Introduction to Angular

Web Beginner Course

Angular คืออะไร?

Angular คือ Framework ที่พัฒนาจาก TypeScript ใช้สำหรับพัฒนา Web Application ต่างๆ แบบ component-based ประกอบด้วย features ต่างๆ ให้ใช้งานมากมาย เช่น Routing, Forms, Client/Server communication และ library อื่นๆ อีกมากมาย

การติดตั้ง Angular

ติดตั้ง Angular ผ่าน npm ด้วยคำสั่งต่อไปนี้

```
npm install -g @angular/cli
```

เมื่อติดตั้งแล้ว สามารถตรวจสอบเวอร์ชั่นได้ด้วยคำสั่งต่อไปนี้

```
ng version
```

การสร้าง Angular Project

ใช้คำสั่งต่อไปนี้ที่ Terminal เพื่อสร้าง Angular project ใหม่

```
ng new my-app
```

หลังจากที่ project สร้างเสร็จแล้ว ลองรัน project ที่สร้างขึ้นมาด้วยคำสั่งต่อไปนี้

```
cd my-app
ng serve --open
```

จากนั้นโปรแกรมจะทำการรัน และเปิด app ขึ้นมาที่ URL http://localhost:4200/

Components

Component คือชิ้นส่วนต่างๆ ของ Web Application ที่สร้างด้วย Angular โดยในแต่ละ component จะประกอบด้วย HTML template (รวมไปถึง CSS) ที่ทำหน้าที่เป็น UI ของ component นั้นๆ และอาจประกอบด้วย attributes และ methods อื่นๆ ที่จำเป็นต่อการใช้งานภายในนั้นด้วย

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
  `
})
export class HelloWorldComponent {
  // The code in this class drives the component's behavior.
}
```

Components (ต่อ)

ในการนำ component ต่างๆ ไปแสดงผลใน template สามารถทำได้โดยการเขียนในรูปแบบของ tag <selector-name></selector-name>

```
<div class="container">
    <hello-world></hello-world>
    <p>The component HelloWorldComponent will appear above.</p>
</div>
```

เมื่อแสดงผล Angular จะ render ออกมาเป็น

```
<div class="container">
    <hello-world>
        <h2>Hello World</h2>
        <p>This is my first component!</p>
    </hello-world>
    <p>The component HelloWorldComponent will appear above.</p>
</div>
```

Templates

นอกจากการประกาศ Template เป็น static HTML ใน property template แล้ว ยังสามารถสร้าง HTML template แยกไว้ในไฟล์ .html อีกไฟล์หนึ่งก็ได้ โดยภายใน template เราสามารถอ้างถึง attribute ต่างๆ ภายใน component ได้โดยการใส่วงเล็บปีกกา 2 ชั้นครอบชื่อ attribute ไว้

```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world-interpolation',
  templateUrl: './hello-world-interpolation.component.html'
})
export class HelloWorldInterpolationComponent {
  message = 'Hello, World!';
}
```

hello-world-interpolation.component.ts

```html
<p>{{ message }}</p>
```

hello-world-interpolation.component.html

Templates (ต่อ)

นอกจากนี้ยังสามารถ bind property ต่างๆ กับ HTML attribute ได้ด้วย โดยการใส่วงเล็บสี่เหลี่ยมครอบชื่อ attribute ไว้

```ts
export class HelloWorldInterpolationComponent {
  sayHelloId = 1;
  fontColor = 'green';
}
```

hello-world-interpolation.component.ts

```html
<p
  [id]="sayHelloId"
  [style.color]="fontColor">
  You can set my color in the component!
</p>
```

hello-world-interpolation.component.html

Templates (ต่อ)

การ bind method กับ action ต่างๆ (เช่น click) สามารถทำได้โดยใส่เครื่องหมายวงเล็บครอบชื่อ HTML action เอาไว้ แล้วเซตค่าเป็น method ที่อยู่ใน component (ถ้ามี parameter ที่ต้องส่ง ให้ส่งเข้าไปด้วย)

```
export class HelloWorldInterpolationComponent {
  message = 'Hello, World!';
  sayMessage() {
    alert(this.message);
  }
}
```

hello-world-interpolation.component.ts

```
<button
  type="button"
  [disabled]="canClick"
  (click)="sayMessage()">
  Trigger alert message
</button>
```

hello-world-interpolation.component.html

Templates (ต่อ)

ในกรณีที่ต้องการแสดงบางชิ้นส่วนแบบมีเงื่อนไข สามารถใช้ directive *ngIf เพื่อควบคุมการแสดงผลได้

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world-ngif',
  templateUrl: './hello-world-ngif.component.html'
})
export class HelloWorldNgIfComponent {
  message = "I'm read only!";
  canEdit = false;

  onEditClick() {
    this.canEdit = !this.canEdit;
    if (this.canEdit) {
      this.message = 'You can edit me!';
    } else {
      this.message = "I'm read only!";
    }
  }
}
```

hello-world-ngif.component.ts

```
<h2>Hello World: ngIf!</h2>

<button type="button" (click)="onEditClick()">Make
text editable!</button>

<div *ngIf="canEdit; else noEdit">
    <p>You can edit the following paragraph.</p>
</div>

<ng-template #noEdit>
    <p>The following paragraph is read only. Try
clicking the button!</p>
</ng-template>

<p [contentEditable]="canEdit">{{ message }}</p>
```

hello-world-ngif.component.html

Templates (ต่อ)

ในกรณีที่ต้องการวนลูปเพื่อสร้าง element ที่ฝั่ง template สามารถใช้ directive *ngFor เพื่อนำ Array ไปวนลูปแล้วสร้าง element ออกมาได้

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world-ngfor',
  templateUrl: './hello-world-ngfor.component.html'
})
export class HelloWorldNgForComponent {
  persons: any[] = [{
    id: 1,
    name: 'John Doe'
  },{
    id: 2,
    name: 'Jane Doe‘
  },{
    id: 3,
    name: 'James Doe'
  }]
}
```

hello-world-ngfor.component.ts

```
<h1>Attending Person List</h1>

<ol>
    <li *ngFor="let person of persons"
[id]="person.id">{{ person.name }}</li>
</ol>
```

hello-world-ngfor.component.html

Styling

นอกจากการใส่ style แบบ inline แล้ว เราสามารถสร้าง stylesheet แยกออกมาอีกไฟล์ แล้วนำมา import ที่ property `styleUrls` ของ `@Component` ก็ได้ (สามารถใส่ stylesheet ได้มากกว่า 1 ไฟล์)

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.scss']
})
export class HelloWorldComponent {
  ...
}
```

hello-world.component.ts

Modules

โดยทั่วไปแล้ว ในแอพของเรามักจะประกอบไปด้วย component มากกว่า 1 อัน และแต่ละอันอาจมีการรับ-ส่งข้อมูลกัน ในการที่จะทำให้แต่ละ component เชื่อมต่อหากันได้ เราจำเป็นต้องเพิ่ม component ต่างๆ ให้อยู่ใน module เดียวกันก่อน รวมทั้งสร้าง routing module เพื่อระบุ default component ที่จะนำมาแสดงผล

- สร้าง xxx-routing.module.ts และระบุ default component ที่จะแสดงใน path: ''
- import routing module เข้ามาที่ฟิลด์ imports ใน xxx.module.ts
- นำ component ทุกอันที่ต้องการให้เชื่อมต่อหากันได้ ใส่ไว้ในฟิลด์ `declarations` ใน xxx.module.ts

Modules (ต่อ)

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

import { MainRoutingModule } from './main-
routing.module';
import { MainComponent } from
'./main.component';
import { NameChangerComponent } from './name-
changer/name-changer.component';

@NgModule({
  declarations: [MainComponent,
NameChangerComponent],
  imports: [FormsModule, MainRoutingModule],
  providers: [],
  bootstrap: [MainComponent],
})
export class MainModule {}
```

นำ component ที่ต้องการให้เชื่อมต่อหากันได้มาประกาศในนี้ให้หมด

main.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from
'@angular/router';

import { MainComponent } from './main.component';

const routes: Routes = [
  {
    path: '',
    component: MainComponent,
  },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
})
export class MainRoutingModule {}
```

ระบุ default component เป็น MainComponent

main-routing.module.html

Routing

หากต้องการแสดงหน้าเว็บใหม่ตาม path ที่กรอกลงไป ให้เซตค่าไว้ที่ routing module โดย component/module ที่เซตไว้จะ render ออกมาที่แท็ก `<router-outlet></router-outlet>` ใน template

```
...
const routes: Routes = [
  {
    path: '',
    component: MainComponent,
  },
  {
    path: 'sub',
    component: SubComponent,
  }
];
...
```

Routing ด้วย component

```
...
const routes: Routes = [
  {
    path: '',
    loadChildren: () =>
        import('./components/main/main.module').then((m) =>
            m.MainModule),
  },
  {
    path: 'sub',
    loadChildren: () =>
        import('./components/sub/sub.module').then((m) =>
            m.SubModule),
  }
];
...
```

Routing ด้วย module

Routing (ต่อ)

ใช้ class `Router` ของ Angular ในการ redirect page ไปยัง path ที่ตั้งไว้ โดยก่อนอื่นต้อง inject Router เข้ามาที่ constructor ของ component class ก่อน จากนั้นเรียกใช้งาน `this.router.navigate()` เมื่อต้องการ redirect

```
...
import { Router } from '@angular/router';

@Component({
  selector: 'navbar',
  templateUrl: './navbar.component.html',
})
export class NavbarComponent {
  ...
  constructor(private router: Router){}
  ...
  toSub(){
    this.router.navigate(['sub']);
  }
}
```

navbar.component.ts

Routing (ต่อ)

นอกจากนี้ ยังสามารถ redirect จาก template ได้ด้วยแท็ก <a> ที่ระบุ attribute routerLink เอาไว้

```
<ul>
    <li><a routerLink="/">Main</a></li>
    <li><a routerLink="sub">Sub</a></li>
</ul>
```

navbar.component.html

Two-way Binding

ในกรณีของ form element ต่างๆ เช่น <input> ที่จำเป็นต้องแสดงค่าและเก็บค่าที่กรอกในคราวเดียวกัน ควรใช้ directive [(ngModel)] เพื่อ bind ค่าระหว่างตัวแปร กับ <input> element ให้เปลี่ยนค่าไปพร้อมกัน

* ต้อง import module FormsModule ไปที่ module ที่ต้องการจะใช้งาน NgModel ก่อน

```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world-ngfor',
  templateUrl: './hello-world-ngfor.component.html'
})
export class HelloWorldNgForComponent {
  name: string = ''

  submit() {
    alert(this.name + 'has been registered.');
  }
}
```

hello-world-ngmodel.component.ts

```html
<h1>Registration</h1>

<form>
    <input type="text" [(ngModel)]="name" />
    <button (click)="submit()">Submit</button>
</form>
```

hello-world-ngmodel.component.html

การส่งค่าระหว่าง Component

ในการส่งค่ากันไป-มาระหว่าง component จำเป็นต้องประกาศให้ property ที่ต้องการส่งค่าไปมาเป็น @Input() และ @Output() โดย property ที่เป็น Input ของ component จะสามารถรับค่าได้โดยการใส่วงเล็บสี่เหลี่ยมครอบ และ Output จะส่งค่าออกมาโดยใช้ความสามารถของ class EventEmitter และส่งค่ากลับมาผ่าน property ที่ใส่วงเล็บ

MainComponent

```
name: string = 'John Doe'

changeName(event: any){
  this.name = event.value;
}
```

ส่งค่าของตัวแปร name เข้า Input username

```
<name-changer
  [username]="name"
  (onSubmit)="changeName($event)"></name-changer>
```

เมื่อคลิกที่ปุ่ม submit ใน <name-changer> component จะ emit ค่าออกมาใส่ในตัวแปร $event จากนั้นนำตัวแปร $event ส่งเป็น parameter ของ handler method เพื่ออัพเดตค่า

NameChangerComponent

```
@Input() username: string
@Output() onSubmit = new EventEmitter<any>()

onClickSubmitButton(){
  this.onSubmit.emit({ value: this.username });
}
```

การส่งค่าระหว่าง Component (ต่อ)

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'main',
  templateUrl: './main.component.html'
})
export class MainComponent {
  name: string = ''

  changeName(event: any) {
    this.name = event.value;
  }
}
```

main.component.ts

```html
<h1>Edit Profile</h1>

<name-changer
    [username]="name"
    (onSubmit)="changeName($event)">
</name-changer>
```

main.component.html

การส่งค่าระหว่าง Component (ต่อ)

```ts
import { Component, Input, Output, EventEmitter }
from '@angular/core';

@Component({
  selector: 'name-changer',
  templateUrl: './name-changer.component.html'
})
export class NameChangerComponent {
  @Input() username: string = ''
  @Output() onSubmit = new EventEmitter<any>();

  onClickSubmitButton() {
    this.onSubmit.emit({ value: this.username });
  }
}
```

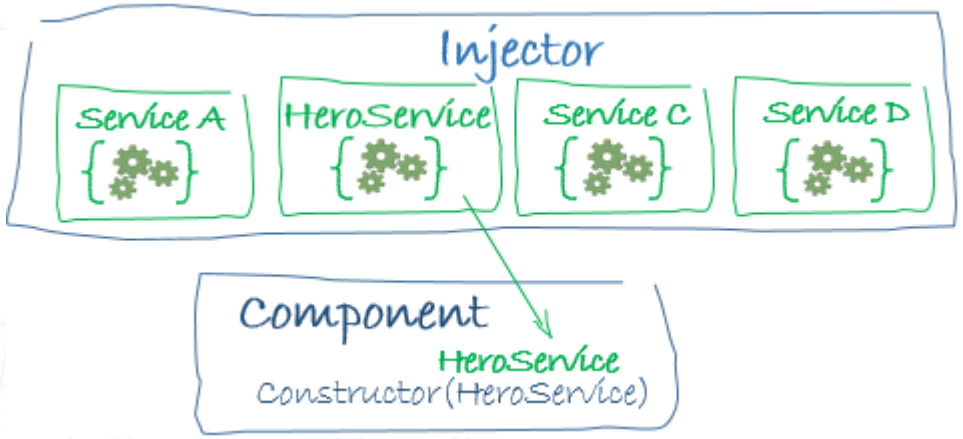
name-changer.component.ts

```html
<div>
    <input type="text" [(ngModel)]="username" />
    <button type="button"
        (click)="onClickSubmitButton()">
        Submit</button>
</div>
```

name-changer.component.html

Dependency Injection

ในกรณีที่หลายๆ Component จำเป็นต้องเข้าถึง data ชุดเดียวกัน หรือมี method บางอย่างที่ต้องใช้ร่วมกัน ทั้ง app เราสามารถสร้าง Service แบบ Injectable แล้วเพิ่มลงไปที่ฟิลด์ providers ของ xxx.module.ts เพื่อให้ component ทุกตัวใน module เดียวกันสามารถใช้งานได้



Service

สร้าง Injectable Service ขึ้นมา ในไฟล์ xxx.service.ts โดยตัวอย่างต่อไปนี้จะใช้ service ดังกล่าวเพื่อดึงค่าจาก Backend มาใช้งานใน App (มีการเรียกใช้ HttpClient)

```
import { Injectable } from '@angular/core'
import { HttpClient } from '@angular/common/http'

@Injectable({
  providedIn: 'root',
})
export class PersonService {
  constructor(private http: HttpClient) { }

  getAllPersons() {
    return this.http.get('http://backend-url/persons');
  }
}
```

person.service.ts

Service (ต่อ)

import ไฟล์ service.ts แล้วเพิ่ม Service เข้าไปใน providers ของ module

```
...
import { PersonService } from '../services/person.service'

@NgModule({
  ...
  providers: [PersonService],
  ...
})
export class PersonModule {}
```

person.module.ts

Service (ต่อ)

Import และ Inject Service เข้าไปที่ constructor ของ Component ที่ต้องการใช้งาน

```
import { Component } from '@angular/core';
import { PersonService } from '../services/person.service';

@Component({
  selector: ‘person',
  templateUrl: './person.component.html',
})
export class PersonComponent {
  persons: any[] = []

  constructor(private personService: PersonService){
    // Initialize persons
    this.personService.getAllPersons().subscribe((response) => {
      if(response.data){
        this.persons = response.data
      }
    })
  }
}
```

person.component.ts

References

- https://angular.io/docs

End of Chapter