\*\* 需要安裝firebase-admin套件:

npm install firebase-admin –save

(Firebase流程按照上一次筆記4運作)

\*\* Firebase列表資料 複雜且重要!!!

今天資料庫我們沒透過前端到後端再到DB，而是直接後端跟DB操作!

以下就可以寫進DB囉!

let express = require('express')

let app = express()

let parser = require("body-parser")

// 初始化firebase

var admin = require("firebase-admin");

var serviceAccount = require("./serviceAccountKey.json"); // 改成自己的路徑

admin.initializeApp({

credential: admin.credential.cert(serviceAccount), //告訴google你是誰

databaseURL: "https://node-backend-train.firebaseio.com"

});

// 操作資料庫

let database = admin.database();

let ref = database.ref("/user")

ref.set({ // 寫進DB

name: 'ply', email:'ply@aaa.com'

}, function(errors){

if(errors){

console.log("Error");

}else{

console.log("Success");

}

});

app.listen(3000, function () {

console.log('Example app listening on port 3000!')

})

以下就可以更新DB資料囉!!!

let express = require('express')

let app = express()

let parser = require("body-parser")

// 初始化firebase

var admin = require("firebase-admin");

var serviceAccount = require("./serviceAccountKey.json"); // 改成自己的路徑

admin.initializeApp({

credential: admin.credential.cert(serviceAccount), //告訴google你是誰

databaseURL: "https://node-backend-train.firebaseio.com"

});

// 操作資料庫

let database = admin.database();

let ref = database.ref("/user")

ref.update({ // 寫進DB

name: 'abc', time:(new Date()).getTime()

}, function(errors){

if(errors){

console.log("Error");

}else{

console.log("Success");

}

});

app.listen(3000, function () {

console.log('Example app listening on port 3000!')

})

\*\* push (處理列表資料，DB會自動生成unique的Key不會重複)

(但也可以用自己的邏輯產生那個key再用set或update)

(my: 一般框架不是都內建用ID嗎? 這裡可以這樣嗎?)

像是可以處理使用者的留言

let express = require('express')

let app = express()

let parser = require("body-parser")

// 初始化firebase

var admin = require("firebase-admin");

var serviceAccount = require("./serviceAccountKey.json"); // 改成自己的路徑

admin.initializeApp({

credential: admin.credential.cert(serviceAccount), //告訴google你是誰

databaseURL: "https://node-backend-train.firebaseio.com"

});

// 操作資料庫

let database = admin.database();

let ref = database.ref("/messages")

// push

ref.push({ // 更新DB

content: 'Hello World', author: 'Steven',

time:(new Date()).getTime()

}, function(errors){

if(errors){

console.log("Error");

}else{

console.log("Success");

}

});

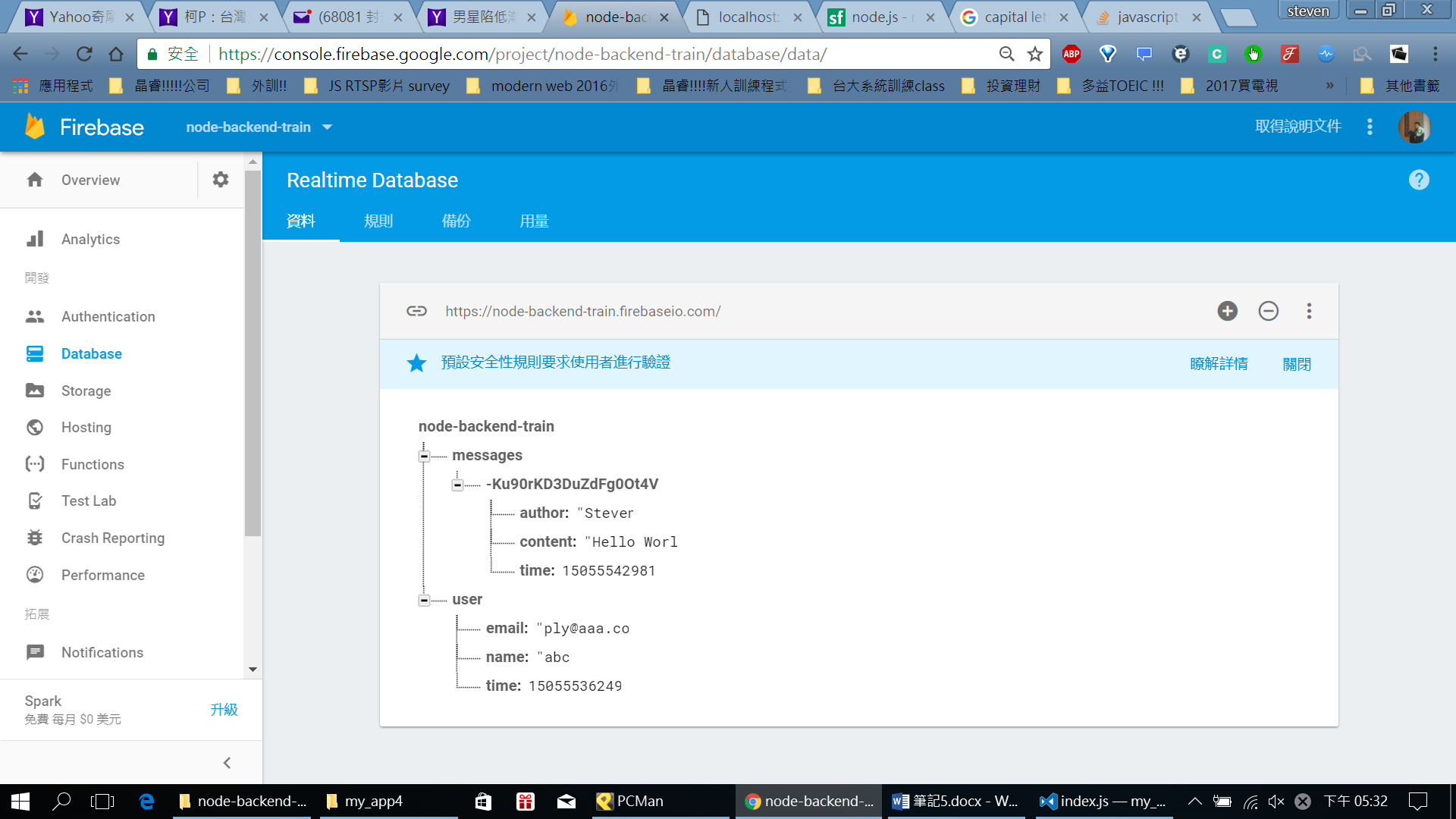
app.listen(3000, function () {

console.log('Example app listening on port 3000!')

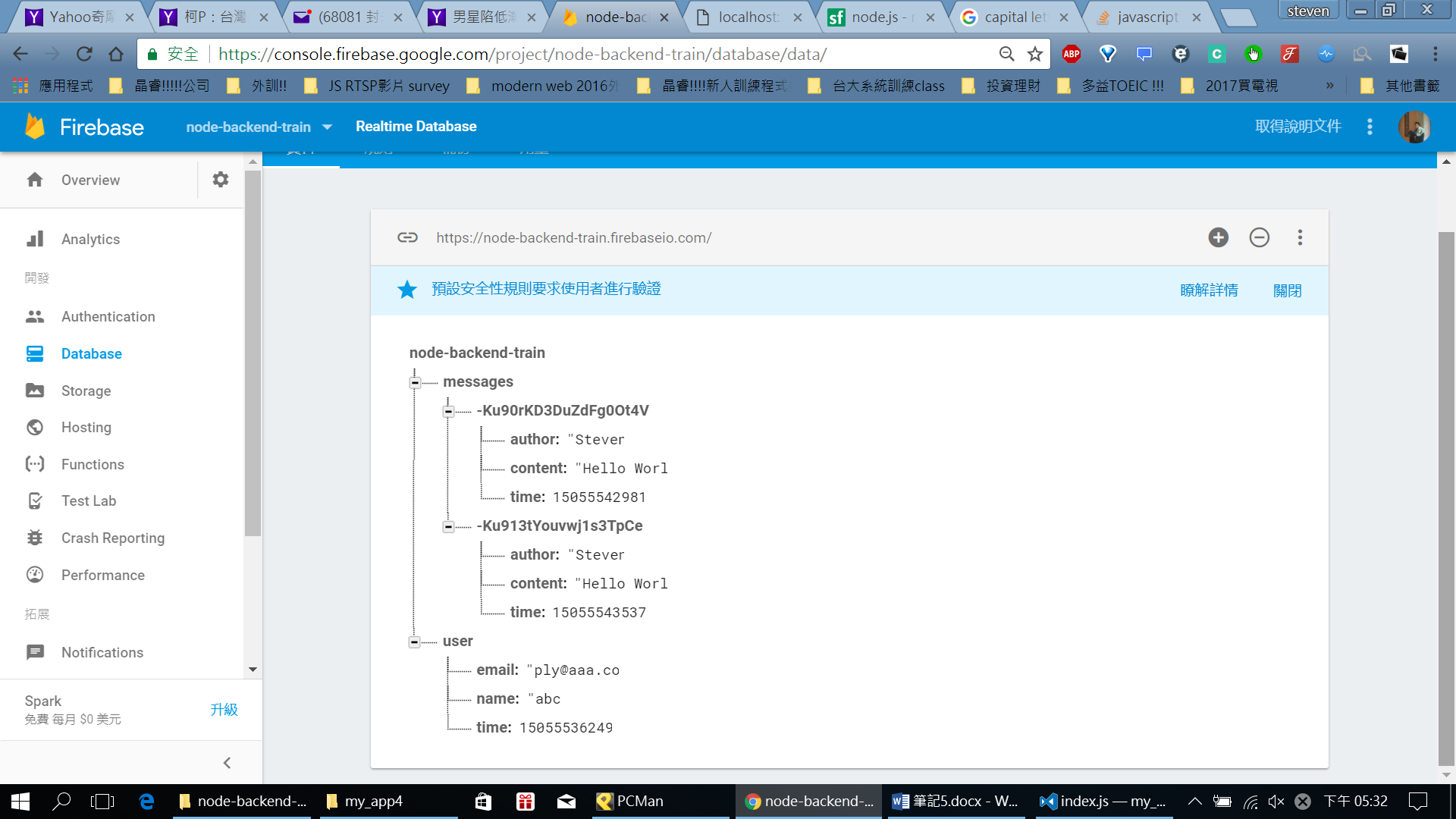
})

每一次多push一筆進去:

第一次啟動:



在重啟Server: 多push一次



\*\*可以再利用set更改某一個列表的資料:

let express = require('express')

let app = express()

let parser = require("body-parser")

// 初始化firebase

var admin = require("firebase-admin");

var serviceAccount = require("./serviceAccountKey.json"); // 改成自己的路徑

admin.initializeApp({

credential: admin.credential.cert(serviceAccount), //告訴google你是誰

databaseURL: "https://node-backend-train.firebaseio.com"

});

// 操作資料庫

let database = admin.database();

// 如果要更新某個message

// 但通常你會用code去取得中間那個key

let ref = database.ref("/messages/-Ku90rKD3DuZdFg0Ot4V/author");

ref.set("Steven Hsiao", function(errors){ // 指定要把author修改為前面那個值!

if(errors){

console.log("Error");

}else{

console.log("Success");

}

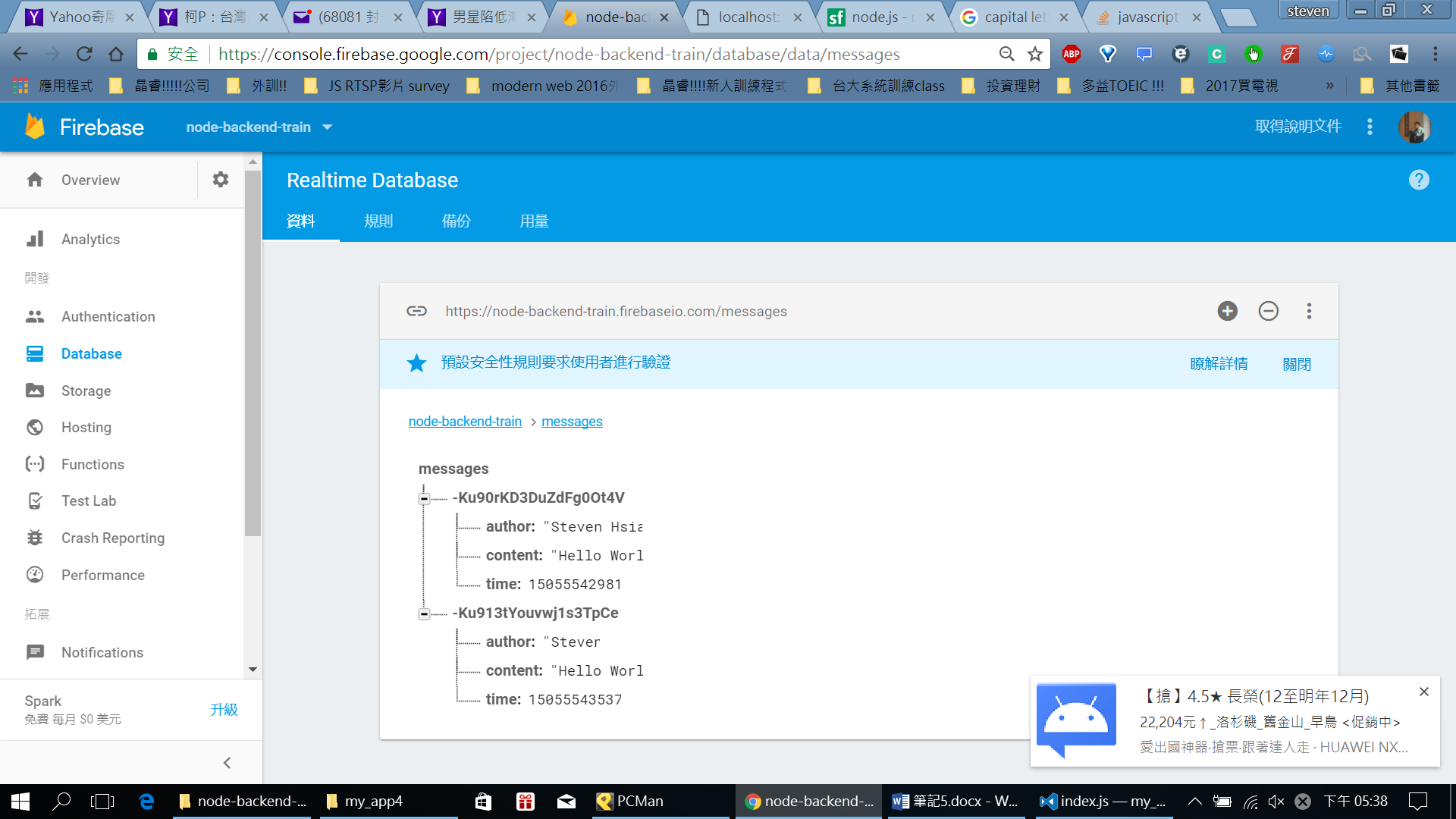
});

app.listen(3000, function () {

console.log('Example app listening on port 3000!')

})

則DB變為:



(以上可以再去研究這邊unique的方式)

\*\* 刪除資料 很簡單:

let ref = database.ref("/messages/-Ku90rKD3DuZdFg0Ot4V/");

ref.remove(function(errors){

if(errors){

console.log("Error");

}else{

console.log("Success");

}

});

\*\*\* Firebase 讀取資料流程! (跟一般很不一樣喔!!!)

1. 註冊資料讀取事件:

如以下code 就可以看到把所有messages的內容全部拿到囉!!!

// 操作資料庫

let database = admin.database();

// 讀取

let ref = database.ref("/messages")

// 成功取得資料，得到snapshot物件

ref.on("value", function(snapshot){

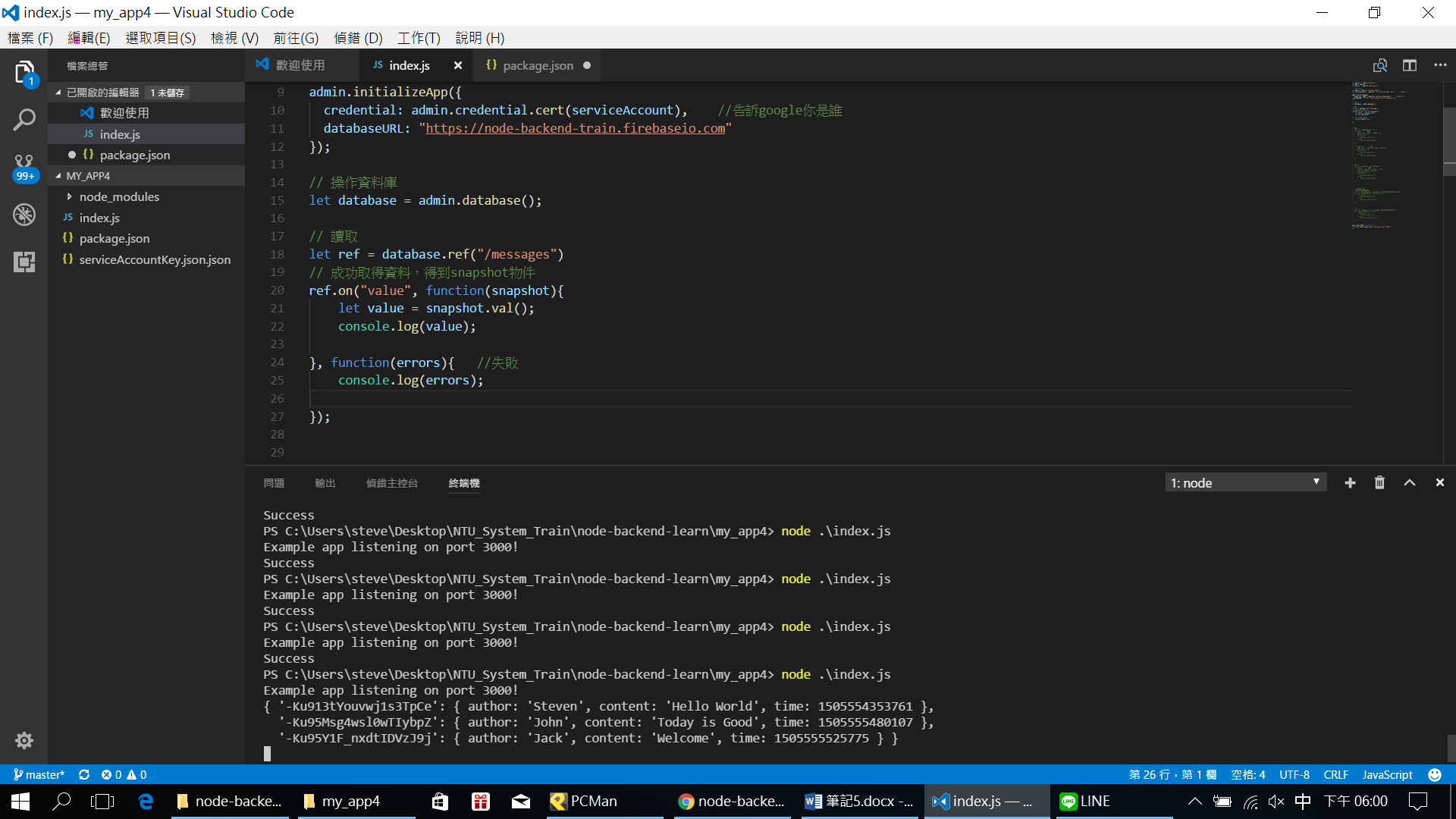
let value = snapshot.val();

console.log(value);

}, function(errors){ //失敗

console.log(errors);

});



可以利用設timeout來放心的一筆資料

// 讀取

let ref = database.ref("/messages")

// 成功取得資料，得到snapshot物件

ref.on("value", function(snapshot){

let value = snapshot.val();

console.log(value);

}, function(errors){ //失敗

console.log(errors);

});

// 再放一個新的資料，但其實是幾乎在同一時間發出這2個request 來push，也可以用timeout來等上一個做完來側

setTimeout(function(){

ref.push({

content: 'Test Firebase', author: 'Jack',

time:(new Date()).getTime()

}, function(errors){

if(errors){

console.log("Error");

}else{

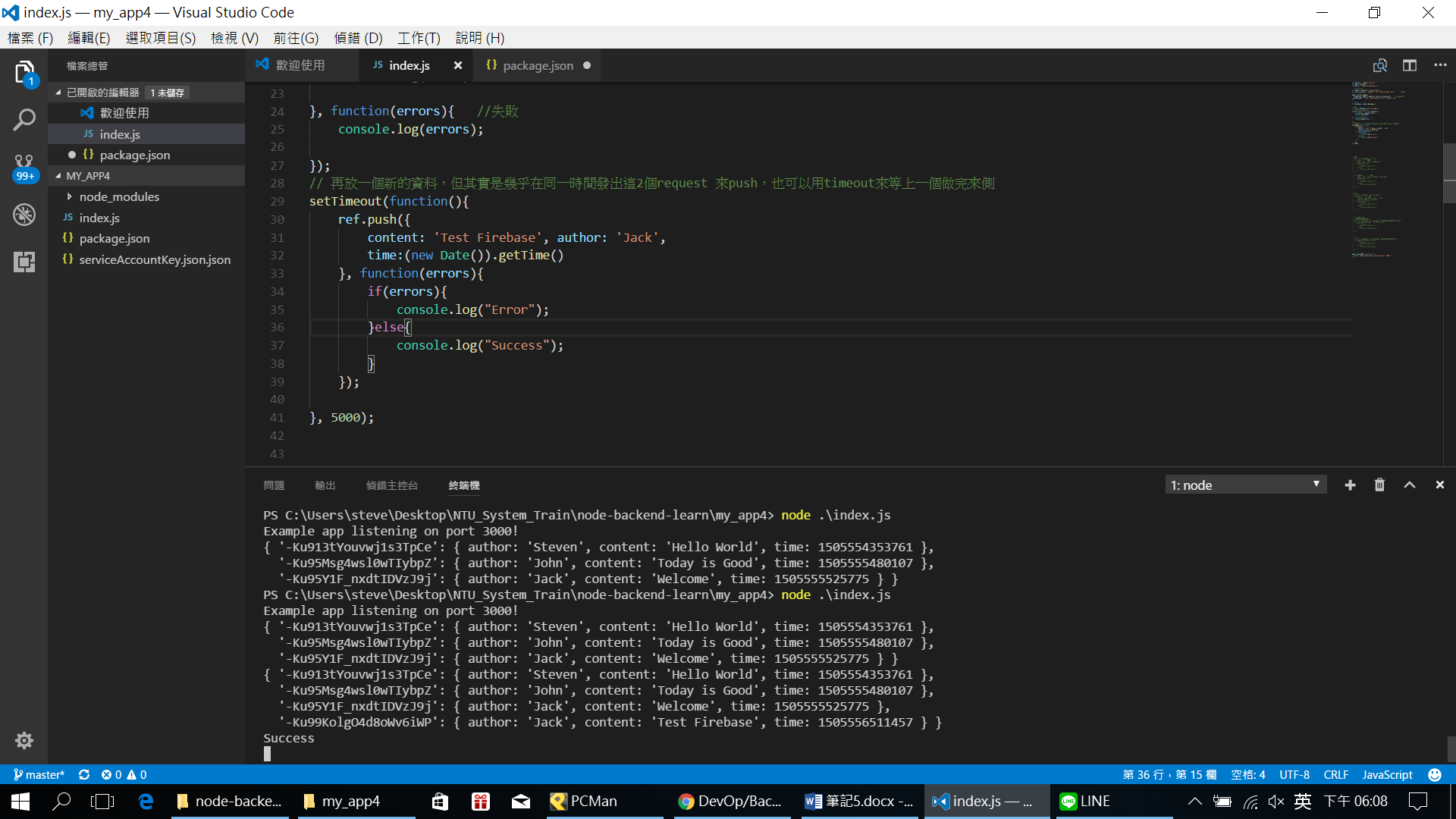
console.log("Success");

}

});

}, 5000);

會發現怎麼先印了一個是還沒push資料的，結果[push新的一筆完，怎麼又回去再跑一次秀資料!!



這是因為:

此即為事件註冊(on 代表一職等在那邊，如果該資料(messages)有任何調整，上面那個讀取資料都會再重新反映，是一個聆聽，是一個事件的處理)

>>> 若是不想要一有更新就坐(剛剛的ref.on)

改用once 其實背後也是事件的邏輯，但是他事後會幫你取消註冊

ref.once("value", function(snapshot){ //假設不想用聆聽的方式只要改變就重新再跑一次，則使用once

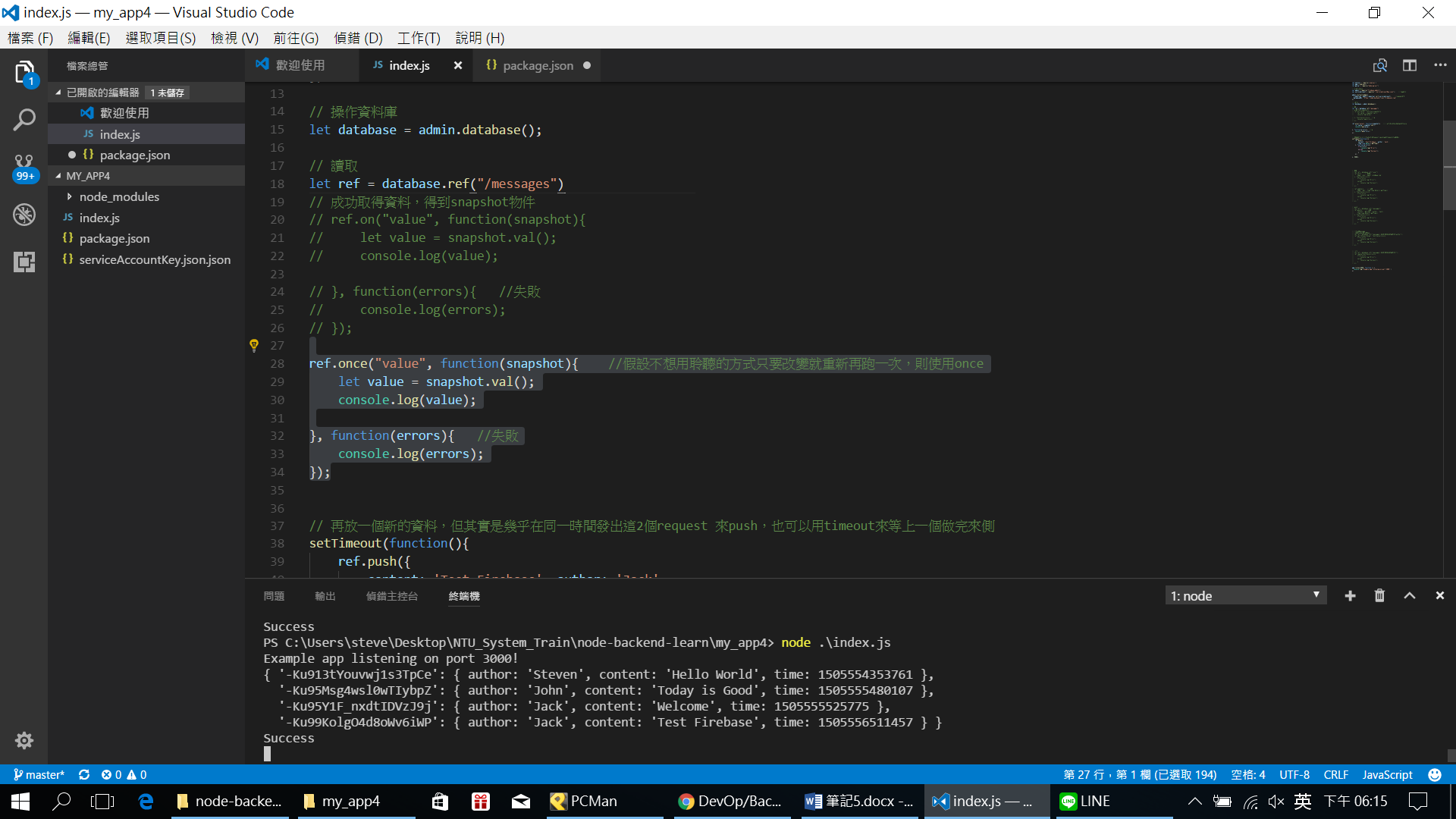
let value = snapshot.val();

console.log(value);

}, function(errors){ //失敗

console.log(errors);

});



\*\* 如果ref.on的事件不是value (依次讀近來), 改為用child\_added事件(他是一個一個讀取近來)

但因為這裡也是用on，所以如果value有新增料就會在新增 新增進來的那一筆!!! 這時候搭配order非常有用!!!!

let ref = database.ref("/messages")

ref.on("child\_added", function(snapshot){

let value = snapshot.val();

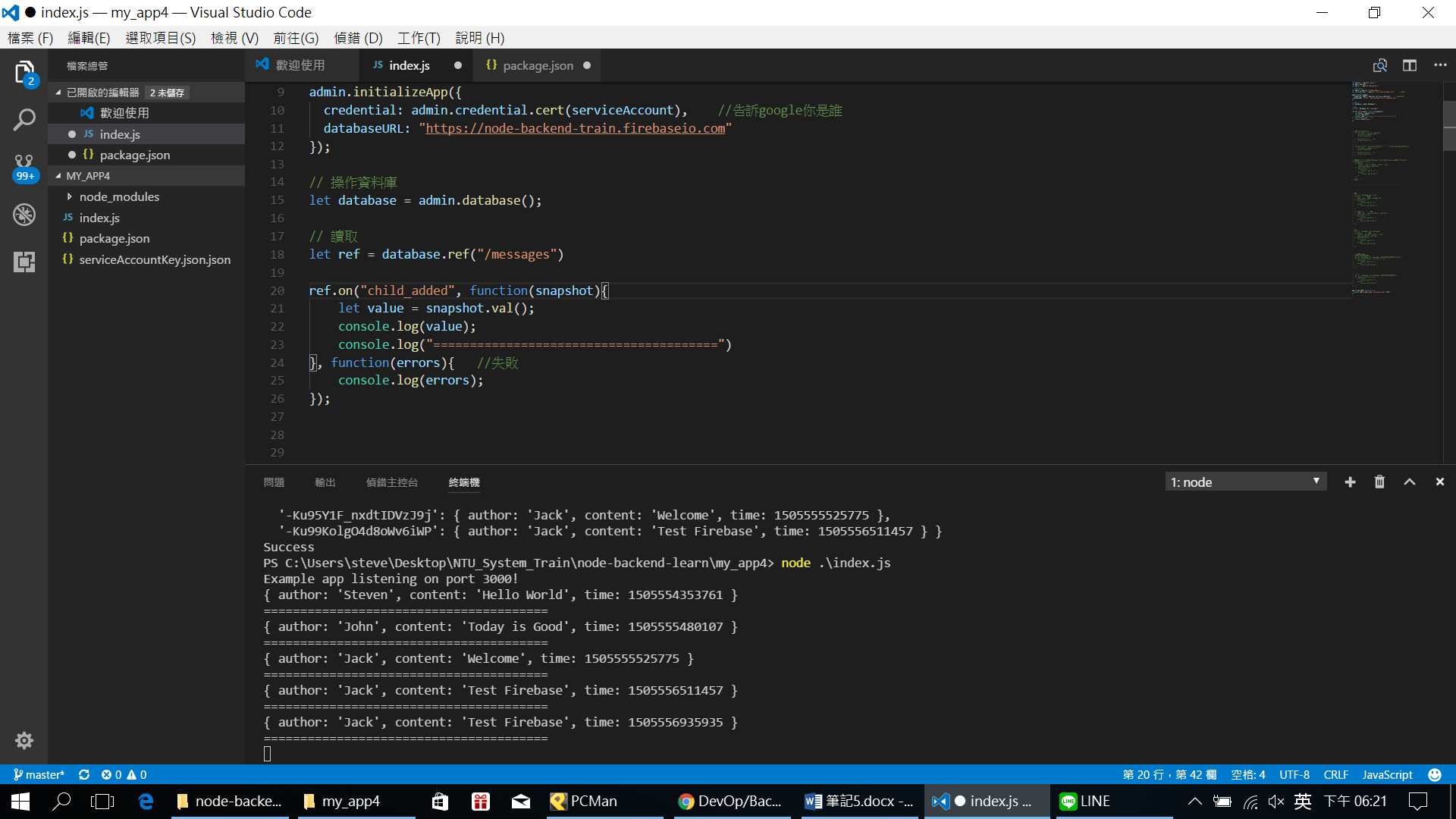
console.log(value);

console.log("=======================================")

}, function(errors){ //失敗

console.log(errors);

});



Order:

Firebase 有承諾，他生成的Key其實有照時間來牌喔!!! 所以如果你對key做order，其實有照時間來排喔!!!!

當然也可以根據你裡面的欄位內容作排序!

例如: orederbychild:

ref.orderByChild("author").on("child\_added", function(snapshot){

let value = snapshot.val();

console.log(value);

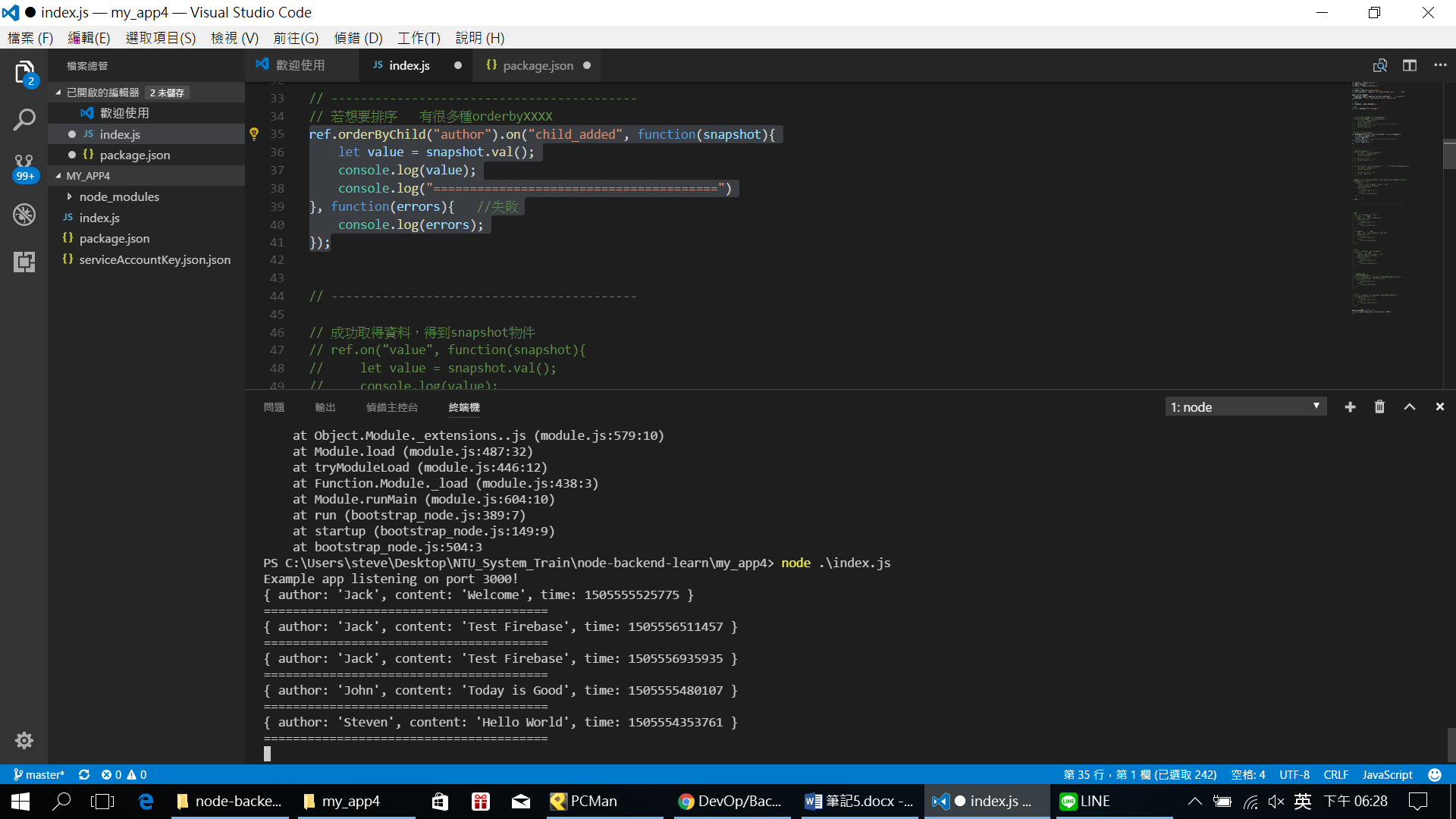
console.log("=======================================")

}, function(errors){ //失敗

console.log(errors);

});

會發現他傳回來ˇ的value是照指定的author排序:



對key排序:

ref.orderByKey().on("child\_added", function(snapshot){

let value = snapshot.val();

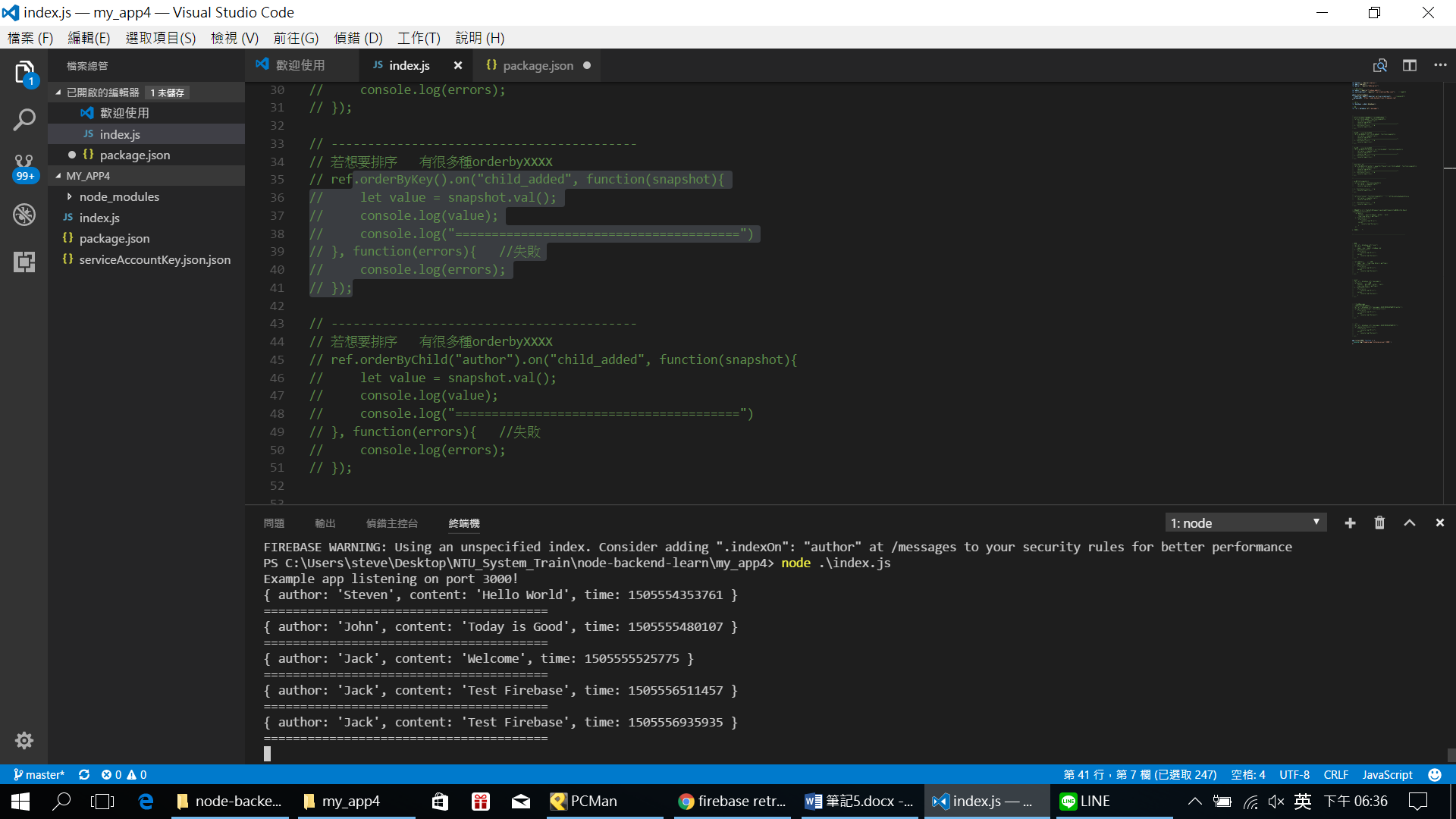
console.log(value);

console.log("=======================================")

}, function(errors){ //失敗

console.log(errors);

});



>> 還可以從列表中做一些篩選，算是簡單的搜尋功能:

ref.orderByChild("author").equalTo("Steven").on("child\_added", function(snapshot){

let value = snapshot.val();

console.log(value);

console.log("=======================================")

}, function(errors){ //失敗

console.log(errors);

});

可以看到只會列出author是”Steven”

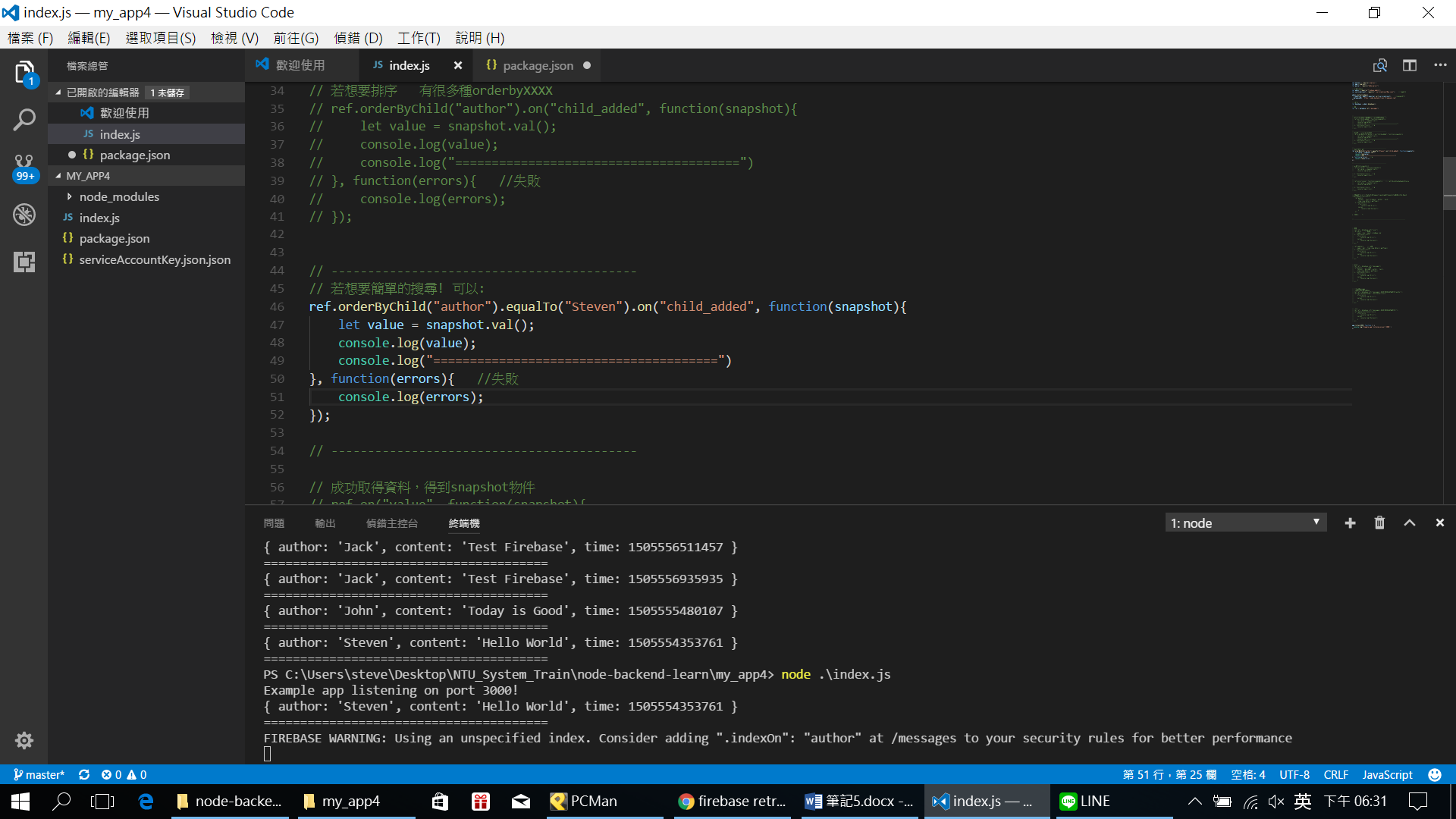
篩選機制:

orderByChild("author").equalTo("Steven")

是對author做排序，就對author做篩選

orderByKey.equalTo("Steven")

這樣就是對key做篩選



\*\*\*\*!!!!!\*\*\*\*

我們如果想從網頁連上後取得這些資料:

最終如果要使用留言板，我們把拿到的訊息放到全域陣列裡(server開著就會存在，)

// 這是一思考而已，當然也可以項傳統每一次傳進來都去資料庫拿!

// 最終如果要使用留言板，我們把拿到的訊息放到全域陣列裡(server開著就會存在，)

// 這是一思考而已，當然也可以項傳統每一次傳進來都去資料庫拿!

let messages = []

ref.on("value", function(snapshot){

let value = snapshot.val();

messages.push(value)

console.log(value);

console.log("=======================================")

}, function(errors){ //失敗

console.log(errors);

});

app.get("/", function(req, res){ //如此的模式，是一種可考慮的pattern，因為資料庫隨時都準備好，messages陣列都準備好，使用者連上後只是去拿到

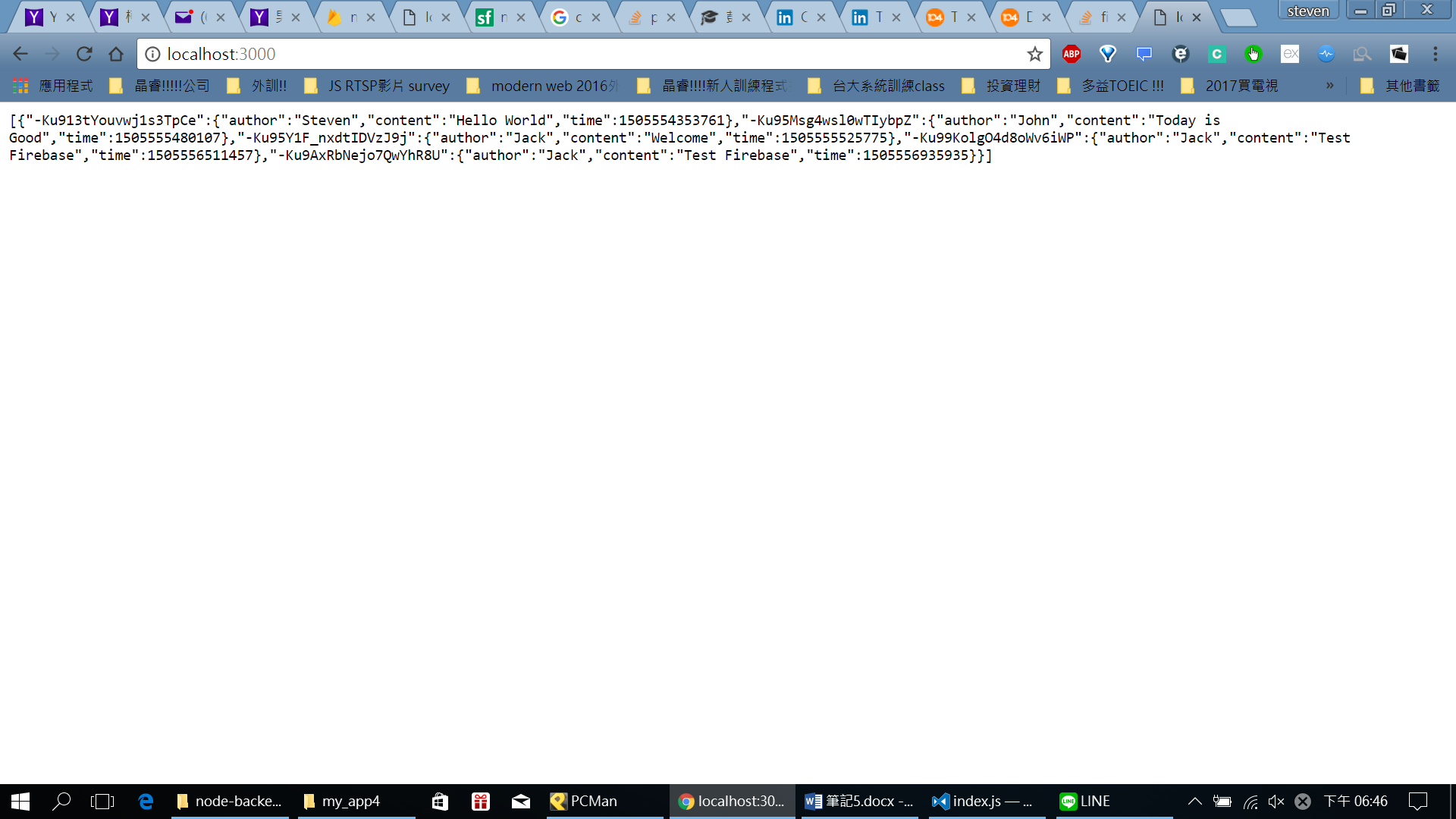
// messages陣列資料，並沒有去讀資料庫喔!!

res.send(messages)

});

如此使用者瀏覽器輸入localhost:3000後:

就可以看到:



記得:

//如此的模式，是一種可考慮的pattern，因為資料庫隨時都準備好，messages陣列都準備好，使用者連上後只是去拿到

// messages陣列資料，並沒有去讀資料庫喔!!