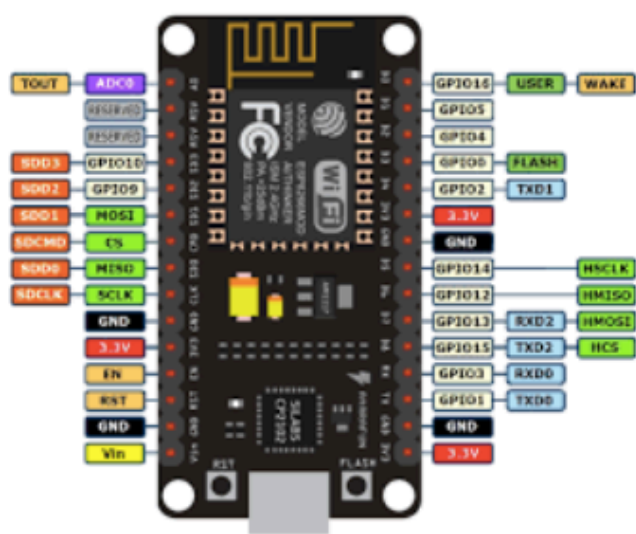


สร้าง IoT protocol + database + dashboard ไว้ใช้งานเอง ด้วย Google Cloud platform/Node.js/MongoDB/Chart.js | code.isaranu.com

Like Share

Isaranu | 11.11.2017, (14.5hrs)  
8277 viewed



— code.isaranu.com —

แค่ซื้อบทความก็ยาวววววแล้ว... แต่รับรองว่า จะเขียนให้สั้น, ให้เข้าใจง่าย และให้สามารถนำไปใช้งานได้ทันทีครับ.

## [ที่มาของบทความนี้]

เริ่มจากว่า มี user ใน [loTtweet.com](https://loTtweet.com) คนหนึ่งสอบถามมาว่า

*"ถ้าผมต้องการจะสร้าง IoT platform อย่างง่ายๆ แบบเก็บ data ใน database แล้วก็เอา data ออกมาแสดงในหน้า dashboard ครับ. ผมจะทำได้บ้าง ?"*

ก็เลยเป็นที่มาของบทความนี้ และผมคิดว่าเนื้อหาที่น่าจะมีประโยชน์กับหลายท่านที่กำลังมองหา IoT solution ประมานนี้อยู่ครับ.

ในบทความนี้ ผมจะใช้ทรัพยากรดังต่อไปนี้ครับ แต่..อ๊ะๆ, บอกก่อนว่า จริงๆมันก็มีวิธีที่ง่ายกว่านี้ แต่ถ้ามันจะง่ายเกินไป แล้วเมื่อไรเราจะพัฒนาตัวเองให้เก่งได้ล่ะครับ จริงมั๊ย.

# [ทรัพยากรที่ต้องใช้]

**Google account** และ Credit card. (เดี๋ยวผมจะอธิบายต่อไปว่า ทำไมต้องใช้ credit card)

**Code editor** ใช้สำหรับการเขียนโปรแกรมฝั่ง server และ frontend. ผมใช้ atom อยู่ครับ, ท่านใดถนัดอะไรก็ใช้ได้เลย.

**Arduino IDE software** ใช้สำหรับเขียนโปรแกรมฝั่ง device IoT. ให้ดาวน์โหลดที่เว็บไซต์ Arduino ได้เลยครับ.

**IoT device** แนนอน, ไม่อย่างนั้นก็ไม่ว่าจะเอาอะไรส่งค่านะครับ. ผมใช้ NodeMCU (chip : esp8266) หาซื้อได้ทั่วไปครับ.

อันนี้เป็นแค่หลักๆที่ต้องมีนะครับ, ส่วนที่เป็นรายละเอียดปลีกย่อย แนะนำให้ดูในบทความแล้วกันนะครับ.

## [แบ่งภาคกันก่อน]

เนื่องจากการทำมีหลายขั้นตอน และหลายส่วนมากๆ ดังนั้นผมจะแยกเป็นส่วนหลักๆดังนี้ครับ

**ฝั่ง IoT device** เขียน code บน NodeMCU ให้ connect wifi และส่งค่ามาที่ server เพื่อเก็บลง database.

**ฝั่ง server** รับค่า, แล้วเก็บลง database.

**ฝั่ง Browser หรือ Frontend** แสดงค่าที่เก็บไว้ เอาออกมาแสดงใน chart.

## [พื้นฐานที่ต้องมีก่อนเริ่ม]

ต้องมีพื้นฐานเบื้องต้น และมีความรู้ในการเขียน code มาก่อนบ้างนะครับ มีดังนี้

**IoT ภาษา C++** การใช้ Arduino IDE upload code และต้อง install resources ทั้งหมดสำหรับ esp8266 มาแล้วนะ.

**CLI บน Linux** บทความนี้ผมจะสร้าง VM(Virtual Machine) บน Google Cloud Platform โดยใช้ Linux OS เป็น Debian 8 นะครับ. ดังนั้น, เมื่อเราทำงานต่างๆบน VM, เราจะทำผ่าน Secure Shell หรือ SSH terminal ด้วย CLI ครับ.

**Javascript** อันนี้ใช้เยอะสุดครับ เพราะว่ามันเป็นภาษาทั้งบน server และ Frontend.

- Server (Node.js, Express และ mongojs) เข้าใจการเขียนแบบ non-blocking IO programming, ใช้

promise เป็น และเข้าใจการทำงานแบบ async/await ด้วยก็จะดีมาก.

- Frontend ใช้ parse ข้อมูลและแปลงข้อมูลไป render บน chart ครับ. ส่วน chart element ที่เราจะใช้ในบทความนี้คือ Chart.js นะครับ

**HTML/CSS** พื้นฐานการสร้างเว็บไซต์ และตกแต่งบ้างเล็กน้อยครับ

เอาล่ะ, มาเริ่มกันเลยครับ

## [1. Google account, เริ่มใช้งาน Google Cloud platform]



มาเริ่มตรงที่เรามี Google account กันแล้วนะครับ. ให้เราไปที่เว็บไซต์ [Google Cloud Platform](#) ซึ่งเป็นการให้บริการต่างๆบนระบบ cloud. ให้เราทำการเปิดใช้งานได้เลย ในขั้นตอนนี้ จะมีการให้กรอกรายละเอียดต่างๆครับ ซึ่งเป็นการกรอก ที่อยู่และข้อมูลทั่วไป.

แต่ในตอนท้าย จะให้กรอกเลขบัตร credit ด้วย. ซึ่งเมื่อเราสมัครใช้งานแล้ว เราจะได้ credit สำหรับทดลองใช้งานมา 300 usd หรือ 365 วัน นะครับ.

# Build What's Next Better software. Faster.

- ✓ Use Google's core infrastructure, data analytics and machine learning.
- ✓ Secure and fully featured for all enterprises.
- ✓ Committed to open source and industry leading price-performance.

[GO TO CONSOLE](#)[CONTACT SALES](#)

แต่เมื่อใช้หมดแล้ว (ทั้งเงิน หรือเวลา แล้วแต่อย่างใดอย่างหนึ่งหมดก่อน) ถ้าเราไม่สั่งอัปเกรดเป็น paid plan, ทาง Google Cloud Platform (ต่อไปจะเขียนสั้นๆว่า GCP นะ) จะหยุดให้บริการทันทีครับ.

ดังนั้น, ข้อมูลที่เราเก็บไว้ทั้งหมด เราก็ไม่สามารถนำออกมาได้ครับ. ถ้าจะช้าๆๆ แนะนำว่าตอนใกล้ๆ หมด credit 300 usd ให้กดอัปเกรดไปเลยครับ.

\*\* แนะนำให้ศึกษารายละเอียด GCP เพิ่มเติมนะครับ \*\*

## อัปเกรดเป็นบัญชีแบบเสียค่าใช้จ่าย

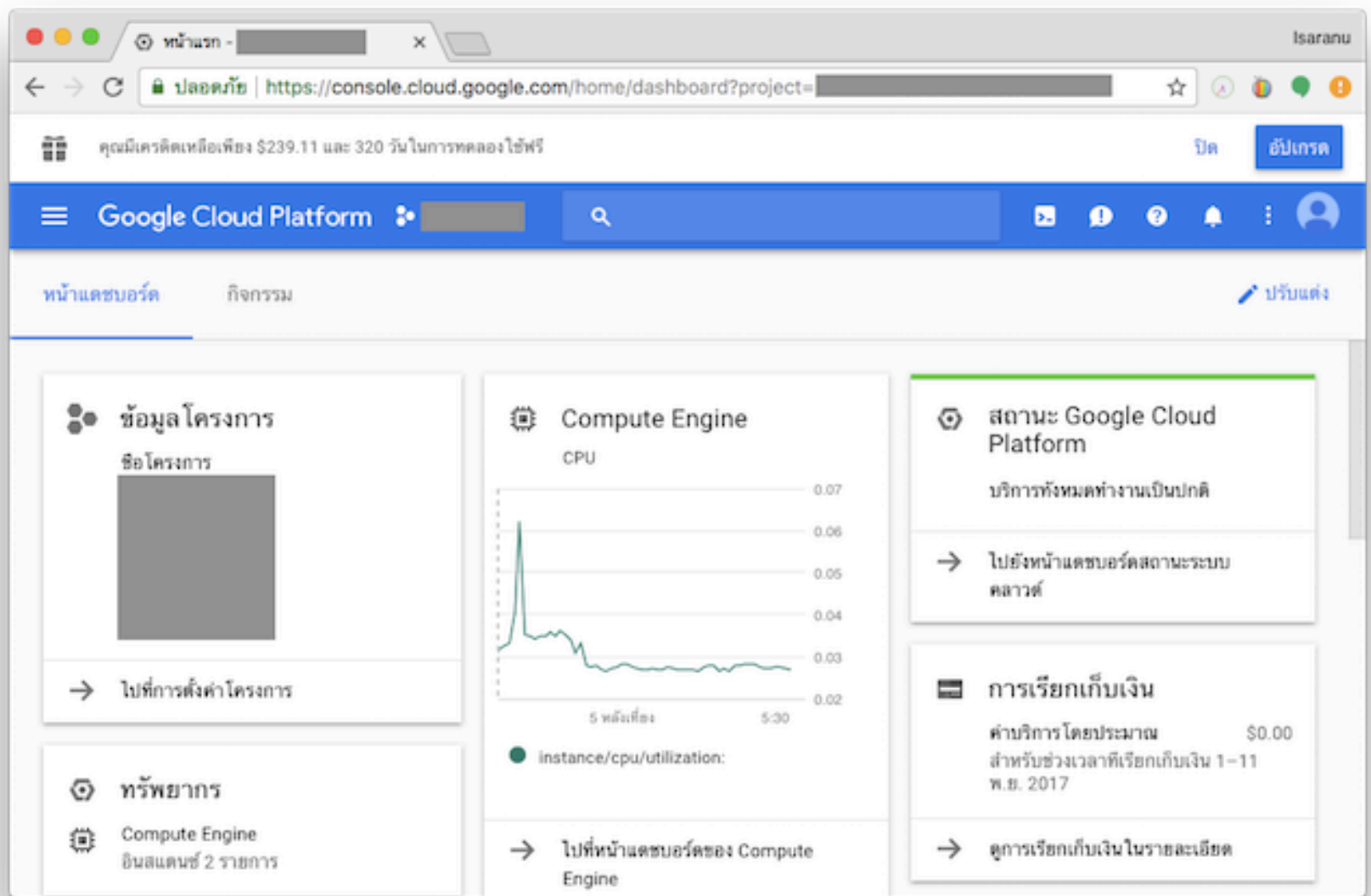
อัปเกรดบัญชีตอนนี้เพื่อให้บริการทำงานต่อหลังจากที่เครดิตฟรีของคุณหมดหรือหมดอายุ นอกจากนี้คุณจะได้สิทธิ์เข้าถึงเครื่องเสมือน Compute Engine ที่มีประสิทธิภาพมากขึ้นด้วย

เมื่อคุณใช้บัญชีที่อัปเกรด ระบบจะเรียกเก็บเงินจากคุณโดยอัตโนมัติหลังใช้เครดิตฟรีจนหมดหรือเครดิตหมดอายุ ขึ้นอยู่กับว่ากรณีใดเกิดก่อน [เรียนรู้เพิ่มเติม](#)

โครงสร้างราคาของเรานั้นเรียบง่ายและโปร่งใส โดยคุณสามารถเข้าดูพร้อมควบคุมได้อย่างสมบูรณ์ และจ่ายเฉพาะที่คุณใช้ไปเท่านั้น [ดูรายละเอียดราคา](#)

[ยกเลิก](#) [อัปเกรด](#)

เมื่อพร้อมใช้งานแล้ว หน้าตา console GCP ของเราจะเป็นประมาณนี้

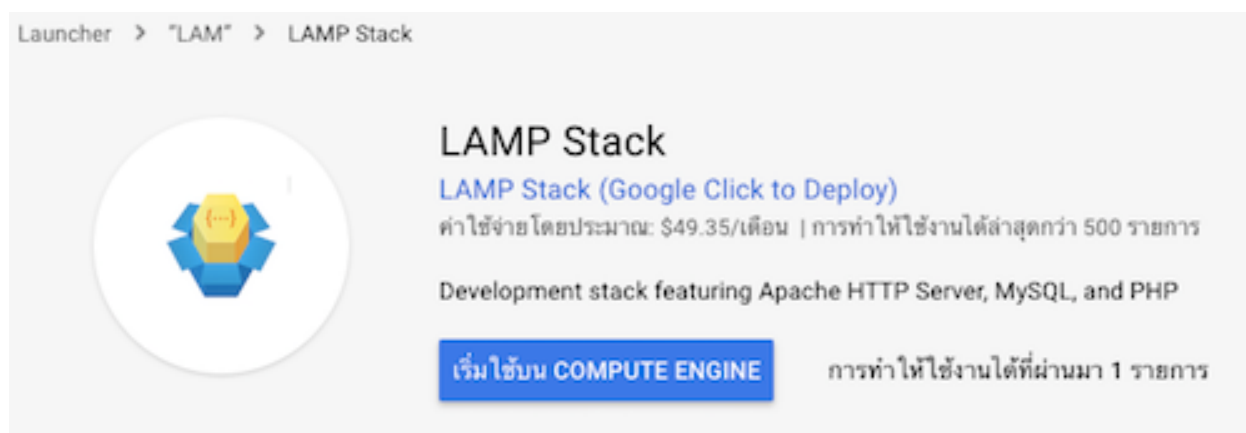
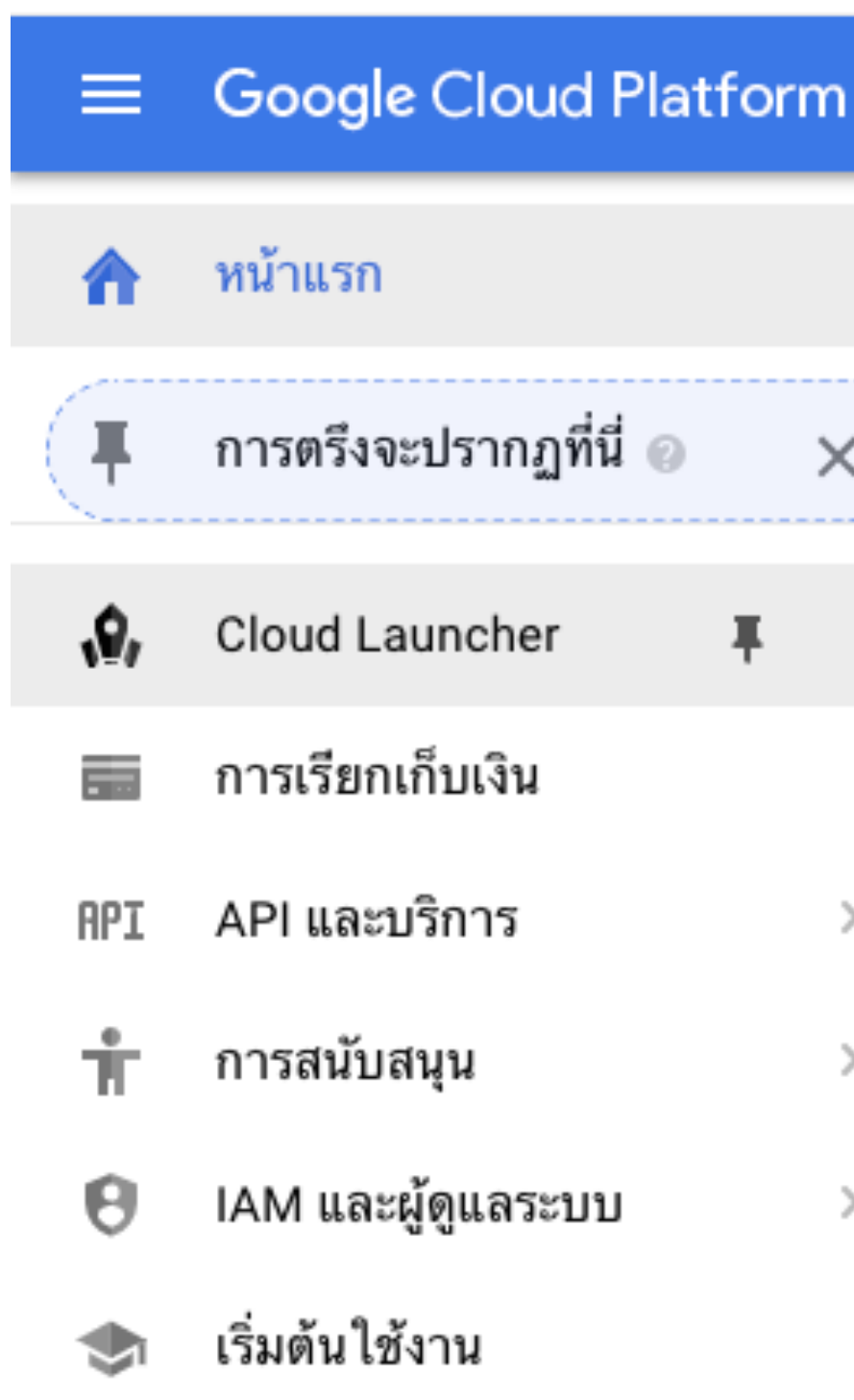


## [2. สร้าง VM และเปิด port]

ต่อไปเป็นขั้นตอนการสร้าง Virtual Machine (VM) ครับ. ให้ไปที่แถบเมนูด้านซ้าย > **Cloud Launcher**

จากนั้นพิมพ์ในช่องค้นหา solution ว่า "LAMP" ครับ. LAMP ย่อมาจาก "Linux Apache MySQL และ PHP" ซึ่งทาง GCP จะทำเป็น One click deploy เอาไว้ให้เลย มันจะขึ้นมาเยอะมาก แต่ให้เลือกตัวนี้ครับ ตัวนี้จะ เป็น Linux OS Debian 8 Jessie นครับ.

ให้กดเริ่มใช้บน compute engine ได้เลย แล้วรอสักครู.



เมื่อติดตั้งเสร็จแล้ว ก็กลับไปหน้า Home. คราวนี้จะเห็นว่าที่หัวข้อ "ทรัพยากร" จะมีอินสแตนซ์มา 1 รายการ. กดเข้าไปเลยครับ หน้าตาแบบนี้.



กรองอินสแตนซ์ VM

<input type="checkbox"/>	ชื่อ ^	โซน	คำแนะนำ	IP ภายใน	IP ภายนอก	เชื่อมต่อ
<input type="checkbox"/>	✔️ lotprotocol	asia-southeast1-a				SSH ▾

ตอนนี้อินสแตนซ์เรา(หรือ VM) สามารถใช้งานได้แล้วครับ. ถ้าจะลองดูหน้า page ของเรา ก็สามารถทำได้ โดยกดที่ปุ่มลูกศรข้างๆ External IP address ครับ. กดไปก็จะเจอหน้า landing page ของ Apache

ต่อไปเป็นการเปิดพอร์ตสำหรับ Node.js ครับ. ตอนนี้ที่เรา run Apache ผ่านพอร์ต 80 อยู่ แต่เราต้องเพิ่ม port ใหม่ในการสั่งรัน Node.js เพื่อรับข้อมูลจาก IoT device.

Google Cloud Platform

เครื่องมือ VPC

กฎไฟร์วอลล์

สร้างกฎไฟร์วอลล์ รีเฟรช ลบ

เครื่องมือ VPC

ที่อยู่ IP ภายนอก

กฎไฟร์วอลล์

เส้นทาง

เครื่องมือการเชื่อมต่อแบบเพีย...

VPC ที่แชร์

กฎไฟร์วอลล์จะควบคุมการจราจรของข้อมูลขาเข้าหรือขาออกไปยังอินสแตนซ์ โดยค่าเริ่มต้น ระบบจะบล็อกการจราจรของข้อมูลขาเข้าที่มาจากภายนอกเครือข่ายของคุณ [เรียนรู้เพิ่มเติม](#)

หมายเหตุ: มีการจัดการไฟร์วอลล์ของ App Engine ที่

ขาเข้า

ขาออก


<input type="checkbox"/>	ชื่อ	เป้าหมาย	ตัวกรองต้นทาง	โปรโตคอล / พอร์ต
<input type="checkbox"/>	default-allow-http	http-server	ช่วง IP: 0.0.0.0/0	tcp:80
<input type="checkbox"/>	default-allow-https	https-server	ช่วง IP: 0.0.0.0/0	tcp:443


โดยปกติแล้ว, GCP จะเปิด port เข้าถึง VM เฉพาะที่จำเป็นเท่านั้นครับ ให้เราไปที่เมนูด้านข้าง > เครื่องมือ VPC > กฎไฟร์วอลล์


ในหน้านี้ จะมีรายละเอียดไฟร์วอลล์ของ App Engine อยู่ครับ ทั้งขาเข้าและขาออก. ให้เราเลือกที่ขาเข้า, แล้วกดปุ่ม "สร้างกฎไฟร์วอลล์" ด้านบนครับ.


กรอกรายละเอียดให้ครบถ้วน, ในท้ายสุด ให้ใส่ port ที่เราต้องการครับ. ในบทความนี้จะใช้ tcp พอร์ต 4000 นะครับ. เสร็จแล้วก็กด "สร้าง" ได้เลยครับ.


กลับมาที่รายละเอียดกฎไฟร์วอลล์ ก็ sẽเห็น port ที่เราเปิด เป็นอันเสร็จขั้นตอน


 เครือข่าย VPC


 เครือข่าย VPC

 ที่อยู่ IP ภายนอก

 กฎไฟร์วอลล์


 เส้นทาง


 เครือข่ายการเชื่อมต่อแบบเพีย...

 VPC ที่แชร์


← สร้างกฎไฟร์วอลล์


☐ ปฏิเสธ


เป้าหมาย 

แท็กเป้าหมายที่ระบุ 


แท็กเป้าหมาย


ตัวกรองค้นหา 


ช่วง IP 

ช่วง IP ค้นหา 

เช่น 0.0.0.0/0, 192.168.2.0/24

ตัวกรองค้นหาลำดับที่สอง 

ไม่มี 

โปรโตคอลและพอร์ต 

☐ อนุญาตทั้งหมด

☒ โปรโตคอลและพอร์ตที่ระบุ

ค้นโดยใช้อักขระ ( ) เช่น tcp; udp:80; udp:5000-6000

สร้าง

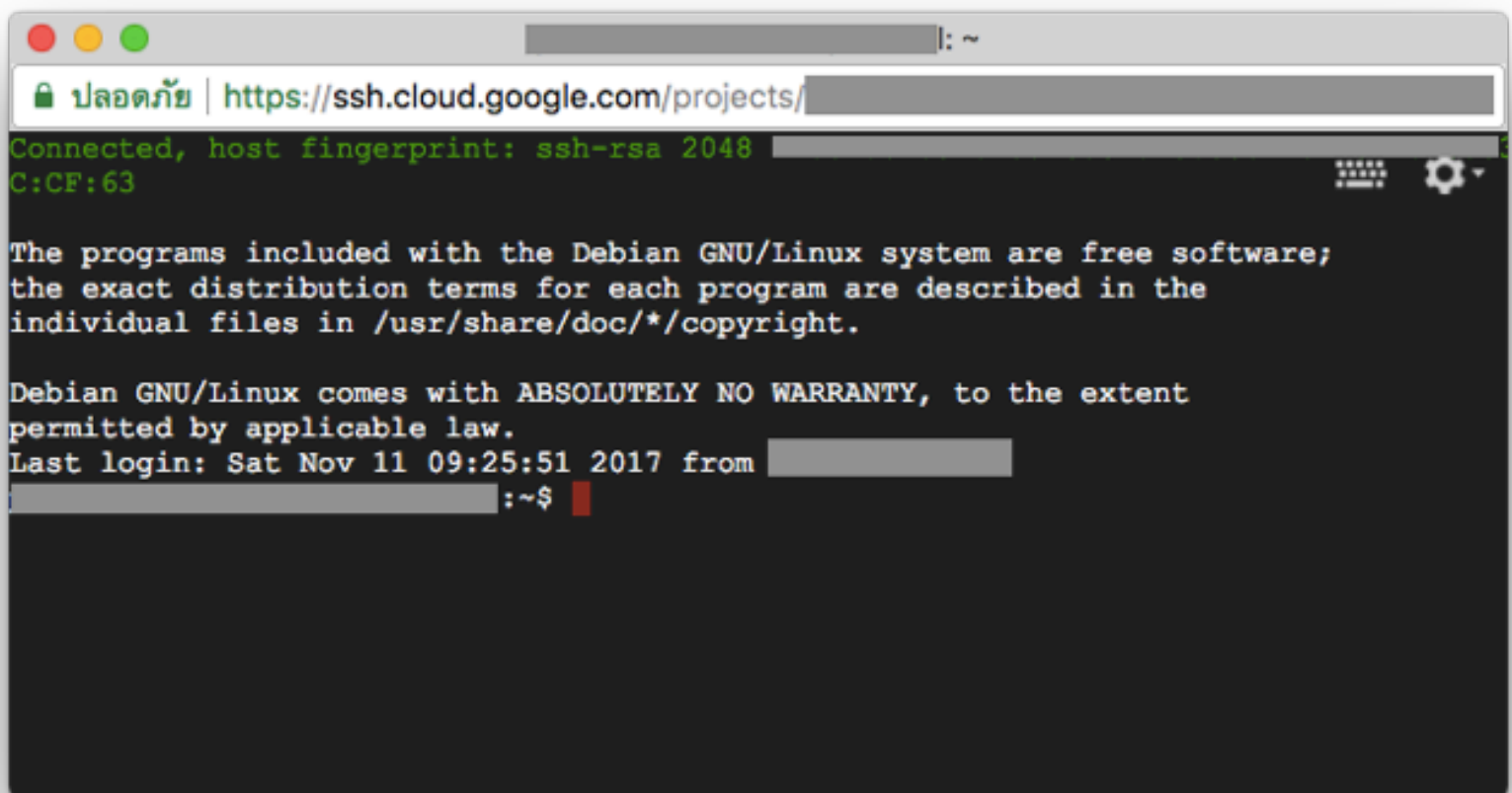
ยกเลิก

REST หรือ บรรทัดคำสั่ง ที่เทียบเท่า

### [3. install MongoDB, Node.js(v8.9.0), npm, express, promise, mongojs และ forever]

ขั้นตอนต่อไป จะทำงานผ่าน SSH นะครับ ซึ่งตอนนี้ต้องใช้ skill CLI กันแล้ว.  
เริ่มจากไปที่หน้าอินสแตนซ์, แล้วกดปุ่ม SSH. หลังจากนั้นจะปรากฏจอขึ้นมาใหม่ รอการเชื่อมต่อสักครู่.





```
ปลดภัย | https://ssh.cloud.google.com/projects/
Connected, host fingerprint: ssh-rsa 2048
C:CF:63

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Nov 11 09:25:51 2017 from
:~$
```

ตรงนี้แนะนำให้ใช้ Browser Chrome นะครับ ถ้าเป็น Browser อื่นๆ จะทำงานแปลกๆ หรือไม่ก็ไม่ขึ้นหน้าจอมาเลย.



มาติดตั้งฐานข้อมูล MongoDB กันก่อน. รายละเอียดการติดตั้งบน Debian, ให้ทำตามนี้เลยครับ  
[How to install MongoDB](#)

\*\* ทำตาม Debian 8 "Jessie" เลยนะครับ / จะไม่ขออธิบายในนี้นะ \*\*



ถัดไป, เราจะติดตั้ง Node.js กันครับ. ตอนที่เขียนบทความนี้ เราจะใช้ Node.js v8.9.x นะครับ (เพราะดูเหมือน v9.0 จะมีปัญหากับ Express ตัวปัจจุบัน, และเวอร์ชัน 8.9.x นี้รองรับการเขียนแบบ async/await ด้วย). ว่าแล้วก็ไปที่ SSH terminal กันเลย พิมพ์คำสั่ง install.

```
$ sudo apt-get update
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
$ sudo apt-get install -y nodejs
```

ตรวจสอบเวอร์ชันของ node ด้วย

```
$ node -v
```



ต่อไปก็ติดตั้ง npm ครับ (npm คือ Node package manager) เป็นตัวช่วยติดตั้ง modules ต่างๆ ให้เราครับ. ใครมองเหมือนผมบ้าง โลโก้มันเป็นหมีหัวเถิก >\_<

```
$ sudo apt-get install npm
```

ตรวจสอบเวอร์ชันของ npm ด้วย

```
$ npm -v
```

ต่อไปก็สร้าง Project folder ของเราขึ้นมาก่อนครับ ชื่อ myiotprotocol ด้วยคำสั่ง

```
$ sudo mkdir myiotprotocol
```

# express

เอาล่ะ, ต่อไปก็ติดตั้ง modules ที่จำเป็นสำหรับการทำงานกันครับ.

เริ่มจาก Express ก่อนเลย, Express เป็น web framework modules สำหรับการทำ routes การ request ต่างๆ เช่น GET, POST, PUT และ DELETE. มาเริ่มติดตั้งผ่าน npm กันเลย

([website express](#))

```
$ sudo npm install express --save
```



ถัดไปเป็น "promise". ตัวนี้สำคัญมากสำหรับการเขียน code บน node.js เป็น tool สำหรับการเขียนแบบ promise ใน ES6. ถ้าไม่ใช่ล่ะก็ เจอ callback hell แน่ๆครับ เวลาที่เราต้องเขียน code ที่ซับซ้อนกว่านี้. ([website promise](#))

```
$ sudo npm install promise --save
```

mongojs public



A **node.js** module for mongodb, that emulates **the official mongodb API** as much as possible. It wraps **mongodb-native** and is available through **npm**

```
npm install mongojs
```

build passing code style standard

ถัดไป, ติดตั้ง modules "mongojs" สำหรับเชื่อมต่อกับ database MongoDB. ตัวนี้ใช้งานง่าย และรูปแบบคำสั่งจะคล้ายบน CLI บน Mongo shell.

([website mongojs](#))

```
$ sudo npm install mongojs --save
```

forever public



gitter join chat

npm v0.15.3 downloads 815k/month build passing dependencies insecure docs



```
npm install forever -g
```

15 dependencies version 0.15.3  
263 dependents updated a year ago

modules สุดท้าย, "forever".

forever เป็น tool สำหรับสั่งให้ node.js ของเรา รัน code อยู่ตลอดเวลา.  
เมื่อมี error เกิดขึ้น, forever จะสั่ง restart ให้ทันทีครับ (สมชื่อจริงๆ :D).

([website forever](#))

```
$ sudo npm install -g forever
```

ก่อนจะไป section ถัดไป,  
มาลองใช้งาน express กันก่อนครับ สร้างไฟล์ app.js ขึ้นมา แล้ววาง code นี้ลงไปครับ.

```
const express = require('express')
const app = express()

app.get('/', (req, res) => res.send('Hello World!'))

5. app.listen(4000, () => console.log('Example app listening on port 4000!'))
```

แล้วก็รันด้วยคำสั่ง

```
$ node app.js
```

## [4. เริ่มเขียน IoT protocol]

แค่ติดตั้งสิ่งต่างๆก็เหนื่อยแล้ว ใช่มั้ยครับ ? แต่อย่าเพิ่งยอมแพ้นะ

มาต่อกันที่การเขียน code สำหรับรับ GET, POST request ที่เรียกมาจาก IoT device กันครับ.

โดยที่ code ฉบับเต็มๆ จะเก็บอยู่ใน [github:simple-iot-protocol](#) นะครับ. ตามไปดาว์นโหลดกันได้เลย

แต่อย่าลืมกด star ให้ด้วยนะครับ :D

ผมจะอธิบายแค่บางบรรทัดนะครับ,

เริ่มจากด้านล่างนี้ เป็นการเรียก modules ต่างๆเข้ามาใช้งานใน code ของเราครับ

```
const express = require('express');
const app = express();
var port = 4000;

5. var mongojs = require('mongojs');
var Promise = require('promise');
var myiotdb = mongojs('myiotdb');
var devid, data, datasize, dataset='';
```

ถัดลงมาเป็นการกำหนด route ต่างๆ ใน express ว่าถ้ามีการเรียกแบบไหน ให้ไปทำอะไรต่อครับ

จาก code, มีเรียกอยู่ 2 mode คือ เขียน data ที่ได้จาก IoT ลง database และอ่านข้อมูลจาก database มา render บน chart ครับ.

```
app.get('/', function (req, res) {
  res.send("my iot Protocol ready !");
});

5. app.get('/write/:data', function (req, res) {
  var strParseWriteReq = JSON.stringify(req.params);
  var strWriteReq = JSON.parse(strParseWriteReq);
  data = strWriteReq.data;
  writedata(data, res);
10. });

app.get('/read/:datasize', function (req, res) {
  var strParseReadReq = JSON.stringify(req.params);
  var strReadReq = JSON.parse(strParseReadReq);
15. datasize = strReadReq.datasize;
  readdata(datasize, res);
  });
```

ที่นี้มาดูการใช้งาน promise และ async/await กันครับ

จริงๆแล้ว ถ้าการทำงานของ code ไม่ได้ซับซ้อนมาก (หมายถึงไม่ได้ไปทำอะไรอีกหลายๆอย่าง)

เราใช้แค่ promise อย่างเดียวก็นพอ

แต่ไหนๆจะลองเขียนแล้ว ก็ลองใช้ async/await ไปเลย เพราะว่าท่านใดจะไปทำที่ซับซ้อนกว่านี้ จะได้มีความรู้ไปเลย



**async/await** คืออะไร,

ผมขอเล่าแบบนี้แล้วกันนะครับ แต่ก่อนเวลาเราจะเขียนโปรแกรมให้มันไปทำสิ่งอื่นๆต่อ เราจะใช้วิธี

callback function.

แต่ถ้าเรามีหลาย callback function ที่ต่อกันยาวมากๆ , เราจะเจอปัญหา callback ล้น หรือที่เรียกว่า callback hell.

จึงเป็นที่กำเนิดของการเขียนแบบ 'promise' จะเป็นการที่สัญญาว่า หลังจากอันนี้ จะไปทำอันนั้นต่อ ให้เสร็จนะ

โดยที่จะเป็น function แล้วก็ .then().then().then... ไปเรื่อยๆ.

แต่แบบนี้ก็ทำให้ code เราอ่านยากขึ้นอีก ก็เลยเป็นที่มาของ async/await

โดยที่การใช้ async/await นั้นไม่ยากครับ ให้จำไว้ 2 อย่างว่า

1. จะใช้ await ใน function ได้, function นั้นจะต้องระบุ 'async' นำหน้า
2. function ที่อยู่หลัง await, จะต้องเป็น promise เท่านั้น

เท่านี้ก็ทำให้ code ของเราดูเป็นระเบียบขึ้นมาทันที แถมยังรู้สึกได้ว่า เราได้เขียนโปรแกรมแบบปกติ (Synchronous)

ทั้งที่จริงๆเวลารันบน Node.js มันเป็น Asynchronous.

จาก code ด้านล่าง,

เป็นการใช้ promise และ async/await ในการเขียนค่าจาก IoT ลงใน database MongoDB ผ่าน module mongojs ครับ.



```

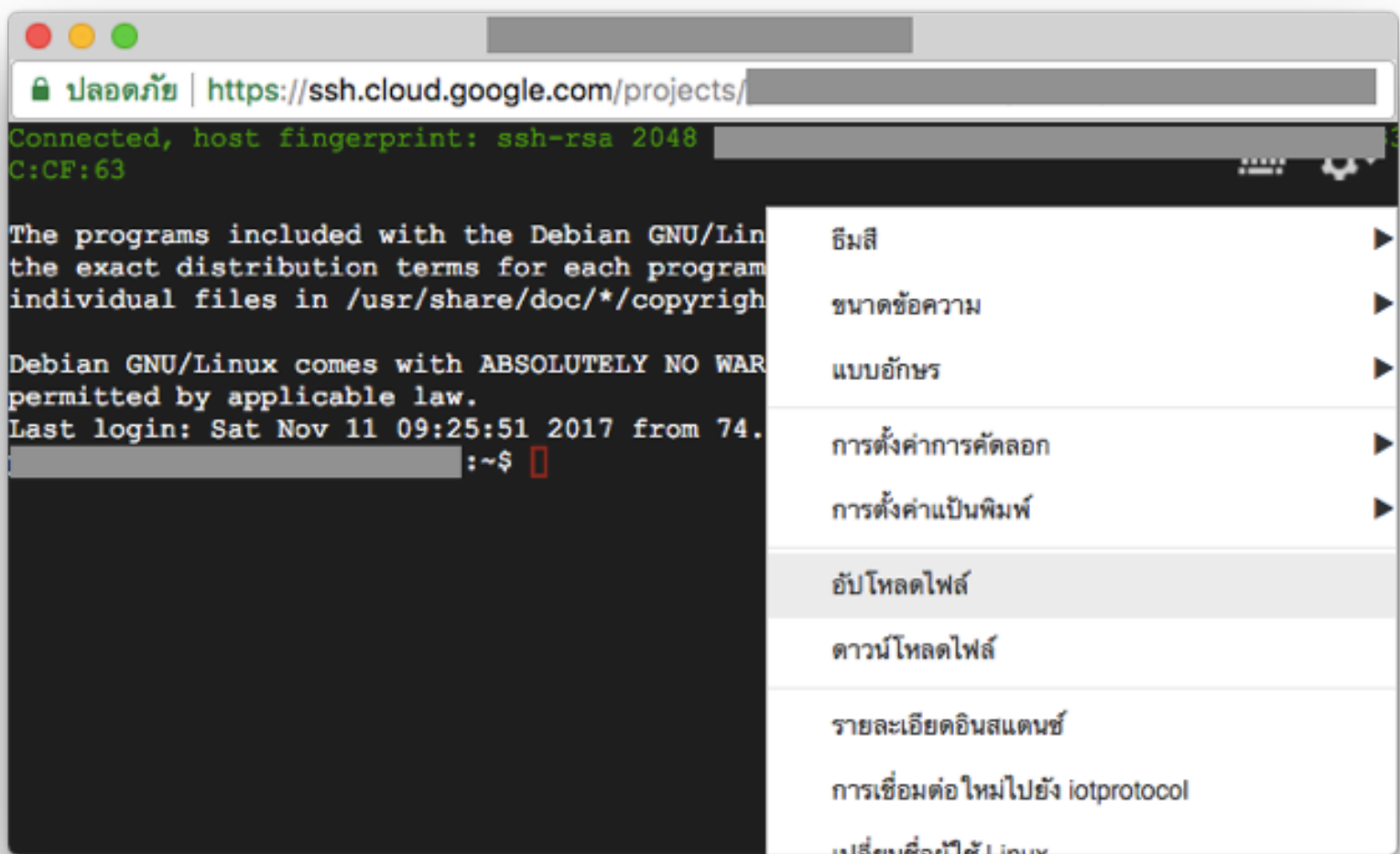
/* -- ASYNC / AWAIT function -- */
async function writedata(_data, res){
  await writeToMongo(_data, res);
}
5.
function writeToMongo(_savedata, res){
  return new Promise(function(resolve, reject){
    var mywritecollection = myiotdb.collection('mycollection');
    mywritecollection.insert({
10.      data: Number(_savedata),
        recordTime: new Date()
    }, function(err){
        if(err){
            console.log(err);
15.      res.send(String(err));
        }else {
            console.log('record data ok');
            res.send('record data ok');
        }
20.    });
  });
}

```

ต่อไปคือขั้นตอนในการนำ app.js ไปเก็บไว้ที่ VM ของเรา

ไปที่ SSH กันเลยครับ และก็นำ app.js ไปไว้ที่ folder ของ project เราได้เลย.

ไปที่เมนู > upload file., เลือกไฟล์ app.js.



จากนั้นก็ move file ไปที่ folder myiotprotocol

```
$ sudo mv app.js myiotprotocol
$ cd myiotprotocol
$ node app.js
```

จะเห็น console log พิมพ์ออกมาแบบนี้ แสดงว่า app.js เราทำงานแล้วครับ

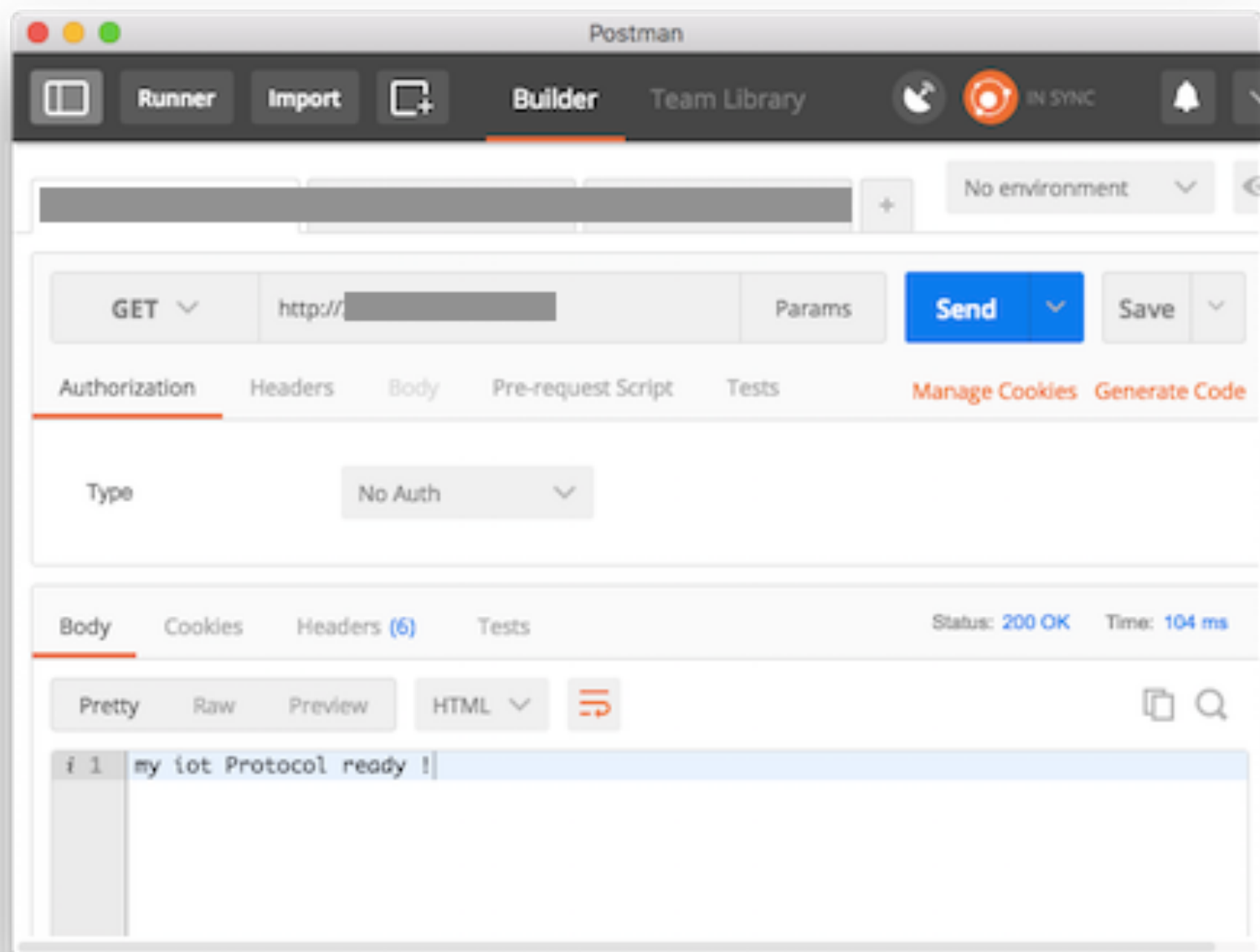
```
"My IoT protocol running on port 4000 start at Sat Nov 11 2017 01:58:57
GMT+0000 (UTC)"
```

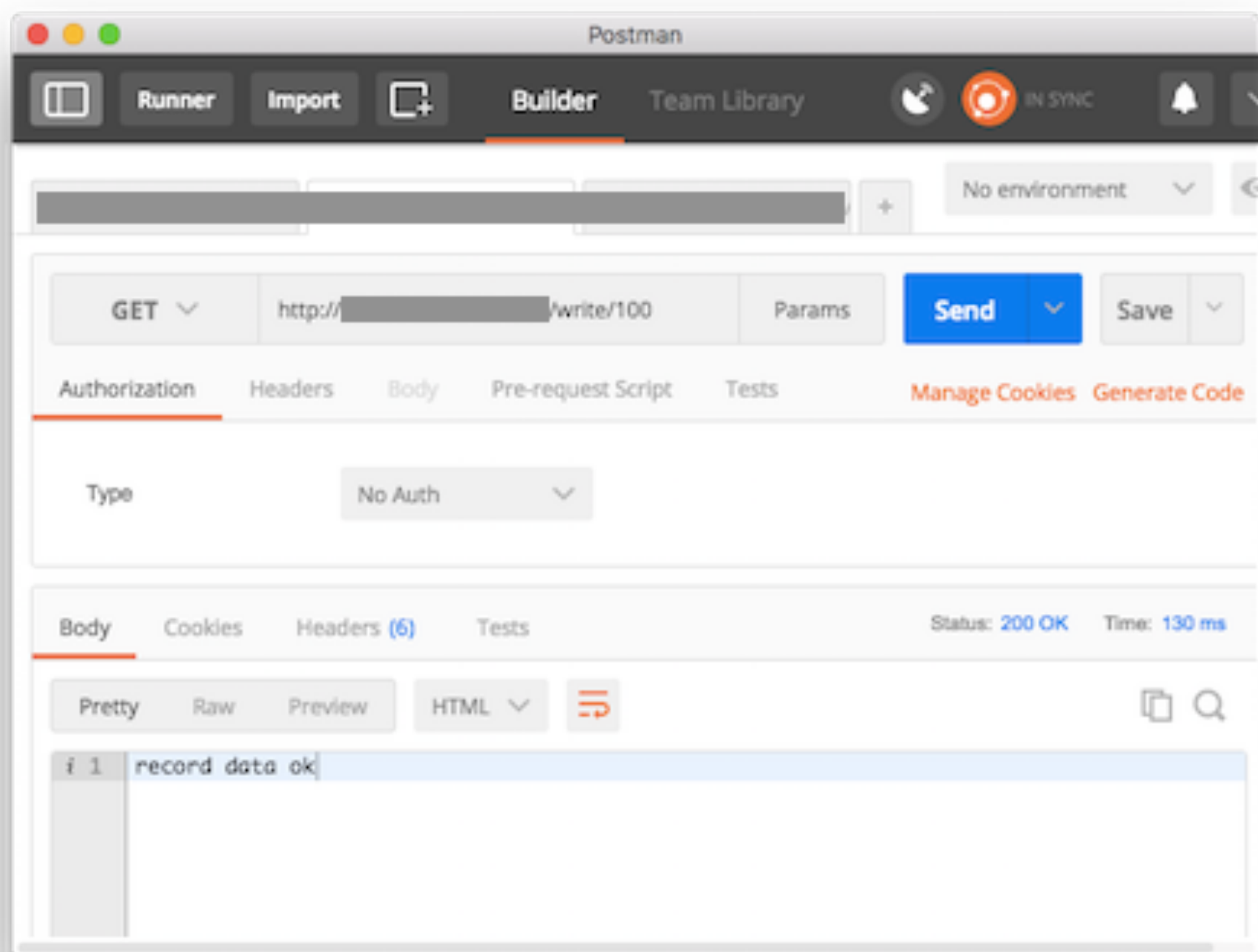
## [5. ทดสอบ IoT protocol ด้วย Postman]

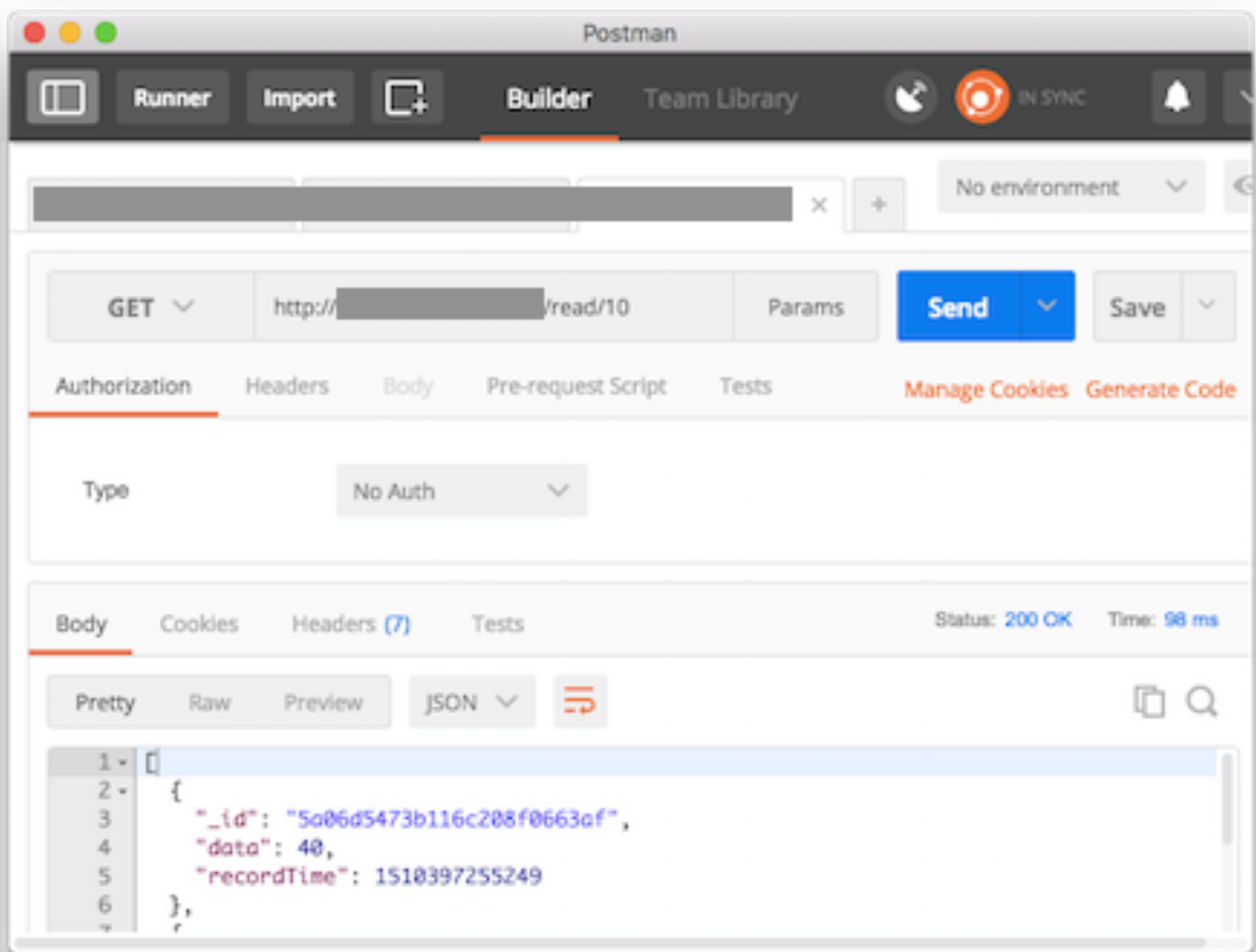


ก่อนจะไปเขียน library บน IoT device จริงๆ เราจะมาทดสอบ app.js ของเราก่อน โดยการส่ง request จากโปรแกรมที่ชื่อว่า "POSTMAN" ครับ (หาดาวน์โหลดได้เลยครับ มีเกลื่อน).

ใส่ path ให้ถูกต้องตามที่เราเขียนไว้ใน app.js นะครับ เริ่มจาก Check ready, Write data แล้วจบด้วย Read data ครับ.





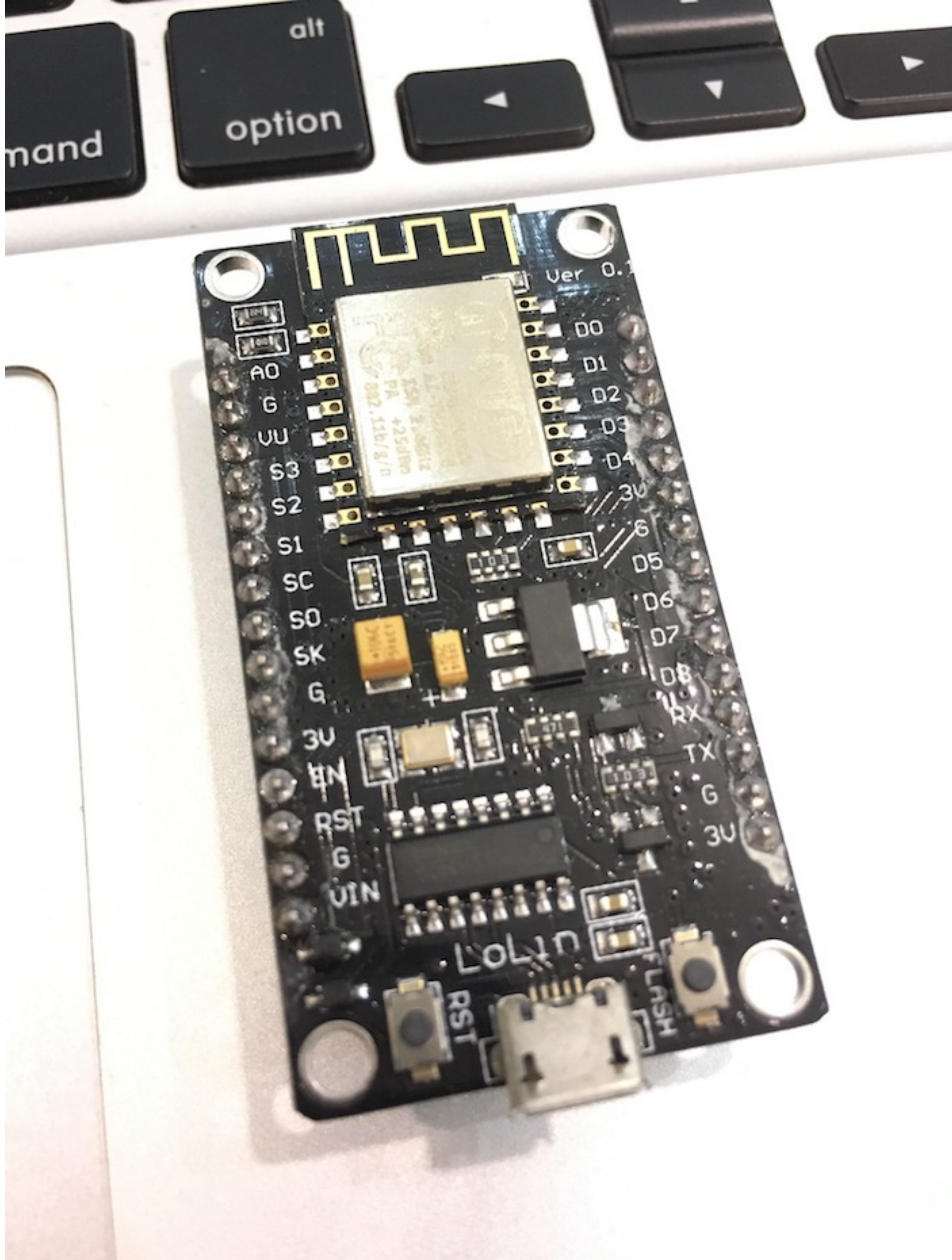


## [6. เขียน code บน IoT device + สร้าง library]



มาถึงส่วนที่เป็น Internet of Things (IoT) device กันแล้วครับ. สิ่งที่เราจะใช้ในนี้คือ Arduino IDE version 1.8.4, พื้นฐาน C++ และ IoT board ครับ. บทความนี้ผมใช้ NodeMCU Esp8266.





อย่างที่ได้อธิบายไว้ข้างบนบทความนะครับว่า ผู้อ่านจะต้องลง Board manager กับ Esp8266WiFi.h library มาให้พร้อม เพราะผมจะไม่กล่าวถึงขั้นตอนการติดตั้งนะครับ มันจะยาวมาก หากหาใน google ได้เลยครับ



มีหลายท่านเคยบทความไว้เยอะ

วันนี้จะมาสอนการทำ library ของเราเอง เพื่อการเรียกใช้ function ในการส่งข้อมูลครับ.  
ผมเตรียมไฟล์ไว้ใน github แล้วครับ สามารถดาวน์โหลดมาใช้งานได้เลย  
แต่อยากให้อ่านคำอธิบายข้างล่างนี้ด้วยครับ เพื่อความเข้าใจรายละเอียด code.

ใน github จะมีไฟล์ library อยู่ 2 ไฟล์คือ myprotocol.h และ myprotocol.cpp  
ให้นำโฟลเดอร์ myprotocol บน github ไปเก็บไว้ที่โฟลเดอร์ **arduino** > **libraries** บนเครื่องที่เรา  
จะใช้ compile IoT board ครับ

ต่อไปจะอธิบาย code นะครับ, เริ่มจากไฟล์ myprotocol.h เป็นการสร้าง class เพื่อไว้ในการเรียก  
function ใช้งาน ทั้ง function และกำหนด private variable.

```
#ifndef myprotocol_h
#define myprotocol_h

#include "Arduino.h"
5.  #include "ESP8266WiFi.h"
    #include "ArduinoJson.h"

class myprotocol{
10.  public:
        bool begin(const char *ssid, const char *passw);
        String sayhi();
        String WriteDashboard(float val);
        String getVersion();
15.  private:
        const char *_ssid, *_passw, *_libversion;
        float _val;
        String _res;
20.  bool _conn;
};

#endif
```

ส่วนไฟล์ myprotocol.cpp, จะเป็นการเขียน function ที่จะใช้งานครับ. ข้างใน code เป็นการทำให้ IoT  
ของเราเชื่อมต่อกับ WiFi ด้วยคำสั่ง begin(ssid, passw);.

เมื่อเชื่อมต่อได้แล้ว เราสามารถตรวจสอบว่า server ของเราพร้อมหรือไม่ โดยคำสั่ง sayhi(); ด้วยการ เรียก  
แบบ GET request ไปที่ app.js บน server ครับ. เราจะได้ response มาว่า "my iot Protocol ready !".

ส่วนการส่งค่าจาก IoT ไปเก็บบน server, เราจะใช้คำสั่ง WriteDashboard(float val); โดยส่ง GET request เป็นเลขทศนิยมไป.

เมื่อข้อมูลบันทึกลงใน database แล้ว, server จะส่ง response กลับมาว่า "record data ok" ครับ.

เราสามารถ print response ที่ส่งกลับมาดูได้ผ่านทาง serial monitor นะครับ.

ส่วนอันสุดท้าย จะเป็นการเรียกดูเวอร์ชันของ library ของเรา อันนี้ก็แล้วแต่เราจะกำหนดเลยครับ

```
bool myprotocol::begin(const char *ssid, const char *passw){
    _ssid = ssid;
    _passw = passw;

5.    int _cnt = 0;

    WiFi.begin(_ssid, _passw);
    Serial.print("myprotocol connecting..");

10.   while((WiFi.status() != WL_CONNECTED) && (_cnt <= CONN_RETRY_LIMIT)){
        delay(200);
        Serial.print(".");
        _cnt++;
    }

15.   if(WiFi.status() == WL_CONNECTED){
        _conn = true;
        Serial.println("Connected !");
    }else{

20.   _conn = false;
        Serial.println("Connection Timeout.");
    }

    return _conn;

25. }

String myprotocol::sayhi(){

    WiFiClient client;

30.   if(client.connect(host, port)){
        _str = "GET /";
        _str += " HTTP/1.1\r\n";
```

```

    _str += "Host: ";
    _str += host;
35.   _str += ":";
    _str += port;
    _str += "\r\n";
    _str += "Connection: keep-alive\r\n\r\n";

40.   client.print(_str);

    delay(LAG_TIME);

    while(client.available()){
45.     _res = client.readStringUntil('\r');
    }

    return _res;

50.   }else{
        //..
    }
}

55.   String myprotocol::WriteDashboard(float val){

        _val = String(val);

        WiFiClient client;
60.   if(client.connect(host, port)){

        _str = "GET /write/";
        _str += _val;
        _str += " HTTP/1.1\r\n";
65.   _str += "Host: ";
        _str += host;
        _str += ":";
        _str += port;
        _str += "\r\n";
70.   _str += "Connection: keep-alive\r\n\r\n";

        client.print(_str);

        delay(LAG_TIME);

75.   while(client.available()){
        _res = client.readStringUntil('\r');
    }
}

```

```
80.         return _res;
        }else{
            //Nothing..
        }
    }

85. String myprotocol::getVersion(){
    return pn_libversion;
}
```

พอเราสร้าง library ไว้ใช้เองได้แล้ว เราก็มาเริ่มเขียนโปรแกรมที่จะนำไป upload ลงบน IoT board ครับ.

เปิด Arduino IDE ขึ้นมา แล้วก็วาง code นี้ลงไปได้เลย (อันนี้ก็เตรียมไว้ให้ใน github เหมือนเดิมครับ).  
ไฟล์ชื่อ 'sendtomyprotocol.ino' นะครับ.

sendtomyprotocol

```
/*
Code for Internet of things device
connect to server.
Coder : Isaranu Janthong
Created : 2017.Nov.11
*/

#include <myprotocol.h>

const char *ssid = "";
const char *passw = "";

String response;
float value;

myprotocol myiot;

void setup() {

  Serial.begin(115200);
  bool conn = myiot.begin(ssid, passw);
```

Done compiling.

Archiving built core (caching) in: /var/folders/7r/47j7bfxd43l65tdbl4l2gykc0000gn/T  
Sketch uses 232193 bytes (22%) of program storage space. Maximum is 1044464 bytes.  
Global variables use 32172 bytes (39%) of dynamic memory, leaving 49748 bytes for 1

```

#include "myprotocol.h"

const char *ssid = "";
const char *passw = "";

5. String response;
float value;

myprotocol myiot;

10. void setup() {

    Serial.begin(115200);
    bool conn = myiot.begin(ssid, passw);

15. if(conn){
        Serial.println("myprotocol connected.");
    }else{
        Serial.println("re-connect again.");
20.    }

    response = myiot.sayhi();
    Serial.print("Are you ready ? :" + String(response));

25. response = myiot.getVersion();
    Serial.println("myprotocol library version is " + String(response));
}

void loop() {

30. response = "";

    value = random(40,50);
    response = myiot.WriteDashboard(value);
35. Serial.println(response);

    delay(5000);
}

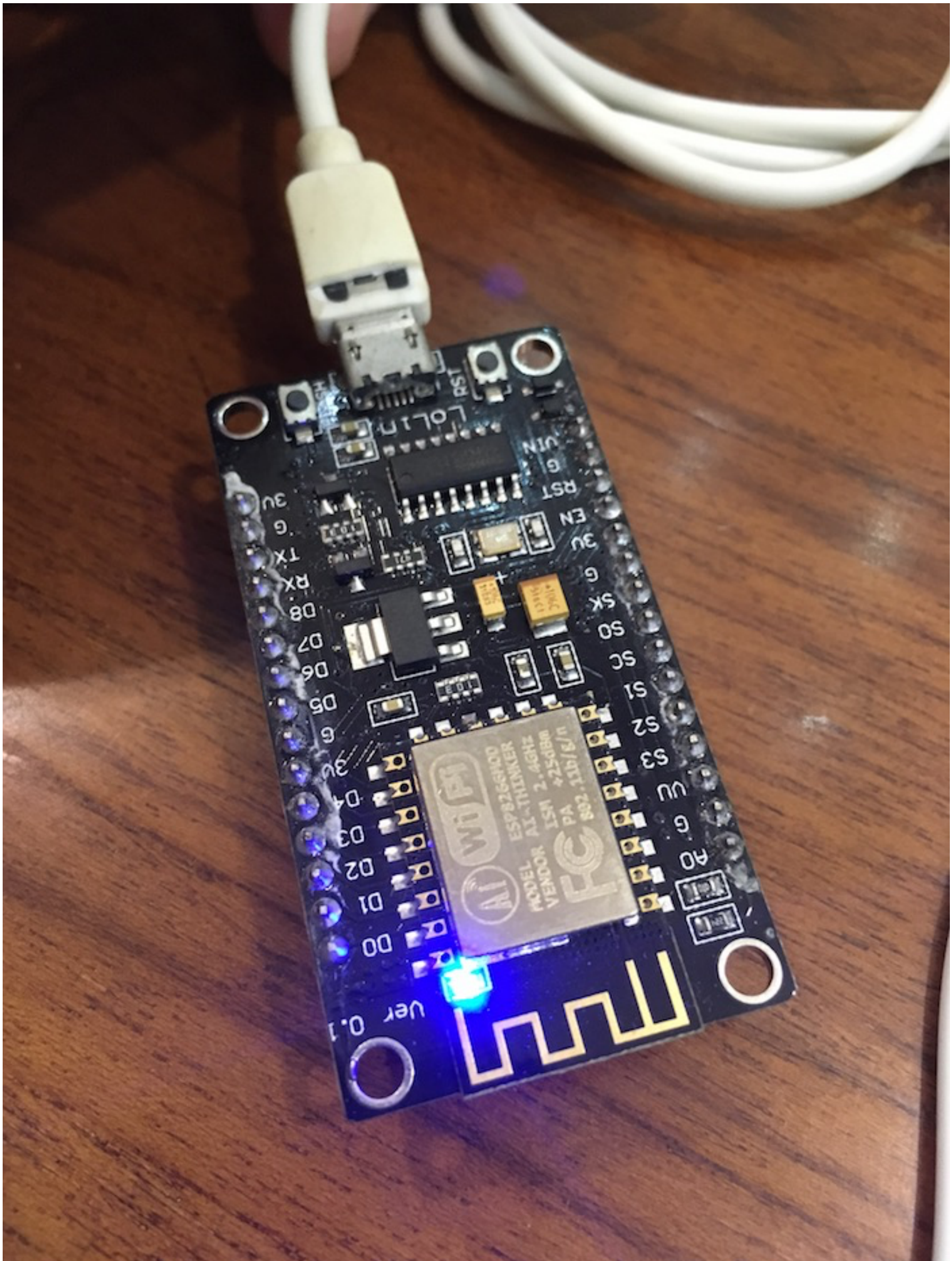
```

หลักก็คือการเรียก library 'myprotocol.h' ที่เราเขียนและบันทึกไว้ที่ arduino > libraries มาใช้งาน

จากนั้นก็ใช้คำสั่ง begin(ssid, passw); ในการสั่งให้ IoT board เชื่อมต่อ WiFi ครับ. เข้ามาใน void loop(), จะเป็นการส่งข้อมูลไปยัง server เราด้วยคำสั่ง WriteDashboard();



ถัดไปนำ code ด้านบน, upload ลงในบอร์ด IoT เราได้เลย. Upload ผ่านโปรแกรม Arduino IDE ครับ.

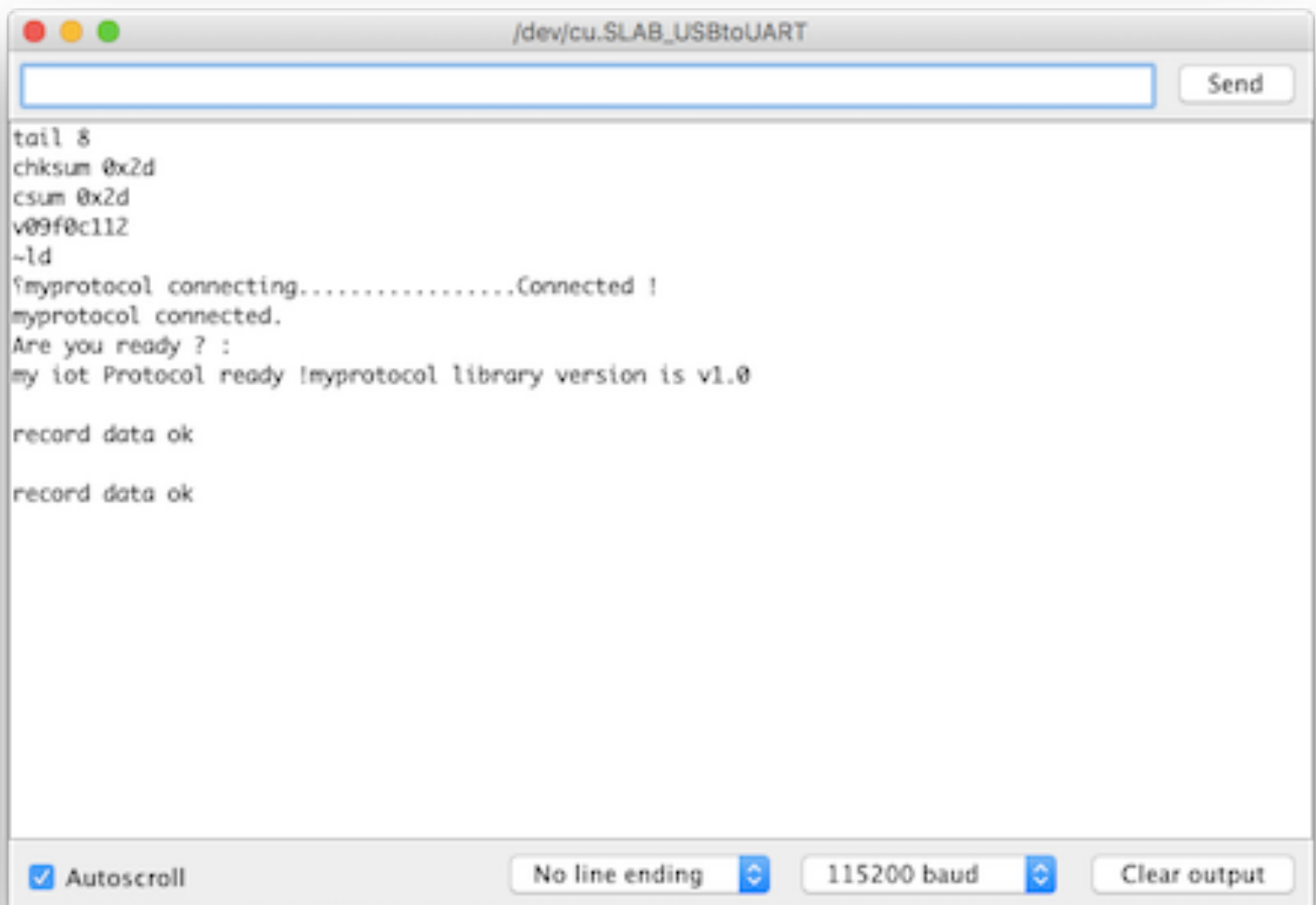


## [7. ทดสอบส่งค่าจาก IoT มายัง Server database]

หลังจาก upload code เสร็จเรียบร้อยแล้ว ก็มาลองรันกันดูเลย. ให้เปิด Serial monitor บน Arduino IDE เพื่อดูค่าด้วยนะครับ.

เมื่อเริ่มการทำงาน จะเป็นการ connect WiFi. หลังจาก connect ได้แล้ว ก็จะเริ่มส่งค่า.

ในตัวอย่าง, ผมให้ random เลขแล้วส่งไป. ในการใช้งานจริง ก็สามารถนำค่าอื่นๆที่อยากจะส่ง มาแทนใน ตัวแปรชื่อ 'value' นี้ได้เลยครับ.



## [8. Dashboard HTML]

มาถึงฝั่ง browser กันบ้างครับ, เช่นเคย code เก็บอยู่ใน github. ไฟล์ชื่อ "mychart.html" ในบทความนี้ผมจะอธิบายเฉพาะบางส่วนนะครับ

เริ่มจาก code ในส่วนของการ render chart.

ในครั้งนี้จะใช้ Chart.js โดยจะสร้างเป็นกราฟเส้น ที่ render ด้วยข้อมูล 30 data ล่าสุดที่ส่งมาจาก IoT.



```

var path, dataset=[], timeset=[];
var chart_canvas = document.getElementById('mychart').getContext('2d');

getdata(30);
5. drawchart(dataset, timeset);

setInterval(function(){
    getdata(30);
    chartupdate();
10. },1000);

/* script สำหรับเรียกข้อมูลจาก app.js */
function getdata(_datasize) {
    path = 'http://xx.xxx.xxx.xxx:4000/read/'; /* ใส่ IP address server ของ
เราลงไปครับ */
15.   path += _datasize;
    path += '?output=jsonp&callback=?';

    $.getJSON(path,{
    })
20.   .done(function(data){
        var output;
        output = JSON.stringify(data);
        var output_json = '{"dataset":"' + output + '"';
        console.log(output_json);
25.   parsejson(output_json);
    });
}

function parsejson(_txtParse){
30.   var parsed = JSON.parse(_txtParse);
    var parsed_obj = parsed.dataset;

    var browser_unix = new Date();
    var browser_time = new Date();
35.

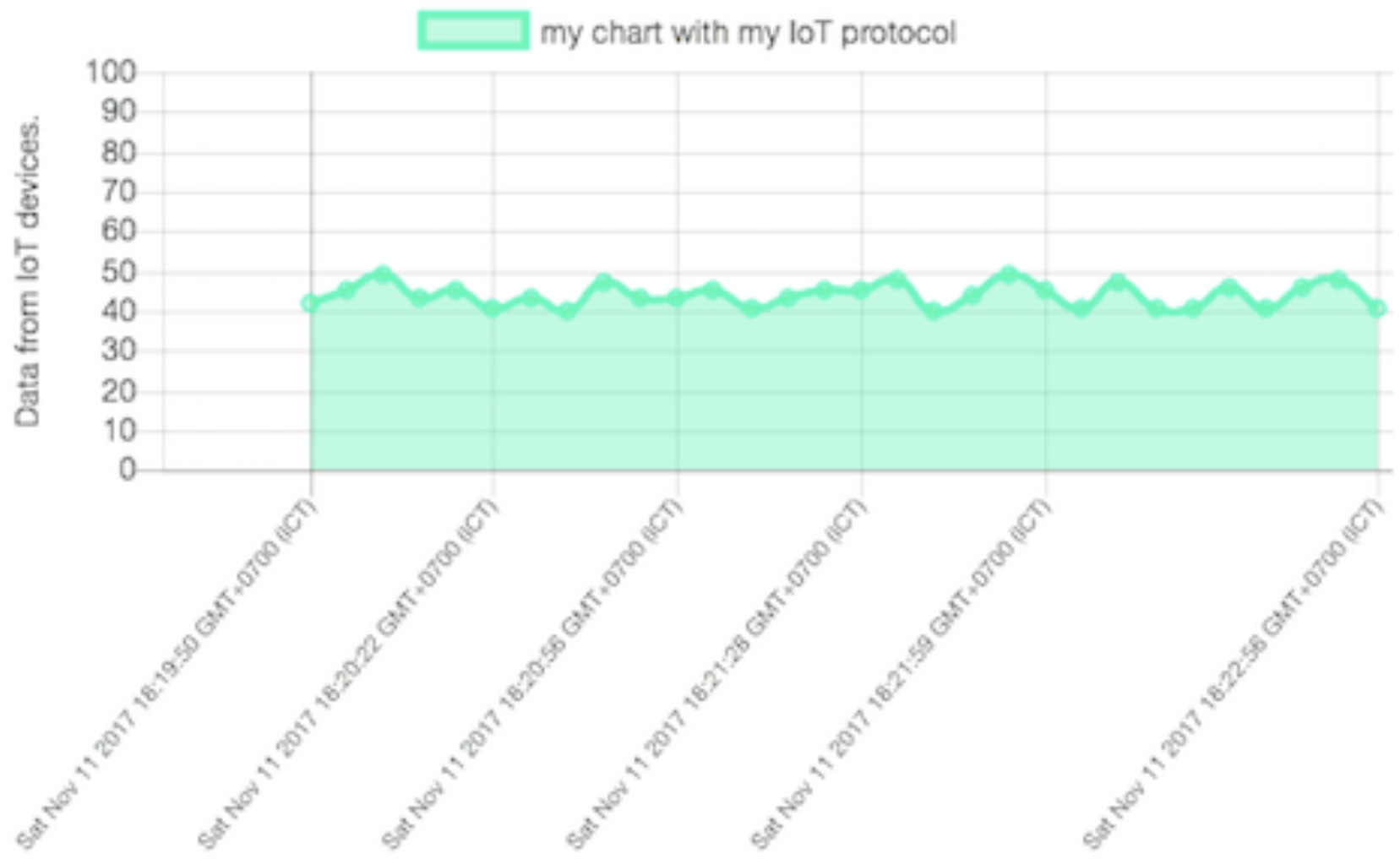
    for(var i in parsed_obj){
        dataset[i] = parsed_obj[i].data;
        timeset[i] = new Date(Number(parsed_obj[i].recordTime));
    }
40.   dataset = re_numbering(dataset);
    timeset = re_numbering(timeset);

}

```

# Simple Internet of Things Protocol

powered by [loTtweet.com](https://loTtweet.com)



## [9. ทดสอบทั้งระบบ]

มาทดสอบทั้งระบบกันดีกว่าครับ.

กลับไป SSH ของ GCP เรา, ให้เข้าไปที่ folder ที่เก็บ app.js

แล้วใช้คำสั่งด้านล่างนี้เพื่อรัน app.js ของเราครับ

```
$ forever start app.js
```

คำสั่งนี้จะทำให้ app.js ของเรา รันแบบ background. และเมื่อมี error ที่ทำให้โปรแกรมหยุด forever ก็จะสั่งให้รันขึ้นมาเองใหม่โดยอัตโนมัติครับ.

เมื่อนำทุกอย่างมาเปิดดูพร้อมกัน ก็จะได้แบบนี้เลยจ้า.

## [สรุป]

สรุปแบบสั้นๆ แล้วกันนะครับ

ถ้าต้องการสร้าง website เพื่อรับข้อมูลจาก IoT device ที่ส่งมาจากที่ใดก็ได้ในโลกนี้ (ที่ต่ออินเทอร์เน็ตได้)

สามารถทำได้ง่ายๆ จากบทความนี้เลยครับ.

หรือถ้าต้องการใช้ server ที่อื่น ก็ทำได้ครับ เช่น AWS หรือ Microsoft Azure. ก็สามารถนำไปประยุกต์ใช้ได้ครับ.

สำหรับผู้สนใจการสร้าง IoT protocol และ Website dashboard สำหรับแสดงผล, สามารถติดต่อมาที่ผมได้เลยนะครับ ยินดีเปิดเป็นคอร์สอบรมครับ.

จบจ้า.

ขอบคุณทุกการติดตามนะครับ :D

Isaranu.

 Follow @isaranu 19

 Like  Share



---

## CODE.ISARANU.COM

เว็บไซต์รวบรวมการ coding ต่างๆจากงาน production website และประสบการณ์อื่นๆ

นำมาแบ่งปันให้ชาว geek ได้เรียนรู้และนำไปใช้งานกันครับ.

## FOLLOW ME AT



code.isaranu.com

subscribe with code.isaranu.com

SIGN UP