พื้นฐานการใช้งาน ANGULAR

สร้างเว็บแอปพลิเคชันให้ได้รอบทิศ

PHP,MySQL,Firebase,Bootstrap,Material Design,PrimeNg,NgRx

ดร.ธีระพล ลิ้มศรัทธา

พื้นฐานการใช้งาน Angular

โดย ธีระพล ลิ้มศรัทธา

ราคา 280 บาท
จำนวนหน้า 357 หน้า
พิมพ์ครั้งแรก สิงหาคม พุทธศักราช 2567 (ฉบับ e-book)
ออกแบบปกโดย เตชพัฒน์ จรูญพันธุ์เกษม
จัดทำโดย ดร.ธีระพล ลิ้มศรัทธา
e-mail: theerapol.lim@gmail.com

รุ่นที่ 1.3

คำนำ

เทคโนโลยีเว็บ มีการเปลี่ยนแปลงอย่างรวดเร็ว เช่น การเขียนโดยใช้ภาษา JavaScript ได้มีการเปลี่ยนมาใช้ภาษา TypeScript เนื่องด้วยกำหนดชนิดข้อมูลได้ ซึ่งจะช่วยลดความผิดพลาดขณะเขียนโปรแกรมได้ง่าย รวมทั้งรูปแบบสร้างคลาส หรืออินเทอร์เฟสที่ทำได้ดีขึ้นที่ลดขั้นตอนการสร้างออบเจ็กต์ได้มาก หรือจากเดิมเคยส่งเสริมการใช้งานในรูปแบบโมดูล แต่ล่าสุดส่งเสริมการทำงานแบบ standalone และส่งเสริมการเขียนโปรแกรมเชิงฟังก์ชัน ซึ่งจะเป็นได้ว่าแนวทางการเขียนเป็นคลาส หลายตัวถูกแทนที่ด้วยฟังก์ชันที่เรียกใช้งานได้ทันที เทคโนโลยีทางด้าน User Interface ก็เช่นกัน ต้องติดตามบ่อย ๆ และอาจมีตัวใหม่เกิดขึ้นที่ดีกว่าจนอาจได้รับความนิยมใช้มากกว่า หนังสือเล่มนี้นำเสนอแนวทางพื้นฐานในการใช้งานแองกูล่าร์ ที่พร้อมให้ศึกษาต่อด้วยตนเองต่อไปได้เร็วขึ้น พร้อมทั้งยังรวมความสามารถของไลบรารี่อื่น ๆ หรือโปรแกรมประกอบอื่น ๆ เช่น Firebase และ MySQL ที่ใช้ทำงานกับฐานข้อมูลประเภท NoSQL และ SQL การทำงานรวมกับ PHP Web Server ที่ทำงานเป็นส่วนหลังบ้าน การสร้าง Rest API ของ PHP เพื่อรองรับการให้บริการของส่วนหน้าบ้านซึ่งก็คือ แองกูล่าร์ นอกจากนี้ยังมีส่วนการใช้งาน User Interface ทั้งของ Bootstrap, Material Design และ PrimeNG เพื่อความหลากหลายในการใช้งาน สำหรับความรู้พื้นฐานของ แองกู-ล่าร์ สามารถอ่านได้ตั้งแต่บทที่ 1 – 8 ส่วนบทต่อไปถือว่า เป็นการฝึกฝนการใช้งานกับโปรแกรมอื่น ๆ ซึ่งก็จำเป็น เพราะคงไม่มีใครใช้งานเพียง แองกูล่าร์ อย่างเดียว และโดยลำพัง

หนังสือทางเทคโนโลยีเว็บเล่มนี้ อาจมีบางอย่างผิดพลาด ทั้งผิดพลาดจากความเข้าใจ การเขียนโปรแกรม รวมทั้งความผิดพลาดจากความล้าหลังของเทคโนโลยีเมื่อเวลาผ่านไป ผู้เขียนขออภัย และยินดีรับคำติชมและคำแนะนำได้ตามช่องทางอีเมลผู้เขียน

ธีระพล ลิ้มศรัทธา

theerapol.lim@gmail.com

สารบัญย่อ

สารบัญ

หน้า

3. Reactive Form

4. Template Form

ภาคผนวก

ก. พื้นฐานภาษา TypeScript

คำแนะนำในการใช้หนังสือ

หนังสือเล่มนี้ มีข้อกำหนดหลายอย่างที่ควรทำความเข้าใจให้ตรงกัน ดังนี้

- ใช้ลำดับการอ้างอิงของโปรแกรม ในชื่อ **Code xx** โดย xx คือเลขลำดับของ **Code** ในแต่ละบท และแต่ละ **Code** ที่อ้างอิง จะมีชื่อไฟล์ที่ใช้พิมพ์กำกับไปด้วย เช่น

  Code. 1.1 src/app/app.component.ts (standalone)

  จะใช้โปรแกรมนี้พิมพ์ในไฟล์ app.compoent.ts และยังแจ้งบอกด้วยว่าเป็นกับระบบ standalone
- **standalone** กับ **module** ในหนังสือเล่มนี้ในช่วงแรกจะให้แบบ standalone **แต่ตั้งแต่บทที่ 3** เป็นต้นไป ถ้าไม่ระบุจะหมายถึงใช้แบบ **module**
- ในตัวอย่างการใช้ **Code xx** จะไม่เขียนแสดงโปรแกรมทั้งหมด จะแสดงส่วนที่สำคัญ หรือส่วนที่มีการเปลี่ยนแปลงเท่านั้น การแสดงเพียงบางส่วนไม่ได้หมายความจะต้องลบส่วนอื่น ๆ ที่มีก่อนหน้านี้ออก ให้แก้ไข หรือเพิ่มเติม เฉพาะส่วนที่มีการเปลี่ยนแปลงเท่านั้น
- **CLI** เป็นการใช้คำสั่งผ่าน Command Line Interface (ในโหมด Command Prompt) มีเลขลำดับ เป็น **CLI xx** โดย xx ก็คือลำดับการอ้างอิงของ **CLI** ในแต่ละบท โดยมี เครื่องหมาย “>” แทนจุดเริ่มต้นในพิมพ์คำสั่ง เช่น

  > ng g c User

  เวลาพิมพ์คำสั่ง จึงพิมพ์แค่ **ng g c User** ไม่ต้องพิมพ์เครื่องหมาย “>” ไปด้วย
- ควรอ่านตั้งแต่บทแรก เรียงตามลำดับ โดยเฉพาะบทที่ 1 ถึง 8 อาจข้ามบทที่ 7 ไปก็ได้ ส่วนบทที่เหลืออาจแยกอ่านตามความสนใจใช้งาน สำหรับผู้มีประสบการณ์กับ แองกูล่าร์ อาจเลือกอ่านตามบทที่สนใจ
- สำหรับผู้ยังไม่มีประสบการณ์ในการเขียนโปรแกรมด้วย ภาษา **TypeScript** อาจทบทวนภาษานี้จะภาคผนวก และสำหรับผู้มีพื้นฐานจากภาษาอื่น ๆ มาก่อน แนะนำให้อ่านเรื่อง interface เพราะมีหลายอย่างที่ต่างจากภาษาอื่น ๆ และอ่านในส่วนการสร้างออบเจ็กต์อย่างย่อ ที่สร้างสามาชิกภายในการสตรักเตอร์ เช่น

  constructor(private id:number)

  จะมีความหมายเป็นการสร้างสมาชิกพร้อมกับเป็นตัวแปรของคอนสตรักเตอร์ นอกจากนี้การเขียน**โปรแกรมเชิงฟังก์ชัน หรือแลมบ์ดา** ก็มีบทบาทมาก เพราะยุคนี้เน้นการเขียนโปรแกรมเชิงฟังก์ชันกันมาก
- แองกูล่าร์ มีการเปลี่ยนแปลงที่บ่อยมาก ค่าปริยายต่าง ๆ ในรุ่นใหม่อาจปรับเปลี่ยนได้ ควรศึกษาจากเว็บไซต์อย่างเป็นทางการของแองกูล่าร์
- การรายงานความผิดพลาด และเนื้อหาเสริมผู้เขียนจะให้ข้อมูลเพิ่มเติม ที่เว็บไซต์: http://www.theerapone.com/#!/book

การมาใหม่ของ แองกูล่าร์ (Angular) มาพร้อมกับโปรแกรมขนาดใหญ่โต จนน่าตกใจ จากเดิมแองกูล่าร์ เจเอส (AngularJs) มีเพียงไฟล์ angular.min.js ตัวเดียว ของใหม่เริ่มต้นก็ต้องใช้ไฟล์จำนวนมาก ได้แก่โมดูลจำนวนมาก ที่จำเป็น ที่ใช้สร้างเว็บใช้งานขนาดใหญ่ และโมดูลมีที่เกินความจำเป็น โดยเฉพาะหากต้องการสร้างเว็บใช้งานขนาดเล็ก ซึ่งต่อไปเมื่อสร้างเสร็จรอการใช้งาน ก็จะเหลือแต่โมดูลที่จำเป็นต้องใช้งาน ในจำนวนไฟล์ไม่มากเหมือนตอนแรกที่ใช้สร้างเว็บ ซึ่งแน่นอนว่า จากไฟล์ประมาณ 250 MB จะเหลือประมาณ 250 KB ลดลงไปหลายร้อยเท่าเลยทีเดียว และส่วนที่เป็นไฟล์ที่สร้างเพื่อใช้งานเสร็จแล้วนี้ เป็นไฟล์ที่อ่านแทบไม่รู้เรื่องเลย เพราะผ่านกระบวนการลดช่องว่าง ปรับเปลี่ยนตัวแปร ตัดส่วนที่ไปใช้งาน และรวมหลายไฟล์เป็นไฟล์เดียวกัน อย่างไรก็ตาม การเปลี่ยนครั้งสำคัญก็เกิดขึ้นอีกครั้ง เมื่อ มีการแนะนำให้ใช้งาน คอมโพเน้นท์ ในรูปแบบ ยืนเดี่ยว (standalone) แทนที่จะเป็นโมดูล การเปลี่ยนแปลงเป็นอย่างไร เราจะมาสำรวจกัน

ในบทนี้จะนำแนะการเริ่มต้นการใช้งานแองกูล่าร์ ดังจะทำความเข้าใจในประเด็นต่อไปนี้

- การติดตั้งโปรแกรมแองกูล่าร์
- การสร้างแอปพลิเคชันแองกูล่าร์ แบบ standalone และ NgModule
- การใช้งานผ่าน Visual Studio Code
- สำรวจส่วนประกอบต่าง ๆ ของแองกูล่าร์
- การนำไปใช้งานจริงบนเซิร์บเวอร์

พัฒนาการของแองกูล่าร์

การมาใหม่จากของแองกูล่าร์เริ่มต้นที่รุ่น 2 เป็นต้นไป มีการปรับเปลี่ยนอย่างมากทั้งในรูปแบบสถาปัตยกรรม ภาษาที่ใช้ในการพัฒนา แน่นอนว่าต้องใช้เวลาในการเรียนรู้ใหม่เกือบหมด แต่ทุกอย่างต้องมีการพัฒนา การเริ่มต้นศึกษาในสิ่งใหม่ทุกคนก็เริ่มต้นเหมือนกัน แต่เราหลีกหนีไม่พ้นถ้าต้องการพัฒนาระบบเว็บสมัยใหม่ ดังได้เห็นต่อไปว่าหลาย ๆ ส่วนของระบบเว็บ ไม่จะเป็น UI, หรือไลบรารีอื่น ๆ ก็ผูกติดใช้งานกับแองกูล่ารุ่นใหม่กันเกือบหมด

ตาราง 1.1. เปรียบเทียบ AngularJs กับ Angular

คุณลักษณะ	**AngularJs**	**Angular**
รุ่นที่	**1.x**	**2 ขึ้นไป**
ภาษา	JavaScript	TypeScript
ระบบ	Library	Framework
รูปแบบสถาปัตยกรรม	Controller	Component

คุณลักษณะ	AngularJs	Angular
ใช้เป็นโมบายแอปพลิเคชัน	ต้องปรับปรุงหลายส่วน	ได้
ใช้งาน CLI	ไม่ได้	ได้
ทำงานบนเซิร์บเวอร์	ไม่จำเป็น	จำเป็น
SEO	ไม่ได้	ได้
Lazy loading	ไม่ได้	ได้
จำเพาะเซิร์บเวอร์	ไม่จำเป็น	ใช้กับ NodeJs

Module กับ Standalone

โมดูล (Module) มีการใช้งานมาตั้งแต่แรกเริ่ม ที่เป็นศูนย์ลงทะเบียนโปรแกรมต่าง ๆ โดยมี NgModule เป็นตัวรับลงทะเบียน แต่พอมาในแองกูล่าร์รุ่น 14 ขึ้นไป คณะทำงานของแองกูล่าร์ได้เพิ่ม รูปแบบทำงานแบบยืนเดี่ยว เนื่องจากว่าต้องการลดความซับซ้อนและขนาด โดยไฟล์เดียวสามารถนำเข้าไลบรารีต่าง ๆ มาใช้ได้โดยตรงโดยไม่ต้องลงทะเบียนกับ NgModule อีกต่อไป

ข้อได้เปรียบในการใช้งานแบบยืนเดี่ยว

- ไม่มีเรื่องโมดูล ซึ่งอาจจะดูยากสำหรับผู้เริ่มต้น ดังนั้นจึงเป็นการลดความยุ่งยากลงไป จึงทำให้เรียนรู้ได้ง่ายสำหรับผู้เริ่มต้น
- ลดจำนวนไฟล์ที่ใช้จัดการโมดูล ทำให้ลดการเขียนโปรแกรมฝากไว้ในส่วนต่าง ๆ ทำให้ภาพรวมคือลดขนาดโปรแกรมรวมได้
- ด้วยตัวไฟล์ยืนเดี่ยวเอง ทำให้อ่านโปรแกรมได้ง่าย โดยไม่จำเป็นต้องดูส่วนโปรแกรมอื่น และง่ายต่อการดูแล
- นำไปใช้ซ้ำได้ง่าย ด้วยตัวเองมีความอิสระกว่า เพราะทุกอย่างอยู่ในตัวเองหมดแล้ว

ข้อเสียในการใช้งานแบบยืนเดี่ยว

- ทุกอย่างต้องจัดการด้วยตนเองหมด อันนี้ดูเหมือนยาก เช่น การนำเข้าไลบรารี่หรือโปรแกรมที่ต้องใช้งาน แต่ด้วยความเก่งของตัวเขียนโปรแกรมสมัยใหม่ เช่น Visual Studio Code จะแจ้งส่วนที่ขาดไปทำให้ไม่ต้องจำทุกอย่าง แต่ก็ไม่ทุกอย่างที่จะช่วยได้หมด
- เนื่องจากการส่งเสริมการยืนเดี่ยว อาจมีความยุ่งยากในการสื่อสารกับส่วนประกอบอื่น เช่น การจัดการเรื่อง สถานะ (state)
- เมื่อต้องการใช้กับงานที่มีขนาดใหญ่ มีความซับซ้อน การจัดการจะทำได้ยาก อันเป็นผลมาจากการสื่อสารระหว่างกันทำได้ยาก เพราะต่างคนต่างก็ยืนแยกเดี่ยว

ติดตั้ง Angular

แองกูล่าร์ใช้ คำสั่ง CLI (Command Line Interface) ในการสร้างแอปพลิเคชัน และสร้างส่วนประกอบต่าง ๆ ในการพัฒนานาแอปพลิเคชัน แต่ก่อนที่จะติดตั้ง Angular จะต้องติดตั้ง NodeJS ซึ่งเป็นเซิร์บเวอร์ที่ทำงานด้วยภาษา JavaScript ตัวโปรแกรมติดตั้งหาได้จากเว็บไซต์ https://nodejs.org/en/

หลังจากติดตั้ง NodeJs แล้วแต่ใช้คอมมานไลน์ สำหรับระบบ Windows ให้เปิด Command Prompt และตอนนี้ต้องให้คอมพิวเตอร์เชื่อมต่อกับอินเทอร์เน็ตได้ด้วย แล้วพิมพ์:

CLI 1.1

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

ผลการติดตั้ง จะแจ้งเลขที่รุ่นไว้ด้วย (ถ้าไม่แสดงเลขรุ่น ใช้คำสั่ง ng version) แค่นี้ก็ถือว่าการลงโปรแกรมสมบูรณ์แล้ว กรณีต้องการปรับปรุง เป็นรุ่นใหม่ ให้ถอดแองกูล่าร์ออกก่อน แล้วติดตั้งใหม่

CLI 1.2

```
> npm uninstall -g @angular/cli
> npm cache clean --force
```

เริ่มสร้างเว็บแอปพลิเคชัน

ก่อนการสร้างเว็บแอปฯ ด้วยแองกูล่าร์จะต้องมีพื้นที่มากพอ ซึ่งไฟล์ทั้งหมดก็ใช้ประมาณ 250 MB การติดตั้งใช้ผ่าน CLI และต้องต่อเชื่อมอินเทอร์เน็ตด้วย มาเริ่มต้นสร้างกันตามลำดับดังนี้

1. เมื่อสร้างแอปพลิเคชัน (สำหรับรุ่น 17) จะมีคำถามให้เลือก อย่างแรกที่ถามคือจะใช้รูปแบบ Style Sheet แบบใด ในที่เลือกแบบทั่วไปคือ CSS ให้กด Enter คำถามต่อมาจะถามว่า SSR และ SSG หรือไม่ให้ตอบ N หรือ y ก็ได้ หรือ กด Enter ทุกรายการ จะเป็นการเลือกตามค่าปริยาย และรอจนกว่าจะสร้างโปรเจ็กต์เสร็จ

```
> ng new first-app
```

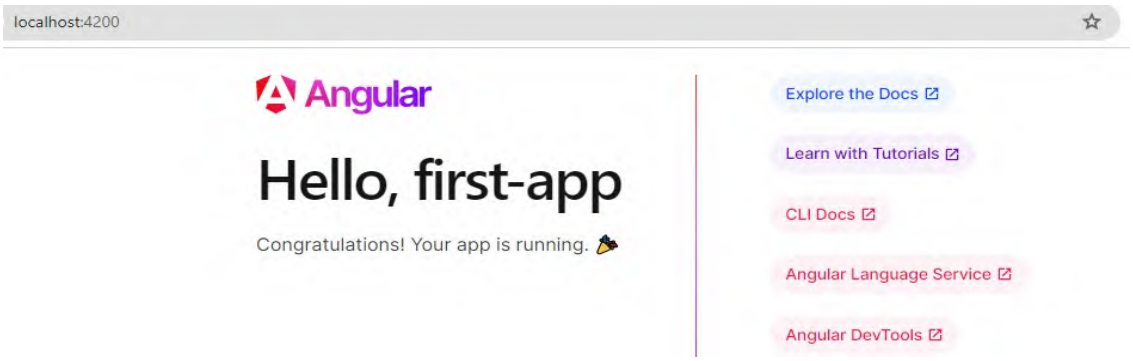
2. ทดสอบการทำงานผ่านเว็บเซิร์บเวอร์ที่มีในตัวอยู่แล้ว ซึ่งจะเปิดเว็บไซต์เริ่มต้นผ่าน localhost:4200

```
> cd first-app
> ng serve –open
```

ผลการสร้างจะได้โหมดต่าง ที่ต่างจากรุ่นเดิม ดังนี้

- กำหนดให้ใช้เป็น standalone = true ผลการการใช้ standalone จะทำให้ไม่มีไฟล์ app.module.ts อีกต่อไป
- กำหนดการใช้เส้นทาง ดูจากการนำเข้า RouterOutlet

- กำหนดการใช้ภาษา TS ที่เคร่งครัดไวยากรณ์ strict = true ดูที่ไฟล์ tsconfig.json

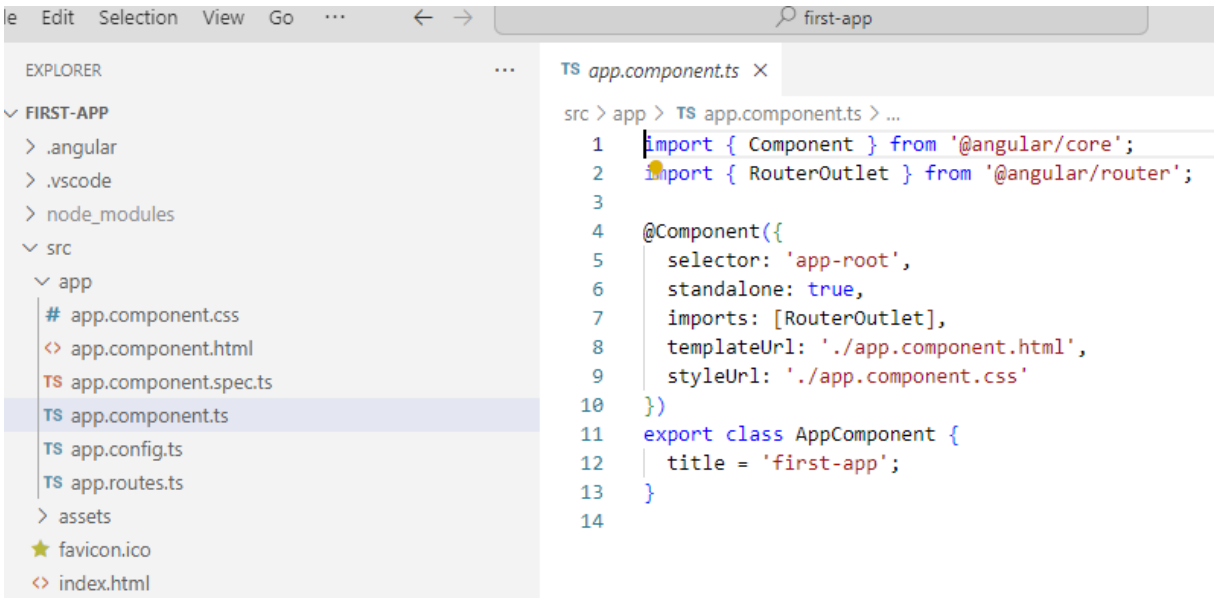


รูป 1.1 หน้าเว็บเริ่มต้นจากผลการสร้างของแองกูล่าร์ (version 17)

การใช้งานผ่าน Visual Studio Code

การสร้างส่วนประกอบต่าง ๆ จะสร้างผ่าน CLI การแก้ไขไฟล์จะพิมพ์ผ่านโปรแกรมพิมพ์งานทั่วไป เช่น โปรแกรม Notepad++ หรือ Notepad ธรรมดาก็ได้ แต่ที่อยากแนะนำให้ใช้งานผ่านโปรแกรม Visual Studio Code (VS Code) ซึ่งหาดาวน์โหลดได้โดยไม่มีค่าบริการ โปรแกรมนี้จะช่วยตรวจสอบการพิมพ์ผิดได้ดี และยังสามารถพิมพ์ผ่าน CLI ของ VS Code ได้ด้วย

การใช้งาน ให้เปิดโฟลเดอร์งานแองกูล่าร์ อย่างในกรณีงานแองกูล่าร์ที่ได้สร้าง ให้เปิดโฟลเดอร์ first-app ซึ่งได้ผลดังรูปต่อไปนี้



รูป 1.2 การใช้งานผ่านโปรแกรม Visual Studio Code (Standalone)

จากหน้าต่างโปรแกรม VS Code เราสามารถเปิดหน้าต่าง TERMINAL ได้ผ่าน เมนู View > Terminal ซึ่งทำให้เราใช้ CLI ผ่าน Terminal นี้ได้

ในกรณีที่เราเปิด VS Code ไม่ได้อยู่ในโหมด Administrator การเรียกใช้คำสั่งผ่าน Terminal อาจถูกป้องกันการทำงาน และมีการแจ้งให้เข้าไปหาข้อมูลเพิ่มเต็มที่

```
https:/go.microsoft.com/fwlink/?LinkID=135170
```

เมื่อเข้าสืบค้นข้อมูล ก็จะพบตัวอย่างการกำหนดนโยบายความปลอดภัยต่าง ๆ ถ้าทดลองใช้คำสั่งผ่าน CLI ของ Windows PowerShell

CLI 1.3

```
> Get-ExecutionPolicy -Scope CurrentUser
```

ผลคือ Undefined เราจึงต้องกำหนดนโยบายสำหรับผู้ใช้ปัจจุบันใหม่ดังนี้

CLI 1.4

```
> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

ซึ่งตามด้วยคำถามให้ยืนยันนโยบายใหม่ ให้ตอบ Y เพียงเท่านี้ก็จะสามารถใช้คำสั่งของแองกูล่าร์ผ่าน Terminal ของ VS Code ได้

ติดตั้ง Extension เพิ่มเติม เพื่อการเขียนโปรแกรมได้ง่ายขึ้น และแสดงข้อมูลคำแนะนำในการเขียนโปรแกรม ซึ่งมีหลายตัว แต่ตัวเริ่มต้นที่น่าติดตั้งไว้คือ Angular Language Service

รูป 1.3 Extension เพิ่มเติมที่น่าติดตั้งใน VS Code

ยืนเดี่ยว

ดูเหมือนในรุ่น 17 ของแองกูล่าร์ จะใช้คำปริยายเป็นแบบ ยืนเดี่ยว หรือ Standalone แทนที่จะเป็นแบบ NgModule ดังนั้นแล้วเรามาทำความรู้จักรูปแบบใหม่นี้กัน ซึ่งรูปแบบนี้มีตั้งแต่ แองกูล่าร์ 14 แล้ว

ยืนเดี่ยว จัดเป็นวิธีการสร้างอย่างง่ายในการสร้างแอปพลิเคชัน เพราะลดการใช้งานในรูปแบบโมดูล ที่เกินความจำเป็น ทุกอย่างประกาศใช้ในไฟล์คอมโพเน้นท์เพียงไฟล์เดียว ในความง่ายก็มีความยากอยู่บ้าง คือต่อไป จะใช้งานคุณสมบัติใดของแองกูล่าร์ เราจะต้องนำเข้า (import) ชื่อโมดูลที่ถูกต้อง แม้แต่โมดูลใช้งานประจำก็ไม่ใส่ให้มา จะต้องเขียนนำเข้าเอง จากที่ไม่เคยจำ ตอนนี้จะต้องจำให้ได้ว่าคุณสมบัตินี้ใช้งานผ่านโมดูลใด

โครงสร้างไฟล์หลัก

สิ่งสำคัญที่แองกูล่าร์ คือแนวคิด คอมโพเน้นท์ หน้าเว็บหนึ่งคือ คอมโพเน้นท์หนึ่ง คอมโพเน้นท์จะกำหนดหน้าเว็บ หรือวิว (View) ซึ่งจะเรียกใช้งานบริการอื่น ๆ ได้ การเรียกใช้งานผ่านวิธีการที่เรียกกันว่าเดคคอเรเตอร์ (Decorator) เป็นรูปแบบการออกแบบหนึ่งเพื่อเสริมคุณสมบัติให้หน้าเว็บหนึ่งจะต้องใช้คุณสมบัติอะไรได้บ้าง

ในโครงสร้างของไฟล์ที่สร้างมาให้ src/app จะพบไฟล์สำคัญ คือ:

1. app.component.css เป็นไฟล์ CSS ใช้เพื่อจัดรูปแบบเอกสาร HTML
2. app.component.html เป็นไฟล์หน้าเว็บเริ่มต้น
3. app.component.ts เป็นไฟล์ภาษา TypeScript อยู่ในรูปแบบคลาส ใช้ดำเนินการคอมโพเน้นท์
4. app.module.ts กรณีใช้งานแบบโมดูล ไฟล์นี้ใช้ จัดการใช้งานโมดูล ที่จะต้องใช้ในคอมโพเน้นท์ต่าง ๆ
5. app.config.ts กรณีใช้งานแบบยืนเดี่ยว ไฟล์นี้ใช้ จัดการใช้งานโมดูล
6. app.routes.ts กรณีใช้งานแบบยืนเดี่ยว จะเป็นไฟล์ที่เก็บเส้นทางที่เดินทางไปยังคอมโพเน้นท์หรือหน้าเว็บต่าง ๆ แต่ถ้าใช้งานแบบโมดูลได้ไฟล์ app-routing.module.ts

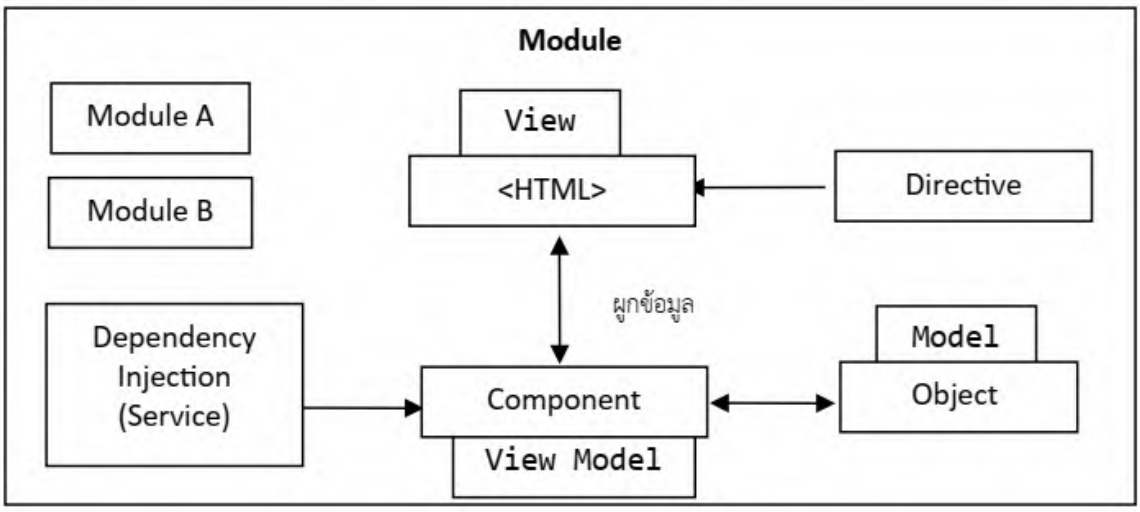
ดังนั้นแล้ว โครงสร้างระบบ หรือสถาปัตยกรรมของแองกูล่าร์ ประกอบด้วยส่วนคอมโพเน้นท์เป็นแกนกลางหลัก มีการเรียกใช้งานงานบริการในรูปแบบ Dependency Injection งานบริการจึงเป็นสิ่งอิสระ ไฟล์ JavaScript หรือโมดูลต่าง ๆ ในความหมาย JavaScript หนึ่งโมดูลคือหนึ่งไฟล์ แต่โมดูลในแองกูล่าร์ ถือเป็นไลบรารีไฟล์ของ JavaScript หลาย ๆ ไฟล์

คอมโพเน้นท์ ทำงานร่วมกับ เทมเพลต (Template) โดยข้อมูลออบเจ็กต์ถือเป็นโมเดล (Model) ที่กำหนดไว้ในคอมโพเน้นท์ นำมาอ้างอิงในส่วนแสดงผล (View) ที่อยู่ในรูปแบบไฟล์ HTML และไฟล์ CSS ภายในหน้าแสดงผลนี้ เรียกข้อมูล โดยใช้คำสั่งฝัง หรือคำสั่งเฉพาะของแองกูล่าร์ (Directive) หรือใช้การผูกข้อมูล (Data/Property binding) หรือข้อมูลมีลักษณะไดนามิกได้ด้วยการใช้ผ่านการผูกกับอีเว้นท์ (Event Binding) การผูกข้อมูลนี้รวมหน้าแสดงผลและโมเดลไว้ด้วยกันโดยคอมโพ-เน้นท์ เรียกได้ว่าเป็นแสดงโมเดล (View Model) ทำให้ชื่อรูปแบบสถาปัตยกรรมว่า Model - View - View Model หรือเรียกย่อ ๆ ว่า MVVM

สำรวจไฟล์คอมโพเน้นท์

คอมโพเน้นท์ส่วนส่วนประกอบหลักของแองกูล่าร์ ที่ใช้ควบคุมการแสดงผล แทนการใช้คอนโทรเลอร์ (Controller) เมื่อเทียบกับ แองกูล่าร์เจเอส ไฟล์ประเภท component.ts จึงเป็นการนิยามข้อมูลที่ต้องการใช้ในการแสดงผล ซึ่งประกอบได้ด้วยไฟล์ HTML และไฟล์ CSS โดยชื่ออิลีเม้นท์ (Element) ที่เป็น

ที่ส่วน HTML จะไปแสดงผล ลักษณะการกำหนดนี้ถือเป็นเมตาดาต้าของคอมโพเน้นท์ และใช้เป็นส่วนเพิ่มไปยังคลาส เรียกส่วนเพิ่มนี้ว่า เดคคอเรเตอร์ ที่กำกับด้วยคำสำคัญ @Component



รูป 1.4 สถาปัตยกรรมเว็บแอปฯ ของ Angular ในรูปแบบ โมดูล

จากไฟล์ app.component.ts เป็นคอมโพเน้นท์ชื่อ AppComponent กำหนดเป็นคลาสหนึ่ง การสร้างคลาสประเภทคอมโพเน้นท์ จะต้องประกาศ เดคอเรเตอร์ ว่า @Component และกำหนดเมตาดาต้า ในตัวอย่างนี้กำหนดส่วนที่แสดงผลในอีลีเม้นท์ <app-root> ซึ่งอยู่ในไฟล์หลัก (src/index.html) และนำไฟล์ app/app.component.html ไปใช้ใน อิลีเม้นท์ <app-root> พร้อมกับควบคุมรูปแบบการแสดงผลด้วยไฟล์ app/app.component.css

สำหรับการใช้งานแบบ ยืนเดี่ยว จะมีคำสำคัญเพิ่ม คือ standalone = true และถ้ามีการใช้งานเส้นทางด้วย จะต้องนำเข้า RouterOutlet ด้วย และนำเข้าในส่วนที่เป็นอาร์เรย์ (imports) รายการที่ต้องนำเข้า จะเป็น component, directive, pipe, NgModule ซึ่งในต้นเริ่มต้นนี้เราไม่ได้งานอะไรส่วนนี้

Code 1.1. src/app/app.component.ts (standalone)

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'first-app';
}
```

คลาสนี้จะใช้งานได้จะต้องประกาศคำสำคัญคือ export เพื่อให้นำไปใช้ได้ในโมดูลนี้หรือคอมโพเน้นท์อื่น การนำคอมโพเน้นท์ไปใช้จะใช้คำสำคัญว่า import ในตัวอย่างนี้ใช้คอมเน้นท์ Component ซึ่งอยู่ในไลบรารี @angular/core นอกจากนี้คลาส AppComponent ประกาศตัวแปร title ซึ่งจะกลายเป็นตัวแปรของออบเจ็กต์ ประจำคลาสนี้

การใช้งานแบบยืนเดี่ยว ถึงแม้ไม่ระบบโมดูลให้ใช้งาน แต่ก็มีไฟล์ที่คอยจัดการโมดูล และลงทะเบียนเส้นทาง คือ app.config.ts โดยค่าปริยาย ได้ลงทะเบียนเส้นทางผ่าน providers ให้แล้ว

Code 1.2. src/app/app.config.ts (standalone)

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes)]
};
```

โมดูล

นอกจากใช้คอมโพเน้นท์เป็นแกนกลางของระบบแล้ว ยังมีส่วนประกอบภายนอกที่คอยเรียกใช้งานไลบรารีในคอมโพเน้นท์ เรียกส่วนนี้ว่าโมดูล โมดูลประกอบด้วยคอมโพเน้นท์ และการเรียกใช้งานโมดูลอื่น ๆ(เมื่อใช้งานแองกูล่าร์ ในรูปแบบ โมดูล) ระบบเว็บแอปฯ หนึ่ง ๆ ของแองกูล่าร์ จะมีอย่างน้อยหนึ่งโมดูล ซึ่งเรียกว่า รูทโมดูล (Root module) หรือโมดูลหลัก และเป็นธรรมเนียมที่จะใช้ชื่อว่า AppModule โดยโมดูลหลักนี้ ทำงานเริ่มต้นทำงานเรียกคอมโพเน้นท์หลักให้ทำงานอีกที ผ่านการประกาศ @NgModule ซึ่งเป็นเดคอเรเตอร์ประเภทโมดูล ของคลาส AppModule

เดคอเรเตอร์ นี้จะประกาศเมตาดาต้าที่จะต้องนำไปใช้งานในระดับโมดูล เมตาดาต้า มีหลายอย่างที่น่าสังเกตว่าข้อมูลใดเป็นอาร์เรย์ จะเขียนในรูป พหูพจน์ (Plural) โดยมากจะใช้อักษร s ต่อท้าย ดังประกาศใช้เช่น:

- declarations เป็นการประกาศคอมโพเน้นท์ ซึ่งอาจจะมี directive, pip ต่าง ๆ นอกเหนือจากคอมโพเน้นท์ ให้อยู่ในโมดูลนี้
- imports เป็นการนำเข้าโมดูลอื่น ๆ ที่จะใช้ในโมดูลนี้ ในตัวอย่างนี้ได้นำเข้าโมดูล BrowserModule ซึ่งใช้ทำงาน กับเบราเซอร์ เช่น Google Chrome หรือ FireFox แองกูล่าร์ไม่จำเป็นต้องทำงานกับเบราเซอร์ เช่น ทำงานกับเซิร์บเวอร์ NodeJs ก็ไม่จำเป็นต้องนำเข้า BrowserModule
- providers เป็นการประกาศการสร้างงานบริการ (service) ที่โมดูลนี้มีให้ใช้

- bootstrap เป็นประกาศคอมโพเน้นท์หลัก ที่จะเริ่มทำงาน คอมโพเน้นท์หลัก (root) และมักใช้ชื่อว่า AppComponent ในหนึ่งแอปพลิเคชันของแองกูล่าร์จะมีเพียงหนึ่งโมดูลเท่านั้นที่จะมี bootstrap

Code 1.3. src/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [ AppComponent ],
  imports: [  BrowserModule  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

การใช้สร้างแอปพลิเคชัน ในรูปแบบโมดูลจะมีคำสั่งที่เพิ่มเติมอีกเล็กน้อย ซึ่งต่อไปไม่แน่ใจว่า จะเปลี่ยนแปลงอะไรอีก ต้องติดตามที่หน้าเว็บหลักของแองกูล่าร์

CLI 1.5

```
> ng new first-app --no-standalone
```

ต่อไปก็มาดู รายละเอียดไฟล์ที่เพิ่งสร้าง โดยที่ยังไม่เขียนอะไรเพิ่มลงไป ในรายการบรรทัดแรก ๆ จะเป็นการนำเข้าไลบรารี ที่โมดูลจะเรียกใช้งาน โดยรายไลบรารีแรก เป็น ไลบรารี platform-browser เพื่อใช้งานกับเบราเซอร์ ไลบรารี core เพื่อใช้งานโมดูล NgModule และไลบรารี app.component เป็นคอมโพเน้นท์หรือหน้าเว็บ

ในขณะสร้างเริ่มต้น ยังไม่มีงานบริการอะไร ทำให้ providers ยังเป็นอาร์เรย์ว่างอยู่ สำหรับการส่งต่อไปยังการใช้งานในโมดูลอื่นด้วยคำสำคัญ export โมดูลนี้ซึ่งเป็นโมดูลหลัก หากว่าโมดูลอื่น ต้องการใช้งานก็จะนำโมดูลนี้ไปใช้งานต่อได้ แต่ก็ไม่จำเป็นเพราะเราสร้างโมดูลนี้เพื่อการใช้งานในตัวเองเท่านั้น

สร้างคอมโพเน้นท์

ที่ผ่านมาเราจะเห็นคอมโพเน้นท์หลัก ดังจะนำอิลีเม้นท์ <app-root> ไปประกาศต่อที่ไฟล์ src/index.html ในตอนนี้เรามาทำลองสร้างอีกคอมโพเน้นท์หนึ่ง ในชื่อ myConponent ด้วยการพิมพ์ที่ CLI แต่ก่อนอื่นต้องออกจากทำงานของเซิร์บเวอร์ก่อน:

กด ctrl + c เพื่อออกจากโหมดทำงานของเซิร์บเวอร์ และปัจจุบันเรายังอยู่ในรูปแบบการสร้างแอปพลิเคชันแบบ standalone ในชื่อ first-app ดังนั้นเมื่อกด ctrl + c ก็ยังอยู่ในโฟลเดอร์ first-app ต่อไปก็ใช้คำสั่งสร้างคอมโพเน้นท์ได้

CLI 1.6

```
> ng generate component my
```

หรือเราอาจพิมพ์ย่อ ๆ ได้ใหม่ว่า:

CLI 1.7

```
> ng g c my
```

โดย g แทน generate และ c แทน component ต่อมาเข้าสู่การทำงานของเซิร์บเวอร์อีกครั้ง

CLI 1.8

```
> ng s --o
```

โดย s แทน serve และ o แทน open เมื่อหน้าเว็บทำงานจะได้หน้าเว็บเดิม ถ้าต้องการให้มีการแสดงผลของหน้าเว็บใหม่ของ my ที่ได้เพิ่งสร้าง ให้เพิ่มข้อมูลเส้นทางในไฟล์ app.routes.ts ด้วยตัวอย่างต่อไปนี้

Code 1.4. src/app/app.routes.ts (standalone)

```
import { Routes } from '@angular/router';
import { MyComponent } from './my/my.component';
export const routes: Routes = [
    {path: '', component:MyComponent},
];
```

เมื่อบันทึกไฟล์นี้จะได้ผลเป็นข้อความที่อยู่ในหน้า src/app/my/my.compopnent.html ซึ่งก็เป็นข้อความ "my works!" แสดงผล ในส่วนที่เป็น <router-outlet /> ที่อยู่ท้ายสุดของไฟล์ app.component.html (หรือจะลบข้อมูลทั้งหน้าของไฟล์นี้แล้วให้เหลือเฉพาะ <router-outlet />)

เดคคอเรเตอร์

แนวการเขียนโปรแกรมที่ต้องการเสริมคุณสมบัติของสิ่งที่ต้องการขยาย โดยการประกาศแจ้ง (annotating) ด้วยเครื่องหมาย @ นำหน้า แยกได้ได้หลายประเภท ซึ่งทั้งหมดถือเป็นเขียนโปรแกรมแบบเมตา-ดาต้า (Metadata-programming) หรือการให้ความหมายใหม่ตามประกาศ มีประเภทหลัก ๆ คือ

- ประกาศบนคลาส (class decorators)
- ประกาศบนฟังก์ชัน/เมธอด (method decorators)
- ประกาศภายในคลาส (property decorators)
- ประกาศเป็นตัวแปรของฟังก์ชัน (parameter decorator)

ในเดคคอเรเตอร์ประเภทต่าง ๆ มีตัวหนึ่งที่พบบ่อยคือ @Component({ }) เป็นการประกาศก่อนประกาศคลาส ภายใน @Component นี้ก็จะมีจะให้ความหมายเสริม ในรูปออบเจ็กต์ อันประกอบด้วยสมาชิก selector, templateUrl, และ styleUrls ตามข้อกำหนดมีได้ใน @Component นี้

ต่อไปจะยกตัวอย่างการสร้าง เดคคอเรเตอร์ Directive หรือเป็นการสร้างส่วนเสริมคำสั่งที่ใช้ กับ HTML สมมุติต้องการให้ชื่อว่า item จะใช้คำสั่งใน CLI จะทำให้เกิดไฟล์ใหม่คือ item.directive.ts และเพิ่มส่วน item นี้ลงในไฟล์ app.module.ts ต่อไปนี้ (ตอนนี้อยู่ในไดเรกทรอรี่ first-app) : ใช้ ctrl + c เพื่อหยุดการทำงานเว็บเซิร์บเวอร์ และถ้าต้องการจะเปิดการทำงานของเว็บเซิร์บเวอร์อีกครั้งใช้ ng serve –open

CLI 1.9

```
ctrl + c
> ng generate directive item
> ng serve --open
```

หรือ พิมพ์แบบย่อ ๆ

CLI 1.10

```
ctrl + c
> ng g d item
> ng s --o
```

หรือจะสร้างไฟล์ขึ้นมาเองก็ได้ ในชื่อ item.directive.ts ถ้าใช้แบบ standalone จะเขียนลงไปไฟล์เดียวกันกับคอมโพเน้นท์หลักก็ได้ แต่ต้องเขียนก่อน (เขียนบรรทัดบน ๆ ก่อนเขียนคอมโพเน้นท์) มีการเรียกใช้คำสั่งไดเร็กทีฟ

@Directive

คำสั่งที่ฝังใน HTML ที่สร้างขึ้นเอง เรียกกันว่า ไดเร็กทีฟ (directive) เช่น ต้องการสร้าง คำสั่งฝังที่จะใช้เป็นหนึ่งของ อิลีเม้นท์ HTML ว่า appItem จะต้องสร้างไฟล์ คำสั่งฝังนี้ก่อน และดัดแปลงเพิ่มการนำไปใช้กับ CSS ดังตัวอย่างต่อไปนี้

Code 1.5. src/app/item.directive.ts (standalone)

```
import { Directive , ElementRef } from '@angular/core';
@Directive({
  selector: '[appItem]',
  standalone: true
})
export class ItemDirective {
  constructor(el:ElementRef) {
      el.nativeElement.style.backgroundColor = '#ddd';
  }
}
```

ในตัวอย่างนี้ใช้งานในแบบ standalone ซึ่งอาจจะเขียนในไฟล์ app.component.ts รวมในไฟล์เดียวกันเลยก็ได้ หรือจะทำแบบแยกไฟล์ต่างหาก ตามตัวอย่างนี้ก็ได้ แต่ถ้าต้องการทำงานแบบ NgModule ก็เพียงตัดคำว่า standalone = true ทิ้งไป

การเพิ่ม ElementRef นี้ถือเป็นการใช้งานกับ อิลีเม้นท์ของ HTML และนำไปประยุกต์กับ CSS ได้ ในที่นี้กำหนดให้สีพื้นหลังเป็นสีเทาอ่อน (#ddd)

เมื่อสร้างไฟล์สำหรับคำสั่งฝังแล้ว จะต้องประกาศในโมดูล ให้รู้ที่อยู่ว่าอยู่ที่ไฟล์ใด ซึ่งใช้คำสั่ง import ในชื่อคลาส และชื่อไฟล์

นอกจากนี้ยังต้องประกาศในส่วน declarations ด้วยว่าจะใช้คำสั่งฝังนี้ในโมดูล ซึ่งเพิ่มเติมจากการประกาศคอมโพเน้นท์ ให้อยู่ในรูปอาร์เรย์ ดังตัวอย่างต่อไปนี้

Code 1.6. src/app.module.ts (บางส่วน **กรณีใช้งานแบบ NgModule**)

```
import { ItemDirective } from './item.directive';
declarations: [
  AppComponent,
  ItemDirective
],
```

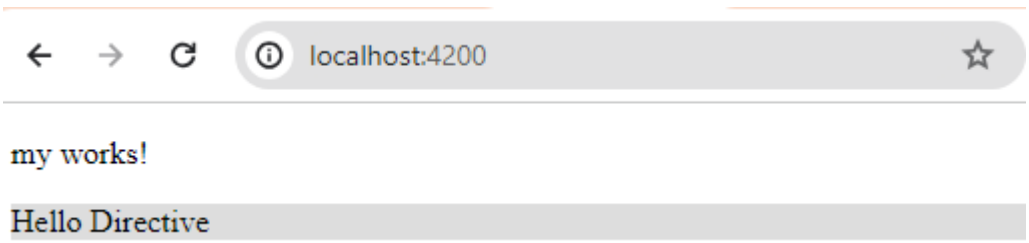
Code 1.7. src/app.component.ts (บางส่วน **กรณีใช้งานแบบ standalone**)

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { ItemDirective } from './item.directive';
@Component({
      selector: 'app-root',
      standalone: true,
      imports: [RouterOutlet, ItemDirective],
      templateUrl: './app.component.html',
      styleUrl: './app.component.css',
})
export class AppComponent {
  title = 'first-app';
}
```

ต่อมาก็นำ Directive นี้ไปใช้งาน ด้วยการใส่ appItem ไว้ใน <p> ให้เพิ่มอิลีเม้นท์ <p> ต่อไปนี้ไว้บรรทัดสุดท้ายของไฟล์ app.component.html

Code 1.8. src/app.component.html

```
<router-outlet />
<p appItem>Hello Directive</p>
```



รูป 1.5 ผลการใช้คำสั่งไดเร็กทีฟ

การนำไปใช้งานจริง

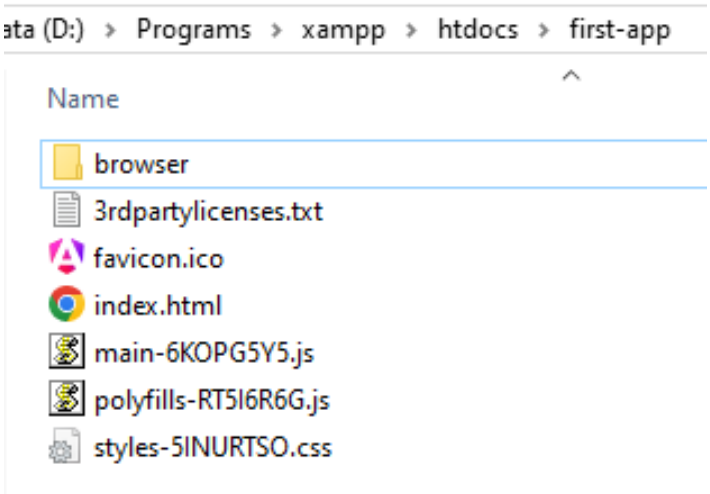
ณ ขณะสร้างนี้ เราสร้างขึ้นแล้วหยุดที่การสร้างคำสั่ง ไดเร็กทีฟ ต่อไปเมื่อนำไปใช้งานจริง (เว็บหน้าเดียวนี้) จะต้องทำการสร้างเป็นไฟล์ ลดขนาด ให้เล็กลง ขั้นตอนนี้เรียกว่า การผลิตงาน (Production) ให้เข้าไปยังตำแหน่ง โฟลเดอร์งานปัจจุบัน เช่น ณ ตอนแรกที่สร้างจะได้โฟลเดอร์ first-app ให้พิมพ์คำสั่งต่อไปนี้ผ่าน CLI

CLI 1.11

```
> ng build
```

ขั้นตอนนี้แองกูล่าร์ จะสร้างไฟล์ใหม่อยู่ใน โฟลเดอร์ dist ซึ่งย่อมาจากคำว่า distribution แปลว่านำไปแจกจ่ายใช้งานได้

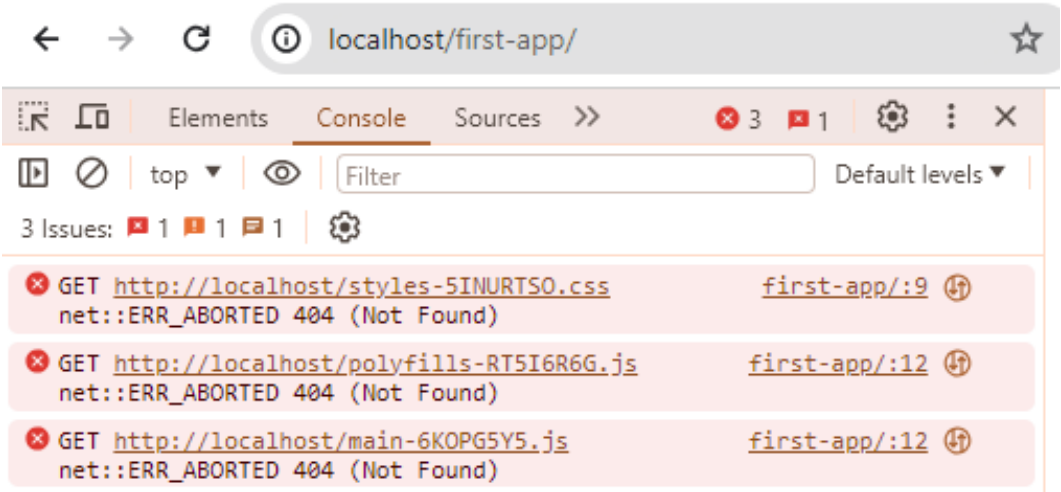
ไฟล์ต่าง ๆ ที่อยู่ ในโฟลเดอร์ dist ทั้งหมด เมื่อเปิดไฟล์ ดูรายละเอียดข้างในดู จะพบว่า ไฟล์ มีการลดขนาดลงไปมาก จะมีเพียงไฟล์ index.html เท่านั้นที่อ่านเข้าใจได้และมีเพียงไม่กี่บรรทัด ส่วนไฟล์อื่น ๆ ถูกย่อจนอ่านแทบไม่ได้เลย ไฟล์ต่าง ๆ เหล่านี้ สามารถนำไปใช้งานได้ โดยนำไปเก็บที่เว็บเซิร์บเวอร์ได้ โดยการที่ราก (root) ของเว็บ แล้วทดลองเปิดผ่านเบราเซอร์ ถ้าไม่มีอะไรผิดพลาดจะเปิดได้เหมือนกับที่เคยเปิดมาก่อนหน้านี้



รูป 1.6 ผลการผลิตงาน นำมาเก็บ XAMPP แทน

แต่อย่างไรก็ตาม ในการทดลองใช้งานนี้เรามักวางไฟล์เหล่านี้ ในตำแหน่ง ที่ไม่ได้อยู่ที่ราก หรือตำแหน่งแรกสุดของเว็บเซิร์บเวอร์ เช่น ถ้าวาง ใน XAMPP จะต้องวางในโฟลเดอร์ htdocs เช่นวางที่ htdocs/first-app โดยไฟล์ทั้งหมดของที่อยู่ใน dist/first-app/browser เราอาจย้ายไฟล์ทั้งหมดที่อยู่ในโฟลเดอร์ browser มาไว้ใน first-app แทน

ทดสอบโดยการเปิดเว็บเซิร์บเวอร์ของ XAMPP แล้วไปยัง URL: localhost/first-app จะพบว่าไม่พบหน้าตาม URL ที่ระบุ และถ้าใช้ Google Chrome เปิดหน้าต่าง Developer Tools จะพบ การหาไฟล์ที่ต้องใช้งานไม่เจอ นี้เป็นสาเหตุเว็บที่ทำงานไม่ได้ หรือเว็บที่ไม่ได้อยู่ที่ตำแหน่งราก ของเว็บเซิร์บเวอร์



รูป 1.7 หน้าต่าง More Tools / Developer Tools / Console

วิธีการแก้ไขให้กำหนดเส้นทางของไฟล์ใหม่ โดยการเปิด ไฟล์ index.html ซึ่งจะพบ ว่า <base href="/"> ในส่วนอิลีเม้นท์ <head> จะเขียนตำแหน่งไฟล์เริ่มต้นที่รากของเว็บเซิร์บเวอร์ ทำให้เกิดความผิดพลาดในการหาไฟล์ไม่พบ ให้แก้โดยการใส่เส้นทาง เช่น กรณีใช้ http://localhost/ first-app มีส่วนต่อท้ายเป็น first-app เติมต่อมาจากตำแหน่งราก จึงต้องแก้ให้ การอ้างอิงในส่วนเติมจากรากของเว็บเซิร์บเวอร์ด้วย จึงต้องแก้ไขในส่วน <base href="/">

Code 1.9. htdocs/first-app/index.html

```
<head>
  <meta charset="utf-8">
  <title>FirstApp</title>
  <base href="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
<link rel="stylesheet" href="styles-5INURTSO.css">
</head>
```

อีกวิธีหนึ่ง ซึ่งได้ผลเหมือนกันกับการเขียนเติมในไฟล์ index.html โดยการแก้ไข ขณะ ที่กระบวนการผลิตงาน จากเดิมใช้เพียงคำสั่ง –base-href แล้วตามด้วย เส้นทางเริ่มต้น เช่น ถ้าต้องการใช้เส้นทาง /sbc/courses/java/

CLI 1.12

```
> ng build --base-href /sbc/courses/java/
```

ยังมีอีกเรื่องที่เป็นปัญหา เมื่อลงโปรแกรมที่เซิร์บเวอร์แล้ว และทดลองเข้าสู่เส้นทางย่อย ตัวอย่าง เช่น URL: theerapone.com/sbc/java/activity โดยที่ URI: javaเป็นเส้นทางหลัก และ activity เป็นเส้นทางตามคอมโพเน้นท์ ActivityComponent ตามที่ระบุไว้ ไฟล์เส้นทาง การทำงานตามเส้นทางไม่ได้มีปัญหาอะไร แต่ปัญหาจะมีเมื่อผู้ใช้กด โหลดหน้าเว็บใหม่บนเบราเซอร์ จะพบว่า เว็บเซิร์บเวอร์แจ้งว่าไม่พบหน้าเว็บที่ต้องการ (ERROR 404)

ที่เป็นเช่นนี้เพราะ เส้นทาง URI: java/activity ไม่มีไฟล์อะไร แต่ถ้าใช้ URI: java จะมีไฟล์ index.html ซึ่งมีอยู่จริงจึงอ่านไฟล์นี้ได้ ปัญหานี้ต้องแก้ที่เว็บเซิร์บเวอร์ ซึ่งแต่ละชนิดเว็บเซิร์บเวอร์ก็แก้ไม่เหมือนกัน

กรณีของ Apache: ให้แก้ไขไฟล์ .htaccess ซึ่งวางไฟล์ที่ตำแหน่งราก หรือตำแหน่งเดียวกับ index.html

Code 1.10. .htaccess

```
RewriteEngine On

# If an existing asset or directory is requested go to it as it is
RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -f [OR]
RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -d
RewriteRule ^ - [L]

# If the requested resource doesn't exist, use index.html
RewriteRule ^ /index.html
```

จากตัวอย่างนี้ใช้กับ URL ที่มีเส้นทางเป็นที่อยู่ไอพีหลัก เช่น theerapone.com แต่ถ้าเราไม่ได้ใช้ที่อยู่เป็นไอพีหลัก เช่น theerapone.com/sbc/java โดยมี URI: java เป็นเส้นทางหลักของแองกูล่าร์ ให้เปลี่ยนเป็น (ลบเครื่องหมาย “/” ออก)

```
RewriteRule ^ index.html
```

> กรณี แก้ปัญหา แล้วยังไม่หาย แม้จะมีไฟล์ .htaccess แล้ว ให้ ทำการลบ ข้อมูลเก่าเก็บของเบราเซอร์ (delete browsing data) เช่น history, cache, cookies

สำหรับกรณีเว็บเซิร์บเวอร์ชนิดอื่น เช่น IIS Server หรือ Nginx ต้องศึกษาเพิ่มเติมว่าจะต้องกำหนดค่าอย่างไร

ถึงตอนนี้ หวังว่า รู้สึกว่าเข้าใจขึ้นมาบ้าง และเห็นประโยชน์อันมาก ของแองกูล่าร์ ความน่าใช้งานน่าเรียนรู้ นี้จะเป็นกระตุ้นให้ศึกษาการใช้งาน แองกูล่าร์ มากขึ้น แล้วพบกันใหม่ในบทต่อไป

สรุป

ในบทนี้ ต้องการทำให้เห็นว่าแองกูล่าร์ น่าใช้งานเพียงใด โดยให้สร้างเป็น แอปพลิเคชั่นอย่างง่าย โดยไม่ต้องเขียนอะไรเพิ่มเติมมากมายนัก ในการสร้างเว็บแอปฯ ต้องการให้ทำความรู้จักโครงสร้างโดยรวมของการใช้งานแบบโมดูล และแบบยืนเดี่ยว โดยมีคอมโพเน้นท์ ที่จะเป็นหน้าเว็บที่แสดงผล โดยแนวคิดใหม่ที่ให้คอมโพเน้นท์หนึ่ง เป็นหนึ่งหน้าเว็บ ที่ประกอบด้วยไฟล์ HTML และไฟล์ TS เป็นหลัก โดย TS ทำหน้าที่ควบคุมการแสดงผล และข้อมูลที่ต้องการใช้ใน HTML นอกจากนี้ ยังให้ทดลองในรูปแบบ Production เพื่อได้ไฟล์ใหม่ในโฟลเดอร์ dist เพื่อนำไปใช้งาน จากตัวอย่างจะพบแล้วว่า ไฟล์มีขนาดเล็กลงมาก ซึ่งเป็นเสน่ห์ อย่างหนึ่งที่น่าสนใจใช้งาน แองกูล่าร์ ขึ้นมาก เราจะได้ศึกษาความน่าสนใจอื่น ๆ อีกที่ทำให้แองกูล่าร์ เหมาะสมกับเว็บแอปฯ ขนาดใหญ่ ๆ และเป็นจุดสำคัญที่ต้องหันมาใช้แองกูล่าร์ แทนแองกูล่าร์เจเอส

คำถามทบทวน

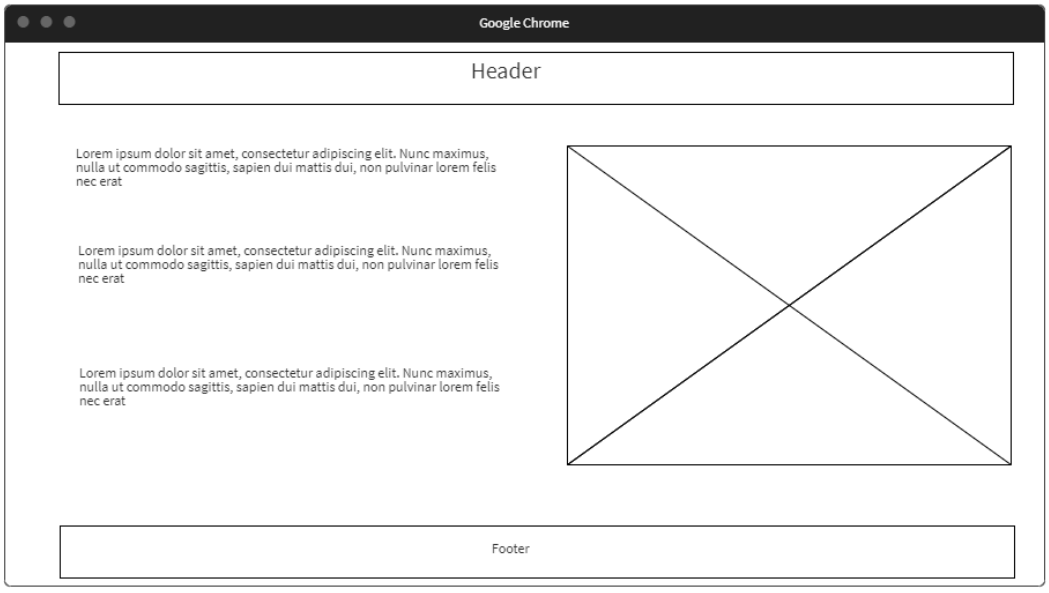
1. คำสั่งใน CLI ใดที่ใช้เพื่อสร้าง แอปพลิเคชันของของ แองกูล่าร์ และผลการสร้างอยู่ในรูปแบบ ยืนเดี่ยว (Standalone) หรือ แบบโมดูล (NgModule)
2. คำสั่งใน CLI ใดที่ใช้เพื่อสร้าง คอมโพเน้นท์ แบบเขียนย่อ ๆ
3. ค่าปริยาย ในการสร้าง แอปพลิเคชัน มีส่วนประกอบใดเพิ่มเข้ามาบ้าง มีเส้นทางมาด้วยหรือไม่ โหมดการใช้ภาษา TS ใช้แบบ strict หรือไม่
4. อิลีเม้นท์ <app-root></app-root> เป็นของคอมโพเน้นท์ใด และประกาศใช้งานที่ไฟล์ใด
5. ส่วนใดของโมดูลที่ระบุถึงคอมโพเน้นท์เริ่มต้นในการทำงาน
6. ทุกครั้งที่มีการสร้างคอมโพเน้นท์ขึ้นมาใหม่ จะต้องลงทะเบียนใช้งานที่ไฟล์ใด
7. คำสั่งใดใน CLI ใช้ในการผลิตแอปพลิเคชันเพื่อนำไปใช้งานจริง
8. อะไรคือข้อดี ข้อเสีย ของการใช้งานแองกูลาร์ ในรูปแบบ standalone เมื่อเทียบกับ รูปแบบ NgModule (ข้อนี้อาจหาเพิ่มเติมจากอินเทอร์เน็ต)

ในบทนี้ เราจะมาเริ่มใช้คอมโพเน้นท์ ซึ่งเป็นส่วนประกอบหลักที่ใช้แสดงผลหน้าเว็บ ในคอมโพเน้นท์หนึ่งจะประกอบด้วยไฟล์สำคัญสามชนิด คือ TS, HTML, และ CSS แต่ละไฟล์แยกการทำหน้าที่ กล่าวคือ ไฟล์ ประเภท TS ทำหน้าควบคุมการทำงาน และจัดการการแสดงผล ไฟล์ HTML, CSS ทำหน้าที่แสดงผลหรือ View หากมองในมุมมอง MVC (Model-View-Controller) ซึ่งเราจะทดลองใช้ทั้งสามไฟล์นี้ เพื่อใช้เป็นโลจิกและแสดงผล ดังนั้นในบทนี้เราจะทำความเข้าใจในเรื่องต่อไปนี้

- การสร้างเทมเพลท (Template)
- การสร้างโลจิกใน HTML: ngFor, ngIf, ngSwitch, @for, @if, @switch
- การสร้างไดเร็กทีฟ (Directive) และไปป์ (Pipe)
- การใช้งาน CSS, Bootstrap, JQuery
- การสื่อสารระหว่างคอมโพเน้นท์ (@input, @output)

สมมุติงานเว็บ

ก่อนที่จะเรียนรู้การใช้แองกูล่าร์ ต่อไป เรามาสมมุติงานที่จะสร้างเว็บกีก่อน โดยคิดว่าจะสร้างเว็บ หน้าแรกเป็น หน้าเว็บแนะนำข้อมูลทั่วไป เช่น ประกาศข่าว โดยให้มีหน้า Home ก็คือหน้าแรกนี้ เป็นรายการแสดงรายวิชา ที่มีให้เลือกเรียนกัน ซึ่งมีประมาณเริ่มต้นที่ 3 วิชา โครงสร้างหน้าเว็บตามรูปต่อไปนี้



รูป 2.1 หน้าแรก (สร้างจาก https://wireframepro.mockflow.com)

และข้อกำหนดเริ่มต้นในการสร้างเว็บนี้ ใช้แบบ **standalone** และไม่มีเรื่องเส้นทาง เราจึงตัดส่วนที่ไม่เกี่ยวข้องออกไปได้เลย

template / templateUrl

คอมโพเน้นท์เริ่มต้นใช้ ตามค่าปริยาย ที่ใช้สร้างก่อนหน้านี้ ที่ไม่ได้แก้ไขอะไร ให้เป็นไปตามที่ Angular สร้างมาให้ จากเดิม ไฟล์ app.component.ts จะมีลักษณะดังต่อไปนี้

Code 2.1. src/app/app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myAngular';
}
```

หากจำกันได้ ไฟล์นี้ ใช้สำหรับแสดงผล โดยมี แม่แบบ (templateUrl) จากไฟล์ app.component.html และกำหนดรูปแบบ จากไฟล์ app.component.css การนำไปแสดงผลในตำแหน่ง <app-root> ที่จะปรากฏในไฟล์ index.html

การแสดงผล นอกจากจะสร้าง จากไฟล์แม่แบบ ยังสามารถสร้างได้ในไฟล์ app.component.ts เอง ซึ่งการสร้างในไฟล์ตัวเอง จะใช้การกำหนดเป็น template แทน templateUrl เราลองมาสร้างในไฟล์ตัวเอง โดยดัดแปลงใหม่ ดังนี้

Code 2.2. src/app/app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  standalone: true,
  template: `
            <h1>Started Computer Learning at SBC</h1>
            `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myAngular';
}
```

ตอนนี้เราสร้างเพียง <h1></h1> ให้สังเกตว่า เราใช้เครื่องหมาย ` (back tick) **สัญลักษณ์นี้อยู่มุมบนสุดซ้ายสุดของคีย์บอร์ด โดยใช้แทน**เครื่องหมายคำพูด เครื่องหมายนี้ใช้เพื่อระบุว่า ข้อความภายในเครื่องหมายนี้เป็นข้อความ HTML และสัญลักษณ์นี้เป็นมาตรฐานของ ภาษา JavaScript รุ่น ECMAScript 2015 เป็นต้นไป

แต่ดูเหมือนว่า หากต้องเขียน HTML จำนวนมาก การใช้ template คงไม่สะดวก เพราะไม่สามารถแยกงานให้เป็นอิสระกว่าการใช้ templateUrl จึงควรกลับมาใช้แบบ templateUrl แบบเดิมน่าจะดีกว่า (ใช้ตาม **Code 2.1**)

ถ้าใช้แบบ NgModule เมื่อสร้างคอมโพเน้นท์ใดขึ้นมา จะมีการเพิ่มคอมโพเน้นท์ลงในไฟล์ app.module.ts ให้อัตโนมัติ การเพิ่มจะเพิ่มในส่วน import และ ในส่วน declarations

> ตามค่าปริยายแล้ว การสร้างคอมโพเน้นท์จะมีเทมเพลตไฟล์มาให้ แต่ถ้าไม่ต้องการให้สร้างเทมเพลตไฟล์ ใช้คำสั่ง:
> `ng generate component component_name --it`
> หรือ
> `ng generate component component_name --inline-template`

กำหนดค่าเริ่มต้นผ่านคอนสตรักเตอร์

ในการกำหนดค่าเริ่มต้น ที่ต้องการใช้กับคอมโพเน้นท์ สามารถกำหนดค่าได้ที่ตัวแปรของในคลาส (ตัวแปรของออบเจ็กต์) อย่างการกำหนดค่า title = ‘myAngular’ ได้ แต่ยังมีอีกวิธีหนึ่งที่กำหนดค่าเริ่มต้นได้ผ่านคอนสตรักเตอร์ (คลาสทุกคลาสสามารถกำหนดคอนสตรักเตอร์ได้) ซึ่งจะได้ค่าที่ใช้สำหรับแต่ละออบเจ็กต์ (คลาสต้องสร้างเป็นออบเจ็กต์ก่อนจึงจะนำใช้งานได้)

จากตัวอย่างงานเว็บสมมุติ ให้ มีรายวิชาที่จะแจ้งเปิดสอน ซึ่งเก็บข้อมูลข่าวในรูปอาร์เรย์ ดังนั้นในที่นี้เราจะสร้างรายการข่าว ที่มีค่าเริ่มต้นในคอนสตรักเตอร์ ของคลาส AppComponent

Code 2.3. src/app/app.component.ts (standalone)

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

@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'first-app';
  news = new Array();
  constructor(){
        this.news.push({title:'C# Programming',
                        content:'C# Programming will start on 12 Jan. 2024',
```

```
                    active:true, important:1, expire:'2024/4/30'});
        this.news.push({title:'JS Programming',
                    content:'JS Programming will start on 12 Feb. 2024',
                    active:true, important:2, expire:'2024/4/30'});
        this.news.push({title:'R Programming',
                    content:'R Programming will start on 12 Apr. 2024',
                    active:false, important:3, expire:'2024/5/30'});
  }
}
```

ในตัวอย่างนี้ ใช้อาร์เรย์แบบ ทูเพิล (tuple) สามารถเพิ่มออบเจ็กต์เป็น JSON หรืออินเทอร์เฟสด้วยฟังก์ชัน push ได้ ซึ่งได้เพิ่มเข้าไป สามข่าว ดังนั้นเมื่อคอมโพเน้นท์นี้ทำงาน จะมีสามข่าวนี้อยู่ในคอมโพเน้นท์นี้ด้วย

สร้างโลจิกใน HTML

ในภาษาเขียนโปรแกรม มีโลจิกที่สำคัญคือ if..else, for.. และ switch..case เราสามารถใส่โลจิกเหล่านี้ได้ในเอกสาร HTML ผ่านคำสั่ง ไดเร็กทีฟ (directives) ซึ่งจะใช้ เครื่องหมาย * นำหน้า หรือใช้ เครื่องหมาย [ ] และเมื่อใช้ในแบบ ยืนเดี่ยว จะต้องนำเข้า CommonModule ในไฟล์ app.component.ts:

Code 2.4. src/app/app.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { CommonModule } from '@angular/common';
@Component({
  selector: 'app-root',
  standalone: true,
  imports:[RouterOutlet, CommonModule],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css',
})
```

*ngIf แทนเงื่อนไข “ถ้า” ซึ่งถ้าเงื่อนไข เป็นจริง (true) ก็จะแสดงผล ในส่วน ที่มี *ngIf ประกาศในนี้ และถ้าเงื่อนไขเป็นเท็จ จะทำงานในส่วนของ else ก็จะแสดงผลใน <ng-template #emptyNews> กรณีที่ไม่ต้องการใช้ else ก็ละส่วนการเพิ่ม else ไปได้เลย และการเพิ่มเงื่อนไข OR และ AND จะแทนด้วยเครื่องหมาย || และ && ดังตัวอย่างการใช้ต่อไปนี้

```
<div *ngIf='Expression'></div>
<div *ngIf='Expression1 || Expression2'></div>
<div *ngIf='Expression1 && Expression2'></div>
```

หรือจะสร้างเป็น ng-template ทั้งในส่วน ngIf และ else จะมีรูปแบบการใช้คือ:

```
<ng-container *ngIf='Expression; then templateYes; else templateNo'>
</ng-container >
<ng-template #templateYes>Yes</ng-template>
<ng-template #templateNo>No</ng-template>
```

ชื่อของเทมเพลต ใช้เกณฑ์การตั้งชื่อตามลักษณะตัวแปรทั่วไป ถ้าตั้งเป็น template-Yes จะใช้ไม่ได้ *ngFor ใช้ในการวนซ้ำ ซึ่งใช้คู่กับ of เช่น ตามรูปแบบต่อไปนี้มี item เป็นตัววิ่งในการวนซ้ำ และมี collection เป็นชุดข้อมูล การวนซ้ำจะทำงานภายใต้ ที่ *ngFor อาศัยอยู่

```
<div *ngFor="let item of collection">item</div>
```

ตัวแปรที่ใส่เพิ่มได้ใน *ngFor index: number: แทนลำดับในอาร์เรย์ เริ่มจากศูนย์

1. count: number: แทนจำนวนที่มีในอาร์เรย์
2. first: boolean: แทนค่าจริง เมื่อเป็นลำดับแรก
3. last: boolean: แทนค่าจริงเมื่อเป็นลำดับสุดท้าย
4. even: boolean: แทนค่าจริงเมื่อเป็นลำดับคู่
5. odd: boolean: แทนค่าจริงเมื่อเป็นลำดับคี่

เมื่อต้องการแทน เลขลำดับในอาร์เรย์ และค่าแรก จะเขียนได้คือ

```
<div *ngFor="let item of collection; index as i; first as isFirst">
      {{i}} {{item}} {{isFirst}}
</div>
```

ยังมีอีกโลจิก คือ [ngSwitch] ซึ่งเป็นการเลือกทำตามตัวเลือก มีรูปแบบการใช้งานดังนี้

```
<div [ngSwitch]='Expression'></div>
<div *ngSwitchCase="'case1'"></div>
<div *ngSwitchCase="'case2'"></div>
<div *ngSwitchCase="'case3'"></div>
<div *ngSwitchDefault></div>
```

Angular 17: Build-in Control flow ได้เพิ่มโลจิกแบบใหม่ จะใช้แบบใหม่ หรือแบบเดิมก็ได้

```
@if(logic){ }, @if(logic){} @else{}, @if(logic){}@else if{} @else{}
@for(logic; track $index){},
@switch(){
 @case (a){} @case (b){} @default{}
}
```

ต่อไปจะนำชุดข้อมูลข่าวแสดงที่หน้าเว็บ ซึ่งอยู่ที่ไฟล์ app.component.html ตามที่ระบุใน templateUrl และตัวอย่างการใช้โลจิกต่าง ๆ ทั้งใช้การวนซ้ำในอาร์เรย์ การตั้งเงื่อนไข ถ้า..แล้ว และ เลือกทำตามตัวเลือก ดังการปรับปรุงไฟล์ในเทมเพลตคือ

Code 2.5. src/app/app.component.html

```
<div >
 <header><h1>Started Learning Information Technology</h1></header>
 <div>
      <div>
            <div *ngFor="let item of news">  {{item.content}}
              <span [ngSwitch]='item.important'>
                    <span *ngSwitchCase=1>สำคัญมาก</span>
                    <span *ngSwitchCase=2>สำคัญ</span>
                    <span *ngSwitchDefault></span>
              </span>
            </div>
            <div *ngIf='news.length>0 else emptyNews'>
                  มีข่าวแจ้ง({{news.length}})
            </div>
            <ng-template #emptyNews>
                  ขณะนี้ไม่มีข่าว
            </ng-template>
      </div>
      <div>
        <img src="./assets/images/sbc_building.jpg">
     </div>
 </div>
  <footer>   <p>Information Technology Department</p> </footer>
</div>
```

หมายเหตุ: โฟลเดอร์ assets อยู่ใน โฟลเดอร์ public

จากตัวอย่างนี้ ใช้ *ngFor เป็นตัววนซ้ำในอาร์เรย์ โดยมีตัววิ่งในอาร์เรย์ซึ่งแทนข้อมูลข่าวแต่ละรายการในชื่อ item และทำการผูกข้อมูลด้วยเครื่องหมายปีกกาคู่ การใช้ [ngSwitch] จับค่า important ซึ่งเป็นค่าตัวเลข 1,2,3 โดยให้ค่า 3 เป็นตัวเลือกปริยาย (*ngSwtchDefault) ให้สังเกตว่า เฉพาะตัว [ngSwitch] ใช้เครื่องหมาย [ ] ส่วนตัวอื่น ๆ ใช้ เครื่องหมาย * สำหรับ *ngIf .. else ใช้เทมเพลต สำหรับ else

ถึงตอนนี้เมื่อทำการบันทึกข้อมูล หน้าเว็บจะเปลี่ยนแปลง โดยแสดงของมูลเป็นรายการทั้งสามข่าว (ดูที่เบราเซอร์) มีหัวข้อ (header) มีข่าว กับรูปภาพ (content) และมีข้อมูลท้ายเว็บ (footer) ตามอีลีเม้นท์ สำหรับรูปภาพต้องนำไฟล์รูปไปใส่ตามเส้นทางโฟลเดอร์ที่ระบบใน src/assets/images

ตาราง 2. 1. เปรียบเทียบการใช้โลจิก if, for, switch ของแองกูล่าร์ 17 และต่ำกว่า 17

Angular รุ่นต่ำกว่า 17	Angular รุ่น 17

`<div *ngIf='news.length>0 else emptyNews'>`
มีข่าวแจ้ง({{news.length}})
	`@if (news.length>0){`
	`<div>`
	มีข่าวแจ้ง ({{ news.length }})