2017 ARM Design Contest 設計競賽

初 賽 報 告

作品名稱：公車、公車站與中央監控中心的物聯網系統

隊名：騰雲的微中子

中華民國 106 年 9 月 8 日

1、摘要

本作品以公車站裝置、公車上裝置以及中央監控伺服器，以這三項設備互相構成一個公車與公車站的物聯網系統。公車上裝置藉由 RFID 技術，可以記錄每位乘客的上下車，並可以以此估算出公車目前的空位數；公車站裝置讓想要搭乘的民眾可以登記要搭車，此資訊會回傳到中央伺服器，公車上裝置可以藉由公車自身定位和下一站的車站資訊，來讓司機知道下一站是否有乘客要上車。所有的公車和車站的裝置皆透過中央伺服器連結，並與政府資料串連，達成一個公車與車站的物聯網監控系統。

2、作品功能與實用性

在現代這個提倡環保、減碳的社會，公車、捷運是許多人通勤的重要工具，在 105 年台北市交通統計中，平均每日公車營運數量高達 3159 輛，而平均每車每日的搭乘人次則是 411 人，大眾運輸與人們顯然已密不可分。然而，公車肇事事件卻屢見不鮮。其中，近十年來，公車因靠站疏忽而肇事的事件，多達上百件。我們希望能改善現狀，並讓搭公車的體驗更加美好。

改善智慧生活

本系統改善了現有的架構資訊呈現不足，我們補足了公車站即時等待人數、公車座位空位數量，並結合現有台北市公開數據中的公車、公車站等資訊，重新讓所有數據彼此連結。

以往乘客可能想要搭乘某班公車，但可能公車站人數太多，公車站剛好停了好幾班公車，或是公車司機一時恍神，造成乘客眼巴巴地望著公車離去，心中滿滿的焦急與無奈，卻一籌莫展，原本計畫好的行程就此打亂，習慣的生活步調無法跟上。

但事實上這個情境是可以避免的。公車司機僅需稍微移動視線留意我們公車物聯網系統中提供的資訊，就可以輕易地知道下一站有人想上車，甚至可以知道到底有多少人。

如此一來即便公車站人數再多、公車停靠數量好幾台，公車司機也不會再遺漏任何一位想搭車的乘客，無乘客想搭乘的公車更可以減少時間並不佔用公車站空間；乘客也能更悠閒思考想搭的公車，知

悉每個公車的到達時間、座位數及乘客人數，在搭車時能智慧地輕易選擇自己偏好的公車。在車次人數眾多的公車站，這些細節相當重要，每台公車的停靠及每位乘客的上下車都是讓公眾運輸更加便捷的關鍵。

結合未來趨勢

我們可以預料未來的交通系統更加自動化與人工智慧化，當一切都是數據紀錄在網路中，經過大數據分析和人工智慧的技術實踐，世界絕對會變得很不一樣。

例如我們將車站有多少等待乘客以及公車上有多少空位數量納入大數據的一環，未來的城市監控 AI 便可以以此做決定，如果公車空位都滿了，或是每個車站都有很多人在等待，可能就要加派公車班次。又例如，平常深夜都有公車在路上跑，目的是為了服務深夜才能回家的上班族，但是通常深夜只會有少數幾人在等待公車，有時甚至沒有，如果某一天剛好深夜整條路線都沒人要搭車，深夜班次就可以減少，不僅替公車營運省錢，也非常環保。

此外，雖然此專案是針對公車的物聯網系統，但是當我們可以更精確掌握某個乘客在某個時間點搭車，某個車站會上車、會下車，並結合其他交通工具得到的數據，例如火車、捷運、飛機，當一切都聯繫在一起，我們可以進行非常深入的社會行為學分析，並以理論來實踐更完善的智慧城市，使得科技進步不斷加速，人們生活達到最高理想。

3、設計創意性

緣起

目前生活中已有智慧公車亭及等公車 app，且廣泛使用，但智慧公車亭及等公車 app 兩者皆僅能顯示到站時間，等車乘客對於車上人數及座位數並不知曉，無法確定自己是否適合搭乘這班公車(尤其是身心或行動不便的乘客)，滿心期盼等待，當公車來到站牌前，才發現車上早已擠滿了人，這時前面的等車時間毫無意義，面臨無奈的窘境。

最初想法

因此，第一個想法提出，便是紀錄車上人數並同時顯示於智慧公車亭，可讓乘客獲得更多資訊並視情況做出更有利的選擇。悠遊卡辨識本就利用 RFID 系統，假使上下車都要刷卡的情況，就能掌握車上確切人數，在目前系統上新增此功能是非常可行，甚至也許原本公車系統就有紀錄人數僅是沒有提供給乘客，但這是讓乘客判斷自己是否要搭車的重要因素，可讓整個公車系統更智慧。

想法衍伸

反過來思考，公車司機同樣有資訊不充足的問題，無法確切得知是否有人甚至多少人要上車，僅能以揮手判斷，但公車乘客數有限，尖峰時段熱門路段的公車時常是擁擠得令人不舒服，對於乘客感受

及司機體會都是不良現象；且公車靠站停車其實有相當大的危險性，因機車、自行車時常位在公車的死角，導致公車停靠站時發生意外釀成悲劇。因此若能同時於智慧公車亭紀錄是否有人想搭車及預備搭車的乘客人數，公車司機將能更準確選擇停靠車站，若是能與其他同號司機進行溝通分配乘客，更能有效進行分流，減少多台同號車同時停靠的危險性和車上時間空間的浪費。

4、作品應用技術

本作品所有程式碼皆放在 Github 上

https://github.com/tigercosmos/bus-iot

檢視程式碼：

```
git clone https://github.com/tigercosmos/bus-iot.git
cd bus-iot
```

i. 系統整合部分

使用台北市開放數據，獲得台北市所有公車站、公車所有相關數據，例如所有公車列表、公車型號、公車到站時間、公車路線等等，以此為基礎開發我們自有的 API，中央監控中心會處理所有 API，並以 API 串連物聯網中的所有設備。

> API 都是以 PHP 來實現，DB 採用 NOSQL 的方式與 firebase 連結，伺服器環境為 Linux。完整程式碼請參閱
>
> https://github.com/tigercosmos/bus-iot/tree/master/center/backend。

a. 公車上裝置

- 某乘客持號碼為 `rfid_number` 的卡，上車或下車編號為 `bus_id` 公車時，使用本 API 傳到控管中心，並用來確認乘客是否從公車站上車，以及計算公車上空位數量
  GET URL: */rfid.php?bus_id={bus_id}&rfid={rfid_number}*

```
// rfid
[
    bus_id: {
        rfid_number
    }
]
```

- 某路線 `path_id` 編號為 `bus_id` 的公車， API 會自動根據公車目前的位置，回傳下一站有沒有人想搭這路線的車，呈現在裝置上給司機參考，讓司機不會遺漏乘客
  GET URL: */next{path_id}/{bus_id}/next.json*

```
// bus_next
{
    bool // if next step has people
}
```

b. 公車站裝置

- 某個人想在 station_id 的公車站，搭乘 path_id 號路線的公車，就會在公車站裝置上點選，裝置會將訊息以 API 傳至控管中心，該站要搭乘某路線的人數加一，並會在公車上裝置呈現給司機看。
  GET URL: */stop_people.php?path={path_id}&station={station_id}*

```
// stop_people
{
    station_id
    path_id
}
```

- 公車站 station_id 的裝置，用此 API 來取得這個車站會有那些公車經過，並回傳 bus_path_id 和 bus_path_name
  GET URL: */stop_people.php?path={path_id}&station={station_id}*

```
// station_path
{[
    bus_path_id
    bus_path_name
]}
```

- 公車站 station_id 的裝置，在某乘客點選要搭乘某路線 path_id 公車，裝置在 CALL 完 stop_people API 後，會再呼叫此 API，並在裝置上呈現該路線公車，最快抵達的那輛公車的目前資訊。
  GET URL: */stop_people.php?path={path_id}&station={station_id}*

```
//stop_bus_info
{
    path_id: [{
        station_id: [{
            estimate_time,
            next_bus_type,
            next_bus_people
        }]
    }]
}
```

c. 監測系統

- 中央監測系統的介面呼叫此 API，得到所有公車的列表，並呈現在選單上。
  GET URL: */bus-all/path.json*

```
//path
{
    bus: [ bus_path ]
}
```

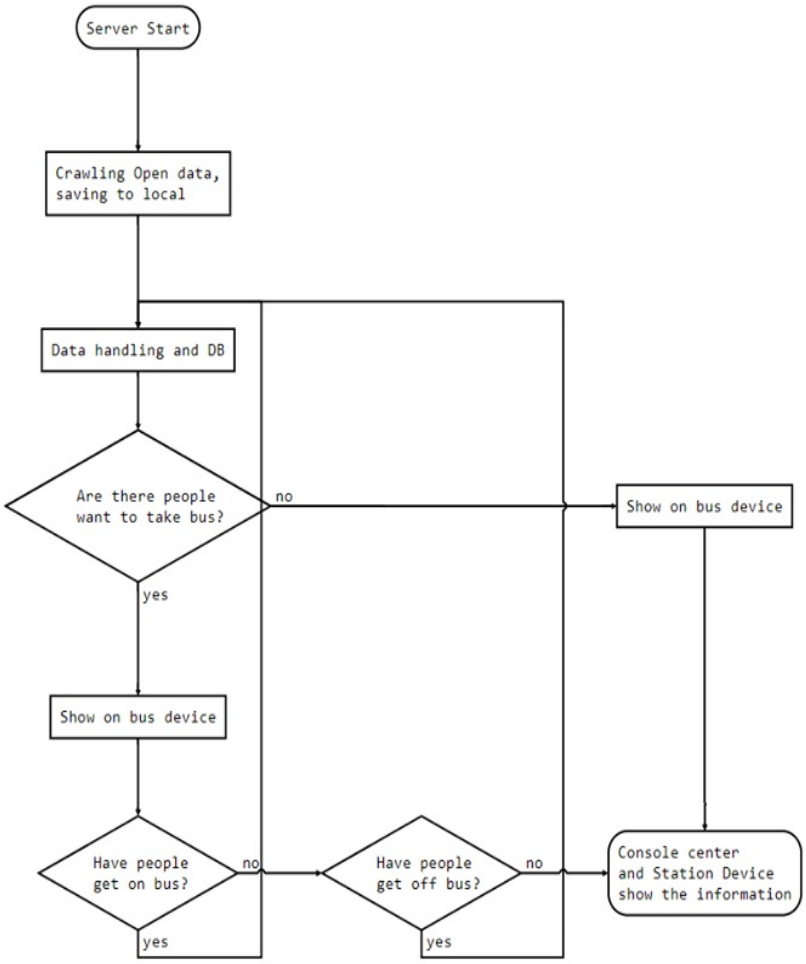
- 中央監測系統的介面在選擇某路公車後，會呼叫此 API，並依據得到的回傳資訊，在介面上畫出目前此路線的所有公車與公車站的實況。
  GET URL: */bus-all/monitor_bus_info/{path_id}.json*

```
// monitor_bus_info
{
    bus_number,
    station_number,
    order,
    stations: [{
        name,
        id,
        station_people
    }]
    buses: [{
        id,
        no,
        position,
        bus_peple,
    }]
}
```

d. 後端程式

整體邏輯

每分鐘會執行一次，其中車站與公車皆為全部處理，在此流程圖中只以一個代表。



API Curl

所有 API 的呼叫，包含對 Firebase 的 DB 處理，都以 `Curl` 處理。

```php
$ch = curl_init();
//set curl options
curl_setopt($ch, CURLOPT_URL, "$base_path");
curl_setopt($ch, CURLOPT_HEADER, false);
curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
//patch optiomns
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PATCH");
curl_setopt($ch, CURLOPT_POSTFIELDS,$data);
//execute curl
curl_exec($ch);
//close curl
curl_close($ch);
```

ii. 公車站裝置介紹

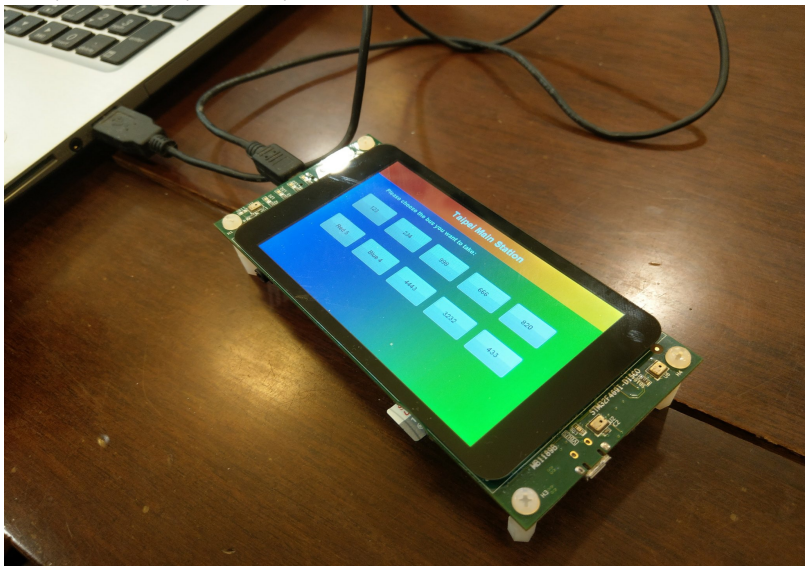
> 公車站裝置以 Keil 開發， Src 請參閱
>
> https://github.com/tigercosmos/bus-iot/tree/master/station/project/Src

使用 STM32F469I Discovery 開發版，預設是假定此開發板作為觸控裝置裝設在公車站，乘客在公車站時能與此裝置進行互動。

裝置圖

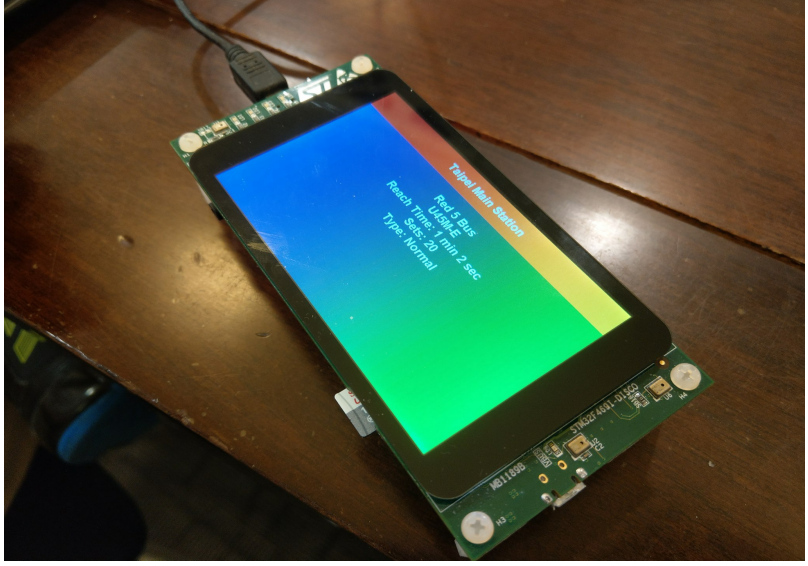
用此開發版原本就有的 TFT LCD 當作螢幕介面，使用者一進去之後會有選單，點選進入某路公車，代表他想搭乘此公車，此時裝置會傳 API 給中央伺服器，中央伺服器會記錄此資訊並更新資料庫。

公車站想搭乘的選單



點擊後會呈現目前該路線最快抵達的公車的資訊給使用者看，包含公車的編號、是否為低底盤、車上空位數量、還有多久會到等等。

點擊進去之後的資訊



實作技術

STM32F469I Discovery 的開發採用 `STemWin` 函式庫來做介面呈現，包含顯示文字、顯示按鈕等，此函式庫也同時處理了觸控部分的底層設定。

透過 `STemWin` 中 `GUI.h` 來做到一般背景、文字等處理。

```
// Backgreound
GUI_SetBkColor(GUI_BLACK);
GUI_DrawGradientH(0, 71, 800, 480, GUI_BLUE, GUI_GREEN);
// Text setting
GUI_SetTextMode(GUI_TM_TRANS);
GUI_SetColor(GUI_WHITE);
// Print Text
GUI_SetFont(&GUI_Font20B_ASCII);
GUI_DispStringHCenterAt("Some text here...", 250, 90);
```

透過 `STemWin` 中 `BUTTON.h` 、 `WM.h` 來實現按鈕的就面與觸控事件控制。

```
// Button Demo Smaple

/* Bus button binding */
BUTTON_Handle busBt[bus_number];

/* Create Button */
busBt[i] = BUTTON_CreateEx(x, y, x_size, y_size, WM_CF_SHOW, 0, i);
/* Button font */
BUTTON_SetFont(busBt[i], &GUI_Font20_ASCII);
/* Button name */
BUTTON_SetText(busBt[i], bus_name[i]);
/* Background color when button is not pressed */
BUTTON_SetBkColor(busBt[i], BUTTON_CI_UNPRESSED, GUI_DARKGREEN);
/* Background color when button is pressed */
BUTTON_SetBkColor(busBt[i], BUTTON_CI_PRESSED, GUI_GREEN);

/* Show buttons */
GUI_Exec();

/* Get pressed key to know which button is pressed */
if (GUI_GetKey() == Button_Key){
    // Call API return info via wifi
}
```

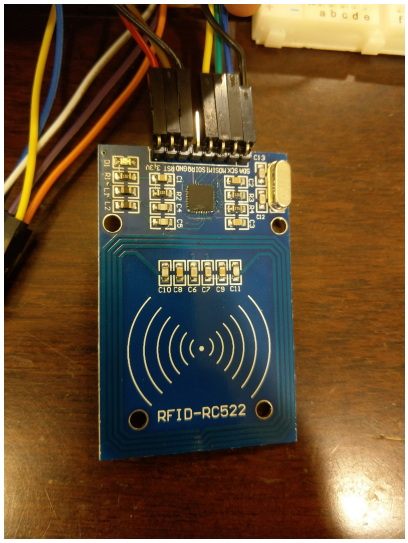
iii. 公車上裝置介紹

公車上裝置使用 `mbed` 開發環境，專案內容請參閱

https://github.com/tigercosmos/bus-iot/tree/master/bus/rfid

RFID 辨識系統使用市面上易取得的 RC522 模組（ 13.56 Mhz ，傳輸速度為中低速，常用於小卡片）連接主板 STM32F469I ，使用 mbed 開發環境。

RFID晶片



程式碼：

```
RfChip.PCD_Init();
uint8_t uid[30];

while (true)
{
    LedGreen = 1;
    lcd.Clear(LCD_COLOR_BLACK);
    lcd.SetBackColor(LCD_COLOR_BLACK);
    lcd.SetTextColor(LCD_COLOR_WHITE);
    lcd.DisplayStringAt(0, LINE(4), (uint8_t *)"Start", CENTER_MODE);
    // Look for new cards
    if (!RfChip.PICC_IsNewCardPresent())
    {
      wait_ms(500);
      continue;
    }

    // Select one of the cards
    if (!RfChip.PICC_ReadCardSerial())
    {
      wait_ms(500);
      continue;
    }
}
```

RFID 晶片的初始化並設定等待讀取時間，並使用 mbed 中的 LCD 函式顯示於觸控螢幕上方便觀看。

```
/* Print Card UID */
lcd.DisplayStringAt(0, LINE(5), (uint8_t *)"Card UID: ", CENTER_MODE);
for (uint8_t i = 0; i < RfChip.uid.size; i++)
{
  sprintf((char*)uid, " %X", RfChip.uid.uidByte[i]);
  lcd.DisplayStringAt(0, LINE(i+6), (uint8_t *)&uid, CENTER_MODE);
}

/* Print Card type */
uint8_t piccType = RfChip.PICC_GetType(RfChip.uid.sak);
lcd.DisplayStringAt(0, LINE(7), (uint8_t *)"PICC Type: ", RIGHT_MODE);
lcd.DisplayStringAt(0, LINE(8),
                    (uint8_t *)RfChip.PICC_GetTypeName(piccType),
                    RIGHT_MODE);
```

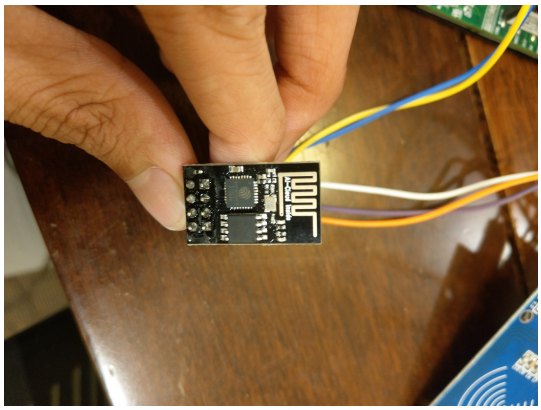
讀取到 RFID 的卡片後，顯示識別元（UID）及卡片類型於觸控螢幕。

iV. 開發版的網路連接介紹

> 網路模組使用 mbed 開發環境，專案內容請參閱
>
> https://github.com/tigercosmos/bus-iot/tree/master/Util/wifi

開發版的網路連接使用市面上易取得的 ESP8266 晶片模組（ESP-01）連接主板 STM32F469I ，使用 mbed 開發環境。

ESP8266 晶片



程式碼：

```
void scan_demo(WiFiInterface *wifi)
{
    WiFiAccessPoint *ap;

    printf("Scan:\r\n");

    int count = wifi->scan(NULL,0);

    /* Limit number of network arbitrary to 15 */
    count = count < 15 ? count : 15;

    ap = new WiFiAccessPoint[count];
    count = wifi->scan(ap, count);

    delete[] ap;
}
```

掃描可用 AP (SSID 及密碼設定在 json 檔供讀取)，並使用 Serial 連接直接在電腦端觀看搜尋情況（可使用 Tera Term 等類似軟體,鮑率為 9600 ，設定 new line 為 auto ）。

```
int ret = wifi.connect(WIFI_SSID,WIFI_PASSWORD, NSAPI_SECURITY_WPA_WPA2);
if (ret != 0) {
    printf("\r\nConnection error\r\n");
    return -1;
}
```

掃描到相符 AP 後進行連線,相關連線資訊也可使用函式列出。

```
void http_demo(NetworkInterface *net)
{
    TCPSocket socket;
    // Open a socket on the network interface, and create a TCP connection
    socket.open(net);
    socket.connect("url", 80);
    // Send a simple http request
    char sbuffer[] = "GET / HTTP/1.1\r\nHost: url\r\n\r\n";
    int scount = socket.send(sbuffer, sizeof sbuffer);
    // Recieve a simple http response and print out the response line
    char rbuffer[64];
    int rcount = socket.recv(rbuffer, sizeof rbuffer);
    socket.close();
}
```

使用 socket 連線並使用 HTTP 傳輸

V. 監控介面介紹

監控介面用來給中央控管中心的人員做監控，雖然說物聯網本身的概念是可以不需要人為介入，但有時仍不免發生意外，監管人員便可以用此介面來掌控第一時間的狀況，此外此介面也可以給一般有興趣的民眾查詢使用。

介面樣貌

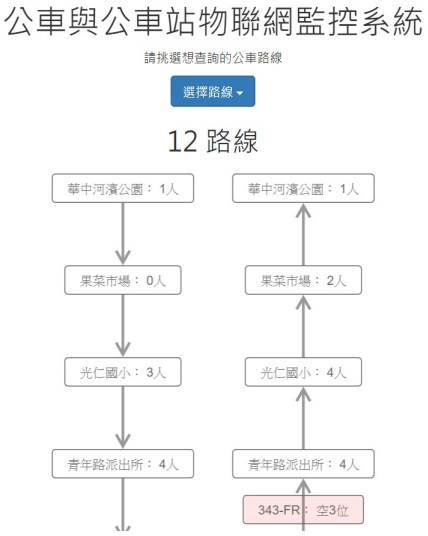
實際做好的介面，可以到以下網址實際操作

https://tigercosmos.github.io/bus-monitor/

進入選單

呈現某路公車現在狀況

包含路線中所有公車與所有公車站



介面技術

> 源始碼請參閱
>
> https://github.com/tigercosmos/bus-iot/tree/master/center/frontend

前端技術採用 jQuery 框架搭配 AntV G6 函式庫與 Bootstrap 樣貌模板。
AntV G6 可以劃出各種數學定義上的 Graph，我們使用他來繪製公車路徑圖和將公車放在路線上。

簡單事例如何處理， node 便是公車站， edge 就是車站的連結關係(方向)，如此一來就可做出公車路線圖

```
var data = {
   // Which implies stations
   nodes: [{
           "id": "node1",
           "x": 100,
           "y": 160
       },
       {
           "id": "node2",
           "x": 290,
           "y": 160
       }
   ],
   // Which implies buses
   edges: [{
       "id": "node1-node2",
       "target": "node1",
       "source": "node2"
   }]
};
// new G6 object
var net = new G6.Net({
   id: 'c1', // DOM ID
   height: 450 // DOM Height
});
net.source(data.nodes, data.edges);
net.render(); // refresh screen
```

5、實作設計考量

i. 系統整合層面

a. 處理資料

所有公車與公車站的資料多達數百 MB，處理起來非常費時，但是公車動態非常重視即時更新，每分鐘都要更新一次來確定公車確切位置，因而如何有效在短時間快速處理資料便非常重要。

同樣資料盡量只讀取一次，減少重複讀寫的次數，藉由 Firebase 已經優化過的 DB 架構，可以只針對特定 patch 進行快速讀寫，因此專案的資料型態比較適合以物件處理，故採用 NoSQL 而不採用 SQL DB。

伺服器每分鐘固定抓取台北市公車開放資料，並且依據物聯網裝置回傳的API，進行 DB 的更新，此部分只針對特定的 patch 進行修改，故可以達到快速更新。

b. API 架構

所有 API 設計皆採用 RESTful API 風格設計，資料皆採用 json 格式，主要是因為有以下優點：

- RESTful API 處理效率高，物聯網講求高效快速
- REST 可以同時接受多個 Requests，物聯網中可能同時會有很多裝置傳送 Requests
- REST 技術讓任何裝置只要上網就可以傳送 API，相容性非常高

- Json 格式在資料儲存上體積非常小，符合物聯網裝置的需求

c. 後端語言

因為此專案需要在短時間內完成，故選用 PHP 開發，每個 PHP 都可以當作是一支 API。但如果未來想要拓展規模，加入平行運算與分散式系統，採用 Java 、 Rust 等語言會比較安全且穩定。

ii. 開發板設計層面

a. emWin

emWin 已經將所有介面包裝成 API，我們不必再從底層慢慢刻劃畫面，對於開發速度很有幫助，他整合了 GUI 和觸控，甚至可以畫出視窗介面，搭配開發本原本就有的 TFT LCD 使用起來非常方便。

b. RFID

mbed 開發環境的 cookbook 有使用 RX 及 TX 傳輸數據讀取 RFID 的範例程式，但因使用的 RFID 晶片使用 SPI 傳輸所以不合用，另外從 mbed 社群找到先進針對這塊模組寫的額外 library ，並且搭配 LCD 的 library 完成 RFID 的讀取及偵錯。

c. WIFI MODULE

mbed 開發環境的 cookbook 依然是使用 RX 及 TX 傳輸 Wi-Fi 的範例程式，而我們拿到的 Wi-Fi 擴展版則是 SPI 傳輸，但因 SPI 轉 Wi-Fi 傳輸的步驟較為繁複，因此選擇易取得的 RX 及 TX 傳輸的 ESP8266 ，搭配其 driver 並使用 Serial 的方式從電腦端監控 Wi-Fi 連線情況完成無線連線的設定。

6、實作結果

當我們組的理念達成共識時，就發現為了達成理想上的公車物聯網系統，真正完成我們的目標，不只需要專注於開發版的部分，還有網頁、使用者介面、資料庫、伺服器及 API 等需要準備，必須善加分配人力，因此分成伺服器後端及資料庫 API 的串接、開發版 GUI 及網頁前端、 RFID 讀取及 Wi-Fi 連接等三個部分，希望達成理想上的公車物聯網系統開發。

而初步結果架構相當完整，上述提到的部分皆有成果且可以整合使用：從台北市政府開放資料抓取資料，且藉由 Firebase 已經優化過的 DB 架構速度進行快速讀寫、API 皆為 Json 格式因此通用且明瞭、監控介面簡潔清晰、開發版GUI的使用者畫面採用設計思考，以使用者立場設計容易使用的介面、 RFID 可明確辨識不同張感應卡、Wi-Fi 可進行 HTTP 傳輸。將以上實際整合至現今公車亭及等公車 app，將會讓公車這項大眾運輸更有智慧更親民。

7、結論

我們這組曾有使用過 Arduino、 linkit 7688、 RPI 等微控制板等經驗，且有組員相當熟悉前端及網路系統的操作，但此次皆為第一次使用 ARM 架構的 STM32 晶片且附有觸控螢幕的板子，對其架構、語法、開發環境及討論社群等都不甚熟悉，好似發現一片新大陸般驚奇，但為達成我們的理念需要開發編寫的部分較多，且暑假大家都有各自的規畫安排（實驗室、實習等），為了專案的進行過程能更順利並分工，一位組員二話不說自掏腰包多買了一塊同樣的開發版。

在這次準備的過程中，不僅對 STM32F4 系列開發版的架構更了解，並更深入學習 `Keil` 、 `emWin` 、 `RTOS` 、 `mbed` 的開發環境及其各式各樣的特點，並對編寫 `php` 、 `DB` 架構有更多探討，思慮物聯網所需的網路系統及建構一個達成我們理念的專案所需考慮的眾多細節。為了 coding 犧牲許多睡眠時間而日夜顛倒的大家，都在此次比賽收穫極大。

8、參考文獻

- emWin API Document
- Ant G6 Official website
- Firebase Document
- 嵌入式系統建構：開發運作於STM32的韌體程式
- MBED RFID-RC522 Document
- MBED ESP8266 Document

tags: `STM32F469I Discovery` , `bus-iot` , `iot-device`