



# พัฒนา Web Application ด้วย JavaScript และ Node.js

- เรียนรู้พื้นฐานและเทคนิคการพัฒนาเว็บไซต์ด้วยภาษา JavaScript อย่างละเอียด
- อธิบายการใช้งาน JavaScript ร่วมกับ Express และ MongoDB
- แนะนำการใช้งาน JavaScript/TypeScript บน Node และ Deno
- แนะนำเผยแพร่บริการ Web API แบบสาธารณะด้วย Heroku ร่วมกับ GitHub
- มีโค้ดตัวอย่างและคำอธิบายอย่างละเอียด อ่านเข้าใจง่าย
- อธิบายเป็นขั้นตอน เหมาะสำหรับผู้เริ่มต้น



ไฟล์ตัวอย่างภายในเล่ม  
<https://serazu.com/>  
9786164873582

ศุภชัย สมพานิช

# สารบัญ

<b>บทที่ 1</b>	<b>JavaScript World 101 .....</b>	<b>1</b>
	รูปแบบการนำเสนอสคริปต์ในหนังสือเล่มนี้ .....	1
	JavaScript คืออะไร .....	2
	การดาวน์โหลดและติดตั้ง PowerShell .....	3
	การดาวน์โหลดและติดตั้ง Windows Terminal .....	4
	การดาวน์โหลดและติดตั้งโปรแกรม Visual Studio Code .....	6
	การติดตั้งฟอนต์ Cascadia Code สำหรับเขียนโปรแกรม .....	7
	การกำหนดให้โปรแกรม VS Code ใช้ฟอนต์ Cascadia Code .....	9
<b>บทที่ 2</b>	<b>พื้นฐานการใช้งานภาษา JavaScript .....</b>	<b>11</b>
	พื้นฐานการสร้างโปรเจกต์ JavaScript ในโปรแกรม Visual Studio Code .....	11
	พื้นฐานการใช้งาน JavaScript ร่วมกับ HTML .....	13
	การฝัง JavaScript กับอีลีเมนต์ตัวใดตัวหนึ่ง .....	14
	การเขียน JavaScript ระหว่างอีลีเมนต์ <code>&lt;script&gt;...&lt;/script&gt;</code> .....	15
	การแยก JavaScript ออกเป็นไฟล์ต่างหาก .....	16
	การใช้งาน Console ของบราวเซอร์ .....	18
	ทำความเข้าใจกับตัวแปร (Variable) ใน JavaScript (คำสั่ง var และ let) .....	19
	ค่าคงที่ (คำสั่ง const) .....	22
	ฟังก์ชัน (Function) และ Arrow Function .....	24
	การกำหนดค่าเริ่มต้นของพารามิเตอร์ (Default Parameter) .....	27
	ทำงานกับข้อความด้วย Template Literals (หรือ Template strings) .....	31
	การสร้างพารามิเตอร์แบบรับได้หลายค่าด้วยตัวดำเนินการ .....	33
	การแทรกอีลีเมนต์ HTML ด้วย JavaScript .....	35

<b>บทที่ 3</b>	<b>ทำงานกับโครงสร้างข้อมูลอาร์เรย์ (Array)</b> .....	<b>37</b>
	พื้นฐานการสร้างอาร์เรย์ขึ้นมาใช้งาน.....	37
	การสร้างอาร์เรย์จากฟังก์ชัน from().....	39
	การอ่านลำดับสมาชิกในอาร์เรย์ด้วยฟังก์ชัน keys().....	40
	การเพิ่มข้อมูลในอาร์เรย์ด้วยตัวดำเนินการ ..	41
	พื้นฐานการทำงานกับโครงสร้างข้อมูลแบบอาร์เรย์.....	43
	การกรองข้อมูลด้วยฟังก์ชัน filter().....	43
	การอ่านข้อมูลในอาร์เรย์ด้วยฟังก์ชัน map().....	44
	การวนลูปอ่านข้อมูลในอาร์เรย์ด้วยฟังก์ชัน forEach().....	45
	การค้นหาข้อมูลในอาร์เรย์ด้วยฟังก์ชัน find().....	45
	การเพิ่มข้อมูลต่อท้ายเข้าไปในอาร์เรย์ด้วยฟังก์ชัน push().....	46
	การเพิ่มข้อมูลลำดับแรกเข้าไปในอาร์เรย์ด้วยฟังก์ชัน unshift().....	47
	การถอดข้อมูลล่าสุดออกจากอาร์เรย์ด้วยฟังก์ชัน pop().....	47
	การถอดข้อมูลลำดับแรกออกจากอาร์เรย์ด้วยฟังก์ชัน shift().....	48
	การถอดข้อมูลออกจากอาร์เรย์ด้วยฟังก์ชัน splice().....	49
	การตัด/แบ่งข้อมูลด้วยฟังก์ชัน slice().....	51
	การเรียงลำดับข้อมูลในอาร์เรย์ด้วยฟังก์ชัน sort().....	54
	การตรวจสอบข้อมูลในอาร์เรย์ด้วยฟังก์ชัน includes().....	55
<b>บทที่ 4</b>	<b>คำสั่งตรวจสอบและการวนลูป</b> .....	<b>57</b>
	การตรวจสอบเงื่อนไขด้วยคำสั่ง if.....	57
	การตรวจสอบหลายเงื่อนไขด้วยคำสั่ง switch.....	59
	การวนลูปแบบ for.....	61
	การวนลูปแบบ for in.....	62
	การวนลูปแบบ for of.....	63
	การวนลูปแบบ while.....	65
	การวนลูปแบบ do while.....	66

<b>บทที่ 5</b>	<b>การเขียนโปรแกรมเชิงวัตถุใน JavaScript.....</b>	<b>69</b>
	พื้นฐานการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming - OOP).....	69
	การสร้างออบเจกต์เก็บข้อมูลตามโครงสร้างที่เรากำหนด .....	73
	การสร้างเมธอด (หรือฟังก์ชัน) ในคลาส .....	74
	การสร้างเมธอดขอบเขตแบบ Private.....	75
	การแปลงข้อมูลในออบเจกต์เป็นอาร์เรย์ด้วยฟังก์ชัน entries() และฟังก์ชัน values() .....	78
<b>บทที่ 6</b>	<b>พื้นฐานการใช้งาน JavaScript ฟัง Server ด้วย Node.....</b>	<b>81</b>
	การดาวน์โหลดและติดตั้ง Node .....	81
	พื้นฐานการใช้งานภาษา JavaScript บน Node .....	83
	Express คืออะไร?.....	84
	การดาวน์โหลดและติดตั้ง Express .....	85
	พื้นฐานการสร้างโปรเจกต์ Express.....	86
	การทดสอบรันและหยุดโปรเจกต์ Express.....	91
	วิธีการจัดเก็บโปรเจกต์ Express.....	93
	การเปิดโปรเจกต์ Express แบบปกติ .....	94
	วิธีการ Restart โปรเจกต์ Express โดยอัตโนมัติ .....	96
	ระบบเส้นทาง (Route) ของ Express ในขั้นต้น .....	101
	พื้นฐานการจัดการพาธอื่นๆ ที่ไม่มี (404 page not found).....	102
	การใช้งาน JavaScript ร่วมกับ Express เบื้องต้น .....	103
	การใช้งานไฟล์ Static ร่วมกับ Express (Static File).....	105
<b>บทที่ 7</b>	<b>พื้นฐานการใช้งานระบบฐานข้อมูล MongoDB .....</b>	<b>109</b>
	ระบบฐานข้อมูล NoSQL แบบ MongoDB คืออะไร.....	109
	การดาวน์โหลดและติดตั้งฐานข้อมูล MongoDB .....	110
	พื้นฐานการเชื่อมต่อกับฐานข้อมูล MongoDB ด้วย MongoDB Compass.....	112
	การสร้างฐานข้อมูล MongoDB ใหม่.....	113
	การจัดการข้อมูลในฐานข้อมูล MongoDB เบื้องต้น .....	118
	การ Export ข้อมูลออกจากฐานข้อมูล MongoDB.....	121
	การ Import ข้อมูลเข้ามาในระบบฐานข้อมูล MongoDB.....	124

<b>บทที่ 8</b>	<b>การสร้างบริการ CRUD ด้วย Express ร่วมกับฐานข้อมูล MongoDB .....</b>	<b>125</b>
	การเชื่อมต่อ Express กับฐานข้อมูล MongoDB (Read).....	125
	การทดสอบการทำงานของ Web API ด้วย REST Client.....	139
	การสร้างบริการเพิ่ม, แก้ไข และลบข้อมูล (Create, Update และ Delete) .....	143
<b>บทที่ 9</b>	<b>การสร้างส่วนแสดงผลด้วย Angular .....</b>	<b>155</b>
	การดาวน์โหลดและติดตั้ง Angular.....	156
	การสร้างโปรเจกต์ Angular แรก.....	157
	การทดสอบและหยุดรันโปรเจกต์ Angular.....	159
	การติดตั้งส่วนขยาย (Extension) สำหรับ Angular ให้กับโปรแกรม	
	Visual Studio Code.....	161
	หลักการสร้างส่วนแสดงผลด้วย Angular.....	161
	การสร้างส่วนแสดงผลสำหรับแสดงในหน้าจอทุกขนาดด้วย Bootstrap	
	(Responsive Design) .....	168
	การติดตั้ง Bootstrap ให้กับโปรเจกต์ Angular.....	168
	ทำความรู้จักกับตัวบรรจุ container ของ Bootstrap .....	172
	ทำความรู้จักกับระบบ Grid ของ Bootstrap .....	174
	ทำความรู้จักกับขนาดของ Grid .....	176
<b>บทที่ 10</b>	<b>การใช้งาน Express ร่วมกับ Angular (MEAN Stack) .....</b>	<b>181</b>
	การเพิ่ม Component สำหรับเมนูหลัก.....	182
	การสร้างแถบ Footer .....	188
	การสร้างส่วนแสดงผลสำหรับงาน CRUD ด้วย Angular.....	190
	การสร้างบริการส่วนกลาง (Service) ใน Angular .....	190
	การสร้างส่วนแสดงผลสำหรับงาน CRUD .....	195
	การกำหนดเส้นทาง (Route) สำหรับงาน CRUD.....	204
	การทดสอบการทำงานของ CRUD ของ MEAN Stack .....	206
	อธิบายการทำงานของสคริปต์การทำงาน CRUD .....	212
<b>บทที่ 11</b>	<b>การสร้าง Express แบบมีส่วนแสดงผลด้วย Pug .....</b>	<b>227</b>
	การสร้างโปรเจกต์ Express ด้วย Express Generator.....	228
	หลักการสร้างส่วนแสดงผลด้วย Pug .....	233

<b>บทที่ 12</b>	<b>การสร้าง CRUD ด้วย Express ร่วมกับ Pug .....</b>	<b>237</b>
	การสร้างโปรเจกต์ Express และ Pug ด้วย Express Generator .....	237
	การเขียนสคริปต์สำหรับโปรเจกต์ CRUD และสร้างส่วนแสดงผลด้วย Pug .....	241
	การสร้างส่วนแสดงผลด้วย Pug.....	247
	การทำงาน CRUD ของ Express ร่วมกับ Pug.....	253
	อธิบายการทำงานของสคริปต์ของโปรเจกต์ Express และ Pug ด้วย Express Generator .....	258
<b>บทที่ 13</b>	<b>การใช้งาน JavaScript/TypeScript บนแพลตฟอร์ม Deno.....</b>	<b>267</b>
	การดาวน์โหลดและติดตั้ง Deno.....	268
	พื้นฐานการรัน JavaScript บน Deno .....	269
	การติดตั้งส่วนขยายสำหรับพัฒนาแอปบน Deno.....	270
	พื้นฐานการสร้าง Web API บน Deno ด้วย OAK Framework .....	272
	การระบุเวอร์ชันให้กับ OAK.....	275
	การจัดการเส้นทาง (Route) ของ OAK.....	277
	พื้นฐานการสร้างพารอร์รับงาน CRUD .....	279
<b>บทที่ 14</b>	<b>การใช้งานฐานข้อมูล MongoDB ด้วยบริการคลาวด์ของ MongoDB Atlas ....</b>	<b>287</b>
	การเปิดใช้บริการคลาวด์ของ MongoDB Atlas .....	288
	การเชื่อมต่อโปรเจกต์ Express เข้ากับบริการ MongoDB Atlas แบบ localhost.....	298
<b>บทที่ 15</b>	<b>การเปิดบริการ Web API สู่ภายนอกด้วย Heroku .....</b>	<b>301</b>
	Git คืออะไร? .....	301
	การดาวน์โหลดและติดตั้งโปรแกรม Git.....	302
	การจัดเก็บโปรเจกต์ใน GitHub.....	307
	การเตรียมโปรเจกต์ Express ให้พร้อมใช้งานบริการของ Heroku .....	310
	การอัปโหลด (push) โปรเจกต์ Express ไปเก็บใน GitHub .....	313
	การใช้บริการ Heroku เบื้องต้น .....	319
	การอัปโหลดโปรเจกต์ Express เข้าสู่ Heroku ด้วยวิธี Heroku CLI.....	322
	การอัปโหลดโปรเจกต์ Express เข้าสู่ Heroku ด้วยวิธีดึงโปรเจกต์จาก GitHub .....	328





# บทที่ 4

## คำสั่งตรวจสอบและการวนลูป

การศึกษาเขียนโปรแกรมของทุกๆ ภาษา ก็จะมีเนื้อหาพื้นฐานหัวข้อแรกๆ ที่มีใหม่ต้องมาทำความรู้จักและศึกษาก่อนเลย นั่นคือ การตรวจสอบเงื่อนไขเพื่อสร้างทางเลือกและการสั่งให้การทำงานซ้ำ หรือที่เรียกว่า **การวนลูป**

### การตรวจสอบเงื่อนไขด้วยคำสั่ง if

การตรวจสอบเงื่อนไขในระดับพื้นฐานลำดับแรกคือ การใช้คำสั่ง **if** มีความหมายว่า "ถ้าเงื่อนไขถูกต้อง ให้ทำอะไร?" เช่น สร้างตัวแปรที่ชื่อว่า x เก็บค่าเริ่มต้นเป็นตัวเลข 10 (`let x = 10`) จากนั้นตรวจสอบว่า ถ้าตัวแปร x มีค่าน้อยกว่าหรือเท่ากับ 100 สั่งให้ตัวแปร result เก็บข้อความ "เงื่อนไขถูกต้อง!!!" ดังสคริปต์ต่อไปนี้

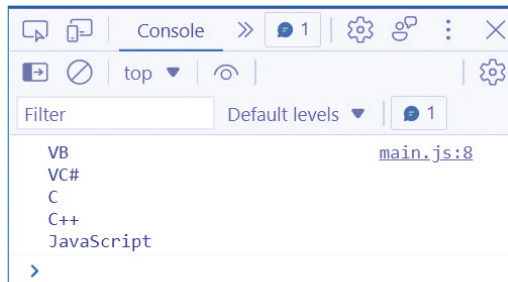
```
js/main.js
```

```
let x = 10;
let result = "";
if (x <= 100){
  result = "เงื่อนไขถูกต้อง!!!";
}
```

```
console.log(result);
```



การวนลูปแต่ละรอบ กำหนดให้อ่านข้อมูลในอาร์เรย์ data ตามตำแหน่งปัจจุบันของตัวแปร i จากนั้นกำหนดให้สะสมค่าในตัวแปร result ร่วมกับตัวอักษรควบคุม "\n" หมายถึง การขึ้นบรรทัดใหม่ ผลการทำงานแสดงดังรูปที่ 4-6



รูปที่ 4-6 ผลการอ่านข้อมูลที่ได้จากตัวแปร result

## การวนลูปแบบ for in

ในกรณีที่เรากำลังทำงานกับข้อมูลออบเจกต์ที่มีโครงสร้างซับซ้อน เราต้องใช้รูปแบบ **for in** เช่น ค่าคงที่ data เก็บข้อมูลที่มีโครงสร้างภายใน ประกอบด้วย 3 ส่วนคือ fullname, Address และ age เราสามารถใช้รูปแบบ for in ทำหน้าที่อ่านข้อมูล ดังสคริปต์ต่อไปนี้

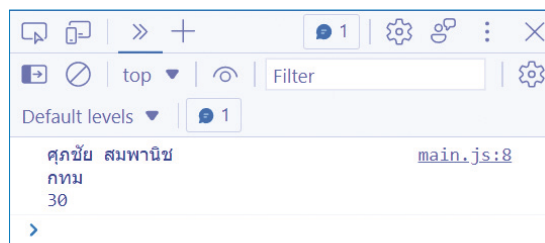
js/main.js

```
const data = { fullname: "ศุภชัย สมพานิช", Address: "กทม", age: 30 };

let result = "";
for (let i in data) {
  result += data[i] + "\n";
}

console.log(result);
```

เมื่อรันสคริปต์จะได้ผลลัพธ์ดังนี้



รูปที่ 4-7 ผลการอ่านข้อมูลด้วยลูปแบบ for in

# การแปลงข้อมูลในออบเจ็กต์เป็นอาร์เรย์ด้วย ฟังก์ชัน `entries()` และฟังก์ชัน `values()`

ข้อมูลที่เราเก็บอยู่ในคุณสมบัติของออบเจ็กต์ สามารถใช้ฟังก์ชัน `entries()` เข้ามาทำหน้าที่ช่วยแปลงให้เป็นอาร์เรย์ได้อีกด้วย มีลักษณะโครงสร้างข้อมูลเป็นแบบ Key/Value

คลาส `Customer` เก็บข้อมูล 2 ส่วนคือ รหัส (`id`) และชื่อ-สกุล (`fullname`) ดังสคริปต์ต่อไปนี้

```
\\js\customer.js
```

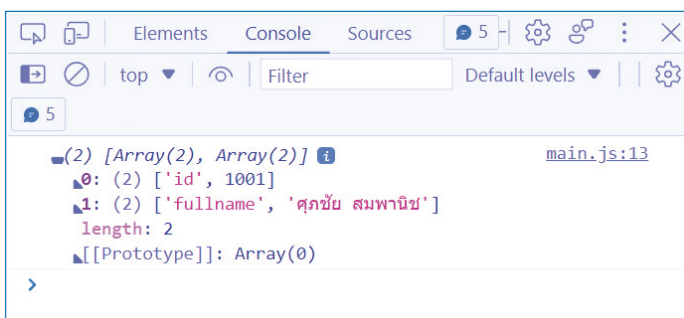
```
class Customer {
  constructor(_id, _fullname) {
    this.id = _id;
    this.fullname = _fullname;
  }

  walk() {
    console.log("ลูกค้ากำลังเดิน");
  }
}

let c = new Customer(1001, "ศุภชัย สมพานิช");
console.log(Object.entries(c));
```

เมื่อใช้ฟังก์ชัน `entries()` เข้ามาทำหน้าที่แปลงตัวแปรออบเจ็กต์ `Customer` ที่ชื่อว่า `c` ส่งผลให้ได้อาร์เรย์ที่มีโครงสร้าง ประกอบไปด้วย

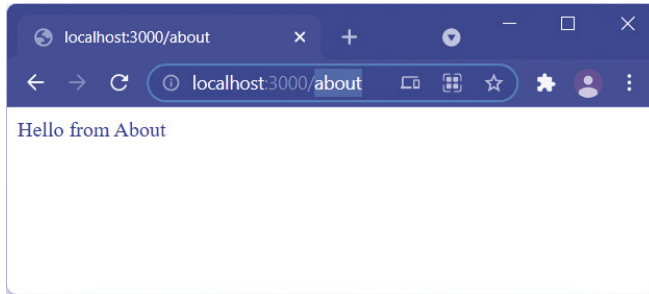
- **id** เก็บค่า 1001
- **fullname** เก็บค่า "ศุภชัย สมพานิช"



รูปที่ 5-8 ผลการแปลงข้อมูลด้วยฟังก์ชัน `entries()`

```
app.listen(3000, () => {  
  console.log('Start server at port 3000')  
})
```

จากสคริปต์ข้างต้นมีความหมายว่า ตอนนี้เราเปิดให้บริการ 2 พาทแล้วคือ พาท / กับ /About ลองใช้เบราว์เซอร์เข้าที่พาทใหม่ /about ผลการทำงานแสดงดังรูปที่ 6-30



รูปที่ 6-30 ผลการทำงานของพาท /About

## พื้นฐานการจัดการพาทอื่นๆ ที่ไม่มี (404 page not found)

ความรู้พื้นฐานสำหรับการจัดการเส้นทาง หรือพาทอีกอย่างหนึ่งก็คือ ในกรณีที่มีการร้องขอข้อมูลมาที่พาทที่ไม่มีอยู่จริงใน Web API ของเราคือ ข้อผิดพลาดที่เรียกว่า **404 page not found**

วิธีการแก้ไขเบื้องต้นก็คือ ให้ใช้เครื่องหมาย \* เพื่อระบุครอบคลุมพาทอื่นๆ ที่เราไม่ได้กำหนดไว้ ดังสคริปต์ต่อไปนี้

### index.js

```
const express = require('express')  
const app = express()  
  
app.get('/', (req, res) => {  
  res.send('Hello World!!!')  
})  
  
app.get('/About', (req, res) => {  
  res.send('Hello from About')  
});
```

# การสร้างบริการเพิ่ม, แก้ไข และลบข้อมูล (Create, Update และ Delete)

การสร้างบริการประเภทอ่านข้อมูลแบบไม่มีเงื่อนไข ถือเป็นการทำงาน 1 ใน 4 อย่างของ CRUD ผู้เขียนจะนำโปรเจกต์ที่แล้วมาทำต่อให้ครบ

**ตัวอย่างที่ 8-2** การสร้างบริการเพิ่ม, แก้ไข และลบข้อมูล (Create, Update และ Delete) มีขั้นตอนดังต่อไปนี้

1. ที่ส่วนของการทำงาน `customer.controller.js` เขียนสคริปต์เพิ่มการทำงานดังต่อไปนี้

## สคริปต์ Express ที่ 8-2 การสร้างบริการเพิ่ม, แก้ไข และลบข้อมูล (Create, Update และ Delete) (`controllers\customer.controller.js`)

```
const customer = require('../models/customer.js');

exports.index = (req, res) => {
  res.send('<h1>Customer Apps</h1><hr /><a href="/api/customer">รายชื่อลูกค้า</a>');
}

exports.create = (req, res) => {
  const c = new customer(req.body);

  c.save()
    .then(data => {
      res.json(data)
    }).catch(err => {
      return res.status(500).json({
        msg: "ไม่สามารถเพิ่มข้อมูลลูกค้าได้ เนื่องจาก : " + err.message
      });
    });
};

exports.findAll = (req, res) => {
  customer.find()
    .then(data => {
      res.json(data);
    }).catch(err => {
      res.status(500).send({
        msg: err.message
      });
    });
};
};
```

## อธิบายการทำงานของสคริปต์

1. การเพิ่มข้อมูล ให้สร้างฟังก์ชันที่ชื่อว่า `create` (`exports.create`) ขึ้นมาก่อน การทำงานภายในมี 2 ขั้นตอนย่อยคือ
  - สร้างค่าคงที่ที่ชื่อว่า `c` (`const c`) ขึ้นมาก่อน กำหนดให้อ่านข้อมูลลูกค้าคนใหม่มาเก็บไว้ (`new customer(req.body)`)
  - ใช้ฟังก์ชัน `save()` ทำหน้าที่บันทึกข้อมูลลูกค้าคนปัจจุบันเข้าสู่ฐานข้อมูล MongoDB

### \controllers\customer.controller.js

```
exports.create = (req, res) => {
  const c = new customer(req.body);

  c.save()
    .then(data => {
      res.json(data)
    }).catch(err => {
      return res.status(500).json({
        msg: "ไม่สามารถเพิ่มข้อมูลลูกค้าได้ เนื่องจาก : " + err.message
      });
    });
};
```

2. การเพิ่มข้อมูลของฟังก์ชัน `create` กำหนดให้เรียกด้วยพาธ `/api/customer` แบบ POST

### \routes\customer.route.js

```
app.post('/api/customer', customer.create);
```

3. การแสดงข้อมูลรายการเดียว เป็นหน้าที่ของฟังก์ชัน `findById` (`exports.findById`) โดยการใช้ฟังก์ชัน `findById()` ค้นหาข้อมูลลูกค้าตามรหัสลูกค้าที่ส่งมาทางพารามิเตอร์ `customerId` (`customer.findById(req.params.customerId)`)

### \controllers\customer.controller.js

```
exports.findById = (req, res) => {
  customer.findById(req.params.customerId)
    .then(data => {
      if (!data) {
        return res.status(404).json({
          msg: "ไม่พบลูกค้ารหัส : " + req.params.customerId
        });
      }
      res.json(data);
    });
};
```

# การสร้างส่วนแสดงผลสำหรับแสดงในหน้าจอทุกขนาดด้วย Bootstrap (Responsive Design)

หลักการออกแบบส่วนแสดงผลในยุคปัจจุบัน ต้องสามารถรองรับการแสดงผลในหน้าจอหลายขนาด เรียกว่า **Responsive Design** เพราะว่ามีผู้ใช้งานเข้าหน้าเว็บไซต์จากอุปกรณ์ PC, มือถือ และแท็บเล็ต อุปกรณ์เหล่านี้มีขนาดความละเอียดแตกต่างกัน จึงทำให้เราต้องสร้างส่วนแสดงผลที่สามารถแสดง Layout ให้เหมาะสมกับอุปกรณ์ที่มีความหลากหลาย

เนื้อหาที่นำเสนอในหนังสือเล่มนี้ เลือกใช้ Bootstrap เวอร์ชัน 5.1 ขึ้นไป ทำงานร่วมกับ Angular เพื่อสร้างส่วนแสดงผลที่สามารถแสดงผลในหน้าจอได้ทุกขนาด

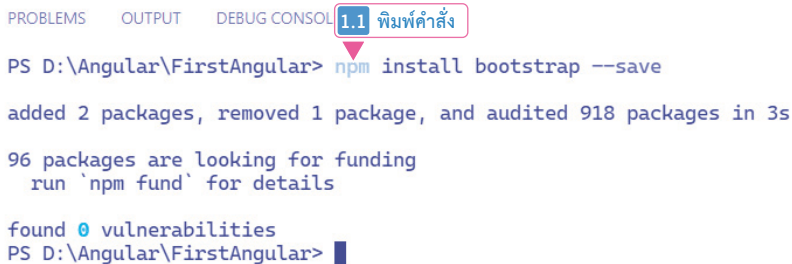
## การติดตั้ง Bootstrap ให้กับโปรเจกต์ Angular

มีขั้นตอนดังนี้

1. ที่หน้าต่าง TERMINAL ของโปรแกรม Visual Studio Code ให้พิมพ์คำสั่งต่อไปนี้เพื่อดาวน์โหลด Bootstrap มาไว้ที่โปรเจกต์ FirstAngular ก่อน

### TERMINAL

```
npm install bootstrap --save
```



```
PROBLEMS OUTPUT DEBUG CONSOLE 1.1 พิมพ์คำสั่ง
PS D:\Angular\FirstAngular> npm install bootstrap --save
added 2 packages, removed 1 package, and audited 918 packages in 3s
96 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS D:\Angular\FirstAngular> █
```

# การสร้างแถบ Footer

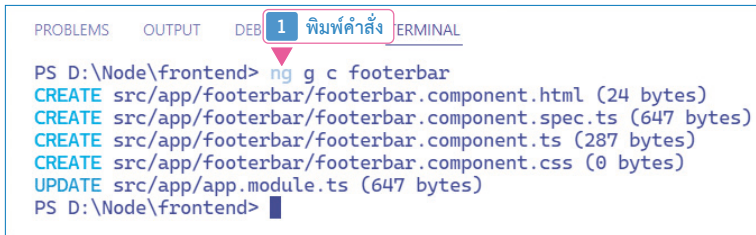
แถบ **Footer** คือ พื้นที่บริเวณขีดขอบด้านล่างของส่วนแสดงผล มักจะใช้แสดงข้อมูลพื้นฐานต่างๆ เช่น ผู้พัฒนา, วิธีการติดต่อ, นโยบายข้อมูล เป็นต้น ถือเป็นส่วนแสดงผลประเภทหนึ่งที่ต้องมี

**ตัวอย่างที่ 10-2** การสร้างแถบ Footer มีขั้นตอนดังนี้

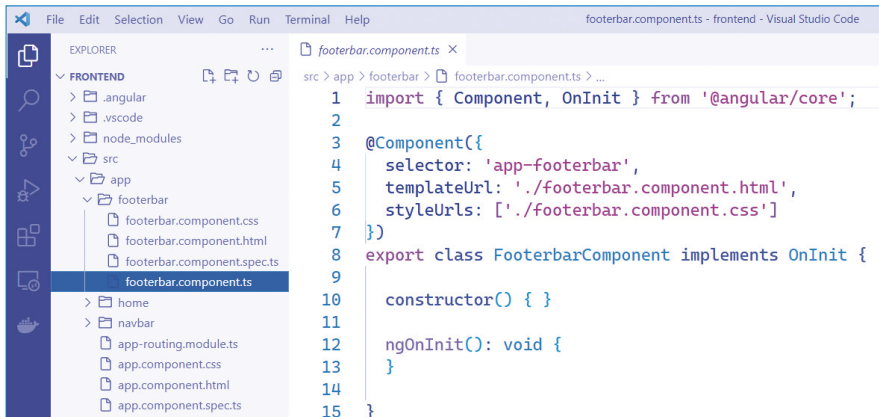
1. ให้พิมพ์คำสั่งต่อไปนี้เพื่อสร้าง FooterbarComponent ดังรูปที่ 10-10

## TERMINAL

```
ng g c footerbar
```



```
PS D:\Node\frontend> ng g c footerbar
CREATE src/app/footerbar/footerbar.component.html (24 bytes)
CREATE src/app/footerbar/footerbar.component.spec.ts (647 bytes)
CREATE src/app/footerbar/footerbar.component.ts (287 bytes)
CREATE src/app/footerbar/footerbar.component.css (0 bytes)
UPDATE src/app/app.module.ts (647 bytes)
PS D:\Node\frontend>
```

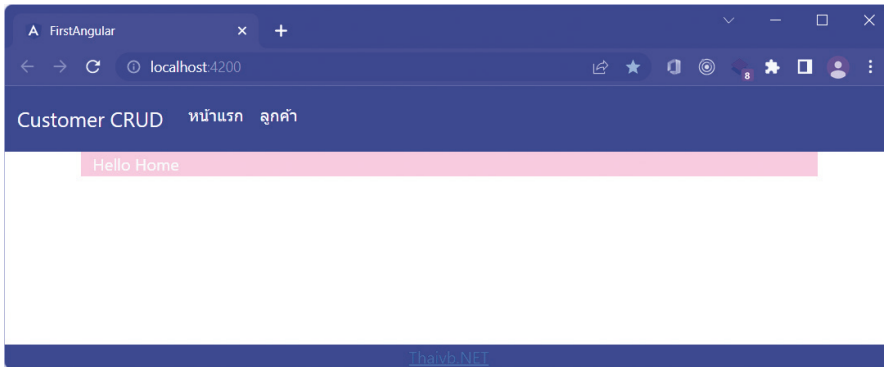


```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-footerbar',
5   templateUrl: './footerbar.component.html',
6   styleUrls: ['./footerbar.component.css']
7 })
8 export class FooterbarComponent implements OnInit {
9
10  constructor() { }
11
12  ngOnInit(): void {
13  }
14
15 }
```

รูปที่ 10-10 แสดง FooterbarComponent

จากรูปที่ 10-10 FooterbarComponent มีชื่อเรียกว่า app-footerbar

4. ให้ผู้อ่านทดสอบรันโปรเจกต์ (ng serve) ก็จะได้แถบ Footer อยู่ชิดด้านล่างสุดของส่วนแสดงผล ดังรูปที่ 10-13



รูปที่ 10-13 ส่วนแสดงผลที่ได้

## การสร้างส่วนแสดงผลสำหรับงาน CRUD ด้วย Angular

เรามีโปรเจกต์ Angular ในโพลเดอร์ frontend และมีโปรเจกต์ Express ในโพลเดอร์ backend ทั้ง 2 โปรเจกต์ทำงานร่วมกัน เรียกว่า MEAN Stack เพื่อสร้างงานประเภท CRUD โดยที่งานฝั่งหลังบ้าน Express เราทำเสร็จเรียบร้อยแล้ว เหลือฝั่งหน้าบ้าน Angular

### การสร้างบริการส่วนกลาง (Service) ใน Angular

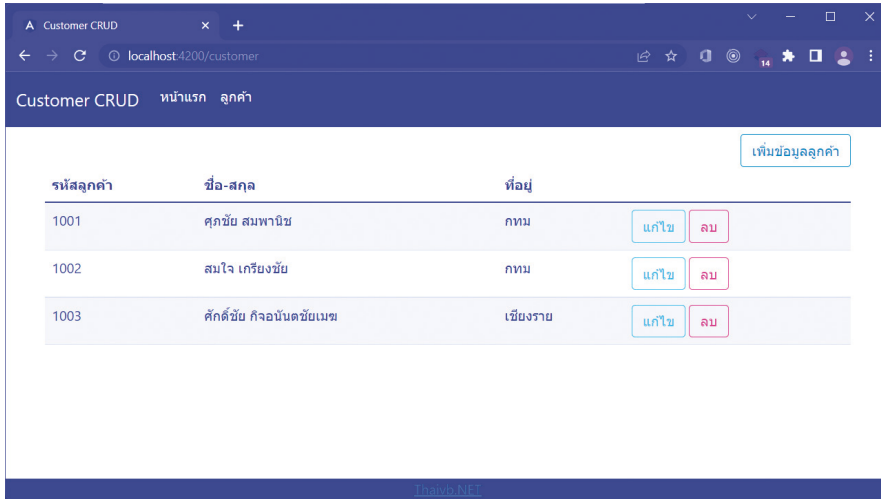
การเชื่อมต่อโปรเจกต์ Angular ไปที่ฝั่งหลังบ้าน เราไม่ได้ใช้วิธีการกำหนดให้แต่ละ Component เชื่อมต่อกันเอง แต่ใช้วิธีสร้างบริการส่วนกลาง เรียกว่า **Service** ขึ้นมาทำหน้าที่เชื่อมต่อไปที่ฝั่งหลังบ้าน จากนั้นเราเป็นผู้กำหนดเองว่า ให้ Component เรียกใช้บริการต่างๆ จาก Service กลางนี้เท่านั้น เพราะว่า Component แต่ละตัวมีหน้าที่เฉพาะแตกต่างกันอยู่แล้ว

บริการกลาง Service ของ Angular ใช้ระบบคลาสของภาษา TypeScript มีหน้าที่เชื่อมต่อไปที่ฝั่งหลังบ้านกับงานพื้นฐาน 4 อย่างคือ CRUD ที่ได้จากฝั่งหลังบ้าน

**ตัวอย่างที่ 10-3** การสร้างบริการส่วนกลาง (Service) ใน Angular มีขั้นตอนดังนี้

1. ที่โปรเจกต์ frontend (Angular) ให้พิมพ์คำสั่งต่อไปนี้ เพื่อสร้างบริการกลางที่ชื่อว่า CustomerService (ไฟล์ customer.service.ts) เก็บไว้ในโพลเดอร์ที่ชื่อว่า shared ทำหน้าที่บริการงาน CRUD ของลูกค้าเพียงอย่างเดียวเท่านั้น ดังรูปที่ 10-14





รูปที่ 10-27 กรณีสลบข้อมูลลูกค้า

## อธิบายการทำงานของสคริปต์การทำงาน CRUD

- การเพิ่มและลบข้อมูลลูกค้า เริ่มต้นที่ไฟล์ `customer-read.component.ts` ส่วนที่ต้องอ้างอิงก่อน ประกอบด้วย
  - บริการกลาง `CustomerService`** ที่อยู่ในไฟล์ `customer.service.ts` จากที่ผู้เขียนกล่าวไว้ตั้งแต่ตอนต้นแล้วว่า Component แต่ละตัวไม่มีการเชื่อมต่อไปที่บริการ Web API ของ Express ให้เรียกผ่านทางบริการกลางเท่านั้น
  - โครงสร้างข้อมูล `Customer`** เก็บอยู่ในไฟล์ `customer.model.ts`

```
\src\app\customer\customer-read\customer-read.component.ts
```

```
import { Component, OnInit } from '@angular/core';
import { CustomerService } from "../../shared/customer.service";
import { Customer } from "../../shared/customer.model";
```

- สร้างตัวแปรที่ชื่อว่า `delId` ทำหน้าที่เก็บรหัสลูกค้าที่ต้องการลบข้อมูล และสร้างตัวแปรอาร์เรย์ที่ชื่อว่า `customers` ทำหน้าที่เก็บรายการลูกค้าที่ได้มาจากฝั่งหลังบ้าน Express

```
\src\app\customer\customer-read\customer-read.component.ts
```

```
export class CustomerReadComponent implements OnInit {
  delId: string = "";
  customers: Customer[];
```

# หลักการสร้างส่วนแสดงผลด้วย Pug

หลักการสร้างส่วนแสดงผลด้วย Pug ผู้เขียนขอให้อีดี 3 หลักการนี้ไว้คือ

1. ไฟล์ส่วนแสดงผล Pug มีนามสกุล .pug ใช้หลักการเดียวกับ HTML มีข้อแตกต่างคือ อีลีเมนต์ต่างๆ ไม่ต้องใช้เครื่องหมาย < > กำกับไว้ เช่น <div>...</div> ใน HTML กลายเป็น div ใน Pug
2. ย่อหน้ามีความสำคัญเป็นอย่างมากใน Pug แนะนำให้ใช้ปุ่ม `Tab` เป็นหลักทั้งหมด
3. การส่งค่าของข้อมูลจาก Express ไปสู่ส่วนแสดงผลของ Pug ทำผ่านทางตัวแปรที่สามารถแทรกเข้าไปในไฟล์ .pug ได้

ผู้เขียนขอใช้โปรเจกต์ firstexpress เพื่ออธิบายการทำงานร่วมกันระหว่าง Express กับ Pug มีรายละเอียดดังนี้

1. สร้างตัวแปร indexRouter ทำหน้าที่แทนไฟล์ index.js และสร้างตัวแปร usersRouter ทำหน้าที่แทนไฟล์ users.js ที่อยู่ในโฟลเดอร์ routes

## app.js

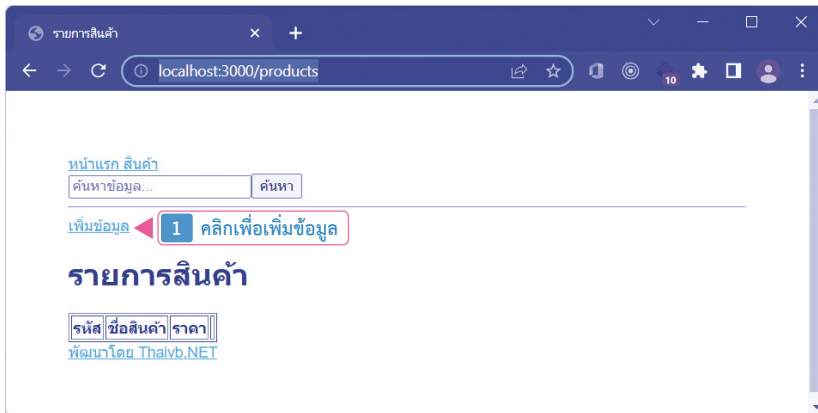
```
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
```

2. การกำหนดให้โปรเจกต์ Express สามารถใช้ส่วนแสดงผลของ Pug ทำได้โดยการกำหนดในไฟล์ app.js ให้ใช้โฟลเดอร์ views และโฟลเดอร์ public ระบุให้ใช้ส่วนแสดงผลประเภท pug

## app.js

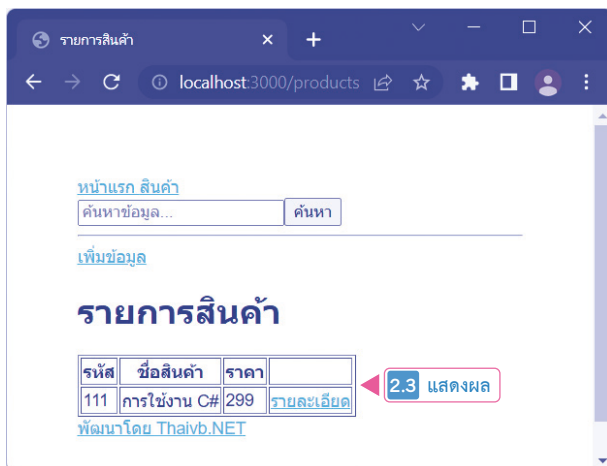
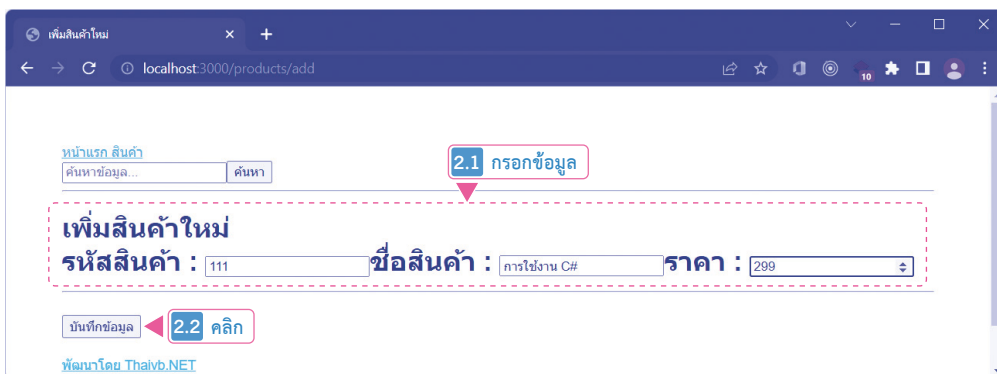
```
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

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')));
```



รูปที่ 12-17 แสดงการรันโปรเจกต์ crudpgu ครั้งแรก

2. การเพิ่มข้อมูลสินค้าใหม่ ให้คลิกที่ลิงค์เพิ่มข้อมูล ก็จะได้แบบฟอร์มเพิ่มข้อมูลสินค้าใหม่ ใส่ข้อมูลให้ครบ ดังรูปที่ 12-18



รูปที่ 12-18 แสดงการเพิ่มข้อมูลสินค้ารายการแรก

## 7. การค้นหาข้อมูลจะใช้ชื่อสินค้าเป็นเงื่อนไขในการค้นหา ดังรูปที่ 12-23



รูปที่ 12-23 กรณีค้นหาข้อมูลสินค้า

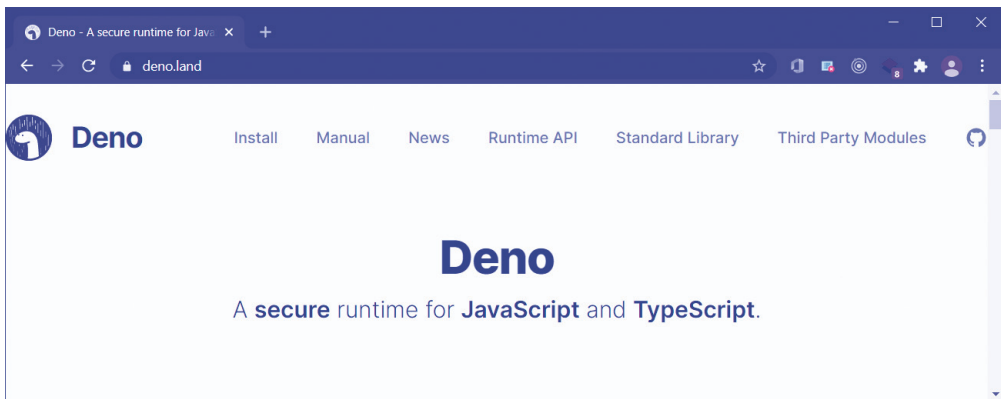
## อธิบายการทำงานของสคริปต์ของโปรเจกต์ Express และ Pug ด้วย Express Generator

- การทำงานเริ่มต้นที่ไฟล์ app.js ของ Express ให้กำหนดโครงสร้างข้อมูลสินค้าขึ้นมาก่อน ตั้งชื่อว่า ProductSchema ประกอบด้วย
  - รหัสสินค้า (id)
  - ชื่อสินค้า (productname)
  - ราคาสินค้า (price)

บทที่  
13

# การใช้งาน JavaScript/TypeScript บนแพลตฟอร์ม Deno

แพลตฟอร์ม Deno มีหลักการแตกต่างจาก Node หลายประการ ที่สำคัญก็คือ แพลตฟอร์ม Deno ยึดถือหลักการว่า ปกปิดหรือป้องกันไว้ก่อนแล้วค่อยเปิดทีหลัง มาจากคำนิยามหน้าแรก "[A secure runtime for JavaScript and TypeScript](#)"



รูปที่ 13-1 แสดงหน้าแรกของเว็บไซต์ deno.land

# การจัดเก็บโปรเจกต์ใน GitHub

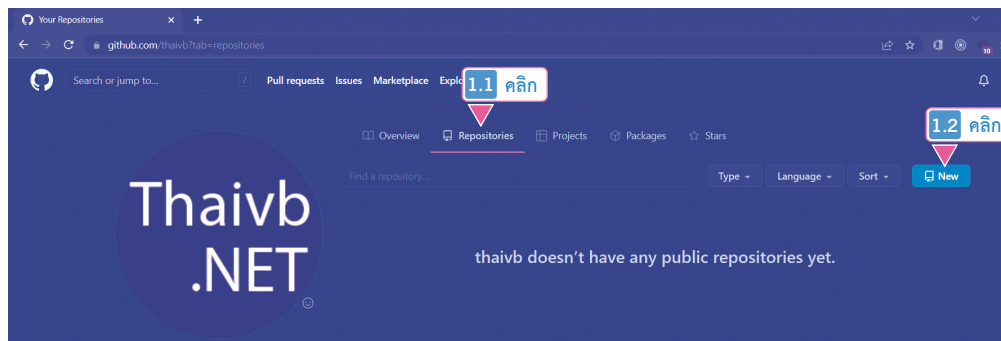
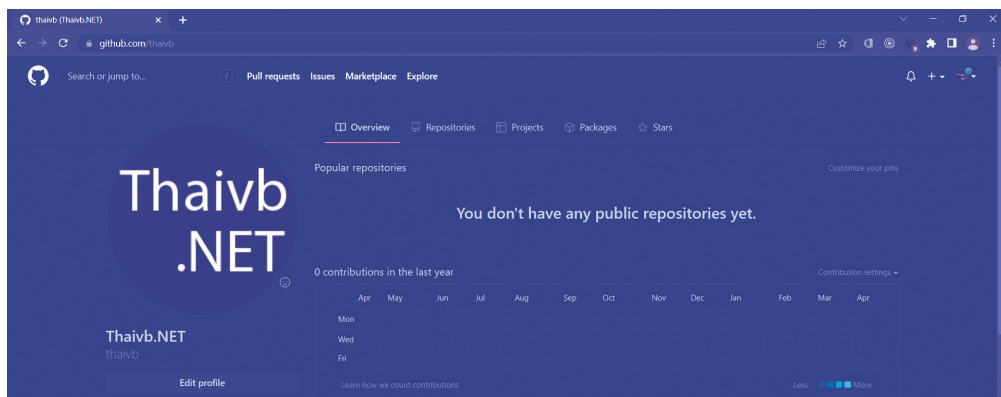
**GitHub** เป็นเว็บที่มีบริการเก็บโปรเจกต์ของเรา ผู้อ่านสามารถอัปโหลดโปรเจกต์ได้ด้วยวิธีพิมพ์คำสั่งต่างๆ ที่มากับโปรแกรม Git การอัปโหลดโปรเจกต์มาเก็บไว้ที่เว็บ GitHub เรียกว่า **push** โดยมีขั้นตอนดังนี้

## Note

ในขณะที่หนังสือเล่มนี้จัดทำจำหน่าย ผู้เขียนลบโปรเจกต์ต่างๆ ที่อยู่ใน GitHub ไปเรียบร้อยแล้ว ขอให้ผู้อ่านสร้างบัญชีและใช้โปรเจกต์ของผู้อ่านเป็นหลัก

1. ไปที่เว็บไซต์ <https://github.com/> สมัครเป็นสมาชิกให้เรียบร้อยก่อน ในกรณีนี้ ผู้เขียนตั้งชื่อว่า thaivb จึงได้พาธ [github.com/thaivb](https://github.com/thaivb)

วิธีการเก็บโปรเจกต์ของเราใน GitHub ทำได้โดยการคลิกแท็บ Repositories ก่อน จากนั้นคลิกปุ่ม **New** เพื่อสร้าง Repository หรือสร้างโฟลเดอร์ขึ้นมาเก็บโปรเจกต์ ดังรูปที่ 15-6

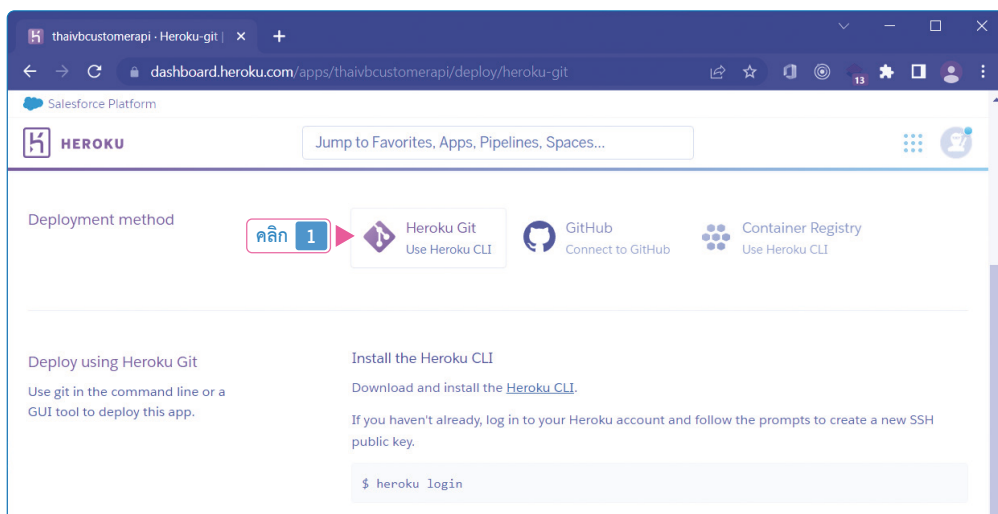


รูปที่ 15-6 แสดงการสร้าง Repository (หรือสร้างโฟลเดอร์) เพื่อเก็บโปรเจกต์ของเรา

# การอัปเดตโปรเจกต์ Express เข้าสู่ Heroku ด้วยวิธี Heroku CLI

วิธีการพื้นฐานที่สุดที่จะนำโปรเจกต์ Express (backend) ของเรามาเปิดบริการแบบสาธารณะ โดยอาศัยบริการฟรีของ Heroku ก็คือ ใช้คำสั่ง **Heroku CLI** มีขั้นตอนดังนี้

1. คลิกที่ Heroku Git เพื่อดูคำสั่งที่เราต้องทำอะไรบ้าง ดังรูปที่ 15-27



รูปที่ 15-27 แสดงคำสั่งของ Heroku CLI

2. การใช้งานคำสั่ง Heroku CLI ต้องมีการติดตั้งโปรแกรมก่อน ในกรณีนี้ ผู้เขียนใช้ Windows 10/11 แบบ 64 บิต และเมื่อติดตั้งโปรแกรมจนเสร็จสมบูรณ์ และ Restart เครื่อง 1 ครั้ง ดังรูปที่ 15-28

