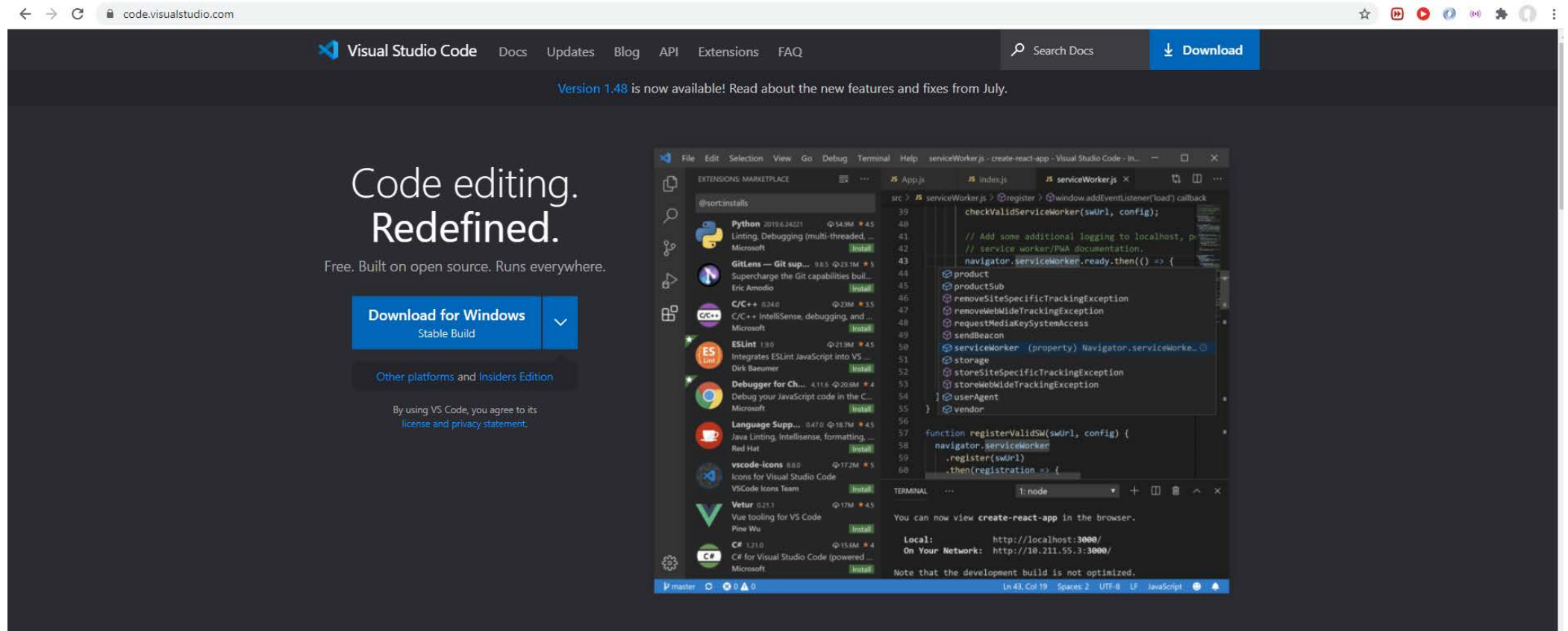


BASIC JAVASCRIPT

INSTALL IDE & EXTENSION

DOWNLOAD VS CODE

[HTTPS://CODE.VISUALSTUDIO.COM/](https://code.visualstudio.com/)



The screenshot shows the Visual Studio Code website in a web browser. The website has a dark theme and features the text "Code editing. Redefined." and "Free. Built on open source. Runs everywhere." Below this is a "Download for Windows" button with a dropdown arrow, and a link for "Other platforms and Insiders Edition". A note states: "By using VS Code, you agree to its license and privacy statement."


Below the website content is a screenshot of the Visual Studio Code application running. The interface shows the "EXTENSIONS: MARKETPLACE" sidebar on the left with a list of extensions including Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support, vscode-icons, Vetur, and C#. The main editor area displays a JavaScript file named "serviceWorker.js" with code for registering a service worker. The terminal at the bottom shows the command "node" and the output "You can now view create-react-app in the browser." and "Local: http://localhost:3000/".

DOWNLOAD NODE.JS


<https://nodejs.org/en/download/>


ดาวน์โหลดตัว LTS (Long Term Support)

LTS
Recommended For Most Users


Windows Installer
node-v12.18.4-x64.msi

Current
Latest Features


macOS Installer
node-v12.18.4.pkg


Source Code
node-v12.18.4.tar.gz

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	
macOS Binary (.tar.gz)	64-bit	
Linux Binaries (x64)	64-bit	
Linux Binaries (ARM)	ARMv7	ARMv8
Source Code	node-v12.18.4.tar.gz	

วิธีเช็ค ว่า **install** สำเร็จหรือไม่ โดยพิมพ์คำสั่ง **node -v** ทาง **command line**

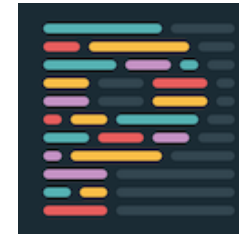
EXTENSION VS CODE



JavaScript (ES6) code snippets



Bracket Pair Colorizer



Prettier - Code formatter

INTRO

JavaScript คืออะไร

หลายคนคงเคยได้ยินหลายคนเรียก JAVASCRIPT เวอร์ชันใหม่ค่อนข้างหลากหลาย
เท่าที่เคยเจอมาก็มี ECMASCRIPT 2015, ES2015, ES6 เป็นต้น ชื่อพวกนี้มาจากไหน ?

JAVASCRIPT นั้นถูกกำหนดมาตรฐานโดย EUROPEAN COMPUTER MANUFACTURERS
ASSOCIATION (ECMA)

Edition	Name	Short Name	Public Year
JavaScript Edition 3			1999
JavaScript Edition 5			2009
JavaScript Edition 6	ECMAScript 2015	ES6	2015
JavaScript Edition 7	ECMAScript 2016	ES7	2016
JavaScript Edition 8	ECMAScript 2017	ES8	2017
JavaScript Edition 9	ECMAScript 2018	ES9	2018
JavaScript Edition 10	ECMAScript 2019	ES10	2019

ข้อมูลอ้างอิง <https://en.wikipedia.org/wiki/ECMAScript>

มีอะไรเปลี่ยนไปบ้าง

- ไม่ต้องเขียน SEMI-COLON แบ่งการทำงานโดยการเว้นบรรทัด
- รูปแบบการเขียนที่กระชับขึ้น
- เกิดตัวแปรชนิดใหม่ (LET & CONST)
- การเขียน FUNCTION แบบใหม่ (ARROW FUNCTION, DEFAULT PARAM)
- จัดการกับ ARRAY และ OBJECT ได้สละพัสดุมากยิ่งขึ้น (MAP, FILTER, REDUCE, DESTRUCTURING)
- การมาของ CLASS
- การมาของ MODULE
- รูปแบบการใช้งาน STRING แบบใหม่
- การจัดการ ASYNCHRONOUS แบบใหม่ (PROMISE, ASYNC/AWAIT)
- มาตรฐานการดึงค่าจาก API แบบใหม่ (FETCH)

ศึกษาต่อได้ที่ <https://exploringjs.com/es6/index.html>

UNIT 1

การใช้งาน JS เบื้องต้น

การสร้างไฟล์ และ การเรียกใช้งาน

✓ BASICJAVASCRIPT

JS simple.js

ทำการสร้างไฟล์ โดยใช้นามสกุลไฟล์เป็น .js เช่น ชื่อไฟล์.js

JS simple.js X

JS simple.js

```
1 console.log("Hello world")
```

ใช้คำสั่ง `console.log(<สิ่งที่ต้องการแสดงผลออกทางหน้าจอ>)`

```
PS D:\basicJavaScript> node ./simple.js
Hello world
```

```
PS D:\basicJavaScript> node simple.js
Hello world
```

เรียกใช้งานโดยผ่านทาง command line

`node <path file>`

หรือหากไฟล์อยู่ที่ root folder

`node <ชื่อไฟล์>`

UNIT 2

VARIABLES & OPERATOR

ตัวแปรของ JS จะเป็น DYNAMIC TYPING กล่าวคือ ชนิดตัวแปรจะเป็นอะไรก็ได้
เปลี่ยนได้เรื่อยๆ ตามค่าที่ตัวมันเก็บ

ชนิดตัวแปร หลักๆ คือ

- | | |
|-------------|--|
| 1. Number | -10,10.0,123 |
| 2. String | 'Your Name', "MY NAME", `this is a string` |
| 3. Boolean | true,false |
| 4. object | new Object(), {} |
| 5. function | function name (param) , (param) => { } |
| 6. array | [],[1,2,3,4],[1, "two" , '3'] |
| 7. Date | new Date(), new Date(2020,08,17) |

ค่าว่างในภาษา JS

- | | |
|--------------|--|
| 1. null | คือค่าว่าง ไม่มีอะไรเลยเหมือน ในภาษาอื่นๆ |
| 2. undefined | คือ ตัวแปรใหม่ที่ JS กำหนดค่าเริ่มต้นมาให้ คุณสมบัติเหมือน null ทุกอย่างแต่เพียงคนละตัวแปร |

```
null === undefined    // false
null == undefined     // true
```

ประกาศตัวแปร & การกำหนดค่าให้ตัวแปร

การประกาศตัวแปรมีอยู่ 4 วิธีคือใช้ **var**, **let**, **const** หรือจะไม่ใช่อะไรเลย เช่น

```
const MY_VAR = 7
```

```
var MY_VAR = 20
```

```
let MY_VAR = 20
```

```
MY_VAR = 20
```

*** ไม่แนะนำให้ใช้

วิธีเช็ค type ของ ตัวแปร จะใช้ **typeof**

ตัวอย่าง

```
var MY_VAR = 10  
var MY_STRING = 'my string'  
console.log(typeof MY_VAR);  
console.log(typeof MY_STRING);
```

number

string

> |

*** type ของ null คือ object

*** type ของ undefined จะไม่ใช่ null

ARITHMETIC OPERATORS

OPERATORS	ใช้สำหรับ
+	บวก
-	ลบ
*	คูณ
/	หาร
%	เป็นการหารแบบเอาเฉพาะ "เศษ" เช่น $13\%5=3$ หรือ $13\%3=1$
**	ยกกำลัง

ASSIGNMENT OPERATORS

OPERATORS	ใช้สำหรับ	ตัวอย่าง	เหมือนกับ
=	กำหนดค่า	x=10;	x = 10
+=	บวกค่าเดิมของตัวแปรด้วยค่าที่ระบุ	x=10; x += 8; // x=18	x = x + 8
-=	ลดค่าเดิมของตัวแปรด้วยค่าที่ระบุ	x=10; x -= 8; // x=2	x = x-8
/=	คูณค่าเดิมของตัวแปรด้วยค่าที่ระบุ	x=10; x /= 8; // x=2	x =x/8
*=	หารค่าเดิมของตัวแปรด้วยค่าที่ระบุ	x=10; x *= 8; // x=80	x = x*8
%=	นำค่าที่ระบุไปหารแบบเอาเศษด้วยค่าเดิมของตัวแปร	x=10; x %= 3; // x=1	x = x % 3
++	เป็นการเพิ่มค่าตัวแปรขึ้นไปอีก 1	x=10 ; x++; // x=11	x = x + 1
--	เป็นการลดค่าตัวแปรลงอีก 1	x=10; x--; //x=9	x = x-1

OPERATOR PRECEDENCE

()

**

* / %

+ -



ลำดับความสำคัญมาก

ลำดับความสำคัญน้อย



EXERCISE

```
let MY_VAR ;  
console.log(MY_VAR);  
MY_VAR = 15  
console.log(MY_VAR);
```

```
let MY_VAR = 5 + 5 * 2 - 5 * 2;  
console.log(MY_VAR);  
MY_VAR = (5 + 5) * 2 - (5 * 2);  
console.log(MY_VAR);
```

```
let MY_VAR = 7;  
const MY_VAR = 20;  
console.log(MY_VAR);
```

```
MY_VAR = 7  
MY_VAR += 7  
console.log(MY_VAR);  
MY_VAR = 20;  
console.log(MY_VAR);
```

```
var x = 20  
console.log(x)  
var x  
console.log(x)
```

UNIT 3

LOGICAL & CONDITIONS

LOGIC OPERATOR

OPERATORS	ความหมาย	ตัวอย่าง	ผลลัพธ์
<	น้อยกว่า	5 < 8	true
<=	น้อยกว่า หรือ เท่ากับ	8 <= 8	true
>	มากกว่า	5 > 8	false
>=	มากกว่า หรือ เท่ากับ	5 >= 8	false
==	เท่ากับ	5 == '5'	true
===	เท่ากับทั้งค่า และ type	5 === '5'	false
!=	ไม่เท่ากับ	5 != '5'	false
!==	ไม่เท่ากับทั้งค่า และ type	5 !== '5'	true
&&	and	(5 < 8) && (8 < 5)	true
	or	(5 != '5') (5 !== '5')	true
!	not	!(true)	false

IF ... ELSE

if (เงื่อนไข)

{ // คำสั่งกรณีตรงกับเงื่อนไข }

else

{ // คำสั่งกรณีที่ตรงกันข้ามกับเงื่อนไข }

ตัวอย่าง

```
var x = 5
if( x >= 8) {
    console.log("condition is true");
}
else{
    console.log("condition is false");
}
```

condition is false.

NESTED IF & ELSE IF

```
var x = 5
if( 5 <= 8) {
  console.log("condition1 is true");
}
if( 5 < 8)
{
  console.log("condition2 is true");
}
else if(5 <= 8){
  console.log("condition3 is true");
}
else{
  console.log("condition is false");
}
```

condition1 is true.
condition2 is true.

```
var x = 5
if(false) {
  console.log("condition1 is true");
}
if(false)
{
  console.log("condition2 is true");
}
else if(5 <= 8){
  console.log("condition3 is true");
}
else{
  console.log("condition is false");
}
```

condition3 is true.

SWITCH CASE

ตัวอย่าง

```
switch ("สิ่งที่ต้องการตรวจสอบ") {  
    case 1:  
        //คำสั่งที่ 1  
        break;  
    case 2:  
        //คำสั่งที่ 1  
        break;  
    default:  
        //กรณีที่ตรงกับคำสั่งใดๆเลย  
}
```

```
var i = 200;  
switch (i) {  
    case 100:  
        score = "100";  
        break;  
    case 200:  
        score = "200";  
        break;  
    case 300:  
        score = "300";  
        break;  
    default:  
        score = "ไม่ตรงกับค่าไหนเลย";  
}
```

คะแนนของคุณ 200 เต็ม

EXERCISE

หาค่า max min โดยกำหนด ตัวแปร $x = 10$ และ $y = 20$

แล้ว แสดงออกมาบนหน้าเว็บ

“ <ชื่อตัวแปรที่มีค่า max> is max number.”

“ <ชื่อตัวแปรที่มีค่า min> is min number.”

ถ้าหากเท่ากัน ให้แสดงเป็น “number is equal”

test case

$x = 10$ $y = 20$ \Rightarrow y is max number.

x is min number.

$x = 30$ $y = 30$ \Rightarrow number is equal

ต่ำกว่า 22 ปี ห้ามเข้า โดยกำหนดให้ name = ชื่อ year = ปี พ.ศ. เกิด

** ไม่ต้องคำนวณเดือน

หากอายุ 22 ปีขึ้นไป ให้แสดง

“Welcome <ชื่อ> to JavaScript Course”

หากอายุต่ำกว่า 22 ปี หรือ 22 ปี พอดี ให้แสดง

“Please wait for team to carry you. Welcome <ชื่อ> to JavaScript Course”

UNIT 4

FOR WHILE LOOP

FOR LOOP

for (ตัวแปร=ค่าเริ่มต้น;เงื่อนไข;การเปลี่ยนค่าตัวแปร)
{
// เมื่อเงื่อนไขเป็นจริงจะทำตามคำสั่งนี้;
}

ตัวอย่าง

```
for (let i = 0; i <= 10; i++) {  
    console.log(i);  
}
```

0
1
2
3
4
5
6
7
8
9
10

WHILE LOOP

while (เงื่อนไข)

```
{
// เมื่อเงื่อนไขเป็นจริงจะทำตามคำสั่งนี้;
}
```

*** ต้องทำให้เงื่อนไขเป็นเท็จ จึงจะจบ loop while

ตัวอย่าง

```
let i = 0
while (i <= 10) {
  console.log(i)
  i++
}
```

0
1
2
3
4
5
6
7
8
9
10

```
do {
// เมื่อเงื่อนไขเป็นจริงจะทำตามคำสั่งนี้;
} while (เงื่อนไข)
```

*** เปรียบเสมือน while แต่จะทำงานโดยไม่สนเงื่อนไขก่อนครั้งแรก แล้วจึงจะเช็คเงื่อนไข

```
let i = 0
do {
  console.log(i)
  i++
}
while(i <= 10)
```

0
1
2
3
4
5
6
7
8
9
10

NESTED LOOP

ตัวอย่าง มาวาดรูปกันเถอะ

```

      X
      X
  X   X   X   X   X
      X
      X

```

EXERCISE

Factorial คือการคำนวณผลคูณรูปแบบหนึ่งโดยจะทำการคูณตั้งแต่ 1 ไปถึงจำนวนนับที่ต้องการ

เช่น

5! จะมีค่าเท่ากับ $5 \times 4 \times 3 \times 2 \times 1$ ผลลัพธ์คือ 120

3! จะมีค่าเท่ากับ $3 \times 2 \times 1$ เท่ากับ 6

10! จะมีค่าเท่ากับ $10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$ ได้ 3628800

UNIT 5

FUNCTIONS

FUNCTIONS คืออะไร

ถ้าหากให้อธิบายง่ายๆ

Functions คือ ชุดคำสั่งที่ใช้ในการทำงานอย่างใดอย่างหนึ่ง ซึ่งจะทำงานเมื่อถูกเรียกใช้งานเท่านั้น

*** syntax เดิม

```
function ชื่อฟังก์ชัน (อาร์กิวเมนต์) {  
    /// code...  
}
```

```
function name(params) {  
  
}
```

สามารถกำหนดค่าเริ่มต้นให้ อาร์กิวเมนต์ได้ เช่น

```
function abFunction( a = 10 , b = 'some thing' ) {  
}
```

*** syntax Arrow function

<var,let,const> ชื่อฟังก์ชัน = (อาร์กิวเมนต์) => {
 /// code...
}

```
const name = (params) => {  
  /// code...  
}
```

หากเป็นคำสั่ง บรรทัดเดียวสามารถ ย่อ ได้เหลือแค่

<var,let,const> ชื่อฟังก์ชัน = (อาร์กิวเมนต์) => // คำสั่ง

```
const name = (params = 10) => console.log(params);
```

FUNCTION ตัวอย่าง และ การใช้งาน

```
function plus ( a, b )  
{  
    return a + b  
}
```

```
console.log(plus( 1,2))    // 3  
console.log(plus( '1', '2')) // 12
```

```
function helloName ( name )  
{  
    return "Hello " + name  
}
```

```
console.log(helloName("Admin"))    // Hello Admin
```

```
let plus = ( a, b ) =>  
{  
    return a + b  
}
```

```
console.log(plus( 1,2))    // 3  
console.log(plus( '1', '2')) // 12
```

```
let helloName = ( name ) => `Hello ${name}`
```

```
console.log(helloName("Admin"))    // Hello Admin
```

```
let noneParameter = () => `Hello`
```

```
console.log(helloName())    // Hello
```


EXERCISE

สร้าง function สูตรคูณ

กำหนดให้ $n =$ แม่สูตรคูณ

แสดง

$n \times 1 = \dots$

.

.

$n \times 12 = \dots$

สร้าง function แสดงลำดับแบบขั้นบันได

กำหนดให้ $\text{row} = 5, \text{orderBy} = \text{'asc'}$

แสดง

case $\text{orderBy} = \text{'asc'}$

1

12

123

1234

12345

case $\text{orderBy} = \text{'desc'}$

54321

4321

321

21

1

UNIT 6

IMPORT & EXPORT

IMPORT & EXPORT

ปกติแล้วเวลาเราเขียนโค้ด JavaScript ก็จะมีรวมไว้ในไฟล์เดียว และนำไปใช้งาน แต่รู้หรือไม่ว่าเราสามารถทำการเปลี่ยนให้ไฟล์ JS ของเรากลายร่างเป็น module หรือก็คือชิ้นส่วนที่จะนำไปใช้ให้ไฟล์ JS ไฟล์อื่นเรียก หรือจะเอาไปแชร์ให้คนอื่นนำไปใช้ก็ยังได้
แล้วจริงๆ JavaScript module มันคืออะไร และทำงานยังไง ลองไปดูกัน

JavaScript module ที่ใช้กันส่วนใหญ่นั้นคือ ES module ซึ่งออกมาให้ได้ใช้กันตั้งแต่ปี 2015 โดยมีการเรียกใช้งานแบบนี้

```
import moduleName from 'path_of_module';  
// หรือ  
const moduleName = require('path_of_module');
```

โดยในไฟล์ที่เราต้องการจะทำให้เป็น module เพื่อให้ไฟล์อื่นเรียกใช้เราจะต้องทำการ export

1. Export ด้วยชื่อ (Named Export)

```
export const pi = 3.1416;  
export const hello = (name) => {  
  console.log(`Hello ${name}`);  
};  
export function sum(x, y) {  
  return x + y;  
}
```

สร้างไฟล์ lib.js

Import เฉพาะชื่อตัวแปรหรือฟังก์ชันที่ต้องการ

```
import { pi, hello, sum } from "./lib";  
console.log(pi); // 3.1416  
hello("es6"); // Hello es6  
console.log(sum(1, 2)); // 3
```

ทดสอบเรียกใช้ที่ index.html ภายใต้

```
<script type="module">  
/// โค้ด  
</script>
```

Import ทั้งหมด

```
import * as lib from "./lib";  
console.log(lib.pi); // 3.1416  
lib.hello("es6"); // Hello es6  
console.log(lib.sum(1, 2)); // 3
```

EXPORT CONST VS EXPORT DEFAULT

export default สามารถมีได้แค่ตัวเดียว ต่อ 1 ไฟล์

syntax

export default <ตัวแปร หรือ ฟังก์ชัน>

หรือใช้แบบ export หลายๆ ตัวแปร

```
export default {
```

```
...
}
```

export const 1 ไฟล์สามารถใช้ export const ได้กี่อันก็ได้แต่ ชื่อห้ามซ้ำ

syntax

export const <ตัวแปร หรือ ฟังก์ชัน>

หรือใช้แบบ export หลายๆ ตัว

```
export const <ตัวแปร หรือ ฟังก์ชัน>
```

```
....
```

```
export const <ตัวแปร หรือ ฟังก์ชัน>
```

การใช้งานตอน import

```
import <ตัวแปรชื่ออะไรก็ได้> from 'path_file'
```

ตัวแปรที่รับจะมีค่าเหมือนกับ export default มาทุกประการ

การใช้งานตอน import

```
import { <ตัวแปรชื่อเดียวกับตัวที่แปรที่ export > } from 'path_file'
```

การใช้งาน **IMPORT *** และการ **RENAME**

การเปลี่ยนชื่อจะใช้ **as** ในการเปลี่ยนชื่อเป็นชื่อตัวแปรที่เราต้องการ

syntax

```
import * as <ตั้งชื่อตัวแปรใหม่> from "module-name";
```

หรือ

```
import { export1 as alias1 } from "module-name";
```

การใช้ **import ***

syntax

```
import * as <ตั้งชื่อตัวแปรที่ต้องการรับ> from "module-name";
```

ตัวแปรที่รับจะได้ค่าทั้งหมดที่ **export** มาของไฟล์นั้น

ศึกษาเพิ่มเติมได้ที่ <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import>

UNIT 7

STRING & ARRAYS

STRING

มารู้จัก **string** เพิ่มเติมกันอีกนิด โดยปกติแล้วการใช้งาน **string** ได้โดยใช้ `'` (single quote) `"` (double quote)

ถ้าหากต้องการเขียน **single quote** ใน **string** ต้องทำอะไร ?

เราจะใช้ **double quote** ในการประกาศ **string**

```
let my_string = " I'm a robot. "
```

ถ้าหากต้องการเขียน **double quote** ใน **string** ต้องทำอะไร ?

เราจะใช้ **single quote** ในการประกาศ **string**

```
let my_string = ' Call me "robot". '
```

การใช้งานตัวแปร กับ **string** จะให้งานโดยการต่อ **string**
ตัวอย่าง เช่น

```
let name = 'robot'
```

```
let my_string = " I'm a " + name
```

แล้วถ้าต้องการเขียนทั้ง **double quote** และ **single quote** ต้องทำอะไร ?

es6 จะมี ``` (backtick) มาแก้ปัญหาเรื่อง **string** ให้ใช้งานง่ายขึ้น เรียกว่า **template string** โดยสามารถ ทำให้ **string** ทำ **multiline** ได้ และสามารถจัดการตัวแปรได้ง่ายขึ้น โดยใช้ `${ตัวแปร}`

```
let name = "robot"
```

```
let my_string = `I'm a robot .
```

```
  Please call me "${name} ".`
```

STRING METHODS

string ก็เป็น array ชนิดหนึ่งสามารถเข้าถึงแบบ array ได้ แต่จะไม่สามารถแก้ไขค่าโดยใช้ index ได้

```
let my_string = "Hello world"  
my_string[0] = 'S'  
console.log(my_string[0]) // H  
console.log(my_string[1]) // e
```

length

return ความยาวของ string (จะนับ space ด้วย)

เช่น

```
let my_string = "Hello world"  
console.log(my_string.length) // 11
```

trim()

return ความยาวของ string (จะนับ space ด้วย)

เช่น

```
let my_string = "   Hello World!   "  
console.log(my_string.trim()) // Hello World!  
console.log(my_string.length) // 27  
console.log(my_string.trim().length) // 12
```


STRING METHODS

split()

return array โดยแบ่ง index ตามสิ่งที่ต้องการ split
เช่น

```
let txt = "a,b,c,d,e"  
console.log(txt.split(",")); // ["a", "b", "c", "d", "e"]
```

หรือ

```
let txt = "a b c d e"  
console.log(txt.split(" ")); // ["a", "b", "c", "d", "e"]
```

search()

return index แรกที่พบ หากไม่พบที่ search จะ return -1
เช่น

```
let txt = " Hello World "  
console.log(txt.search("a")) // -1
```

toUpperCase() and toLowerCase()

return แปลง string เป็นตัว พิมพ์ใหญ่/พิมพ์เล็ก ทั้งหมด
เช่น

```
let my_string = "HelloWorld!"  
console.log(my_string.toUpperCase()); // HELLO WORLD!  
console.log(my_string.toLowerCase()); // hello world!
```

replace()

return แทนที่คำ หรือ ตัวอักษร เป็นตัวกำหนด
เช่น

ต้องการแทนที่ S ด้วยคำว่า

```
let my_string = 'Hello s World'  
console.log(my_string.replace("s", ""));
```

หากต้องการแทนที่ทั้งหมด ต้องใช้ RegEx (/คำหรือตัวอักษร/g)
เช่น

```
let my_string = 'Hello s World ss ss'  
console.log(my_string.replace( /s/g, ""));
```

ARRAYS

JavaScript จะมอง array เป็น type object ซึ่งความแตกต่างของ array ที่ต่างจากภาษาอื่นคือ ชนิดของสิ่งที่เก็บใน array ไม่จำเป็นต้องเป็นชนิดเดียวกันก็ได้

การประกาศ array สามารถทำได้โดย

```
let my_array = []  
let first_array = new Array()  
let array_number = [1,2,3,4,5]  
let multi_array = ["Tom" , 1 , 2 , "Jerry" , true ]  
let multi_dimension = [ 1 ,[ 1 ,2 ] , [1,2,3] , ["one", ["two","three"]] ]
```

การเข้าถึงค่าใน array

*** syntax array[ตำแหน่งที่ต้องการเข้าถึง] โดยเริ่มจาก 0 ถึง (length-1)

```
console.log(my_array[0]);           // undefined   เนื่องจาก my_array ไม่มีค่าใดๆ  
console.log(array_number [0]);      // 1  
console.log(array_number [1]);      // 2  
console.log(multi_array [3]);       // "Jerry"  
console.log(multi_dimension[3])     // ["one", ["two","three"]] ]  
console.log(multi_dimension[3][1])  // ["two","three"]  
console.log(multi_dimension[3][1][0]) // "two"
```

ARRAYS (ต่อ)

การกำหนดค่าใน array

*** syntax array[ตำแหน่งที่ต้องการเข้าถึง] = ค่าที่ต้องการ

เช่น

```
let my_array = [];
my_array[0] = 1
console.log(my_array); // [ 1 ]
my_array[1] = my_array[0] + 1
console.log(my_array); // [ 1, 2 ]
```

ตัวอย่าง เช่น

```
let languages = ["c","c++","java"]
languages.push("javaScript")
console.log(languages); // ["c", "c++", "java", "javaScript"]
languages.shift()
console.log(languages); // ["c++", "java", "javaScript"]
languages.unshift("JS ES9")
console.log(languages); // ["JS ES9", "c++", "java", "javaScript"]
languages.pop()
console.log(languages); // ["JS ES9", "c++", "java"]
```

หรือ

ใช้ method ของ array ในการจัดการ array

Method	การใช้งาน
push	เพิ่มค่าไปต่อ last index ของ array
pop	ลบค่าตำแหน่งสุดท้าย ของ array
shift	ลบค่า index แรก ของ array
unshift	เพิ่มค่าไปแทรกใน index แรกของ array

LOOP ARRAY

ใช้ loop for ปกติ

ตัวอย่าง

```
let languages = ["c", "c++", "java", "javaScript"]
for (let i = 0; i < languages.length; i++) {
  console.log(languages[i]);
}
```

c
c++
java
javaScript

ใช้ loop for ... of

ตัวอย่าง

```
let languages = ["c", "c++", "java", "javaScript"]
for (const language of languages) {
  console.log(language);
}
```

c
c++
java
javaScript



EXERCISE

กำหนด `text = ["hello", "world", "i am", "zebra"]`

ให้ตัดสระในภาษาอังกฤษของแต่ละตัวออก สระในภาษาอังกฤษ (a,e,i,o,u)

ผลลัพธ์

`["hll", "wrld", " m", "zbr"]`

`replace()`

`return` แทนที่คำ หรือ ตัวอักษร เป็นตัวกำหนด

เช่น

ต้องการแทนที่ **s** ด้วยคำว่า

หากต้องการแทนที่ทั้งหมด ต้องใช้ **Regex** (/คำหรือตัวอักษร/g)

เช่น

```
let my_string = 'Hello s World ss ss'
console.log(my_string.replace( /s/g, "" ));
```

ARRAY METHODS ES6

Method	การใช้งาน
map	สร้าง array ใหม่ด้วยผลลัพธ์จากการ return ของแต่ละ element ของ function
filter	สร้าง array ใหม่ด้วย element เดิมที่ผ่านเงื่อนไข
reduce	return ออกมาค่าเดียว ด้วยการ รวม array
some	return true จากการผ่านเงื่อนไขเพียงกรณีเดียว return false หากไม่ผ่านเงื่อนไขเลย
every	return true เมื่อผ่านเงื่อนไขทุก element return false หากไม่ผ่านเงื่อนไขเพียงกรณีเดียว

ศึกษาเพิ่มเติมได้ที่ https://www.w3schools.com/jsref/jsref_obj_array.asp

MAP

***syntax

```
Array.map( <callbackFunction> (value , index) {
  <return element ของ new Array >
})
```

ตัวอย่างการใช้งาน

```
let languages = ["c","c++","java","javaScript"]
let array = languages.map( function(value,index){
  console.log(value , " => value in languages");
  return value
})

console.log(array);
```

c => value in languages

c++ => value in languages

java => value in languages

javaScript => value in languages

► (4) ["c", "c++", "java", "javaScript"]

***syntax

```
Array.map( <callbackFunction> : (value , index) => {
  <return element ของ new Array >
})
```

ตัวอย่างการใช้งาน (การจัดการค่าใน array ใหม่)

```
let languages = ["c","c++","java","javaScript"]
let newArray = languages.map((value,index) => {
  return `index: ${index} | value : ${value}`
})

console.log(newArray);
```

```
(4) ["index: 0 | value : c", "index: 1 |
0: "index: 0 | value : c"
1: "index: 1 | value : c++"
2: "index: 2 | value : java"
3: "index: 3 | value : javaScript"
length: 4]
```

FILTER

***syntax

```
Array.filter( <callbackFunction> (value , index) {  
  <return element หาก เงื่อนไขเป็นจริง>  
})
```

ตัวอย่างการใช้งาน (หาเลขคู่ใน array)

```
let list_number = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
let newArray = list_number.filter( (value, index) => {  
  | return value % 2 == 0  
});  
console.log(newArray);
```

► (4) [2, 4, 6, 8]

***syntax

```
Array.filter( <callbackFunction> : (value , index) => {  
  <return element หาก เงื่อนไขเป็นจริง >  
})
```

ตัวอย่างการใช้งาน (search ชื่อใน array)

```
let list_number = ["Juieta", "Annice", "Orsa", "Hewett", "Eulalie"]  
let newArray = list_number.filter( (value, index) => {  
  | return value.search("et") !== -1  
});  
console.log(newArray);
```

► (2) ["Juieta", "Hewett"]

REDUCE

***syntax

```
Array.reduce( <callBackFunction> (current , next,index)  
{  
  <return logic operation>  
})
```

***syntax

```
Array.reduce( <callBackFunction> : (current , next, index) =>  
{  
  <return logic operation>  
})
```

ตัวอย่างการใช้งาน (หา **sum** ของ **array**)

```
let list_number = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];  
let sumArray = list_number.reduce((curr,next,index) => {  
  return curr+next;  
}));  
console.log(sumArray);
```

```
// 55
```

SOME

***syntax

```
Array.some( <callbackFunction> (value,index) {  
  <return เงื่อนไข>  
})
```

***syntax

```
Array.some( <callbackFunction> : (value, index) => {  
  <return เงื่อนไข >  
})
```

ตัวอย่างการใช้งาน (เช็คสินค้าหมดหรือไม่)

```
let stock_of_product = [10, 50, 0, 20, 0];  
let out_of_stock = stock_of_product.some((value, index) => {  
  return value === 0  
});  
console.log(out_of_stock);
```

true

EVERY

***syntax

```
Array.every( <callBackFunction> (value,index) {  
  <return เงื่อนไข>  
})
```

***syntax

```
Array.every( <callBackFunction> : (value, index) => {  
  <return เงื่อนไข >  
})
```

ตัวอย่างการใช้งาน (เช็คสินค้าคงเหลือทั้งหมดหรือไม่)

```
let stock_of_product = [10, 50, 1, 20, 0];  
let out_of_stock = stock_of_product.every((value, index) => {  
  return value !== 0  
});  
console.log(out_of_stock);
```

false

.map ใช้สำหรับ จัด array ใหม่

.filter ใช้สำหรับ search ข้อมูลที่สนใจ

.reduce ใช้สำหรับ กระทำกับ array แล้ว return ค่าออกมา 1 ค่า

.some และ .every ใช้สำหรับ การเช็คค่าใน array

EXERCISE

สร้าง function สำหรับเข้ารหัส array string โดยให้ เลื่อนตัวอักษรไป 1 อักษร และ
ค่าใน array มี เว้นวรรคได้ และ เป็น string [a -z] เท่านั้น

*** z --> a

กำหนด plain_text = ["hello", "world", "I am", "zebra"]

Output ["ifmmp", "xpsme", "j bn", "afcsb"]

*** hint

charCodeAt() // แปลงเป็น ascii code

String.fromCharCode() // แปลง ascii code กลับมาเป็น string

UNIT 8

OBJECT

Object คืออะไร

ถึงเราจะเรียกมันว่า **OBJECT** แต่ถ้าเอาจริงๆ น่าจะเรียกมันว่า **COLLECTION** หรือ **MAP** ซะมากกว่า โดย **OBJECT** ในภาษานี้จะมีการเก็บข้อมูลในรูปแบบของ **KEY-VALUE** โดย **KEY** จะอยู่ในรูปของ **STRING** เท่านั้น และ **VALUE** สามารถเป็นอะไรก็ได้ เช่น **STRING , NUMBER, FUNCTION , OBJECT** เป็นต้น **VALUE** ของ **OBJECT** นี้เราจะเรียกว่า **PROPERTY**

*** syntax

<var,let,const> ชื่อ object = new Object()

หรือ

<var,let,const> ชื่อ object = { }

*** การ assign value ของ object

ชื่อ object [“ ชื่อ property ”] = value

หรือ

ชื่อ object . ชื่อ property = value

*** หากใช้แบบนี้ ชื่อ property ต้องไม่มี space

*** การเข้าถึง property ของ object

```
var person = {};
  person["firstname"] = "demo person";
  person.surname = "last person";
  person["full name"] = person.firstname+" "+person.surname
  person.getFullname = () => {
    return person['full name']
  }
  console.log(person);
  console.log(person.getFullname());
```

Object

*** การ ASSIGN VALUE ของ OBJECT

```
var person = {};  
  person["firstname"] = "demo person";  
  person.surname = "last person";  
  person["full name"] = person.firstname+" "+person.surname  
  person.getFullname = () => {  
    return person['full name']  
  }  
  console.log(person);  
  console.log(person.getFullname());
```

```
▼ {firstname: "demo person", surname: "last person", f  
  firstname: "demo person"  
  full name: "demo person last person"  
  ► getFullname: () => { return person['full name'] }  
  surname: "last person"  
  ► __proto__: Object
```

```
demo person last person
```


OBJECT DE-STRUCTURING

```
let student = {  
  rollno:20,  
  name:'Prijin',  
  cgpa:7.2  
}
```

เราสามารถทำ **destructuring** โดยประกาศตัวแปรด้วย **property** ของ **object** นั้นได้เลยเช่น

```
let {name,cgpa} = student  
  console.log(name)  
  console.log(cgpa)
```

เราสามารถทำ **destructuring** และเปลี่ยนชื่อตัวแปรได้ในคราวเดียวกัน ตัวอย่างเช่น

```
let {name:student_name,cgpa:student_cgpa}=student  
  console.log(student_cgpa)  
  console.log("student_name",student_name)
```

*** ข้อควรระวังของ OBJECT

```
var val1 = {name: "Tom"};  
var val2 = {name: "Tom"};  
console.log(val1 == val2) // return false  
console.log(val1 === val2) // return false
```

หากเป็นแบบนี้จะเป็นการใช้ VAL2 ตัวเดียวกับ VAL1

```
var val1 = {name: "Tom"};  
var val2 = val1  
console.log(val1 == val2) // return true  
console.log(val1 === val2) // return true  
  
val2.name = "Tom2"  
console.log(val1, val2); // {name: "Tom2"} {name: "Tom2"};
```

แก้ปัญหากการใช้ **VAL2** ตัวให้มีค่าเริ่มต้นเดียวกับ **VAL1**

โดยใช้วิธีการ **copy object** ของ **val1** มาโดย

Spread Operator

Syntax

การเรียกใช้เพื่อรับพารามิเตอร์ใน function

```
myFunction(...iterableObj);
```

การ copy object

```
let objClone = { ...obj };
```

การใช้ copy array

```
let arrayClone = [...arr];
```

```
var val1 = {name: "Tom"};
var val2 = {...val1}
console.log(val1 == val2) // return true
console.log(val1 === val2) // return true

val2.name = "Tom2"
console.log(val1, val2); // { name: 'Tom' } { name: 'Tom2' }
```

ศึกษาต่อได้ที่ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

การใช้งานตัวแปร กับ Object

การใช้งานตัวแปรมาเป็น value ของ Object

```
let age = 20
let firstName = 'Natcharin'
let lastName = 'Janhom'
let obj = {
  name : firstName + " " + lastName ,
  age : age
}
```

```
/// { name: 'Natcharin Janhom', age: 20 }
```

การใช้งานตัวแปรมาเป็น value ของ Object

การใช้งานตัวแปรมาเป็น value และใช้ชื่อ key เดียวกับตัวแปร ของ Object สามารถใช้ตัวแปรแทนลงไปได้เลย

```
let age = 20
let firstName = 'Natcharin'
let lastName = 'Janhom'
let obj = {
  firstName,
  lastName,
  age
}
```

```
/// { firstName: 'Natcharin', lastName: 'Janhom', age: 20 }
```

การใช้งานตัวแปรใช้เป็นชื่อ key ของ Object

ใช้ [] ในการตั้ง scope สำหรับ string หรือ ตัวแปร ที่จะใช้เป็นชื่อ key ของ object

```
let keyName = 'firstName'  
let keyLastName = 'lastName'  
let obj = {  
  [keyName] : 'Natcharin',  
  [keyLastName] : 'Janhom',  
  ['age'] : 20  
}
```

```
/// { firstName: 'Natcharin', lastName: 'Janhom', age: 20 }
```

OBJECT METHODS

Object.keys(<object ที่ต้องการ get key>)

return array ของ property names

เช่น

```
let obj = {  
  id: '62530',  
  name: 'Natcharin',  
}  
console.log(Object.keys(obj)) // ['id', 'name']
```

Object.values(<object ที่ต้องการ get value>)

return array ของ property values

เช่น

```
let obj = {  
  id: '62530',  
  name: 'Natcharin',  
}  
console.log(Object.values(obj)) // ['62530', 'Natcharin']
```

ARRAY OBJECT

การประยุกต์ใช้งาน **array** เพื่อใช้เก็บ **object** หรือไม่ว่าจะเป็นการใช้ **object** เก็บ **array** โดยการใช้งานจะไม่ซับซ้อนหรือเข้าใจยากอย่างที่จินตนาการ การเข้าถึงค่าจะเป็นไปตามพื้นฐาน

ตัวอย่างเช่น

object ที่เก็บค่า **array** (บุคคล 1 มีรถยนต์หลายคัน)

```
let person = {  
  name: "Jothanan",  
  age: 28,  
  car: ["Tsubaru", "Honda"],  
};
```

array ของ **object** (หมู่บ้าน 1 มี คนอยู่หลายคน)

```
let vilage = [  
  {  
    name: "Jothanan",  
    age: 28,  
    car: ["Tsubaru", "Honda"],  
  },  
  {  
    name: "Wonstin",  
    age: 50,  
    car: ["BMW"],  
  }  
]
```


ARRAY OBJECT

```
let employee = {  
  id: '000000',  
  name: 'FirstEmployee',  
}  
let employee2 = {  
  id: '62530',  
  name: 'Natcharin',  
}  
let team = [ {...employee} , {...employee2}]  
for (let i = 0; i < team.length; i++) {  
  console.log(team[i]);  
  console.log(team[i].name);  
}
```

```
// { id: '000000', name: 'FirstEmployee' }  
// FirstEmployee  
// { id: '62530', name: 'Natcharin' }  
// Natcharin
```

EXERCISE

ทำการ import ค่า salesRecord จากไฟล์ salesRecord.js
แล้วทำการหา sumA จาก array ของ product ที่มี name == 'a'
และ sumB จาก array ของ product ที่มี name == 'b'
ของแต่ละ day

เช่น

```
salesRecord = [ {  
  day: 1, product:[  
    { name: 'a', value: 5 }, { name: 'b', value: 2 },  
    { name: 'a', value: 7 }, { name: 'b', value: 4 } ]  
}]
```

ผลลัพธ์ที่ได้คือ

```
[  
  {  
    day: 1,  
    product:[  
      { name: 'a', value: 5 }, { name: 'b', value: 2 },  
      { name: 'a', value: 7 }, { name: 'b', value: 4 }  
    ],  
    sumA: 12 , sumB: 6  
  }  
]
```

EXERCISE

จากไฟล์ `data_people.js` ให้เรียกข้อมูลมาใช้งานกรองข้อมูลให้ได้ผลลัพธ์ดังนี้

1. แสดง จำนวน เพศ ชาย ทั้งหมด ก็ คน และมีเพศ หญิง ทั้งหมดก็คน
2. มีใครบ้าง ที่อยู่ประเทศ จีน (China)
3. มีใครบ้างที่มี II อยู่ในชื่อ
4. แสดงผลลัพธ์ของ จำนวนของผู้คนทั้งหมด จำนวนของเพศหญิง และ จำนวนของเพศชาย ของแต่ละประเทศ

UNIT 9

ASYNCHRONOUS

จากปัญหา **Callback Function** นั้นจะต้องนำฟังก์ชันมาซ้อนกันเข้าไปข้างในไปเรื่อยๆ หากการทำงานซับซ้อนมากขึ้น หลายขั้นตอนมากขึ้น จะทำให้ดูและเขียนยากขึ้นไปอีก จึงกำเนิด **Promise** ขึ้นมา

การทำงานของ **promise** นั้นง่ายมาก เราแค่ใช้ **.then** หลังจากการเรียกฟังก์ชัน **doFirst** แล้วใส่ฟังก์ชัน **doSecond** ลงไป แบบนี้

```
function doFirst(callback){
    callback();
}
function doSecond(callback){
    callback();
}

doFirst(function(){
    doSecond(function(){
        console.log('success');
    });
});
```

```
doFirst()
    .then(doSecond)
    .then(function () {
        console.log("success");
    });
```

```
function doFirst() {
    return new Promise(function (resolve, reject) {
        if (error) reject(error);
        else resolve(respond);
    });
}

function doSecond() {
    return new Promise(function (resolve, reject) {
        if (error) reject(error);
        else resolve(respond);
    });
}
```

โดยที่ฟังก์ชันของเราจะต้อง **return** **promise** โดยที่ **promise** จะใส่ **callback** พารามิเตอร์ **resolve** หากทำงานปกติ, **reject** เป็นเคสที่ต้องการให้เกิด **error**

โดยที่ฟังก์ชันของเราจะต้อง **return promise** โดยที่ **promise** จะใส่ **callback** พารามิเตอร์ **resolve** หากทำงานปกติ, **reject** เป็นเคสที่ต้องการให้เกิด **error**

```
function doFirst() {  
  return new Promise(function (resolve, reject) {  
    if (error) reject(error);  
    else resolve(respond);  
  });  
}  
  
function doSecond() {  
  return new Promise(function (resolve, reject) {  
    if (error) reject(error);  
    else resolve(respond);  
  });  
}
```

การทำงานของ **promise** นั้นง่ายมาก เราแค่ใช้ **.then** หลังจากการเรียกฟังก์ชัน **doFirst** แล้วใส่ฟังก์ชัน **doSecond** ลงไป แบบนี้

```
doFirst()  
  .then(doSecond)  
  .then(function () {  
    console.log("success");  
  });
```

แต่ promise นั้นก็ยังมีข้อเสีย callback hell เช่น ถ้าหากเราต้องใช้งาน .then สัก 10 ครั้งละ แค่ Promise อย่างเดียวถึงยังไม่พอ จนต้องเกิดเป็น Async / Await ขึ้นมา

```
function taskOne() {  
  setTimeout(function () {  
    console.log("this is task 1");  
  }, 500);  
}  
function taskTwo() {  
  console.log("this is task 2");  
}  
function taskThree() {  
  setTimeout(function() {  
    console.log("this is task 3");  
  }, 1000)  
}
```

```
taskOne().then(function () {  
  taskTwo().then(function () {  
    taskThree().then(function () {  
      taskFour().then(function () {  
        taskFive().then(function () {  
          taskSix().then(function () {  
            taskSeven().then(function () {  
              taskEight().then(function () {  
                taskNine().then(function () {  
                  taskTen();  
                })  
              })  
            })  
          })  
        })  
      })  
    })  
  })  
});
```

Await / Async นี่มันใช้งานยังไง

await ใช้เพื่อบอกให้ JavaScript รอจนกว่าคำสั่งนั้นจะเสร็จ ถึงค่อยไปทำงานอันต่อไป โดยฟังก์ชันที่จะมี **await** อยู่ข้างในได้ต้องประกาศเป็น **async** เสมอ

```
async function main() {  
  await taskOne();  
  await taskTwo();  
  await taskThree();  
  await taskFour();  
  await taskFive();  
  await taskSix();  
  await taskSeven();  
  await taskEight();  
  await taskNine();  
  await taskTen();  
}  
main();
```

เท่านี้โค้ดของเราก็สวยงามอ่านง่ายแล้ว Easy สุดๆ

UNIT 10

HTTP REQUEST

Http Request

ประกอบด้วย

URL : ที่อยู่ของ Server ที่จะไปขอข้อมูล

HTTP Headers : ข้อมูลเพิ่มเติมที่แนบไปให้ Server

Cookies : ข้อมูลขนาดเล็กที่ทาง Server สามารถฝากไว้ที่ Client ได้

Request Body : ข้อมูลที่ระบุว่าคำสั่ง request นั้นๆ มีรายละเอียดอะไรบ้าง ใช้ใน (POST / PUT เป็นต้น)

Method : ข้อมูลที่บอกว่า Client ต้องการทำอะไรกับข้อมูล

Http Request Method

Method	Description
GET	เป็น method สำหรับร้องขอข้อมูลจาก resource
POST	เป็น method สำหรับสร้างข้อมูลใหม่ใน resource ซึ่งจะส่งข้อมูลใน body
PUT	เป็น method สำหรับอัปเดตข้อมูลของข้อมูลที่มีอยู่แล้วใน resource
DELETE	เป็น method สำหรับลบข้อมูลที่มีอยู่ใน resource

Http Response

ประกอบด้วย

HTTP Status Code : บอกสถานะว่าการ **Request** ครั้งนั้นได้รับการตอบรับเป็นอย่างไร

HTTP Response Headers : ข้อมูลเพิ่มเติมที่แนบมาโดย **Server**

Response Body : ข้อมูลจริงๆที่ **Server** นั้นตอบกลับมาที่ **Client**

Http Response Code

Code	Status code
2xx (Success)	200 Ok
	201 Create ข้อมูลใหม่ได้ถูกสร้าง
	204 No Content ดาเนินการ Success แต่ไม่ได้ return ข้อมูลกลับ
3xx (Redirection)	304 Not Modified (client ได้รับการ response แล้วอยู่ใน cache)
4xx (Client error)	400 Bad Request (request ที่ส่งมาไม่ถูกดำเนินการ และ Server ไม่เข้าใจ request)
	401 Unauthorized (client ไม่ได้รับอนุญาตในการเข้าถึง Resource)
	403 Forbidden (Client ไม่ได้รับการอนุญาตให้เข้าถึง Resource ด้วยเหตุผลบางประการ)
	404 Not Found (resource ที่ request มา ไม่ว่างใช้งานตอนนี้ หรือ ไม่พบ resource)
	405 Gone (resource ที่ต้องการนั้นไม่มีอยู่แล้ว)
5xx (Server error)	500 Internal Server Error (request ถูกต้องแต่server ชัดข้อง)
	503 Service Unavailable (server ใช้งานไม่ได้ หรือไม่ว่าง)

UNIT 11

RESTFUL API

JSON

คือมาตรฐานหนึ่งที่ได้รับคามนิยม ใช้สำหรับ การรับ/ส่งข้อมูล ติดต่อกันระหว่างโปรแกรมหรือระบบต่างๆ

ภายใน JSON ประกอบด้วยอะไรบ้าง ?

JSON เป็นข้อมูลรูปแบบ text ที่มีรูปแบบที่จะเก็บข้อมูลแบบ key, value โดยการเขียนข้อมูลชนิด JSON มีรูปแบบคือ ชื่อฟิลด์ครอบด้วยเครื่องหมาย “ (double quote), เครื่องหมาย : (colon), value แล้วครอบทั้งหมดด้วยเครื่องหมายปีกกา ตัวอย่างที่มีข้อมูล 1 อย่างจะเป็นดังนี้

```
{"key": "value"}
```

ประเภทข้อมูลที่ JSON เก็บได้มีดังนี้

- string
- number
- object (JSON object)
- array
- boolean
- null

ตัวอย่าง JSON จาก <https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/intro-to-tweet-json>

```
{
  "created_at": "Thu Apr 06 15:24:15 +0000 2017",
  "id_str": "850006245121695744",
  "text": "1\ Today we\u2019re sharing our vision for the future of the Twitter API platform!\nhttps://t.co/XweGngmxIP",
  "user": {
    "id": 2244994945,
    "name": "Twitter Dev",
    "screen_name": "TwitterDev",
    "location": "Internet",
    "url": "https://dev.twitter.com/",
    "description": "Your official source for Twitter Platform news, updates & events. Need technical help? Visit https://twittercommunity.com/\u2328\u2013 #TapIntoTwitter"
  },
  "place": {
  },
  "entities": {
    "hashtags": [
    ],
    "urls": [
      {
        "url": "https://t.co/XweGngmxIP",
        "unwound": {
          "url": "https://cards.twitter.com/cards/18ce53wgo4h/3xo1c",
          "title": "Building the Future of the Twitter API Platform"
        }
      }
    ]
  }
}
```

API (Application Programming Interface) เป็นส่วน interface ที่เชื่อม
แอปพลิเคชันกับภายนอกและใช้โดย Developer เพื่อใช้ข้อมูลและเข้าถึงข้อมูล

คำศัพท์ที่สำคัญที่เกี่ยวกับ REST API

Resource เป็น object หรือเป็นตัวแทนของบางสิ่งบางอย่าง ซึ่งมีการเชื่อมโยงกับข้อมูล ซึ่งสามารถ
set เป็น method ที่จะกระทำอย่างใดอย่างหนึ่งกับข้อมูล เช่น Animals, Schools และ Employees
เป็น resource และ GET, POST, PUT และ DELET เป็น Method หรือวิธีการที่จะกระทำกับ
Resource

API Endpoint

URL ย่อมาจาก **Universal Resource Locator** เป็น path หรือเส้นทางไปยังที่อยู่ของ resource และบาง action ที่จะกระทำกับ resource

API Endpoint ที่ดีควรมีการจัดการ API Versioning

เช่น
 https://endpoint/v1/ <resource>
 https://endpoint/v2/ <resource>

URL ควรเป็นคำนาม (nouns) เท่านั้น เช่น employee, company, sale เป็นต้น **ไม่ควรมีคำกริยา** (verbs) ซึ่งได้แก่ addNew, delete, insert เป็นต้น ถ้าหากเป็น API ของ Employee ควรกำหนด API Endpoint ดังนี้

Method	Template	API Endpoint	การทำงาน
GET	/<nouns>	/employees	ส่งกลับ list Employees ทั้งหมด
GET	/<nouns>/<params>	/employees/1	ส่งกลับรายละเอียดของ Employees 1
POST	/<nouns>	/employees	insert Employee ใหม่
PUT	/<nouns>/<params>	/employees/1	แก้ไข Employees 1
DELETE	/<nouns>/<params>	/employees/1	ลบ Employees 1

UNIT 12

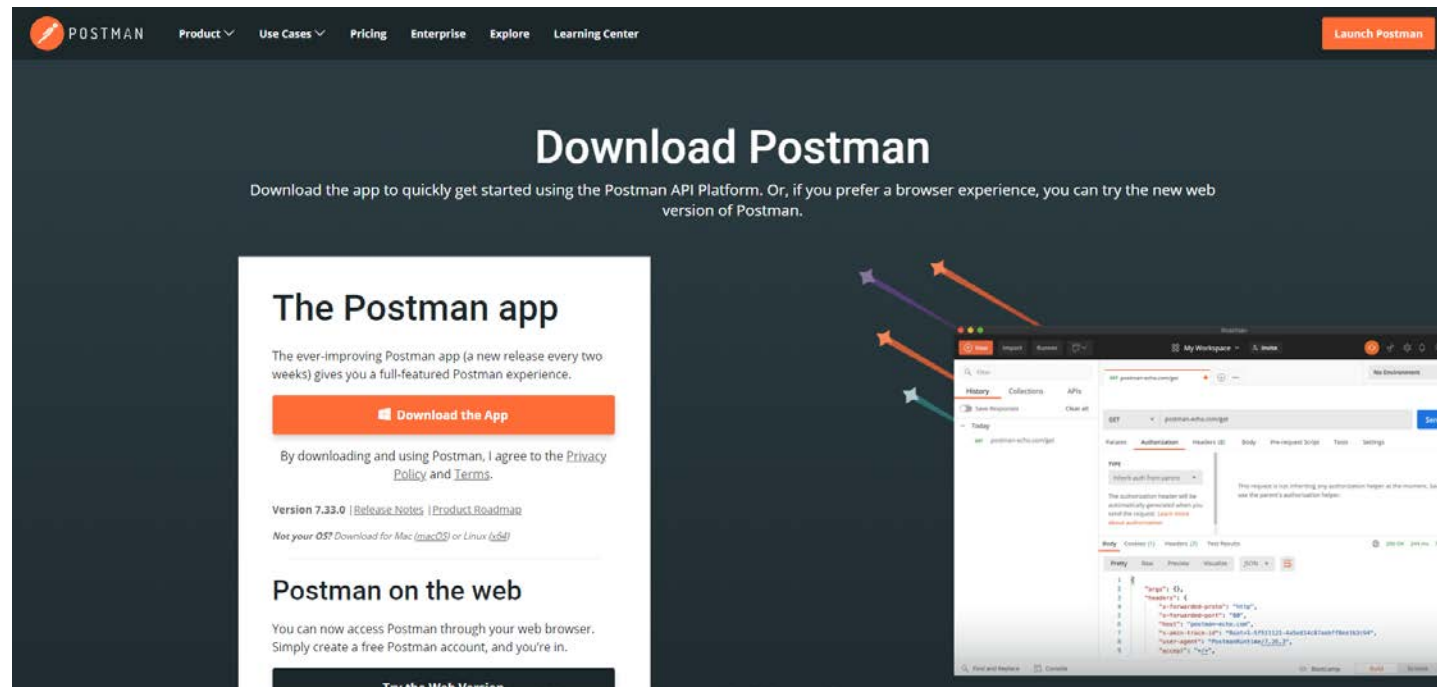
NODE.JS EXPRESS

DOWNLOAD POSTMAN

[HTTPS://WWW.POSTMAN.COM/DOWNLOADS/](https://www.postman.com/downloads/)

หรือใช้ Extension ของ google chrome

[HTTPS://CHROME.GOOGLE.COM/WEBSTORE/DETAIL/POSTMAN/FHBJGBIFLINJBDGGEHCDDCBNCDDDOMOP?HL=TH](https://chrome.google.com/webstore/detail/postman/fhbjgbfijnjbdggehcddcbncdddomop?hl=th)



NodeJS คืออะไร

เป็น JavaScript runtime ที่ทำให้ JavaScript ที่ปกติเขียนอยู่บนเว็บ ในส่วนของ frontend ให้สามารถใช้งานในส่วนของ backend หรือ server ได้

Backend Framework

Express

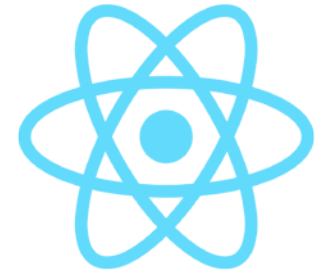


METEOR

Front-end Framework



Vanilla JS



npm (Package Manger for JavaScript)

npm หรือ node package manager จะถูกติดตั้งมาเมื่อเราทำการติดตั้ง nodejs ในเครื่องของเรา ถ้าแปลตามความหมายเลย คือ ตัวจัดการ node package นั้นเอง

วิธีการติดตั้งโมดูลด้วย npm

ติดตั้งแบบ **local**

อยู่ในโฟลเดอร์ **node_modules** ในโฟลเดอร์ปัจจุบัน

`npm install <module_name>` หรือแบบย่อ `npm i <module_name>`

ติดตั้งแบบ **Global**

`npm install -g <module_name>` หรือแบบย่อ `npm i -g <module_name>`

การ **install node_modules** จาก ไฟล์ **package.json** ที่มีอยู่แล้ว

`npm i`

Create Express Project

-ติดตั้ง **express-generator**

`npm i -g express-generator`

-สร้างโปรเจค **express**

***** `express -v` [ชื่อ template engine] [ชื่อโฟลเดอร์ app] โปรเจ็ค]**

-เปิด command line folder root ที่ต้องการสร้างโปรเจค
โดยใช้คำสั่งนี้

`express -v ejs expressAPI`

-เข้าไปเปิดไฟล์เดอริโปรเจคขึ้นมา

```
TON@DESKTOP-4V04MVN MINGW64 /d
$ express -v ejs expressAPI

create : expressAPI\
create : expressAPI\public\
create : expressAPI\public\javascripts\
create : expressAPI\public\images\
create : expressAPI\public\stylesheets\
create : expressAPI\public\stylesheets\style.css
create : expressAPI\routes\
create : expressAPI\routes\index.js
create : expressAPI\routes\users.js
create : expressAPI\views\
create : expressAPI\views\error.ejs
create : expressAPI\views\index.ejs
create : expressAPI\app.js
create : expressAPI\package.json
create : expressAPI\bin\
create : expressAPI\bin\www

change directory:
$ cd expressAPI

install dependencies:
$ npm install

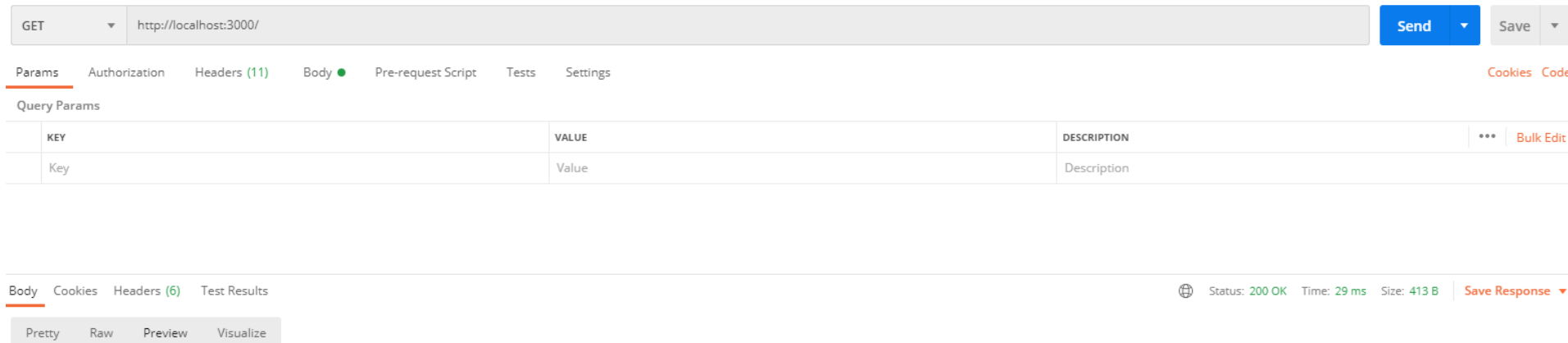
run the app:
$ DEBUG=expressapi:* npm start

TON@DESKTOP-4V04MVN MINGW64 /d
$ |
```

-ติดตั้ง node_modules
npm i

-ลองรันโปรเจค
npm start

-ทดสอบ api โดยใช้ Postman ส่ง Get ไปที่ url <http://localhost:3000/>



The screenshot shows the Postman interface for a GET request to `http://localhost:3000/`. The request is successful, returning a 200 OK status with a response time of 29 ms and a size of 413 B. The response body is displayed in the 'Body' tab, showing the text 'Welcome to Express'.

GET `http://localhost:3000/` Send Save

Params Authorization Headers (11) Body ● Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize

Status: 200 OK Time: 29 ms Size: 413 B Save Response

Express

Welcome to Express

โครงสร้างของ project

```

> bin
> node_modules
✓ public
  > images
  > javascripts
  > stylesheets
  ✓ routes
    JS index.js
    JS users.js
  ✓ views
    <% error.ejs
    <% index.ejs
    JS app.js
    package-lock.json
    package.json
  
```

/bin ใช้สำหรับเก็บ config express ที่ใช้สำหรับการ run เป็น server

/node_modules เป็นโฟลเดอร์ใช้เก็บไฟล์ package ของ library ต่างๆของโปรเจค

/public เป็นโฟลเดอร์ใช้สำหรับเก็บ ไฟล์ต่างๆที่ใช้ในโปรเจค เช่น รูปที่อัปโหลดมาเก็บไว้ ไฟล์ utils ที่เขียนไว้เป็น function กลางที่ใช้ในโปรเจคทั้งหมด

/routes เป็นโฟลเดอร์ที่ใช้เก็บ resource ที่ใช้ติดต่อจัดการข้อมูลต่างๆ

/views เป็นโฟลเดอร์ที่เก็บไฟล์ ejs ใช้สำหรับการ render html ด้วย js

app.js เป็นไฟล์ที่ใช้สำหรับ custom engine ของ server ของ api เรา

package.json เป็นไฟล์ JSON ที่บ่งบอกข้อมูลต่างๆของ project ที่ใช้ node package โดยทำหน้าที่เป็น meta data และคอยบอกว่า project นั้นทำงานอย่างไร

Install cors เพื่อให้ host อื่นสามารถใช้งาน service ของเราได้

npm i cors

เมื่อเสร็จแล้วมาที่ไฟล์ app.js

ทำการเพิ่มโค้ดดังนี้

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();
var cors = require('cors');
app.use(cors());
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

install nodemon ใช้สำหรับการรันอัปเดตโค้ดใหม่ เมื่อมีการ **save** โค้ดในโปรเจค
โดยไม่ต้อง รันโปรเจคใหม่ทุกครั้งที่เราแก้ไขโค้ด

npm i nodemon

เวลาต้องการรันโปรเจคจะใช้คำสั่ง **nodemon**

*** หากไม่สามารถรันได้ แก้ไขได้โดย

ไปที่ไฟล์ **package.json** แก้ไข **script** ดังนี้

```
"scripts": {  
  | "start": "node ./bin/www"  
  },  
}
```



```
"scripts": {  
  | "start": "nodemon ./bin/www"  
  },  
}
```

แล้วรันโปรเจคด้วย

npm start

การทำงานของ api จะทำงานด้วย Routing

Routing คืออะไร

Routing คือการกำหนด URL หรือ Path ที่ใช้ในการเรียกข้อมูลผ่าน HTTP request เช่น GET POST PUT หรือ DELETE แล้วให้ทำการ **response** หรือส่งกลับข้อมูลหรือการทำงานใดๆ ให้สอดคล้องกับ URL ใดๆ ที่ระบุเข้ามา การ **response** หรือส่งกลับข้อมูลด้วยการทำคำสั่งใดๆ นั้น เราสามารถกำหนดได้ว่า จะให้ทำคำสั่งเดียวหรือหลายคำสั่ง เมื่อผู้ใช้ **request** เข้ามายัง **path** และเข้าเงื่อนไขการทำงาน

Syntax การประกาศใช้งาน route

```
router.METHOD(PATH, HANDLER)
```

ตัวอย่างที่ express สร้างมาให้ที่ไฟล์ **/routes/user.js**

```
app.use("/", indexRouter);
app.use("/users", usersRouter);

var express = require("express");
var router = express.Router();

/* GET users listing. */
router.get("/", function (req, res, next) {
  res.send("method get");
});
```

โดย

router คือ instance ของ express (มองว่าเป็นตัวแทนของ express ก็ได้)

METHOD คือ HTTP request method ใช้เป็นตัวเล็ก หมายถึง ทำการเรียก ข้อมูลผ่าน HTTP ด้วยวิธี (get | post | put | delete)

PATH คือ URL หรือ ส่วนของโดเมนนับตั้งแต่ root path เป็นต้นไปในตัวอย่างจะได้ว่า เริ่ม root ที่ **/users** แล้วเรียกใช้งาน root ของ

userRouter จะได้ path URL ดังนี้ **:endpoint/users/**

หรือ **:endpoint/users**

HANDLER คือฟังก์ชันที่กำหนดให้ทำงาน เมื่อ PATH ตรงกับค่าที่กำหนด

สร้าง API สำหรับ CRUD สินค้า

โดยเริ่มไปที่ สร้าง ไฟล์ products.js ใน folder routes

และภายในสร้าง api method get post put delete ไว้สำหรับการทำ CRUD ต่อไป

```
var express = require("express");
var router = express.Router();

/* GET users listing. */
router.get("/", function (req, res, next) {
  res.send("method get");
});

router.post("/", function (req, res, next) {
  res.send("method post");
});

router.put("/", function (req, res, next) {
  res.send("method put");
});

router.delete("/", function (req, res, next) {
  res.send("method delete");
});

module.exports = router;
```

การรับค่าจาก Request ที่เข้ามา

headers

ใช้รับข้อมูลที่ส่งผ่านมาทาง Header

นิยมใช้ในการส่ง authentication เช่น token

body

ใช้รับข้อมูลที่ส่งผ่านมาทาง Body

สามารถรับข้อมูลได้หลายรูปแบบ เช่น form-data , JSON, binary, html , text

นิยมใช้คือ JSON และ form-data

params

ใช้รับข้อมูลที่ส่งผ่านมาทาง URL

นิยมใช้เมื่อต้องการส่งพารามิเตอร์ 1 ตัวเพิ่มมาเพื่อระบุสิ่งที่ต้องการ

เช่น

ต้องการดึงข้อมูล product ที่มี id = 1

/products/1

query

ใช้รับข้อมูลที่ส่งผ่านมาทาง URL

นิยมใช้เมื่อต้องการส่งพารามิเตอร์มาเป็น option (จะส่งหรือไม่ส่งก็ได้)

เช่น

ต้องการดึงข้อมูล product ที่อยู่หน้าที่ 10 และ จำนวนต่อหน้า 10 row

/products?page=10&limit=10

การส่ง Response ต่อการ Request

`send()`

มาสามารถตอบกลับเป็นอะไรก็ได้ หรือสามารถกำหนด status code ไปด้วยก็ได้โดยใช้ status ตัวอย่างเช่น

```
res.send(new Buffer('wahoo'));  
res.send({ some: 'json' });  
res.send('<p>some html</p>');  
res.status(404).send('Sorry, cant find that');
```

การ response ที่นิยมควรมีรายละเอียดดังนี้

```
{  
  data: ใช้ส่ง resource ตอบกลับเมื่อมีการ request ขอข้อมูล,  
  message: ใช้ส่ง text อธิบายเพิ่มเติม เช่น “create success”,  
  error: ใช้ในการส่ง error text จาก Bad request ที่เข้ามา ตัวอย่างเช่น [ ‘username is duplicate’ , ‘password is require’ ],  
  success: ใช้ในการส่งสถานะ true,false ของการทำงานว่าสำเร็จหรือไม่  
}
```


ทำการเรียกใช้งาน routes ของ products
ไปที่ file app.js เรียกใช้งานดังนี้

```
var indexRouter = require("./routes/index");
var usersRouter = require("./routes/users");
const productsRouter = require("./routes/products");
var app = express();
var cors = require("cors");
app.use(cors());
// view engine setup
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");

app.use(logger("dev"));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, "public")));

app.use("/", indexRouter);
app.use("/users", usersRouter);
app.use("/products", productsRouter)
```

เรียกใช้งาน router products

ประกาศ path ที่ใช้สำหรับการทำงานกับ router

UNIT 13

MONGO DB & MONGOOSE



เป็นฐานข้อมูลแบบ NoSQL คือไม่มี relation

โดยถ้าเปรียบเทียบกับ MySQL จะเป็นดังนี้

MySQL	MongoDB
-------	---------

Table	Collection
-------	------------

Row	Document
-----	----------

Column	Field
--------	-------

Document ซึ่งจะเก็บค่าเป็น key และ value จะเห็นว่ามันก็คือรูปแบบ JSON นั่นเอง
ตัวอย่างเช่น

```
{
  "_id": ObjectId("554b8ee746e04bc5503aef47"),
  "name": "Chai"
}
```

*** ใน MongoDB ข้อมูล document ที่เก็บใน collection จะมีคีย์ _id ทำหน้าที่เปรียบเสมือน primary key อยู่ด้วย

Schemaless

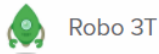
คือ การไม่ต้องกำหนดโครงสร้างใดๆให้มันเหมือน SQL ปกติทั่วไป เช่น collection User มีเก็บแค่ name ต่อมาเราสามารถเพิ่มการเก็บ position เข้ามาได้เลย แบบนี้

```
{  
  "name": "Chai"  
}
```

```
{  
  "name": "Chai" ,  
  "position": "Developer"  
}
```

DOWNLOAD ROBO 3T

[HTTPS://ROBOMONGO.ORG/DOWNLOAD](https://robomongo.org/download)

[Download](#)[Blog](#)[Account](#)

Simplicity Meets Power

Two products. One download. Double the MongoDB GUI power.

[Download your Double Pack](#)

Studio 3T: Professional IDE for MongoDB

Download Studio 3T, the professional GUI and IDE for MongoDB preferred by over 100,000 developers and DBAs. Build queries fast, generate instant code, import/export in multiple formats, and much more. Available for Windows, macOS, and Linux..

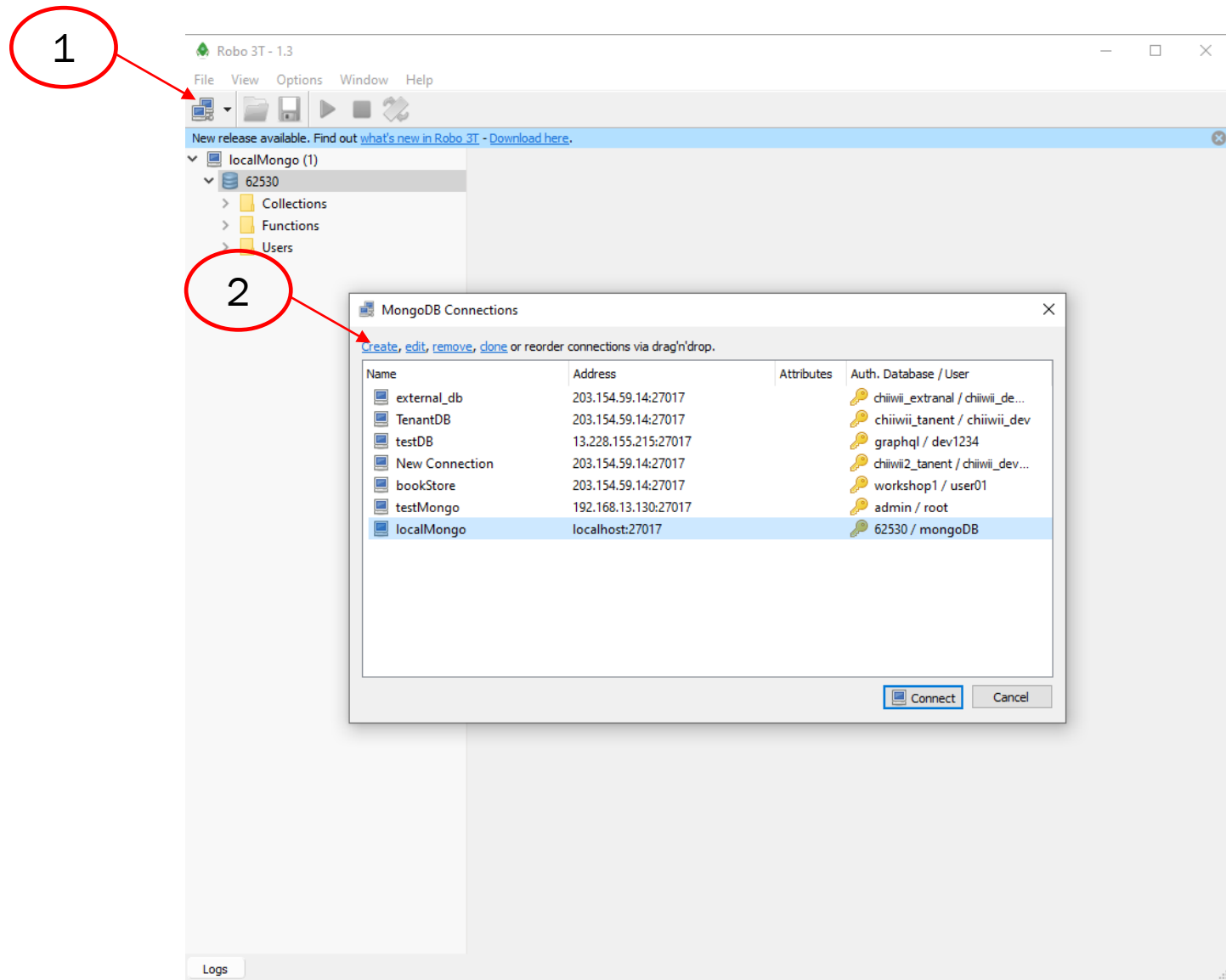
[Download Studio 3T](#)

Robo 3T: Simple GUI for beginners

Robo 3T 1.4 brings support for MongoDB 4.2, a mongo shell upgrade from 4.0 to 4.2, the ability to manually specify visible databases, and many other fixes and improvements. [View the full blog post.](#)

[Download Robo 3T](#)

การใช้งาน ROBO 3T เบื้องต้น



Connection

Connection Settings

Connection Authentication SSH SSL Advanced

Type: Direct Connection

Name: New Connection

Address: localhost : 27017

Specify host and port of MongoDB server. Host can be either IPv4, IPv6 or domain name.

From SRV Import connection details from MongoDB SRV connection string

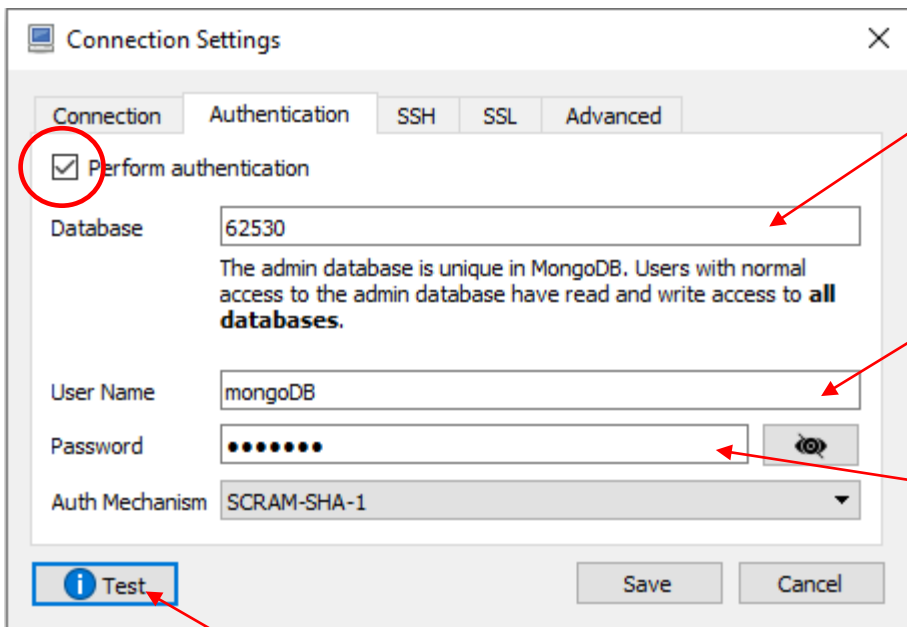
Test Save Cancel

ชื่อ label connection

Port ของ Database

IP Host ของ Database
หรือ
Domain name ก็ได้

Authen Connection



The image shows a 'Connection Settings' dialog box with the 'Authentication' tab selected. The 'Perform authentication' checkbox is checked and circled in red. The 'Database' field contains '62530'. Below it, a note states: 'The admin database is unique in MongoDB. Users with normal access to the admin database have read and write access to **all** databases.' The 'User Name' field contains 'mongoDB'. The 'Password' field is masked with dots and has a red eye icon to its right. The 'Auth Mechanism' dropdown is set to 'SCRAM-SHA-1'. At the bottom, there are 'Test', 'Save', and 'Cancel' buttons. The 'Test' button is highlighted with a blue border and a red arrow points to it from the bottom text.

ชื่อ database ที่ต้องการเชื่อมต่อ

Username สำหรับ connect Database

Password สำหรับ connect Database

เมื่อกรอกรายละเอียดเรียบร้อยแล้ว กด Test เพื่อทดสอบ connection

ตัวอย่าง collections ข้อมูล

Robo 3T - 1.3

File View Options Window Help

New release available. Find out [what's new in Robo 3T](#) - [Download here](#).

localMongo (1)

62530

Collections (1)

users

Functions

Users

db.getCollection('users').find({})

localMongo localhost:27017 62530

db.getCollection('users').find({})

users 0.001 sec. 0 50

Key	Value	Type
(1) ObjectId("5f73a7a8e38bdf6ce...")	{ 3 fields }	Object
_id	ObjectId("5f73a7a8e38bdf6ce965d4dc")	ObjectId
first_name	Natcharin	String
last_name	Janhom	String
(2) ObjectId("5f73a7c1e38bdf6ce...")	{ 3 fields }	Object
_id	ObjectId("5f73a7c1e38bdf6ce965d4f0")	ObjectId
first_name	Test	String
last_name	Test	String

Logs

การเรียกใช้งาน CONFIG จาก ไฟล์ .ENV

สร้างไฟล์ .env เพื่อใช้สำหรับการเก็บ config ค่าต่างๆใน Project

Config
สำหรับ connecting Database

```
.env
1 DB_HOST=localhost
2 DB_PORT=27017
3 DB_NAME=62530
4 DB_USER=mongoDB
5 DB_PASS=mongoDB
```

Install dotenv

npm i dotenv

สำหรับการเรียกใช้งานไฟล์ dotenv

```
var createError = require("http-errors");
var express = require("express");
var path = require("path");
var cookieParser = require("cookie-parser");
var logger = require("morgan");
require('dotenv').config();
var indexRouter = require("./routes/index");
var usersRouter = require("./routes/users");
const productsRouter = require("./routes/products");
```

เรียกใช้งานไว้ที่ app.js จะสามารถเรียกใช้ ตัวแปรจากไฟล์ env ได้ทั้ง Project

การเรียกใช้งานตัวแปรในไฟล์ env

ต้องใช้ผ่าน process.env

เช่น

ไฟล์ .env

```
DB_HOST=localhost  
DB_PORT=27017  
DB_NAME=62530  
DB_USER=mongoDB  
DB_PASS=mongoDB
```

ตัวอย่างการใช้งาน

```
console.log(process.env.DB_HOST)  
// localhost
```

Install mongoose

```
npm i mongoose
```

Mongoose เป็น ODM (Object Document Mapping) ตัวหนึ่งของ Mongodb จริงๆ ก็ไม่ใช่อะไรใหม่ หน้าตาของมันก็เหมือนกับ ORM ฝั่ง DBMS เลยครับแต่เป็นของ NoSQL และที่เป็น “Object Document Mapping” เพราะว่า MongoDB เป็น NoSQL ที่เก็บข้อมูลแบบ Document Store ซึ่งเป็นรูปแบบหนึ่งของ NoSQL

การทำงานของ Object mapping คือ การสร้าง object ความสัมพันธ์, โครงสร้างของข้อมูลใน database เวลาเรียกใช้ก็เรียกเหมือน การ access class หรือ object ลดการเขียนคำสั่ง SQL หรือคำสั่งสำหรับ query ลง

การใช้งานก็เหมือนกัน ORM คือกำหนด schema, กำหนด model และเรียกใช้งานผ่าน model ซึ่งตอนนี้เราก็ไม่ต้องสนใจแล้วว่าฐานข้อมูลข้างหลังมีโครงสร้างเป็นยังไง

MONGOOSE CONNECTING

Connecting Database ด้วย mongoose

```
JS app.js ×
JS app.js > ...

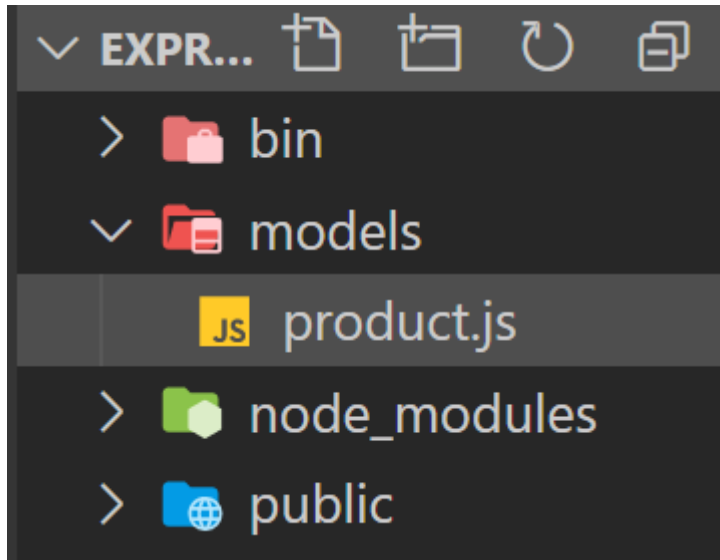
9  const productsRouter = require("./routes/products");
10 var app = express();
11 var cors = require("cors");
12 const mongoose = require('mongoose')
13 const { DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASS } = process.env
14 mongoose.connect(`mongodb://${DB_HOST}:${DB_PORT}/${DB_NAME}`, {
15   user: DB_USER,
16   pass: DB_PASS,
17   useUnifiedTopology: true,
18   useNewUrlParser: true
19 }).then(() => {
20   console.log('DB connect!!')
21 }).catch(err => {
22   console.log('DB connect fail !!')
23 })
24
25 app.use(cors());
```

เพิ่มโค้ด connect DB
ในไฟล์ app.js

สร้าง Model ที่ใช้สำหรับการ Mapping กับ Document Database

สร้างไฟล์ product.js

โดย product ต้องการเก็บข้อมูลดังต่อไปนี้



```
const mongoose = require("mongoose");
const products = new mongoose.Schema({
  product_name: { type: String },
  price: { type: Number },
  amount: { type: Number },
});
module.exports = mongoose.model("products", products);
```

Create Product

```
router.post("/", async function (req, res, next) {  
  try {  
    const { product_name, price, amount } = req.body;  
    let newProduct = new productModel({  
      product_name: product_name,  
      price: price,  
      amount: amount,  
    });  
    let product = await newProduct.save();  
    return res.status(201).send({  
      data: product,  
      message: "create success",  
      success: true,  
    });  
  } catch (error) {  
    return res.status(500).send({  
      message: "create fail",  
      success: false,  
    });  
  }  
});
```

ตัวอย่างการใช้งาน



POST http://localhost:3000/products Send Save

Params Authorization Headers (11) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

```
1 {
2   "product_name": "แวนดา",
3   "price": 1000,
4   "amount": 10
5 }
```

Body Cookies Headers (7) Test Results Status: 201 Created Time: 68 ms Size: 404 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "data": {
3     "_id": "5f73c3162db4d93780afe0d2",
4     "product_name": "แวนดา",
5     "price": 1000,
6     "amount": 10,
7     "__v": 0
8   },
9   "message": "create success",
10  "success": true
11 }
```

POST http://localhost:3000/products Send Save

Params Authorization Headers (11) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

```
1 {  
2   "product_name": "แก้วตา",  
3   "price": "1,000",  
4   "amount": 10  
5 }
```

Body Cookies Headers (7) Test Results 🌐 Status: 500 Internal Server Error Time: 24 ms Size: 304 B Save Response

Pretty Raw Preview Visualize JSON 🔍

```
1 {  
2   "message": "create fail",  
3   "success": false  
4 }
```

Get All Product

```
router.get("/", async function (req, res, next) {  
  try {  
    let products = await productModel.find();  
    return res.status(200).send({  
      data: products,  
      message: "success",  
      success: true,  
    });  
  } catch (error) {  
    return res.status(500).send({  
      message: "server error",  
      success: false,  
    });  
  }  
});
```

Get Product By Id

```
router.get("/:id", async function (req, res, next) {  
  try {  
    let id = req.params.id;  
    if (!mongoose.Types.ObjectId.isValid(id)) {  
      return res.status(400).send({  
        message: "id Invalid",  
        success: false,  
        error: ["id is not a ObjectId"],  
      });  
    }  
    let products = await productModel.findById(id);  
    return res.status(200).send({  
      data: products,  
      message: "success",  
      success: true,  
    });  
  } catch (error) {  
    return res.status(500).send({  
      message: "server error",  
      success: false,  
    });  
  }  
});
```

Update Product

```
router.put("/:id", async function (req, res, next) {
  try {
    let id = req.params.id;
    if (!mongoose.Types.ObjectId.isValid(id)) {
      return res.status(400).send({
        message: "id Invalid",
        success: false,
        error: ["id is not a ObjectId"],
      });
    }
    await productModel.updateOne(
      { _id: mongoose.Types.ObjectId(id) },
      { $set: req.body }
    );
    let product = await productModel.findById(id);
    return res.status(201).send({
      data: product,
      message: "update success",
      success: true,
    });
  } catch (error) {
    console.log(error.message);
    return res.status(500).send({
```


Delete Product

```
router.delete("/:id", async function (req, res, next) {
  try {
    let id = req.params.id;
    if (!mongoose.Types.ObjectId.isValid(id)) {
      return res.status(400).send({
        message: "id Invalid",
        success: false,
        error: ["id is not a ObjectId"],
      });
    }
    await productModel.deleteOne({ _id: mongoose.Types.ObjectId(id) });
    let products = await productModel.find();
    return res.status(200).send({
      data: products,
      message: "delete success",
      success: true,
    });
  } catch (error) {
    return res.status(500).send({
      message: "delete fail",
      success: false,
    });
  }
});
```




EXERCISE

สร้างระบบสั่งซื้อสินค้าด้วย schema ที่กำหนดให้

1. Api create,update,delete,getAll,getById ของ product
2. Api create,update,delete,getAll,getById ของ user โดย password ต้องเข้ารหัสด้วย bcrpyt
3. Api create , get order ทั้งหมด , get order By Id และ get order By ชื่อของ user

โดย

create ต้องเช็คด้วยว่า order product ที่เข้ามา มีจำนวนการสั่งซื้อ มากกว่า stock ที่มีหรือไม่ ถ้าเกินกว่าที่มีใน stock ให้ return {

success : false,

message: 'product out of stock'

error: เป็น array ของรายชื่อ product ที่ไม่สามารถซื้อได้

}

4. api สำหรับ login

-input คือ username และ password

หาก login ผ่าน return data เป็น object ของ user

หาก login ไม่ผ่าน return {

success : false,

message: 'unauthorization'

}

UNIT 14

UPLOAD FILE

Upload file ด้วย multer

```
npm i multer
```

```
const multer = require("multer");
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "./public/images");
  },
  filename: function (req, file, cb) {
    cb(null, new Date().getTime() + "_" + file.originalname);
  },
});
const upload = multer({ storage: storage });
```

กำหนดปลายทางไฟล์เดอร์ที่ต้องการบันทึกไฟล์

กำหนด ชื่อไฟล์

การนำไปใช้งาน

```
router.post("/", upload.single("image"), async function (req, res, next) {
```

แทรก `upload.single(<ชื่อตัวแปรที่กำหนดให้ส่งไฟล์>)`

ตัว `upload` ของ `multer` มีหลาย method เช่น

`.single` อัปโหลดไฟล์ 1 ไฟล์

`.array` อัปโหลดหลายไฟล์ สามารถกำหนดจำนวนไฟล์ได้ เช่น `.array('image',2)` รับไฟล์จากตัวแปร `image` ได้มากที่สุด 2 ไฟล์

`.none` ไม่อัปโหลดไฟล์ใดๆ

`.fields` อัปโหลดไฟล์หลายๆไฟล์ โดย ผ่านตัวแปรต่างกัน เช่น `.fields([{ name: 'avatar', maxCount: 1 }, { name: 'gallery', maxCount: 8 }])`

ศึกษาต่อได้ที่ <https://www.npmjs.com/package/multer>

ตัวอย่างการอัปโหลดไฟล์รูปภาพ และ เก็บชื่อไฟล์ลง data base

config ปลายทางไฟล์เดอร์และ ชื่อไฟล์ดังนี้

```
const multer = require("multer");
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    | cb(null, "./public/images");
  },
  filename: function (req, file, cb) {
    | cb(null, new Date().getTime() + "_" + file.originalname);
  },
});

const upload = multer({ storage: storage });
```

```

router.post("/", upload.single("image"), async function (req, res, next) {
  try {
    let nameImage = "rambo.jpg";
    if (req.file) {
      nameImage = req.file.filename;
    }
    const { name, price, amount } = req.body;
    let newProduct = new productModel({
      name: name,
      price: price,
      amount: amount,
      img: nameImage,
    });
    let product = await newProduct.save();
    res.send({
      data: product,
      message: "create success",
      success: true,
    });
  } catch (error) {
    res.status(500).send({
      message: "create fail",
      success: false,
    });
  }
});

```

ตัวอย่างการอัปโหลดไฟล์รูปภาพ
และ เก็บชื่อไฟล์ลง data base

UNIT 15

BCRYPT HASH PASSWORD

การเก็บ Password แบบไหน? ถึงจะปลอดภัย

การเก็บ password ลงฐานข้อมูล (Database) เราจะได้เก็บกันตรงๆ จะต้องนำ password มาทำกระบวนการเข้ารหัสบางอย่างก่อน
ในที่นี้เราเลือกใช้วิธีการ hashing โดยใช้ Bcrypt

เช่น

“1234” หลังจาก hash แล้วจะได้ “a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9”
จะเห็นว่า hash function จะแปลงพาสเวิร์ด “1234” เป็นข้อความลับอะไรซักอย่างที่อ่านไม่ออก ทั้งนี้ขนาดข้อความที่ได้จาก hash function จะคงที่ (fixed size)


```
npm i bcrypt
```

วิธีการเลือก install version

Syntax

```
npm i <libs>@<version>
```

Ex.

ในกรณีนี้ใช้ node version 12 ต้อง install version $\geq 3.0.6$ จึงเลือกใช้ version ล่าสุดโดย

```
npm i bcrypt ได้เลย
```

Version Compatibility

Node Version	Bcrypt Version
0.4	≤ 0.4
0.6, 0.8, 0.10	≥ 0.5
0.11	≥ 0.8
4	$\leq 2.1.0$
8	$\geq 1.0.3 < 4.0.0$
10, 11	≥ 3
12	$\geq 3.0.6$

การใช้งานจะแบ่งเป็น 2 ประเภท คือ **เข้ารหัส(hash)** และ **การเปรียบเทียบ (compare)**

```
bcrypt.hash(<password> , <saltRound>)
```

password: TEXT ที่ต้องการเข้ารหัส

saltRound: string (เป็น salt ที่ generate จาก bcrypt)

หรือ

saltRound: number (จำนวนครั้งในการเข้ารหัส)

*** ในการเข้ารหัส text เดียวกันอาจจะได้ hash ไม่เหมือนกัน

ตัวอย่างเช่น

```
let hashPassword = await bcrypt.hash("1234", 10);
```

```
// "$2b$10$FYZ/hnRONRqj.w5GYjKWouZ6wFjd9arO1FywVOctzKlkiDRQZwP2y"
```

```
bcrypt.compare(<password> , <passwordHashing>)
```

password : text ที่ต้องการเปรียบเทียบ

passwordHashing : text ที่ hash เรียบร้อยแล้ว

โดยผลลัพธ์จะเป็น true, false

EX.

```
let hashPassword = await bcrypt.hash("1234",10);
```

```
let result = await bcrypt.compare("12345",hashPassword)  
// false
```

```
let result = await bcrypt.compare("1234",hashPassword)  
  
// true
```

ระบบสมาชิก

สร้าง userModel โดยสร้างไฟล์ user.js ในโฟลเดอร์ model

```
const mongoose = require("mongoose");
const users = new mongoose.Schema({
  username: { type: String, unique:true },
  password: { type: String },
  firstName: { type: String },
  lastName:{ type: String },
  email:{ type: String }
});
module.exports = mongoose.model("users", users);
```

สร้าง userRouter โดยสร้างไฟล์ users.js ในโฟลเดอร์ routes

```
var express = require("express");
var router = express.Router();

/* GET users listing. */
router.get("/", function (req, res, next) {
  res.send("method get user");
});

module.exports = router;
```

```
var createError = require("http-errors");
var express = require("express");
var path = require("path");
var cookieParser = require("cookie-parser");
var logger = require("morgan");
require("dotenv").config();
var indexRouter = require("./routes/index");
var usersRouter = require("./routes/users");
const productsRouter = require("./routes/products");
const uploadRouter = require("./routes/upload");
var app = express();
```

เรียกใช้งาน route โดยเพิ่มส่วนนี้ที่ไฟล์ app.js

```
app.use("/", indexRouter);
app.use("/users", usersRouter);
app.use("/products", productsRouter);
app.use("/files", uploadRouter);
// catch 404 and forward to error handler
app.use(function (req, res, next) {
  next(createError(404));
});
```

Create User

```
router.post("/", async function (req, res, next) {  
  try {  
    let { password, username, firstName, lastName, email } = req.body;  
    let hashPassword = await bcrypt.hash(password, 10);  
    const newUser = new Users({  
      username,  
      password: hashPassword,  
      firstName,  
      lastName,  
      email,  
    });  
    const user = await newUser.save();  
    return res.status(200).send({  
      data: { _id: user._id, username, firstName, lastName, email },  
      message: "create success",  
      success: true,  
    });  
  } catch (error) {  
    return res.status(500).send({  
      message: "create fail",  
      success: false,  
    });  
  }  
});
```

```
var express = require("express");  
var router = express.Router();  
const bcrypt = require("bcrypt");  
const Users = require("../models/user");  
/* GET users listing */
```

POST http://localhost:3000/users Send

Params Auth Headers (8) **Body** Pre-req. Tests Settings

raw JSON

```
1 {  
2   "username": "user",  
3   "password": "1234",  
4   "firstName": "somchai",  
5   "lastName": "somsong",  
6   "email": "somboon@gmail.com"  
7 }
```

Body 200 OK 75 ms 398 B Save

Pretty Raw Preview Visualize JSON

```
1 {  
2   "data": {  
3     "_id": "5fbcaaa4c997c93d94f92a99",  
4     "username": "user",  
5     "firstName": "somchai",  
6     "email": "somboon@gmail.com"  
7   },  
8   "message": "create success",  
9   "success": true  
10 }
```


Get All Users

เพิ่ม โค้ดส่วนนี้ในไฟล์ routes/users.js

```
router.get("/", async function (req, res, next) {  
  try {  
    const users = await Users.find();  
    return res.status(200).send({  
      data: users,  
      message: "success",  
      success: true,  
    });  
  } catch (error) {  
    return res.status(500).send({  
      message: "fail",  
      success: false,  
    });  
  }  
});
```

GET ▼ http://localhost:3000/users Send ▼

Params Auth ● Headers (7) Body Pre-req. Tests Settings

none ▼

This request does not have a body

Body ▼ 🌐 200 OK 25 ms 475 B Save

Pretty Raw Preview Visualize JSON ↺

```
1 {
2   "data": [
3     {
4       "_id": "5fbcaaaa4c997c93d94f92a99",
5       "username": "user",
6       "password": "$2b$10$FFHRYjmambUkAMpDnQkJpuRa6mEQihhr9xDtPkSEeVFs.6mxMYmdu",
7       "firstName": "somchai",
8       "email": "somboon@gmail.com",
9       "__v": 0
10    }
11  ],
12  "message": "success",
13  "success": true
14 }
```

เพิ่ม route login ไปที่
routes/index.js

```
const Users = require("../models/user");
const bcrypt = require("bcrypt");
router.post("/login", async function (req, res, next) {
  try {
    let { password, username } = req.body;
    let user = await Users.findOne({
      username: username,
    });
    if (!user) {
      return res.status(500).send({
        message: "login fail",
        success: false,
      });
    }
    const checkPassword = await bcrypt.compare(password, user.password);
    if (!checkPassword) {
      return res.status(500).send({
        message: "login fail",
        success: false,
      });
    }
    const { _id, firstName, lastName, email } = user;
    return res.status(201).send({
      data: { _id, firstName, lastName, email },
      message: "login success",
      success: true,
    });
  } catch (error) {
    return res.status(500).send({
      message: "login fail",
      success: false,
    });
  }
});
```

User Login

```
var express = require("express");
var router = express.Router();
const Users = require("../models/user");
const bcrypt = require("bcrypt");
router.post("/login", async function (req, res, next) {
  try {
    let { password, username } = req.body;
    let user = await Users.findOne({
      username: username,
    });
    if (!user) {
      return res.status(500).send({
        message: "login fail",
        success: false,
      });
    }
    const checkPassword = await bcrypt.compare(password, user.password);
    if (!checkPassword) {
      return res.status(500).send({
        message: "login fail",
        success: false,
      });
    }
  }
});
```

ตัวอย่างการนำไปใช้ตรวจสอบ password
ในการ login โดยใช้ **bcrypt.compare**

POST http://localhost:3000/login Send

Params Auth Headers (8) **Body** Pre-req. Tests Settings

raw JSON

```
1 {  
2   "username": "user",  
3   "password": "1234"  
4 }
```

Body 201 Created 75 ms 384 B Save

Pretty Raw Preview Visualize JSON

```
1 {  
2   "data": {  
3     "_id": "5fbcaaa4c997c93d94f92a99",  
4     "firstName": "somchai",  
5     "email": "somboon@gmail.com"  
6   },  
7   "message": "login success",  
8   "success": true  
9 }
```

JWT TOKEN

Middleware คืออะไร



โดยปกติแล้วเนี่ย การติดต่อร้องขอ **Request** ไปยังเซิร์ฟเวอร์ ไม่มีการคัดกรองอะไร ดังรูป

Middleware คืออะไร



ถ้าข้อมูลตอบกลับเป็นข้อมูลที่มีความเป็นส่วนตัวระดับหนึ่ง เช่น การร้องขอ ยอดเงินคงเหลือ ของบัญชี นายบอย เราคงไม่อยากจะให้ใครที่ไม่ใช่ นายบอย มาดูได้ เราจึงต้องใช้ “middleware” หรือตัวกลาง ที่จะมาคัดกรองการร้องขอนี้ และคัดกรองก่อนว่าจะอนุญาตให้คืนกลับไปไหม

JWT คืออะไร

JWT ย่อมาจาก JSON Web Token หมายความว่า การเก็บข้อมูล JSON ไว้ใน token อันหนึ่ง ที่เป็นสายสตริงตัวอักษร ยาวๆ

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzQ5MDIyLjE1KXwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

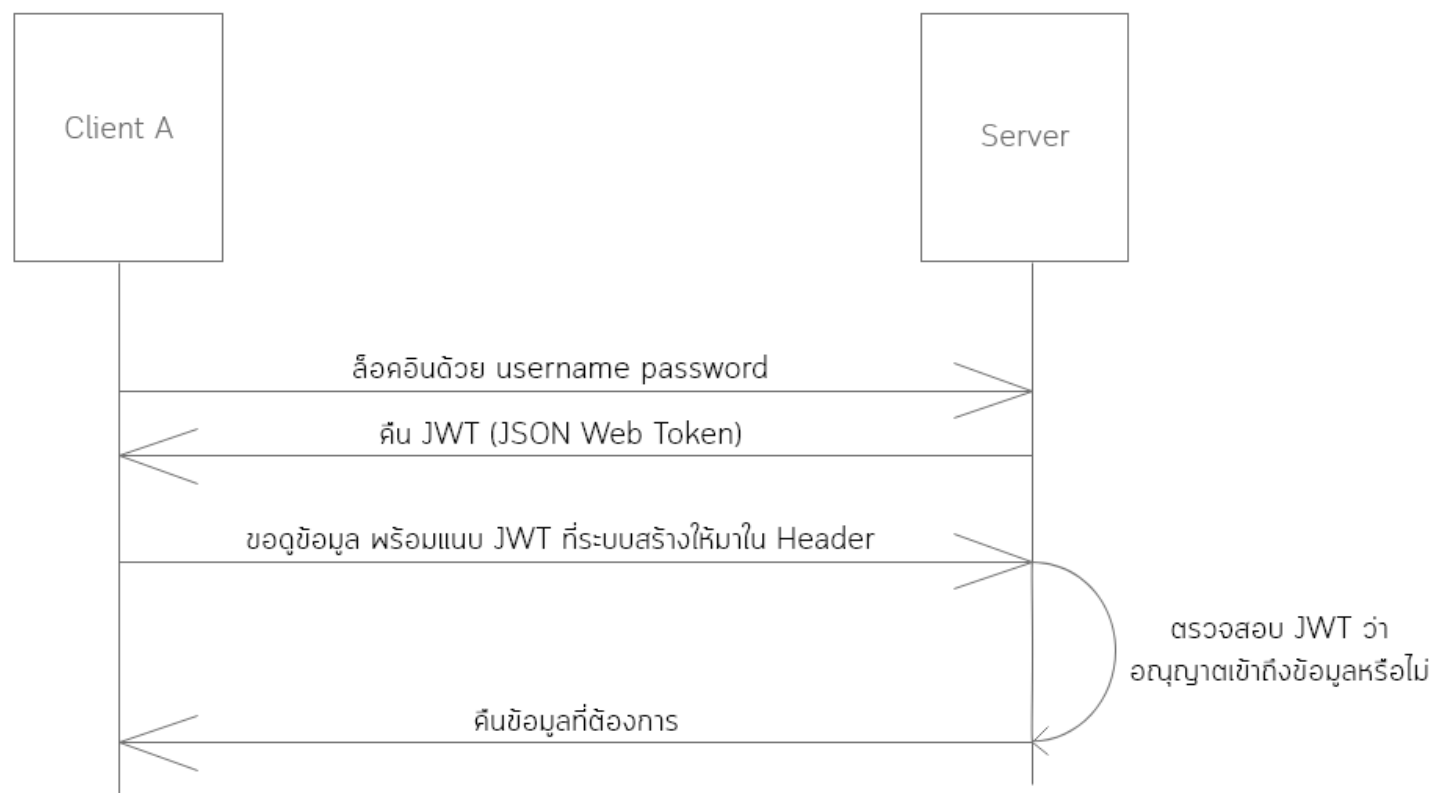
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

โดยตัว JWT จะประกอบด้วยกัน 3 ส่วน คั่นด้วย "." ดังนี้ <Header>.<Payload>.<Signature>

Header จะประกอบด้วยการบอกว่าข้อมูลข้างต้นถูกเข้ารหัสด้วย อัลกอริทึมอะไรและมีชนิดอะไร

Payload จะเป็นข้อมูลที่เรารู้เก็บ ที่ไม่สำคัญ ห้ามเก็บ password โดยตรงเพราะสามารถนำไปถอดรหัสกลับได้ นิยมใช้เก็บไอดีของผู้ใช้ หรือ “sub”
มาจากคำว่า subject

Signature จะเป็นการนำ Header และ Payload มาเข้ารหัสด้วย Secret-key ที่เรารู้ไว้ เพื่อเป็นการยืนยันว่า JWT นี้ไม่ถูกปลอมแปลงจากคนอื่น



```
npm i jsonwebtoken
```

ตัวอย่างที่ การสร้าง token จากการ login

กำหนด private key ที่ใช้สร้าง jwt token ไว้ที่ไฟล์ .env
ตัวอย่างเช่น

JWT_KEY=1JFAi4s5D5L4MYkUgfDHyeLkGnum73d

ประกาศ jwt ไว้สำหรับการเรียกใช้งานการ generate token

```
const jwt = require("jsonwebtoken");
```

คำสั่งใช้การเข้ารหัสโดย
Syntax คือ

```
jwt.sign(payload, secretOrPrivateKey, [options, callback])
```

payload คือ ค่าที่จะเก็บไว้ใน jwt ควรเป็น ค่าที่ไม่สำคัญแต่สามารถยืนยันตัวตนได้
เช่น id หรือ username

<https://www.npmjs.com/package/jsonwebtoken>

```
const token = jwt.sign(  
  {  
    _id: user._id,  
    name: user.name,  
    username: username,  
    position: user.position,  
  },  
  process.env.JWT_KEY  
);  
return res.status(200).send({  
  token: token,  
  data: {  
    _id: user._id,  
    name: user.name,  
    username: username,  
    position: user.position,  
  },  
  message: "success",  
  success: true,  
});
```

การ Decode JWT token

คำสั่งใช้การ decode โดย
Syntax คือ

```
jwt.verify(token, secretOrPublicKey, [options, callback])
```

ตัวอย่างการใช้งาน

```
try {  
  const token = req.headers.authorization.split("Bearer ")[1];  
  const decoded = jwt.verify(token, process.env.JWT_KEY);  
  req.auth = decoded;  
  return next();  
}  
catch (error) {  
  return res.status(401).json({  
    message: 'Auth failed'  
  })  
}
```

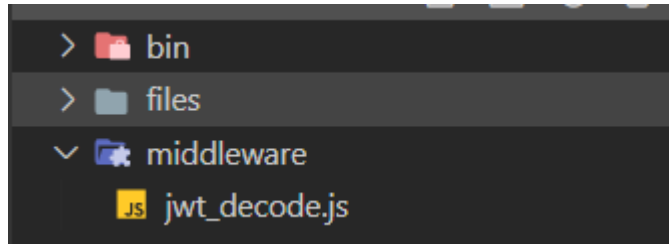
โดยใช้มาตรฐานการส่ง token ด้วย Bearer โดยการส่ง token มาทาง authorization header

Syntax

Authorization: Bearer <token>

การนำ JWT เป็น middleware สำหรับกรอง request ไม่ให้ร้องขอข้อมูล

สร้างไฟล์ `jwt_decode` ในโฟลเดอร์ `middleware` สำหรับการกรอง token (หากไม่มีโฟลเดอร์ `middleware` สร้างขึ้นมาใหม่ได้เลย)



```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  try {
    const token = req.headers.authorization.split("Bearer ")[1];
    const decoded = jwt.verify(token, process.env.JWT_KEY);
    req.auth = decoded;
    return next();
  } catch (error) {
    return res.status(401).json({
      message: 'Auth failed'
    });
  }
}
```

ดึง token ออกจาก Authorization header
และ ตัด "Bearer " ออกเพื่อได้ token ที่ถูกต้อง

ทำการตรวจสอบ token

กำหนดค่าให้ request (มีหรือไม่มีก็ได้ set เพื่อไปใช้ต่อ)

ทำการเรียกใช้ middleware สำหรับ route ที่ต้องการตรวจสอบ token

```
// ...  
const verifyToken = require('./middleware/jwt_decode')  
app.use(cors());  
  
// view engine setup  
app.set("views", path.join(__dirname, "views"));  
app.set("view engine", "ejs");  
  
app.use(logger("dev"));  
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));  
app.use(cookieParser());  
app.use(express.static(path.join(__dirname, "public")));  
  
app.use("/", indexRouter);  
app.use("/users", verifyToken, usersRouter);  
app.use("/products", productsRouter);  
app.use("/files", uploadRouter);  
// catch 404 and forward to error handler
```

ในกรณีที่ต้องการตรวจสอบทุก request ของ path นั้นๆ
สามารถใส่ไปที่ root path ของ api นั้นได้เลย

หรือ ในกรณีที่ต้องการตรวจสอบเฉพาะ path ของ api นั้นๆ เช่น
ไฟล์ user.js

```
const verifyToken = require('./middleware/jwt_decode')  
/* GET users listing. */  
router.get("/", verifyToken, async function (req, res, next) {  
  try {  
    const users = await Users.find();  
    return res.status(200).send({  
      data: users,  
      message: "success",  
      success: true,  
    });  
  } catch (error) {  
    return res.status(500).send({  
      message: "fail",  
      success: false,  
    });  
  }  
});
```


Generate token ผ่าน การ login

```
var router = express.Router();  
const Users = require("../models/user");  
const bcrypt = require("bcrypt");  
const jwt = require('jsonwebtoken');  
router.post("/login", async function (req, res) {  
  try {
```

```
    const checkPassword = await bcrypt.compare(password, user.password);  
    if (!checkPassword) {  
      return res.status(500).send({  
        message: "login fail",  
        success: false,  
      });  
    }  
    const { _id, firstName, lastName, email } = user;  
    const token = jwt.sign({ _id, firstName, lastName, email }, process.env.JWT_KEY)  
    return res.status(201).send({  
      data: { _id, firstName, lastName, email, token },  
      message: "login success",  
      success: true,  
    });  
  } catch (error) {  
    console.log(error);  
  }  
});
```

ไฟล์ .env

```
DB_HOST=localhost  
DB_PORT=27017  
DB_NAME=62530  
DB_USER=mongoDB  
DB_PASS=mongoDB  
JWT_KEY=1JFAi4s5D5L4MYkUgfdHyeLKGnum73d
```

ตัวอย่างการนำไปใช้งาน

1.

POST ▼ http://localhost:3000/login Send Save ▼

Params Auth ● Headers (9) Body ● Pre-req. Tests Settings Cookies Code


raw ▼ JSON ▼ Beautify

```

1 {
2   "username": "user",
3   "password": "1234"
4 }

```

Body ▼ 201 Created 117 ms 612 B Save Response ▼

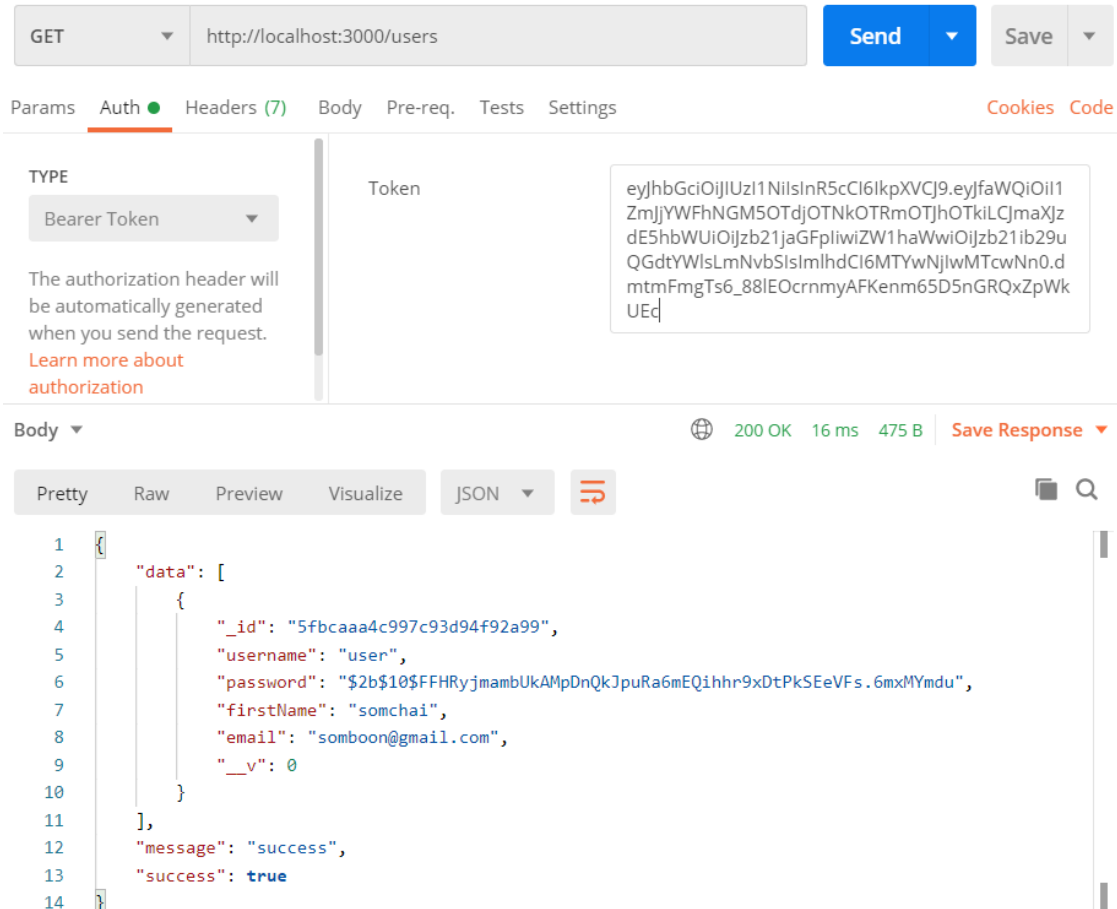
Pretty Raw Preview Visualize JSON ▼ 

```

1 {
2   "data": {
3     "_id": "5fbcaaa4c997c93d94f92a99",
4     "firstName": "somchai",
5     "email": "somboon@gmail.com",
6     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1ZmJjYWZhNGM5OTdjOTNkOTRmOTJhOTkiLCJmaXJzdE5hbWUiOiJzb21jaGFpIiwiaWwiOiJzb21ib29uQGdtYWlsLmNvbSI6Im1hdCI6MTYwNjIwMzQ2ODQ0Lm1hZCJ9.TQw0ZVAV-HTYlpVE1lsuoWjTccqb2ase340kk4vnoTI"
7   },
8   "message": "login success",
9   "success": true
10 }

```

2. copy token ที่ได้จาก login มาใส่ที่ Auth postman



GET http://localhost:3000/users

Send Save

Params Auth Headers (7) Body Pre-req. Tests Settings Cookies Code

TYPE
Bearer Token

The authorization header will be automatically generated when you send the request.
[Learn more about authorization](#)

Token

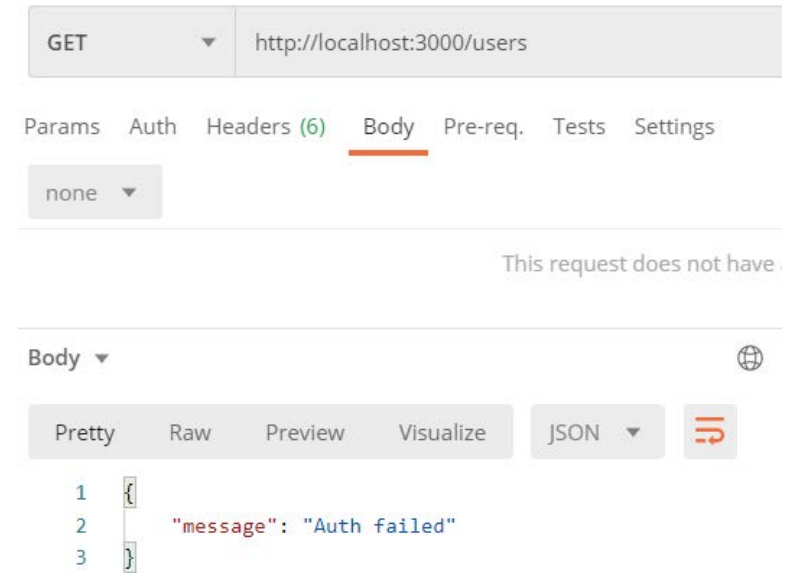
```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1ZmJjYWZhNGM5OTdjOTNkOTRmOTJhOTkiLCJmaXJzIjoiZmFpbWUuOiIjZb21jaGFpIiwiaWF0IjZb21ib29uQGdtYWlsLmNvbSIsImh0dCI6MTYwNjIwMTcwNn0.dmtmFmgTs6_88IEOcrnmyAFKenm65D5nGRQxZpWkUEd
```

200 OK 16 ms 475 B Save Response

Pretty Raw Preview Visualize JSON

```
{
  "data": [
    {
      "_id": "5fbcaaa4c997c93d94f92a99",
      "username": "user",
      "password": "$2b$10$FFHryjmambUkAMpDnQkJpuRa6mEQihhr9xDtPkSEeVfs.6mxMYndu",
      "firstName": "somchai",
      "email": "somboon@gmail.com",
      "__v": 0
    }
  ],
  "message": "success",
  "success": true
}
```

หากไม่ทำการส่ง token



GET http://localhost:3000/users

Params Auth Headers (6) Body Pre-req. Tests Settings

none

This request does not have

Body

Pretty Raw Preview Visualize JSON

```
{
  "message": "Auth failed"
}
```

BASIC JAVASCRIPT

COURSE COMPLETED