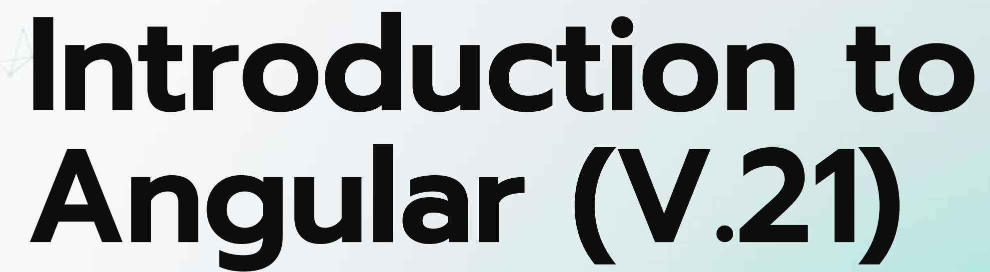
gis



Web Beginner

Overview

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

*** เอกสารนี้อ้างอิงตามเวอร์ชั่น 21 ***

Prerequisites

ก่อนใช้งาน Angular Framework คุณควรมีพื้นฐานในเรื่องต่อไปนี้

- พื้นฐาน JavaScript: https://developer.mozilla.org/docs/Web/JavaScript/Reference/Classes
- พื้นฐาน TypeScript: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html https://www.typescriptlang.org/docs/handbook/decorators.html

Installation

Angular v.18 ต้องการ Node.js v.18.19.1 หรือใหม่กว่านี้

จากนั้นติดตั้งผ่าน npm ด้วยคำสั่งต่อไปนี้

```
$ npm install –g @angular/cli
```

หลังจากติดตั้งแล้ว ตรวจสอบเวอร์ชั่นด้วยคำสั่งต่อไปนี้

```
$ ng version
```

New Angular Project

ที่ Terminal ทำการสร้าง Angular Project ใหม่ด้วยคำสั่งต่อไปนี้

```
$ ng new <project-name>
```

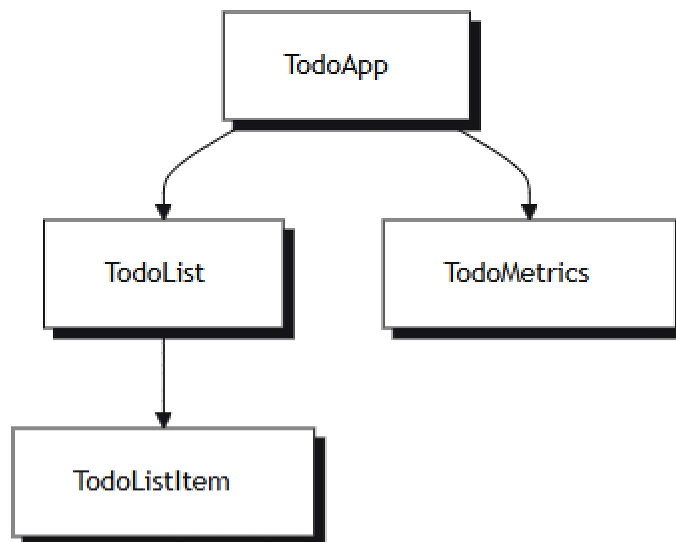
หลังจากสร้างแล้ว ให้ย้ายไปที่ Directory ที่ทำการสร้าง Project และทดสอบรัน ด้วยคำสั่ง ต่อไปนี้

```
$ ng serve
```

gis

Components

หัวใจของการพัฒนาโปรแกรมด้วย Angular คือการแยกส่วนต่างๆ ของโปรแกรมให้อยู่ในรูปของ Component ที่เชื่อมต่อถึงกัน และเข้าใจง่าย



Components (ต่อ)

ส่วนประกอบของ Component Class (TypeScript)

```
// todo-list-item.component.ts
@Component({
  standalone: true,
  selector: 'todo-list-item',
  templateUrl: './todo-list-item.component.html',
  styleUrl: './todo-list-item.component.css',
})
export class TodoListItem {
  /* Component behavior is defined in here */
}
```

standalone flag ถ้าเป็น false จะต้องทำการ declare ในไฟล์ .module.ts เพื่อให้สามารถเข้าถึง module/service/component อื่นๆ แต่ถ้าเป็น true จะสามารถ import module/service ภายใต้ @Component ได้เลย

selector คือชื่อ HTML tag ของ component นี้ เมื่อจะนำไปใช้ใน html ของ component อื่นๆ

templateUrl ชี้ไปหาไฟล์ HTML ที่เป็น view ของ component นี้

styleUrl ชี้ไปหาไฟล์ CSS ที่เป็น stylesheet ของ component นี้

Components (ต่อ)

ส่วนประกอบของ Component Class (HTML)

```html
<!-- todo-list-item.component.html -->
<li>(TODO) Read Angular Essentials Guide</li>
```

ส่วนประกอบของ Component Class (CSS)

```css
/* todo-list-item.component.css */
li {
  color: red;
  font-weight: 300;
}
```

Components (ต่อ)

ถ้าหาก Component เป็น standalone: true ให้ใส่ไว้ใน imports แล้วใส่เป็น HTML tag ใน template ตอนจะใช้งาน

```
// todo-list.component.ts
import {TodoListItem} from './todo-list-item.component.ts';
@Component({
  standalone: true,
  imports: [TodoListItem],
  template: `
    <ul>
      <todo-list-item></todo-list-item>
    </ul>
  `,
})
export class TodoList {}
```

Dynamic Content

คุณสามารถประกาศตัวแปรที่ฝั่ง TypeScript เพื่อเก็บค่า และแสดงผลใน HTML template ได้ โดยใส่

{{ VARIABLE_NAME }}

```
// todo-list.component.ts
import { TodoListItem } from './todo-list-item.component.ts';

@Component({
  standalone: true,
  imports: [TodoListItem],
  selector: 'todo-list',
  templateUrl: './todo-list.component.html',
  styleUrl: './todo-list.component.css',
})
export class TodoList {
  /* Component behavior is defined in here */
  title: string = “To-Do List”
}
```

```
<!-- todo-list.component.html -->
<h1>{{ title }}</h1>
<todo-list-item></todo-list-item>
```

Dynamic Content

ในกรณีที่ตัวแปรนั้นๆ มีการเปลี่ยนแปลงค่าบ่อยๆ และ Angular 21 ใช้ Change Detection แบบ Zoneless เป็น default ทำให้บางครั้งตัวแปรที่ใช้ใน UI อาจไม่ได้อัพเดตค่าตามที่ assign ลงไปใหม่ ในกรณีนี้เราสามารถใช้ signal เพื่อ

```
// todo-list.component.ts
import { TodoListItem } from './todo-list-item.component.ts';

@Component({
  standalone: true,
  imports: [TodoListItem],
  selector: 'todo-list',
  templateUrl: './todo-list.component.html',
  styleUrl: './todo-list.component.css',
})
export class TodoList {
  /* Component behavior is defined in here */
  title: string = "To-Do List"
}
```

```
<!-- todo-list.component.html -->
<h1>{{ title }}</h1>
<todo-list-item></todo-list-item>
```

Dynamic Content (ต่อ)

คุณสามารถประกาศตัวแปร boolean ที่ฝั่ง TypeScript และควบคุมเงื่อนไขการแสดงผลใน HTML template ได้ โดยใช้ directives *ngIf

```
// todo-list.component.ts
import { TodoListItem } from './todo-list-item.component.ts';

@Component({
  standalone: true,
  imports: [TodoListItem],
  selector: 'todo-list',
  templateUrl: './todo-list.component.html',
  styleUrl: './todo-list.component.css',
})
export class TodoList {
  /* Component behavior is defined in here */
  title: string = "To-Do List"
  displayed: boolean = true;
}
```

```
<!-- todo-list.component.html -->
<h1 *ngIf="displayed">{{ title }}</h1>
<todo-list-item></todo-list-item>
```

Dynamic Content (ต่อ)

ในกรณีที่มีตัวแปร Array ที่ฝั่ง TypeScript คุณสามารถควบคุมเพื่อวนลูปแสดงผล element ทุกตัวใน Array ใน HTML template ได้ โดยใช้ directives *ngFor

```
// todo-list-item.component.ts
@Component({
  standalone: true,
  selector: 'todo-list-item',
  templateUrl: './todo-list-item.component.html',
  styleUrl: './todo-list-item.component.css',
})
export class TodoListItem {
  items: string[] = ['Normal Task', 'Urgent Task', 'Very Urgent Task', 'Very Very Urgent Task']
}
```

```
<!-- todo-list-item.component.html -->
<ul>
<li *ngFor="let item of items">{{ item }}</li>
</ul>
```

Event Handling

คุณสามารถประกาศ method ไว้ภายใน TypeScript เพื่อดำเนินการบางอย่าง (เช่น เปลี่ยนค่าตัวแปร) และนำไปผูกกับ HTML event ต่างๆ เช่น (click) เพื่อรองรับการคลิกของผู้ใช้งาน

```
// login.component.ts
…
@Component({
  imports: [FormsModule]
  …
})
export class TodoList {
  username: string = “”;
  password: string = “”;
  signIn(){
    /* Do something with username and password */
  }
}
```

```
// login.component.html
<div>
  <input type=“text” [(ngModel)]=“username” />
  <input type=“password” [(ngModel)]=“password” />
  <button (click)=“signIn()”>Sign In</button>
</div>
```

Two-Way Binding

ในกรณีของ form element ต่างๆ เช่น <input> ที่จำเป็นต้องแสดงค่า และเก็บค่าที่กรอกในคราวเดียวกัน ควรใช้ directive ngModel เพื่อ bind ค่าระหว่างตัวแปร กับ <input> element ให้เปลี่ยนค่าไปพร้อมกัน

* ต้อง import module FormsModule ไปที่ module/component ที่ต้องการจะใช้งาน ngModel ก่อน

```
// login.component.ts
…
@Component({
  imports: [FormsModule]
  …
})
export class TodoList {
  username: string = “”;
  password: string = “”;
  signIn(){
    /* Do something with username and password */
  }
}
```

```
// login.component.html
<div>
  <input type=“text” [(ngModel)]=“username” />
  <input type=“password” [(ngModel)]=“password” />
  <button (click)=“signIn()”>Sign In</button>
</div>
```

Modules

** สำหรับการพัฒนาด้วย Angular แบบเก่า ที่ใช้ component ที่ standalone: false **

โดยทั่วไปแล้ว ในแอพของเรามักจะประกอบไปด้วย component มากกว่า 1 อัน และแต่ละอันอาจมีการรับ-ส่งข้อมูลกัน ในการที่จะทำให้แต่ละ component เชื่อมต่อหากันได้ เราจาเป็นต้องเพิ่ม 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)

```
// main.module.ts
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/froms';
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 {}
```

```
// main-routing.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 {}
```

การรับส่งค่าระหว่าง Component

ในการส่งค่ากันไปมาระหว่าง component จาเป็นต้องประกาศให้ property ที่ต้องการส่งค่าไปมาเป็น @Input() และ @Output() โดย property ที่เป็น Input ของ component จะสามารถรับค่าได้โดยการใส่วงเล็บสี่เหลี่ยมครอบและ Output จะส่งค่าออกมาโดยใช้ความสามารถของ class EventEmitter และส่งค่ากลับมาผ่าน property ที่ใส่วงเล็บ

```
// main.component.html
…
<name-changer
  [username]="name"
  (onSubmit)="changeName($event)"
></name-changer>
```

```
// main.component.ts
…
@Component({
  …
})
export class MainComponent {
  name: string = "John Doe";
  changeName(event: any) => {
    this.name = event.value;
  }
}
```

ตัวแปรที่มี decorator @Input() นำหน้า
รับค่าจาก attribute ของ component tag

เมื่อแก้ไขค่าและคลิกที่ปุ่ม submit แล้ว
component ย่อยทำการ emit เพื่อส่งค่ากลับ
component หลัก โดยส่งผ่านตัวแปรที่เป็น EventEmitter
และมี decorator @Output() นำหน้า
ส่งกลับมาที่ตัวแปร $event และให้
event handler นำไปใช้งานต่อ

```
// name-changer.component.ts
…
@Component({
  …
})
export class NameChanger {
  @Input() username: string;
  @Output() onSubmiit = new EventEmitter<any>();
  onClickSubmitButton() {
    this.onSubmt.emit({ value: this.username });
  }
}
```

```
// name-changer.component.html
<input [(ngModel)]="username" />
<button (click)="onClickSubmitButton()">Submit</button>
```

Routing

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

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

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

Routing (ต่อ)

เมื่อทำการประกาศ routes แล้ว import provideRouter และ routes จากนั้นนำไปใส่ไว้ที่ app.config.ts (แบบใหม่)
หรือใส่ routes ไว้ใน xxx-routing.module.ts จากนั้น import ที่ xxx.module.ts (แบบเก่า)

```
// app.config.ts
...
import { provideRouter } from '@angular/router';
import { routes } from './app.routes';
...
export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes)]
};
```

```
// app.module.ts
...
import { AppRoutingModule } from './app-routing.module'
...
@NgModule({
  ...
  imports: [AppRoutingModule],
  ...
})
export class AppModule {}
```

Routing (ต่อ)

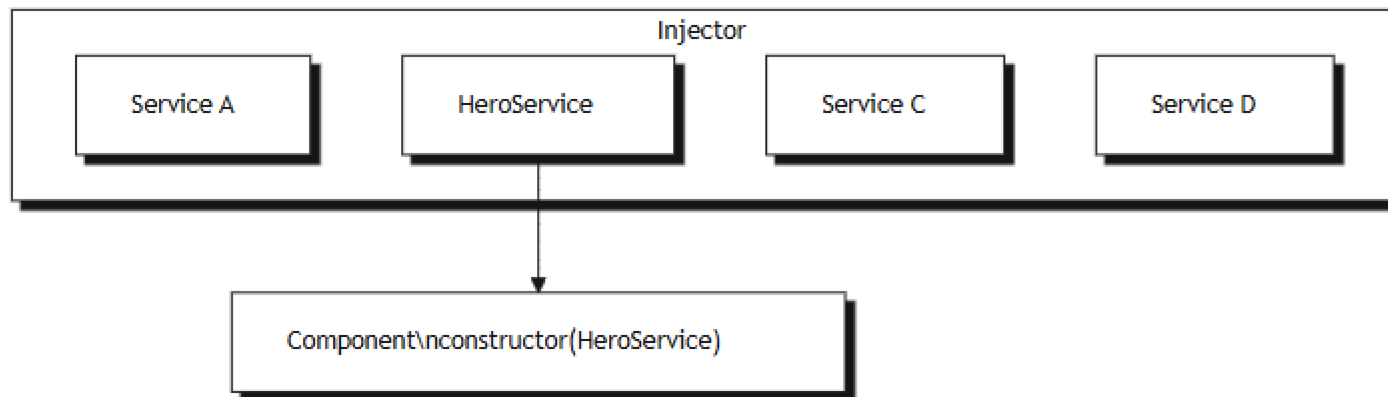
ใช้ class Router ของ Angular ในการ redirect page ไปยัง path ที่ตั้งไว้โดยก่อนอื่นต้อง inject Router service เข้ามาที่ constructor ของ component class ก่อน จากนั้นเรียกใช้งาน this.router.navigate เมื่อต้องการ redirect นอกจากนี้ ยังสามารถ redirect จาก template ได้ด้วยแท็ก <a> ที่ระบุ attribute routerLink เอาไว้

```
// navbar.component.ts
import { Router } from '@angular/router'
@Component({
 …
})
export class NavbarComponent {
 …
 constructor(private router: Router){}
 …
 toSub(){
  this.router.navigate(['sub']);
 }
}
```

```
// navbar.component.html
…
<ul>
 <li><a routerLink ="/">Main</ a></li>
 <li><a routerLink="/sub">Sub</a></li>
</ul>
…
```

Dependency Injection (DI)

- ในกรณีที่หลายๆ Component จำเป็นต้องเข้าถึง data ชุดเดียวกัน หรือมี method บางอย่างที่ต้องใช้ร่วมกันทั้ง app เราสามารถสร้าง Service แบบ Injectable แล้วเพิ่มลงไปที่ฟิลด์ providers ของ xxx.module.ts เพื่อให้ component ทุกตัวใน module เดียวกันสามารถใช้งานได้ (component standalone: false)
- สำหรับ Component ที่ standalone: true ให้เพิ่มที่ providers ภายใต้ @Component()



Service

สร้าง Injectable Service ขึ้นมา ในไฟล์ xxx.service.ts โดยตัวอย่างต่อไปนี้จะใช้ service ดังกล่าวเพื่อดึงค่าจาก Backend มาใช้งานใน App (มีการเรียกใช้ HttpClient) จากนั้น import ไฟล์ service.ts แล้วเพิ่ม Service เข้าไป ใน providers ของ module

```
// person.service.ts
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/getAllPersons');
  }
}
```

```
// person.module.ts
...
import { PersonService } from '../person.service';
@NgModule({
  ...
  providers: [PersonService],
  ...
})
export class PersonModule {}
```

Service (ต่อ)

Import และ Inject Service เข้าไปที่ constructor ของ Component (standalone: false) ที่ต้องการใช้งาน

** กรณีของ Component ที่ standalone: true ให้ใส่ PersonService ไว้ที่ providers ก่อนใช้งาน

```
// person.module.ts
import { Component } from '@angular/core';
import { PersonService } from './person.service';
@Component({
  …
  standalone: false
})
export class PersonComponent {
  persons: any[] = []
  constructor(private personService : PersonService){
    // Initialize persons
    this.personService.getAllPersons().subscribe((response: any) => {
      if(response.data){
        this.persons = response.data
      }
    })
  }
}
```

```
// person.module.ts
import { Component } from '@angular/core';
import { PersonService } from './person.service';
@Component({
  …
  standalone: true,
  providers: [PersonService]
})
export class PersonComponent {
  persons: any[] = []
  constructor(private personService : PersonService){
    // Initialize persons
    this.personService.getAllPersons().subscribe((response: any) => {
      if(response.data){
        this.persons = response.data
      }
    })
  }
}
```

End of Presentation

gis