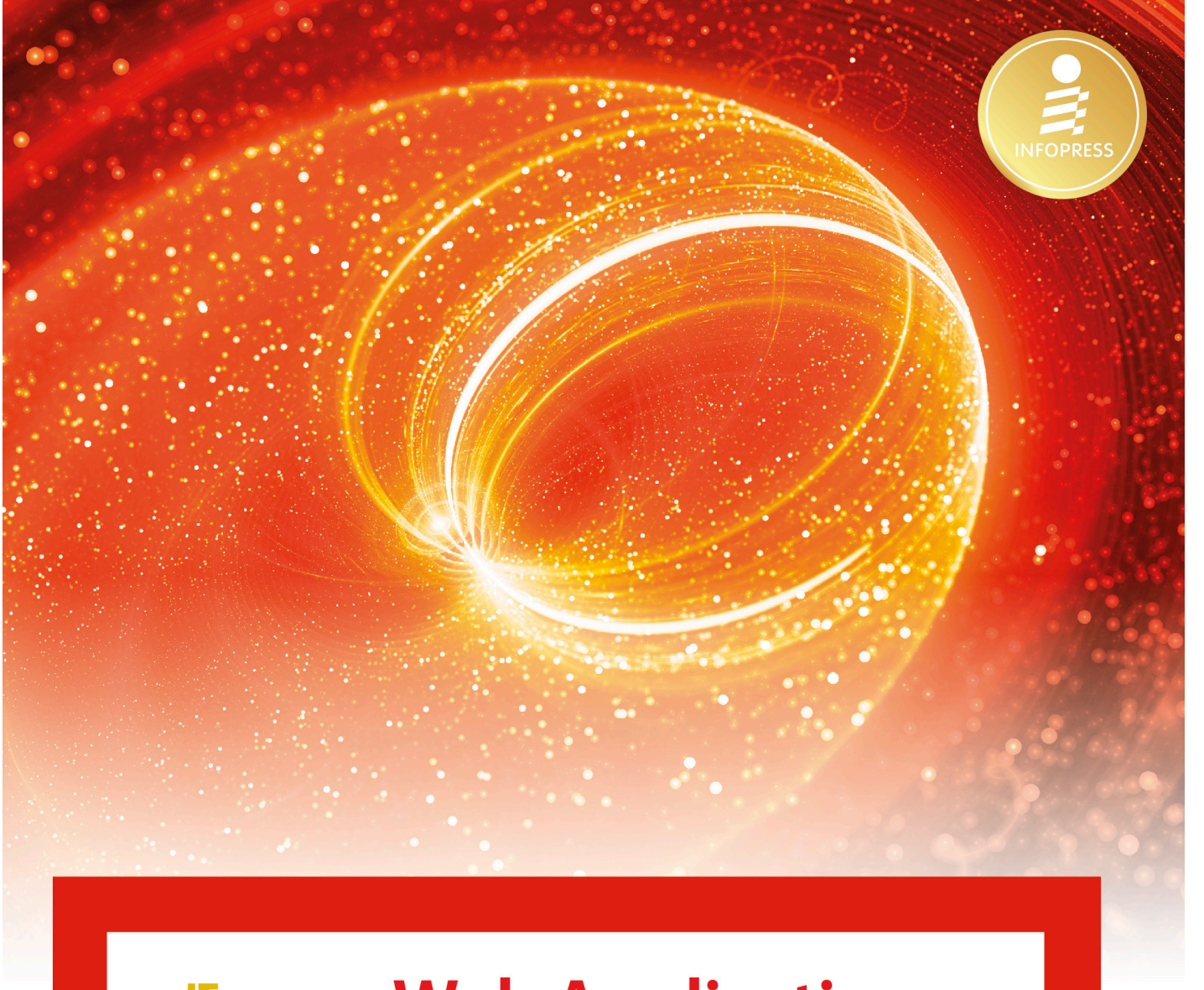


INFOPRESS

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


JAVA

บทที่ 4

คำสั่งตรวจสอบและการวนลูป

การศึกษาเขียนโปรแกรมของทุกๆ ภาษา ก็จะมีเนื้อหาพื้นฐานหัวข้อแรกๆ ที่มือใหม่ต้องมาทำความรู้จักและศึกษาก่อนเลย นั่นคือ การตรวจสอบเงื่อนไขเพื่อสร้างทางเลือกและการสั่งให้เกิดการทำงานซ้ำ หรือที่เรียกว่า **การวนลูป**

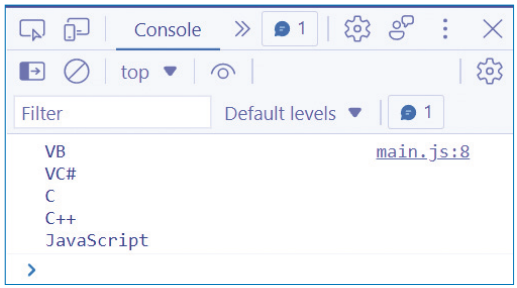
การตรวจสอบเงื่อนไขด้วยคำสั่ง if

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

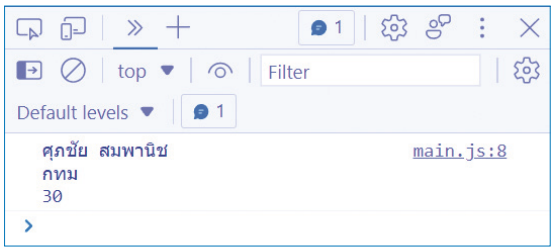
การวนลูปแบบ for in

ในกรณีที่เราต้องทำงานกับข้อมูลออบเจ็กต์ที่มีโครงสร้างซับซ้อน เราต้องใช้ลูปแบบ **for in** เช่น ค่าคงที่ data เก็บข้อมูลที่มีโครงสร้างภายใน ประกอบด้วย 3 ส่วนคือ fullname, Address และ age เราสามารถใช้ลูปแบบ 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

การแปลงข้อมูลในออบเจ็กต์เป็นอาร์เรย์ด้วยฟังก์ชัน entries() และฟังก์ชัน values()

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

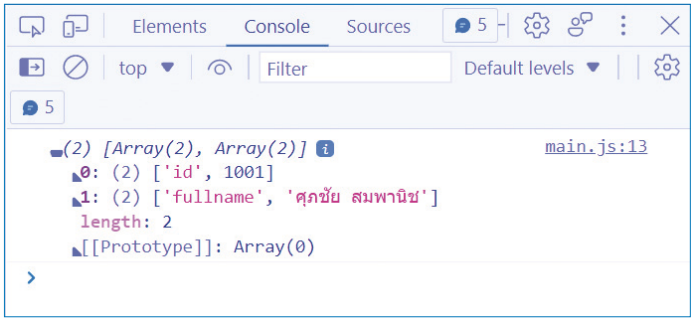
คลาส Customer เก็บข้อมูล 2 ส่วนคือ รหัส (id) และชื่อ-สกุล (fullname) ดังสคริปต์ต่อไปนี้

\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

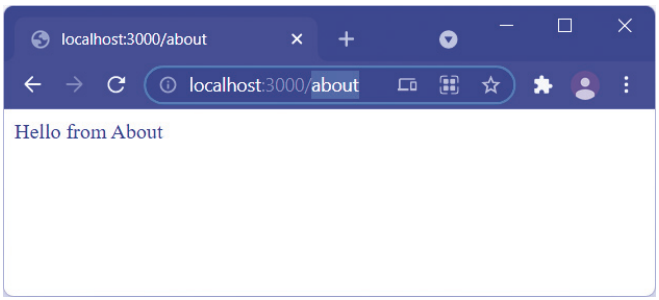
- **id** เก็บค่า 1001
- **fullname** เก็บค่า "ศุภชัย สมพานิช"



รูปที่ 5-8 ผลการแปลงข้อมูลด้วยฟังก์ชัน entries()

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

จากสคริปต์ข้างต้นมีความหมายว่า ตอนนี้เราเปิดให้บริการ 2 พาธแล้วคือ พาธ / กับ /About ลองใช้บราวเซอร์เข้าที่พาธใหม่ /about ผลการทำงานแสดงดังรูปที่ 6-30



รูปที่ 6-30 ผลการทำงานของพาธ /About

พื้นฐานการจัดการพาธอื่นๆ ที่ไม่มี (404 page not found)

ความรู้พื้นฐานสำหรับการจัดการเส้นทาง หรือพาธอีกอย่างหนึ่งก็คือ ในกรณีที่มีการร้องขอข้อมูลมาที่พาธที่ไม่มีอยู่จริงใน Web API ของเราคือ ข้อผิดพลาดที่เรียกว่า **404 page not found**

วิธีการแก้ไขเบื้องต้นก็คือ ให้ใช้เครื่องหมาย * เพื่อระบุครอบคลุมพาธอื่นๆ 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 ผู้เขียนจะนำโปรเจ็กต์ที่แล้วมาทำต่อให้ครบ

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** เพราะว่าผู้ใช้งานเข้าหน้าเว็บไซต์จากอุปกรณ์ PC, มือถือ และแท็บเล็ต อุปกรณ์เหล่านี้มีขนาดความละเอียดแตกต่างกัน จึงทำให้เราต้องสร้างส่วนแสดงผลที่สามารถแสดง Layout ให้เหมาะสมกับอุปกรณ์ที่มีความหลากหลาย

เนื้อหาที่นำเสนอในหนังสือเล่มนี้ เลือกใช้ Bootstrap เวอร์ชัน 5.1 ขึ้นไป ทำงานร่วมกับ Angular เพื่อสร้างส่วนแสดงผลที่สามารถแสดงผลในหน้าจอได้ทุกขนาด

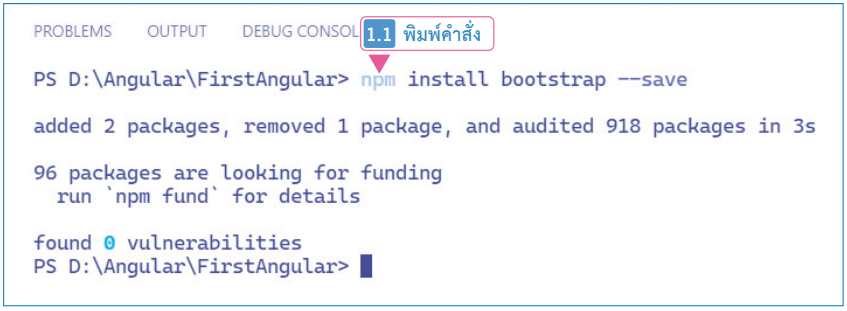
การติดตั้ง Bootstrap ให้กับโปรเจ็กต์ Angular

มีขั้นตอนดังนี้

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

TERMINAL

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



การสร้างแถบ Footer

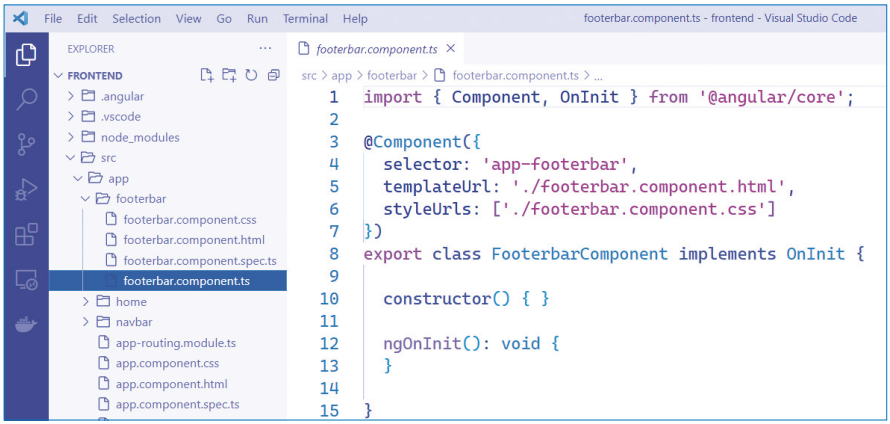
แถบ **Footer** คือ พื้นที่บริเวณชิดขอบด้านล่างของส่วนแสดงผล มักจะใช้แสดงข้อมูลพื้นฐานต่างๆ เช่น ผู้พัฒนา, วิธีการติดต่อ, นโยบายข้อมูล เป็นต้น ถือว่าเป็นส่วนแสดงผลประเภทหนึ่งที่ต้องมี

ตัวอย่างที่ 10-2 การสร้างแถบ Footer มีขั้นตอนดังนี้

1. ให้พิมพ์คำสั่งต่อไปนี้เพื่อสร้าง FooterbarComponet ดังรูปที่ 10-10

TERMINAL

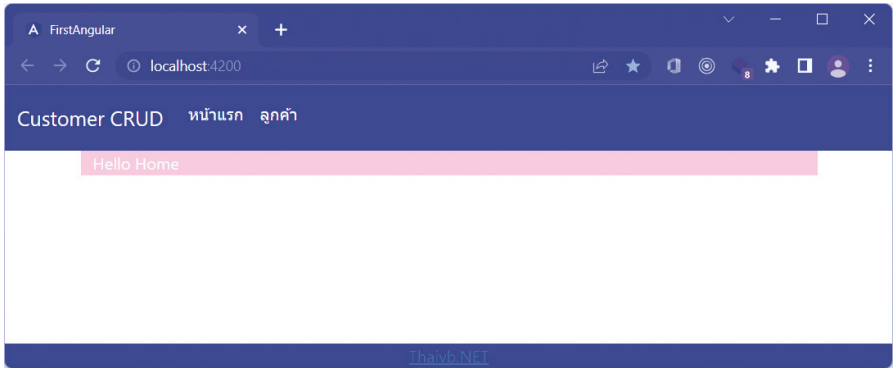
```
ng g c footerbar
```



รูปที่ 10-10 แสดง FooterbarComponent

จากรูปที่ 10-10 FooterbarComponent มีชื่อเรียกว่า app-footerbar

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



รูปที่ 10-13 ส่วนแสดงผลที่ได้

การสร้างส่วนแสดงผลสำหรับงาน CRUD ด้วย Angular

เรามีโปรเจ็กต์ Angular ในโฟลเดอร์ frontend และมีโปรเจ็กต์ Express ในโฟลเดอร์ backend ทั้ง 2 โปรเจ็กต์ทำงานร่วมกัน เรียกว่า MEAN Stack เพื่อสร้างงานประเภท CRUD โดยที่งานฝั่งหลังบ้าน Express เราทำเสร็จเรียบร้อยแล้ว เหลือฝั่งหน้าบ้าน Angular

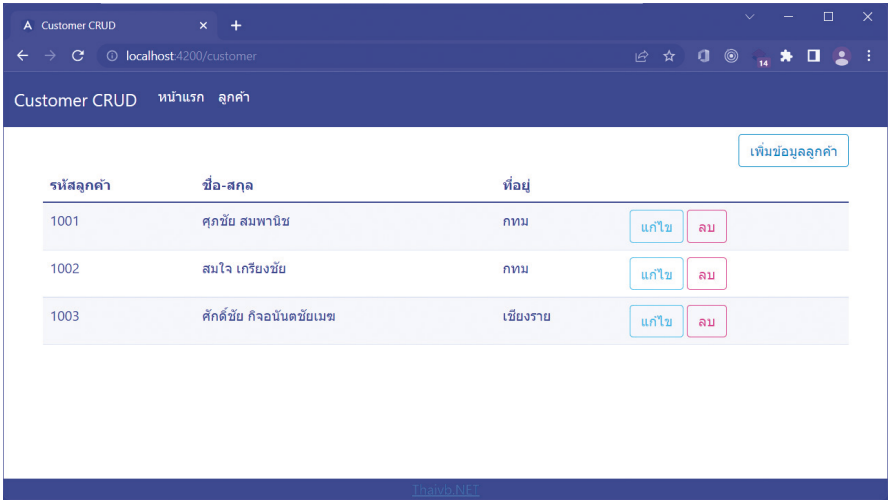
การสร้างบริการส่วนกลาง (Service) ใน Angular

การเชื่อมต่อโปรเจ็กต์ 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. ที่โปรเจ็กต์ frontend (Angular) ให้พิมพ์คำสั่งต่อไปนี้ เพื่อสร้างบริการกลางที่ชื่อว่า CustomerService (ไฟล์ customer.service.ts) เก็บไว้ในโฟลเดอร์ที่ชื่อว่า shared ทำหน้าที่บริการงาน CRUD ของลูกค้าเพียงอย่างเดียวเท่านั้น ดังรูปที่ 10-14



รูปที่ 10-27 กรณีลบข้อมูลลูกค้า

อธิบายการทำงานของสคริปต์การทำงาน CRUD

1. 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";
```

2. 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 ผู้เขียนขอให้ยึด 3 หลักการนี้ไว้คือ

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

ผู้เขียนขอใช้โปรเจ็กต์ firstexpress เพื่ออธิบายการทำงานร่วมกันระหว่าง Express กับ Pug มีรายละเอียดดังนี้

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

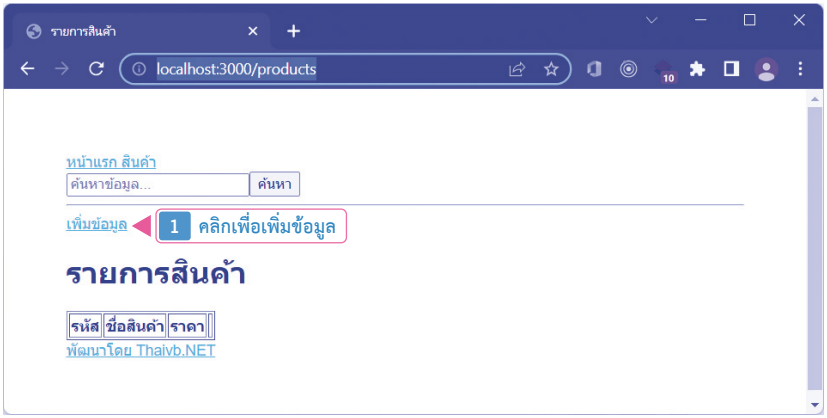
app.js

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

2. การกำหนดให้โปรเจ็กต์ Express สามารถใช้ส่วนแสดงผลของ Pug ทำได้โดยการกำหนดในไฟล์ app.js ให้ใช้โฟลเดอร์ views และโฟลเดอร์ public ระบุให้ใช้ส่วนแสดงประเภท pug

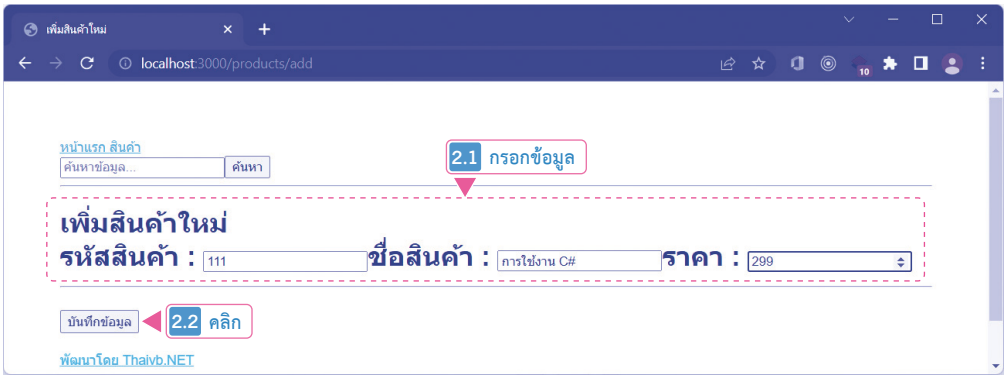
app.js

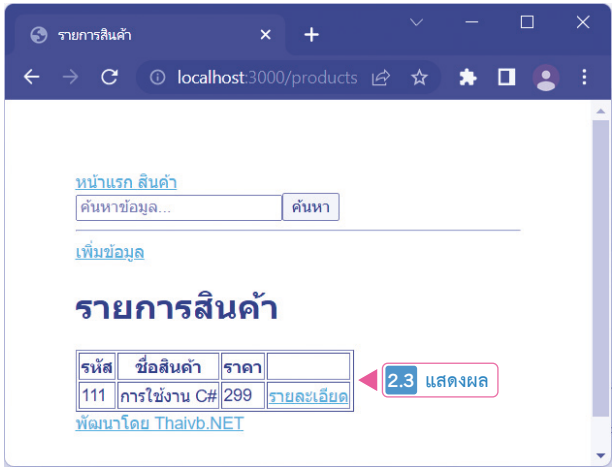
```
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

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 แสดงการรันโปรเจ็กต์ crudpug ครั้งแรก

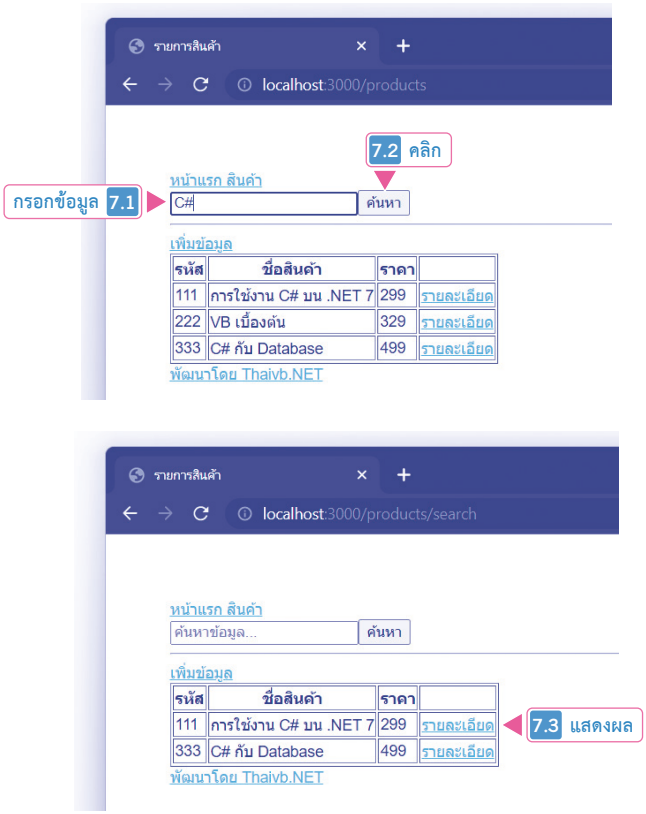
2. การเพิ่มข้อมูลสินค้าใหม่ ให้คลิกที่ลิงค์เพิ่มข้อมูล ก็จะได้แบบฟอร์มเพิ่มข้อมูลสินค้าใหม่ ใส่ข้อมูลให้ครบ ดังรูปที่ 12-18





รูปที่ 12-18 แสดงการเพิ่มข้อมูลสินค้ารายการแรก

7. การค้นหาข้อมูลจะใช้ชื่อสินค้าเป็นเงื่อนไขในการค้นหา ดังรูปที่ 12-23



รูปที่ 12-23 กรณีค้นหาข้อมูลสินค้า

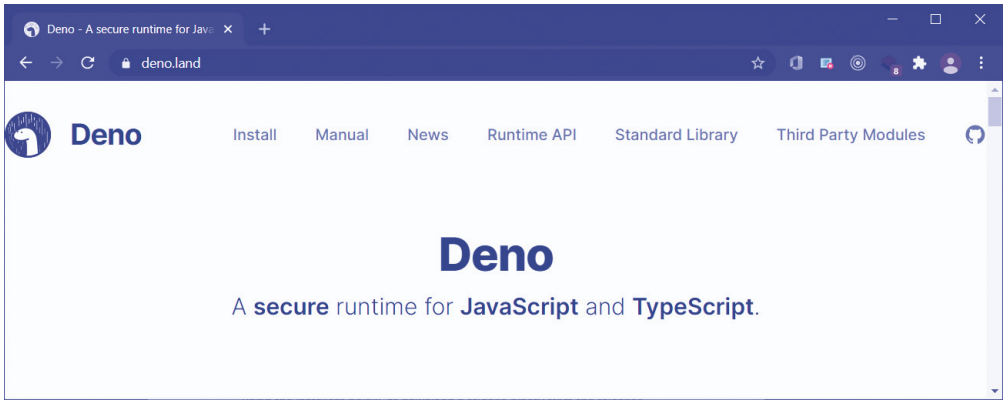
อธิบายการทำงานของสคริปต์ของโปรเจ็กต์ Express และ Pug ด้วย Express Generator

1. 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

การจัดเก็บโปรเจ็กต์ใน GitHub

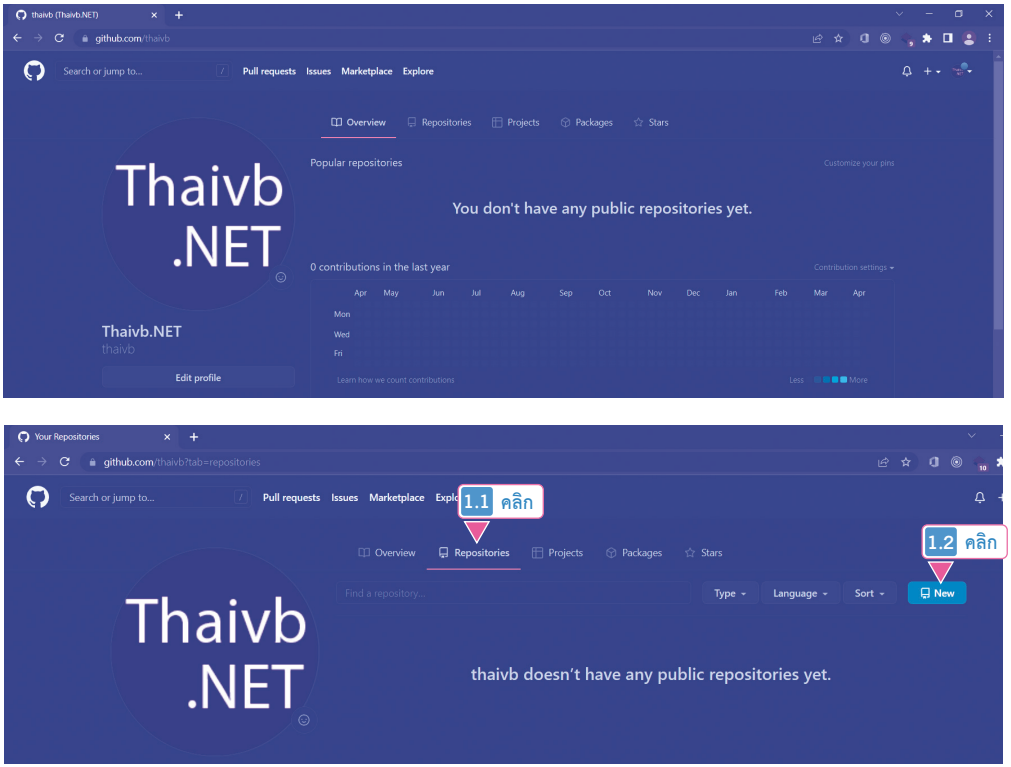
GitHub เป็นเว็บที่มีบริการเก็บโปรเจ็กต์ของเรา ผู้อ่านสามารถอัพโหลดโปรเจ็กต์ได้ด้วยวิธีพิมพ์คำสั่งต่างๆ ที่มากับโปรแกรม Git การอัพโหลดโปรเจ็กต์มาเก็บไว้ที่เว็บ GitHub เรียกว่า **push** โดยมีขั้นตอนดังนี้

Note

ในขณะที่หนังสือเล่มนี้จัดจำหน่าย ผู้เขียนลบโปรเจ็กต์ต่างๆ ที่อยู่ใน GitHub ไปเรียบร้อยแล้ว ขอให้ผู้อ่านสร้างบัญชีและใช้โปรเจ็กต์ของผู้อ่านเป็นหลัก

1. ไปที่เว็บไซต์ https://github.com/ สมัครเป็นสมาชิกให้เรียบร้อยก่อน ในกรณีนี้ ผู้เขียนตั้งชื่อว่า thaivb จึงได้พาธ github.com/thaivb

วิธีการเก็บโปรเจ็กต์ของเราใน GitHub ทำได้โดยการคลิกแท็บ Repositories ก่อน จากนั้นคลิกปุ่ม **New** เพื่อสร้าง Repository หรือสร้างโฟลเดอร์ขึ้นมาเก็บโปรเจ็กต์ ดังรูปที่ 15-6

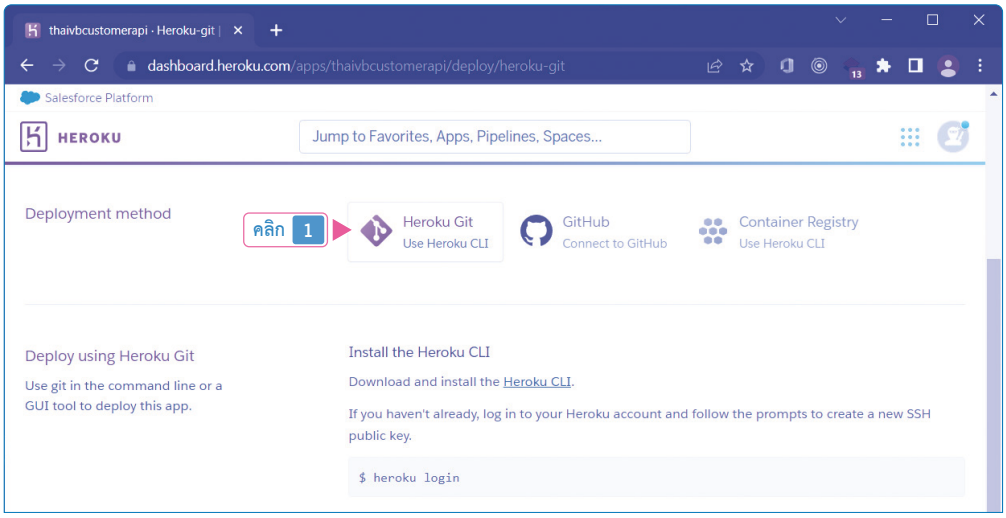


รูปที่ 15-6 แสดงการสร้าง Repository (หรือสร้างโฟลเดอร์) เพื่อเก็บโปรเจ็กต์ของเรา

การอัพโหลดโปรเจ็กต์ Express เข้าสู่ Heroku ด้วยวิธี Heroku CLI

วิธีการพื้นฐานที่สุดที่จะนำโปรเจ็กต์ Express (backend) ของเรามาเปิดบริการแบบสาธารณะโดยอาศัยบริการฟรีของ Heroku ก็คือ ใช้คำสั่ง **Heroku CLI** มีขั้นตอนดังนี้

1. คลิกที่ Heroku Git เพื่อดูคำสั่งว่าเราต้องทำอะไรบ้าง 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