

แองกูลาร์ กับ MySQL

ในการทำงานกับฐานข้อมูล MySQL ซึ่งเป็นเซิร์ฟเวอร์ จำเป็นต้องมีภาษาที่สื่อสารกับเซิร์ฟเวอร์โดยตรง โดยทั่วไปมักมี PHP เป็นภาษาที่นิยมใช้กับฐานข้อมูลนี้ แต่ก็สามารถใช้ภาษาอื่น ๆ ซึ่งเนื้อหาไม่ได้เน้นการใช้ ฐานข้อมูลกับ PHP มากนัก แต่จะเน้นการใช้ แองกูลาร์ สื่อสารทางไกลกับเซิร์ฟเวอร์ โดยใช้ PHP และ MySQL เป็นกรณีศึกษาเท่านั้น ดังนั้นในบทนี้เราจะทำความเข้าใจในเรื่องต่อไปนี้

- การใช้ PHP จำลอง RESTful API
- การใช้ HttpClient ของ แองกูลาร์ เพื่อสื่อสารกับเซิร์ฟเวอร์
- การใช้ RESTful API กับ MySQL

RESTful

ในปี ค.ศ. 2,000 โดย นายรอย ฟิลดิง (Roy Fielding) ได้นำเสนอรูปแบบสถาปัตยกรรม REST (Representational State Transfer) ซึ่งมาจากวิทยานิพนธ์ระดับปริญญาเอก โดยรูปแบบ REST สร้างข้อกำหนดคำขอ บริการบนพื้นฐานโปรโตคอล HTTP อย่างเป็นระบบ และเมื่อรวบรวมให้เป็นเว็บบริการ (Web Service) จึงเรียกชื่อใหม่ว่า RESTful หรือ RESTful web service

การประยุกต์ร่วมกับเว็บบริการ เพื่อการสร้างการขอบริการข้อมูล โดยใช้วิธีการขอบริการ (HTTP method) ที่กำหนดขึ้นเพิ่มเติมจาก ที่ HTML Form มีให้เพียง GET กับ POST แต่ RESTful เพิ่มคำขอบริการ โดยเพิ่ม PUT, PATCH, และ DELETE เพื่อมีการใช้งานที่ประสิทธิภาพ ตามข้อกำหนดของสถาปัตยกรรมนี้ (อ่านเพิ่มเติมเรื่อง Architectural constraints¹) แต่การใช้งาน อาจไม่ห้ามทุกคำขอบริการได้

ตาราง 1 การร้องขอบริการด้วย HTTP Method ต่างๆ ผ่าน URL

| เมธอด | การใช้งาน | ตัวอย่าง URL |
|--------|--------------|---|
| GET | /api/users | อ่านข้อมูล ทั้งหมดของ users |
| | /api/users/1 | อ่านข้อมูล เฉพาะ id=1 |
| POST | /api/users/ | สร้างข้อมูลผู้ใช้ใหม่ |
| PUT | /api/users/1 | ปรับปรุงข้อมูล users id =1 หรือสร้างใหม่ (หากไม่มี) |
| DELETE | /api/users/1 | ลบข้อมูล users id = 1 |

ดังที่ได้กล่าวนำมาก่อนแล้วว่า เว็บบริการ ต้องอยู่บนเว็บเซิร์ฟเวอร์ ดังนั้น ผู้ให้บริการก็คือเว็บเซิร์ฟเวอร์ ซึ่งจะต้องรองรับคำขอบริการตามข้อกำหนดแบบ REST กล่าวคือรองรับคำขอของ GET, POST, PUT, และ DELETE เป็นอย่างน้อย โดยข้อมูลที่สื่อสารอยู่ในรูป JSON (หรือ XML) แต่สำหรับ การทำงานร่วมกับ JavaScript แล้ว รูปแบบ JSON จะมีความเหมาะสมมากกว่าเพราะ จัดเป็นภาษาเดียวกัน

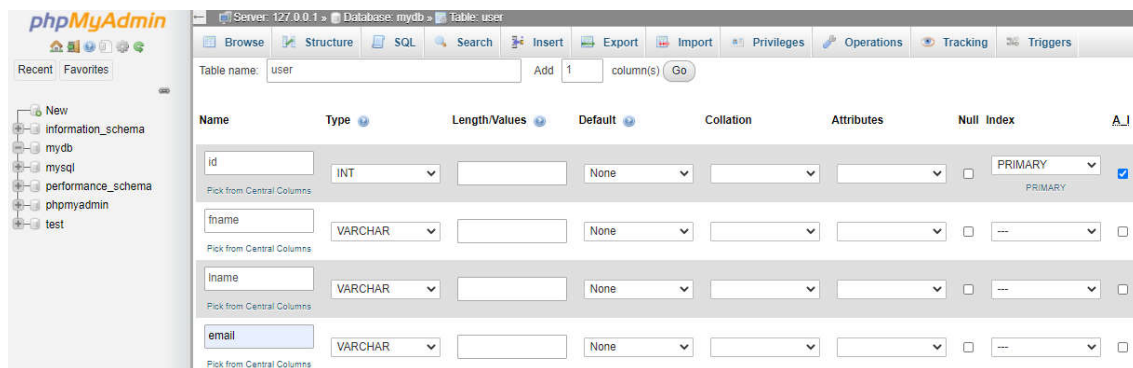
¹ https://en.wikipedia.org/wiki/Representational_state_transfer (2 July, 2018)

ใช้ภาษา PHP อ่านฐานข้อมูล

เพื่อเป็นการจำลองการใช้งาน เราจะให้มีตารางฐานข้อมูล ชื่อ user จากฐานข้อมูลชื่อ mydb ดังนั้นเมื่อสร้างเป็นตาราง user จะมีโครงสร้างตามตาราง 2 และตัวอย่างโครงสร้างตามรูป 1

ตาราง 2 ฐานข้อมูลตาราง User บน MySQL

| เมรอต | ชนิดข้อมูล | คีย์ | ความหมาย |
|-------|--------------|--------------------------|-------------|
| id | int | คีย์หลัก, auto-increment | รหัส ผู้ใช้ |
| fname | varchar(255) | | ชื่อ |
| lname | varchar(255) | | สกุล |
| email | varchar(255) | | อีเมล |



รูป 1 สร้างตาราง user ใน phpMyAdmin

เพื่อให้ดำเนินการฐานข้อมูล ในลักษณะต่าง ๆ จึงทดสอบด้วยการสร้างคลาส UserDataGateway เพื่อรองรับการอ่านที่ละเอียดชื่อ อ่านทุกรายชื่อ และแทรกรายชื่อใหม่ ถึงแม้จะไม่ครบทั้งหมดสำหรับงาน RESTful แต่ทำเท่านี้เพื่อเป็นการสาธิตเพียงบางส่วนของงานของ RESTful

ให้ไฟล์ UserDataGateway.php เก็บที่เซิร์ฟเวอร์ของ Apache เช่น ถ้าใช้บนโปรแกรมรวม XAMPP บนระบบ Windows เก็บในโฟลเดอร์ C:\xampp\htdocs\api\ โดยโฟลเดอร์ api เป็นโฟลเดอร์ที่สร้างขึ้นใหม่เพื่อทดสอบการใช้งานของบนี้

Code 1. UserDataGateway.php

```
<?php
class UserDataGateway{
    private $pdo;
    public function __construct(){
        $dsn = "mysql:dbname=mydb; host=127.0.0.1";
        $user = 'root';
        $password = '';
        $this->pdo= new PDO($dsn, $user, $password);
    }
    public function get($id=null){
        if($id==null){
            $sql = "select * from user";
```

```

        $result = $this->pdo->query($sql);
        return $result;
    }
    else{
        $sql = "select * from user where id=?";
        $stm = $this->pdo->prepare($sql);
        $stm->bindParam(1, $id);
        $stm->execute();
        return $stm;
    }
}

public function insert($array){
    $sql = "insert into user (fname, lname, email) values(?,?,?)";
    $stm = $this->pdo->prepare($sql);
    $stm->execute($array);
}
}

```

จากคลาส UserDataGateway เชื่อมกับฐานข้อมูลผ่านคอนสตรัคเตอร์ โดยรหัสจากตัวแปร \$password ไม่มี(ค่าว่าง) หากมีรหัสผ่านก็ต้องใส่รหัสด้วย และทดสอบแทรกข้อมูล ในตัวอย่างต่อไปนี้ (ไฟล์ test_db.php) ได้ทดลองแทรก 4 แถว ให้เรียกผ่าน url:

http://localhost/api/test_db.php

Code 2. api/test_db.php

```

<?php
include 'UserDataGateway.php';
$userDgw = new UserDataGateway();
$array = Array(["Pol", "L.", "pol@gmail.com"],
               ["Mol", "L.", "mol@gmail.com"],
               ["Dol", "K.", "dol@gmail.com"],
               ["Kol", "P.", "kol@gmail.com"]);
foreach($array as $r){
    $userDgw->insert($r);
}
echo "Complete!";

```

สร้างทางเข้าเซิร์ฟเวอร์

เพื่อให้ทุกคำขอมารวมอยู่ที่จุดเดียวกัน เซิร์ฟเวอร์ของ Apache ให้เขียนคำสั่งการเข้าถึงในไฟล์ .htaccess (ไฟล์ไม่มีชื่อแต่มีนามสกุล) ในคำสั่งนี้ ให้ทุกคำขอบริการที่วิ่งเข้ามา ไปยังไฟล์ index.php

Code 3. api/.htaccess

```

RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ index.php?request=$1 [QSA,NC,L]

Header always set Access-Control-Allow-Origin http://localhost:4200
Header always set Access-Control-Allow-Headers "X-Requested-With, Content-Type, Origin, Authorization, Accept, Client-Security-Token, Accept-Encoding"
Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS, DELETE, PUT"

```

อีกครั้ง บรรทัด Header มีเพียงสามบรรทัด บรรทัดที่ขึ้นต้นด้วย Origin ให้นำไปต่อท้ายบรรทัดก่อนหน้า (ต่อท้ายบรรทัด Header ที่สอง

ไฟล์นี้จะอ่านคำขอบริการที่ไปกับไฟล์ index.php โดยตัวแปรจะอยู่ในชื่อ request และค่าตัวแปรคือ \$1 ต่อไปจะถูกตัดออกเป็นอาร์เรย์

เมื่อคำขอถูกส่งต่อมายัง ไฟล์ index.php ให้ตัดคำขอออกเป็นส่วน ๆ ในรูปตัวแปรต่าง ๆ คือตัวแปรแรก เป็นชื่อประเภทคำขอ เช่น GET, POST, PUT โดยเก็บในชื่อตัวแปร \$verb ตัวแปรที่สองคือ \$args ใช้เก็บข้อมูลในรูปอาร์เรย์ที่ตัดคำได้จากคำของหลังไฟล์ index.php ตัวแปรสุดท้ายคือข้อมูล JSON ที่อาจส่งมากับคำขอ และรวมตัวแปรทั้งหมดส่งผ่านต่อไปยังคลาส RESTful ก่อนที่ส่งผลลัพธ์ออกมาในรูปแบบข้อมูล JSON อีกครั้งจากฟังก์ชัน json_encode()

Code 4. api/index.php

```
<?php
spl_autoload_register(function($class){
    include(ucfirst($class.'.php'));
});
$verb = $_SERVER['REQUEST_METHOD'];
$args = explode('/', rtrim($_REQUEST['request'], '/'));
$data = json_decode(file_get_contents('php://input'),true);
$restful = new RESTful($verb,$args, $data);
echo json_encode($restful->apply());
```

สร้างบริการ RESTfull

เพื่อให้ทุกคำขอมารวมอยู่ที่จุดเดียวกัน เซิร์ฟเวอร์ของ Apache ให้เขียนคำสั่งการเข้าถึงในไฟล์ .htaccess (ไฟล์ไม่มีชื่อแต่มีนามสกุล) ในคำสั่งนี้ ให้ทุกคำขอบริการที่วิ่งเข้ามา ไปยังไฟล์ index.php ดังเคยทำไว้แล้วในบทที่ผ่านมา

เมื่อคำขอถูกส่งต่อมายัง ไฟล์ index.php ให้ตัดคำขอบริการออกเป็นส่วน ๆ ในรูปตัวแปรต่างๆ แล้วเรียก สร้างวัตถุ RESTful ตามด้วยตัวแปรที่ตัดมาก่อนหน้านี้ ก่อนที่จะเรียกเมธอด apply() เพื่อส่งผลลัพธ์กับไปยังผู้ขอบริการ

Code 5. api/index.php

```
<?php
spl_autoload_register(function($class){
    include(ucfirst($class.'.php'));
});
$verb = $_SERVER['REQUEST_METHOD'];
$args = explode('/', rtrim($_REQUEST['request'], '/'));
$data = json_decode(file_get_contents('php://input'),true);
$restful = new RESTful($verb,$args, $data);
echo json_encode($restful->apply());
```

คลาสสำคัญอีกคลาสหนึ่งคือ RESTful จะทำหน้าที่ตาม คำกริยาที่ขอ เช่น ถ้าขอแบบ get ก็ใช้เมธอด get() ทำงาน ซึ่ง การขอแบบ get() นี้ก็มีสองแบบ คือขอแบบมีตัวแปรเข้า กับไม่มีตัวแปรเข้า ทำให้ต้องสร้างฟังก์ชัน get(id=null) ที่รองรับการมีหรือไม่มีตัวแปรไว้ด้วย

Code 6. api/RESTful.php

```
<?php
class RESTful{
```

```

public function __construct($request,$args,$data=null){
    $this->request = $request;
    $this->args = $args;
    $this->data = $data;
}
private function get($id=null){
    $model = $this->args[0];
    $class_data_model = $model.'DataGateway';
    $object_data_model = new $class_data_model();
    if($id!=null){
        $result = $object_data_model->get($id);
    }
    else {
        $result = $object_data_model->get();
    }
    $array = Array();
    foreach($result as $rs){
        //remove the last string if it contains 's'
        $model = rtrim($this->args[0],"s");
        $row = new $model($rs);
        $array[] = $row;
    }
    return $array;
}
private function put(){ }
private function delete(){ }
private function post(){ }
public function apply(){
    switch($this->request){
        case "POST":
            return $this->post();
            break;
        case "GET":
            if(sizeof($this->args)==1)
                return $this->get();
            else
                return $this->get($this->args[1]);
            break;
        case "PUT":
            return $this->put();
            break;
        case "DELETE":
            return $this->delete($this->args[1]);
            break;
        default:
            $this->get();
    }
}
}
}

```

จากคลาส RESTful มีการเรียกใช้โมเดล ซึ่งคือตัวงาน หรือข้อมูลในตัวงานที่ต้องการ ในกรณีการใช้งานกับ ตาราง User จึงสร้างโมเดลเป็นคลาส User คลาสนี้เป็นคลาสอย่างง่าย แต่รองรับการแปลงออกเป็นข้อมูล JSON ด้วย

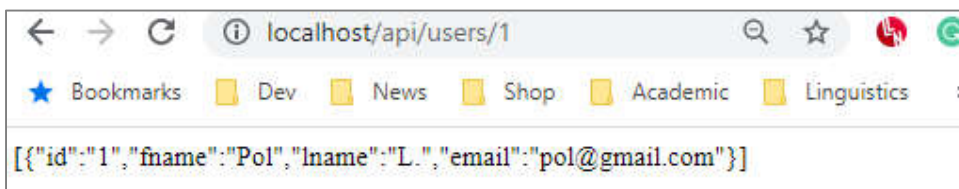
Code 7. api/user.php

```
<?php
class User{
    public $id;
    public $fname;
    public $lname;
    public $email;
    public function __construct($array){
        $this->id=$array[0];
        $this->fname=$array[1];
        $this->lname=$array[2];
        $this->email=$array[3];
    }
}
```

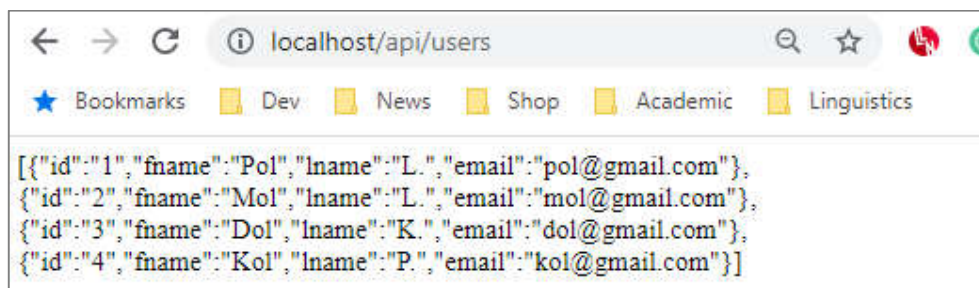
เมื่อทำได้ถึงตอนนี้แล้วก็พร้อมจะทดสอบการเรียกข้อมูลในลักษณะอยู่ URL ที่รองรับได้:

localhost/api/users/1
localhost/api/users

ผลที่ได้ก็จะมีผลในรูปต่อไปนี้



รูป 2 การเรียกรายการ user ที่มี id เป็น 1



รูป 3 การเรียกรายการ users

เตรียมแองกูลาร์ให้พร้อม

ดังที่เคยทำมาแล้ว ในบทที่ผ่านมา ในการอ่าน รายการของ user อาจใช้ของเก่ามาใช้งานอ่านโปรแกรมนี้ แล้ว ดัดแปลงค่า url ที่อ่าน หรือสร้างขึ้นมาใหม่ ในที่เลือกที่จะสร้างใหม่เพื่อเป็นการเริ่มต้นใหม่ มีลำดับดังนี้

สร้างโปรเจ็คใหม่ เลือกให้มีการสร้างเส้นทาง (Routing) และ ใช้รูปแบบ CSS ตามคำสั่ง ใน CLI ต่อไปนี้:

```
ng new myAngular --routing
cd myAngular
ng generate component users
ng generate component user
ng generate class user
ng generate service user
```

```
ng serve --open
```

สำหรับคลาส User ให้ปรับปรุงข้อมูล ให้สอดคล้องกับคลาส User.php ซึ่งจะเป็นโดเมน หรือข้อมูลกลุ่มหนึ่ง ดังตัวอย่างต่อไปนี้

Code 8. src/app/user.ts

```
export class User {  
  id: number;  
  fname: string;  
  lname: string;  
  email: string;  
}
```

อ่านรายการทั้งหมด

จากบทที่ผ่าน ๆ มา การอ่านข้อมูลเป็นงานบริการซึ่งเราควรสร้างให้เป็นหน้าที่ของคลาส UserService ไฟล์ user.service.ts เป็นไฟล์ที่เคยทำมาแล้วในบทที่ผ่านมา แต่สิ่งสำคัญ ที่เปลี่ยนแปลงคือ url

```
private usersUrl = 'http://localhost/api/users';
```

ใช้ http แบบเต็ม URL เพื่อเลี่ยงการชนกันของ เว็บเซิร์ฟเวอร์ของ แองกูลาร์ เอง แต่อย่างไรก็ตามยังใช้ localhost เพียงแต่ไม่ระบุเลขพอร์ต

Code 9. src/app/user.service.ts

```
import { Injectable } from '@angular/core';  
import { HttpClient, HttpHeaders } from '@angular/common/http';  
import { User } from './user';  
import { Observable, of } from 'rxjs';  
import { catchError, map, tap } from 'rxjs/operators';  
  
const httpOptions = {  
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })  
};  
  
@Injectable({  
  providedIn: 'root'  
})  
export class UserService {  
  private usersUrl = 'http://localhost/api/users'; // URL to web api  
  
  constructor(  
    private http: HttpClient,  
  ) { }  
  
  getUsers(): Observable<User[]> {  
    return this.http.get<User[]>(this.usersUrl)  
      .pipe(  
        tap( u => console.log("fetched users")),  
        catchError(this.handleError('getUsers', []))  
      );  
  }  
}
```

```

    private handleError<T> (operation = 'operation', result?: T) {
      return (error: any): Observable<T> => {
        console.error(operation+':' +error.message);
        return of(result as T);
      };
    }
  }
}

```

จากตัวอย่างโปรแกรมนี้ มีการนำเข้า HttpClientModule จึงทำให้ต้องเขียนนำเข้าโมดูลนี้ใน app.module.ts และประกาศในส่วน อาร์เรย์ imports ด้วย

Code 10. src/app/app.moudle.ts

```
import { HttpClientModule } from '@angular/common/http';
```

Code 11. src/app/app.moudle.ts

```
imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule
],
```

นอกจากนี้ยังไม่พอ ยังต้องกำหนดเส้นทาง ใน app-routing.module.ts โดยเขียนเพิ่มเติม 2 ที่ คือนำเข้าคอมโพเน้นท์ UsersComponent กับ userComponent และกำหนดเส้นทาง เส้นทางหลักเป็น users และเส้นทาง user/:id

Code 12. src/app/app-routing.module.ts

```
import { UsersComponent } from './users/users.component';
import { UserComponent } from './user/user.component';

const routes: Routes = [
  {path:'', redirectTo: '/users', pathMatch:'full'},
  {path:'users', component:UsersComponent},
  {path:'users/:id', component:UserComponent},
];
```

เมื่อต้องการแสดงผลรายการทั้งหมดที่หน้า usersComponent ก็ต้องเขียนการอ่านในคอมโพเน้นท์นี้ด้วยทั้งใน ไฟล์ TS และไฟล์ HTML ทั้งสองไฟล์นี้ ยังคงเขียนในรูปแบบเดิมตามบทที่ผ่านมา

Code 13. src/app/users/users.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from '../user';
import { UserService } from '../user.service';

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrls: ['./users.component.css']
})
export class UsersComponent implements OnInit {
  users: User[];
  constructor(
```



```

    private userService: UserService) { }
  ngOnInit() {
    this.getUsers();
  }
  getUsers():void{
    this.userService
      .getUsers()
      .subscribe(users => this.users = users);
  }
}

```

สำหรับหน้าแสดงผลรายการผู้ใช้ทั้งหมดเป็นโครงสร้างตาราง ที่แสดง id, fname, lname, email และคอลัมน์สุดท้ายแสดงรายการดำเนินการกับฐานข้อมูล ด้านการปรับปรุง และลบ

Code 14. src/app/users/users.component.html

```

<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First Name</th>
      <th scope="col">Last Name</th>
      <th scope="col">Email</th>
      <th scope="col">Operation</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of users">
      <th scope="row">{{user.id}}</th>
      <td>{{user.fname}}</td>
      <td>{{user.lname}}</td>
      <td>{{user.email}}</td>
      <td>
        <button routerLink='/users/{{user.id}}' class="btn">Edit</button>
        <button (click)= 'deleteUser(user.id)' class="btn">Delete</button></td>
      </tr>
    </tbody>
  </table>

```

มีข้อสังเกตว่า ตารางแสดงรายชื่อนี้ มีใช้ routerLink ไว้ด้วย เพื่อเชื่อมโยงไปยัง หน้า userComponent ไว้ด้วย และถ้ากำหนดผ่านรูปแบบ CSS ของ Bootstrap ด้วย เมื่อให้ url ไปยัง <http://localhost:4200/users> จะมีหน้าตาเว็บที่ได้ดังนี้

ข้อมูลเหมือนกับที่ภาพก่อนหน้านี้นี้ แต่ใช้งานผ่าน แอ่งกูลาร์ และจัดหน้าเว็บใช้ Bootstrap ดังที่เคยทำในบทที่แล้ว แต่ในบทที่แล้วได้จบการอธิบายไว้เพียงแค่การอ่าน ยังมีการดำเนินอื่น ๆ อีกมาก

Welcome to myAngular!

| # | First Name | Last Name | Email | Operation | |
|---|------------|-----------|-------------------------|-----------|--------|
| 1 | Pol | L. | pol@gmail.com | Edit | Delete |
| 2 | Mol | L. | mol@gmail.com | Edit | Delete |
| 3 | Dol | K. | dol@gmail.com | Edit | Delete |
| 4 | Kol | P. | kol@gmail.com | Edit | Delete |
| 5 | theerapol | limsatta | theerapol.lim@gmail.com | Edit | Delete |

รูป 4 การเรียกรายการ users ผ่าน แอ่งกูลาร์

อ่านข้อมูลที่ส่งกลับมามีค่าเดียว

ทุกฟังก์ชันของ **HttpClient** จะคืนค่า RxJS Observable ของข้อมูล ขณะที่ HTTP เป็นโปรโตคอล ร้องขอและตอบรับ ซึ่งการร้องขอหนึ่งครั้งก็จะมีคำตอบรับหนึ่งครั้ง แต่การตอบรับหนึ่งครั้งนี้จะคืนค่าได้มาหลายค่า โดยที่ HttpClient รับค่าหลายค่าอยู่ในรูปอาร์เรย์ ด้วยฟังก์ชัน `get()` และรับข้อมูลอยู่ในรูปอาร์เรย์ของ `User[]` ซึ่งสอดคล้องกับภาษา PHP ที่ส่งข้อมูลออกมาในรูปอาร์เรย์ของ JSON

การอ่านค่ารหัสเป็นงานหนึ่งที่เว็บทั่วไปทำได้ ตามคำสั่งใน URL: `baseUrl/:id` ในที่นี้ `baseUrl` คือ `usersURL` ที่ได้กำหนดไว้ก่อนหน้านี้ มีค่า `'api/users/'` และ เมื่ออ่านเฉพาะรหัส เช่น `api/users/1` ก็จะหมายถึงอ่านค่ารหัส 1 แม้จะอ่านเพียงข้อมูลเดียว แต่การอ่านในรูปอาร์เรย์ ดังเพิ่มฟังก์ชัน `getUser(id: number)` เข้าไปดังนี้

Code 15. src/app/user.service.ts

```
getUser (id: number): Observable<User[]> {
  const url = `${this.usersUrl}/${id}`;
  return this.http.get<User[]>(url).pipe(
    tap( _ => console.log(`fetched user id=${id}`)),
    catchError(this.handleError<User[]>('getUser'))
  );
}
```

แต่สำหรับคลาส `UserComponent` จะใช้บริการอ่านค่าข้อมูลค่าเดียวแต่ส่งข้อมูลในรูปอาร์เรย์ ดังนั้นข้อมูลเดียวก็คือค่าข้อมูลที่อาร์เรย์ศูนย์นั่นเอง

ฟังก์ชัน `getUser()` ส่วนการรับข้อมูล (subscribe) ตัวแปร `users` ซึ่งเป็นรูปอาร์เรย์ ได้ถูกอ่านข้อมูลที่อาร์เรย์ศูนย์แล้วกำหนดค่าให้ ตัวแปร `user` ซึ่งจะเป็นข้อมูลที่นำไปแสดงผลในไฟล์ HTML ต่อไป

Code 16. src/app/user/user.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Location } from '@angular/common';
```

```

import { User } from '../user';
import { UserService } from '../user.service';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrls: ['./user.component.css']
})
export class UserComponent implements OnInit {
  user: User;
  constructor(
    private route: ActivatedRoute,
    private userService: UserService,
    private location: Location
  ) { }
  ngOnInit(): void {
    this.getUser();
  }
  getUser(): void {
    const id = +this.route.snapshot.paramMap.get('id');
    this.userService
      .getUser(id)
      .subscribe(users => this.user = users[0]);
  }
}

```

ตัวแปรที่อ้างอิงได้เพื่อแสดงผลได้คือ user ในการแสดงผล ในครั้งแรกต้องตรวจสอบก่อนว่ามีหรือไม่ เพราะบางที่ข้อมูลยังไม่ครบ หรือไม่พร้อมจะแสดง การตรวจสอบในที่นี้เลือกใช้คำสั่ง ngIf ตรวจสอบ user ให้ไม่เท่ากับ null กรณีที่เป็นค่า null ก็ให้แสดงผลในอีลิเมนต์ elseBlock

ตัวอย่างต่อไปนี้จะใช้ CSS ของ Bootstrap สร้างรูปแบบแสดงผลร่วมด้วย หากไม่ต้องการใช้ ร่วมกับ Bootstrap ก็จะเป็นการแสดงผลที่ไม่สวยงามอะไร ซึ่งก็ไม่ได้ผิดอะไร

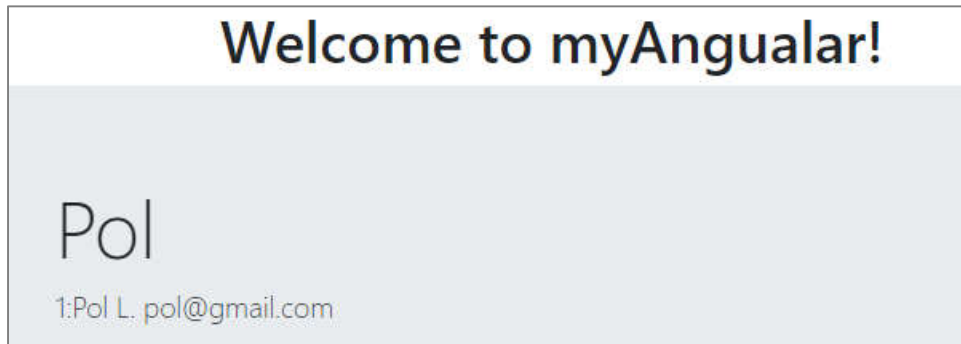
Code 17. src/app/user/user.component.html

```

<div class="jumbotron jumbotron-fluid"
  *ngIf="user != null; else elseBlock">
  <div class="container">
    <h1 class="display-4">{{user.fname}}</h1>
    <p class="lead">{{user.id}}:{{user.fname}} {{user.lname}} {{user.email}}</p>
  </div>
</div>

<div>
<ng-template #elseBlock
  class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-4">Empty</h1>
    <p class="lead">Noone</p>
  </div>
</ng-template>

```



รูป 5 การแสดงรายการเดี่ยวของ user

ปรับปรุงข้อมูล Users

การทำให้มีการปรับปรุงข้อมูลได้ จะต้องสร้างฟังก์ชันให้รับเหตุการณ์จากปุ่มคำสั่งได้ ซึ่งจะต้องสร้างปุ่มคำสั่งเป็นภาษา HTML ในไฟล์ user.component.html ดังเพิ่มการสร้างปุ่มคำสั่งรองรับเหตุการณ์คลิก ภายในอีลีเมนต์ ของคลาส jumbotron

สำหรับฟอร์มกรอกข้อมูล โดยผู้ข้อมูล user ทั้งสามฟิลด์ (fname, lname, email) โดยฟอร์มนี้ ขณะเริ่มต้นให้ซ่อน ฟอร์มนี้จนกว่าผู้ใช้จะคลิก ปุ่ม Edit

Code 18. src/app/user/user.component.html

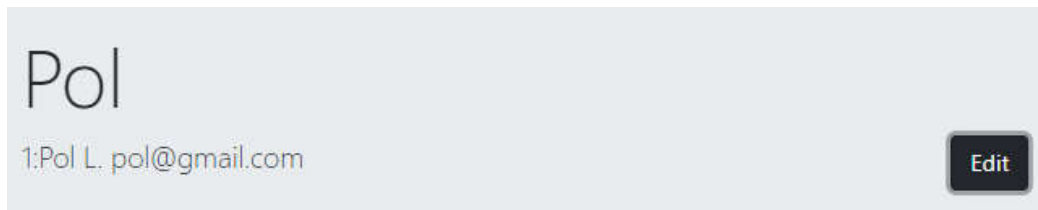
```
<div class="jumbotron jumbotron-fluid" *ngIf="user != null; else elseBlock">
  <div class="container">
    <h1 class="display-4">{{user.fname}}</h1>
    <p class="lead">{{user.id}}:{{user.fname}} {{user.lname}} {{user.email}}
      <button style="float:right" id='btn-edit'(click)='showForm()'
        class="btn btn-dark">Edit</button>
    </p>
  </div>
  <form style="display:none" (ngSubmit)='onSubmit()'>
    <div class='form-group row'>
      <label class='col-sm-2 col-form-label'>First Name</label>
      <div class="col-sm-10">
        <input class='form-control' placeholder='first name'
          type='text' [(ngModel)]='user.fname' name="fname">
      </div>
    </div>
    <div class='form-group row'>
      <label class='col-sm-2 col-form-label'>Last Name</label>
      <div class="col-sm-10">
        <input class='form-control' placeholder='last name'
          type='text' [(ngModel)]='user.lname' name="lname">
      </div>
    </div>
    <div class='form-group row'>
      <label class='col-sm-2 col-form-label'>Email</label>
      <div class="col-sm-10">
        <input class='form-control' placeholder='email'
          type='email' [(ngModel)]='user.email' name="email">
      </div>
    </div>
  </form>
```

```

    <div class='form-group row'>
      <span class='col-sm-2'></span>
      <div class="col-sm-10">
        <button type="submit" class="btn btn-dark"> Save </button>
      </div>
    </div>
  </form>
</div>
<ng-template #elseBlock
  class="jumbotron jumbotron-fluid">
    <div class="container">
      <h1 class="display-4">Empty</h1>
      <p class="lead">No one</p>
    </div>
  </ng-template>

```

การใช้งานฟอร์ม ต้องนำเข้าทั้งในคอมโพเนนต์ (user.component.ts) และแอปโมดูล (app.module.ts) ด้วย (ดูรายละเอียดในบทที่ว่าด้วยเรื่อง ฟอร์ม-เทมเพลต)



รูป 6 เพิ่มปุ่ม Edit และซ่อนฟอร์ม

จากตัวอย่างโปรแกรมนี้ ปุ่ม Edit ได้ผูกเหตุการณ์ click ไว้กับฟังก์ชัน showForm() ใช้ไลบรารีของ JQuery โดยต้องนำเข้าไลบรารีนี้ก่อน (การใช้งาน JQuery ได้อธิบายไว้ในบท ที่ว่าด้วยคอมโพเนนต์) และเพิ่มการอ้างอิงที่หน้า index.html โดยวางไว้หลัง <app-root>

จากตัวอย่างต่อไปนี้ เลือกใช้ ไลบรารี ผ่าน https ของบริษัท ไมโครซอฟต์ ซึ่งการใช้งานนั้นจำเป็นต้องต่อกับอินเทอร์เน็ตได้

Code 19. src/index.html

```

<body>
  <app-root></app-root>
  <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.4.1.min.js">
  </script>
</body>

```

Code 20. src/app/user/user.component.ts

```

import * as $ from 'jquery';

```

ฟังก์ชัน showForm() เลือกตรวจสอบ ข้อความของปุ่ม ถ้ามีข้อความว่า Edit ให้เปลี่ยนข้อความของปุ่มเป็น Hide ต่อมา ใช้ฟังก์ชัน toggle() ให้ แสดง หรือซ่อนฟอร์ม

Code 21. src/app/user/user.component.ts

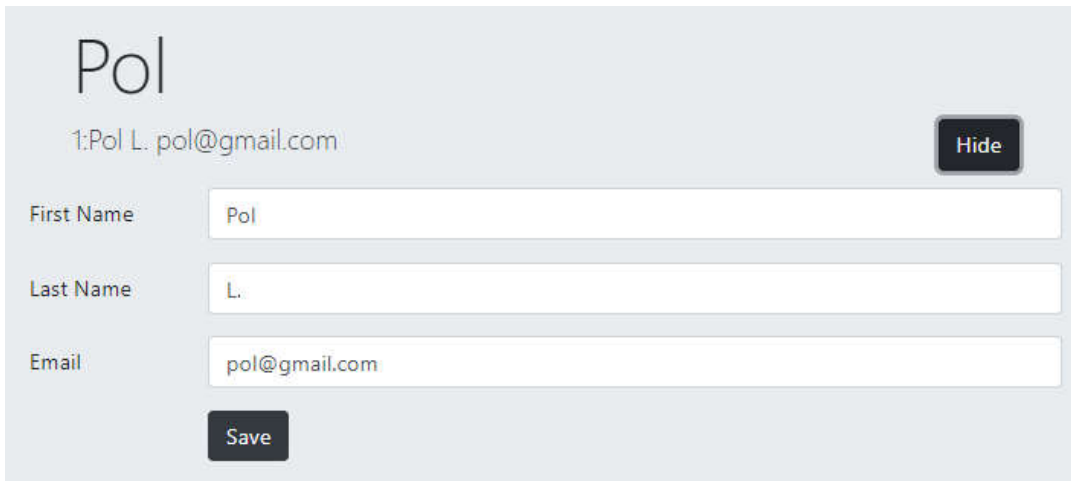
```
showForm():void{
  if($('#btn-edit').text()=='Edit')
    $('#btn-edit').text('Hide');
  else
    $('#btn-edit').text('Edit');

  $('#form').css('margin','20px').toggle('slow','linear');
}
```

สำหรับปุ่ม Save เรียกงานบริการ updateUser() และเรียนฟังก์ชัน showForm() อีกครั้งเพื่อซ่อนฟอร์ม หลังสมัครใช้บริการนี้แล้ว (subscribe())

Code 22. src/app/user/user.component.ts

```
onSubmit():void{
  this.userService.updateUser(this.user)
    .subscribe();
  this.showForm();
}
```



รูป 7 เมื่อคลิกปุ่ม Edit

งานบริการ userService.updateUser(this.user) ที่มีเรียกใช้ onSubmit() ตัวแปรเป็นข้อมูล user ที่จะปรับปรุงโครงสร้างการทำงานคล้าย ๆ กับ ฟังก์ชัน getUsers() แต่เลือกใช้ http.put() เพราะ API ที่เราสร้างเองรองรับ การปรับปรุงข้อมูลด้วย put()

ฟังก์ชัน put() ต้องใส่ตัวแปรเข้า 3 ตัวคือ URL, ข้อมูลที่จะปรับปรุง และข้อมูลทางเลือกอื่น ๆ ใช้ชื่อตัวแปรว่า httpOptions

Code 23. src/app/user.service.ts

```
updateUser(user:User): Observable<any>{
  const url = `${this.usersUrl}/${user.id}`;
  return this.http.put(url, user, httpOptions)
    .pipe(
      tap( u => console.log(`updated user id=${user.id}`)),
      catchError(this.handleError<any>('updateUser'))
    )
}
```

```
);
};
```

สำหรับตัวแปร httpOptions เป็นข้อมูลที่จะส่งไปกับ http Header หรือส่วนแรกของคำขอบริการ ซึ่งจะใช้ระบุว่า มีคำขอบริการที่ส่งไฟล์ไปรูปแบบใด ในที่นี้ ส่งไปในรูปแบบ JSON จึงต้องกำหนดค่าตัวแปรทางเลือกลงนี้ตามเป็นค่าคงที่นี้ก่อน บรรทัด @Injectable

Code 24. src/app/user.service.ts

```
const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};
```

ฝั่งเซิร์ฟเวอร์ ปรับปรุงข้อมูล

ที่ฝั่งเซิร์ฟเวอร์ จะทำหน้าที่อ่านข้อมูล ที่ แอ่งกูลาร์ ส่งข้อมูลมาในรูปแบบ JSON เมื่อข้อมูลส่งมา PHP จะอ่านค่าได้เป็น อาร์เรย์แบบมีคีย์ (Associative Array) ในกรณีนี้ ของข้อมูล JSON ที่ส่งมาจะอยู่ในรูป (ถ้าเป็นข้อมูลที่ส่งเป็นรายการแรก)

```
{id:1, fname:'pol', lname:'L.', email:'pol@gmail.com'}
```

แต่ เซิร์ฟเวอร์จะแปลงได้เป็น

```
{'id'=>1, 'fname'=>'pol', 'lname'=>'L.', 'email'=>'pol@gamil.com'}
```

ข้อมูลนี้จะอยู่ในตัวแปร \$this->data โมเดลฐานข้อมูลคือ users จากคลาสฐานข้อมูล usersDataGateway ในไฟล์ RESTfu.php ต้องสร้างฟังก์ชัน put() เพื่อรองรับการปรับปรุงข้อมูลดังเขียนได้ดังนี้

Code 25. api/RESTful.php

```
private function put(){
    $model = $this->args[0];
    $class_data_model = $model.'DataGateway';
    $object_data_model = new $class_data_model();
    $object_data_model->update($this->data);
}
```

คลาส UsersDataGateway จะต้องมีการฟังก์ชัน update(\$array) ตามการเรียกใช้งานของ RESTful.php ภาษา SQL สำหรับปรับปรุงข้อมูล คือ

```
"update user set fname=?, lname=?, email=? where id=?"
```

จะเห็นว่าตัวแปร มีลำดับการเรียง เริ่มจาก fname, lname, email, id ตามลำดับ จะเห็นว่า ลำดับนี้ ไม่ตรงข้อมูลที่ส่งมา (\$this->data) จึงต้องเรียงข้อมูลใหม่ให้อยู่ในลำดับภาษา SQL นี้ แต่มีวิธีที่ดีกว่า คือไม่ต้องเรียงใหม่ แต่ใช้การใส่ตัวแปรตามชื่อ แต่ต้องให้เป็นชื่อให้ตรงกับ อาร์เรย์ที่ส่งเข้ามา ดังนั้นภาษา SQL ที่ระบุชื่อตัวแปรตามแบบ PDO จะเขียนใหม่ได้ดังนี้

Code 26. api/usersDataGateway.php

```
public function update($array){
    $sql = "update user set fname=:fname, lname=:lname, email=:email where id=:id";
```

```

$stmt = $this->pdo->prepare($sql);
$stmt->execute($array);
}

```

ถึงตอนนี้ให้ทดลอง ปรับปรุงข้อมูล เพื่อดูผลการปรับปรุง โดยแก้ไขข้อมูล และกดปุ่ม SAVE แล้วผลการทำงานการปรับปรุงข้อมูล

แทรก user รายใหม่

การสร้าง user รายใหม่ โดยการแทรกของระบบข้อมูล จะต้องสร้างส่วน html ให้มี <input> ข้อมูลใหม่ได้พร้อมกับสร้างปุ่มคำสั่งให้บันทึกได้

ให้ทำการเพิ่มตัวอย่างโปรแกรมต่อไปนี้ในส่วนบนสุดของไฟล์ users.component.html เดิม

Code 27. src/app/users/users.component.html

```

<tbody>
<!-- แทรก แถว ( <tr>) ต่อไปนี้เป็นแถวสุดท้ายของตาราง -->
  <tr>
    <td><button (click)= 'insertUser()'
      class="btn btn-dark"> Add New </button></td>
    <td><input class='form-control' placeholder='fist name'
      type='text' [(ngModel)]="new_user.fname" name="fname"></td>
    <td><input class='form-control' placeholder='lname'
      type='text' [(ngModel)]="new_user.lname" name="lname"></td>
    <td><input class='form-control' placeholder='email'
      type='email' [(ngModel)]="new_user.email" name="email"></td>
  </tr>
</tbody>

```

ในส่วนของการเหตุการณ์คลิก เพื่อบันทึกนั้น ใช้ฟังก์ชัน insertUser() ในส่วนฟังก์ชันนี้จะต้องนิยามในไฟล์ คอมโพเนนท์ โดยเพิ่มฟังก์ชันนี้ ดังตัวอย่างต่อไปนี้

Code 28. src/app/users/users.component.ts

```

insertUser():void{
  this.new_user.fname = this.new_user.fname.trim();
  this.new_user.lname = this.new_user.lname.trim();
  this.new_user.email = this.new_user.email.trim();
  this.userService
    .insertUser(this.new_user)
    .subscribe(result=>{
      if(result>0) alert('Insert:Complete');
      else alert('Insert: Incomplete');
    });
  window.location.reload();
}

```


จากฟังก์ชันนี้ มีการตัดค่าที่เป็นช่องว่าง หน้า-หลัง ของ new_user ออกก่อน (trim) ต่อด้วยเรียกใช้บริการ userService.insert() และตามด้วยรับใช้บริกาณ์ (subscribe) ภายในรับใช้บริกาณ์นี้ยังรับข้อมูลตอบกลับจาก เซิร์ฟเวอร์ หากมีผลมากกว่าศูนย์ แสดงว่าแทรกข้อมูลสำเร็จ แต่ถ้าไม่ใช้ก็จะแสดงว่าไม่สำเร็จ

สำหรับตัวแปร new_user เป็นตัวแปรใหม่ ต้องเพิ่มในคอมโพเน้นท์นี้ด้วย และสร้างเป็นออบเจกต์ ในฟังก์ชัน ngOnInit() ด้วย

Code 29. src/app/users/users.component.ts

```
new_user:User
ngOnInit() {
  this.getUsers();
  this.new_user = new User();
}
```

ในงานบริการของ user จะต้องเพิ่มฟังก์ชัน insertUser() ตามการเรียกใช้ใน คอมโพเน้นท์ ในครั้งนี้ยังคงใช้รูปแบบ RESTful ซึ่งการแทรกใช้คำกริยา post การเพิ่มนี้มีแต่ชื่อ สกุล และอีเมล ส่วนรหัส เซิร์ฟเวอร์จะมีการสร้างเองอัตโนมัติ

Code 30. src/app/user.service.ts

```
insertUser(user:User): Observable<any>{
  const url = `${this.usersUrl}`;
  return this.http.post(url, user, httpOptions)
    .pipe(
      tap( u =>
        console.log(`insert ${user.fname} ${user.lname} ${user.email}`)),
      catchError(this.handleError<any>('insertUser'))
    )
};
```

ฟังก์ชันเพิ่มรายชื่อในเซิร์ฟเวอร์

ฟังก์ชัน insert() ที่คลาส UsersDataGateway ทำหน้าที่แทรกข้อมูลรายใหม่ โดยจะต้องอ่านรหัสเดิมที่มีค่ามากที่สุดมาก่อน แล้วเพิ่มอีกหนึ่งเพื่อทำเป็นรหัสใหม่ ต่อมาจึงแทรกข้อมูลทั้งหมด สุดท้าย ให้คืนค่าผลการแทรก ใช้การนับผลการแทรก ซึ่งจะได้เลข 1 ถ้าแทรกได้ แต่จะคืนเลข 0 ถ้าแทรกไม่ได้ ดังตัวอย่างต่อไปนี้

แต่มีอย่างหนึ่งที่น่าสังเกตว่า โครงสร้างตารางฐานข้อมูลนี้ใช้ การเพิ่มค่าได้เอง (auto increment) ให้กับค่า id ซึ่งไม่จำเป็นจะต้องทำให้อ่านค่าเองก็ได้ แต่อย่างไรก็ตาม ถ้าเราจะเขียนให้เพิ่มตามที่เรต้องการเองก็ได้ ดังตัวอย่างนี้ ใช้วิธีเพิ่มค่าโดยการเขียนขึ้นเอง

Code 31. api/usersDataGateway.php

```
public function insert($array){
  $sql = "SELECT id FROM user ORDER BY id DESC LIMIT 1;";
  $result = $this->pdo->query($sql);
  $last_id = $result->fetchColumn();
  $last_id++;
  $array['id'] = $last_id;

  $sql = "insert into user (id, fname, lname, email) ";
```

วิทยาลัยเซาธ์อีสท์บางกอก เอกสารประกอบการสอน การเขียนโปรแกรม Angular/AngularJs

```

        $sql .= " values(:id,:fname,:lname,:email)";
        $stm = $this->pdo->prepare($sql);
        $stm->execute($array);
        return $stm->rowCount();
    }

```

สำหรับ ฟังก์ชัน post() ที่รองรับการรับข้อมูลผ่าน Method: POST จาก แองกูลาร์ ซึ่งอยู่ที่คลาส RESTful ในการทำงานรับข้อมูลการแทรก มีแตกต่างกับฟังก์ชันที่ผ่านมาคือ มีการคืนค่าในรูปผลลัพธ์ผลการแทรก คือ ถ้าแทรกได้ผลจะคืนค่า 1 จากฟังก์ชัน insert() จากตัวอย่างโปรแกรมที่ผ่านมา

Code 32. api/RESTful.php

```

private function post(){
    $model = $this->args[0];
    $class_data_model = $model.'DataGateway';
    $object_data_model = new $class_data_model();
    $result = $object_data_model->insert($this->data);
    return $result;
}

```

ถึงตอนนี้ก็พร้อมทดสอบการแทรกข้อมูลใหม่แล้ว ดังรูปต่อไปนี้

| # | First Name | Last Name | Email | Operation | |
|---|------------|-----------|-------------------------|-----------|--------|
| 1 | Pol | L. | pol@gmail.com | Edit | Delete |
| 2 | Mol | L. | mol@gmail.com | Edit | Delete |
| 3 | Dol | K. | dol@gmail.com | Edit | Delete |
| 4 | Kol | P. | kol@gmail.com | Edit | Delete |
| 5 | theerapol | limsatta | theerapol.lim@gmail.com | Edit | Delete |

รูป 8 หน้าตาใหม่ของ /users ที่มีการแทรกข้อมูลใหม่ได้

ลบข้อมูล user

ข้อมูล user แต่ละตัวควรลบได้ด้วยปุ่มคำสั่ง เพื่อการนี้ให้เพิ่มปุ่มคำสั่งไปยังหน้า HTML ของคอมโพเนนต์ UsersComponent ให้ปรับปรุงโดยเพิ่มการผูกกับเหตุการณ์ click กับฟังก์ชัน deleteUser(user.id) ตัวแปรเข้าเป็น id เพื่อให้เป็นการลบตาม id

Code 33. src/app/users/users.component.html

```

<button (click)= 'deleteUser(user.id)' class="btn">Delete</button></td>

```

ปุ่มคำสั่งนี้ ให้เพิ่ม การลบ ตามฟังก์ชัน deleteUser() และให้อ่านหน้าเว็บซ้ำอีกครั้ง เพื่อให้เห็นได้ว่าการลบไป
ออกไปจริงๆ

Code 34. src/app/users/users.component.ts

```
deleteUser(id: number): void {  
    alert("Delete user id: "+ id);  
    this.userService.deleteUser(id).subscribe( );  
    window.location.reload();  
}
```

การลบนี้จะไม่เกิดขึ้นจริง เพราะจะต้องเขียนโปรแกรมเพิ่มในส่วนของงานบริการนี้ก่อน ให้สังเกตว่า การใช้บริการนี้
จะต้องลงทะเบียนสมัครด้วย subscribe() เพราะมีกฎ ว่า Observable จะไม่ทำอะไร จนกระทั่งผ่านการสมัครก่อน
ให้ทำการเพิ่มโปรแกรมงานบริการการลบไปยัง users.service.ts ดังนี้

Code 35. src/app/users.service.ts

```
deleteUser(id: number): Observable<any> {  
    const url = `${this.usersUrl}/${id}`;  
    return this.http.delete(url)  
        .pipe(  
            tap(_ => console.log(`deleted user id=${id}`)),  
            catchError(this.handleError<any>('deleteUser'))  
        );  
}
```

ให้สังเกตว่า บริการ http.delete() เป็นฟังก์ชัน ที่ส่งไปยังเซิร์ฟเวอร์ ที่รองรับการทำงานแบบ delete สำหรับ
RESTful ซึ่งจะลบข้อมูล ตามรูปแบบ URL: usersUrl/id

เขียนการลบบนเซิร์ฟเวอร์

เมื่อโคลเอ็นท์ส่งค่าขอมาในรูป api/users/id มีตัวแปรที่ 0 เป็น users และตัวแปรที่ 1 เป็น id ในคลาส Resful
จึงต้องมีฟังก์ชัน delete(\$id) ตัวแปรเข้าได้ตัดออกมาให้แล้วในฟังก์ชัน apply() ใส่มาให้ฟังก์ชัน delete(\$id) โดยตรง ส่วนตัว
แปรที่ 0 ตัดออกมาเป็นส่วนหนึ่งของคลาส UserDataGateway เพื่อใช้ในการสร้างออบเจกต์ และเรียกฟังก์ชัน
deleteById(\$id)

Code 36. api/RESTful.php

```
private function delete($id){  
    $model = $this->args[0];  
    $class_data_model = $model.'DataGateway';  
    $object_data_model = new $class_data_model();  
    $object_data_model->deleteById($id);  
}
```

ฟังก์ชัน deleteById(\$id) ทำงานเพื่อลบออกจากฐานข้อมูลจริง

Code 37. api/usersDataGateway.php

```

public function deleteById($id){
    $sql = "delete from user where id=?";
    $stm = $this->pdo->prepare($sql);
    $stm->bindParam(1, $id);
    $stm->execute();
}

```

สรุป

ยาวนานก็คุ้มค่า สำหรับการอ่านทั้งหมด อ่านบางรายการ ปรับปรุงข้อมูล ลบข้อมูล ในรูปแบบ RESTful ซึ่งจะพบงานเว็บแบบนี้มากมายในฐานะงานบริการ หรือเว็บเซอร์วิส (Web service) หรือเราอาจไม่ต้องสร้างแบบเซอร์วิสเอง เพียงแค่รู้จักการใช้จากผู้ให้บริการเว็บเซอร์วิส แต่เราจำเป็นต้องรู้จักการติดต่อกับเว็บเซอร์วิสผ่านเครื่องไคลเอ็นต์อย่างไร แน่นอนว่าเราทำได้แล้วจากบทนี้ เราได้ใช้แองกูลาร์สร้างตัวขอบริการจากเว็บเซอร์วิส ครบทุกบริการใน RESTful และเราก็รู้จักสร้างเว็บเซอร์วิสของตัวเองได้ด้วย

คำถามทบทวน

1. RESTful รองรับบริการขอบริการได้แบบใดบ้าง
2. ข้อมูล JSON มีลักษณะ { email: 'theerapol.lim@gmail.com', password: 'password' } ข้อมูลถอดรหัสได้ใน PHP จะเป็นเช่นไร
3. ไฟล์ .htaccess ทำหน้าที่อะไร
4. จากคลาส RESTful ตัวแปร \$args[0] และ \$args[1] ใช้แทนอะไร
5. การส่งข้อมูลแบบปรับปรุงข้อมูลต้องร้องคำขอไปยังเซิร์ฟเวอร์แบบ RESTful จะร้องขอไปด้วยคำกริยาใด
6. Content-Type สำหรับข้อมูล JSON ต้องเป็นแบบใด
7. การสร้างตัวแปรเข้าฟังก์ชัน (ในแบบแองกูลาร์) ให้รับข้อมูลได้ทั้งสองไทป์ จะเขียนอย่างไร ถ้าชื่อฟังก์ชันคือ inset () ตัวแปรเข้าเป็นออบเจกต์ user หรือ number
8. ข้อมูล JSON { 'id':1, 'name': 'pol' } เมื่อแปลงเป็นข้อมูลเป็นภาษา PHP จะอยู่ในรูปแบบใด
9. การในการโหลดข้อมูลหน้าเว็บเข้าใช้วิธีการใด
10. การใส่ข้อมูลของแองกูลาร์ ถ้าไม่ได้ส่งไปแบบ JSON คือต้องการส่งแบบข้อความทั่วไป จะต้องทำอย่างไร

แบบฝึกหัด

1. จากตัวอย่างทั้งหมดขาดการลบรายการข้อมูล
2. สร้างการทำงานของแองกูลาร์ ในรูปแบบ RESTful ที่รองรับการอ่านข้อมูลสินค้าทั้งหมด และอ่านตามรหัส ให้ชื่อตารางบน MySQL ว่า product ภายในตารางประกอบด้วย id, name, price กำหนดไทป์ และกำหนดข้อมูลตัวอย่างในตารางได้เองตามความเหมาะสม

อ่านเพิ่มเติม

PHPenthusiast.(4 Apr.2020). Angular app with PHP:
<https://phpenthusiast.com/blog/develop-angular-php-app-getting-the-list-of-items>