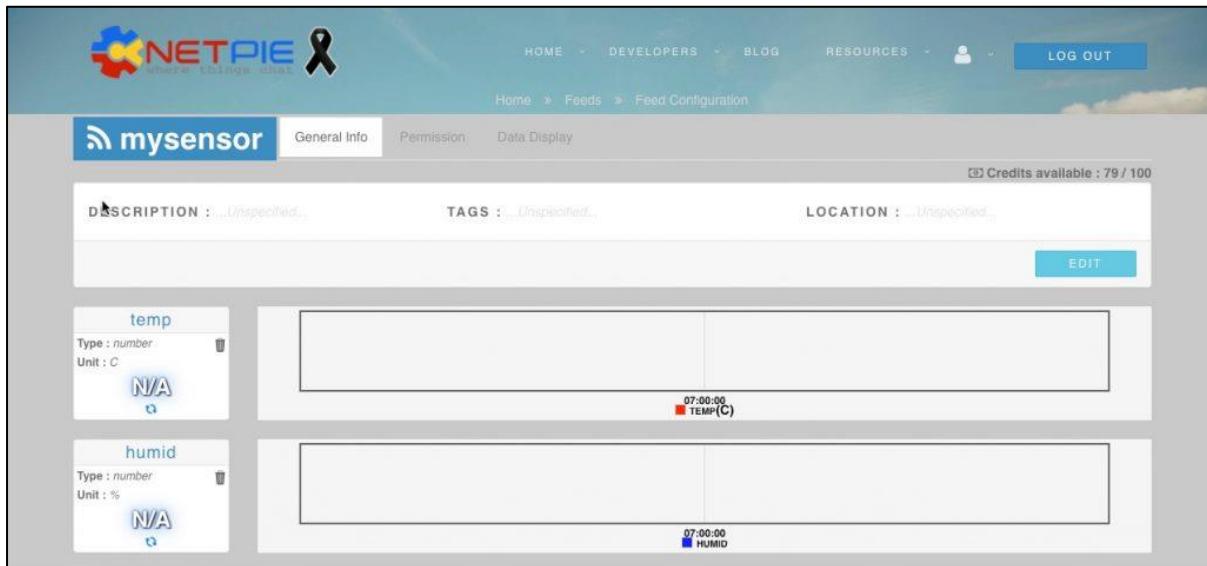
รูปที่ 6.4 หน้าตั้งค่า Feed

5. เนื่องจาก Feed ที่สร้างขึ้นใหม่นี้ยังไม่สามารถรับค่าใดๆ เข้ามาได้ จึงต้องสร้าง Field ขึ้นมารับข้อมูล ให้กดปุ่ม + ADD จะปรากฏหน้าต่างให้ตั้งค่าต่างๆ ในตัวอย่างนี้ เป็นการสร้าง Field จำนวน 2 Field ที่มีชื่อว่า temp และ humid (หรือจะตั้งเป็นชื่ออะไรก็ได้) หากมีหน่วยกระบุไว้เพื่อการแสดงผลต่อไป โดย ณ ขณะนี้ยังรองรับเฉพาะข้อมูลชนิดตัวเลข (Number) เท่านั้น โดยในอนาคตมีการพัฒนาเพิ่มเติมให้รองรับข้อมูลชนิดอื่นๆ



รูปที่ 6.5 การสร้าง Field เพื่อรับข้อมูล Feed

6. คลิก SAVE จะได้ Feed ที่พร้อมรับค่ารายละเอียดการประมวลแต่งอยู่ในลำดับถัดไป



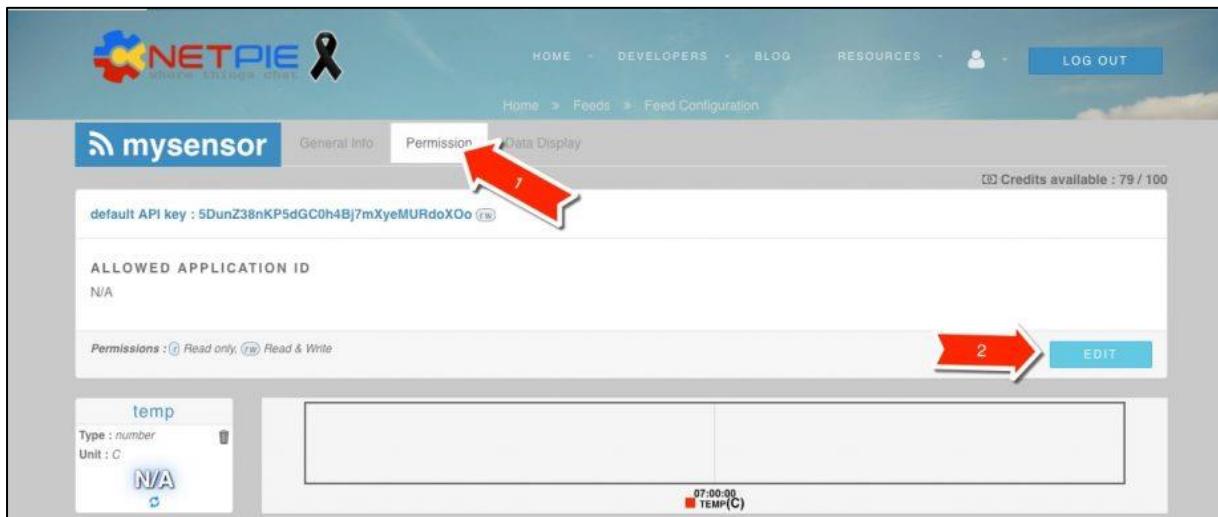
รูปที่ 6.6 ตัวอย่าง Feed ที่สร้างขึ้นด้วย 2 Field

6.2 การกำหนดสิทธิ์ในการเข้าถึง FEED

เนื่องจาก Feed แต่ละอันที่สร้างขึ้นมาแล้ว เป็น Resource อิสระที่ไม่ขึ้นกับ AppID ดังนั้นในการนำไปใช้งานจำเป็นต้องมีเรื่องของกำหนดสิทธิ์ การให้สิทธิ์กับอุปกรณ์ที่จะเข้ามาอ่านหรือเขียน Feed มีด้วยกัน 2 วิธี ได้แก่

- การใช้ API Key ที่ปรากฏอยู่ในแท็บ Permission วิธีนี้จะอนุญาตให้ Client ที่มี API key สามารถอ่านเขียน Feed นี้ได้ วิธีการนำ API key ไปใช้จะกล่าวถึงในต่อไป

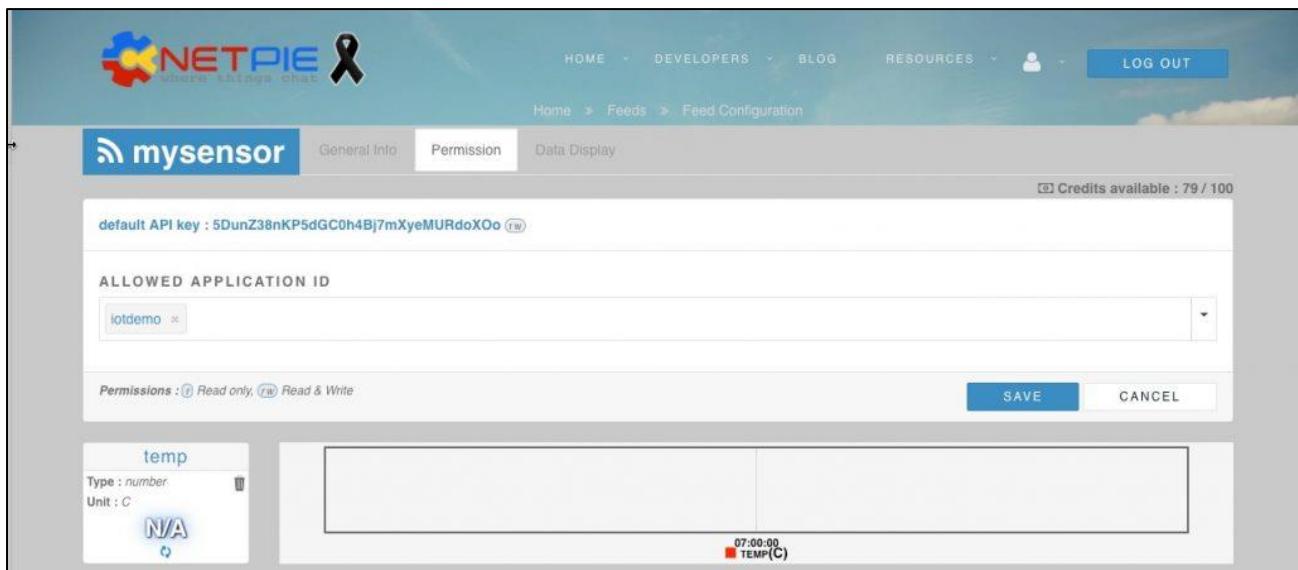
2. การให้สิทธิ์กับ AppID ใช้ได้กับเฉพาะ AppID ของเจ้าของ Feed เท่านั้นวิธีนี้ไว้เพื่อคำนึงถึงความสะดวกในการติดตามนาฬิกาของเจ้าของเป็นการให้สิทธิ์กันผ่านเครือข่าย จึงไม่ต้องเข้าไปแก้โค้ดโปรแกรมการให้สิทธิ์กับ AppID จะทำให้ทุกๆ คุณสามารถอ่านหรือเขียน Feed นี้ เมื่อ กับกันหมด การตั้งค่าทำได้โดยคลิกที่แท็บ Permission และคลิกที่ EDIT ตามภาพ



รูปที่ 6.7 การตั้งค่าสิทธิ์ในการเข้าถึง

จากนั้นพิมพ์ AppID ที่ต้องการให้สิทธิ์ หรือเลือกจากเมนู Drop-Down และคลิก SAVE ในตัวอย่างข้างล่าง เป็นการให้สิทธิ์กับ AppID ที่ชื่อ “iotdemo” ดังนั้นทุกๆ คุณสามารถอ่านหรือเขียน Feed ได้โดยอัตโนมัติ การอนุญาตแบบนี้จะเป็นการแยกสิทธิ์ในรูปแบบเดียวกับการนำ Default API key ของ Feed ไปใช้


 เราสามารถให้สิทธิ์กับ AppID ที่เราเป็นเจ้าของเท่านั้น



รูปที่ 6.8 การให้สิทธิ์การเข้าถึง Feed กับ AppID

6.3 การเขียนข้อมูลลงใน FEED

การเขียนข้อมูลเพื่อเก็บใน Feed มี 2 วิธี ดังนี้

วิธีที่ 1 ใช้ REST API (ข้อมูลและการใช้งานโดยละเอียดของ REST API อยู่ในบทที่ 7)

REST API สามารถเรียกผ่าน Command Line ได้โดยใช้โปรแกรม curl ซึ่งเป็น HTTP Client แบบ Command Line ที่นักพัฒนานิยมใช้กันมาก ในระบบปฏิบัติการ Mac OSX และ Linux โปรแกรมนี้จะถูกติดตั้งมาแบบพร้อมใช้งานอยู่แล้ว เพียงแค่เปิด Terminal ก็สามารถพิมพ์คำสั่งได้เลย แต่สำหรับ Windows อาจจะต้องติดตั้งโปรแกรมเพิ่ม โดยสามารถดาวน์โหลดได้จาก <https://curl.haxx.se/download.html>

การเขียน Feed ผ่าน REST API จะต้องกราทำโดยใช้ API Key ควบคู่กันไปด้วยเสมอ และมีรูปแบบคำสั่งที่เรียกผ่าน curl บน Terminal ดังนี้ (เครื่องหมาย \$ ไม่ต้องพิมพ์ลงไป ใส่ไว้เพื่อสื่อว่าเป็น Command Line)

```
$ curl -X PUT
"https://api.netpie.io/feed/<FEEDID>?apikey=<APIKEY>&data=<DATA>"
```

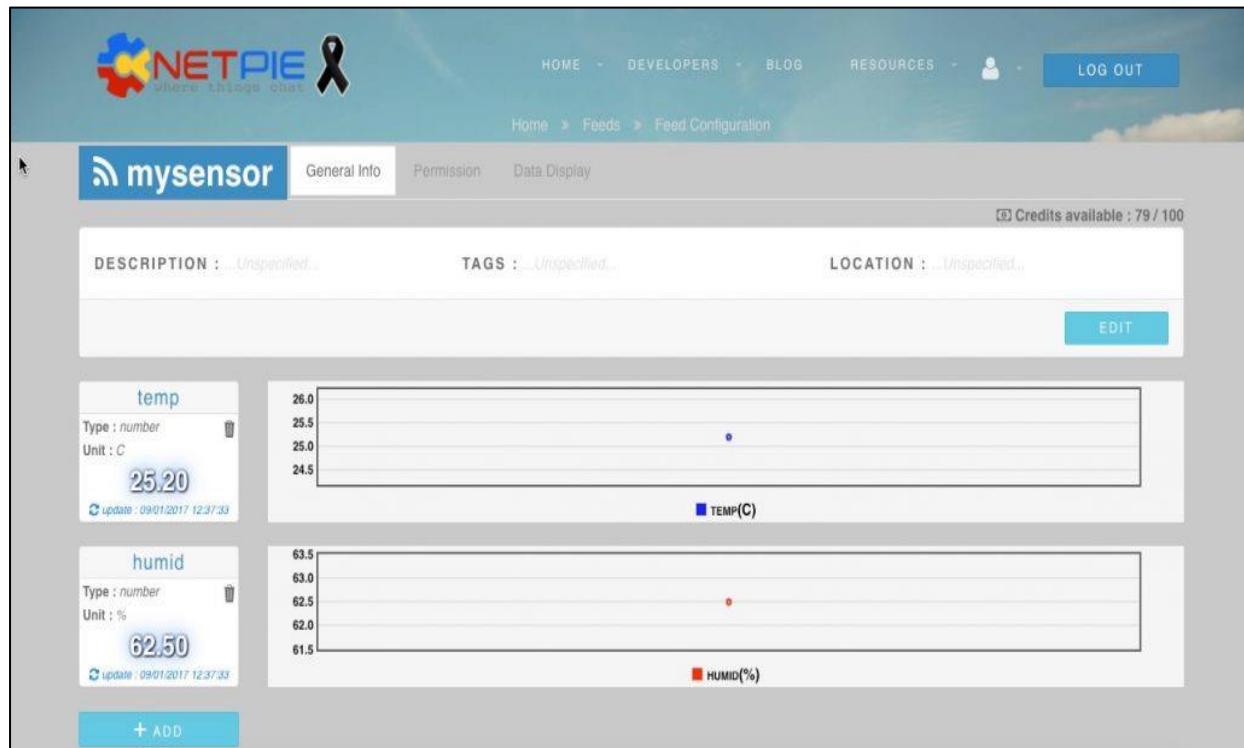
ตัวอย่างการส่งค่า temp = 25.2 และ humid = 62.5 ไปเขียนที่ Feed ชื่อ mysensor จะใช้คำสั่งดังนี้

```
$ curl -X PUT
"https://api.netpie.io/feed/mysensor?apikey=5DunZ38nKP5dGC0h4Bj7mXyeMURdoXOo&data=temp:25.2,humid:62.5"
```

เมื่อพิมพ์เสร็จให้กด Enter หากไม่มีข้อผิดพลาด จะมีข้อความขึ้นมาดังนี้

```
{"code":200,"message":"Update OK"}
```

เมื่อกลับไปดูที่หน้า Feed เราจะเห็นจุดข้อมูลปรากฏขึ้นมาแบบนี้ข้อที่ควรระวังคือ ชื่อ Field ข้อมูลในคำสั่ง (temp และ humid) จะต้องตรงกับชื่อ Field ที่ได้สร้างรอเราไว้บน Feed เพื่อให้ข้อมูลถูกจัดเก็บสำเร็จ



รูปที่ 6.9 การสร้างข้อมูลไปเก็บใน Feed ด้วย command curl

วิธีที่ 2 ใช้ Microgear Function

สำหรับคุณที่ต้อง NETPIE ผ่าน Microgear Library อยู่แล้ว สามารถส่งค่าเข้าไปใน Feed ผ่าน Real-time Message ได้โดยใช้ฟังก์ชัน microgear.writeFeed() ซึ่งจะมีใน Library ปัจจุบันทุกภาษาของ NETPIE อยู่แล้ว (หากไม่มีโปรดอัพเดทให้เป็นเวอร์ชันล่าสุด) การเขียน Feed ด้วย Microgear นี้ ในกรณีที่เจ้าได้ให้สิทธิกับ AppID ไว้แล้ว ก็ไม่จำเป็นต้องใส่ API Key ลงไปรูปแบบการใช้ฟังก์ชันมีดังนี้

```
microgear.writeFeed("<FEEDID>" , "<DATA>")
```

โดยที่ DATA เป็น String ที่มีลักษณะเดียวกับ REST API ดังตัวอย่างนี้

```
microgear.writeFeed("mysensor" , "temp:25.2,humid:62.5")
```

ส่วนของ DATA นอกจากรองรับ String รูปแบบดังกล่าวแล้ว เรายังสามารถส่ง String ที่แปลงมาจาก json ได้ด้วย เช่น "{\"temp\":25.2,\"humid\":62.5}" หรือหากเป็น Node.js หรือ Javascript เรายังสามารถส่งตัวแปรที่มี Type เป็น Object ได้โดยไม่ต้องแปลงเป็น String ก่อน

ในกรณีที่เรา yang ไม่ได้ให้สิทธิ์กับ AppID พังก์ชันข้างต้นจะได้รับข้อความ Error ว่าไม่มีสิทธิเขียน Feed จึงต้องใส่ API Key ลงไว้ในพังก์ชันด้วยซึ่งจะหมายความว่าต้องการให้มีเฉพาะคุปกรณ์เพียงบางตัวเท่านั้นที่สามารถเขียน Feed ได้ รูปแบบการเรียกจะเป็นตามนี้

```
microgear.writeFeed("<FEEDID>" , "<DATA>" , "<APIKEY>")
```

ซึ่งตัวอย่างการใช้จริงจะอยู่ในรูปแบบดังนี้

```
microgear.writeFeed("mysensor" , "temp:25.2,humid:62.5" , "5DunZ38nKP5dGC0h4Bj7mXyeMURdoXOo")
```

และทุกครั้งที่เราส่งค่าเข้าไปที่ Feed ไม่ว่าสำเร็จหรือไม่ จะมี Message ตอบกลับมาทาง Event Info หรือ Error เสมอ เราสามารถเช็คสถานะการเขียน Feed ได้จากการรับ Event info และ Error ซึ่งลักษณะการเรียกใช้จะแตกต่างกันเล็กน้อยตามรูปแบบภาษาของ Library ที่ใช้งาน รายละเอียดเพิ่มเติมสามารถศึกษาได้ในส่วนอธิบาย Events ของเอกสาร readme ใน Microgear Library แต่ละภาษา

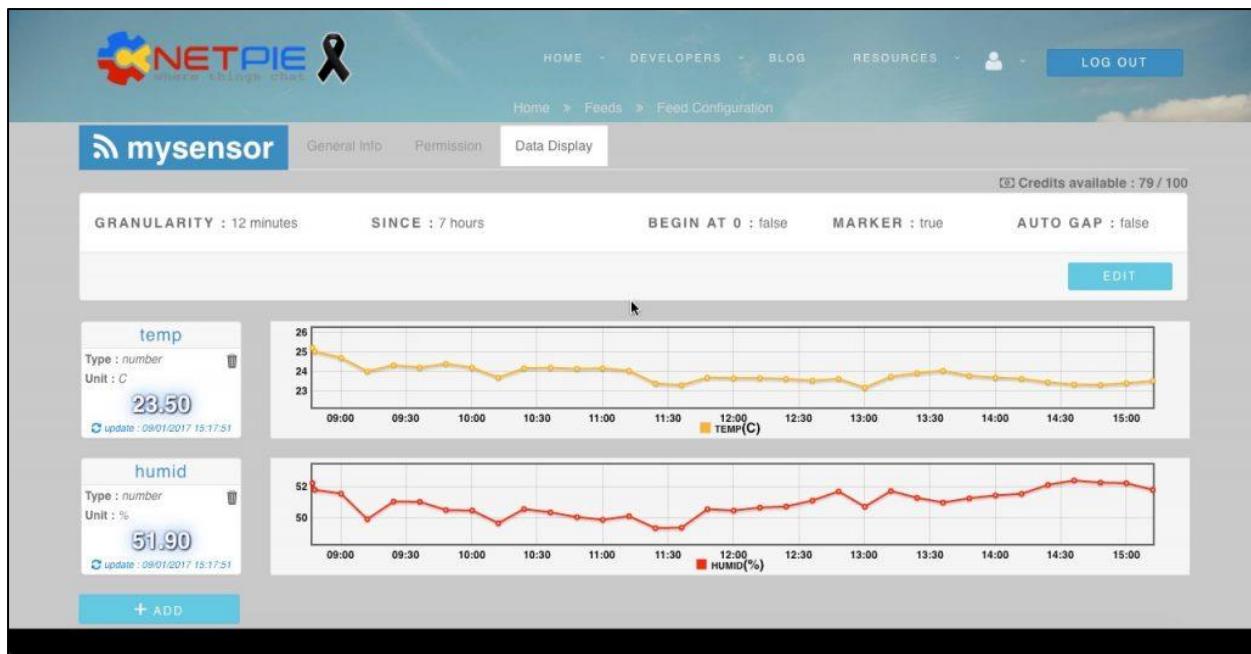
6.4 การดึงข้อมูลจาก FEED มาใช้งาน

NETPIE มีวิธีใช้งานข้อมูลจาก Feed อยู่ 3 แบบดังนี้

วิธีที่ 1 การแสดงผลผ่านหน้าการจัดการของ Feed

เราสามารถใช้หน้า Feed ของ NETPIE.io ในการดูค่าข้อมูลหลังได้ในลักษณะเส้นกราฟและสามารถตั้งค่าการแสดงผลได้จากแท็บ Data Display ดังนี้

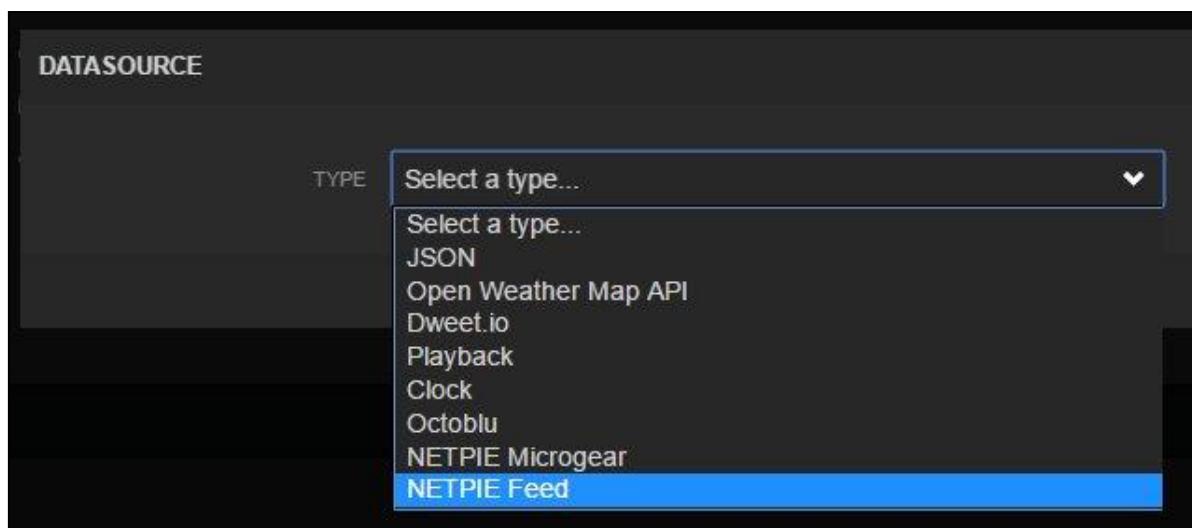
- GRANULARITY คือความละเอียดของจุดข้อมูล ตัวอย่างนี้ตั้งไว้ที่ 12 นาที หมายความว่าจุดข้อมูลต่างๆ ที่เก็บมาในช่วงเวลาหน้าต่างจะ 12 นาทีจะถูกนำมาเฉลี่ยเป็นจุดเดียว
- SINCE เป็นเวลาตั้งต้นย้อนหลังที่จะเรียกดูข้อมูล
- BEGIN AT 0 ตั้งเป็น TRUE จะเป็นการตั้งค่าแกน Y เริ่มต้นที่ 0
- MARKER แสดงวงกลมที่จุด
- AUTO GAP แทรกช่องว่างอัตโนมัติ (ในกรณีไม่มีข้อมูลในช่วงเวลาที่ตั้งไว้)



รูปที่ 6.10 หน้าจอแสดงผลเส้นกราฟของ Feed

วิธีที่ 2 การแสดงผลผ่านหน้า Freeboard

นอกจากดูค่าผ่านหน้าการจัดการ Feed แล้วเรายังสามารถดึงค่าของ Feed ไปแสดงผลบน NETPIE Freeboard ร่วมกับ Widget ต่างๆ ได้อีกด้วย วิธีการใช้งาน Freeboard สามารถรายละเอียดได้จากบทที่แล้ว โดยในบทนี้ เพื่อให้ Freeboard แสดง Feed ให้เลือกชนิด Datasource ของ Freeboard เป็น NETPIE Feed

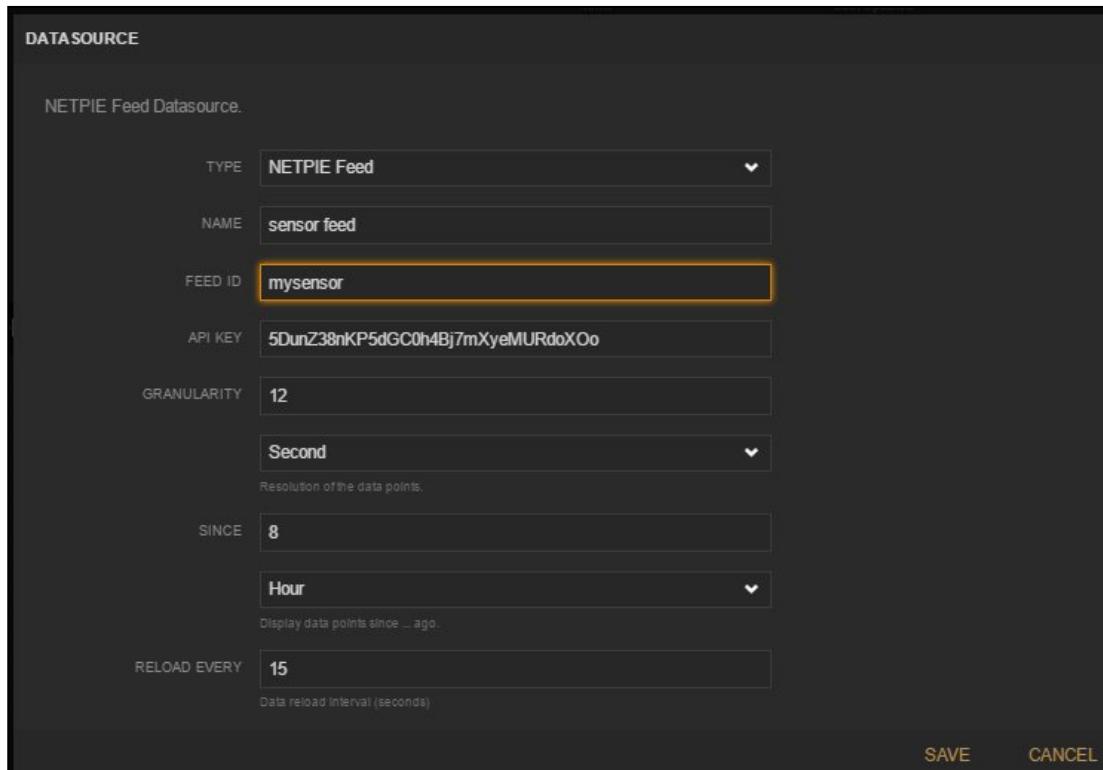


รูปที่ 6.11 การเลือกชนิดของDatasource เพื่อใช้งาน Feed

เมื่อเสร็จแล้วจะเข้าสู่หน้าตั้งค่า Datasource ให้กรอกข้อมูลดังนี้

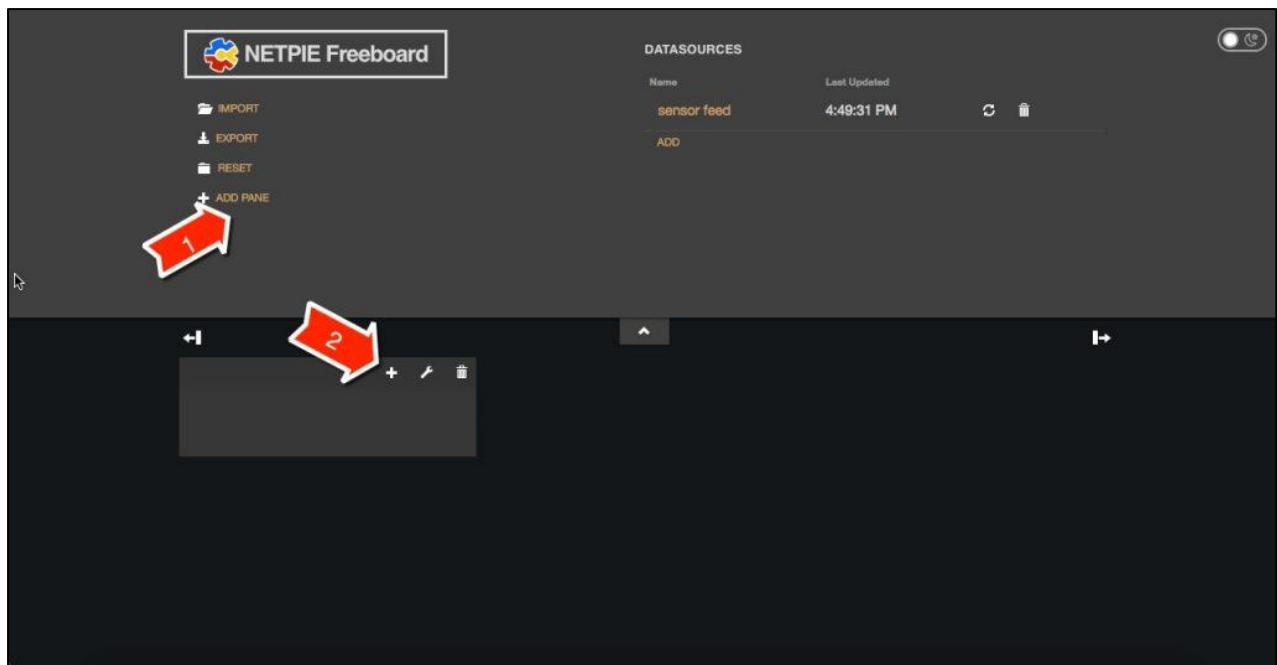
- NAME ตั้งชื่อเป็นอะไรได้ ในตัวอย่างนี้ ตั้งชื่อว่า “sensor feed”
- FEED ID ใส่ Feed ID ให้ตรงกับชื่อ Feed ที่สร้างไว้ตามตัวอย่างในบทที่ 5 คือ “mysensor”
- AKI KEY ใส่ Default API key ที่ได้จากหน้าตอนที่สร้าง Feed

ส่วนตัวเลือกที่เหลือสามารถตั้งได้ตามความเหมาะสม เมื่อเสร็จแล้วกดปุ่ม SAVE

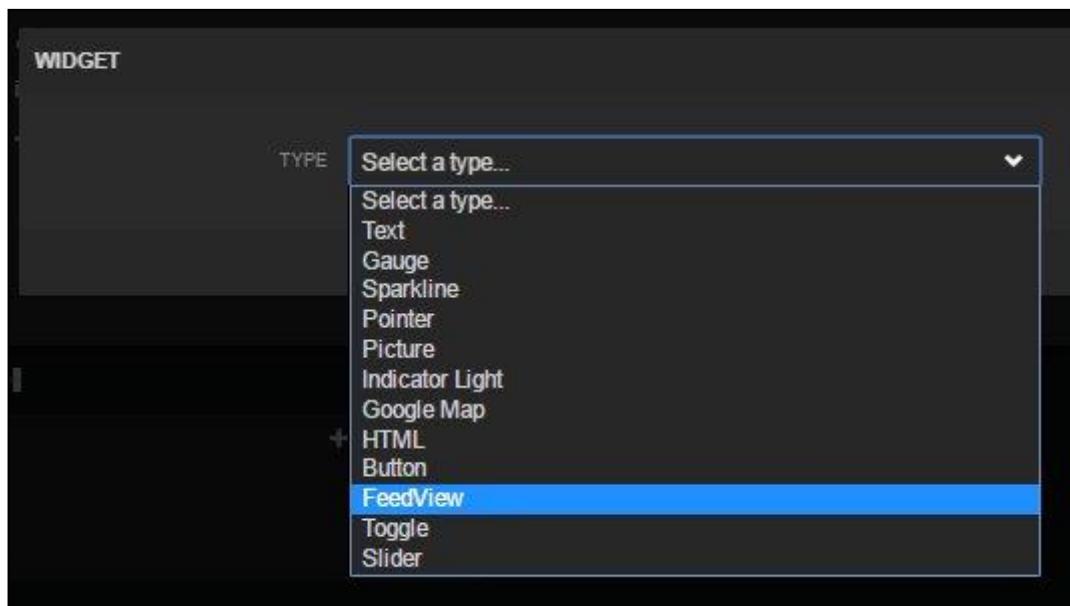


รูปที่ 6.12 การตั้งค่า Feed ใน Datasource

เมื่อตั้งค่า Datasource บน Freeboard สำหรับ Datasource จะเริ่มทำงาน ขั้นต่อไปคือการสร้าง Widget ที่ดึงข้อมูลจาก Datasource ของ Feed มาแสดงผล ซึ่ง NETPIE มี Widget ชื่อว่า Feed View ให้สำหรับใช้งานคู่กับ Feed Datasource การเพิ่ม Widget ใหม่ให้เลือกเพิ่มหน้าต่างก่อนด้วยการกดปุ่ม ADD PANE และให้กดปุ่ม + ที่หน้าต่างเพื่อเพิ่ม Widget และเลือกชนิด Feed View ดังแสดงในภาพ



รูปที่ 6.13 การสร้าง Widget เพื่อดูค่าของ Feed



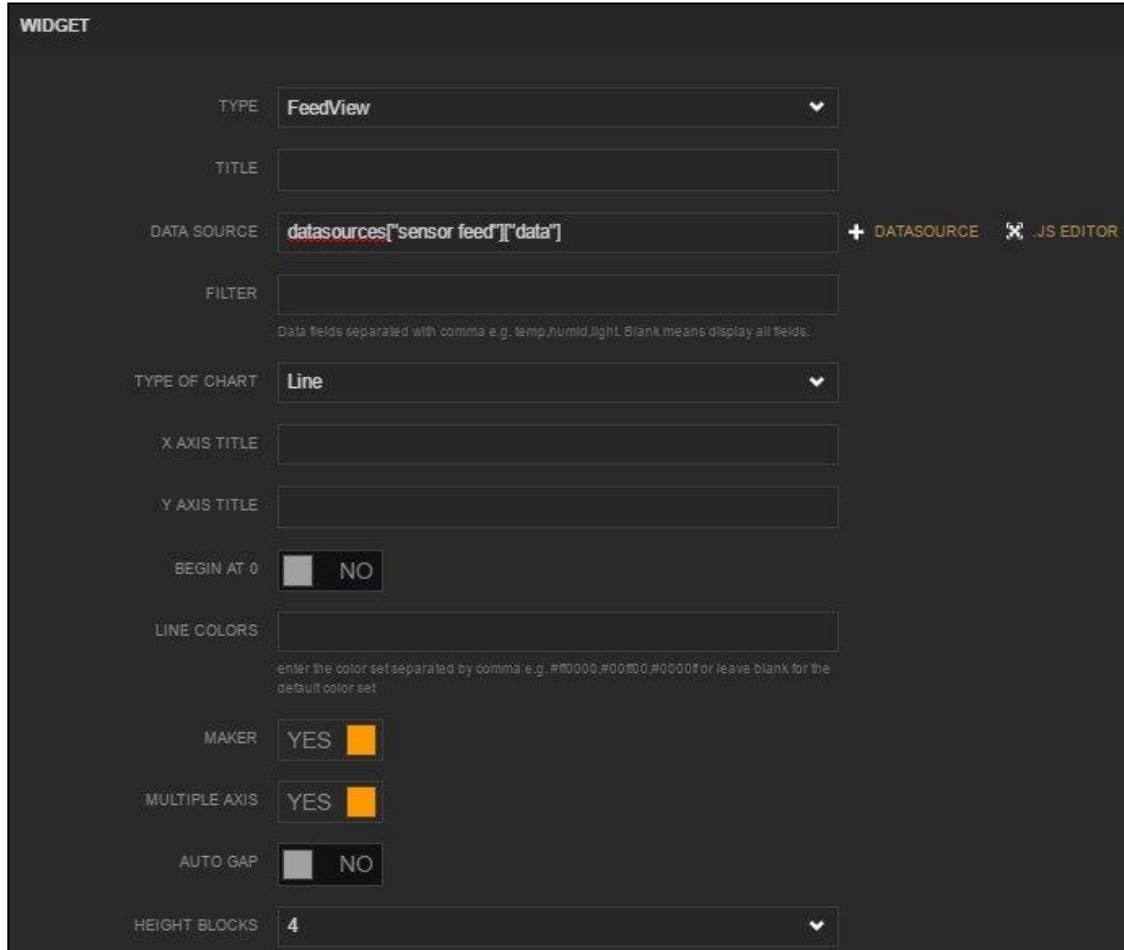
รูปที่ 6.14 การเลือกชนิดของ Widget เป็น FeedView

ในการตั้งค่า FeedView ตามภาพด้านล่าง ช่องที่สำคัญที่สุดคือช่อง Datasource เราจะต้องเลือก Datasource ที่มีชนิดเป็น NETPIE Feed โดยข้างถึง Field ที่มีชื่อว่า data ของ Datasource นั้น สูปคือให้ใส่ค่าในลักษณะนี้

```
datasources["<Feed Datasource Name>"]["data"]
```

โดยที่ Feed Datasource Name ให้แทนที่ด้วยชื่อของ Datasource ของ Feed ที่เราตั้งไว้ก่อนหน้านี้ (ในตัวอย่างใช้ชื่อ "sensor feed") หากไม่สะดวกจะพิมพ์ เราสามารถคลิกตรงปุ่ม +DATASOURCE ข้างหลังช่องจะมีตัวช่วยเลือกในการกรอกค่า ดังนั้นในตัวอย่างนี้ต้องใส่ว่า

```
datasources["sensor feed"]["data"]
```



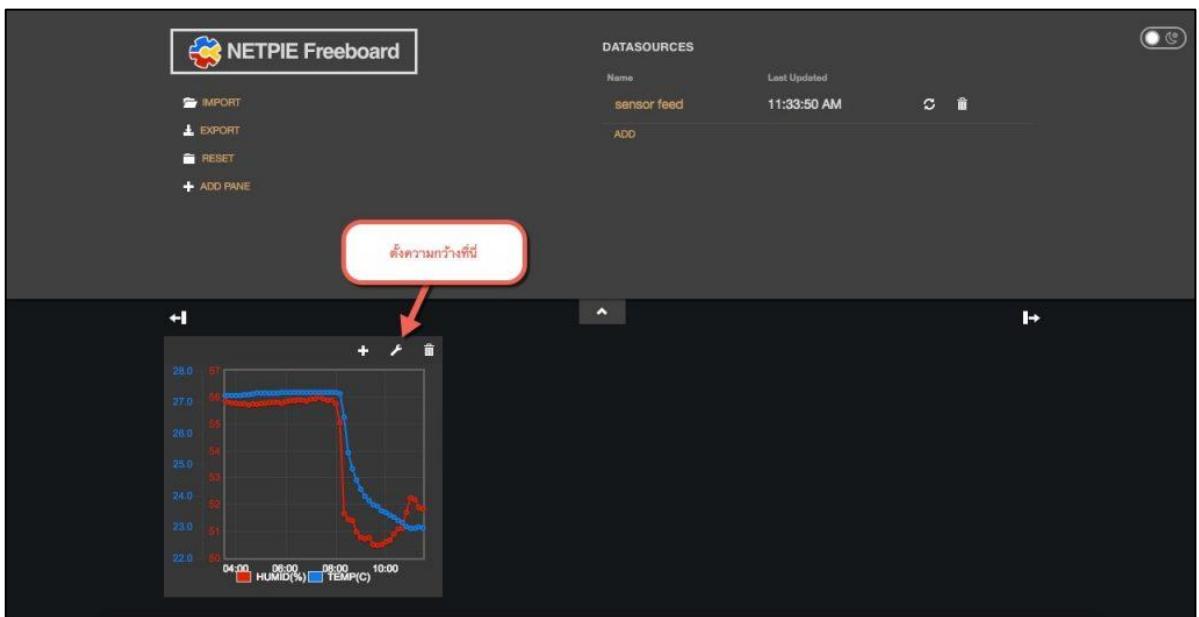
รูปที่ 6.15 การตั้งค่า Widget สำหรับ Feed

ตัวเลือกที่อื่นๆ สามารถปรับเปลี่ยนได้ตามความเหมาะสม

- TITLE ชื่อความที่จะเขียนบนกราฟ
- DATA SOURCE แหล่งข้อมูลที่จะดึงมาแสดงผล
- FILTER เป็นตัวกรอง สมมติว่าข้อมูลจาก data source มี 3 เส้น ได้แก่ temp, humid และ light หากเราอยากรอกราฟแสดงเฉพาะ temp และ humid ก็ใส่เฉพาะคำว่า temp,humid ลงในช่อง filter
- TYPE OF CHART มีสองตัวเลือก ได้แก่ LINE กับ STEP
- X AXIS TITLE ชื่อความกำกับบนแกน X

- Y AXIS TITLE ชื่อความกำกับบนแกน Y
- BEGIN AT 0 ตั้งให้ค่า Y เริ่มต้นที่ 0 หรือไม่
- LINE COLORS หากต้องการ สามารถเลือกสีเส้นกราฟได้ เดຍไสเป็นค่าสี HTML ตัวอย่างเช่น #ff0000,#00ff00,#0000ff หมายถึง กำหนดให้เส้นแรกเป็นสีแดง เส้นที่สองเป็นสีเขียวและเส้นที่สามเป็นสีน้ำเงิน กราฟมากกว่าสามเส้นนี้ จะวนกลับไปเริ่มใช้สีแดงใหม่ในเส้นถัดไป
- MARKER แสดงวงกลมที่ตำแหน่งจุดข้อมูล
- MULTIPLE AXIS แสดงแกน Y แยกสำหรับแต่ละค่าคิวเพร ซึ่งอาจจำเป็นต้องใช้ ในการนี้ที่ค่าข้อมูลแต่ละชุดมีสเกลที่แตกต่างกันมาก ถ้าจับมาพัลล์ตบันแกนเดียวกัน จะแสดงรายละเอียดได้ไม่ดีนัก
- AUTO GAP ตั้งให้ตรวจสอบและแทรกช่องว่างอัตโนมัติ หากข้อมูลหายไปนานผิดปกติ เส้นกราฟจะปรากฏการขาดช่วงให้เห็น

คลิก Save ก็จะได้ Widget สำหรับแสดงผลกราฟที่ดึงข้อมูลจาก Feed เราสามารถขยายขนาดความกว้างของ Widget ได้ โดยการคลิกไอคอนเครื่องมือดังแสดงในรูปภาพ



รูปที่ 6.16 การปรับแต่งกราฟของ Feed



รูปที่ 6.17 การปรับแต่งข่ายความกว้างของกราฟของ Feed

วิธีที่ 3 การแสดงผลผ่านหน้า REST API

สำหรับนักพัฒนาที่ต้องการนำค่าจาก Feed ไปใช้แบบอื่นๆ เช่น แสดงผลใน Visualization Tool ของตนเอง หรือนำไปคำนวณต่อ ก็ยังสามารถดึงค่าของ Feed จาก API ผ่าน Method GET ได้ ในรูปแบบดังนี้

```
https://api.netpie.io/feed/<FEEDID>?apikey=<APIKEY>&granularity=<GRANULARITY>&since=<SINCE>&filter=<FILTER>
```

- FEEDID คือ Feed ID
- APIKEY คือ API Key
- GRANULARITY คือ ระดับความละเอียดของจุดข้อมูล หน่วยที่สามารถเป็นได้ ได้แก่ second, minute, hour, day, month, year และใส่ตัวเลขระบุปริมาณหน่วยน้ำได้ส่วนหน่วยเวลาจะเติม s หรือไม่ก็ได้ ไม่มีความแตกต่าง เช่น 15seconds หรือ 15second, 10minutes, 3hour ฯลฯ การตั้ง Granularity เป็น 5 minutes จะทำให้ทุกจุดข้อมูลที่เก็บได้ในช่วง 5 นาที จะถูกนำมาเฉลี่ยเป็นจุดๆเดียวที่ใช้เป็นตัวแทนของช่วงดังกล่าว
- SINCE คือ ระยะเวลาข้อมูลที่จะดึงข้อมูลออกมามีรูปแบบการเขียนเหมือนกับ Granularity ทุกอย่าง

- FILTER คือ ตัวกรอง จะใส่หรือไม่ใส่ก็ได้ ถ้าไม่ใส่แปลว่าไม่มีตัวกรอง คือดึงมาทุกค่า ถ้าใส่เป็นชื่อ Field ของข้อมูลที่จะดึงออกมาแสดง หากมีหลาย Field ให้คั่นด้วยเครื่องหมายคอมมา (,) เช่น

```
https://api.netpie.io/feed/mysensor?apikey=5DunZ38nKP5dGC0h4Bj7mXyeMURdoXOo
&granularity=10minutes&since=24hours&filter=temp,humid
```

ผลลัพธ์ที่ได้จะเป็น JSON ลักษณะตามด้านล่างนี้ ส่วนของ Timestamp จะเป็น Unix Timestamp แบบ Millisecond ของโซนเวลา GMT ดังนั้นหากจะนำมาใช้ ต้องบวกเพิ่ม 7 ชั่วโมงให้เป็นเวลาประเทศไทยด้วย

```
{
    "feedid": "mysensor",
    "description": "",
    "from": null,
    "to": null,
    "since": [
        24,
        "hours"
    ],
    "granularity": [
        1,
        "minutes"
    ],
    "data": [
        {
            "attr": "humid",
            "unit": "%",
            "values": [
                [
                    1484031488709,
                    62.5
                ],
                [
                    1484031500861,
                    62.6
                ]
            ],
            {
                "attr": "temp",
                "unit": "C",
                "values": [
                    [
                        1484031488709,
                        25.5
                    ],
                    [
                        1484031500861,
                        25.6
                    ]
                ]
            }
        }
    ]
}
```

```
        "unit": "C",
        "values": [
            [
                1484031488709,
                25.2
            ],
            [
                1484031500861,
                25.1
            ]
        ]
    },
    "lastest_data": [
        {
            "attr": "humid",
            "values": [
                [
                    1484031500861,
                    62.6
                ]
            ]
        },
        {
            "attr": "temp",
            "values": [
                [
                    1484031500861,
                    25.1
                ]
            ]
        }
    ]
}
```

6.5 ข้อจำกัดการใช้งาน

บริการ NETPIE Feed เปิดให้ใช้ฟรีสำหรับผู้ใช้ทั่วไป ดังนั้นเพื่อแบ่งปันทรัพยากรกันอย่างเหมาะสม จะจำกัดอัตราการเขียน Feed ต่อ API Key ไว้ที่ 4 ครั้งในเวลาประมาณ 60 วินาที หรือเฉลี่ยให้เขียนได้ 15 วินาที ต่อ 1 จุด และในเบื้องต้นจะให้ระยะเวลาเก็บข้อมูลหนึ่งปี การเขียนข้อมูลลงใน Feed อาจจะเขียนผ่านทาง Microgear หรือ Client หลายตัวเพียงแต่ Quota ของการเขียนก็จะแบ่งกันไประหว่าง Client ที่ใช้ API key เดียวกัน ในส่วนของการอ่านค่านี้ จะจำกัดการเรียก API ไว้ที่ 5 ครั้งใน 5 วินาที การเรียก API แต่ละครั้ง ถ้าอ่านหรือเขียนสำเร็จจะมีสถานะตอบกลับ ตามด้านอย่างนี้

- {"code":200,"message":"Update OK"} หมายความว่า ทำงานสำเร็จ
- {"code":401,"message":"Unauthorized"} หมายความว่า API Key ที่ใช้ไม่มีสิทธิเข้าถึง Feed ดังกล่าว
- {"code":429,"message":"Rate limit exceeded, wait 2.286758 seconds"} หมายความว่า มีการเรียกใช้งานเกินอัตราที่กำหนด

นอกจากสถานะตอบกลับแล้ว ในกรณีของการเรียกผ่าน REST API ส่วนของ Reply ก็จะมี HTTP Header แนบกลับมาด้วยเพื่อบอกสถานะเกี่ยวกับ Rate Limit และ Quota ที่ยังเหลืออยู่ ซึ่งอาจเป็นประโยชน์ต่อนักพัฒนา ตัวอย่างเช่น หากเรียก API สำหรับอ่านข้อมูลจาก Feed จำนวน 5 ครั้งติดกัน ครั้งสุดท้ายจะเห็น Header ดังนี้

```
X-RateLimit-Interval: 5  
X-RateLimit-Quota: 5  
X-RateLimit-Minimum-Wait: 0  
X-RateLimit-ToWait: 1.570335
```

เป็นการบอกว่าต้องรออีกประมาณ 1.57 วินาที จึงจะสามารถ เรียก API เดิมช้าอีกครั้งได้ ซึ่งหากผู้ใช้ไม่รอและเรียก API ต่อทันที ค่าที่ Reply ครั้งถัดไปจะมีสถานะเป็น 429 Rate limit exceeded

Lab 6.1: การแสดงผลข้อมูลเซนเซอร์ด้วย NETPIE Feed

Lab นี้เป็นการทดลองการประยุกต์ใช้ NETPIE Feed ในการดูข้อมูลอุณหภูมิและความชื้นที่ได้รับจากเซนเซอร์โมดูล DHT11 ผู้อบรมสามารถฝึกการนำข้อมูลที่เก็บจาก Feed มาใช้แสดงผลในรูปแบบต่างๆ พร้อมทั้งเข้าใจขีดจำกัดการใช้งานของระบบและบริการ

- สร้างไฟล์ feed_dht.ino ตามโค้ดด้านล่าง โดยระบุข้อมูลการเข้าถึงเครือข่าย Wifi ข้อมูล APPID, KEY และ SECRET และทำการ Upload ไฟล์เข้า NodeMCU ให้เชื่อมต่อกับ NETPIE
- สร้าง Feed เพื่อรับค่าของข้อมูลอุณหภูมิและความชื้นที่เซ็นเซอร์สั่งมา ปรับแต่งการแสดงผลตามความเหมาะสม (ใน Data Display)
- เพิ่ม Datasource สำหรับ Feed ในหน้า Freeboard
- เพิ่ม Widget เพื่อแสดงผล Feed ในหน้า Freeboard
- ทดลองเปลี่ยนอัตราการส่งข้อมูลของ Feed จาก NodeMCU ให้เจ็วกว่าค่าของ INTERVAL ที่ระบุ (หรือเจ็วกว่าอัตราเนลลี่ที่เขียนได้คือ 15 วินาที ต่อ 1 จุด) สังเกตการเปลี่ยนแปลงในการแสดงผลของข้อมูลว่าเป็นอย่างไร มีผลต่อ Rate Limit หรือไม่ ในการดูระยะเวลาที่จะรับข้อมูลในระดับวินาที ควรปรับแต่งช่วงเวลาในการแสดงผลให้สั้นลง เช่น อยู่ในระดับ 1 นาที เป็นต้น

feed_dht.ino

```
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <MicroGear.h>

const char* ssid      = "SSID";
const char* password = "PASSWORD";

#define APPID    "YOUR_APPID"
#define KEY      "YOUR_KEY"
#define SECRET   "YOUR_SECRET"

#define ALIAS    "YOUR_MICROGEAR_NAME"
#define FEEDID   "YOUR_FEED_ID"

#define INTERVAL 15000
#define T_INCREMENT 200
#define T_RECONNECT 5000
#define BAUD_RATE 115200
#define MAX_TEMP 100
#define MAX_HUMID 100

WiFiClient client;
```

```

int timer = 0;
char str[32];

#define DHTTYPE DHT11           //Define sensor type
#define DHTPIN D4               // Define sensor pin
DHT dht(DHTPIN, DHTTYPE);   //Initialize DHT sensor

float humid;
float temp;

MicroGear microgear(client);

// when the other thing send a msg to this board
void onMsgHandler(char *topic, uint8_t* msg, unsigned int msglen) {
    Serial.print("Incoming message --> ");
    msg[msglen] = '\0';
    Serial.println((char *)msg);
}

void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
    microgear.setAlias(ALIAS);
}

void setup() {
    dht.begin();

    microgear.on(MESSAGE, onMsgHandler);
    microgear.on(CONNECTED, onConnected);

    Serial.begin(BAUD_RATE);
    Serial.println("Starting...");

    if (WiFi.begin(ssid, password)) {
        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
        }
    }
}

```

```

Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

microgear.init(KEY,SECRET,ALIAS);
microgear.connect(APPID);

}

void loop() {
    if (microgear.connected()) {
        Serial.print("*");
        microgear.loop();
        if (timer >= INTERVAL) {
            humid = dht.readHumidity();
            temp = dht.readTemperature();

            Serial.print("\nHumidity: ");
            Serial.print(humid);
            Serial.print(" %\t");
            Serial.print("Temperature: ");
            Serial.print(temp);
            Serial.print(" C%\n");

            String data = "{\"humid\":\"";
            data += humid;
            data += ", \"temp\":\"";
            data += temp;
            data += "}";
            if (isnan(humid) || isnan(temp) || humid >= MAX_HUMID || temp>=MAX_TEMP) {
                Serial.println("Failed to read from DHT sensor!");
            }else{
                Serial.print("Sending -->");
                Serial.println((char*) data.c_str());
                microgear.writeFeed(FEEDID,data); //YOUR FEED ID, API KEY
            }
            timer = 0;
        }
        else timer += T_INCREMENT;
    }
}

```

```
}

else {
    Serial.println("connection lost, reconnect...");

    if (timer >= T_RECONNECT) {
        microgear.connect(APPID);
        timer = 0;
    }
    else timer += T_INCREMENT;
}

delay(200);
}
```

หมายเหตุ: ไฟล์ feed_dht.ino ได้รับการทดสอบด้วยการ compile กับ ESP8266 Microgear Version 1.2.2 และ DHT Sensor Library by Adafruit Version 1.2.3

7. NETPIE REST API

7.1 REST API

REST API (Representational State Transfer Application Programming Interface) หรือ บางครั้งเรียกว่า RESTful API หรือ REST Web API หมายถึงการสื่อสารแลกเปลี่ยนข้อมูลในแบบ Web Service ด้วยการใช้ HTTP Method เช่น GET POST PUT DELETE ข้อมูลที่แลกเปลี่ยนกันมักอยู่ในรูปแบบ JSON (JavaScript Object Notation) หรือ XML (eXtensible Markup Language) เนื่องจาก REST API ใช้โครงสร้างแบบ XML ที่มีความซับซ้อนและซับซ้อนกว่า SOAP แต่ REST API ใช้โครงสร้างแบบ JSON ที่มีความซับซ้อนและซับซ้อนกว่า XML แต่ REST API ใช้โครงสร้างแบบ XML ที่มีความซับซ้อนและซับซ้อนกว่า REST API จึงเป็นที่นิยม ปัจจุบันบริการ API ของ Facebook, Twitter, Google ก็เลือกใช้ REST API

คุณสมบติของ Web Service ที่ใช้ REST API จะใช้ URI (Universal Resource Identifier) ในการเรียกข้อมูล เช่น <http://api.example.com/resources> และใช้ HTTP Method ในการระบุการกระทำการต่อข้อมูล เช่น GET <http://api.example.com/resources> หมายถึงการเรียกข้อมูล หรือ POST <http://api.example.com/resources> หมายถึงการเปลี่ยนข้อมูล เป็นต้น

ตัวอย่างเช่น ห้องสมุดอาจจะทำ API ให้ผู้ดูแลระบบดึงข้อมูลของหนังสือ ISBN 0596801688 โดยมีรูปแบบคำร้องขอดังนี้

```
GET https://api.example.com/books/0596801688
```

และในขณะเดียวกัน การลบ Record ของหนังสือ ก็อาจจะทำเป็น API ง่ายๆ แบบนี้

```
DELETE https://api.example.com/books/0596801688
```

ส่วน Operation อื่นๆ เช่น POST ก็อาจจะเอาไว้ใช้สร้างข้อมูลใหม่ และ PUT สำหรับการอัปเดตข้อมูล ฯลฯ การใช้งานแบบนี้ ทำให้ API ดูเป็นธรรมชาติ อธิบายง่าย และด้วยความที่ HTTP เป็นprotoocolมาตรฐานที่มีใช้กันมานานแล้ว เราจึงพบเจอกับ HTTP Client อยู่ทุกหนทุกแห่ง บนทุกภาษาโปรแกรม และในหลากหลายรูปแบบการใช้งาน แม้แต่บน Command Line ก็ยังเรียกใช้งานได้

7.2 NETPIE REST API

NETPIE มีบริการ REST API เตรียมไว้ให้ สำหรับอุปกรณ์ที่ไม่สามารถใช้ Microgear ผ่านprotoocol MQTT ในกรณีที่ไม่มี Microgear ในภาษาที่ต้องการใช้ (เช่น iPhone ที่ใช้ Objective

C หรือ เว็บเซิร์ฟเวอร์ที่ใช้ PHP) ขอเพียงแค่อุปกรณ์รองรับพอร์ต HTTP ที่พอร์ต 80 ก็สามารถเรียกใช้ NETPIE REST API ได้ อุปกรณ์ที่ใช้ REST API สามารถแลกเปลี่ยนข้อมูลกับอุปกรณ์อื่นที่ใช้ REST API หรือกับอุปกรณ์ที่ใช้ Microgear ก็ได้ ข้อดีของการใช้ NETPIE REST API คือไม่จำเป็นต้องเขียนโปรแกรมในภาษาใดๆ ก็ได้ ข้อดีของการใช้ NETPIE REST API คือไม่จำเป็นติดกับ Programming Language ยาร์ดแวร์และระบบปฏิบัติการ สามารถนำไปประยุกต์ใช้กับอุปกรณ์แบบเดิม (Legacy Device) ได้ ใช้งานผ่าน TCP พอร์ต 80 ซึ่งเป็นพอร์ตที่เครือข่ายทั่วไปอนุญาตให้ใช้ (ถ้าใช้ Microgear ซึ่งเป็นพอร์ต MQTT ต้องใช้ TCP พอร์ต 1883, 8083, และ 8080)

การพัฒนา Microgear Library ออกแบบมาให้ครอบคลุมทุก Hardware Platform นั้นเป็นเรื่องที่เป็นไปได้ยากมาก และเป็นงานที่ใช้เวลา NETPIE จึงพัฒนา REST API ขึ้นมาเป็นอีกหนึ่งทางเลือก สำหรับ Hardware Platform ที่ NETPIE ยังไม่มี Library ไปรองรับ และรวมไปถึงระบบหรือบริการที่มีมาอยู่ก่อนแล้ว ก็สามารถเรียก API มาใช้ร่วมกับ NETPIE ได้ เช่นกัน

API Endpoint

REST API ของ NETPIE ให้บริการอยู่ที่ <https://api.netpie.io/>

Authentication

ในการเชื่อมต่อ API Client จะต้องทำการยืนยันตัวตน โดยใช้หนึ่งในสองวิธีนี้

- ส่งผ่าน HTTP Header แบบ Basic Auth โดยใช้

Username: KEY

Password: SECRET

ตัวอย่างการใช้ Basic Auth ด้วย Command Line คำสั่ง cURL

```
$ curl -X GET "http://www.domainname.com/resource" -u key:secret
```

- ส่งผ่านทาง URL Parameter ในรูปแบบ

```
?auth=KEY:SECRET
```

ตัวอย่างการใช้ URL Parameter ด้วย Command Line คำสั่ง cURL

```
$ curl -X GET "http://www.domainname.com/resources?auth=key:secret"
```

Resource

การใช้ REST API ของ NETPIE จะเป็นการระหว่างทำกากบ้างอย่างกับ Resource ซึ่ง Resource ที่เข้าถึงได้ของ NETPIE มีอยู่ 3 แบบด้วยกัน ได้แก่



1. Microgear หมายถึงตัว คุปกรณ์เอง เราสามารถ Chat ตรงไปมันได้ โดยอ้างอิงชื่อ Alias
2. Topic เป็นสิ่งเดียวกับ Topic เวลา Publish/Subscribe เราสามารถ Publish ข้อมูลไปที่ Topic โดยใช้ REST API ได้เช่นกัน โดยคุปกรณ์ที่ Subscribe อยู่ก็จะได้รับข้อมูล Topic นั้น
3. Postbox คือถังเก็บข้อมูลข่าวคราว สามารถส่งข้อมูลเข้าไป หรืออ่านข้อมูลออกมามาได้

HTTP Method

HTTP Method ที่ NETPIE REST API รองรับคือ GET (เพื่อเรียกดูข้อมูล) และ PUT (เพื่อเขียนข้อมูล) วิธีการเรียกใช้งาน

PUT

PUT /microgear/{appid}/{alias}

PUT /topic/{appid}/{topicname}

PUT /postbox/{appid}/{postboxname}

Parameter Method PUT รับ Parameter ชื่อ retain ซึ่งหากกำหนด retain จะมีความหมายว่าให้ NETPIE เก็บค่านี้ไว้ ซึ่งแพลตฟอร์มจะเก็บเฉพาะค่าล่าสุดเพียงค่าเดียว

Body ในเนื้อหา (Payload) ของ HTTP PUT จะเป็นข้อมูลที่ต้องการส่งไปยัง topic/microgear/postbox

ตัวอย่างการเรียกใช้งาน PUT ในการส่งข้อมูล ON แบบ retain ไปยัง Topic ชื่อ mytopic ใน AppID ชื่อ myappid แบบส่ง Authentication ไปใน URL

```
$ curl -X PUT
"https://api.netpie.io/topic/myappid/mytopic?retain&auth=MyKey:MySecret"-d
"ON"
```

ตัวอย่างการเรียกใช้งาน PUT ในการส่งข้อมูล ON แบบ retain ไปยัง Topic ชื่อ mytopic ใน AppID ชื่อ myappid แบบส่ง Authentication ไปใน HTTP Header แบบ Basic Auth

```
$ curl -X PUT "https://api.netpie.io/topic/myappid/mytopic?retain" -d "ON"
-u MyKey:MySecret
```

GET

GET /microgear/{appid}/{alias}

GET /topic/{appid}/{topicname}

GET /postbox/{appid}/{postboxname}

Parameter ไม่มี

Body ไม่มี

ตัวอย่างการเรียกใช้งาน GET ในการอ่านข้อมูลความใน Topic หรือ mytopic ใน AppID ชื่อ myappid แบบส่ง Authentication ไปใน URL

```
$ curl -X GET  
"https://api.netpie.io/topic/myappid/mytopic?auth=MyKey:MySecret"
```

ตัวอย่างการเรียกใช้งาน GET ในการอ่านข้อมูลความใน Topic หรือ mytopic ใน AppID ชื่อ myappid แบบส่ง Authentication ไปใน HTTP Header แบบ Basic Auth

```
$ curl -X GET " https://api.netpie.io/topic/myappid/mytopic"-u  
MyKey:MySecret
```

เราสามารถทดสอบการเรียก Method Get โดยนำ URL ไปใส่ใน Browser โดยตรงได้ เช่น



ถ้าทำงานได้ถูกต้องจะได้รับค่าที่เคย PUT และ retain ไปยัง topic/microgear/postbox นี้ ในรูปแบบ JSON เช่น

```
[{"topic":"/myappid/mytopic","payload":"ON","lastUpdated":1471760882,"retain":true}]
```



การพิมพ์ URL ลงไว้ใน Address Bar ของเบราว์เซอร์ จะเป็นการเรียก HTTP GET Method เท่านั้น ไม่สามารถใช้กับ Method อื่นๆ ได้

Lab 7.1: HTML → HTML

ในการทดลองนี้จะให้เบราว์เซอร์สองตัว (บนเครื่องเดียวกันหรือต่างเครื่องก็ได้) สื่อสารกันผ่าน NETPIE REST API โดยเป็นการเขียนโปรแกรมด้วย HTML และ Javascript โดยเบราว์เซอร์ตัวแรกส่งคำสั่ง PUT และเบราว์เซอร์ตัวที่สองส่งคำสั่ง GET

สร้างไฟล์ SetLampStatus.html ดังนี้

SetLampStatus.html

```
<html>
<body>
<script>

    var APPID= "YOURAPPID"; //enter your appid
    var KEY = "YOURKEY"; //enter your key
    var SECRET = "YOURSECRET"; //enter your secret
    var Topic = "/LampStatus"; //choose any topic name

    function PressButtonOn() {
        var url =
'https://api.netpie.io/topic/' +APPID+Topic+'?retain&auth=' +KEY+':'+SECRET;
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.open('PUT',url,true);
        xmlhttp.send('ON');
        window.alert(url); //for debugging purpose
    }

    function PressButtonOff() {
        var url =
'https://api.netpie.io/topic/' +APPID+Topic+'?retain&auth=' +KEY+':'+SECRET;
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.open('PUT',url,true);
        xmlhttp.send('OFF');
        window.alert(url); //for debugging purpose
    }
</script>

<center>
<button onclick="PressButtonOn()" id = "ButtonOn">ON</button>
<button onclick="PressButtonOff()" id = "ButtonOff">OFF</button>
</center>
</body>
</html>
```

Double click ไฟล์ SetLampStatus.html เลือกเปิดบนเบราว์เซอร์ เช่น IE จะเห็นหน้าต่างแบบนี้



สร้างไฟล์ GetLampStatus.html ดังนี้

GetLampStatus.html

```
<html>
<body>
<script>

    var APPID= "YOURAPPID"; //enter your appid
    var KEY = "YOURKEY"; //enter your key
    var SECRET = "YOURSECRET"; //enter your secret
    var Topic = "/LampStatus";

    function GetButtonStatus() {
        var url = 'https://api.netpie.io/topic/' +APPID+Topic+'?auth='
        +KEY+':'+SECRET;

        var xmlhttp = new XMLHttpRequest();

        xmlhttp.onreadystatechange = function() {
            if (xmlhttp.status == 200 && xmlhttp.readyState==4) {
                document.body.className = 'ok';
                var result = xmlhttp.responseText;
                window.alert(result);
            } else {
                document.body.className = 'error';
            }
        }
        xmlhttp.open('GET',url,true);
        xmlhttp.send(null);
    }

</script>

<center>
<button onclick="GetButtonStatus()" id = "Button">Check Lamp Status</button>
</center>

</body>
</html>
```

เปิดไฟล์ GetLampStatus.html บนเว็บเบราว์เซอร์อีกตัว เช่น Google Chrome หรือ Firefox หรือ เปิดเว็บเบราว์เซอร์บนเครื่องของเพื่อน จะเห็นหน้าต่างแบบนี้



กดปุ่ม Check Lamp Status สังเกตผลลัพธ์ที่ได้ จะอยู่ในรูปแบบ JSON Array ซึ่งประกอบด้วย

[ชื่อ Resource, ข้อความ Payload, Last Update, retain] เช่น

```
[{"topic":"/YOURAPPID/LampStatus", "payload":"ON", "lastUpdated":1471760882, "retain":true}]
```



KEY ที่ใส่ในไฟล์ SetLampStatus.html และ GetLampStatus.html ควรสร้างเป็น Session Key สามารถใช้ KEY เดียวกันทั้งสองไฟล์หรือจะใช้ KEY ที่แตกต่างกันก็ได้

ทดลองเพิ่มเติม

1. พิมพ์ URL ลงใน Address Bar โดยตรง
<https://api.netpie.io/topic/YOURAPPID/LampStatus?auth=YOURKEY:YOURSECRET>
สังเกตผลลัพธ์ที่ได้
2. แก้ไขไฟล์ SetLampStatus.html โดย retain parameter ออกสังเกตผลลัพธ์ที่ได้จากคำสั่ง GET
3. แก้ไขไฟล์ SetLampStatus.html และ GetLampStatus.html โดยเปลี่ยนชื่อ Topic เป็น /LampStatus/bedroom สังเกตผลลัพธ์ที่ได้จากคำสั่ง GET
4. แก้ไขไฟล์ SetLampStatus.html และ GetLampStatus.html เปลี่ยนจากการใช้ Topic เป็น Postbox



Topicname และ Postboxname สามารถตั้งเป็นชื่อใดๆ ที่ต้องการ สามารถเรียกใช้ได้ภายใน AppID ที่กำหนดเท่านั้น

Lab 7.2: HTML → NodeMCU

ในการทดลองนี้จะให้เบราว์เซอร์สื่อสารกับ Microgear บน NodeMCU ผ่าน REST API

- ไฟล์ SetLampStatus.html จาก Lab 7.1 แก้ไข URL สำหรับ PUT ให้อยู่ในรูปแบบ

<https://api.netpie.io/microgear/appid/alias?retain&auth=KEY:SECRET>

โดย alias คือชื่อที่เราตั้งให้กับ NodeMCU Microgear ให้ใช้ตามที่เราเคยตั้งให้ Node MCU ใน Lab 4.2 ในไฟล์ pieled.ino ให้ alias ว่า pieled

- เนื่องจากเราจะใช้ไฟล์ pieled.ino จาก Lab 4.2 แต่เราตั้งเงื่อนไขให้ NodeMCU รับข้อมูล 1 หรือ 0 จึงต้องแก้พังก์ชัน PressButtonOn และ PressButtonOff ในไฟล์ SetLampStatus.html ให้ส่งค่า 1 และ 0 แทน ON และ OFF ตามลำดับ

SetLampStatus.html

```
<html>
<body>
<script>

    var APPID= "YOURAPPID.";// enter your appid
    var KEY ="YOURKEY";// enter your key
    var SECRET ="YOURSECRET";// enter your secret
    var ALIAS = "pieled";// same alias you set on NodeCMU

    function PressButtonOn(){
        var url =
'https//:api.netpie.io/microgear/' +APPID+ALIAS+'?retain&auth='
+KEY+':'+SECRET;

        var xmlhttp =new XMLHttpRequest();
        xmlhttp.open('PUT',url,true);
        xmlhttp.send('1');
        window.alert(url);//for debugging purpose
    }

    function PressButtonOff(){
        var url =
'https//:api.netpie.io/microgear/' +APPID+ALIAS+'?retain&auth='
+KEY+':'+SECRET;

        var xmlhttp =new XMLHttpRequest();
        xmlhttp.open('PUT',url,true);
        xmlhttp.send('0');
        window.alert(url);// for debugging purpose
    }
</script>
</body>
</html>
```

```

        }
    </script>

<center>
<button onclick= "PressButtonOn()" id = "ButtonOn">ON</button>
<button onclick= "PressButtonOff()" id = "ButtonOff">OFF</button>
</center>
</body>
</html>

```

3. อัปโหลดโค้ด pieled.ino (เหมือนใน Lab 4.2) ลงใน NodeMCU

pieled.ino

```

#include <ESP8266WiFi.h>
#include <MicroGear.h>

const char* ssid      = "SSID"; //change this to your SSID
const char* password = "PASSWORD"; //change this to your PASSWORD

#define APPID      "APPID"          //change this to your APPID
#define KEY       "KEY"            //change this to your KEY
#define SECRET    "SECRET"         //change this to your SECRET

#define ALIAS     "pieled"

WiFiClient client;

int timer = 0;

MicroGear microgear(client);

void onMsghandler(char *topic, uint8_t* msg, unsigned int msglen) { //
    Serial.print("Incoming message -->"); 
    msg[msglen] = '\0';
    Serial.println((char *)msg);
    if(*(char *)msg == '1'){
        digitalWrite(LED_BUILTIN, LOW);    // LED on
        microgear.chat("switch","1");
    }else{

```

```

        digitalWrite(LED_BUILTIN, HIGH); // LED off
microgear.chat("switch","0");
}

}

void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
    microgear.setName(ALIAS);
}

void setup() {
    microgear.on(MESSAGE,onMsghandler);
    microgear.on(CONNECTED,onConnected);

    Serial.begin(115200);
    Serial.println("Starting...");

    pinMode(LED_BUILTIN, OUTPUT);

    if (WiFi.begin(ssid, password)) {
        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
        }
    }

    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    microgear.init(KEY,SECRET,ALIAS);
    microgear.connect(APPID);
}

void loop() {
    if (microgear.connected()) {
        Serial.println("...");
        microgear.loop();
        timer = 0;
    }
}

```

```
}

else {
    Serial.println("connection lost, reconnect...");

    if (timer >= 5000) {
        microgear.connect(APPID);
        timer = 0;
    }
    else timer += 100;
}

delay(100);
}
```

4. สั่งเกตผลที่ได้เมื่อกดปุ่ม ON OFF ที่เว็บเบราว์เซอร์ จะสามารถควบคุมไฟที่บอร์ด NodeMCU เช่นเดียวกับการใช้ Microgear บน HTML5

Lab 7.3: KKMC IoT

แอปพลิเคชัน KKMC-IoT พัฒนาโดยคุณคมเดช เพื่อทดสอบ จากการหัวที่อยาลับขอนแก่นและชุมชน ขอนแก่นเมกะเกอร์คลับ (KKMC) เป็น Android Application สำหรับสื่อสารไปยัง Microgear หรืออุปกรณ์ อื่นๆ ที่เชื่อมต่อกับ NETPIE การสื่อสารทำได้ทั้งการรายงานผลสถานะ และการส่งคำสั่งควบคุม รวมถึงการ ตั้งเวลาเพื่อส่งคำสั่งควบคุม เป็นหลังการสื่อสารของแอปพลิเคชันนี้คือการเรียกใช้งาน NETPIE REST API เพื่อเชื่อมต่อกับอุปกรณ์ต่างๆ (เนื่องจาก ณ เวลาที่พัฒนาแอปพลิเคชันนี้ ยังไม่มี Microgear สำหรับ สร้าง Native Android Application)

ข้อมูลรายละเอียดแอปพลิเคชันจาก Google Play

(<https://play.google.com/store/apps/details?id=com.numberx.kkmctimer>)

"KKMC IoT สร้างขึ้นมาเพื่อให้ผู้ใช้งานที่ต้องการจะควบคุมอุปกรณ์ IoT กับ NETPIE หรือมี อุปกรณ์อยู่แล้ว แต่ยังขาด App ดีๆ ซึ่งผู้ใช้บางคนจะต้องสร้างส่วนเชื่อมต่อขึ้นมา เช่น เว็บ หรือ application ต่างๆ เพื่อควบคุมโดยเฉพาะ หลาย ๆ ท่านอาจจะยังไม่มีความรู้ทางด้านโปรแกรมมิ่ง จึงอาจจะมองว่าเป็นอะไรที่ยุ่งยาก KKMC IoT จึงได้ถือกำเนิดขึ้นมาโดยมี Concept ที่ว่า "แอป เดียว ทำได้ทุกอย่าง" โดยตัวแอปนั้นเขียนไว้ให้ผู้ใช้สามารถกดสร้าง Widget การควบคุมอุปกรณ์ ไว้ทั้งหมดแล้วเพียงเข้าไปตั้งค่า Topic สำหรับการเชื่อมต่อ จากนั้นก็สามารถใช้งานได้เลย มีทั้ง การแสดงผล และควบคุมอุปกรณ์อีกทั้งยังเพิ่ม Timer ให้อุปกรณ์ทำงานได้ตามเวลาที่กำหนดอีกด้วย"

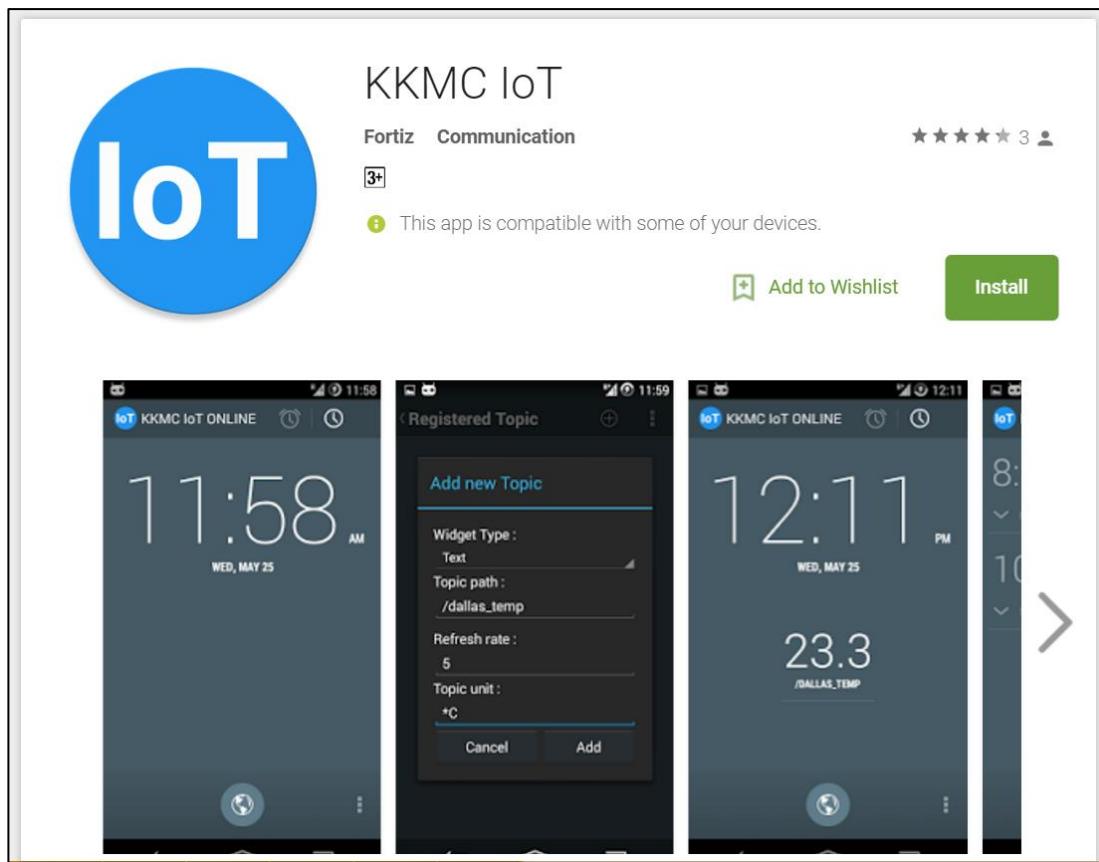
หากใครสนใจรายละเอียดเพิ่มเติม สามารถเข้าไปร่วมกลุ่มกับ Khon Kaen Maker Club ได้ เลยครับ"

Fanpage: <https://www.facebook.com/KhonKaenMakerClub/>

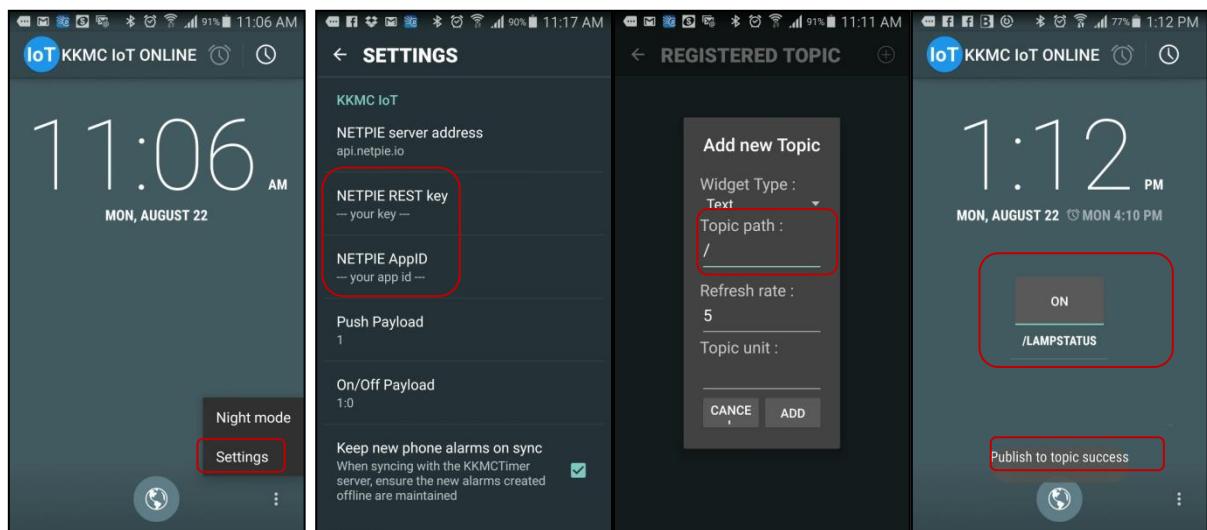
Group: <https://www.facebook.com/groups/KhonKaenMakerClub/>

Distributed from: <https://github.com/carlosperate/LightUpDroid-Alarm>

1. ทดลองติดตั้งแอปพลิเคชัน KKMC-IoT จาก Google Play Store



รูปที่ 7.1 แอปพลิเคชัน KKMC IoT ใน Google Play Store



ตั้งค่าขั้นตอนที่ 1

ตั้งค่าขั้นตอนที่ 2-3

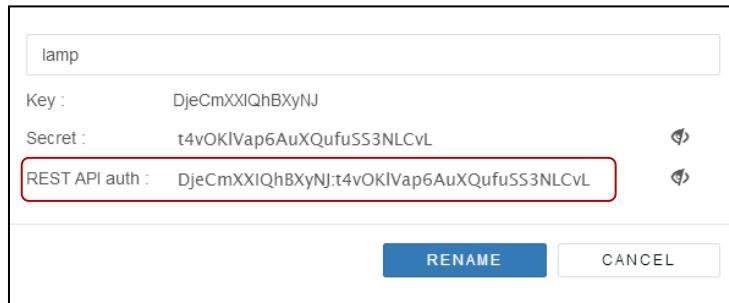
ตั้งค่าขั้นตอนที่ 4

ตั้งค่าขั้นตอนที่ 5

รูปที่ 7.2 ขั้นตอนตั้งค่าแอปพลิเคชัน KKMC IoT

2. เลือกเมนู Settings ที่มุ่งหมาย
3. ใส่ข้อมูล NETPIE AppID

4. ใส่ข้อมูล NETPIE REST Key ซึ่งอยู่ในรูปแบบ KEY: SECRET หรือสามารถก็อปปี้โดยตรงจากหน้า Application Management ของ netpie.io ตามภาพ



รูปที่ 7.3 ข้อมูล REST Key ในหน้า Application Management ของเว็บไซต์ netpie.io

5. ตั้งค่า Topic โดยกดที่รูปໂລກในหน้าแรกของแอปพลิเคชัน แล้วกดเพิ่ม Topic ตามชื่อ Topic ที่เคยตั้งไว้ เช่น /LampStatus เลือก Widget Type เป็น On/Off Switch (เพื่อสร้างปุ่มควบคุมไฟ)
6. กลับมาที่หน้าแรกของแอปพลิเคชันจะเห็นปุ่มให้ทดลองกด เปิด/ปิด ถ้าส่งคำสั่งไปสำเร็จ จะเห็นข้อความว่า “Publish to topic success” ตรวจสอบสถานะที่เปลี่ยนไปโดยเปิดไฟล์ GetLampStatus.html (จาก Lab 7.1) หรือแก้ไขโค้ด pieled.ino (จาก Lab 7.2) ที่ NodeMCU โดยเพิ่มคำสั่ง microgear.subscribe("/LampStatus"); ในฟังก์ชัน onConnected ดังแสดงด้านล่าง จากนั้นทดลองกดเปิด/ปิด เพื่อควบคุม LED บน NodeMCU

```
void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
    microgear.setName(ALIAS);
    microgear.subscribe("/LampStatus");
}
```

เนื่องจากแอปพลิเคชัน KKMC-IoT นี้รองรับเฉพาะการเรียก REST API บน Topic (ยังไม่รองรับ Resource ประเภท Microgear และ Postbox) จึงต้องสั่งให้ Microgear (NodeMCU) รอรับข้อความ (Subscribe) ที่ Topic/LampStatus ก่อน ซึ่งต่างจากใน Lab 7.2 ที่ใช้ REST API ส่งคำสั่งเปิดปิดไปยัง Resource ประเภท Microgear จึงสามารถส่งคำสั่งไปที่ชื่อของ Microgear ได้โดยตรงในรูปแบบ /microgear/appid/alias

ทดลองเพิ่มเติม

1. เพิ่ม Widget ชนิด Push Button (ปุ่มแบบกดติดปล่อยดับ) ที่ Topic /LampStatus สำหรับ LED บน NodeMCU เมื่อกดปุ่ม
2. เพิ่ม Widget ชนิด Text เพื่อแสดงผลข้อมูลอุณหภูมิที่ตรวจวัดจากเซ็นเซอร์ DHT11

รายละเอียดเพิ่มเติมเกี่ยวกับ KKMC-IoT

Source Code <https://github.com/KhonKaenMakerClub/KKMC-IoT>

บทความ <http://www.kkmakerclub.com/2016/05/26/>

วิดีโอสอนวิธีใช้ <https://www.youtube.com/watch?v=VWKrjvDCQQs>

Disclaimer NECTEC/NETPIE ไม่ได้มีส่วนร่วมในการพัฒนาแอปพลิเคชัน KKMC-IoT นี้ ข้อผิดพลาดใดๆ ที่เกิดจากการใช้แอปพลิเคชันนี้ ขอให้แจ้งไปยังผู้พัฒนาโดยตรง

8. NETPIE SECURE CONNECTION

ในการนี้ที่ต้องการให้อุปกรณ์สื่อสารกันอย่างปลอดภัยไม่ให้ใครดักฟังข้อมูลความที่แลกเปลี่ยนกันได้ และไม่ให้ความเป็นส่วนตัวของข้อมูลทางเราจำเป็นต้องเข้ารหัสให้ช่องทางสื่อสาร proto協定ที่นิยมใช้ในการรักษาความปลอดภัยให้ช่องทางสื่อสารได้แก่ TLS (Transport Layer Security) และ SSL (Secure Socket Layer) ซึ่งมักจะเรียกว่า กันว่า SSL/TLS และลิ้นชั้นที่นิยมใช้บน SSL/TLS ได้แก่ เว็บ อีเมล โปรแกรมสนทนาระบบฯ ฯลฯ ที่ต้องการความปลอดภัยในการส่งข้อมูล ตัวอย่างเช่น proto協定 HTTPS หรือ HTTP Secure จะเป็นการส่งข้อมูลเว็บที่เข้ารหัสบน SSL หรือ TLS

การส่งข้อมูลบน SSL/TLS มีข้อดี 3 ด้านคือ

1. Privacy ข้อมูลที่รับส่งจะถูกเข้ารหัส ทำให้มีความเป็นส่วนตัวและปลอดภัยจากการดักฟังโดยบุคคลที่สาม
2. Authentication คู่สนทนาทั้งสองฝ่ายจะได้รับการยืนยันตัวตนโดยการใช้ Public/Private Key ที่ออกให้โดย Certificate Authority ที่ได้รับการยอมรับ ทำให้มั่นใจได้ว่าคู่สนทนาเป็นคนหรืออุปกรณ์ที่ต้องการสื่อสารด้วยจริงๆ ไม่ใช่ตัวปลอมที่มาสวมสิทธิ์
3. Reliability ข้อมูลที่รับส่งไม่สามารถถูกเปลี่ยนแปลงกลางทาง เพราะมีกลไกการตรวจสอบความถูกต้องของข้อมูลโดยใช้ Message Authentication Code

TLS vs SSL

บ่อยครั้งที่เกิดความสับสนระหว่าง TLS กับ SSL ทั้งสอง proto協定ทำงานนี้เหมือนกันคือสร้างช่องทางการสื่อสารที่ปลอดภัยด้วยการเข้ารหัสข้อมูล ยืนยันความถูกต้องของข้อมูล และยืนยันตัวตนของคู่สนทนา proto協定 SSL กำเนิดมาก่อน TLS และได้รับความนิยมสูงโดยเฉพาะกับการใช้งานเว็บ (HTTP over SSL) แต่ในปี ค.ศ.2014 มีการพบช่องโหว่ที่ร้ายแรงที่เรียกว่า POODLE Attack บน SSL เวอร์ชัน 3.0 ตั้งแต่นั้นมา SSL จึงกลายเป็นproto協定ที่ไม่ปลอดภัยและไม่ควรใช้

proto協定 TLS ถูกออกแบบมาที่หลังและออกแบบให้รองรับกลไกสมัยใหม่ที่เกี่ยวกับการสร้าง Key การแลกเปลี่ยน Key การเข้ารหัส จึงทำให้ TLS มีความปลอดภัยมากกว่า SSL

การใช้ TLS กับ NETPIE

NETPIE เลือกใช้proto協定 TLS เวอร์ชัน 1.2 ใน การสร้างความปลอดภัยให้กับการสื่อสารระหว่าง Microgear กับ NETPIE



NETPIE Microgear ที่รองรับการสื่อสารบน TLS ได้แก่ Node.js, HTML5, ESP8266

TCP Port ที่ใช้สื่อสาร

HTML5 Microgear	ESP8266 และ Node.js Microgear
TLS: 8081 and 8084 (default)	Non-TLS: 8080 and 1883 (default)
Non-TLS: 8080 and 8083	TLS: 8081 and 8883

ฟังก์ชันเพื่อเรียกใช้ TLS

บน Node.js และ ESP8266 Microgear จะมีฟังก์ชัน `useTLS(tlsmode)` เพื่อระบุว่าจะเลือกใช้ TLS หรือไม่ใช้การเข้ารหัสแบบ TLS 作为 argument `tlsmode` เป็น Boolean ถ้าตั้งค่า TRUE แปลว่าใช้ TLS ถ้าตั้งค่า FALSE แปลว่าไม่ใช้ TLS หากไม่ได้เรียกใช้ฟังก์ชันนี้ ค่า default ของ Microgear จะไม่ใช้ TLS

บน HTML5 Microgear ก็มีฟังก์ชัน `useTLS(tlsmode)` เช่นกัน แต่ค่า default คือใช้ TLS ดังนั้นการสื่อสารกับ HTML5 จะเป็นการสื่อสารแบบปลอดภัยเสมอ นอกจากนักพัฒนาจะปิดการใช้ TLS ด้วยคำสั่ง `useTLS(false)`

วิธีสร้าง Secure Connection บน ESP8266 Microgear แตกต่างจากการสร้าง Connection ปกติ ดังนี้

1. ประกาศตัวแปร Client เป็นชนิด WiFiClientSecure แทน WiFiClient

```
WiFiClientSecure client;
MicroGear microgear(client);
```

2. ภายในฟังก์ชัน `setup()` เรียกใช้ `useTLS(true)` ก่อนการเรียกฟังก์ชัน `init(KEY, SECRET, ALIAS)`

```
microgear.useTLS(true);
microgear.init(KEY, SECRET, ALIAS);
microgear.connect(APPID);
```

วิธีสร้าง Secure Connection บน Node.js Microgear แตกต่างจากการสร้าง Connection ปกติ ดังนี้

โดยต้องเรียกใช้ `useTLS(true)` ก่อนการเรียกฟังก์ชัน `connect(APPID)`

```
microgear.useTLS(true);  
microgear.connect(APPID);
```



สำหรับการใช้ TLS บน ESP8266 จะใช้ได้กับ ESP8266 SDK 2.1.0 แต่มีปัญหากับ SDK 2.2.0 และกลับมาใช้ได้กับ SDK 2.3.0-rc1

Lab 8.1: สร้าง Secure Connection ระหว่าง NodeMCU และ HTML5

แก้ไขไฟล์ pieled.ino จาก Lab 4.2 สองจุดคือ

1. ประกาศตัวแปร Client เป็นชนิด WiFiClientSecure แทน WiFiClient ก่อนสร้าง Microgear

```
WiFiClientSecure client;  
MicroGear microgear(client);
```

2. ภายในฟังก์ชัน setup() เรียกใช้ useTLS(true) ก่อนการเรียกฟังก์ชัน init(KEY, SECRET, ALIAS)

```
microgear.useTLS(true);  
microgear.init(KEY, SECRET, ALIAS);  
microgear.connect(APPID);
```

บันทึกเป็นไฟล์ใหม่ชื่อ SecureConnect.ino แล้วอัพโหลดไฟล์ SecureConnect.ino ลงบน NodeMCU

SecureConnect.ino

```
/* NETPIE ESP8266 secure microgear connection sample */  
/* It differs from the normal connection by 2 stpes */  
/* 1. Declare a client as WiFiClientSecure instead of WiFiClient. */  
/* 2. Call microgear.useTLS(true) before initial */  
/* More information visit : https://netpie.io */  
  
#include <ESP8266WiFi.h>  
#include <MicroGear.h>  
  
const char* ssid      = "WIFI_SSID";           //change this to your SSID
```

```

const char* password = "WIFI_KEY";           //change this to your WIFI
password

#define APPID    "YOURAPPID"      //change this to your appid
#define KEY      "YOURKEY"        //change this to your key
#define SECRET   "YOURSECRET"     //change this to your secret
#define ALIAS    "esp8266tls"

/* 1. Declare a client as WiFiClientSecure instead of WiFiClient. */
WiFiClientSecure client;

int timer = 0;
MicroGear microgear(client);

void onMsghandler(char *topic, uint8_t* msg, unsigned int msglen) {
    Serial.print("Incoming message -->");

    msg[msglen] = '\0';
    Serial.println((char *)msg);
}

void onFoundgear(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.print("Found new member -->");

    for (int i=0; i<msglen; i++)
        Serial.print((char)msg[i]);
    Serial.println();
}

void onLostgear(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.print("Lost member -->");

    for (int i=0; i<msglen; i++)
        Serial.print((char)msg[i]);
    Serial.println();
}

void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Securely connected to NETPIE...");

    microgear.setAlias(ALIAS);
}

void setup() {

```

```

/* Event listener */

microgear.on(MESSAGE,onMsgHandler);
microgear.on(PRESENT,onFoundgear);
microgear.on(ABSENT,onLostgear);
microgear.on(CONNECTED,onConnected);

Serial.begin(115200);
Serial.println("Starting...");

if (WiFi.begin(ssid, password)) {

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
}

Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

/* 2. Call microgear.useTLS(true) before initial */
microgear.useTLS(true);
microgear.init(KEY,SECRET,ALIAS);
microgear.connect(APPID);
}

void loop() {
    if (microgear.connected()) {
        Serial.println("connected");
        microgear.loop();
        if (timer >= 1000) {
            Serial.println("Publish...");
            microgear.chat(ALIAS,"Hello");
            timer = 0;
        }
        else timer += 100;
    }
    else {
        Serial.println("connection lost, reconnect...");
```

```

        if (timer >= 5000) {
            microgear.connect(APPID);
            timer = 0;
        }
        else timer += 100;
    }
    delay(100);
}

```

เปิดไฟล์ switch.html จาก Lab 4.2 เนื่องจาก HTML5 Microgear จะใช้ TLS เป็นค่า default อุปกรณ์ดังนั้นหากเราไม่แก้ไขอะไร HTML5 จะส่งค่าเปิดปิดไฟ ผ่านการเชื่อมต่อแบบเข้ารหัส หากต้องการยกเลิกการใช้ TLS สามารถเรียกฟังก์ชัน microgear.useTLS(false) ก่อนการเรียก microgear.connect(APPID) และบันทึกไฟล์ใหม่ชื่อ UnsecureSwitch.html ดังตัวอย่างด้านล่าง

UnsecureSwitch.html

```

<script src="https://cdn.netpie.io/microgear.js"></script>
<script>

const APPID = "YOURAPPID";
const KEY = "YOURKEY";
const SECRET = "YOURSECRET";

const ALIAS = "switch";

var microgear = Microgear.create({
    key: KEY,
    secret: SECRET,
    alias : ALIAS
});
function toggle() {
    if(document.getElementById("button").innerText=="off") {
        microgear.chat('pieled','1');
    }
    else{
        microgear.chat('pieled','0');
    }
}

microgear.on('message',function(topic,msg) {
    document.getElementById("data").innerHTML = msg;
}

```

```

if(msg=="1") {
    document.getElementById("button").innerText="on";
} else if(msg=="0") {
    document.getElementById("button").innerText="off";
}
});

microgear.on('connected', function() {
    microgear.setAlias(ALIAS);
    document.getElementById("data").innerHTML = "Now I am connected with
netpie...";
});

//add this line if you don't want a secure connection
microgear.useTLS(false);
microgear.connect(APPID);

```

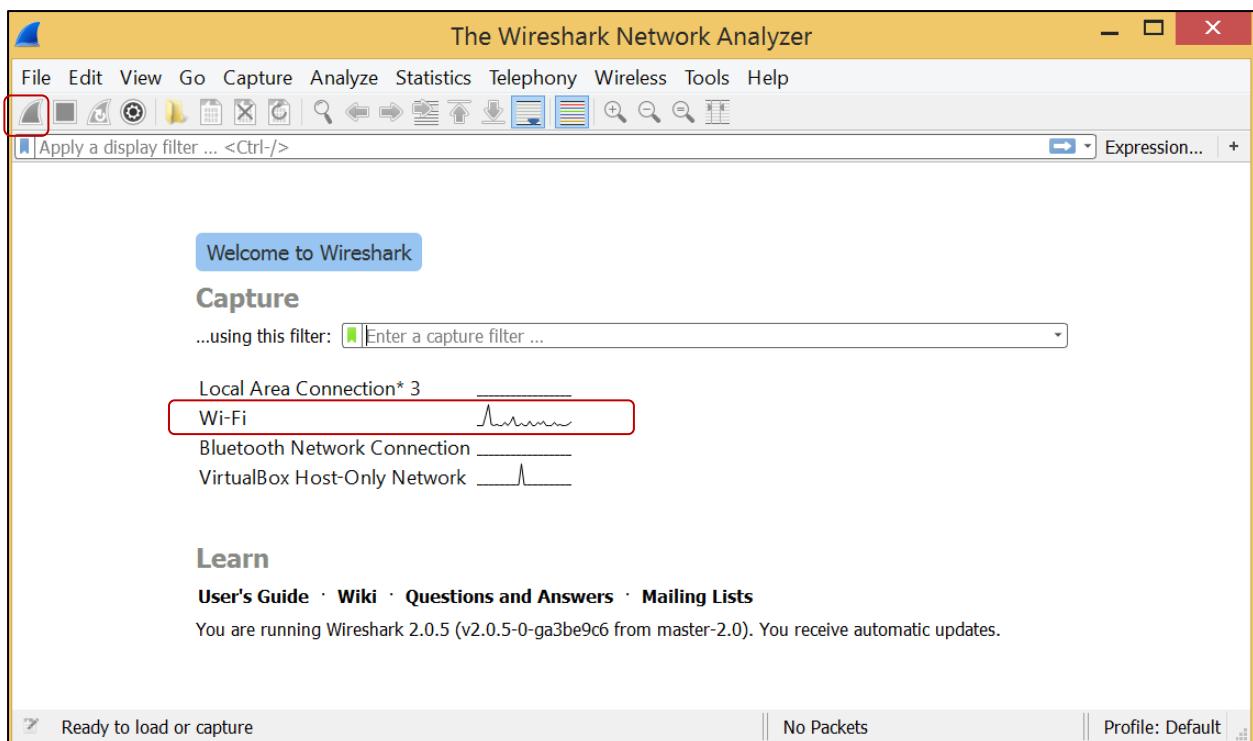
</script>

```

<div id="data">_____</div>
<center>
<button onclick="toggle()" id="button">off</button>
</center>

```

ติดตั้งและใช้งานโปรแกรม Wireshark (<https://www.wireshark.org/download.html>) ซึ่งเป็นโปรแกรมดักจับแพ็กเกตในเครือข่าย (ในขั้นตอนการติดตั้ง Wizard จะถามว่าให้ติดตั้ง libpcap ด้วยหรือไม่ ให้ตอบตกลง) เมื่อเริ่มใช้งานโปรแกรมจะเห็นหน้าต่างในภาพ ให้คลิกเลือกที่ Network Interface ที่ต้องการตักจับข้อมูล (กรณีในภาพคือ Wi-Fi) และคลิกที่ไอคอนรูปคิริบูลาม เพื่อเริ่มตักจับข้อมูล วิธีสังเกตว่าควรเลือก Network Interface ได้ให้เลือก Interface ที่มีเส้นกราฟขยับ (หมายถึงมีข้อมูลวิ่งเข้าออก)



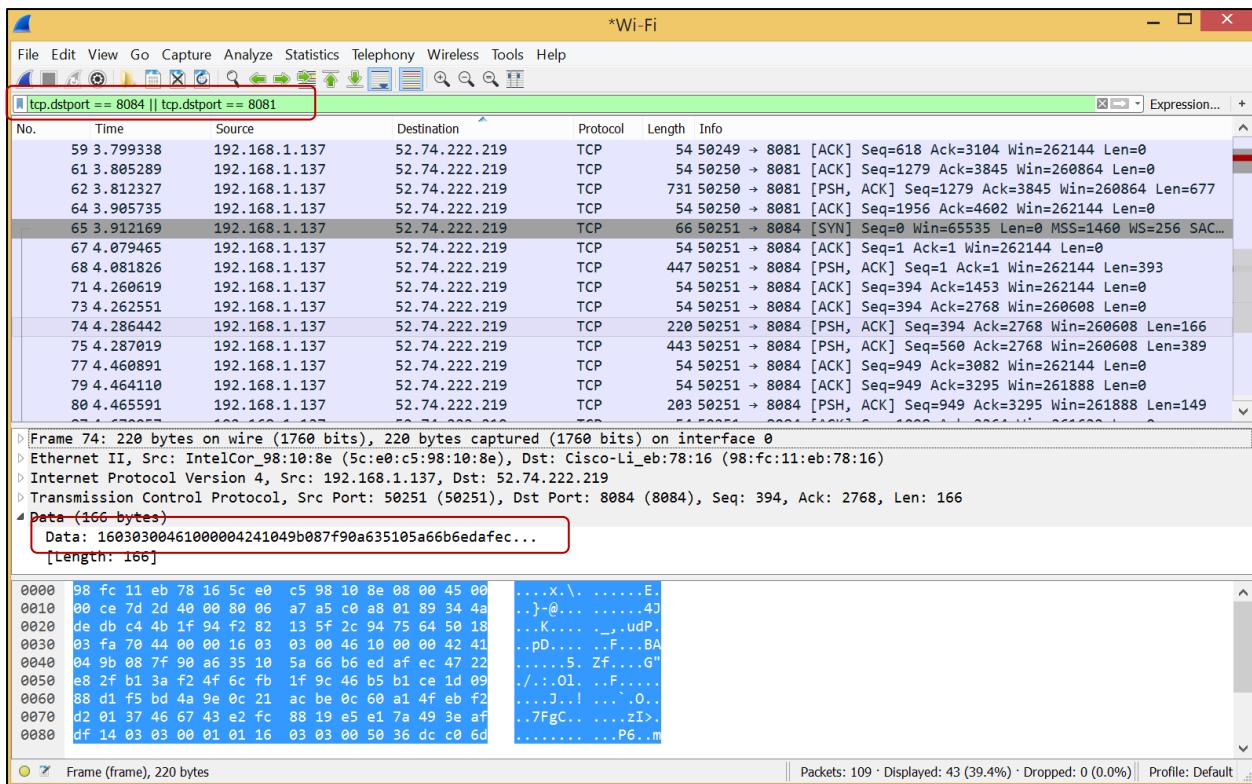
รูปที่ 8.1 หน้าจอโปรแกรม Wireshark

เนื่องจากเราสามารถดักจับได้เฉพาะข้อมูลที่วิ่งเข้าออกคอมพิวเตอร์ของเรา ดังนั้นเราจะทดลองดักจับข้อมูลที่เกิดจาก HTML5 Microgear โดยเปิดไฟล์ switch.html ทดลองกดปุ่ม ON OFF จากนั้นหยุดการดักจับข้อมูลที่ Wireshark โดยกดที่ไอคอนสีเหลืองๆ แล้วลองไล่ดู Packet ที่เกิดจากไฟล์ switch.html วิ่งผ่านคอมพิวเตอร์ของเรา

เนื่องจากมี Packet ที่เกิดจากหลายแอปพลิเคชันบนคอมพิวเตอร์ วิธีเลือกคุณลักษณะของ Destination Port ในกรณีที่ใช้ TLS HTML5 Microgear จะสื่อสารกับ NETPIE โดยใช้ Destination Port 8081 และ 8084

 วิธีเลือกคุณลักษณะแพ็คเกตที่สนใจ คือการกำหนดตัวกรอง ใส่ลงในช่องด้านบนที่เขียนว่า "Apply a display filter ... <Ctrl-/>" ตัวอย่างเช่น	
<code>tcp.dstport == 8084 tcp.dstport == 8081</code>	Filter โดยระบุ Destination Port
<code>ip.src_host==192.168.1.137</code>	Filter โดยระบุ Source IP address
<code>ip.addr==192.168.1.137</code>	Filter โดยระบุ IP address ของคุณท่าน

จากภาพจะพบว่าเราไม่สามารถอ่านข้อมูลที่วิ่งเข้าและออกจาก HTML5 Microgear



รูปที่ 8.2 หน้าจอ Wireshark หลังเรียกใช้ TLS

ทดลองเพิ่มเติม

ใช้ไฟล์ UnsecureSwitch.html แล้วทดสอบดูว่าสามารถเห็นข้อความที่ Chat ระหว่าง Switch และ NodeMCU หรือไม่


 การใช้ TLS ในกรณีนี้เป็นการเข้ารหัสการสื่อสารระหว่างอุปกรณ์ใดๆ ที่มี Microgear กับแพลตฟอร์ม NETPIE เท่านั้นไม่ใช่การเข้ารหัสแบบ End-to-end โดยตรงระหว่างอุปกรณ์สองตัว

ภาคผนวก NETPIE MICROGEAR REFERENCE GUIDE

สำหรับรายละเอียดล่าสุดของ



Microgear

ทุกตัวท่านสามารถดูได้จากเว็บไซต์

<https://github.com/netpieio>

1. ESP8266-ARDUINO MICROGEAR

ESP8266 Arduino Microgear คือ Client Library ที่ทำหน้าที่เป็นตัวกลางในการเชื่อมต่อ ESP8266 เข้ากับบริการของ NETPIE

1.1 ความเข้ากันได้

ทางทีมพัฒนาได้ทำการทดสอบพบว่า Library สามารถใช้ได้กับคุปกรณ์ต่อไปนี้ (อาจมีมากกว่านี้)

- ESP8266-01, 07, 12E, 12F
- NodeMCU v1, v2, V3
- Espresso Lite v2.0

1.2 พอร์ตสื่อสาร

ESP8266 ใช้พอร์ตสื่อสารต่อไปนี้ หากพบปัญหาการใช้งาน กรุณาระบุว่า Port ต่อไปนี้ได้รับอนุญาตให้เข้าถึง NETPIE.

- Non-TLS mode : 8080 and 1883 (ค่า default)
- TLS mode : 8081 and 8883 (อยู่ระหว่างทดสอบ)

1.3 การติดตั้ง

- ดาวน์โหลด Arduino IDE 1.6.9 หรือใหม่กว่า จาก <https://www.arduino.cc/en/Main/Software>
- หลังจากติดตั้งเสร็จ เปิด Preferences

- ใส่ข้อความ http://arduino.esp8266.com/stable/package_esp8266com_index.json ลงในช่อง Additional Board Manager URLs
- เปิด Boards Manager เมนู Tools ค้นหาคำว่า esp8266 และคลิก Install
- ในเมนู Tools จะมีบอร์ด ESP8266 ชนิดต่างๆเพิ่มขึ้นมา เลือกให้ตรงกับชนิดของบอร์ดที่ใช้
- ดาวน์โหลด Microgear Library จาก <https://github.com/netpieio/microgear-esp8266-arduino/archive/master.zip>
- Unzip ไฟล์ใน Folder ชื่อ Libraries ของ Arduino IDE
- รายละเอียดเพิ่มเติมเกี่ยวกับ ESP8266 Arduino IDE ศึกษาได้จาก <https://github.com/esp8266/Arduino>

1.4 ข้อจำกัดที่พบ

- ไฟล์ TLS ทำงานได้บน ESP8266 SDK 2.1.0 และ 2.3.0-rc1 แต่ไม่ทำงานบนเวอร์ชัน 2.2.0

1.5 ตัวอย่างการเรียกใช้

```
/* NETPIE ESP8266 basic sample */  
/* More information visit : https://netpie.io */  
  
#include <ESP8266WiFi.h>  
#include <MicroGear.h>  
  
const char* ssid      = "WIFI_SSID";  
const char* password = "WIFI_KEY";  
  
#define APPID    "YOUR_APPID"  
#define KEY     "YOUR_KEY"  
#define SECRET  "YOUR_SECRET"  
#define ALIAS   "esp8266"  
  
WiFiClient client;  
  
int timer = 0;  
MicroGear microgear(client);  
  
/* If a new message arrives, do this */  
void onMsgHandler(char *topic, uint8_t* msg, unsigned int msglen) {
```

```

    Serial.print("Incoming message --> ");
    msg[msglen] = '\0';
    Serial.println((char *)msg);
}

void onFoundgear(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.print("Found new member --> ");
    for (int i=0; i<msglen; i++)
        Serial.print((char)msg[i]);
    Serial.println();
}

void onLostgear(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.print("Lost member --> ");
    for (int i=0; i<msglen; i++)
        Serial.print((char)msg[i]);
    Serial.println();
}

/* When a microgear is connected, do this */
void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
    /* Set the alias of this microgear ALIAS */
    microgear.setAlias(ALIAS);
}

void setup() {
    /* Add Event listeners */
    /* Call onMsghandler() when new message arraives */
    microgear.on(MESSAGE,onMsghandler);

    /* Call onFoundgear() when new gear appear */
    microgear.on(PRESENT,onFoundgear);

    /* Call onLostgear() when some gear goes offline */
    microgear.on(ABSENT,onLostgear);

    /* Call onConnected() when NETPIE connection is established */
    microgear.on(CONNECTED,onConnected);
}

```

```

Serial.begin(115200);
Serial.println("Starting...");

/* Initial WIFI, this is just a basic method to configure WIFI on
ESP8266.
*/
/* You may want to use other method that is more complicated, but
provide better user experience */

if (WiFi.begin(ssid, password)) {
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
}

Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

/* Initial with KEY, SECRET and also set the ALIAS here */
microgear.init(KEY,SECRET,ALIAS);

/* connect to NETPIE to a specific APPID */
microgear.connect(APPID);
}

void loop() {
    /* To check if the microgear is still connected */
    if (microgear.connected()) {
        Serial.println("connected");

        /* Call this method regularly otherwise the connection may be lost
*/
        microgear.loop();

        if (timer >= 1000) {
            Serial.println("Publish...");
            /* Chat with the microgear named ALIAS which is myself */
            microgear.chat(ALIAS,"Hello");
            timer = 0;
        }
    }
}

```

```

        }
        else timer += 100;
    }
    else {
        Serial.println("connection lost, reconnect...");

        if (timer >= 5000) {
            microgear.connect(APPID);
            timer = 0;
        }
        else timer += 100;
    }
    delay(100);
}

```

1.6 การใช้งาน LIBRARY

Initial library ด้วยคำสั่ง

`int MicroGear::init(char* key, char* secret [,char* alias])`

arguments

- `key` (string) - ใช้ในการอ้างอิงตัวตนของ Microgear
- `secret` (string) - เป็น Secret ของ Key ซึ่งจะใช้ประกอบในกระบวนการการยืนยันตัวตน
- `alias` (string) - เป็นการระบุชื่อเล่น (ตามข้างหน้า) ของอุปกรณ์

```
microgear.init("sXfqDcXHzbFXiLk", "DNonzg2ivwS8ceksykGntrfQjxbL98",
"myplant");
```

`void MicroGear::on(unsigned char event, void (* callback)(char, uint8_t,unsigned int))`

เพิ่มฟังก์ชันที่ตอบสนองต่อ Event

arguments

- `event` - ชื่อ Event ได้แก่ MESSAGE, CONNECTED, PRESENT, ABSENT

- *callback* - พังก์ชัน Callback เมื่อเกิด Event
-

`bool MicroGear::connect(char* appid)`

เข้ามายัง NETPIE Platform ถ้าเข้ามายังสำหรับ จะมี Event ชื่อ CONNECTED เกิดขึ้น ค่าที่ส่งคืนมาจากการ connect มีดังนี้

- *NETPIECLIENT_CONNECTED* - การเข้ามายังสำหรับ
- *NETPIECLIENT_NOTCONNECTED* - การเข้ามายังล้มเหลว เช่นมีปัญหาเรื่องเครือข่าย
- *NETPIECLIENT_TOKENERROR* - ไม่ได้รับ Access Token อาจเป็นเพราะ Appid, Key หรือ Secret ไม่ถูกต้อง

arguments

- *Appid* - AppID
-

`bool MicroGear::connected()`

ส่งค่าสถานะการเข้ามายัง เป็น TRUE หากกำลังเข้ามายังอยู่

`void MicroGear::useTLS(bool* enabled)`

จะบูรณาแบบของทางการสื่อสารว่าต้องการสื่อสารแบบเข้ารหัสโดยใช้ TLS หรือไม่ (ค่า default คือไม่ใช้ TLS) เรียกใช้พังก์ชันนี้ก่อนสร้างการเข้ามายังด้วยพังก์ชัน connect

arguments

- *Enabled* – ตั้งค่า TRUE เพื่อใช้ TLS
-

```
void MicroGear::setAlias(char* alias)
```

Microgear สามารถตั้งชื่อของตัวเองได้ ซึ่งสามารถใช้เป็นชื่อให้คนอื่นเรียกในการใช้ฟังก์ชัน chat() และชื่อที่ตั้งในได้ด้วยไปปรากฏบนหน้าจัดการ Key บนเว็บ netpie.io อย่างอัตโนมัติ

arguments

- *alias* - ชื่อเล่นของ Microgear นี้
-

```
bool MicroGear::chat(char* target, char* message)
```

```
bool MicroGear::chat(char* target, int message)
```

```
bool MicroGear::chat(char* target, double message)
```

```
bool MicroGear::chat(char* target, double, int decimal)
```

```
bool MicroGear::chat(char* target, String message)
```

arguments

- *target* - ชื่อของ Microgear ที่ต้องการจะส่งข้อความไปถึง
 - *decimal* - จำนวนตำแหน่งหลังจุดทศนิยม
 - *message* - ข้อความ
-

```
bool MicroGear::publish(char* topic, char* message [, bool retained])
```

```
bool MicroGear::publish(char* topic, double message [, bool retained])
```

```
bool MicroGear::publish(char* topic, double message, int decimal [, bool retained])
```

```
bool MicroGear::publish(char* topic, int message [, bool retained])
```

```
bool MicroGear::publish(char* topic, String message [, bool retained])
```

ในกรณีที่ต้องการส่งข้อความแบบไม่เฉพาะเจาะจงผู้รับ สามารถใช้ฟังก์ชัน Publish ไปยัง Topic ที่กำหนดได้ ซึ่งจะมีแต่ Microgear ที่ Subscribe Topic นี้เท่านั้น ที่จะได้รับข้อความ

arguments

- *topic* - ชื่อของ Topic ที่ต้องการจะส่งข้อความไปถึง

- *message* - ข้อความ
 - *decimal* - จำนวนตำแหน่งหลังจุดทศนิยม
 - *retained* - ให้ Retain ข้อความไว้หรือไม่ค่า Default เป็น false (optional)
-

`void MicroGear::subscribe(char* topic)`

Microgear อาจจะมีความสนใจใน topic ใดเป็นการเฉพาะ เราสามารถใช้ฟังก์ชัน `subscribe()` ในการบอกรับ Message ของ Topic นั้นได้ และหาก Topic นั้นเคยมีการ Retain ข้อความไว้ Microgear จะได้รับข้อความนั้นทุกครั้งที่ Subscribe

arguments

- *topic* - ชื่อของ Topic ที่ต้องการจะบอกรับข้อความ
-

`void MicroGear::unsubscribe(char* topic)`

ยกเลิกการ Subscribe

arguments

- *topic* - ชื่อของ Topic ที่ต้องการจะยกเลิก
-

`void microgear.writeFeed(feedid, datajson [, apikey])`

เขียนข้อมูลลง Feed Storage

arguments

- *feed* (string) - ชื่อของ feed ที่ต้องการจะเขียนข้อมูล
- *datajson* (string) - ข้อมูลที่จะบันทึก ในรูปแบบ JSON

- *apikey* (string) – API Key สำหรับตรวจสอบสิทธิ หากไม่กำหนด จะใช้ Default API Key ของ feed ที่ให้สิทธิ์ไว้กับ AppID

```
microgear.writeFeed("homesensor", {temp:25.7,humid:62.8,light:8.5});
```

```
void MicroGear::resetToken()
```

ส่งคำสั่ง Revoke Token ไปยัง NETPIE และลบ Token ออกจาก Cache ส่งผลให้ Microgear ต้องขอ Token ใหม่ในการเชื่อมต่อครั้งต่อไป

```
void MicroGear::loop()
```

Method นี้ควรถูกเรียกใน arduino loop() เป็นระยะๆ เพื่อที่ Microgear Library สามารถ Keep alive connection alive และจัดการกับ Message ที่เข้ามา

2. ARDUINO-ETHERNET MICROGEAR

Arduino Ethernet Microgear คือ Client Library ที่ทำหน้าที่เป็นตัวกลางในการเชื่อมต่อ Arduino Board ที่ใช้ Ethernet Shield เข้ากับบริการของ NETPIE

2.1 ความเข้ากันได้

Library สามารถใช้ได้กับ Arduino Mega 2560 และ Ethernet Shield

2.2 ตัวอย่างการเรียกใช้

```
#include <AuthClient.h>
#include <MicroGear.h>
#include <MQTTClient.h>
#include <PubSubClient.h>
#include <SHA1.h>
#include <Arduino.h>
#include <SPI.h>
```

```

#include <Ethernet.h>
#include <EEPROM.h>
#include <MicroGear.h>

#define APPID      "YOUR_APPID"
#define GEARKEY    "YOUR_KEY"
#define GEARSECRET "YOUR_SECRET"
#define SCOPE      ""

EthernetClient client;
AuthClient *authclient;

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
MicroGear microgear(client);
int timer = 0;

void onMsghandler(char *topic, uint8_t* msg, unsigned int msglen) {
    Serial.print("Incoming message --> ");
    msg[msglen] = '\0';
    Serial.println((char *)msg);
}

void onFoundgear(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.print("Found new member --> ");
    for (int i=0; i<msglen; i++)
        Serial.print((char)msg[i]);
    Serial.println();
}

void onLostgear(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.print("Lost member --> ");
    for (int i=0; i<msglen; i++)
        Serial.print((char)msg[i]);
    Serial.println();
}

void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
    Serial.println("Connected to NETPIE...");
    microgear.setAlias("mygear");
}

```

```

void setup() {
    Serial.begin(9600);
    Serial.println("Starting...");

    microgear.on(MESSAGE,onMsghandler);
    microgear.on(PRESENT,onFoundgear);
    microgear.on(ABSENT,onLostgear);
    microgear.on(CONNECTED,onConnected);

    if (Ethernet.begin(mac)) {
        Serial.println(Ethernet.localIP());
        microgear.resetToken();
        microgear.init(GEARKEY,GEARSECRET,SCOPE);
        microgear.connect(APPID);
    }
}

void loop() {
    if (microgear.connected()) {
        Serial.println("connected");
        microgear.loop();
        if (timer >= 1000) {
            microgear.chat("mygear","Hello");
            timer = 0;
        }
        else timer += 100;
    }
    else {
        Serial.println("connection lost, reconnect...");
        if (timer >= 5000) {
            microgear.connect(APPID);
            timer = 0;
        }
        else timer += 100;
    }
    delay(100);
}

```

2.3 การใช้งาน LIBRARY

Initial library ด้วยคำสั่ง

int MicroGear::init(char* *key*, char* *secret* [,char* *alias*])

arguments

key (string) - ใช้ในการอ้างอิงตัวตนของ Microgear

secret (string) - เป็น Secret ของ Key ซึ่งจะใช้ประกอบในกระบวนการยืนยันตัวตน

alias (string) – เป็นการระบุชื่อของอุปกรณ์

```
microgear.init("sXfqDcXHzbFXiLk", "DNonzg2ivwS8ceksykGntrfQjxbL98",
"myplant");
```

void MicroGear::on(unsigned char *event*, void (* *callback*)(char, uint8_t,unsigned int))

เพิ่มฟังก์ชันที่ตอบสนองต่อ Event

arguments

event - ชื่อ Event ได้แก่ MESSAGE, CONNECTED, PRESENT, ABSENT

callback - ฟังก์ชัน Callback

bool MicroGear::connect(char* *appid*)

เชื่อมต่อกับ NETPIE Platform ถ้าเชื่อมต่อสำเร็จ จะมี Event ชื่อ CONNECTED เกิดขึ้น

arguments

appid - AppID

bool MicroGear::connected(char* *appid*)

ส่งค่าสถานะการเชื่อมต่อ เป็น true หากกำลังเชื่อมต่ออยู่

arguments

appid - AppID

```
void MicroGear::setAlias(char* alias)
```

Microgear สามารถตั้งชื่อของตัวเองได้ ซึ่งสามารถใช้เป็นชื่อให้คนอื่นเรียกในการใช้ฟังก์ชัน chat() และชื่อที่ตั้งในได้ด้วยไปปรากฏบนหน้าจัดการ Key บนเว็บ netpie.io อย่างอัตโนมัติ

arguments

alias - ชื่อเล่นของ microgear นี้

```
bool MicroGear::chat(char* target, char* message)
```

```
bool MicroGear::chat(char* target, int message)
```

```
bool MicroGear::chat(char* target, double message)
```

```
bool MicroGear::chat(char* target, double, int decimal)
```

```
bool MicroGear::chat(char* target, String message)
```

arguments

target - ชื่อของ Microgear ที่ต้องการจะส่งข้อความไปถึง

decimal - จำนวนตำแหน่งหลังจุดทศนิยม

message - ข้อความ

```
bool MicroGear::publish(char* topic, char* message [, bool retained])
```

```
bool MicroGear::publish(char* topic, double message [, bool retained])
```

```
bool MicroGear::publish(char* topic, double message, int decimal [, bool retained])
```

```
bool MicroGear::publish(char* topic, int message [, bool retained])
```

```
bool MicroGear::publish(char* topic, String message [, bool retained])
```

ในกรณีที่ต้องการส่งข้อความแบบไม่เจาะจงผู้รับ สามารถใช้ฟังก์ชัน Publish ไปยัง Topic ที่กำหนดได้ ซึ่งจะมีแต่ Microgear ที่ Subscribe Topic นี้เท่านั้น ที่จะได้รับข้อความ

arguments

topic - ชื่อของ Topic ที่ต้องการจะส่งข้อความไปถึง

message - ข้อความ

decimal - จำนวนตำแหน่งหลังจุดทศนิยม

retained - ให้ Retain ข้อความไว้หรือไม่ค่า default เป็น false (optional)

```
void MicroGear::subscribe(char* topic)
```

microgear อาจจะมีความสนใจใน Topic ใดเป็นการเฉพาะ เราสามารถใช้ฟังก์ชัน subscribe() ในการบอกรับ Message ของ Topic นั้นได้ และหาก Topic นั้นเคยมีการ Retain ข้อมูลไว้ Microgear จะได้รับข้อมูลนั้นทุกครั้งที่ Subscribe Topic

arguments

topic - ชื่อของ Topic ที่ต้องการจะบอกรับข้อมูล

```
void MicroGear::unsubscribe(char* topic)
```

ยกเลิกการ Subscribe

arguments

topic - ชื่อของ Topic ที่ต้องการจะยกเลิก

```
void microgear.writeFeed (feedid, datajson [, apikey])
```

เขียนข้อมูลลง Feed Storage

arguments

- *feed* (string) - ชื่อของ feed ที่ต้องการจะเขียนข้อมูล
- *datajson* (string) - ข้อมูลที่จะบันทึก ในรูปแบบ JSON
- *apikey*(string) – API Key สำหรับตรวจสอบสิทธิ หากไม่กำหนด จะใช้ Default API Key ของ feed ที่ให้สิทธิ์ไว้กับ AppID

```
microgear.writeFeed("homesensor", {temp:25.7,humid:62.8,light:8.5});
```

```
void MicroGear::resetToken()
```

ส่งคำสั่ง Revoke Token ไปยัง NETPIE และลบ Token ออกจาก Cache ส่งผลให้ Microgear ต้องขอ Token ใหม่ในการเชื่อมต่อครั้งต่อไป

```
void MicroGear::loop()
```

Method นี้ควรถูกเรียกใน arduino loop() เป็นระยะๆ เพื่อที่ Microgear library จะได้ keep alive connection alive และจัดการกับ Message ที่เข้ามา

3. NODE.JS MICROGEAR

microgear-nodejs คือ Client Library ภาษา Node.js ที่ทำหน้าที่เป็นตัวกลางในการเชื่อมโยง โค้ดแอปพลิเคชันหรือ Hardware เข้ากับบริการของ NETPIE

3.1 พอร์ตสื่อสาร

หากพบปัญหาการใช้งาน กรุณาระบุว่า Port ต่อไปนี้ได้รับอนุญาตให้เข้าถึงจาก NETPIE

- Non-TLS mode : 8080 and 1883 (ค่า default)
- TLS mode : 8081 and 8883

3.2 การติดตั้ง

```
npm install microgear
```

3.3 ตัวอย่างการเรียกใช้

```
var MicroGear = require('microgear');

const APPID   = <APPID>;
const KEY     = <KEY>;
const SECRET = <SECRET>

var microgear = MicroGear.create({
    key : KEY,
    secret : SECRET
});
```

```

microgear.on('connected', function() {
    console.log('Connected...');

    microgear.setAlias("mygear");
    setInterval(function() {
        microgear.chat('mygear', 'Hello world.');
    },1000);
});

microgear.on('message', function(topic,body) {
    console.log('incoming : '+topic+' : '+body);
});

microgear.on('closed', function() {
    console.log('Closed...');
});

microgear.connect(APPID);

```

3.4 การใช้งาน LIBRARY

microgear create (config)

arguments

- config เป็น JSON Object ที่ที่มี Attribute ดังนี้
 - key (string)- ใช้ในการอ้างอิงตัวตนของ Microgear
 - secret (string) - เป็น Secret ของ Key ซึ่งจะใช้ประกอบในกระบวนการการยืนยันตัวตน
 - alias (string) – เป็นการตั้งชื่อเล่น จะใส่ที่นี่หรือเรียกฟังก์ชัน setAlias() ที่หลังๆได้

```

var microgear = MicroGear.create({
    gearkey : "sXfqDcXHzbFXiLk",
    gearsecret : "DNonzg2ivwS8ceksykGntrfQjxbL98",
    alias : "mygear"
});

```

3.4.1 Microgear

void microgear.connect (appid, callback)

arguments

- *Appid* (string) – คือกลุ่มของแอปพลิเคชัน Microgear จะทำการเชื่อมต่อ

```
microgear.connect("happyfarm");
```

void microgear.setAlias (*gearalias*)

Microgear สามารถตั้งนามแฝงของตัวเองได้ ซึ่งสามารถใช้เป็นชื่อให้คนอื่นเรียกในการใช้ฟังก์ชัน `chat()` และชื่อที่ตั้งในโค้ด จะไปปรากฏบนหน้าจัดการ Key บนเว็บ netpie.io อย่างอัตโนมัติ

arguments

- *gearalias* (string) - ชื่อของ Microgear นี้

```
microgear.setAlias("plant");
```

void microgear.chat (*gearname*, *message*)

arguments

- *gearname* (string) - ชื่อของ Microgear ที่ต้องการจะส่งข้อความไปถึง
- *message* (string) - ข้อความ

```
microgear.chat("valve","I need water");
```

void microgear.publish (*topic*, *message*, [*retained*]) ในกรณีที่ต้องการส่งข้อความแบบไม่เจาะจงผู้รับ สามารถใช้ฟังก์ชัน Publish ไปยัง Topic ที่กำหนดได้ ซึ่งจะมีแต่ Microgear ที่ Subscribe Topic นี้เท่านั้น ที่จะได้รับข้อความ

arguments

- *topic* (string) - ชื่อของ Topic ที่ต้องการจะส่งข้อความไปถึง
- *message* (string) - ข้อความ
- *retained* (boolean) - ให้ Retain ข้อความไว้หรือไม่ ค่า Default เป็น False

```
microgear.publish("/outdoor/temp", "28.5") ;  
microgear.publish("/outdoor/humid", "56", true) ;
```

void microgear.subscribe (*topic*)

Microgear อาจจะมีความสนใจใน Topic ใดเป็นการเฉพาะ เราสามารถใช้ฟังก์ชัน `Subscribe()` ในการบอกรับ Message ของ Topic นั้นได้ และหาก Topic นั้นเคยมีการ Retain ข้อมูลไว้ Microgear จะได้รับข้อมูลนั้นทุกครั้งที่ Subscribe Topic

arguments

- *topic* (string) - ชื่อของ Topic ที่ต้องการจะบอกรับข้อมูล

```
microgear.subscribe("/outdoor/temp") ;
```

void microgear.unsubscribe (*topic*) ยกเลิกการ Subscribe

arguments

- *topic* (string) - ชื่อของ Topic ที่ต้องการจะยกเลิก

```
microgear.unsubscribe("/outdoor/temp") ;
```

void `microgear.setCachePath (path)` โดยปกติแล้ว microgear จะเก็บไฟล์ token cache ใน directory เดียวกับ application โดยตั้งชื่อไฟล์เป็น `microgear-.cache` เราสามารถกำหนด path ของ token cache file ใหม่ด้วยฟังก์ชัน `setCachePath()` ซึ่งอาจจำเป็นต้องใช้ หากในไฟล์ Node.js application เดียวกัน มีการสร้าง microgear มากกว่าหนึ่งตัว

arguments

- *path* (string) - path ของไฟล์ cache

```
microgear.setCachePath('microgear-g1.cache') ;
```

```
void microgear.writeFeed (feedid, datajson [, apikey])
```

เขียนข้อมูลลง Feed Storage

arguments

- *feed* (string) - ชื่อของ feed ที่ต้องการจะเขียนข้อมูล
- *datajson* (string) - ข้อมูลที่จะบันทึก ในรูปแบบ JSON
- *apikey* (string) – API Key สำหรับตรวจสอบสิทธิ หากไม่กำหนด จะใช้ Default API Key ของ feed ที่ให้สิทธิ์ไว้กับ AppID

```
microgear.writeFeed("homesensor", {temp:25.7,humid:62.8,light:8.5});
```

```
void microgear.resetToken (callback)
```

ส่งคำสั่ง Revoke Token ไปยัง NETPIE และลบ Token ออกจาก Cache ส่งผลให้ Microgear ต้องขอ Token ใหม่ในการเข้ามายังครั้งต่อไป

arguments

- *callback* (function) – พิงก์ชันที่จะถูกเรียกเมื่อการรีเซ็ต Token เสร็จสิ้น

```
microgear.resetToken(function(result){  
});
```

เนื่องจาก resettoken() เป็น Asynchronous Function หากต้องการ Connect หลังจาก resettoken ต้องเขียนโค้ดในลักษณะนี้

```
microgear.resetToken(function(result){  
    microgear.connect(APPID);  
});
```

```
void microgear.useTLS (tlsmode)
```

เลือกใช้หรือไม่ใช้การเข้ารหัสแบบ TLS. โดยค่าเริ่มต้น microgear-nodejs จะไม่ใช้ TLS

arguments

- *tlsmode* (boolean) - true เมื่อต้องการใช้ TLS

```
microgear.useTLS(false);
```

3.4.2 Events

แอปพลิเคชันที่รันบน Microgear จะมีการทำงานในแบบ Event Driven คือเป็นการทำงานตอบสนองต่อ Event ต่างๆ ด้วยการเขียน Callback Function ขึ้นมารองรับในลักษณะนี้

```
void microgear.on (event, callback)
```

arguments

- *event* (string) - ชื่อ Event
- *callback* (function) - Callback Function

NETPIE Platform เวอร์ชันปัจจุบัน มี Event ดังต่อไปนี้

Event: 'connected' เกิดขึ้นเมื่อ Microgear Library เชื่อมต่อกับ Platform สำเร็จ

```
microgear.on("connected", function() {
    console.log("connected");
});
```

Event: 'closed' เกิดขึ้นเมื่อ Microgear Library ตัดการเชื่อมต่อกับ Platform

```
microgear.on("closed", function() {
    console.log("closed");
});
```

Event: 'error' เป็น Event ที่เกิดมี Error ขึ้นภายใน Microgear

```
microgear.on("error", function(err) {
    console.log("Error: "+err);
});
```

Event: 'warning' เป็น Event ที่เกิดมีเหตุการณ์บางอย่างเกิดขึ้น และมีการเตือนให้ทราบ

```
microgear.on("warning", function(msg) {
    console.log("Connection rejected: "+msg);
});
```

Event: 'info' เป็น Event ที่เกิดมีเหตุการณ์บางอย่างเกิดขึ้นภายใน Microgear

```
microgear.on("info", function(msg) {
    console.log("Connection rejected: "+msg);
});
```

Event: 'message' เมื่อมี Message เข้ามา จะเกิด Event นี้ขึ้น พร้อมกับส่งผ่านข้อมูลเกี่ยวกับ Message นั้นมาทาง Argument ของ Callback Function

```
microgear.on("message", function(topic,msg) {  
    console.log("Incoming message: "+msg);  
});
```

Event: 'present' Event นี้จะเกิดขึ้นเมื่อมี Microgear ใน Appid เดียวกัน Online เข้ามายื่นต่อ NETPIE

```
microgear.on("present", function(event) {  
    console.log("New friend found: "+event.gearkey);  
});
```

Event: 'absent' Event นี้จะเกิดขึ้นเมื่อมี Microgear ใน Appid เดียวกัน Offline หายไป

```
microgear.on("absent", function(event) {  
    console.log("Friend lost: "+event.gearkey);  
});
```

4. HTML5 MICROGEAR

microgear-html5 คือ Client Library ของ NETPIE ที่จะเปลี่ยน Web Browser ให้เป็น Microgear เพื่อสื่อสารกับ Microgear ใน Platform ชื่นๆ เช่น Arduino, Raspberry Pi Library นี้สามารถนำไปพัฒนา IoT Console หรือ Mobile Application ได้โดยการเขียนโปรแกรมภาษา HTML/JavaScript

4.1 การรองรับ

- Chrome
- Firefox
- Opera
- Safari
- Internet Explorer
- Edge

4.2 พอร์ตสื่อสาร

หากพบปัญหาการใช้งาน กรุณาระบุว่า Port ต่อไปนี้ได้รับอนุญาตให้เข้าถึงจาก Network ของคุณ

- TLS mode : 8081 and 8084 (ค่า default)
- Non-TLS mode : 8080 and 8083

4.3 การติดตั้ง

ดาวน์โหลด microgear.js จาก <https://raw.githubusercontent.com/netpieio/microgear-html5/master/microgear.js>

หรือเรียกใช้เวอร์ชันล่าสุดจาก cdn โดยใช้ Tag

```
<script src="https://cdn.netpie.io/microgear.js"></script>
```

4.4 ตัวอย่างการเรียกใช้

```
<script src="https://cdn.netpie.io/microgear.js"></script>

<script>

  const APPID      = "YOUR_APPID";
  const KEY        = "YOUR_KEY";
  const SECRET     = "YOUR_SECRET";

  var microgear = Microgear.create({
    key: KEY,
    secret: SECRET,
    alias : "myhtml"           /* optional */
  });

  microgear.on('message',function(topic,msg) {
    document.getElementById("data").innerHTML = msg;
  });

  microgear.on('connected', function() {
    microgear.setAlias('htmlgear');    /* alias can be renamed anytime
with this function */

    document.getElementById("data").innerHTML = "Now I am connected
with netpie...";
```

```

        setInterval(function() {
            microgear.chat("htmlgear","Hello from myself at "+Date.now());
            },5000);
        });

        microgear.on('present', function(event) {
            console.log(event);
        });

        microgear.on('absent', function(event) {
            console.log(event);
        });

        microgear.connect(APPID);
    </script>

<div id="data">_____</div>

```

4.5 การใช้งาน LIBRARY

microgear create (config)

arguments

- config เป็น JON Object ที่มี Attribute ดังนี้
 - key (string) - เป็น key สำหรับ device
 - secret (string) - เป็น Secret ของ Key ซึ่งจะใช้ประกอบในกระบวนการการยืนยันตัวตน
 - alias (string) – เป็นการตั้งชื่อเล่น จะใส่ที่นี่หรือเรียกฟังก์ชัน setAlias() ภายหลังได้

```

var microgear = MicroGear.create({
    key : "sXfqDcXHzbFXiLk",
    secret : "DNonzg2ivwS8ceksykGntrfQjxbL98",
    alias : "myhtml"
});

```

4.5.1 Microgear

void microgear.connect (appid, callback)

เข้ามายัง NETPIE โดยระบุ Appid เป้าหมาย

arguments

- *appid* (string) - คือ Application ที่ Microgear จะทำการเชื่อมต่อ

```
microgear.connect("happyfarm");
```

void microgear.setAlias(*gearalias*)

Microgear สามารถตั้งชื่อตัวเองได้ ซึ่งสามารถใช้เป็นชื่อเล่นในการใช้ฟังก์ชัน *chat()*

arguments

- *gearalias* (string) - ชื่อของ Microgear นี้

```
microgear.setAlias("plant");
```

void microgear.chat (*gearalias*, *message*)

arguments

- *gearalias*(string)- ชื่อของ Microgear ที่ต้องการจะส่งข้อความไปถึง
- *message*(string)- ข้อความ

```
microgear.chat("valve","I need water");
```

void microgear.publish (*topic*, *message*)

ในกรณีที่ต้องการส่งข้อความแบบไม่เจาะจงผู้รับ สามารถใช้ฟังก์ชัน *Publish* ไปยัง Topic ที่กำหนดได้ ซึ่งจะมีแต่ Microgear ที่ *Subscribe* ใน Topic นี้เท่านั้น ที่จะได้รับข้อความ

arguments

- *topic* (string) - ชื่อของ topic ที่ต้องการจะส่งข้อความไปถึง
- *message* (string) - ข้อความ

```
microgear.publish("/outdoor/temp", "28.5");
```

```
void microgear.subscribe (topic)
```

Microgear อาจจะมีความสนใจใน Topic ใดเป็นการเฉพาะ เราสามารถใช้ฟังก์ชัน subscribe() ในการบอกรับ Message ของ Topic นั้นได้

arguments

- *topic* (string) - ชื่อของ topic ที่ต้องการจะบอกรับข้อมูล

```
microgear.subscribe ("/outdoor/temp");
```

```
void microgear.unsubscribe (topic)
```

ยกเลิกการ Subscribe

arguments

- *topic* (string) - ชื่อของ Topic ที่ต้องการจะยกเลิก

```
microgear.unsubscribe ("/outdoor/temp");
```

```
void microgear.writeFeed (feedid, datajson [, apikey])
```

เขียนข้อมูลลง Feed Storage

arguments

- *feed* (string) - ชื่อของ feed ที่ต้องการจะเขียนข้อมูล
- *datajson* (string) - ข้อมูลที่จะบันทึก ในรูปแบบ JSON
- *apikey* (string) – API Key สำหรับตรวจสอบสิทธิ หากไม่กำหนด จะใช้ Default API Key ของ feed ที่เชื่อมต่อไว้กับ AppID

```
microgear.writeFeed ("homesensor", {temp:25.7,humid:62.8,light:8.5});
```

```
void microgear.resetToken (callback)
```

ถอนไลน์ส่งคำสั่ง Revoke Token และลบ Token ออกจาก Cache แล้วผลให้ Microgear ต้องขอ Token ใหม่ในการเชื่อมต่อครั้งต่อไป

arguments

- *callback* (function) – Callback Function ที่จะถูกเรียกเมื่อการ Reset Token เสร็จสิ้น

```
microgear.resetToken(function(result){  
});
```

เนื่องจาก resetToken() เป็น Asynchronous Function หากต้องการ Connect หลังจาก resetToken ต้องเขียนโค้ดในลักษณะนี้

```
microgear.resetToken(function(result){  
    microgear.connect(APPID);  
});
```

```
void microgear.useTLS (tlsmode)
```

Enable หรือ disable TLS โดยใน HTML5 Microgear จะใช้ TLS เป็นค่า Default

arguments

- *tlsmode* (boolean) - เป็น true หมายถึงใช้ TLS (เป็นค่า Default), false หมายถึงไม่ใช้ TLS

```
microgear.useTLS(false);
```

4.5.2 Events

Application ที่รันบน Microgear จะมีการทำงานในแบบ Event Driven คือเป็นการทำงานตอบสนองต่อ Event ต่างๆ ด้วยการเขียน Callback Function ขึ้นมารองรับในลักษณะนี้

```
void microgear.on (event, callback)
```

arguments

- *event* (string) - ชื่อ Event

- *callback (function)* - Callback Function

NETPIE Platform เวอร์ชันปัจจุบัน มี Event ดังต่อไปนี้

Event: 'connected' เกิดขึ้นเมื่อ Microgear Library เชื่อมต่อกับ NETPIE สำเร็จ

```
microgear.on("connected", function() {
    console.log("connected");
});
```

Event: 'closed' เกิดขึ้นเมื่อ Microgear Library เชื่อมต่อกับ NETPIE สำเร็จ

```
microgear.on("closed", function() {
    console.log("closed");
});
```

Event: 'error' เป็น Event ที่เกิดมี Error ขึ้นภายใน Microgear

```
microgear.on("error", function(err) {
    console.log("Error: "+err);
});
```

Event: 'message' เมื่อมี Message เข้ามา จะเกิด Event นี้ขึ้น พร้อมกับส่งผ่านข้อมูลเกี่ยวกับ Message นั้นมาทาง Argument ของ Callback Function

```
microgear.on("message", function(topic,msg) {
    console.log("Incoming message: "+msg);
});
```

Event: 'present' Event นี้จะเกิดขึ้นเมื่อมี Microgear ใน Appid เดียวกัน Online เข้ามาเชื่อมต่อ NETPIE

```
microgear.on("present", function(event) {
    console.log("New friend found: "+event.gearkey);
});
```

Event: 'absent' Event นี้จะเกิดขึ้นเมื่อมี Microgear ใน Appid เดียวกัน Offline หายไป

```
microgear.on("absent", function(event) {
    console.log("Friend lost: "+event.gearkey);
});
```

5. PYTHON MICROGEAR

microgear -python คือ Client Library ภาษา Python ที่ทำหน้าที่เป็นตัวกลางในการเชื่อมโยง Application Code หรือ Hardware เข้ากับบริการของ NETPIE Platform เพื่อการพัฒนา IoT application

5.1 การติดตั้ง

```
$ pip install microgear
```

5.2 ตัวอย่างการเรียกใช้งาน

```
import microgear.client as microgear
import time

appid = <appid>
gearkey = <key>
gearssecret = <secret>

microgear.create(gearkey,gearssecret,appid,{ 'debugmode' : True })

def connection():
    print "Now I am connected with netpie"

def subscription(topic,message):
    print topic+" "+message

def disconnect():
    print "disconnect is work"

microgear.setalias("doraemon")
microgear.on_connect = connection
microgear.on_message = subscription
microgear.on_disconnect = disconnect
microgear.subscribe("/mails")
microgear.connect(True)
```

5.3 ตัวอย่างเพิ่มเติม

5.4 การใช้งาน LIBRARY

5.4.1 Microgear

client.create(gearkey,gearsecret,appid,args):

arguments

- gearkey (string) - ใช้ในการอ้างอิงตัวตนของ Microgear
- gearkey (string) - เป็น Secret ของ Key ซึ่งจะใช้ประกอบในกระบวนการยืนยันตัวตน
- appid (string) - กลุ่มของ Application ที่ Microgear จะทำการเชื่อมต่อ
- args (dictionary) - เป็นการตั้งค่าเพิ่มเติม สำหรับ Microgear ได้แก่
 - debugmode (boolean) - แสดงข้อความในโหมด debug
 - scope (string) - กำหนด Scope ให้กับ Microgear เพื่อให้จำกัด/สิทธิ์ บางอย่างโดยมีรูปแบบดังนี้
 - [r][w]:</topic/path> - r และ w คือสิทธิ์ในการ Publish ละ Subscribe topic ดังที่ระบุตามลำดับ เช่น rw:/outdoor/temp
 - name:<gearname> - คือสิทธิ์ในการตั้งชื่อตัวเองว่า <gearname>
 - chat:<gearname> - คือสิทธิ์ในการ chat กับ <gearname>
 - alias (string) – กำหนดชื่อเล่นสำหรับ Microgear นี้ โดยจะปรากฏที่หน้า Key Management และสามารถเป็นชื่อที่ Microgear ตัวอื่นใช้สำหรับ chat() ได้

ในขั้นตอนของการสร้าง Key บนเว็บ netpie.io นักพัฒนาสามารถกำหนดสิทธิ์ขั้นพื้นฐานให้แต่ละ Key ได้อยู่แล้ว หากการสร้าง Microgear อยู่ภายใต้ขอบเขตของสิทธิ์ที่มี Token จะถูกจ่ายอัตโนมัติ และ Microgear จะสามารถเชื่อมต่อ NETPIE Platform ได้ทันที แต่หาก Scope ที่ร้องขอนั้นมากเกินกว่าสิทธิ์ที่กำหนดไว้ นักพัฒนาจะได้รับ Notification ให้พิจารณาอนุมัติ Microgear ที่เข้ามาขอเชื่อมต่อ ข้อควรระวังคือ หาก Microgear มีการกระทำการเกินกว่าสิทธิ์ที่ได้รับไป เช่น พยายามจะ Publish ไปยัง Topic ที่ตัวเองไม่มีสิทธิ์ NETPIE จะตัดการเชื่อมต่อของ Microgear โดยอัตโนมัติ

```

gearkey = <gearkey>
gearssecret = <gearssecret>
appid = <appid>

client.create(gearkey,gearssecret,appid, {'debugmode': True, 'scope':
"r:/outdoor/temp,w:/outdoor/valve,name:logger,chat:plant", 'alias':
"logger"})

```

`client.connect(will_block)`: การเชื่อมต่อ Microgear

argument

- `will_block` (boolean) - (optional) ระบุรูปแบบการเชื่อมต่อ ว่าให้มีการ Block หลังจากเรียกฟังก์ชันหรือไม่ ซึ่งจะมีค่า Default เป็น False โดยโปรแกรมจะทำงานในบรรทัดเดียวหลังจากที่ทำการ Connect แล้ว ซึ่งจะทำให้ผู้พัฒนาสามารถเขียนโปรแกรมในการติดต่อกับ NETPIE ต่อไปได้ โดยการเชื่อมต่อจะคงอยู่ตราบเท่าการทำงานของโปรแกรม เช่น

```

client.connect()
while True:
    client.chat("doraemon", "Hello world. "+str(int(time.time())))
    time.sleep(2)

```

หากต้องการให้ Library ทำการ Block หลังจาก Connect แล้ว ทำให้หลังจาก Connect แล้ว โปรแกรมหยุดอยู่ที่การทำงานร่วมกับ Platform โดยจะทำงานตามที่มีเหตุการณ์ callback (`on_*`) ที่ถูกกำหนดไว้ก่อนหน้า โดยสามารถระบุ param เป็น `True` ได้ เช่น

```
client.connect(True)
```

`client.setalias(alias)`:

กำหนดชื่อเรียกสำหรับ Microgear นี้ โดยจะปรากฏที่หน้า Key Management และสามารถเป็นชื่อที่ Microgear ตัวอื่นใช้สำหรับ `chat()` ได้

argument

- `alias` (string) - ชื่อของ Microgear นี้

```
client.setalias("python");
```

`client.chat(gearname, message):`

การส่งข้อความโดยระบุ gearname และข้อความที่ต้องการส่ง

arguments

- `gearname` (string) - ชื่อของ Microgear นี้
- `message` (string) – ข้อความ

```
client.chat("html","hello from python");
```

`client.publish(topic, message, retain):`

ในกรณีที่ต้องการส่งข้อความแบบไม่เฉพาะเจาะจงผู้รับ สามารถใช้ฟังก์ชัน Publish ไปยัง Topic ที่กำหนดได้ ซึ่งจะมีแต่ Microgear ที่ Subscribe ใน Topic นี้เท่านั้น ที่จะได้รับข้อความ

arguments

- `topic` (string) - ชื่อของ topic ที่ต้องการจะส่งข้อความไปถึง
- `message` (string) – ข้อความ
- `retain` (boolean) – ระบุค่า True ถ้าต้องการเก็บข้อความไว้ หากมีการ Subscribe Topic นี้ก็จะได้รับข้อความนี้อีก ค่า Default เป็น False หากไม่ระบุ และถ้าต้องการลบข้อความที่บันทึกไว้ให้ส่งข้อความ ซึ่งมีความยาวเป็น ""0 เพื่อล้างค่าข้อความที่ไว้ทิ้ง

```
client.publish("/outdoor/temp","28.5");
```

`client.subscribe(topic)`

Microgear จะจะมีความสนใจใน Topic ใดเป็นการเฉพาะ เราสามารถใช้ฟังก์ชัน `Subscribe()` ในการบอกรับ Message ของ Topic นั้นได้

argument

- `topic` (string) - ชื่อของ Topic ที่ต้องการบอกรับ โดยต้องขึ้นต้นด้วยเครื่องหมาย "/"

```
client.subscribe("/temp");
```

```
void microgear.writeFeed (feedid, datajson [, apikey])
```

เขียนข้อมูลลง Feed Storage

arguments

- *feed* (string) - ชื่อของ feed ที่ต้องการจะเขียนข้อมูล
- *datajson* (string) - ข้อมูลที่จะบันทึก ในรูปแบบ JSON
- *apikey* (string) – API key สำหรับตรวจสอบสิทธิ หากไม่กำหนด จะใช้ Default API Key
ของ feed ที่ให้สิทธิ์ไว้กับ AppID

```
microgear.writeFeed("homesensor", {temp:25.7,humid:62.8,light:8.5});
```

`client.resettoken()`

ใช้ในการการลบ Token ที่มีอยู่ จาก Cache และบน Platform เมื่อลบแล้ว จะเป็นจะต้องขอ Token ใหม่
ทุกครั้ง

```
client.resettoken();
```

5.4.2 Event

Application ที่รันบน Microgear จะมีการทำงานในแบบ Event Driven คือเป็นการทำงาน
ตอบสนองต่อ Event ต่างๆ ด้วยการเขียน Callback Function ขึ้นมารองรับในลักษณะฯ ดังต่อไปนี้

`client.on_connect` เกิดขึ้นเมื่อ Microgear Library เชื่อมต่อกับ Platform สำเร็จ

ค่าที่ตั้ง

- *Callback* (function) - ฟังก์ชันที่จะทำงาน เมื่อมีการ Connect เกิดขึ้น

```
def callback_connect() :  
    print "Now I am connected with netpie"  
client.on_connect = callback_connect
```

client.on_disconnect เกิดขึ้นเมื่อ Microgear Library ตัดการเชื่อมต่อกับ Platform

ค่าที่ตั้ง

- *Callback (function)* – Callback Function

```
def callback_disconnect() :  
    print "Disconnected"  
  
client.on_disconnect = callback_disconnect
```

client.on_message เกิดขึ้นเมื่อได้รับข้อความจากการ Chat หรือ หัวข้อที่ Subscribe

ค่าที่ตั้ง

- *Callback (function)* - พื้นที่จะทำงานเมื่อได้รับข้อความ โดยพื้นที่จะรับพารามิเตอร์ 2 ตัวคือ
 - *topic* - ชื่อ Topic ที่ได้รับข้อความนี้
 - *message* - ข้อความที่ได้รับ

```
def callback_message(topic, message) :  
    print "I got message from ", topic, ":", message  
  
client.on_message= callback_message
```

client.on_present Event นี้จะเกิดขึ้นเมื่อมี Microgear ใน Appid เดียวกัน Online เข้ามาเชื่อมต่อ NETPIE

ค่าที่ตั้ง

- *callback (function)* - จะทำงานเมื่อเกิดเหตุการณ์นี้ โดยจะรับค่าพารามิเตอร์ คือ
 - *gearkey* - ระบุค่าของ gearkey ที่เกี่ยวข้องกับเหตุการณ์นี้

```
def callback_present(gearkey) :  
    print gearkey+" become online."  
  
client.on_present = callback_present
```

client.on_present Event นี้จะเกิดขึ้นเมื่อมี Microgear ใน Appid เดียวกัน Offline หายไป

ค่าที่ตั้ง

- *callback* (function) - จะทำงานเมื่อเกิดเหตุการณ์นี้ โดยจะรับค่าพารามิเตอร์ คือ
 - gearkey - ระบุค่าของ gearkey ที่เกี่ยวข้องกับเหตุการณ์นี้

```
def callback_absent(gearkey) :  
    print gearkey+" become offline."  
  
client.on_absent = callback_absent
```

client.on_warning เป็น Event ที่เกิดเมื่อมีเหตุการณ์บางอย่างเกิดขึ้น และมีการเตือนให้ทราบ

ค่าที่ตั้ง

- *callback* (function) - จะทำงานเมื่อเกิดเหตุการณ์นี้ โดยจะรับค่าพารามิเตอร์ คือ
 - msg - ระบุข้อความที่เกี่ยวข้องกับเหตุการณ์นี้

```
def callback_warning(msg) :  
    print msg  
  
client.on_warning = callback_warning
```

client.on_info เป็น Event ที่เกิดมีเหตุการณ์บางอย่างเกิดขึ้นขึ้นภายใน Microgear

ค่าที่ตั้ง

- *callback* (function) - จะทำงานเมื่อเกิดเหตุการณ์นี้ โดยจะรับค่าพารามิเตอร์ คือ
 - msg - ระบุข้อความที่เกี่ยวข้องกับเหตุการณ์นี้

```
def callback_info(msg) :  
    print msg  
  
client.on_info = callback_info
```

client.on_error Event นี้จะเกิดขึ้นเมื่อมี Error

ค่าที่ตั้ง

- *callback* (function) - จะทำงานเมื่อเกิดเหตุการณ์นี้ โดยจะรับค่าพารามิเตอร์ คือ
 - *msg* - ระบุ Error ที่เกี่ยวข้องกับเหตุการณ์นี้

```
def callback_error(msg) :
    print msg
client.on_error = callback_error
```

6. REST API

NETPIE Platform ได้จัดเตรียม REST API สำหรับการติดต่อสื่อสารกับ Microgear ชนิดอื่นๆ ผ่านทาง HTTP Protocol ที่เข้าถึงได้ง่าย โดยไม่ยึดติดกับ programming Language หรือ Hardware สามารถนำไปประยุกต์ใช้กับ Web Server แบบเดิม หรือเรียกผ่าน Command Line รวมไปถึงการเชื่อมต่อกับ Web Server ต่างๆ

6.1 API ENDPOINT

REST API ของ NETPIE ให้บริการอยู่ที่ <https://api.netpie.io/>

6.2 AUTHENTICATION

ในการเชื่อมต่อ API Client จะต้องทำการยืนยันตัวตน โดยใช้หนึ่งในสองวิธีนี้

1. ส่งผ่าน HTTP Header แบบ Basic Auth โดยใช้

```
Username : KEY
Password : SECRET
```

ตัวอย่างการใช้ Basic Auth ด้วย cURL

```
$ curl -X GET "http://www.domainname.com/resources" -u key:secret
```

2. ส่งผ่านทาง URL Parameter ในรูปแบบ

```
?auth=KEY:SECRET
```

ตัวอย่างการใช้ URL Parameter ด้วย cURL

```
$ curl -X GET "http://www.domainname.com/resources?auth=key:secret"
```

6.3 RESOURCE TYPES

Topic

Topic เป็นจุดแลกเปลี่ยน Message ระหว่าง Microgear ลักษณะการเขียนจะอยู่ในรูปของ Path เช่น /home/bedroom/temp โดย Microgear สามารถ PUT/publish, GET/subscribe ไปที่ Topic นี้ได้

Microgear

Microgear คือ คุปกรณ์ที่เชื่อมต่อกับ NETPIE เรายสามารถสื่อสารตรงไปที่ Microgear โดยอ้างอิงชื่อ alias ที่ตั้งให้อุปกรณ์นั้น

Postbox

Postbox เป็นพื้นที่สำหรับเก็บข้อมูลแบบ Queue โดย Message ที่ถูกส่งเข้าไปใน Postbox จะถูกเก็บสะสมไว้ จนกว่าจะมีการอ่านออกไป Message ที่ถูกอ่านแล้วจะหายไปจาก Postbox ทันทีจึงหมายความว่าจะใช้เป็นเครื่องมือสื่อสารกับ Microgear ที่ไม่สามารถ Online ได้ตลอดเวลา เช่น PHP Script

6.3.1 Topic

PUT /topic/{appid}/{topic}

เขียนข้อความไปยัง Topic ของ Appid ตามที่ระบุ

URL parameter

- *retain* สั่งให้เก็บค่าไว้ (เฉพาะค่าล่าสุดเพียงค่าเดียว)

Body

เป็น Message ที่จะส่ง หากต้องการลบค่าที่ Retain ไว้ ให้ส่งแบบ Retain และให้ Body เป็น String ปล่า

ตัวอย่างการเรียก REST API ด้วย cURL สมมติว่าเราได้สร้าง App บน NETPIE ชื่อ myappid และมี Key และ Secret ดังนี้

AppID : myappid

[delete](#) [hide](#)

App Key : jVjzJXaJwdJKHhF

App Secret : StOAKIZhXB5CaqnIHeb7s1DfiW7mQj Show characters

เราจะใช้ REST API ในการส่ง Message เป็น Retain ว่า ON ไปยัง microgear ที่บอกรับ Topic /home/bedroom/light ได้ด้วย cURL command line นี้

```
$ curl -X PUT  
"https://api.netpie.io/topic/myappid/home/bedroom/light?retain" -d "ON" -u  
jVjzJXaJwdJKHhF:StOAKIZhXB5CaqnIHeb7s1DfiW7mQj
```

GET /topic/{appid}/{topic}

อ่าน Message จาก Appid ที่ Topic ตามที่ระบุโดย Client จะได้รับเฉพาะ Message ล่าสุดที่ถูก Retain ไว้ก่อนหน้านี้

URL parameter

- ไม่มี

Body

- ไม่มี

ตัวอย่าง

```
$ curl -X GET "https://api.netpie.io/topic/myappid/home/bedroom/light" -u  
jVjzJXaJwdJKHhF:StOAKIZhXB5CaqnIHeb7s1DfiW7mQj
```

6.3.2 Microgear

PUT /microgear/{appid}/{gearalias}

ส่ง Message ไปยัง Microgear ที่ตั้งชื่อว่า gearalias ของ Appid ชื่อ appid

Body

- Message ที่จะส่ง เป็น Plain Text String หากมีการเข้ารหัสด้วยอูปแบบ JSON ทางปลายทางจะต้องนำ String ไป Parse เอง

6.3.3 Postbox

PUT /postbox/{appid}/{postboxname}

ส่ง Message ไปยัง Postbox ชื่อ *postboxname* ของ Appid ชื่อ *appid*

URL parameter

- *tag* ผู้ส่งสามารถติด Tag ให้ Message ได้ เพื่อความสะดวกในการเลือกอ่านเฉพาะ Message ที่สนใจ

Body

- *message* ที่จะส่ง เป็น Plain Text String หากมีการเข้ารหัสด้วยรูปแบบ JSON ทางปลายทางจะต้องนำ String ไป Parse เอง

```
$ curl -X PUT "https://api.netpie.io/postbox/myappid/webbox?tag=error" -d "ON" -u jVjzJXaJwdJKHhF:StOAKIZhXB5CaqnIHeb7s1DfiW7mQj
```

GET /postbox/{appid}/{postboxname}

อ่าน Message ครั้งล่าสุดที่ส่งข้อความจาก Postbox ชื่อ *postboxname* ของ Appid ชื่อ *appid* โดยจะเรียงตามลำดับเวลา Message ที่เข้ามาก่อน จะถูกอ่านก่อน

URL parameter

- *tag* ไม่จำเป็นต้องระบุ แต่หากมีการระบุ Tag จะเป็นการเฉพาะจาะจงค่าเฉพาะ Message ที่ติด Tag นี้เท่านั้น

Body

- ไม่มี

```
$ curl -X GET "https://api.netpie.io/postbox/myappid/webbox?tag=error" -u jVjzJXaJwdJKHhF:StOAKIZhXB5CaqnIHeb7s1DfiW7mQj
```

ផ្តែមឱន

1. ពនិតា ងមីព្យូលី
2. ទារីវី ឯសិរិយភាព
3. កុលម៉ាទិ មីទុរីយ៍លាក
4. កែមខ័ណ្ឌ និវ៉នតសុខវត្ថុ
5. បេវរដី កើមសុវត្ថិភាព
6. អនុនៅ ប៊ូណ្ឌា
7. ឃីវិទ្យា សេនវីសុុ

គួរពនៃការងារនៃក្រុមហ៊ុនបច្ចុប្បន្ន

សំណងការងារនៃក្រុមហ៊ុនបច្ចុប្បន្ន

112 ភូមិសាស្ត្របឹងកេងកង សង្កាត់បឹងកេងកង រាជធានីភ្នំពេញ

តំបន់ក្រុមហ៊ុនបច្ចុប្បន្ន ក្រុងក្រុមហ៊ុនបច្ចុប្បន្ន រាជធានីភ្នំពេញ 12120

ទូរ 02-564-6900

អ៊ីមែល support@netpie.io

Website <https://netpie.io>

Facebook Group “NETPIE” <https://www.facebook.com/groups/netpie/>

Github “netpieio” <https://github.com/netpieio>