HTTP

ในบทนี้ เราจะทำความรู้จักกับ งานบริการ HTTP เพื่อดำเนินการกับข้อมูลระยะไกลกับเว็บเซิร์บเวอร์ ในลักษณะต่าง ๆ ดังนี้

- ให้ UserService อ่านข้อมูลด้วย การร้องขอบริการผ่าน HTTP
- สามารถสืบค้นข้อมูลโดยการผ่านตัวแปรกับ HTTP และจัดการกับออบเจ็กต์ ตัวให้สังเกตการณ์
- สามารถจัดการความผิดพลาดทั้งระดับตัวแปรภายในฟังก์ชันและระดับโมดูล

สิ่งต้องเตรียม คือแอปพลิเคชันเริ่มต้น ในชื่อ myAngular เป็นไฟล์ที่สร้างขึ้นมาใหม่หมด สร้างแบบมีเส้นทาง (Router) หรือไม่มีเส้นทางก็ได้ หรือจะใช้แอปพลิเคชันที่เคยสร้างจากบทที่ผ่านมาก็ได้ นอกจากนี้ต้องมีเว็บเซิร์บเวอร์ Apache Web Sever ที่มาพร้อมกับตัวติดตั้งโปรแกรมรวมมิตรอย่าง XAMPP

การใช้บริการ HTTP

HttpClient ของ Angular ใช้สำหรับการสื่อสารระยะไกลกับเซิร์บเวอร์ผ่านโปรโตคอลสื่อสาร HTTP เราจะใช้บริการนี้สามารถใช้งานได้ทุก ๆ ของเว็บ เราจะต้องเพิ่มนำเข้าโมดูล AppModule ที่ไฟล์โมดูลหลัก

Code 1. src/app/app.module.ts

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

แล้วทำการเพิ่มโมดูลนี้ในอาร์เรย์ของ @NgModule หาส่วนที่เป็น import**s** แล้วให้เพิ่ม HttpClientModule ไว้ตำแหน่งใดก็ได้ แต่ละตัวของอาร์เรย์ จะมีเครื่องหมาย จุลภาค คั้นกลาง

Code 2. src/app/app.module.ts

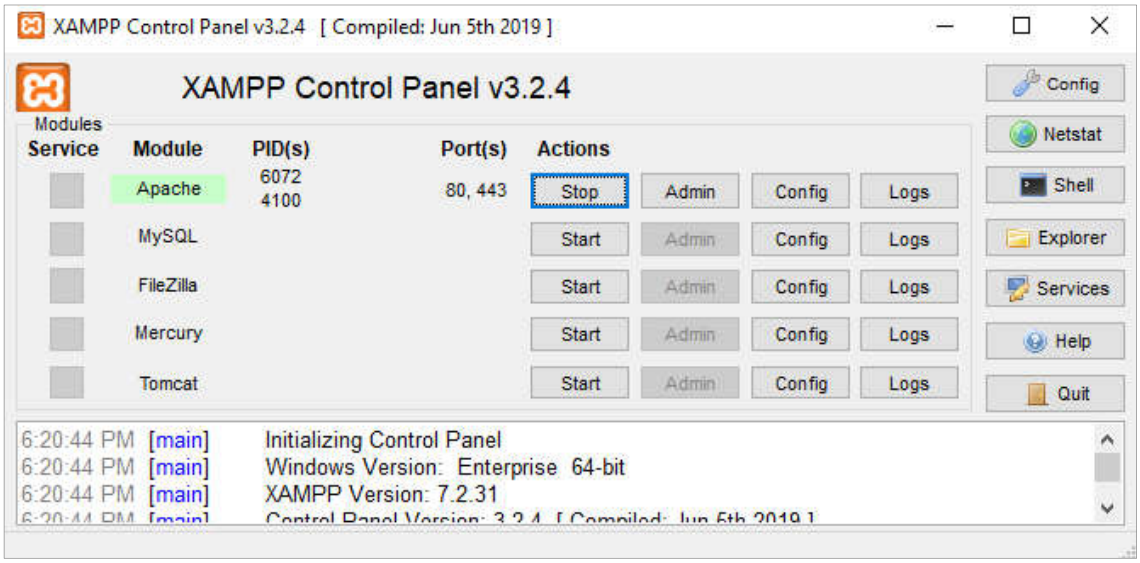
```
imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
],
```

สร้าง Web API

เนื่องจากในการทดลองของบทนี้ใช้ การจำลองจากเซิร์บเวอร์ที่ Apache Web Server โดยสมมุติให้เว็บเซิร์บเวอร์มีข้อมูลเป็นรายการผู้ใช้ ประมาณ 5 ราย ดังนั้นจำเป็นต้องมีเว็บเซิร์บเวอร์ติดตั้งที่เครื่องที่ทดสอบด้วย อาจใช้โปรแกรม XAMPP[1] หลังจากได้ติดตั้งให้เปิดหน้าต่าง XAMPP Control Panel และคลิกเลือก Start ที่ Apache

สำหรับฐานข้อมูลใช้การจำลองในจากไฟล์ PHP ไม่ได้มาจากฐานข้อมูลจริง จึงไม่จำเป็นต้องเปิด MySQL สำหรับ MySQL จะอธิบายการทำงานกับฐานข้อมูลนี้ในบทต่อไป

[1] ดาวน์โหลด ที่ : https://www.apachefriends.org/download.html



รูป 1 หน้าต่าง XAMPP Control Panel

สร้างทางเข้าเซิร์บเวอร์

เพื่อให้ทุกคำขอมารวมอยู่ที่จุดเดียวกัน เซิร์บเวอร์ของ Apache ให้เขียนคำสั่งการเข้าถึงในไฟล์ .htaccess (ไฟล์ไม่มีชื่อแต่มีนามสกุล) ในคำสั่งนี้ ให้ทุกคำขอบริการผ่าน port:4200 ได้

Code 3. xampp/httdocs/myApi/.htaccess

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f

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"
```

บรรทัดที่ขึ้นต้นด้วย Header ซึ่งมีสามบรรทัด ในบรรทัดที่สองของ Header ยาวจนขึ้นบรรทัดใหม่ตามตัวอย่างโปรแกรมนี้ ให้ทำเป็นบรรทัดเดียวกัน ยกบรรทัด Original ขึ้นไปต่อท้ายบรรทัดบน เพราะบางที่ Apache อาจทำงานผิดพลาด

สร้างฐานข้อมูลจำลอง

เมื่อการทดลองทำงานกับฐานข้อมูล การสร้างข้อมูลจำลอง ในเว็บเซิร์บเวอร์ ให้สร้างในโฟลเดอร์ myApp และใส่ไฟล์ db.php

หลังการสร้างจะได้ไฟล์ db.php ให้เพิ่มข้อมูลในรูปอาร์เรย์ ชื่อ users จากคลาส User ซึ่งข้อมูล users จะกลายเป็นข้อมูลจำลองที่ใช้การศึกษาบทนี้

Code 4. xampp/httdocs/myApi/db.php

```
<?php
class User{
 public $id;
 public $fname;
 public $lname;
```

```
 public $email;
 public function __construct($id, $fname, $lname, $email){
       $this->id=$id;
       $this->fname=$fname;
       $this->lname=$lname;
       $this->email=$email;
 }
}

$users = array(new User(1, 'Tee', 'L.', 'tee@sbc.com'),
               new User(2, 'RA', 'T.', 'ra@sbc.com'),
               new User(3, 'Pol', 'K.', 'pol@sbc.com'),
               new User(4, 'Lim', 'M.', 'lim@sbc.com'),
               new User(5, 'Satta', 'C.','satta@sbc.com'));
```

ต่อมาให้สร้างไฟล์ users.php โดยไฟล์นี้ทำหน้าที่ส่งข้อมูล users ทั้งหมดออกไปในรูปแบบ JSON ดังตัวอย่างต่อไปนี้

Code 5. xampp/httdocs/myApi/users.php

```
<?php
include 'db.php';
echo json_encode($users);
```

ถึงตอนนี้ก็สามารถทดสอบการทำงานได้แล้ว โดยไปที่ URL : localhost/myApp/users.php ซึ่งจะได้ผลออกมาในรูปของอาร์เรย์ของ JSON

รูป 2 ผลการอ่านไฟล์ users.php

สร้างงานบริการ เพื่อใช้ HTTP

เมื่อได้ลำลองข้อมูลสำเร็จแล้ว ต่อไปก็จะเป็นการทำงานของฝั่งไคลเอ็นท์เพื่ออ่านข้อมูลจากเซิร์เวอร์ ก่อนอื่นต้องสร้างคลาส User เพื่อรับข้อมูลในรูปออบเจ็กต์ได้

```
ng generate class user
```

และปรับแต่งสมาชิกในคลาสให้มีข้อมูลเหมือนกับเซิร์บเวอร์

Code 6. src/app/user.ts

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

ในระหว่างการทำงาน ให้บันทึกไฟล์ต่าง ๆ ที่สร้าง และให้สังเกตว่า ที่หน้าต่าง CLI และหน้าเว็บ ให้เปิด โหมด Developer tool (สำหรับ Google Chrome) มีการแจ้งความผิดพลาดอะไรบ้าง ให้ทำการแก้ไข ถ้ามีความผิดพลาด ตามที่ข้อมูลแจ้งความผิดพลาด ก่อนการดำเนินการต่อไป

ต่อไปควรสร้างงานบริการ ให้ชื่อ userService ที่อ่านฐานข้อมูล Users ด้วยโมดูล http

```
ng generate service user
```

ให้เพิ่มการนำเข้า คลาสในโมดูล HttpClient, HttpHeaders ที่จำเป็นต้องใช้ในการสื่อสารกับเซิร์บเวอร์ และมีการอ้างอิงคลาส User ในบริการนี้

Code 7. src/app/user.service.ts

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { User } from './user’;
```

หลังจากนั้นสร้างสมาชิกในคอนสตรักเตอร์ เป็นแบบ private ให้สังเกตว่าสร้างสมาชิกภายในวงเล็บ ไม่ใช่ภายในปีกกา การเขียนแบบนี้จะถือเป็นสร้างสมาชิกให้คลาสอย่างหนึ่ง

Code 8. src/app/user.service.ts

```
constructor(
  private http: HttpClient){ }
```

อ่านข้อมูลด้วย HttpClient

ก่อนอื่นต้องนิยาม usersUrl จากรูปแบบ :base/:collectionName ซึ่งเป็น URL ของข้อมูลบนเซิร์บเวอร์ ในที่นี้มี base เป็นชื่อการร้องขอที่ตั้งชื่อว่า api และ collectionName เป็นชื่อข้อมูล users

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

```
private usersUrl = 'http://localhost/myApi/users.php';  // URL to web api
```

ก่อนหน้านี้เราใช้การอ่านข้อมูลด้วย งานบริการ UserService.getUsers( ) ด้วยการใช้ of( ) ของ RxJs ซึ่งจะคืนค่าคงที่อาร์เรย์ของ USERS และต้องไม่ลืม **นำเข้าคลาส User ด้วย** ดังครั้งที่แล้วเขียนไว้ว่า :

Code 10. src/app/user.service.ts (v1)

```
getUsers(): Observable<User[]> {
  return of(USERS);
}
```

แต่ตอนนี้เราใช้บริการ HTTP แล้ว ซึ่งฟังก์ชัน get<T>(URL) จึงต้องเปลี่ยนโปรแกรมใหม่ดังนี้

Code 11. src/app/user.service.ts (v2)

```
/** อ่านข้อมูลจากเซิร์บเวอร์ Apache */
getUsers (): Observable<User[]> {
  return this.http.get<User[]>(this.usersUrl);
}
```

ตอนนี้ได้อ้างอิง อินเทอร์เฟสต์สำคัญ คือ Observable จึงต้องนำเข้าด้วย

Code 12. src/app/user.service.ts

```
import { Observable, of } from 'rxjs';
```

ฟังก์ชัน http.get( ) ต้องการอ่านเว็บปลายทาง (URL) ที่ส่งข้อมูลกลับมาในรูปอาร์เรย์ หากเราใช้เว็บเซิร์บเวอร์แบบอื่น ๆ การส่งข้อมูลกลับที่ดีควรอยู่ในรูปของอาร์เรย์ และควรเป็น อาร์เรย์แบบ JavaScript หรือไม่ก็เป็นอาร์เรย์ของ JSON

นอกจากใส่ตัวแปรให้กับฟังก์ชัน get( ) ด้วย this.usersUrl แล้ว ยังจะมีทางเลือกอื่น ให้มีได้คือ

```
options: {
        headers?: แทนส่วน HttpHeaders,
        observe?: แทนส่วน จำนวนการตอบสนอง,
        params?: แทนส่วน ตัวแปรที่ส่งไปกับ Http,
        reportProgress?: แทนส่วน Boolean ของรายงานความก้าวหน้า,
        responseType?: แทนส่วน ชนิดการตอบสนอง,
        withCredentials?: แทนส่วน Boolean ของความปลอดภัย,
}
```

ซึ่งในแต่ละทางเลือกที่สำคัญจะได้แสดงให้เห็นต่อไป ในตอนนี้ ในการส่งคำขอ อาจใส่หรือไม่ใส่ส่วนของ header ก็ได้ ในกรณีที่ส่งคำขอที่ผ่านมา ต้องการเพิ่มคำขอ ในลักษณะข้อมูล Json โดยส่งไปกับ header ส่งที่ต้องเพิ่มเติมคือ การนำเข้า HttpHeaders และสร้างตัวแปร httpOptions ต่อไปนี้ โดยวางต่อท้าย คำสั่ง import (ไม่ได้เป็นสมาชิกของคลาส)

Code 13. src/app/user.service.ts

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
const httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};
```

ดังนั้น เมื่อนำไปใช้กับฟังก์ชัน get( ) จะมีการปรับปรุงใหม่คือ

Code 14. src/app/user.service.ts (v3)

```
/** อ่านข้อมูลจากเซิร์บเวอร์ Apache */
getUsers (): Observable<User[]> {
  return this.http.get<User[]>(this.usersUrl, httpOptions);
}
```

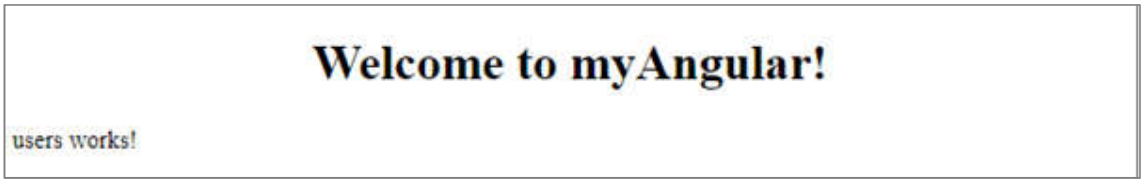
ถึงตอนนี้ใกล้จะทดสอบการแสดงผลแล้ว แต่ยังไม่มีหน้าแสดงผลรายการ Users จึงควรสร้างคอมโพเน้นท์ User ขึ้นมาก่อน

```
ng generate component users
```

เมื่อสร้างคอมโพเน้นท์ users แล้วนำคอมโพเน้นท์นี้ ไปแสดงที่หน้าแรก โดยลบข้อมูลเดิมทั้งหมด แล้วใส่ข้อมูลใหม่ต่อไปนี้

Code 15. src/app/app.component.html

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
</div>
<app-users></app-users>
```



รูป 3 หน้าเว็บขณะเริ่มต้น

ภายในไฟล์ users.component.html ให้ มีเพียง รายการ ผู้ใช้ ซึ่งอ่านข้อมูลในรูปอาร์เรย์ ใช้การวนซ้ำ โดยใช้ *ngFor เพื่อวนซ้ำอ่านค่า users ซึ่งเป็นที่ได้จากสังเกตการณ์ ดังนี้

Code 16. src/app/users/users.component.html

```
<div>
<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>
    </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>
    </tr>
  </tbody>
</table>
</div>
```

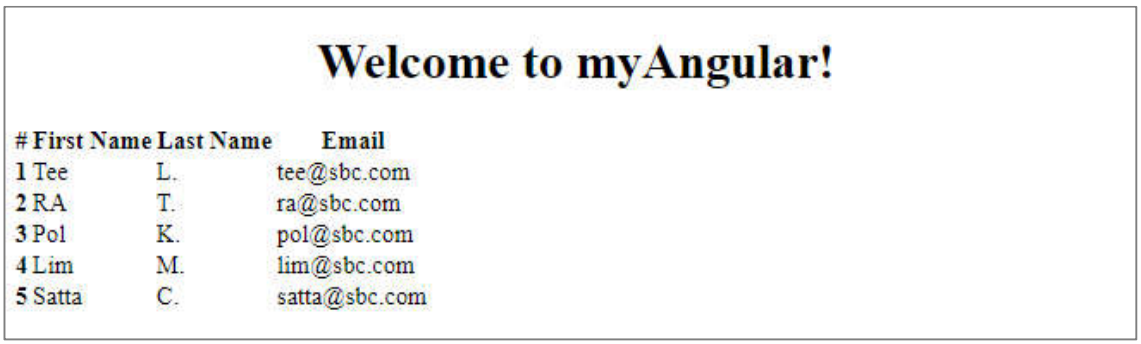
สำหรับ ไฟล์ users.component.ts ให้ มีรายการข้อมูลดังนี้

Code 17. 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 {
  constructor(
  private userService: UserService) { }
  ngOnInit() {
           this.getUsers();
  }
  getUsers():void{
           this.userService
                .getUsers()
                .subscribe(users => this.users = users);
  }
  users: User[];
}
```



รูป 4 ข้อมูลที่อ่านได้จากเซิร์บเวอร์จำลอง

หากไม่สนใจรูปแบบความสวยงาม ก็ใช้แบบนี้ไปก็ได้ หากว่าทนเห็นความไม่มีระเบียบ หรือดไม่ได้ที่จัดรูปแบบ เช่นใส่ CSS ของ Bootstrap[2] ไปหน่อยก็ทำได้ โดยใส่ css ต่อไปนี้ไว้ที่อิลีเม้นท์ <head> ของไฟล์ index.html

Code 18. src/index.html

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
integrity="sha384-aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
crossorigin="anonymous">
```

[2] ดู URL ของ CSS Bootrap ที่ https://getbootstrap.com/docs/4.5/getting-started/introduction/

รูป 5 จัดรูปแบบ CSS ด้วย Bootstrap การข้อมูลที่อ่านได้จากเซิร์บเวอร์จำลอง

อ่านข้อมูลที่ส่งกลับมาค่าเดียว

ทุกฟังก์ชันของ **HttpClient** จะคืนค่า ตัวให้สังเกตการณ์ หรือ RxJS Observable ของข้อมูล ขณะที่ HTTP เป็นโปรโตคอล ร้องขอและตอบรับ ซึ่งการร้องขอหนึ่งครั้งก็จะมีการตอบรับหนึ่งครั้ง แต่การตอบรับหนึ่งครั้งนี้ จะคืนค่าได้มาหลายค่า โดยที่ HttpClient รับค่าหลายค่าอยู่ในรูปอาร์เรย์ ด้วยฟังก์ชัน get( ) สำหรับการจะรับอ่านค่าเดียว ก็คืออ่านอาร์เรย์ค่าแรกนั้นเอง เช่น ต้องการอ่านว่ามีรายชื่อ (users) ซึ่งเป็นอาร์เรย์มีขนาดมากกว่าศูนย์หรือไม่ หากมากกว่าศูนย์ ก็ให้แสดงรายการ แต่ถ้าไม่ใช่ให้แสดงว่า “ไม่มีรายชื่อใด” ด้วยใช้คำสั่ง ngIf ร่วมกับอิลิเม้นท์ ng-template

Code 19. src/app/users/users.component.html

```
<div *ngIf="users?.length>0; else elseBlock">
     <table class="table">
       . . . . .
     </table>
</div>
<ng-template #elseBlock>ไม่มีรายชื่อใด</ng-template>
```

จากตัวอย่างนี้ได้ละส่วนแสดงตารางไว้ ดังที่เคยได้แสดงไว้ก่อนหน้านี้ ซึ่งต้องใส่แบบเต็มแทนลงในเครื่องหมายจุดต่อเนื่อง

ข้อสังเกตการใช้ users?.length เพื่อตรวจสอบขนาดอาร์เรย์ จะใช้ users.length โดยตรงไม่ได้เพราะ users ขณะเริ่มต้นโหลดหน้าเว็บ users ยังไม่มีค่าอะไร จึงต้องถามตรวจสอบก่อน (?) เมื่อโหลดเสร็จจึงมีค่าให้ตรวจสอบขนาดอาร์เรย์ได้

การจัดการความผิดพลาด

เมื่อเกิดความผิดพลาดขึ้น อย่างเช่น การอ่านข้อมูลจากเซิร์บเวอร์ระยะไกล ความผิดพลาดนี้ควรจะดักจับ และจัดการบางอย่างกับความผิดพลาดนี้

ความผิดพลาดมีอยู่ด้วยกันสองแบบที่ คือ ความผิดพลาดฝั่งเซิร์เบอร์ การตอบสนองจาก HTTP เช่น ให้สถานะ (status) ตามที่ระบุ url ไม่เจอ (code 404) กับความผิดพลาดบนฝั่งไคลเอ็นท์เอง เช่น ไม่สามารถต่อเชื่อมเคลื่อข่ายได้ ซึ่งเป็น

ชนิดหนึ่งของ ErrorEvent การตรวจสอบความผิดพลาดทั้งสองลักษณะนี้ใช้ HttpErrorResponse จึงต้องนำเข้า HttpErrorResponse ด้วย

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

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

ตัวอย่างต่อไปนี้ ใช้การฟังก์ชัน handleError<>( ) เพื่อจัดการกับความผิดพลาด ในสองลักษณะดังกล่าวข้างต้น ตัวแปรเข้าตัวแรก (operation) ตั้งใจจะแทนชื่อฟังก์ชันที่เรียกใช้ที่ทำให้เกิดความผิดพลาด หากไม่ระบุก็จะใช้ชื่อว่า 'operation' ซึ่งเป็นค่าปริยาย ตัวแปรที่สอง แทนการดำเนินการ ทั้งที่ได้ผลและไม่ได้ผล (มีความผิดพลาด)

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

```
private handleError<T> (operation = 'operation', result?: T) {
      return (er: HttpErrorResponse): Observable<T> => {
            if(er.error instanceof ErrorEvent){
                  //client-side error
                  console.log('Client:error:', er.error.message);
            }
            else{
                  //server-side error
                  console.log('Server:error:'+operation+':'+er.status
                  + "("+ er.url+ ")" + "\n"+er.message)
            }
            return of(result as T);
      };
}
```

จากตัวอย่างนี้ ด้วยการตรวจชนิดความผิดพลาดว่าเป็นความผิดพลาดที่มาจาก ไคลเอ็นท์ หรือ เซิร์บเวอร์ จาก ErrorEvent เช่น ถ้าร้องขอบบริการไปยัง url ที่ผิด เช่น api/users1 ซึ่งที่อยู่นี้ไม่มีให้บริการ (จัดเป็นความผิดพลาบนเซิร์บเวอร์) ความผิดพลาดที่จับได้จะอยู่ในส่วน else ซึ่งถ้าใส่ตัวแปรแรกของฟังก์ชันนี้ว่า getUsers จะแสดงความผิดพลาดว่า:

```
      Sever:error:getUsers:404(http://localhost/myApp/users1.php/)
      Http failure response for http://localhost/myApp/users1.php/: 0 Unknown
Error
```

การดักจับความผิดพลาด ใช้ฟังก์ชัน pipe( ) เป็นการเชื่อมต่อคำสั่งจากผลการคืนค่าของ Observable จากผลการใช้คำสั่ง http.get( ) การดักจับความผิดพลาด มีตัวดำเนินการอื่นร่วมด้วย ดังจะต้องนำเข้า catchError, tap ดังตัวอย่างต่อไปนี้

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

```
import { catchError, tap, retry } from 'rxjs/operators';
```

เมื่อได้นำเข้าข้อมูลที่ต้องใช้ ดังกล่าวแล้ว จึงสามารถใช้คำสั่ง pip( ) และใช้ตัวดำเนินการซิมโบลภายใน pip( ) ได้

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

```
getUsers(): Observable<User[]> {
  return this.http.get<User[]>(this.usersUrl)
    .pipe(
     catchError(this.handleError('getUsers', []))
```

```
    );
}
```

สำหรับตัวแปรเข้าแรก (operation) มีค่าปริยายเป็น คำว่า 'operation' แต่ในฟังก์ชัน ที่เรียกใช้จริงได้ใส่คำว่า 'getUsers' แทนชื่อฟังก์ชันที่เรียกใช้

สำหรับตัวแปรเข้าที่สอง เป็นทางเลือกมีก็ได้ ไม่มีก็ได้ กรณี ทำให้ result เป็นจริง จะคืนค่า T (generic data type) ซึ่งคือ Observable ในรูปอาร์เรย์ ในตัวอย่างการเรียกใช้ใส่เป็นตัวแปร [ ] หรืออาร์เรย์ว่าง

ถึงตอนนี้อาจทดสอบความผิดพลาดบนเซิร์บเวอร์ โดยการใส่ URL ที่ผิดแล้วดูผล (อาจไม่ขึ้นผลที่ console.log( ) ทันที ต้องใช้เวลารอเล็กน้อย) หรือทดสอบความผิดพลาดบนไคลเอ็นท์

ฟังก์ชัน tap, retry

จากที่ได้นำเข้า tap ซึ่งทำหน้าที่อ่านข้อมูลที่มาจาก การดำเนินการ RxJS ในที่นี้ฟังก์ชัน tap ไม่ได้ทำอะไรนอกเสียจากว่า จะส่งค่าไปยัง log( ) ด้วยการใส่ตัวแปรเข้าเป็นแลมป์ดา มีตัวแปรของแลมป์ดาเป็น users ซึ่งก็ไม่ได้ทำอะไรกับค่าตัวแปรนี้

นอกจากนี้ในบางครั้งการทำงานของ http อาจยังไม่ได้ผลทันที โดยเฉพาะการร้องขอบริการขอการใช้งานบนระบบโทรศัพท์เคลื่อนที่ การร้องขอครั้งแรกอาจไม่สำเร็จ จึงต้องร้องขอซ้ำอีกครับ ด้วยฟังก์ชัน retry( ) ตัวแปรเป็นตัวเลขแทนจำนวนครั้งที่ร้องขอซ้ำ ดังตัวอย่างต่อไปนี้

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

```
getUsers(): Observable<User[]> {
     return this.http.get<User[]>(this.usersUrl)
          .pipe(
             retry(3),
             tap( _ => console.log("fetched users")),
             catchError(this.handleError('getUsers', []))
          )
}
```

ตรวจสอบการตอบสนอง HTTP (HttpInterceptor)

ในการร้องขอบริการของ HTTP จากไคลเอ็นท์ไปสู่เซิร์บเวอร์ หรือรับตอบสนองจาก เซิร์บเวอร์มาสู่ไคลเอ็นท์ อาจต้องการตรวจสอบผลที่กำลังส่งไป หรือผลที่กำลังรับมา เช่น การส่งรูปแบบข้อมูลไปใหม่ระหว่างส่ง หรือตรวจดูความก้าวหน้าในการรับข้อมูล รวมทั้งตรวจสอบความผิดพลาดระหว่างส่งข้อมูล และรับข้อมูล

การตรวจสอบความผิดพลาดที่ผ่านมา เป็นการจับความผิดพลาดที่ละจุดหรือแต่ละงานบริการ เช่น UserService การแก้ไขความผิดพลาดในการตรวจสอบใหม่ จะต้องตามไปแก้ไขทั้งหมดที่มี ซึ่งมีโอกาสที่จะพลาดได้ง่าย วิธีการที่ดีกว่าคือการใช้ HttpInterceptor ลงทะเบียนงานบริการความผิดพลาดในระดับมูโดล ซึ่งจะสามารถจัดการกับความผิดพลาดได้ทั้งแอปพลิเคชัน เพื่อการนี้ให้สร้างคลาส httpErrorInterceptor

```
ng generate class httpErrorInterceptor
```

คลาสนี้ต้องการทำหน้าที่จับความผิดพลาด

Code 25. src/app/http-error-interceptor.ts

```
import {
 HttpEvent, HttpInterceptor, HttpHandler,
 HttpRequest, HttpResponse, HttpErrorResponse
} from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { retry, catchError } from 'rxjs/operators';

export class HttpErrorInterceptor implements HttpInterceptor {
 intercept(request: HttpRequest<any>,
           next: HttpHandler): Observable<HttpEvent<any>> {
   return next.handle(request)
     .pipe(
       retry(1),
       catchError((error: HttpErrorResponse) => {
         let errorMessage = '';
         if (error.error instanceof ErrorEvent) {
           // client-side error
           errorMessage = `Error: ${error.error.message}`;
         } else {
           // server-side error
           errorMessage = `Error Code: ${error.status}\nMessage: ${error.url}`;
         }
         alert(errorMessage);
         return throwError(errorMessage);
       })
     )
 }
}
```

ต่อมาลงทะเบียนใช้ในโมดูล ซึ่งเพิ่มในส่วน import และในส่วน providers

Code 26. src/app/app.module.ts

```
import { HttpErrorInterceptor } from './http-error-interceptor';
import { HTTP_INTERCEPTORS } from '@angular/common/http';
```

Code 27. src/app/app.module.ts

```
providers: [
   {
     provide: HTTP_INTERCEPTORS,
     useClass: HttpErrorInterceptor,
     multi: true
   }
],
```

เมื่อมีคลาส httpErrorInterceptor รองรับความผิดพลาดระดับโมดูลแล้ว การตรวจจับความผิดที่เคยเขียนไว้ใน UserService ก็ไม่จำเป็นต้องมีแล้ว ดังแก้ไขโดยการเอาส่วนจัดการความผิดพลาดออกไปได้

Code 28. src/app/user.service.ts

```
getUsers(): Observable<User[]> {
       return this.http.get<User[]>(this.usersUrl, httpOptions);
```

```
        /* หลังจากใช้ interceptor ส่วนต่อไปนี้ก็ไม่จำเป็นต้องมี
        return this.http.get<User[]>(this.usersUrl, httpOptions)
                    .pipe(
                            tap( u => console.log("fetched users")),
                            catchError(this.handleError('getUsers', []))
                     );
      */
}
```

ส่งตัวแปรผ่าน Headers/params

การส่งตัวแปรผ่าน URL จะส่งผ่านโดยตรงกับ URL หรือส่งผ่าน options ของฟังก์ชัน http.get( ) ที่เคยสร้างก่อนหน้านี้ก็ได้ สมมุติว่า เราต้องการให้คนหาชื่อ (fname) ซึ่งต้องส่งเป็นตัวแปรใส่เป็นทางเลือกหนึ่ง ของฟังก์ชัน http.get( ) การสร้างตัวแปรนี้ จะต้องนำเข้า HttpParams ก่อนจะเขียนเป็นฟังก์ชัน searchUsers( ) ดังนี้

Code 29. src/app/user.service.ts

```
import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
```

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

```
searchUsers(term:string):Observable<User[]>{
   term = term.trim();
   const options = term ?
             { headers: new HttpHeaders({ 'Content-Type': 'application/json' }),
              params:new HttpParams().set('qurey',term)}:{};
   return this.http.get<User[]>(this.usersUrl, options);
}
```

ฟังก์ชันนี้ รับตัวแปรเข้าเป็นคำค้น (term) ที่ตัดช่องว่างหน้า-หลังแล้ว (trim( )) มีค่าคืนเป็น Observable<> ภายใต้ต่อด้วยการสร้างตัวแปร options ซึ่งจะได้ค่าตามเงื่อนไข ถ้าอย่างสั้น (short if) คือ ถ้ามีคำค้นจริง จะกำหนดตัวแปรตามชื่อคำค้น (query) แต่ถ้าไม่มีคำค้น ก็จะคืนค่าออบเจ็กต์เปล่า ({ }) ซึ่งหมายถึงเลือกทั้งหมดนั้นเอง

ถ้ามีคำที่ต้องการค้นหา k จะสามารถเขียนในรูป URL แบบ GET อยู่ในรูป

```
http://localhost/myApp/users.php?query=k
```

รูปแบบการตั้งค่า options ทำได้หลายแบบ หากต้องการใช้เป็น รูปแแบบ JSON ก็ทำได้ ดังเช่น

```
{
       headers: { 'Content-Type': 'application/json' },
       params: {query:term}
}:{};
```

สำหรับการเพิ่มค่าทางเลือกหากต้องการก็เพิ่มได้ เช่น ต้องการส่ง ตัวแปร params เพิ่มได้อีกเช่น ต้องการส่งตัวแปรสองตัว โดยเพิ่มตัวแปรอีกตัว ในชื่อ id จะเขียนในส่ง params คือ

```
params: { query:term, id: 1 }
```

ดังนั้น ข้อมูลที่ส่งผ่านแบบ GET จะอยู่ในรูปแบบดังนี้

```
http://localhost/myApp/users.php?query=k&id=1
```

เมื่อกำหนดงานบริการไว้แล้ว ในคอมโพเน้นท์ Users ก็จะสร้างฟังก์ชันอ่านงานบริการนี้ โดยกำหนดเป็นฟังก์ชันชื่อ searchUsers( ) โดยมีตัวแปรเข้าเป็นคำค้น (term) ภายในฟังก์ชันนี้ เรียกใช้บริการผ่านตัวแปร this.userService และเรียกฟังก์ชันในงานบริการ (searchUser( )) มีตัวแปรเข้าเป็นคำค้นเช่นเดียวที่นิยามในงานบริการ ต่อด้วยสมัครรับบริการนี้เป็นการรับค่าให้สังเกตการณ์ (Observable)

Code 31. src/app/users/users.component.ts

```
searchUsers(term:string):void{
        this.userService
        .searchUsers(term)
        .subscribe(users => this.users = users);
}
```

ยังต้องแก้ไขในส่วนไฟล์ html ของคอมโพเน้นท์นี้ ในการนี้ต้องการสร้างกล่องคำค้น และปุ่มคำสั่งค้น โดยผูกไว้กับเหตุการณ์ click เพื่อเรียกฟังก์ชัน searchUsers ที่ได้สร้างในคอนโพเน้นท์นี้ ดังมีตัวอย่างเขียนได้คือ (วางบนสุดของไฟล์)

Code 32. src/app/users/users.component.ts

```
<nav class="navbar navbar-light bg-light" style="float:right;margin-right:10px">
  <form class="form-inline">
    <input #serarchTerm class="form-control"
             placeholder="Search Name" aria-label="Search">
    <button class="btn btn-outline-success"
             (click)="searchUsers(serarchTerm.value)">Search</button>
  </form>
</nav>
```

ตัวอย่างกล่องคำค้น เป็นฟอร์มที่จำเป็นต้องนำเข้า FormModule ในโมดูล ด้วยหากลืมใส่อาจเกิดเหตุการณ์ไม่คาดเดาได้ เช่น มีการสร้างคอมโพเน้นท์ใหม่ หลังจากส่งคำค้นไปเซิร์บเวอร์ ทำให้ onInit( ) ทำงานใหม่อีกครั้ง

การนำเข้า FormModule ต้องเพิ่มสองส่วนคือ ส่วน import และส่วน imports อาร์เรย์

Code 33. src/app/app.module.ts

```
import { FormsModule }   from '@angular/forms';
```

Code 34. src/app/app.module.ts

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

ส่วนสำคัญอีกตัวหนึ่ง คือไฟล์ users.php ที่ต้องทำให้รองรับ การสืนค้นได้ด้วย จากคำสำคัญ(key) ที่ส่งมาชื่อ query ดังนั้นเซิร์บเวอร์ ต้องรับคำสำคัญนี้ก่อนผ่าน $_GET[‘query’] และค่อยไปดำเนินการอ่านค่าของที่ต้องการค้น

Code 35. xampp/httpdocs/myApp/db.php

```
<?php
```

```
include 'db.php';

$query =(isset($_GET['query']))?$_GET['query']:null;

if($query!=null){
	$result = array();
	foreach($users as $u){
	$strlower = strtolower($query);
	if(preg_match("/($strlower)/", strtolower($u->fname))
	   ||
	   preg_match("/($strlower)/", strtolower($u->lname))
	   ||
	   preg_match("/($strlower)/", strtolower($u->email))
	)
		array_push($result, $u);
	}
	echo json_encode($result);
}
else {
	echo json_encode($users);
}
```

จากตัวอย่างในนี้ ใช้การค้นทุกค่าใน รายการผู้ใช้ (users) ทั้ง fname, lname, email ถ้ามีส่วนตรงใดตรงก็ให้เก็บเข้าอาร์เรย์ (array_pusht( ) ไว้ในผลลัพธ์ ($result) แต่คำค้นต้องทำให้เป็นอักษรตัวเล็กให้หมดก่อน (strtolower( )) สุดท้ายก็ส่งค่าในรูปแบบ JSON

ส่วนแสดงการค้นตามชื่อนี้ ใช้ร่วมกับ Bootstrap เพียงเพื่อความเป็นระเบียบเท่านั้น สิ่งที่น่าสนใจคือ <input> ไม่ได้ใช้ชื่อข้อมูลเข้า ตามแบบ html ทั่วไป แต่ใช้ # นำหน้าชื่อข้อมูลของ <input> นี้แทน เป็นการอ้างอิงของ Angular เอง ซึ่งจะเห็นว่านำไปเป็นตัวแปรเข้าของฟังก์ชัน searchUsers( )

เมื่อเขียนโปรแกรมของส่วนตาม ครบแล้วก็ทดสอบได้ จะได้ผลตามรูปต่อไปนี้ ซึ่งถ้า ค้นคำที่มี หรือมีบางส่วนก็จะแสดงได้หลายรายได้ หรือถ้าไม่ใส่อะไรเลย ก็จะแสดงรายชื่อทั้งหมดที่มีในฐานข้อมูลจำลอง

รูป 6 การแสดงผลส่วนการค้นหา ใช้ CSS ของ Bootstrap

ตัวให้สังเกตการณ์ (Observable)

การค้นคำที่ผ่านมาทำงานโดยตรงกับเซิร์บเวอร์ จากการเขียนเงื่อนไขการกรองบนเซิร์บเวอร์ แต่ยังมีอีกวิธีที่กรองข้อมูลบนไคลเอ็นท์ จากเลือกทั้งหมดที่เซิร์บเวอร์ส่งมาให้ เราทำได้กับข้อมูลที่ได้มานี้ โดยดำเนินการกับตัวให้สังเกตการณ์นี้

วิธีการนี้ทำให้ที่ฝั่งเซิร์บเวอร์ไม่จำเป็นเขียนโปรแกรมเพิ่มในส่วนการกรองข้อมูล แต่ก็มีข้อเสียอย่างหนึ่งคือถ้ามีข้อมูลจำนวนมาก การทำงานฝั่งไคลเอ็นท์จะทำงานได้ช้าลง

ยกตัวอย่างเช่น ใช้ฟังก์ชัน getUsers() ตัวเดิมที่เคยเขียนไว้ก่อนหน้านี้ อ่านค่า ได้ User[ ] แล้วทำการเชื่อมคำสั่งต่อไป (pipe) ให้กรองค่า

Code 36. src/app/users/users.component.ts

```
searchUsers2(term:string):void{
        const chars = term.trim().toString().toLowerCase();
        console.log(chars);
        this.userService
            .getUsers()
              .pipe(
                   map((users:User[])=>users.filter((user:User)=>
                    user.id.toString().search(chars)!=-1 ||
                    user.fname.toString().toLowerCase().search(chars)!=-1 ||
                    user.lname.toString().toLowerCase().search(chars)!=-1 ||
                    user.email.toString().toLowerCase().search(chars)!=-1))
              )
            .subscribe(users => this.users = users);
  }
```

ฟังก์ชัน searchUser2( ) ตัวใหม่นี้ทำงานโดยการกรองข้อมูล ทั้ง id, fname, lname, email โดยการเชื่อมด้วยหรือ (||) ในการกรองเลือกที่สร้างเป็นอักษรพิมพ์เล็กทั้งหมดและต่อด้วยการใช้คำสั่ง search( ) โดยการผ่านตัวแปรคำที่ต้องการค้น คำสั่งนี้ถือเป็นการใช้คำสั่งรีกูล่าร์แบบหนึ่งซึ่งสามารถใส่ในรูปแบบรีกูล่าร์ก็ได้

อ่านข้อมูลเป็นไฟล์ข้อความ

ไม่ใช่ทุกครั้งที่เซิร์บเวอร์รองรับข้อมูลในรูปแบบ JSON เซิร์บแจ้งสถานะในรูปแบบอักษรตัวเลขเพื่อแจ้งสถานะของเซสชัน (session) หรือใช้การส่งค่าตัวเลขเพื่อแจ้งผลดำเนินการกับฐานข้อมูล หรือข้อมูลที่ได้มาอาจอยู่ในรูปไฟล์ข้อความทั่วไป เช่น users.txt หรือ users.csv ซึ่งเป็นไฟล์ข้อมูลที่อาจนำไปมวลผลต่อไปได้ ผลตอบกลับจะอยู่ในรูป Observable<string> การตอบสนนองในรูป string จึงไม่จำเป็นใส่ชนิดของ Observable ก็ได้

การตอบสนองในรูปข้อความ มีตัวแปรที่ใส่เพิ่มให้กับ get( ) คือ {responseType:'text'} โดยมี URL เป็นเส้นทางของไฟล์ที่เก็บไว้ ต่อไปจะสมมุติให้อ่านไฟล์ users.csv ซึ่งจะต้องมีไฟล์นี้อยู่จริงบนเซิร์บเวอร์

นอกจากนี้ยังใช้ pipe เพื่อตรวจสอบความผิดพลาดผ่านตัวแปร data และ error ดังตัวอย่างฟังก์ชัน getTextFile( ) ที่ใส่เพิ่มในคลาส UserService

Code 37. src/app/user.service.ts

```
getTextFile(filename:string) {
        const url = 'http://localhost/myApp/';
        return this.http.get(url+filename, {responseType: 'text'})
            .pipe(
              tap( // Log the result or error
                   data => console.log(filename, data),
                   error => console.log(filename, error)
            ));
```

```
 }
```

สำหรับคลาส UserComponent จะเรียนกใช้งานงานบริการนี้ ด้วยการสมัคร (subscript( )) ซึ่งจะมีผลตอบกับมาในรูปตัวแปร text โดยชื่อตัวแปรนี้ใช้เป็นชื่ออะไรก็ได้ ผลการอ่านข้อความจากไฟล์ได้ ทดสอบด้วย console.log( )

Code 38. src/app/users/users.component.ts

```
getTextFile(filename:string):void{
       this.userService.getTextFile(filename)
           .subscribe((text) => {
              console.log('"Text" RESPONSE :', text);
           });
}
```

แต่ก่อนจะทดสอบให้สร้าง <button> ไว้ในหน้าเว็บของ UserComponent เลือกใช้ <button> รับเหตุการณ์ click และชื่อชื่อฟังก์ชันพร้อมกับชื่อไฟล์ users.csv

Code 39. src/app/users/users.component.html

```
<form class="form-inline">
<button class="btn btn-outline-success col-sm-2"
       (click)="getTextFile('users.csv')">Download</button>
</form>
```

เมื่อทดลองใช้งานจะเห็นข้อความปรากฏที่ console.log( ) ไฟล์ CSV เป็นไฟล์ที่ใช้แทนข้อมูลอย่าง Microsoft Excel ได้ แต่ใช้งานยากสำหรับ JavaScript ทางที่ดีควรจะจัดการให้เป็น JSON เพื่อนำไปประมวลได้ง่ายกว่า

ResponseType ยังมีแบบอื่น เช่น blob มีค่าที่รับได้เป็น Observale<blobe> และ arraybuffer มีค่าที่รับได้เป็น Observable<arraybuffer> ซึ่งเป็นรูปแบไฟล์ไบนารี และไบนารีโดยตรง การใช้งานก็ใช้งานในลักษณะเดียวกับตัวอย่างที่ยกมา

แปลงไฟล์ CSV เป็น JSON

ตัวอย่างต่อไปนี้แนะนำการแปลง ไฟล์ประเภท CSV ไปเป็นออบเจ็กต์ JSON ซึ่งนำไปแสดงผลแทนอาร์เรย์ User[ ] ได้ โดยสมมุติให้ไฟล์ CSV มีข้อมูลเหมือนกับข้อมูลที่จำลอง ดังมีข้อมูลไฟล์ users.cvs แต่มีเพียง 4 แถว เพื่อให้แตกต่างจากข้อมูลเดิม

Code 40. xampp/httdocs/myApp/users.csv

```
id, fname, lname, email
1, Tee, L.,  tee@sbc.com
2, RA, T., ra@sbc.com
3, Pol, K., pol@sbc.com
4, Lim, M., lim@sbc.com
```

ให้สังเกตว่า แถวที่สอง ตรงอีเมล์ tee@sbc.com มีช่องว่างมากว่าหนึ่งที่ และแถวสุดท้าย ที่ขึ้นบรรทัดใหม่ แต่มีมีข้อความ ซึ่งความจริงมีบรรทัดทั้งหมด 5 แถว ซึ่งเป็นไปได้ว่าข้อมูลที่ได้มาไม่ได้เป็นระเบียน

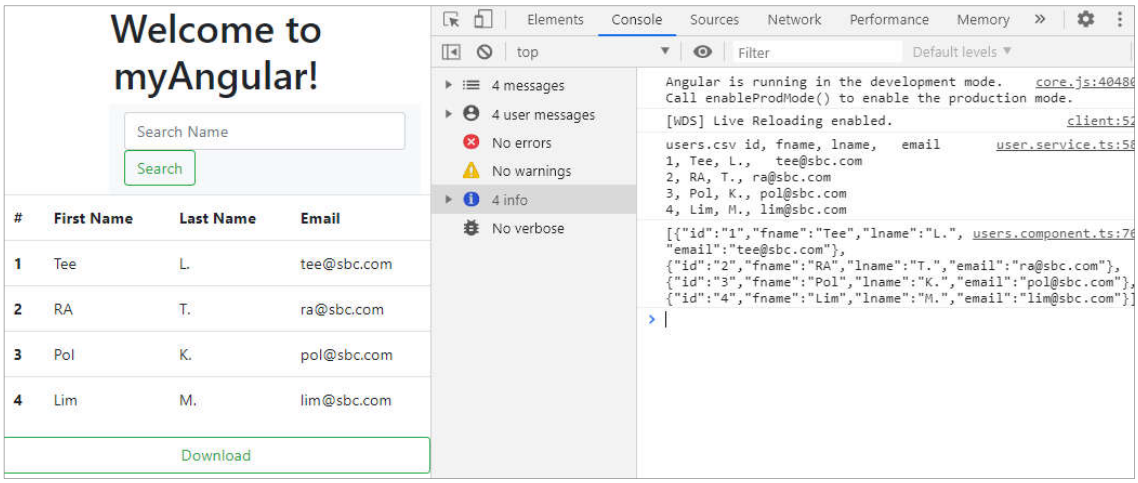
Code 41. src/app/users/users.component.ts

```
csvJSON(csv):User[]{
   let lines:string[] = csv.trim().split("\r\n");
   let users=[];
   let heads:string[] = lines[0].split(",");
   for (let i = 1; i < lines.length; i++) {
       let user = {};
       let rows:string[] = lines[i].split(",");
            rows.forEach(u=>u.trim());
       for (let j = 0; j < heads.length; j++) {
                  if(typeof rows[j]==='string')
                        rows[j] = rows[j].trim();

                  if(typeof heads[j]==='string')
                        user[heads[j].trim()] = rows[j];
       }
       users.push(user);
   }
   console.log(JSON.stringify(users)); //JSON
   return users;
 }
```

ฟังก์ชัน csvJSON นี้นอกจากจะแปลงให้เป็นอาร์เรย์ของ JSON ได้แล้วยังสร้างเป็นอาร์เรย์ของ User เพื่อนำไปใช้แทนข้อมูลที่แสดงผลได้ที่หน้าเว็บ

ฟังก์ชันนี้ เริ่มจากตัด ช่องว่างหน้าหลังด้วย trim( ) ทำให้บรรทัดสุดท้ายที่ว่างถูกตัดทิ้งไป ต่อด้วยตัด การขึ้นบรรทัดใหม่ (บางระบบใช้เบียง \r หรือ \n) ต่อมาใช้การวนซ้ำเพื่อตัดคำ ให้สังเกตว่ามีการตรวจสอบการเป็นชนิดอักษร เพื่อให้แน่ใจว่าใช้การตัดช่องว่างได้



รูป 7 การแสดงผลส่วนการอ่านไฟล์ users.csv

ฟังก์ชันนี้สามารถทำไปแทน console.log( ) เดิมที่อยู่ฟังก์ชัน getTextFile( ) ได้ การแสดงผลจะได้ผลเป็นรายการผู้ใช้เพียง 4 รายตามที่มีไฟล์ users.csv

Code 42. src/app/users/users.component.ts

```
getTextFile(filename:string):void{
```

```
        this.userService.getTextFile(filename)
            .subscribe((text) => {
                    //console.log(this.csvJSON(text));
                    this.users = this.csvJSON(text);

             });
}
```

การแปลงข้อมูลให้อยู่ในรูปแบบ JSON ต่อไปจะมีส่วนสำคัญที่จะนำไปแสดงผลแผนภาพกราฟชนิดต่างๆ ในบทที่ว่าด้วยเรื่อง PrimeNg ต่อไป

สรุป

บทนี้ให้แนวคิดการใช้ httpClient ขั้นพื้นฐานในการใช้ ฟังก์ชัน get( ) กับ เว็บเซิร์บเวอร์ที่ใช้ ภาษา PHP ซึ่งเขียนให้รองรับคำสั่ง get( ) อย่างง่าย เพียงเพื่อการสาธิตการใช้งาน ซึ่งจะสังเกตว่ายังคงแยกเป็นงานบริการออกจากส่วนคอมโพเน้นท์ที่ใช้สำหรับแสดงผลอย่างเดียว ในงานบริการใช้การอ่านข้อมูลผ่าน http.get( ) ที่ได้ต้องใส่ตัวแปรที่สำคัญคือ URL ที่ต้องการไปยังเซิร์บเวอร์ นอกจากนี้ยังตัวแปรที่สอง คือ options ในรูปแบบต่างๆ เช่น การ headers, params, responsType เพื่อเลือกข้อมูลที่ไม่ใช้ข้อความ JSON คือใส่เป็น text ข้อมูลที่อ่านได้มาจัดเป็น ค่าให้สังเกตการณ์ที่นำมาจัดการได้ดังได้ยกตัวอย่างการกรองข้อมูลของ User[ ] นอกจากนี้ยังแนะนำการจัดการความผิดพลาดซึ่งทำได้ในระดับฟังก์ชัน และระดับโมดูล (HttpInterceptor)

ยังหลายเรื่องข้อ HttpClient ยังไม่ได้กล่าวถึง เช่น การปรับปรุงข้อมูล การแทรกข้อมูล การลบข้อมูล โดยการใช้คำสั่งอื่น put( ), post( ), delete( ) เพื่อรองรับการดำเนินการกับข้อมูลในลักษณะต่าง ๆ กัน ซึ่งจะศึกษากันต่อในบทต่อไปกับการทำงานกับฐานข้อมูล MySql