Lab 5 - Express

Node as a server

จาก Lab ที่ผ่านมา เราสามารถใช้ Nodejs ในการทำงานตั้งแต่เรื่องพื้นฐาน การเขียนโปรแกรมผ่าน Socket มาถึงบทนี้จะกล่าวถึง Nodejs กับ library HTTP ที่สามารถเขียนให้ตัวเองเป็น web server ง่าย ๆ ดังตัวอย่างนี้

index.js

```
var http = require('http');
var server = http.createServer(function(req, res){
   res.writeHead(200, {'Content-type': 'text/plain'});
   res.end('Hello world\n');
});

server.listen(8000);
console.log('Server is ready!');
```

โปรแกรมมีการเปิด port 8000 รอรับการเชื่อมต่อ และ ถ้ามี client ร้องขอหน้า page ที่ server ก็จะตอบ status 200 พร้อมกับข้อความ Hello world ในลักษณะของ plain text ธรรมดา
ทดลองเปิด http://locahost:8000/

Express

การจัดการกับ client ที่ร้องขอหน้า web โดยใช้ library HTTP อย่างเดียว ไม่สะดวก และมีความซับซ้อน จึงได้มี Express framework ช่วยทำให้ Nodejs เป็น web server ที่จัดการได้ง่ายขึ้นดังนี้

ก่อนใช้ต้องมีการติดตั้ง express ก่อน ด้วยคำสั่ง npm i express --save หรือ yarn add express (หากยังไม่มีไฟล์ package.json และ node_modules ให้ใช้คำสั่ง npm init --y เพื่อสร้าง npm project ขึ้นมาก่อน)

ตัวอย่างนี้ มีการทำงานเหมือนกับตัวอย่างข้างบน แต่ code จะสั้นกว่าและ สามารถจัดการ route URL ได้ ง่ายกว่า ดังนี้

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   res.send('Hello world')
});

app.listen(8000);
```

สร้าง express instance เพื่อเอามาสร้าง app instance จากนั้น สั่ง app.get (เป็นการรอรับ GET method) และส่งข้อมูลกลับไปให้ client โดยคำสั่ง res.send ด้วยข้อความ Hello world
ทดลองเปิด http://locahost:8000/

Query String (req.query)

Query string คือ ส่วนของค่าใน URL ที่มีข้อมูลต่อท้าย ผ่าน method HTTP GET เช่น http://localhost:8000/greeting?str1=Hello&str2=world จะใช้เครื่องหมาย ? ในการแบ่งระหว่างข้อมูลกับ URL และข้อมูลที่อยู่ในรูปแบบของ key=value แบ่งด้วย &

Nodejs สามารถอ่านค่าตัวแปรจาก query โดยตรงได้ ผ่าน req.query และตามด้วยชื่อของ key ที่ระบุใน Query string ดังตัวอย่าง (ในตัวอย่างมีการใช้เครื่องหมาย ` (backquote) ใช้ในการ render ตัวแปรที่อยู่รวมในข้อความ ในลักษณะของ string template อีกทีด้วย)

```
var express = require('express');
var app = express();

app.get('greeting', (req, res) => {
   let greetText = req.query.str1 + " " + req.query.str2
   res.send(`<html><h1 style="align:center;">${greetText}</h1></body></html>`)
})

app.listen(8000);
```

ทดลองเปิด http://localhost:8000/greeting?str1=Hello&str2=world ให้สังเกตการทำงานของ req.query.str1

หมายเหตุ: res.write() + res.end() ส่งเป็น plain text (ไม่สนับสนุน html) ใช้ res.send() ส่งเป็น html เสมือนใช้ write() + end()

Params (req.params)

ข้อมูลอีกประเภทที่ส่งผ่าน URL ในลักษณะ parameters เช่น http://localhost:8000/greeting/Hello/world

```
var express = require('express');
var app = express();

app.get('/greeting/:str1/:str2', (req, res) => {
  console.log(req)
  let greetText = req.params.str1 + " " + req.params.str2
  res.send(`<html><h1 style="align:center;">Hey: ${greetText}</h1></body></html>`)
})

app.listen(8000);
```

ทดลองเปิด http://localhost:8000/greeting/Hello/world ให้สังเกตการทำงานของ req.params.str1

Body parser

การรับข้อมูลจาก web form ไม่ว่าจะเป็น method GET หรือ POST สามารถอ่านได้จาก body parser ดังนี้

จำเป็นต้องติดตั้ง body-parser ก่อน ดังนี้ npm i body-parser --save ตัว index.js ทำหน้าที่เป็น web back-end ไว้รับข้อมูลจาก form.html ที่ส่งมา

มีการใช้ app.use หมายถึง ไม่ว่า client ร้องขอ url แบบใดก็ตาม app.use จะถูกเรียกให้ทำงาน โดยทำหน้าที่กำหนด web root directory สำหรับ static content ได้แก่ html, images, css สามารถกำหนดผ่าน express.static ได้

app.post ใช้สำหรับการรับการร้องขอแบบ HTTP Post และ มีการใช้ middleware body-parser เพื่อใช้ในการดึงข้อมูลจาก form ที่ส่งผ่านมา

index.js

```
var express = require('express'),
   app = express(),
   bodyParser = require('body-parser');

var urlencodedParser = bodyParser.urlencoded({ extended: false });
app.use(express.static(__dirname + '/public'));

app.post('/add', urlencodedParser, function(req, res){
   var result = parseInt(req.body.a) + parseInt(req.body.b);
   res.send('Result = ' + result);
});

app.listen(8000);
```

./public/form.html

```
<html>
<head>
   <title>Adding Form</title>
</head>
<body>
<form action="/add" method="post">
   A: <input type="number" name="a"/><br/>
   B: <input type="number" name="b"/><br/>
   <button type="submit">Add</button>
</form>
</body>
</html>
```

ทดลองเปิด http://localhost:8000/form.html และลองใส่ตัวเลขเพื่อบวกหาผลลัพธ์

ในกรณีที่ใช้ axios ส่งผ่าน json จะกำหนด `app.use(bodyParser.json())` ซึ่งสามารถใช้รวมกับ body parser แบบนี้ได้ `app.use(bodyParser.urlencoded({ extended: false }))`

Cookie

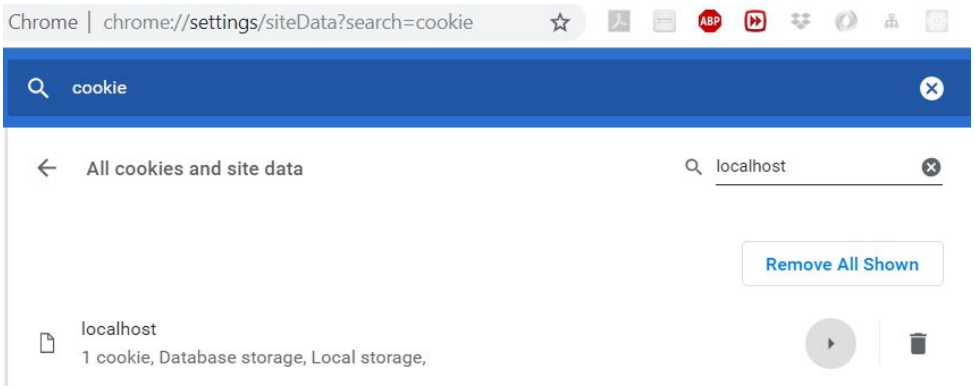
Cookie เป็นเครื่องมือ ที่ใช้ในการเก็บข้อมูล ที่ฝั่งผู้ใช้ ผ่าน middleware ที่ชื่อ cookie-parser ก่อนอื่นต้องติดตั้ง โดย npm i cookie-parser --save

index.js

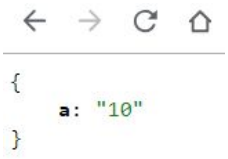
```
var express = require('express')
var app = express()
var cookieParser = require('cookie-parser')
app.use(cookieParser('keyboard cat')) //'keyboard cat' is a secret key to sign cookie (prevent cookie tamper)
app.get('/ck_get', function(req, res) {
   res.send(req.cookies)
})

app.get('/ck_set', function(req, res) {
   res.cookie('a', 10)
   res.send('ok')
})

app.listen(8000)
```

หากใช้ Google Chrome สามารถตรวจสอบ cookie ที่มีในเครื่อง โดยเข้าที่ chrome://settings/siteData?search=cookie ให้สังเกต IP ของเครื่องที่ทำการทดลอง

จากนั้นทดสอบเปิด http://localhost:8000/ck_set ค้นหา localhost ที่ ช่องค้นหาด้านบนซ้าย จะพบ cookie a



การอ่าน cookie ให้เปิด http://localhost:8000/ck_get ได้ผลลัพธ์



จะเห็นได้ว่า สามารถเก็บข้อมูล ลงใน cookie ที่อยู่ใน web browser ได้

Session (req.session)

Session เป็นเครื่องมือ ที่ใช้ในการเก็บข้อมูล ที่ฝั่งเครื่องแม่ข่าย ผ่าน middleware ที่ชื่อ session ก่อนอื่นต้องติดตั้งโดย npm i express-session --save

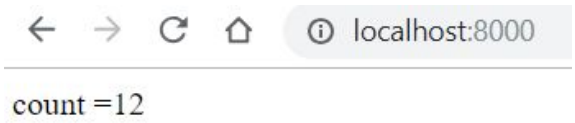
Session สามารถเก็บได้ 4 รูปแบบ คือ 1) application memory 2) cookie 3) memory cache 4) memory database แบบแรก application memory นั้น ใช้สำหรับ development mode เนื่องจากหากโปรแกรม crash ข้อมูล session จะหายหมด ส่วนแบบที่ 2 จะสร้าง session เป็น cookie ส่งไปเก็บไว้ที่ client ในช่วงเวลาที่กำหนด (ไม่ค่อยปลอดภัย) โดย session-express จะจัดการเรื่องการเข้ารหัส แต่ถ้าผู้ไม่หวังดี รู้ secret ก็อาจจะแกะ cookie ได้ แบบที่ 3 และ 4 จะใช้ cookie เก็บค่าเฉพาะ session Id แล้วส่งไปให้ client เก็บเฉพาะ sessionId เท่านั้น ถือว่าปลอดภัยกว่าแบบที่ 2

Reference: https://nodewebapps.com/2017/06/18/how-do-nodejs-sessions-work/

index.js

```
var express = require('express')
var app = express()
var session = require('express-session')

                          // sign cookie (for a session)
app.use(session({ secret: 'keyboard cat', cookie: { maxAge: 60000 },
   resave : false, saveUninitialized: false }))
   // resave => Forces the session to be saved back to the session store, even if the session was never modified
   // saveUninitialized => the cookie will not be set on a response with an uninitialized session

app.use(function(req, res, next) {
   var sess = req.session
   if (sess.views) {
       sess.views++
   } else {
       sess.views = 1
   }
   console.log(sess.views)
   next();
})
app.get('/',function(req, res) {
   res.send('count =' + req.session.views)
})
app.listen(8000)
```

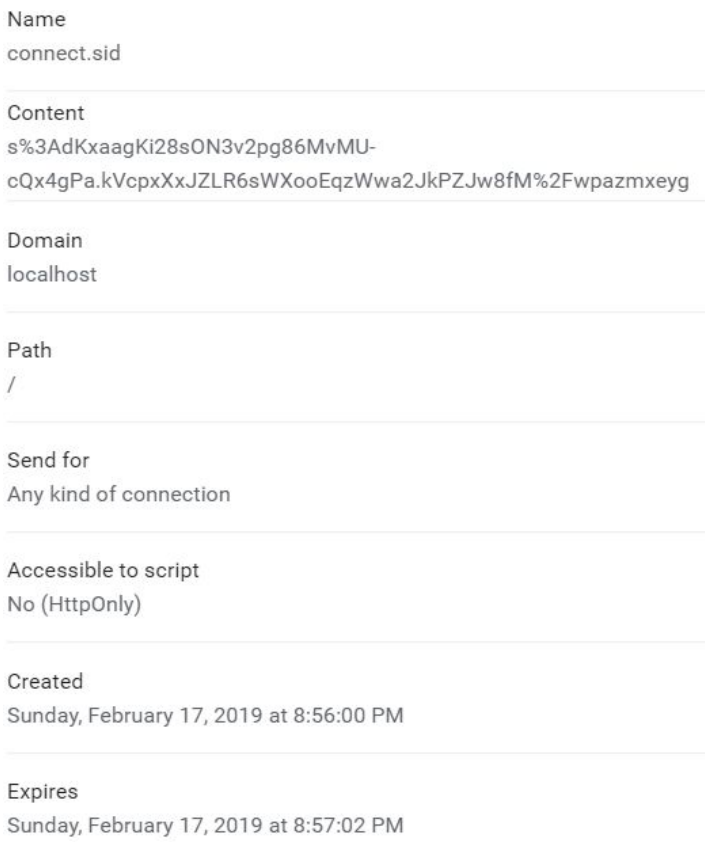
ทดลองเปิด http://localhost:8000/ แล้วกด Refresh สังเกตการเปลี่ยนแปลงของตัวแปร count



ภายใน app.use() มีการใช้ next() เพื่อโยน request ที่รับมาให้กับ middleware ตัวต่อไป ก็คือ app.get() หากทดลอง comment next() ออกก็จะพบว่า หลังจากที่ client request เว็บก็จะค้างไม่มีการ return ค่าของ count จาก app.get ออกไป

สังเกตที่ตัวแปร views ที่สร้างมาใหม่ ภายใต้ req.session กำหนดให้เพิ่มค่าทุกครั้งที่ถูกเรียกใช้ หน้าเว็บสามารถตั้งชื่อเป็นตัวแปรอะไรก็ได้ เช่น req.session.user = “john”, req.session.password = “mypass” เป็นต้น)

ตัวอย่าง session Id ที่เก็บที่ client (เรียกดูได้จาก cookie ของ browser)



หลังจากเรียก session แล้ว ก็จะเห็นว่ามีการใช้ cookie เพื่อเก็บเฉพาะ Id แต่ตัวข้อมูลจะเก็บใน application memory หากต้องการเก็บใน memory cache อย่างพวก MemCache หรือ Redis จะต้องมีการ configure session เพิ่มเติม

Template engine (ejs)

ejs เป็นเท็มเพล็ต ที่ใช้ใน ดึงข้อมูลจากฝั่ง backend มาแสดงที่หน้าเว็บไซต์ ผ่าน middleware ที่ชื่อ ejs ก่อนอื่นต้องติดตั้ง โดย npm i ejs --save การทำงาน ฝั่ง backend จะส่งข้อมูลผ่าน res.render( )

ข้อมูลเพิ่มเติมเกี่ยวกับ ejs สามารถอ่านได้ที่ http://www.embeddedjs.com/getting_started.html

index.js

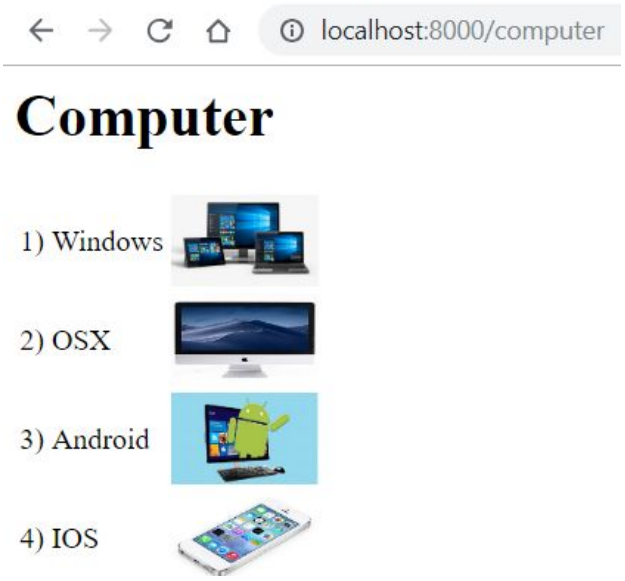
```
var express = require('express')
var app = express()

app.set('views', './views')
app.set('view engine', 'ejs')

app.get('/fruit', function(req, res){
   res.render('fruit', {fruits: ['banana', 'apple'] , foo: 'bar'})
})
app.listen(8000)

/views/page.ejs
<h1> Fruits  <%= foo %> </h1>

<ul>
  <% fruits.forEach(function(fruit){ %>
    <li><%= fruit %></li>
    <% }); %>
</ul>
```

views/fruit.ejs

```
<html>
<ul>
  <% fruits.forEach(function(fruit){ %>
    <li><%= fruit %></li>
    <% }); %>
</ul>
</html>
```

ตัวอย่างการทำงาน เมื่อทดลองเปิดหน้า http://localhost:8000/fruit

1. จงพัฒนา web โดยใช้ express และ ejs template engine ในการแสดงผล ข้อมูล และ รูปภาพต่อไปนี้ (กำหนดให้รูปทุกรูปต้องอยู่ folder images แยกกับ file.ejs และ ต้องมีการใช้ loop) โดยเรียกจาก http://localhost:8000/computer



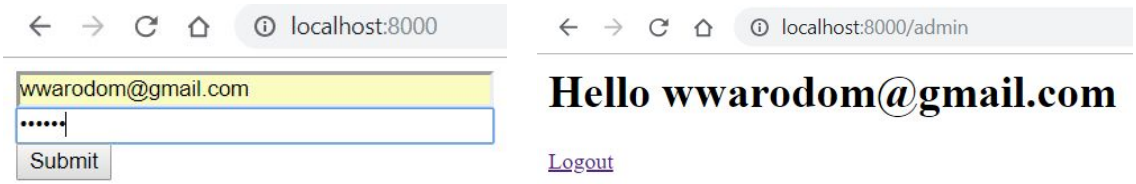
Ans:

2. จงพัฒนา web authentication (login) โดยใช้ session body-parser และ ejs template โดยกำหนด route ดังนี้

- http://localhost:8000/ ให้เปิดไฟล์ Index.html และส่งแบบฟอร์มที่ กำหนดให้กรอก email และ password หากไม่ป้อน email หรือ password !== 240311 จะส่งไปหน้า /admin และบังคับให้ Login first

- หากป้อน email และ password เป็น 240311 จะแสดง ข้อความ Hello ตามด้วย Email พร้อม Link Logout อยู่ด้านล่าง

- ในกรณีที่กด Logout แล้ว หากเข้าที่ /admin โดยไม่ login ก่อน จะต้องไม่แสดงข้อความ Hello แต่ให้แสดงข้อความ Please login first



ตกแต่งให้สวยงามตามความถนัด

Ans: