

# Node.js (2)

ผศ. ดร. เก็จแก้ว ธเนศวร

[kejkaew.tha@kmutt.ac.th](mailto:kejkaew.tha@kmutt.ac.th)

# ตารางเรียน

สัปดาห์	วันที่	หัวข้อ/รายละเอียด	การบ้าน (ผู้ตรวจ)	ส่ง Assignment
1	5/8/2568	Introduction to Web programming, Basic HTML 1	Assignment 1 (KT,KN)	1
	12/8/2568	หยุด		
2	19/8/2568	Basic HTML 2 + CSS 1	Assignment 2 (KT, KN)	1,2
3	26/8/2568	CSS 2	Assignment 3 (KT)	1,2,3
4	2/9/2568	Responsive Web design	Assignment 4 (KT, KN)	2,3,4
5	9/9/2568	JavaScript 1 + DOM	Assignment 5 (KT, KN)	3,4,5
6	16/9/2568	JavaScript 2 + GitHub	Assignment 6 (KT)	4,5,6
7	23/9/2568	JavaScript 3	Assignment 7 (KT, KN)	5,6,7
8	30/9/2568	AJAX + JSON	Assignment 8 (KT, KN)	6,7,8
9	7/10/2568	Node.js 1	Assignment 9 (KT)	7,8,9
10	14/10/2568	Node.js 2	Assignment 10 (KT, KN)	8,9,10
	21/10/2568	หยุด		
11	28/10/2568	Node.js 3	Assignment 11 (KT, KN)	9,10,11
12	4/11/2568	Node.js 4 + MySQL	Assignment 12 (KT, KN)	10,11,12
13	11/11/2568			11,12
14	18/11/2568			12
16	25/11/2568	ส่งโปรเจก (9:00-16:00)	KT	
17	/12/2568	สอบปลายภาค ในตาราง		

# Web Programming Project: Web application

---

- Web application เป็นเหมือนแอปพลิเคชัน ที่ถูกพัฒนามาให้แสดงผลผ่าน web browser
- คะแนน 35 คะแนน
- กลุ่มละไม่เกิน 3 คน (จับกลุ่มกับเพื่อนต่าง section ได้)

- 
- เลือกทำ 1 อย่างจากหัวข้อต่อไปนี้
    - Social media platform
    - Browser game platform
    - Online shopping platform
  - กรณาดูรายละเอียดใน slide หน้าถัดไปด้วย
  - สามารถใช้ CSS, JavaScript, Backend framework ได้
  - ไม่ให้ใช้ web builder หรือ CMS ต่าง ๆ เช่น word press, strapi
  - ไม่ใช้ Unity
  - ส่งรายชื่อ พร้อมเขียน project plan ว่าเราจะทำอะไร พร้อมแผนคร่าว ๆ ส่งวันที่ 25 ต.ค. 2567

## รายละเอียด

---

- Social media platform: สิ่งที่ต้องมีเป็นอย่างน้อย
  - Login page
  - Register page
  - Profile page ของผู้ใช้
  - Feed หรือ Dashboard แสดงข้อความต่างๆ ที่คนใน social media post
  - ผู้ใช้สามารถ post ข้อความ และแสดงใน feed ได้ กด like กด love post ของคนอื่นได้
  - 1 database

- 
- Online shopping platform: สิ่งที่ต้องมีเป็นอย่างน้อย
    - Login page
    - Register page
    - มีหน้า Home แสดงสินค้าต่างๆ ที่กำลังมีขาย
    - ผู้ใช้เป็นผู้ได้ทั้งคนซื้อและคนขาย
    - ผู้ใช้สามารถลงสินค้าใหม่เพื่อขายหรือซื้อสินค้าได้
    - มีหน้า checkout แสดงราคาของทั้งหมดที่ซื้อ
    - 1 database

- 
- Browser game: สิ่งที่ต้องมีเป็นอย่างน้อย
    - Login page
    - Register page
    - Game page (หาเกมที่มีอยู่แล้วได้)
    - Leader board page
    - ต้องกด Like/Love และ comment ใน leader board ได้
    - มีหน้า profile ของผู้ใช้
    - 1 database

# เรื่องที่เรียนไปสัปดาห์ที่แล้ว

---

- Node.js
- http module
- File system module
- Promises

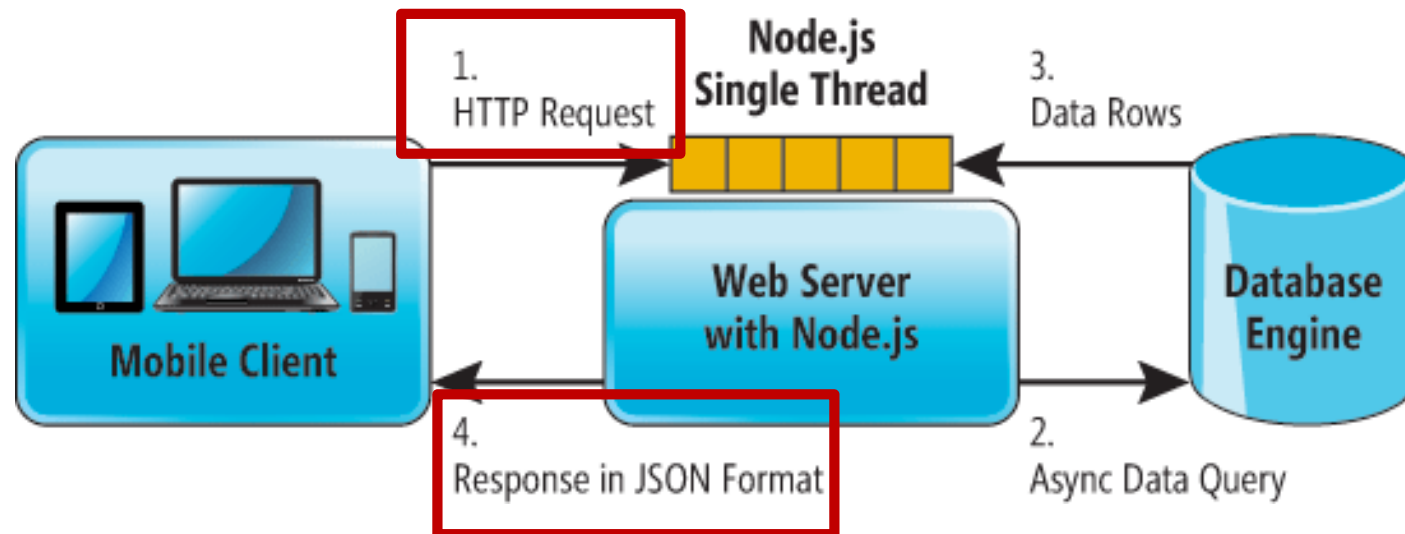


# เรื่องที่จะเรียนวันนี้

---

- Express module (Backend)
- Fetch API (Frontend)
- Body-Parser (Frontend-Backend)
- Async / Await (Frontend -Backend)
- Assignment 10

# ทบทวน



- Client – Server relationship

# NPM

---

- Node Package Manager
- NPM คือ package manager สำหรับ Node.js packages หรือ modules.

วิธีการ install

`npm install <Module Name>`

- ตัวอย่างเช่น install express framework

`npm install express`

# Nodemon module

---

- ใช้สำหรับรัน server ได้เหมือนกับคำสั่ง node
- แต่ข้อดีของ module นี้คือ จะ restart server ได้เองเมื่อ server file มีการเปลี่ยนแปลง

- Installation

npm install -g nodemon

- วิธีการเรียกใช้

**nodemon** server.js

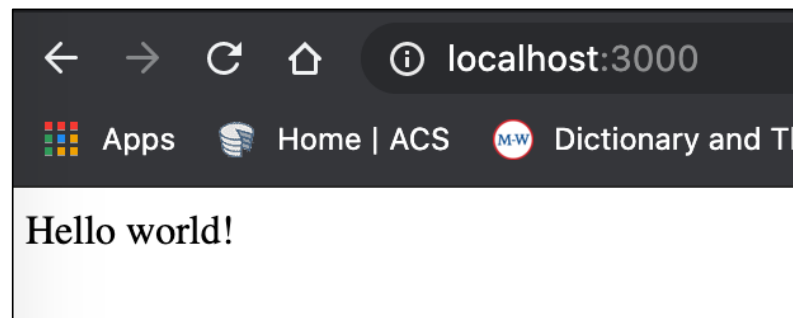
# Express Framework

---

- Express คือ Node.js web application framework ที่ใช้สำหรับ พัฒนา web และ mobile applications.
- ทำให้พัฒนา web applications ได้ง่ายและไวขึ้น
- ตัวอย่างที่ Express สามารถทำได้
  - สร้าง middlewares สำหรับจัดการกับ HTTP Requests.
  - กำหนด routing เพื่อทำงานบางอย่างได้ โดยใช้ร่วมกับ HTTP Method และ URL.
  - สามารถ render dynamic HTML Pages ได้ โดยการกำหนด arguments

## Express: hello world (ใน Node-2)

```
1 const express = require('express');
2 const app = express();
3 const hostname = 'localhost';
4 const port = 3000;
5
6 app.get('/', (req, res) => {
7   res.send("Hello world!");
8 });
9
10 app.listen(port, hostname, () => {
11   console.log(`Server running at http://${hostname}:${port}/`);
12 });
```



**Output**

# Routing

---

- Express framework สามารถจัดการ route (เส้นทางการทำงาน, event) ต่างๆ ของ web application ได้ โดยใช้

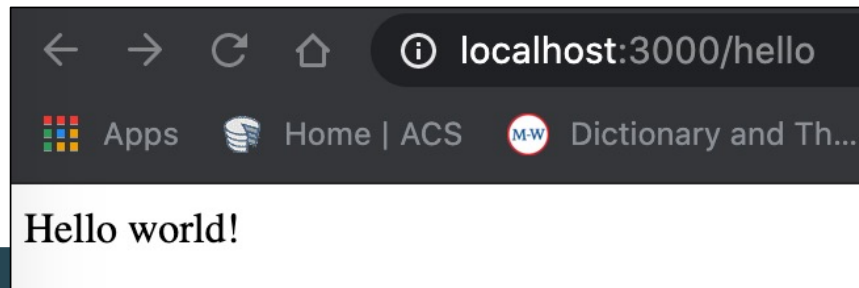
`app.method(path, callback)`

- Method สามารถเป็น get, post, use เป็นต้น
- path คือ route ที่ต้องการให้ทำงานบางอย่าง

## ตัวอย่าง

```
14 //routing
15 const express = require('express');
16 const app = express();
17 const hostname = 'localhost';
18 const port = 3000;
19
20 app.get('/hello', (req, res) => {
21   res.send("Hello world!");
22 });
23
24 app.listen(port, hostname, () => {
25   console.log(`Server running at http://\${hostname}:\${port}/`);
26 });
```

## Output





# Middleware

---

- ตัวกลางที่ไว้จัดการ request และ response
- functions ที่สามารถเข้าถึง request object (req), response object (res), และ middleware function ถัดไป (next function) ของ web application
- Middleware functions เหล่านี้ ใช้สำหรับ แก้ไข req และ res objects เช่น กำหนดประเภทของข้อมูลสำหรับการส่ง
- `app.use(callback function)`

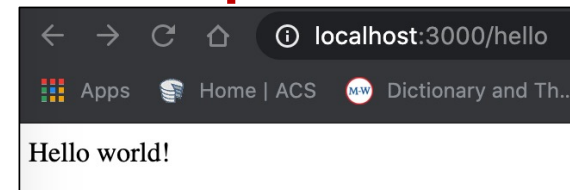
## ตัวอย่าง

```
28 // middleware
29 const express = require('express');
30 const app = express();
31 const hostname = 'localhost';
32 const port = 3000;
33
34 app.use((req, res, next) => {
35   console.log("A request for things received at " + Date.now());
36   next();
37 })
38
39 app.get('/hello', (req, res) => {
40   res.send("Hello World!");
41 });
42
43 app.listen(port, hostname, () => {
44   console.log(`Server running at http://${hostname}:${port}/`);
45 });
```

### Output: server

```
Server running at http://localhost:3000/
A request for things received at 1617008777380
```

### Output: html



## Order of Middleware Calls

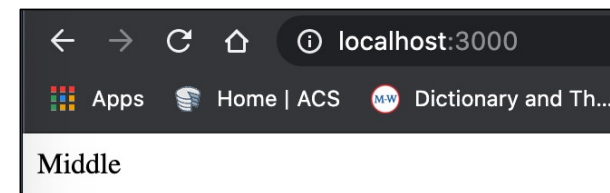
---

- ลำดับการเขียน middleware และ route handler มีความสำคัญ
- โดยโปรแกรมจะทำงานตามลำดับที่เขียนไว้

# ตัวอย่าง

```
47 const express = require('express');
48 const app = express();
49 const hostname = 'localhost';
50 const port = 3000;
51
52 app.use((req, res, next) => {
53   console.log("Start");
54   next();
55 })
56
57 app.get('/', (req, res, next) => {
58   res.send("Middle");
59   next();
60 });
61
62 app.get('/', (req, res) => {
63   console.log("End");
64 });
65
66 app.listen(port, hostname, () => {
67   console.log(`Server running at http://\${hostname}:\${port}/`);
68 });
```

## Output: html



## Output: server

```
Server running at http://localhost:3000/
Start
End
```

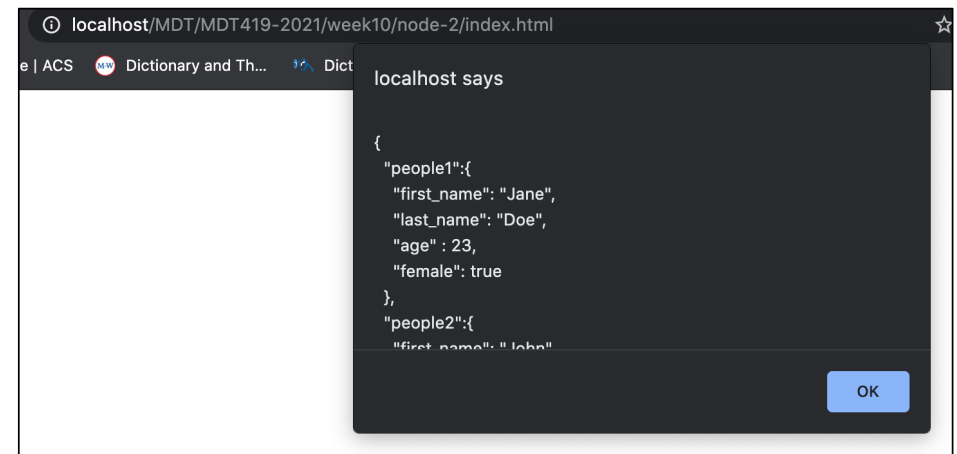
# Fetch API

---

- `fetch()` มีการทำงานคล้ายกับ XMLHttpRequest (XHR).
- แต่สิ่งที่แตกต่างคือ Fetch API มีการใช้ Promises ทำให้หลีกเลี่ยงการเกิด callback hell ได้
- แต่ AJAX ยังสามารถเกิด callback hell ได้
- `fetch` function รันบน JavaScript ผ่านทาง browser

## XHR vs. Fetch: XHR (AJAX ใน script.js)

```
2 window.onload = pageLoad;
3 function pageLoad(){
4     var xhr = new XMLHttpRequest();
5     xhr.open("GET", "data_2.json",true);
6     xhr.onload = function() {
7         alert(xhr.responseText);
8     };
9     xhr.onerror = function() {
10        alert("ERROR!");
11    };
12    xhr.send();
13 }
```



**Output: html**

## XHR vs. Fetch : fetch (ใน script.js)

```
16 const readLog = (() => {
17     fetch("data_2.json").then((response) => {
18         response.json().then((data) => {
19             console.log(data);
20         });
21     }).catch((err) => {
22         console.log(err);
23     })
24 })
25
26 readLog();
```

**Output: html**

```
▼ Object ⓘ
  ► people1: {first_name: "Jane", last_name: "Doe", age: 23, female: true}
  ► people2: {first_name: "John", last_name: "Dee", age: 30, female: false}
  ► __proto__: Object
```

# Fetch and Node.js

---

- Fetch function รันบน client browser
  - ส่ง (request) ข้อมูลไปให้ server และรอรับ (response) ข้อมูลที่ส่งกลับมา
- Node.js รันบน server
  - โดยเราจะใช้ express module ในการรับ request จาก client และส่ง response กลับ



## การใช้ Fetch API

- การใช้ Fetch API เริ่มจากเรียกใช้ fetch method และมี parameter คือ url  
`fetch(url)`
- Fetch defaults คือ GET requests แต่จะใช้ POST ก็ได้
- หลังจากเรียกใช้ the fetch() method จะทำการเรียกใช้ promise method นั้น  
คือ then()  
`.then(function() { })`

## การใช้ Fetch API

---

- ใน fetch() method ถ้าทำงานสำเร็จจะ return resolve แล้วนำค่าที่ได้จากการ return นั้นไปทำงานต่อใน then()
- ต่อจาก then() method จะเป็น catch() method ใช้สำหรับจัดการกับ error ที่ได้จาก reject ใน promise

`.catch(function() { });`

# สรุป

---

- Fetch syntax

```
fetch(url)
.then(function() {

})
.catch(function() {

});
```

## การใช้ response ของ Fetch API

- ต้องสร้าง function ที่รับ response argument โดย
  - response ที่เป็น string
    - จะต้องสร้าง resolve promise จาก response.text() โดยสร้าง function ใน then() ที่รับ argument 1 ตัว และ argument ตัวนั้น จะเป็น response ที่เราต้องการ
  - response ที่เป็น JSON
    - จะต้องสร้าง resolve promise จาก response.json() โดยสร้าง function ใน then() ที่รับ argument 1 ตัว และ argument ตัวนั้น จะเป็น response ที่เราต้องการ

## ตัวอย่าง fetch ใน (client-server folder)

- ใน client-server folder -> index.js ยังไม่ต้องรัน

```
2  const sendMsg = (() => {  
3      fetch("/message").then((response) => {  
4          response.text().then((data) => {  
5              console.log(data);  
6          });  
7      }).catch((err) => {  
8          console.log(err);  
9      })  
10 })  
11 window.onload = sendMsg;
```

## Serving static files in Express

---

- Node.js ไม่รู้ว่า client content (html, JS, CSS) อยู่ที่ไหน ไม่รู้ว่าจะรอ request จาก client ไหน
- เราต้องบอกว่า server ว่า client content อยู่ใน directory ไດ โดยใช้ express.static method

`express.static(root, [options])`

- ใช้ Middleware function เข้ามาช่วยในการจัดการ request และ response
- `__dirname` เป็น global object ที่เก็บชื่อ directory ที่รัน node.js อยู่
- เช่น ถ้ารัน node ที่ `/home/user/env` ตัว `__dirname` จะเป็น  
`/home/user/env`

ดังนั้นจะได้

- `app.use(express.static(__dirname));`
- เป็นการบอกว่า request ที่เข้ามาจะต้องอยู่ใน directory เดียวกันกับ server นะ  
ถึงจะทำงานต่อได้

## ตัวอย่าง server.js

```
2 const express = require('express');
3 const app = express();
4 const hostname = 'localhost';
5 const port = 3000;
6 const bodyParser = require('body-parser');
7
8 app.use(express.static(__dirname));
9
10 app.get('/message', (req, res) => {
11   console.log("test");
12   res.send("Hello world!");
13 });
14
15 app.listen(port, hostname, () => {
16   console.log(`Server running at http://\${hostname}:\${port}/`);
17 });
```

ลองรัน: <http://localhost:3000>

ลองรัน: <http://localhost:3000/message>

## Output: server

```
(base) Kejkaews-MacBook-Pro:client-server Kejkaew$ node server.js
Server running at http://localhost:3000/
test
```

## Output: html

```
Navigated to http://localhost:3000/
Hello world!
▶ Fetch finished loading: GET "http://localhost:3000/message".
▶ XHR finished loading: GET "chrome-extension://mgijmajocgfcbeboacabfgobmjgcoja/content.min.css".
```

ผลที่ได้แตกต่างกันอย่างไร



## Fetch: POST

---

- Fetch defaults คือ GET requests แต่จะใช้ POST ก็ได้ เช่น เมื่อต้องการส่ง JSON data ไปให้ server ซึ่งสามารถทำได้โดยกำหนด
  - method เป็น POST
  - body เป็น JSON data
  - headers เป็น {'Accept': 'application/json', 'Content-Type': 'application/json'}

## ตัวอย่าง fetch: POST

```
14 const sendMsg = (msg) => {  
15     fetch("/message", {  
16         method: "POST",  
17         headers: {  
18             'Accept': 'application/json',  
19             'Content-Type': 'application/json'  
20         },  
21         body: JSON.stringify({  
22             name: "John",  
23             age: 60  
24         })  
25     }).then((response) => {  
26         response.json().then((data) => {  
27             console.log(data);  
28         });  
29     }).catch((err) => {  
30         console.log(err);  
31     });  
32 };  
34 window.onload = sendMsg;
```

## Body-parser middleware

- เนื่องจาก การส่ง input จาก user นั้น ตัว input สามารถเป็นได้หลากหลายแบบ เช่น string, json data etc. เราอาจจะต้องเช็คก่อนว่า input ที่ส่งมาเป็นไปตาม format ที่เราต้องการหรือไม่
- โดยการใช้ body-parser

Install body-parser via terminal

```
npm install body-parser
```

- 
- เริ่มต้นจากการทำการ import body-parser

```
const bodyParser = require('body-parser');
```

- จะใช้ body-parser ร่วมกับ middleware เพื่อจัดการกับ request ที่เป็น JSON data โดยใช้

```
bodyParser.json([options])
```

- Returns middleware ที่มีรูปแบบเป็น JSON และจัดการกับ request ที่มี content-type เป็น JSON data ให้เป็น JS object

```
app.use(bodyParser.json());
```

## bodyParser.urlencoded([options])

- Returns middleware ที่มีรูปแบบเป็น object จาก url encoded bodies
- ใช้สำหรับกำหนด body object ใหม่
- ใน options สามารถเพิ่ม argument ได้ และใน extended ถ้า value เป็น true จะบอกว่า object ใหม่นั้น มี value เป็นประเภทอะไรก็ได้ แต่ถ้าเป็น false ค่า value จะเป็น string หรือ array

```
app.use(bodyParser.urlencoded({extended: false}));
```

## ตัวอย่าง server สำหรับ POST

```
20 const express = require('express');
21 const app = express();
22 const hostname = 'localhost';
23 const port = 3000;
24 const bodyParser = require('body-parser');
25
26 app.use(express.static(__dirname));
27 app.use(bodyParser.json());
28 app.use(bodyParser.urlencoded({extended: false}));
29
30 app.post('/message', (req, res) => {
31   console.log(req.body)
32   const outMsg = req.body
33   outMsg["lastname"] = "Doe";
34   console.log(outMsg);
35   res.json(outMsg);
36 })
37
38 app.listen(port, hostname, () => {
39   console.log(`Server running at http://\${hostname}:\${port}`);
40 });
```

### Output: server

```
Server running at http://localhost:3000/
{ name: 'John', age: 60 }
{ name: 'John', age: 60, lastname: 'Doe' }
```

### Output: html

```
Navigated to http://localhost:3000/
▶ Fetch finished loading: POST "http://localhost:3000/message".
▶ {name: "John", age: 60, lastname: "Doe"}
```

## ทบทวน Promises

- Promises เป็นวิธีที่ใช้จัดการกับ asynchronous code เพื่อไม่ให้เกิด callback hell
- เมื่อ promise ถูกเรียกใช้ มันจะอยู่ในสถานะ pending state ซึ่งหมายความว่า callback ทุกตัวที่มี promise ยังทำงานอยู่ และเมื่อตัว callback ก่อนหน้าทำงานสำเร็จ (resolve) แล้ว callback ตัวนั้นก็จะเริ่มทำงานต่อ

- 
- ดังนั้นในการสร้าง promise สุดท้ายแล้วจะเกิดได้สองสถานะ คือ resolved state หรือ rejected state ไปเรื่อยๆ จนจบ chain ของ promise
  - Promise API เริ่มต้นโดยการเรียกใช้ Promise constructor ซึ่งมี syntax คือ `new Promise(callback):`



## วิธีการใช้ promise ที่สร้างไว้

- ในการใช้ promise ถ้าทำงานสำเร็จจะ return resolve method พร้อมกับค่าที่ได้จากการทำงานสำเร็จ นั้นไปทำงานต่อใน then()
- ต่อจาก then() method จะเป็น catch() method ใช้สำหรับจัดการกับ error ที่ได้จาก reject method ใน promise

## ตัวอย่าง ที่ไม่ใช่ promises (ครั้งที่แล้ว)

```
84 const fs=require('fs');
85 fs.readFile('test.txt', (err, data) => {
86     if (err)
87         throw err;
88     else{
89         fs.writeFile('message.txt', data , (err) => {
90             if (err)
91                 throw err;
92             console.log('saved!');
93         });
94     }
95 });
```

### Output

```
(base) Kejkaews-MacBook-Pro:eventloop Kejkaew$ node server.js
saved!
```

## ตัวอย่าง Promises (ครั้งที่แล้ว)

```
97 const fs = require('fs');
98 const readData = () => {
99     return new Promise((resolve, reject) => {
100         fs.readFile('test.txt', (err, data) => {
101             if (err)
102                 reject(err);
103             else
104                 resolve(data);
105         });
106     });
107 }
```

```
109 const writeData = (data) => {
110     return new Promise((resolve, reject) => {
111         fs.writeFile('message.txt', data, (err) => {
112             if (err)
113                 reject(err);
114             else
115                 resolve("saved!");
116         });
117     });
118 }
119
120 readData().then(writeData).then((out) => console.log(out));
```

## Output

```
(base) Kejkaews-MacBook-Pro:eventloop Kejkaew$ node server.js
saved!
```

## การใช้ Async/Await

---

- ทำให้การใช้ promise ใน JavaScript เข้าใจได้ง่ายขึ้น
- สามารถทำได้โดย
  - ประกาศ async หน้า function ที่จะทำการเรียก function ที่มี promise
  - ประกาศ await หน้า ตัวแปรผลลัพธ์ของการทำงานของ function ที่มี promise หรือ ตัวแปรที่เรารอผลจาก server เช่น response

## ตัวอย่าง Async/Await อ่าน- เขียน ไฟล์

server.js

```
28 // async and await
29 // create promise
30 const fs = require('fs');
31 const readData = () => {
32     return new Promise((resolve, reject) => {
33         fs.readFile('jfile.json', 'utf8', (err,
34             data) => {
35             if (err)
36                 reject(err);
37             else
38                 {
39                     console.log(data);
40                     resolve(data);
41                 }
42             });
43     });
44 }
45
46 const writeData = (data) => {
47     return new Promise((resolve, reject) => {
48         fs.writeFile('new_jfile.json', data, (err)
49             => {
50             if (err)
51                 reject(err);
52             else
53                 resolve("saved!")
54             });
55     });
56 }
```

## ตัวอย่าง Async/Await อ่าน-เขียน ไฟล์

```
57 const callFun = async () => {  
58     let data_r = await readData();  
59     let text_w = await writeData(data_r);  
60     console.log(data_r);  
61     console.log(text_w);  
62 }  
63 callFun();
```

server.js

Output: server

```
^C(base) Kejkaews-MacBook-Pro:client-server Kejkaew$ node server.js  
{  
  "name": "Joe",  
  "lastname": "Doe",  
  "age": 55  
}  
{  
  "name": "Joe",  
  "lastname": "Doe",  
  "age": 55  
}  
saved!
```

## ตัวอย่าง Async/Await ใน fetch

index.js

```
36 // async await in fetch
37 const sendMsg = (async (msg) => {
38     let response = await fetch("/message", {
39         method: "POST",
40         headers: {
41             'Accept': 'application/json',
42             'Content-Type': 'application/json'
43         },
44         body: JSON.stringify({
45             name: "John",
46             age: 60
47         })
48     });
49     let content = await response.json();
50     console.log(content);
51 });
52
53 window.onload = sendMsg;
```

48

## ตัวอย่างใน server.js สำหรับ fetch

```
20 const express = require('express');
21 const app = express();
22 const hostname = 'localhost';
23 const port = 3000;
24 const bodyParser = require('body-parser');
25
26 app.use(express.static(__dirname));
27 app.use(bodyParser.json());
28 app.use(bodyParser.urlencoded({extended: false}));
29
30 app.post('/message', (req, res) => {
31   console.log(req.body)
32   const outMsg = req.body
33   outMsg["lastname"] = "Doe";
34   console.log(outMsg);
35   res.json(outMsg);
36 })
37
38 app.listen(port, hostname, () => {
39   console.log(`Server running at http://\${hostname}:\${port}`);
40 });
```



## ตัวอย่าง ผลที่ได้

### Output: server

```
Server running at http://localhost:3000/  
{ name: 'John', age: 60 }  
{ name: 'John', age: 60, lastname: 'Doe' }
```

### Output: html

```
Navigated to http://localhost:3000/  
▶ Fetch finished loading: POST "http://localhost:3000/message".  
▶ {name: "John", age: 60, lastname: "Doe"}  
▶ XHR finished loading: GET "chrome-extension://mgijmajocgfcbeboacabfgobmjgjcoja/content.min.css".
```

## Assignment 10:

---

- Download assignment10.zip file
- เราจะทำการสร้าง chatroom ง่าย ๆ โดยใช้ node.js เริ่มจาก
  - ให้ user กรอกชื่อ
  - อ่าน json data จาก log.json แสดงใน chatroom
  - ให้ user พิมพ์ข้อความที่ต้องการส่ง
  - ทำการ save ข้อความที่ user พิมพ์ใน chatroom ของเราลงใน file ที่มีชื่อว่า log.json และทำการอ่าน file นั้นทุก ๆ 3 วินาทีเพื่อ update ข้อความใน chatroom
- ใช้ Async และ Await
- แก้ไขใน chatter.js กับ server.js
- ส่งด้วยนะคะ

## Week 10: Classroom game

---

- ทำ classroom game ใน LEB2 ก่อนสัปดาห์หน้า

Username:

Welcome,