Angular 4

```
import { Component } from '@angular/core';
@Component({
    selector: 'intro',
    template: `
        <h1>{{name}}</h1>
        <h2>{{title}}</h2>
        <h3>{{work()}}</h3>
    `
})
export class IntroComponent {
    name: string = "박용준";
    title: string = "Microsoft MVP";
    work() { return "http://www.devlec.com"; }
}
```

Microsoft Most Valuable Professional

데브렉(http://www.devlec.com)

- 앵귤러는 자바스크립트 프레임워크입니다.

앵귤러는 **모바일**과 **데스크톱**을 위한

하나의 완성된 자바스크립트 프레임워크입니다.

현재 가장 인기있는 자바스크립트 프레임워크입니다.

구글과 마이크로소프트

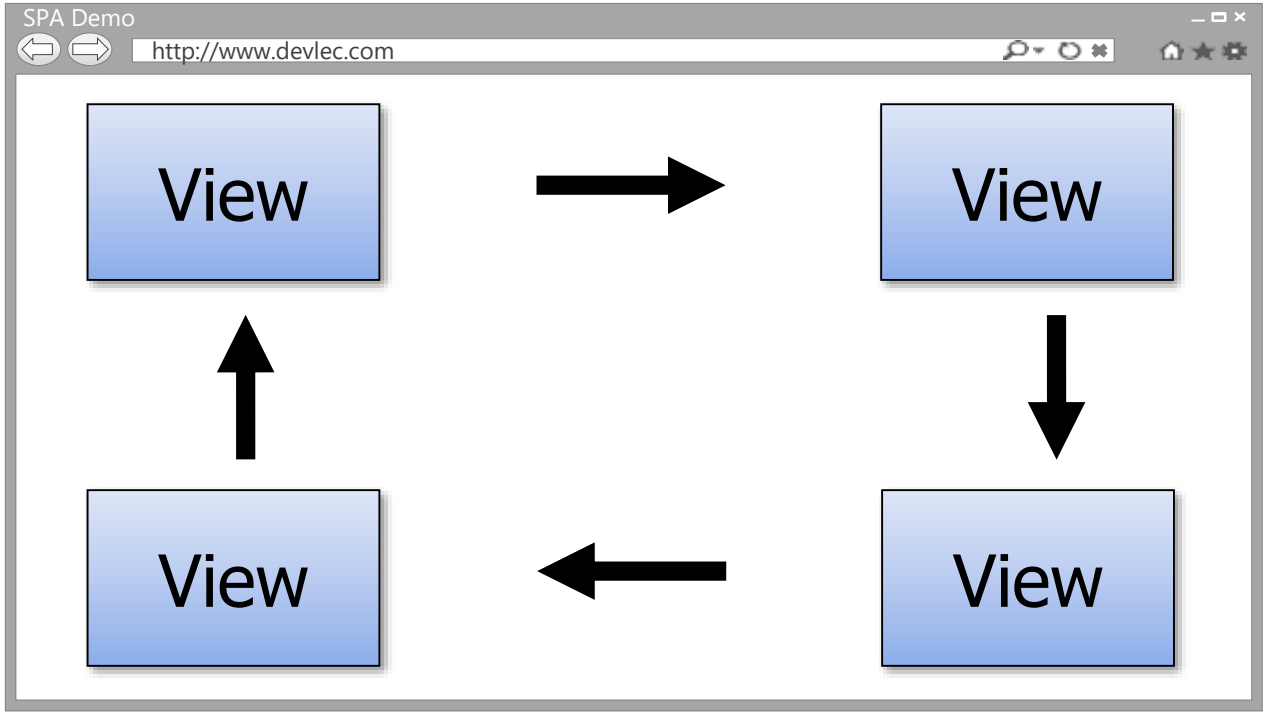
앵귤러는

서버측 프레임워크가 아닌 클라이언트측 프레임워크입니다.

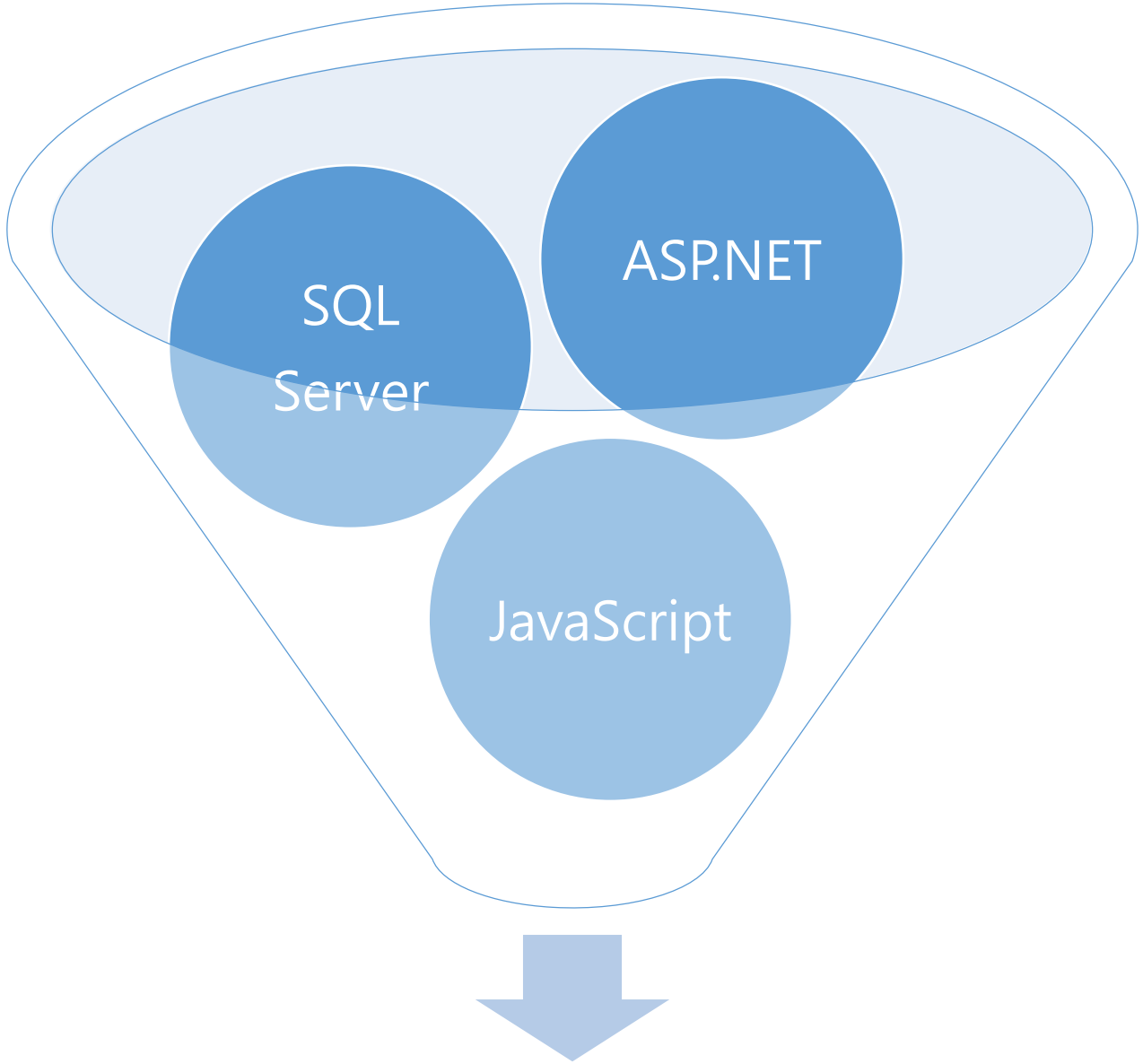
jQuery, React같은 자바스크립트 라이브러리가 아닙니다.

MVC와 같은 디자인 패턴이 아닙니다.

Single Page Application (SPA)



SPA 필요 없는 경우 Angular도 필요 없...



앵귤러4 + ASP.NET Core

선수 학습(데브렉 강의 기준)

- 필수
  - Web
    - HTML5
    - CSS3
    - JavaScript
    - Bootstrap
  - C#
  - SQL Server 기초
  - ASP.NET 기초 입문
    - ASP.NET 4.6 기초 입문
    - ASP.NET Core 1.0 기초 입문
  - TypeScript
- 선택
  - Angular 1.X

제 강의 기준으로
Angular 4는
가장 상위 과정입니다.

서버측, 클라이언트측 기술을
어느정도 알고 있다고 가정하고
진행합니다.

이 강의는 친절한 버전의 강의가 아닙니다. 이미 위와 같은 많은 기술을 알고 있다고 가정하고 진행합니다.

선수 학습 필수 과목(서버 측, 백 엔드)

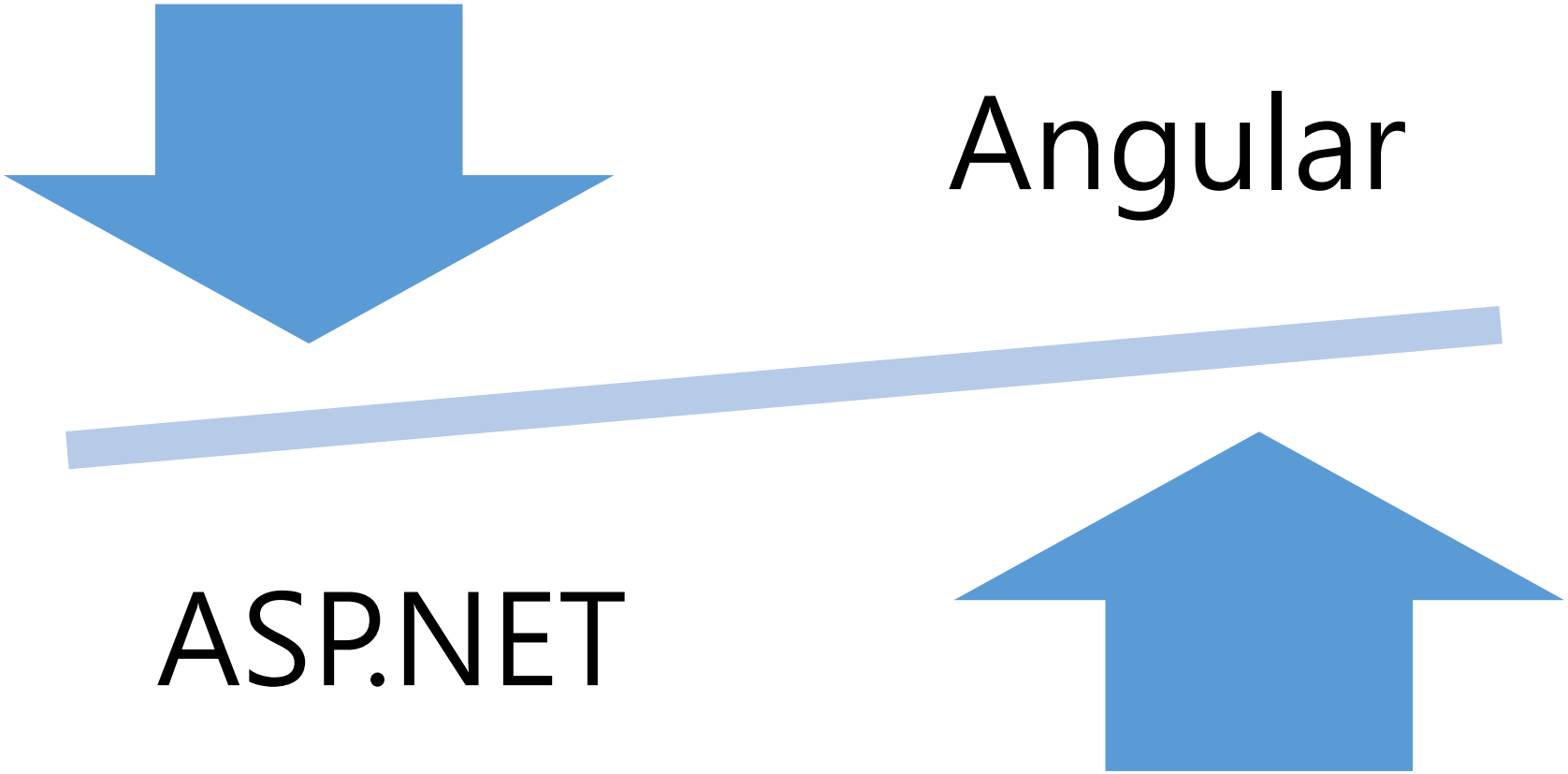
- ASP.NET Core 기본
  - 컨트롤러
  - 액션
  - 모델 바인딩
  - Web API(GET, POST, PUT/PATCH, DELETE)
- Dapper 기본
  - CRUD(Create, Read, Update, Delete)
- 서버 측 기능은 강의 후반부에 함께 다룹니다.

이 과정을 통해서 우리가 배울 내용

- 앵귤러4 기본(강의 대부분은 앵귤러 2 기반으로 구성됨)
  - 컴포넌트와 템플릿
  - 템플릿 문법(속성 바인딩과 이벤트 바인딩)
  - 서비스와 의존성 주입
  - 폼
  - 라우팅
  - HTTP
  - 모듈

- 배포에 대해서는 따로 다루지 않습니다.
  - Visual Studio 2017에서 Azure Web App에 업로드하는 내용만 다룹니다.

클라이언트측과 서버측



앵귤러4와 어울리는 서버 측 기술들

ASP.NET Core

SQL Server

Node.js

MongoDB

ASP.NET Core Web API

- 최고의 RESTful 서비스 구축
- JSON
  - XML

언어 선택

- Angular 4는 타입스크립트와 함께
  - 타입스크립트 설치
    - npm install –g TypeScript

- JavaScript
  - ES
  - ES3
  - **ES5**
    - **모든 브라우저 지원, 따로 컴파일 필요없음**
  - ES 2015
    - class, let, arrow

데모: 개발 환경 구축

- Node.js와 NPM 설치
- Visual Studio Code 설치
- Angular CLI 설치
  - Angular 프로젝트 생성
  - Angular 프로젝트 실행
  - Visual Studio Code로 Hello World 수정
  - 다시 실행

https://youtu.be/g76bb5U6YUs

강의 환경

- Visual Studio 2017 + ASP.NET Core SpaTemplates
  - Visual Studio 2015 Update 3
  - https://www.visualstudio.com
- Visual Studio Code
- Postman
  - https://www.getpostman.com
- 웹 브라우저
- Node.js & NPM
- Git
- Angular CLI

강의 소스

- AngularNote.zip
  - 데모용으로 화면에서 볼 수 있는 코드는 포함되지 않을 수 있습니다.

[참고] 표시 강좌

- [참고] 표시된 강좌는 반드시 볼 필요 없습니다.
  - 참고용으로 보시면 됩니다.

- [구강좌]
  - 최근 새롭게 방법이 변경된 부분 이전에 촬영된 강좌입니다.

강의 시작합니다.

Angular

- JavaScript 프레임워크
  - 크로스 플랫폼 서버측 렌더링
- 클라이언트 사이드
  - 서버 사이드: Node.js, ASP.NET
- HTML, CSS, JavaScript와 함께 사용
- TypeScript 지원
  - ES6(ES2015)

. 1995 - Introduction to JavaScript

. 2002 - XMLHttpRequest (XHR)

. 2006 - jQuery

. 2008 - Chrome browser and V8 Engine – The web gets faster

. 2009 - Nodejs

. 2010 - Knockout JS

. 2010 - Angular 1.0 by Google

. 2013 – React

. 2013 - TypeScript

. 2016 – Angular 2.0 and ES6

. 2017 – Angular 4.0

Australia 2017

Microsoft Ignite

Angular의 특징

- HTML과 로직의 분리
- 서버와 클라이언트의 분리
- 컴포넌트 기반 UI
  - 컴포넌트: UI + 데이터 + 로직
  - 타입스크립트 클래스
- 큰 규모의 프로젝트에 적합
  - 하나의 완성된 개발 형태를 가지고 진행
    - 형태가 정해짐
    - 클래스, 컴포넌트, 모듈, ...
  - 작은 규모면 jQuery로도 충분

Angular 장점

- 빠름(Fast)
- 최신(Modern)
  - IE9 이상 모두 지원
  - ES6 등의 최신 기술 사용
- 깔끔(Simple, Clean)
  - 유지보수하기 편함
- 생산성(Productivity) 높음
- 쉬움(Easy)
  - ?
- 신기술
  - Lazy Loading
  - 서버측 렌더링
  - 멀티 렌더링 타겟

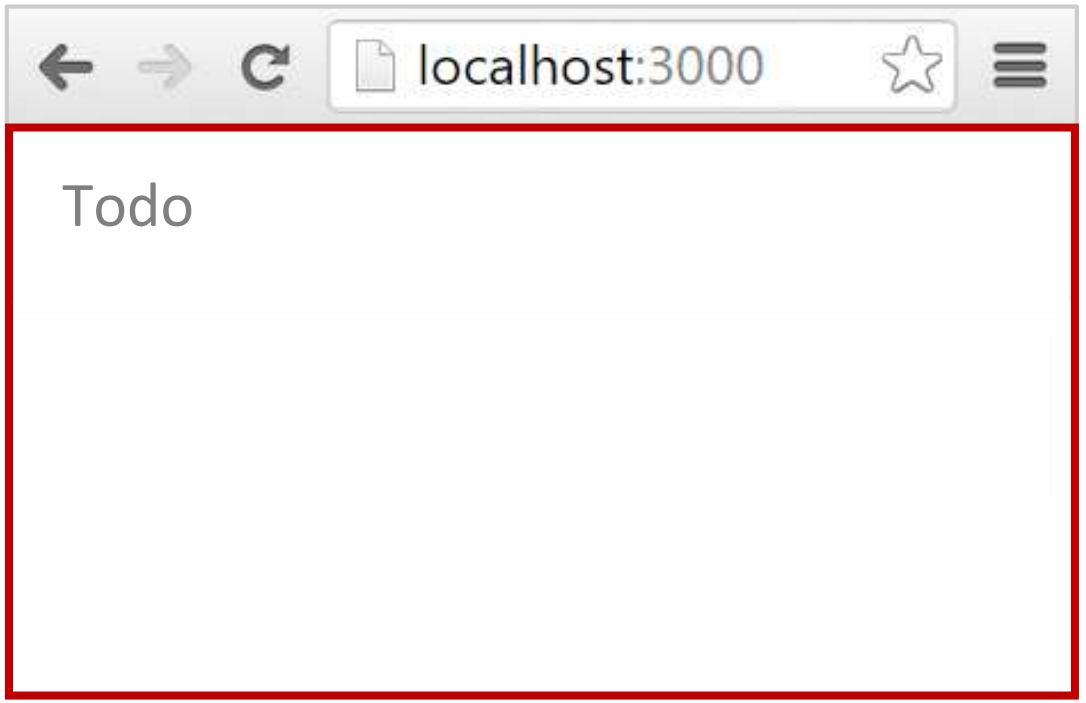
Angular 기초

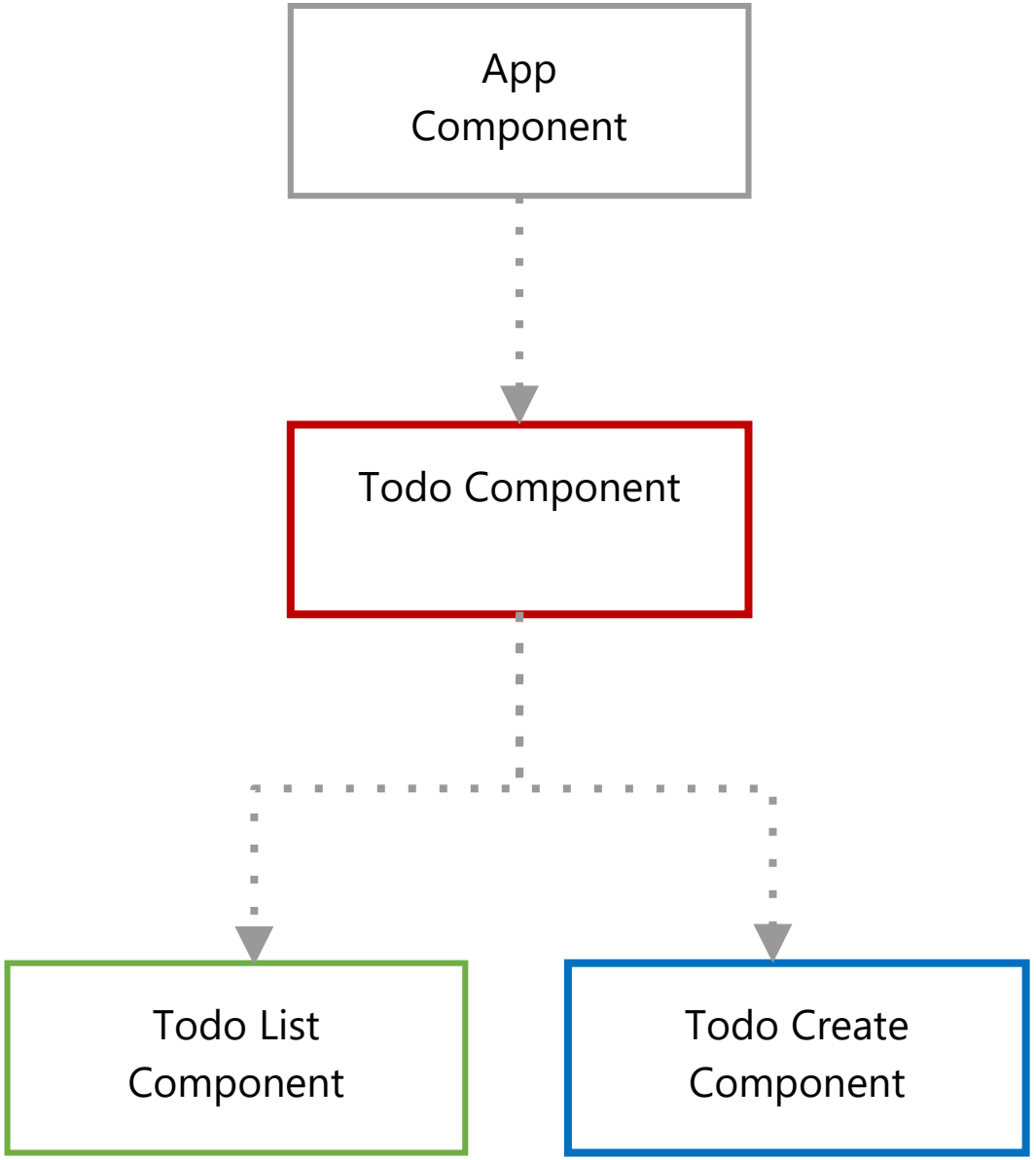
- Angular 앱 = 컴포넌트 + 컴포넌트 + ... + 서비스
- 컴포넌트(Components)
  - 뷰 컴포넌트 = 데이터 + UI(템플릿)
  - 타입스크립트 클래스
    - 속성과 메서드
  - 컨트롤러
  - 컴포넌트 지시자
    - 메타 데이터
- 서비스(Services)
- 데코레이터 지시자(Directives)
- 라우터(Routers)
- 파이프
  - 필터

Angular 핵심 용어

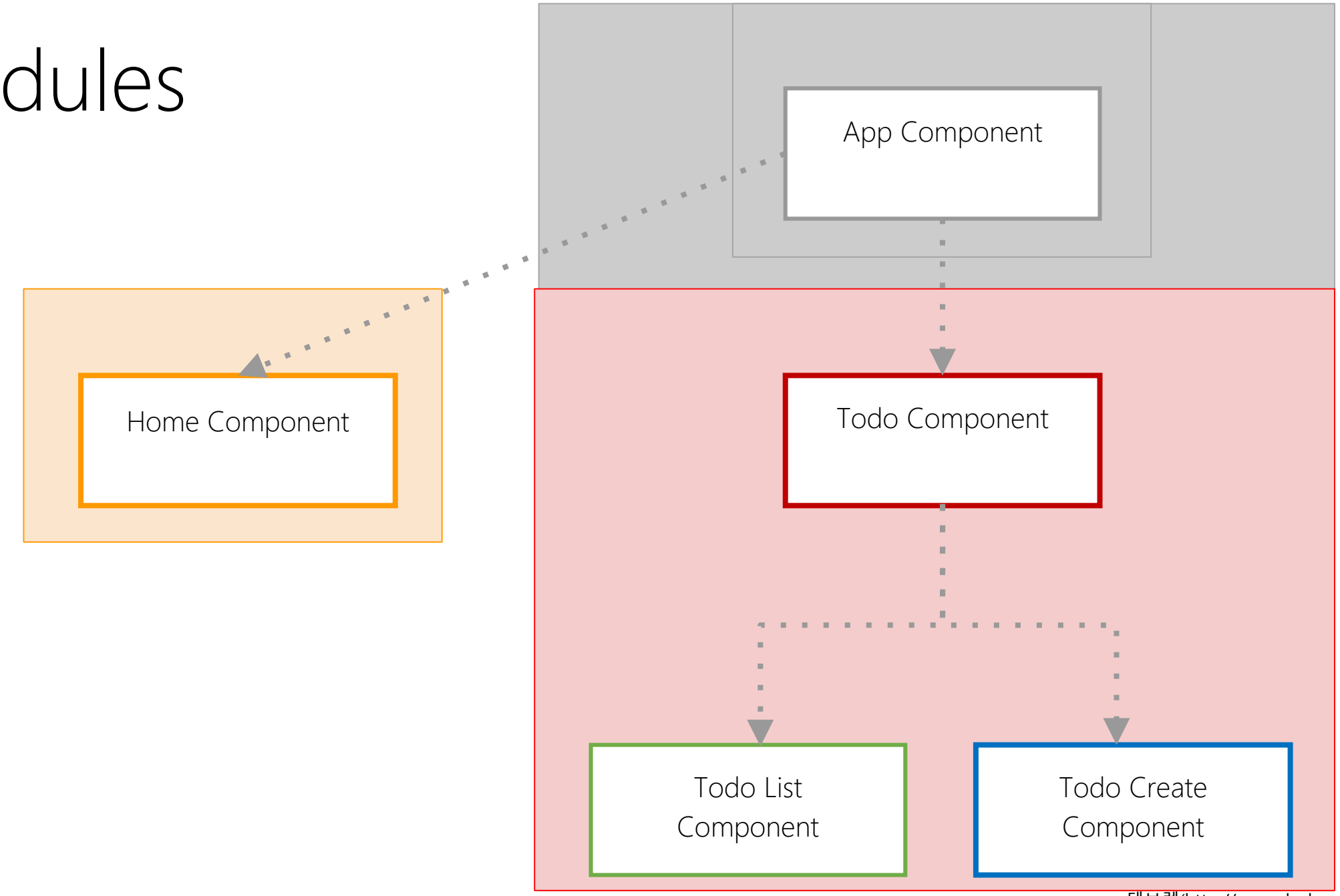
- 모듈
- 컴포넌트
- 템플릿
- 메타데이터
- 데이터 바인딩
- 지시자
- 서비스
- 의존성 주입

컴포넌트 트리


localhost:3000
Todo


App
Component
Todo Component
Todo List
Component
Todo Create
Component

NgModules


App Component
Home Component
Todo Component
Todo List Component
Todo Create Component

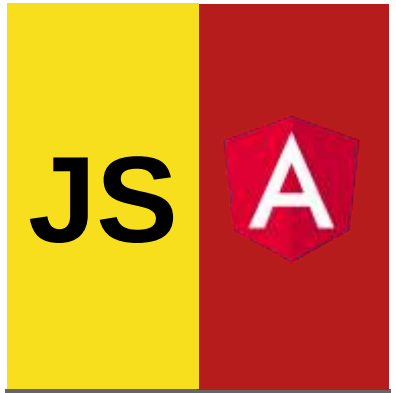
Angular 4 자체는 쉬우나

최근 웹 개발 관련된 주요 기술들을

선수학습으로 알아야 한다.

TypeScript

RxJS



Module Loader

TypeScript

..... or not

ES6

Dart

TypeScript

- 강력한 형식
- 강력한 도구와 리팩터링
- Angular 자체가 타입스크립트
- 모든게 타입스트립트
- 사용 안하는게 이상

… or not

- …

Microsoft Ignite

TypeScript 특징(Angular 4를 위한...)

특징들

클래스(Classes)

모듈(Modules)

데코레이터(Decorators)

TypeScript 특징(Angular 4를 위한...)

특징들

클래스(Classes) 

모듈(Modules)

데코레이터(Decorators)

클래스(Classes)

```
class C {
    constructor () { ... }
    property: string;
    method () { ... }
}
```

클래스 기반 프로그래밍

TypeScript 특징(Angular 4를 위한...)

특징들

클래스(Classes)

모듈(Modules) 

데코레이터(Decorators)

모듈(Modules)

```
import { add, divide } from "math";

import * as Math from "math";

export function bar() { ... }
```

표준 ES6 문법

TypeScript 특징(Angular 4를 위한...)

특징들

클래스(Classes)

모듈(Modules)

데코레이터(Decorators)

데코레이터(Decorators)

```
@Component({template:"<b>hi</b>"})
class C {
    ...
}
```

메타데이터

관심의 분리

앵귤러 4와 타입스크립트의 만남

특징들

최신 자바스크립트 사용

모듈화

미래에서 온 자바스크립트 특징 사용

최신의 클래스와 모듈 사용(ES6 표준)

모듈 기반으로 좀 더 개체 지향적 코드 작성 가능

현재 시점에서 미래 버전의 JS 특징들 사용

도구 선택

- 도구
  - Visual Studio Code
    - Visual Studio 2015
  - WebStorm, Atom, Eclipse, …
- 웹 툴
  - Node.js
  - NPM
  - Git
  - TSC

- 빠르고 가볍다.

- C#과 Angular를 위한 최고의 개발 도구

- 최고의 타입스크립트 제공
- 빠르고 가볍다.
- Angular 관련 확장 기능 제공

- Angular 관련 확장 기능 제공

Microsoft Ignite

Angular 4 앱 설정

- 직접 설정
  - angular.io 퀵 스타트

- Angular4 퀵스타터 다운로드
  - GitHub에서 다운로드
    - https://github.com/angular/quickstart/
- AngularCli(https://github.com/angular/angular-cli)
  - angular-cli
    - npm install –g angular-cli
    - ng new ProjectName
    - cd ProjectName
    - ng serve
    - ng generate component 컴포넌트이름
      - ng generate component About
    - ng g 컴포넌트이름

```
C:₩Temp₩code₩aa>ng generate component contact

installing component
  create src₩app₩contact₩contact.component.css
  create src₩app₩contact₩contact.component.html
  create src₩app₩contact₩contact.component.spec.ts
  create src₩app₩contact₩contact.component.ts
  update src₩app₩app.module.ts
```

Angular 퀵스타트

- https://github.com/angular/quickstart
  - git clone https://github.com/angular/quickstart MyFirstAngular
  - npm i
    - npm install
  - npm start

  - 기타 참고용 명령어
    - npm run tsc
    - npm run server
    - npm start

[참고] Angular 2 환경 설정

- 앱 폴더 생성
- 타입스크립트 설정 파일
  - tsconfig.json 파일 생성
- npm 패키지 파일
  - package.json 파일 생성
- 타입스크립트 정의 파일
  - typings.json 파일 생성
- 라이브러리 및 타이핑파일 설치
- Index.html HTML 페이지에 호스팅
- main.ts 파일 생성: 진입점(Entry Point)
  - C언어의 main()과 동일로 보면 됨
  - 부트스트랩퍼
- SystemJS 모듈 로더

tsconfig.json

- https://www.typescriptlang.org/docs/handbook/tsconfig-json.html

Visual Studio Code에서 *.js 숨기기

- settings.json

```
"files.exclude": {
    "**/.git": true,
    "**/.js": {
        "when": "$(basename).ts"
    },
    "**/*.js.map":{
        "when": "$(basename)"
    },
    "**/.DS_Store": true
}
```

[참고] Angular 2 기본 흐름 순서

- Index.html
  - System.import('app')
  - 디렉티브
- Systemjs.config.js
- main.ts : 엔트리 포인트
  - AppModule
- app.module.ts
  - @NgModule
- app.component.ts
  - @Component 데코레이터

Visual Studio 2017에서 앵귤러2 개발하기

박용준
Microsoft MVP

- Visual Studio 2017을 사용하여 앵귤러2 웹 응용 프로그램 개발하기
  - 클라이언트측: Angular2
  - 서버측: ASP.NET Core
  - 호스팅: Azure Web App

SPA 템플릿 소개

- ASP.NET Core와 Angular2를 하나의 프로젝트에서 관리
- 서버측 렌더링 지원
- Webpack dev 미들웨어 지원(ASP.NET Core Startup.cs)
- 핫모듈 교체 지원(HMR)
- 2가지 방식으로 빌드
  - Dev : 개발
  - Prod : 실제 프로덕션
- 웹 팩 사용
  - https://blogs.msdn.microsoft.com/webdev/2017/02/14/building-single-page-applications-on-asp-net-core-with-javascriptservices/

Visual Studio 2017 커뮤니티 설치

- https://youtu.be/5wTIUs8gnh0

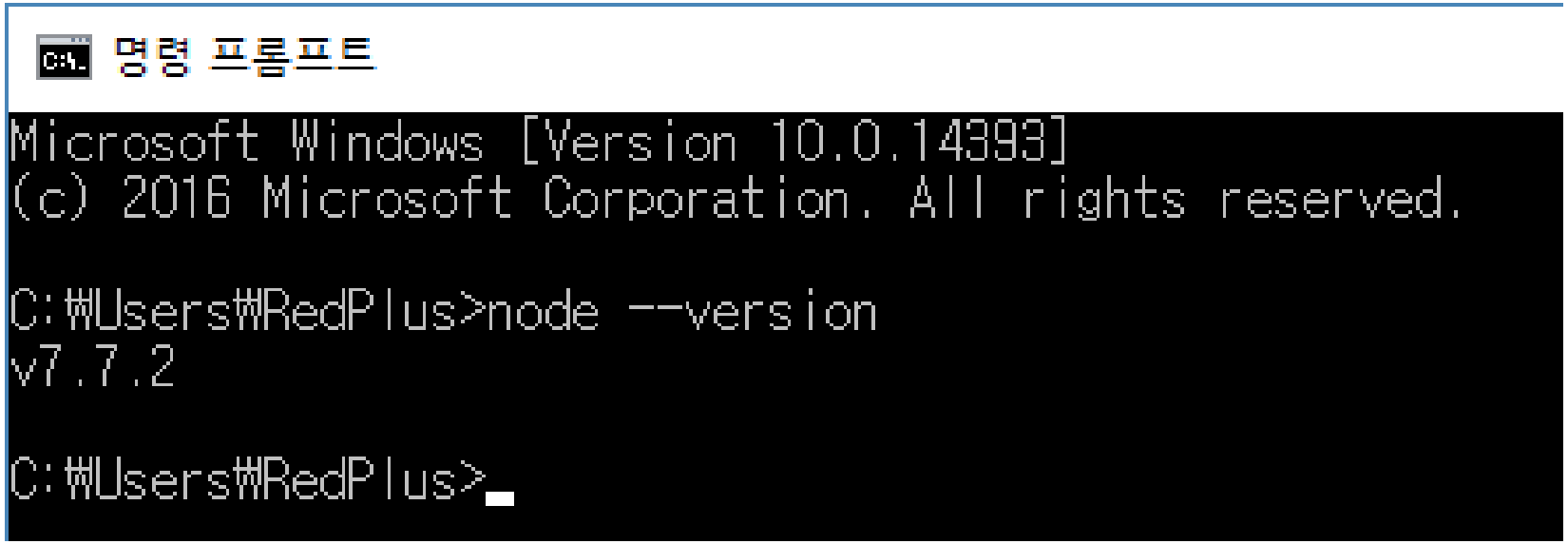
Web Essentials 2017 설치

- https://marketplace.visualstudio.com/items?itemName=MadsKristensen.WebExtensionPack2017

Angular 2 Snippet Pack 설치

- https://marketplace.visualstudio.com/items?itemName=MadsKristensen.Angular2SnippetPack

Node.js 최신 버전 설치: 6.X 버전 이상



Angular SPA 프로젝트 설치 및 생성

- dotnet new --install Microsoft.AspNetCore.SpaTemplates::*
  - 크로스 플랫폼 환경에서는 yoman 사용
- dotnet new angular

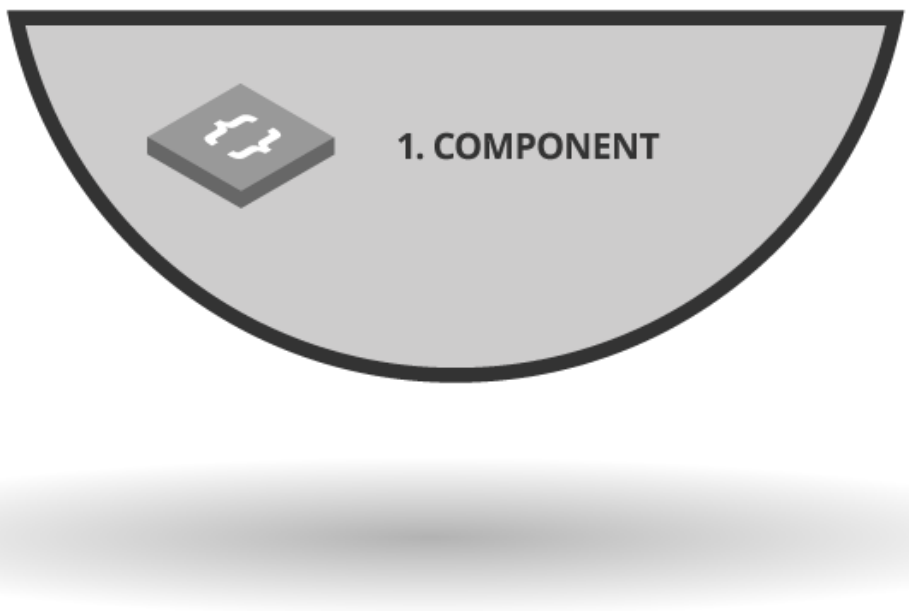
- dotnet restore

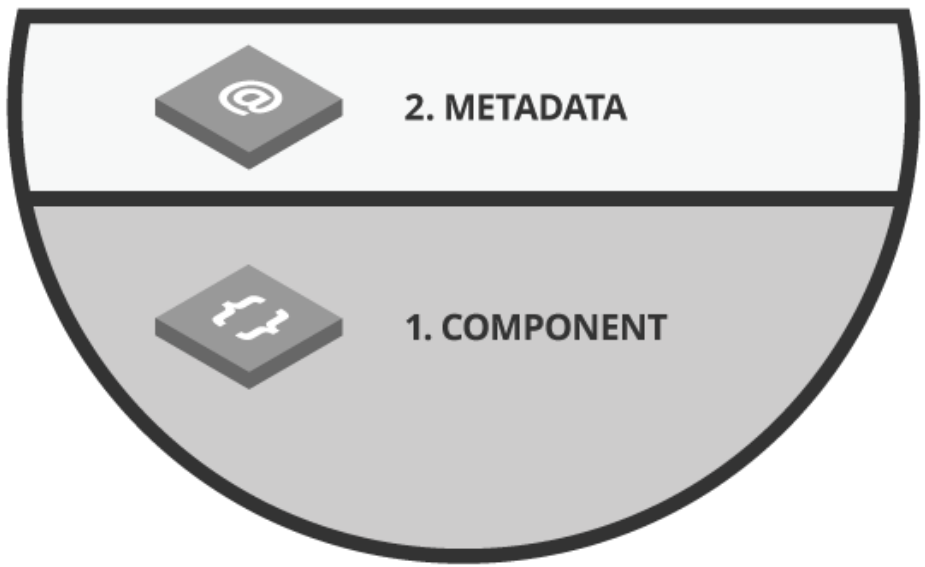
- npm install

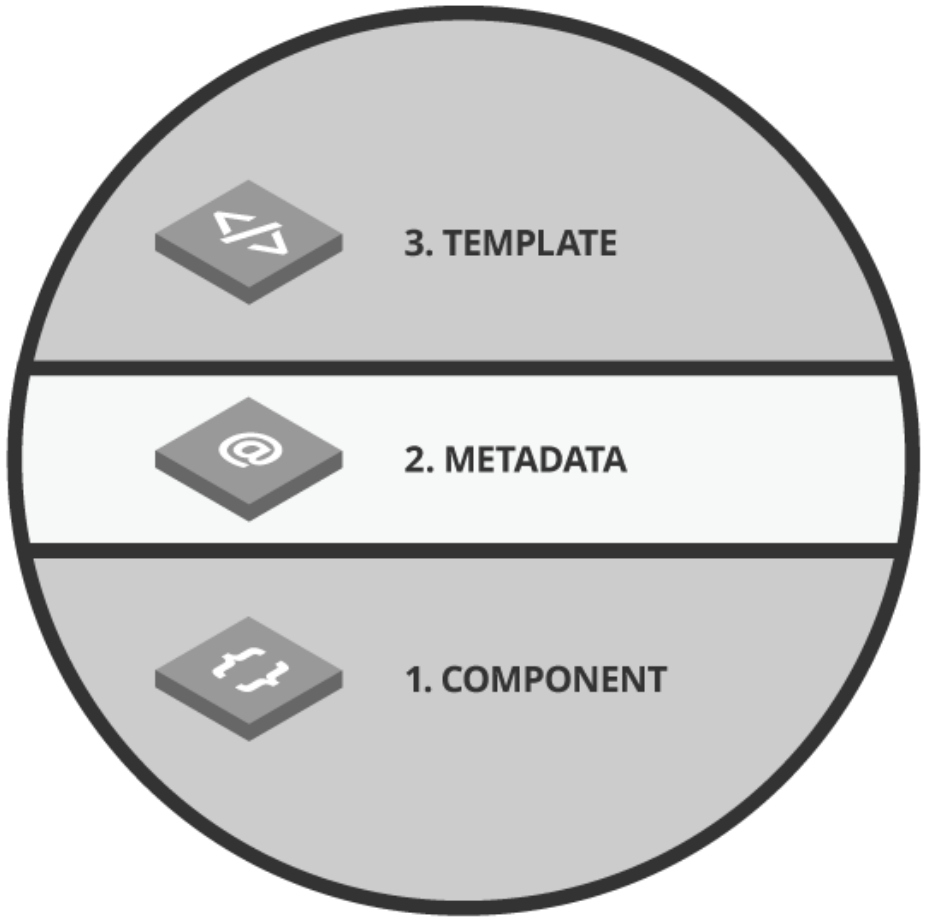
- start AngularAspNet.csproj
  - Visual Studio 다시 시작 권장

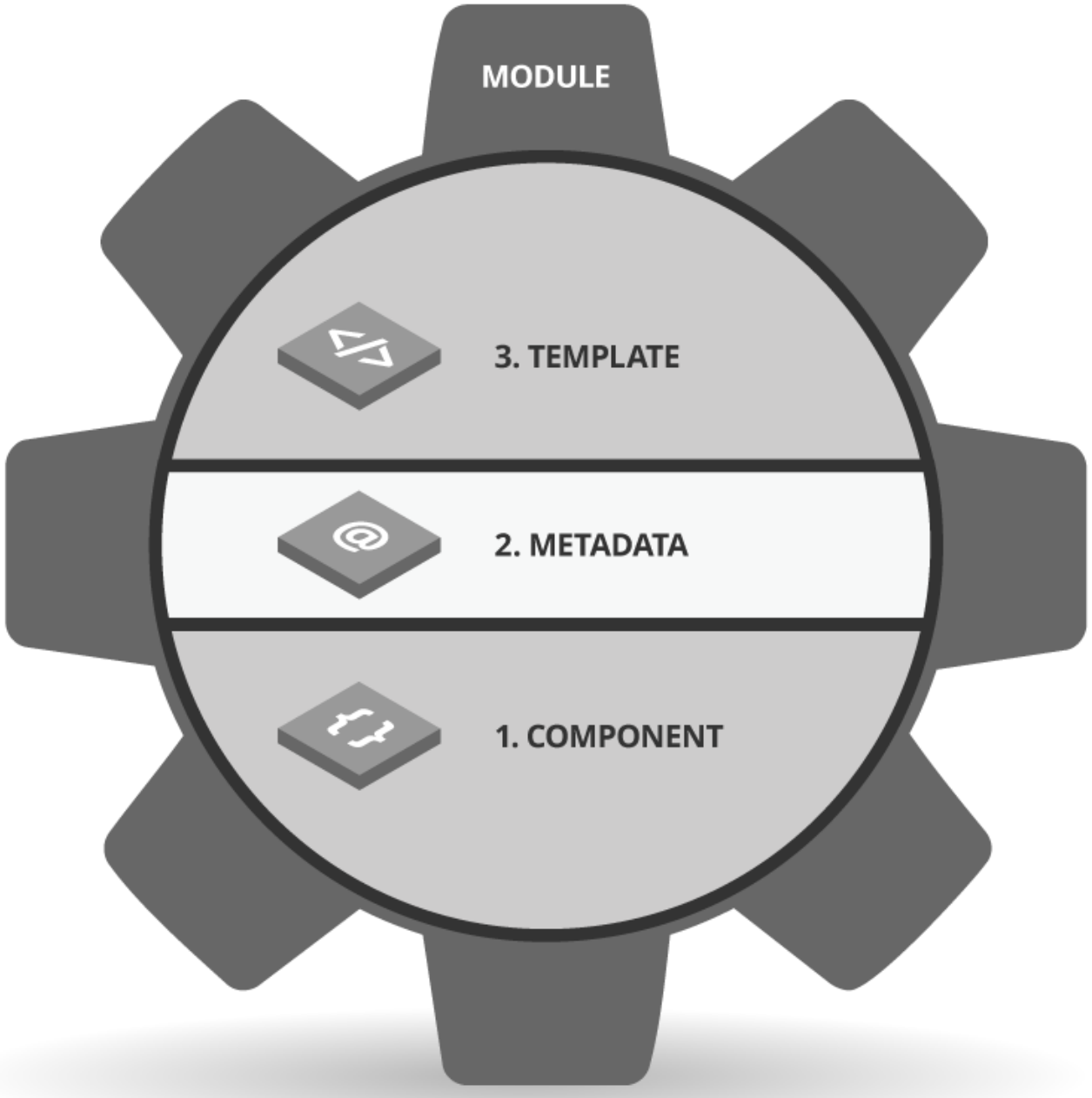
앵귤러에서 알아야 할 8가지 핵심 개념

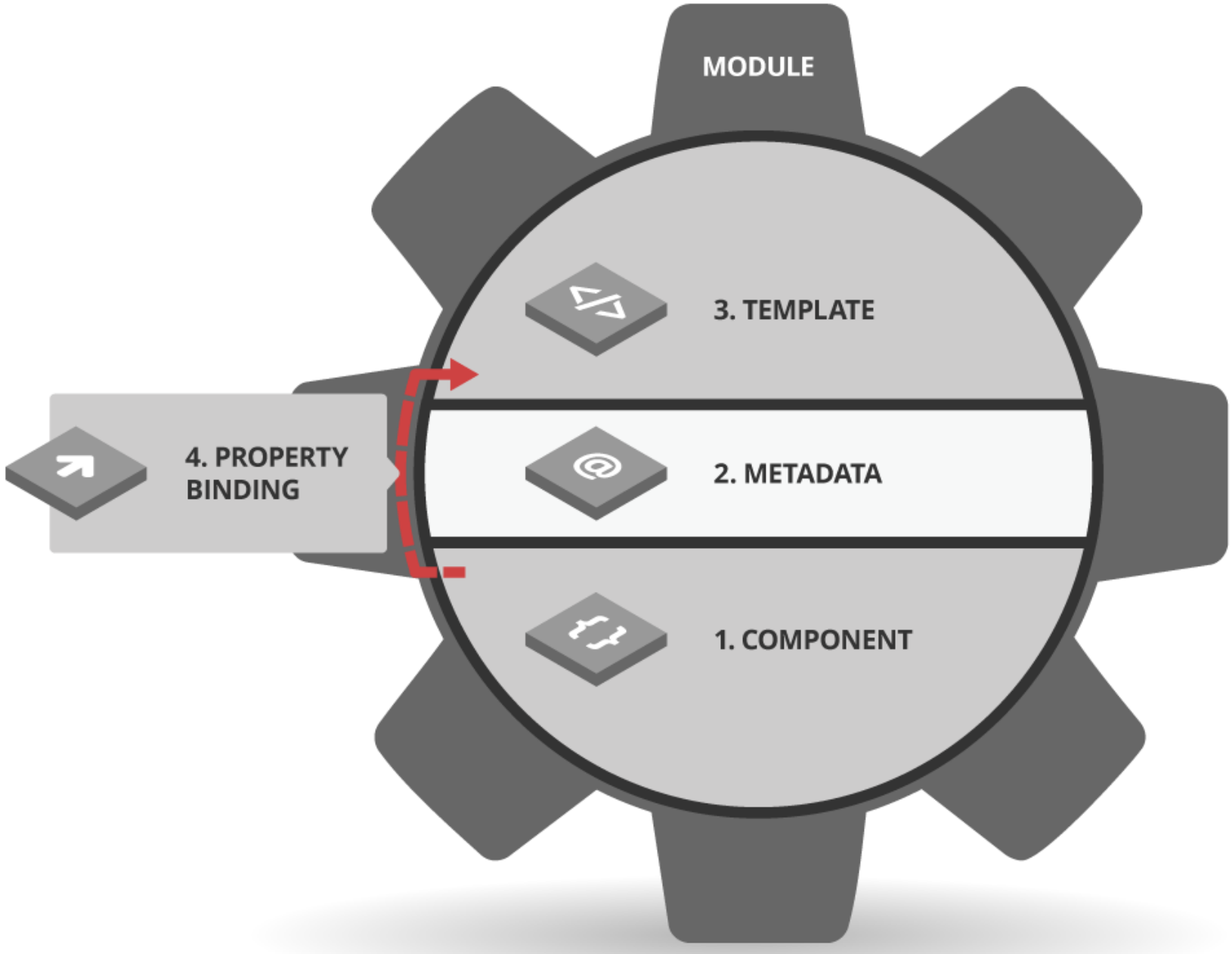
https://channel9.msdn.com/Events/Ignite/Australia-2017/OPEN341 참조

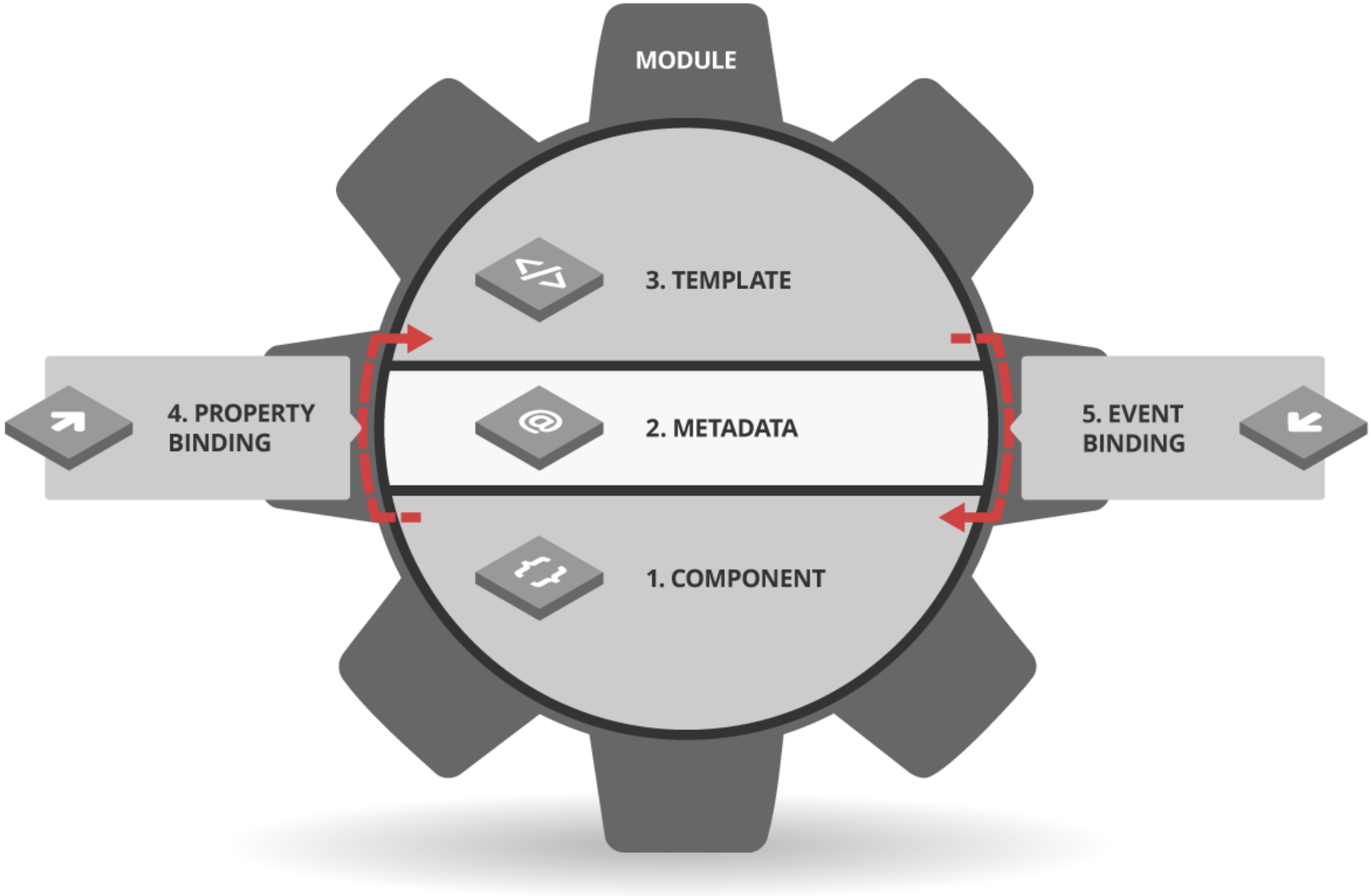

1. COMPONENT

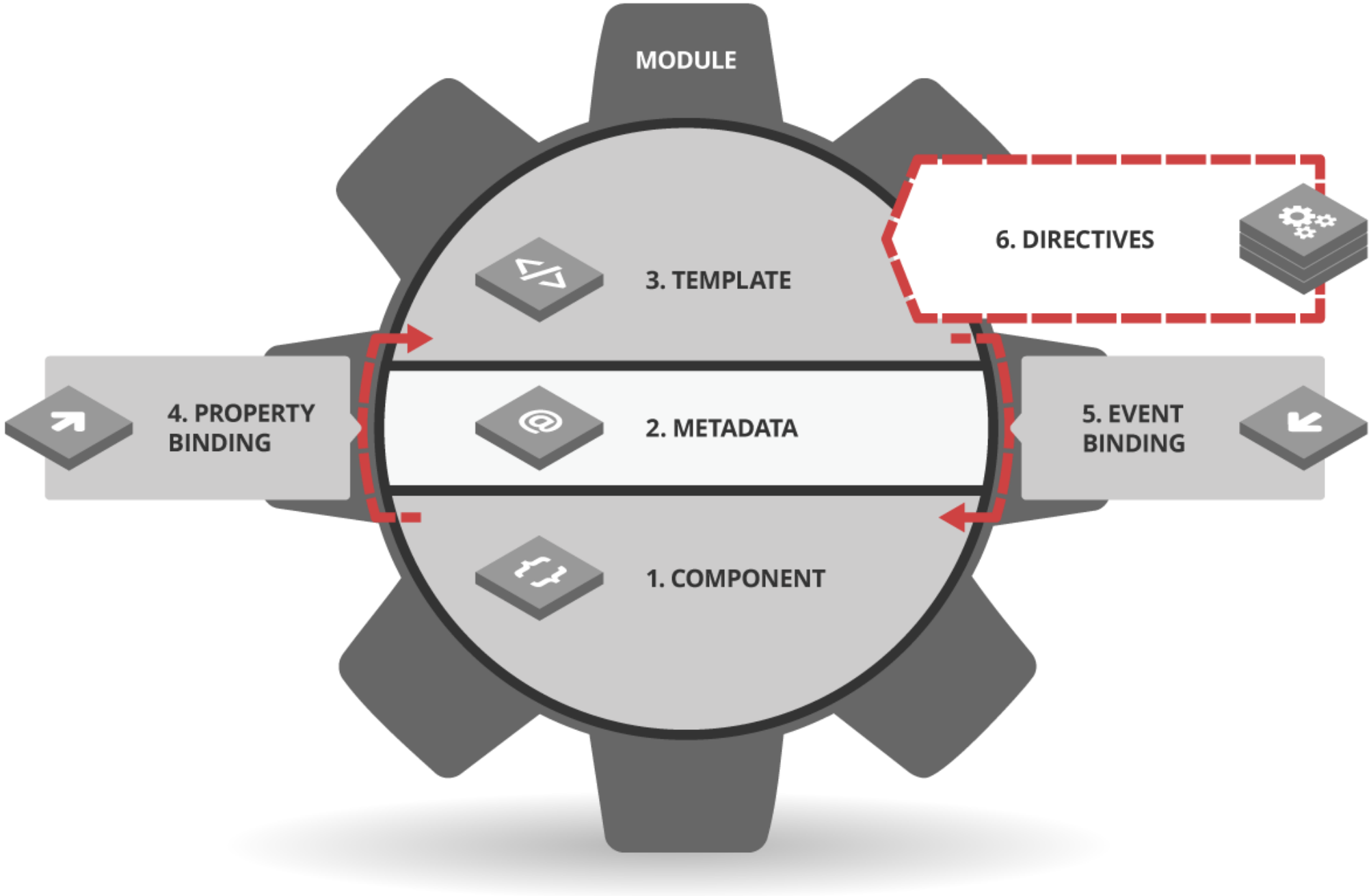

2. METADATA
1. COMPONENT

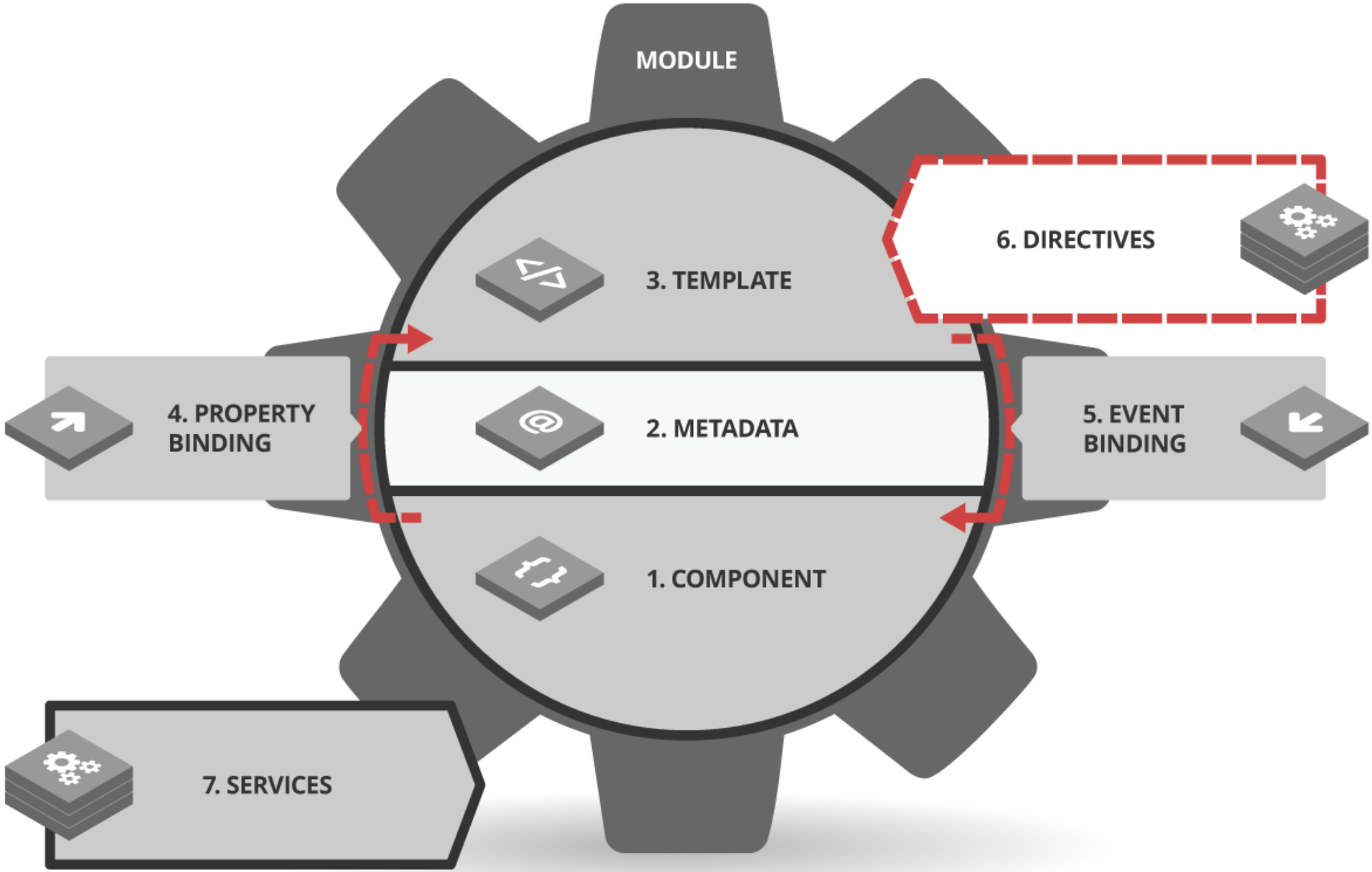

3. TEMPLATE
2. METADATA
1. COMPONENT

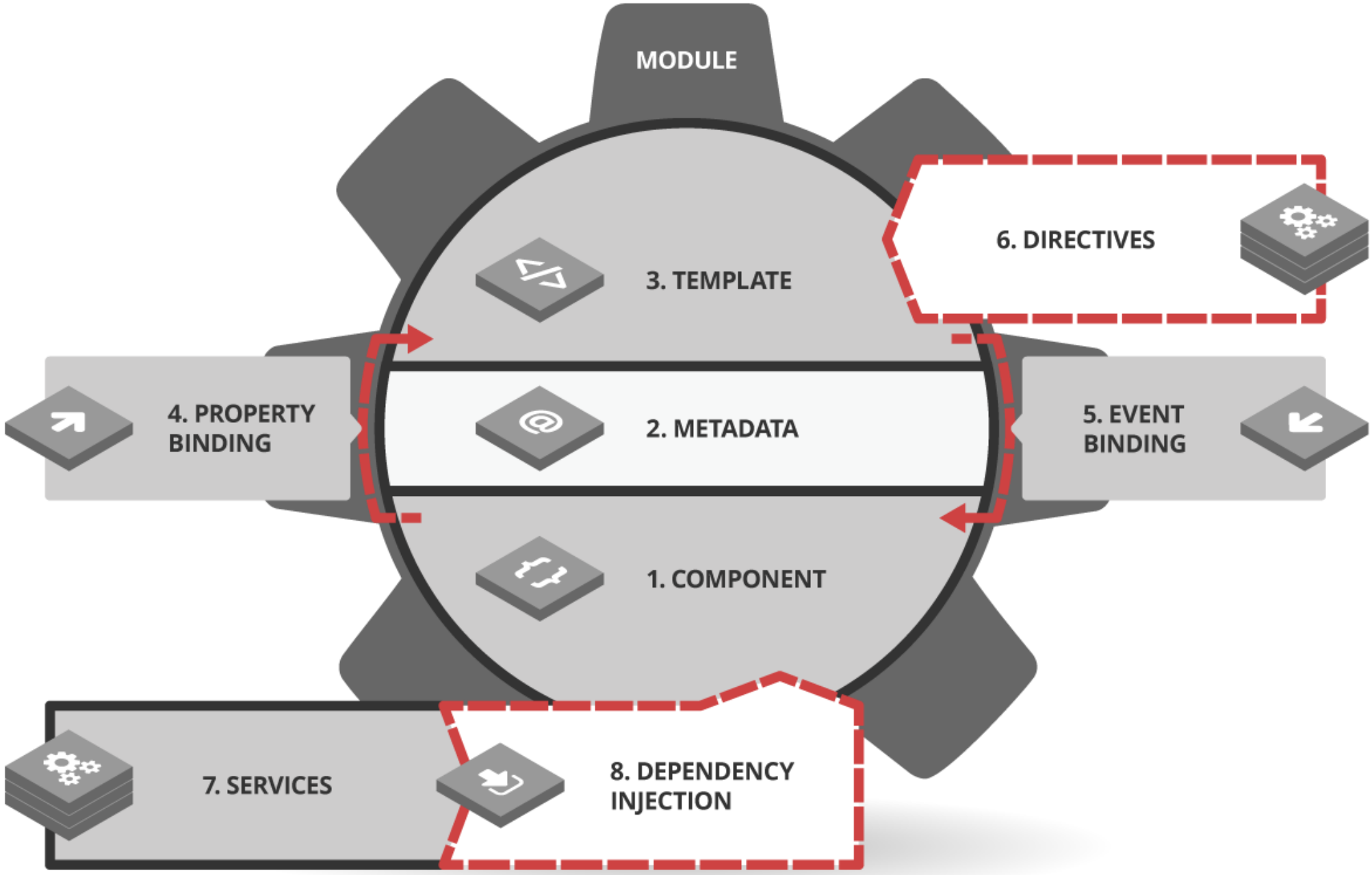

MODULE
3. TEMPLATE
2. METADATA
1. COMPONENT


MODULE
3. TEMPLATE
2. METADATA
1. COMPONENT
4. PROPERTY
BINDING

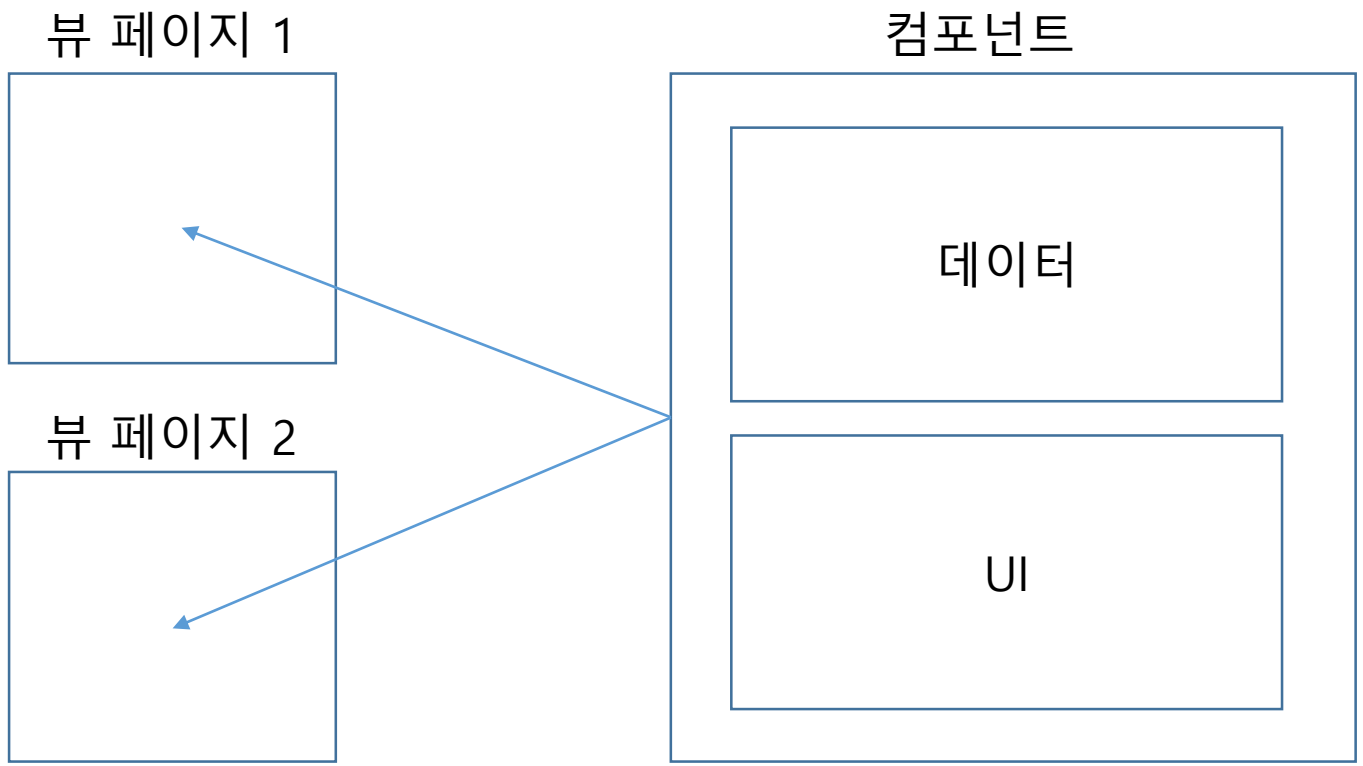

MODULE
3. TEMPLATE
2. METADATA
1. COMPONENT
4. PROPERTY BINDING
5. EVENT BINDING


MODULE
3. TEMPLATE
6. DIRECTIVES
4. PROPERTY BINDING
2. METADATA
5. EVENT BINDING
1. COMPONENT




MODULE
3. TEMPLATE
6. DIRECTIVES
4. PROPERTY BINDING
2. METADATA
5. EVENT BINDING
1. COMPONENT
7. SERVICES
8. DEPENDENCY INJECTION

컴포넌트(Component): 뷰 컴포넌트



Angular 2 컴포넌트(컨트롤러)

- App 컴포넌트
- Child 컴포넌트
- Service 컴포넌트
- Pipe 컴포넌트

컴포넌트(Component)

- 컴포넌트 = 템플릿 + 클래스 + 메타데이터
  - 템플릿
    - 레이아웃 뷰
    - HTML으로 생성
    - 바인딩과 지시자 포함
  - 클래스(TypeScript 클래스) : 대문자로 시작
    - export 키워드 지정: public 개념으로 외부에서 참조해서 사용하도록
    - 클래스 코드(속성, 메서드, 생성자)
      - 속성으로 데이터 처리: 소문자로 시작
      - 메서드로 로직 처리
    - 타입스크립트로 작성
  - 메타데이터(C#의 특성(Attribute) : @Component() 데코레이터
    - 데코레이터로 정의
    - 외부 기능(데이터) 제공

컴포넌트 클래스 생성

app.component.ts

```
1   import { Component } from '@angular/core';
2
3   @Component({
4       selector: 'my-app',
5       templateUrl: 'app/app.component.html'
6   })
7   export class AppComponent {
8       title: string = 'Angular 퀵스타트'; // 속성(필드)
9       description: string = 'Angular2 퀵스타트 [illegible]';
10      name: string = 'RedPlus';
11
12      // 컬렉션 형태의 데이터: *ngFor 구문 사용 출력
13      teches = [
14          {id: 1, title:'Angular'},
15          {id: 2, title:'ASP.NET Core'},
16          {id: 3, title:'Azure Web App'}
17      ];
18
19  }
```

Import(외부 모듈)

메타데이터와
템플릿
데코레이터 함수

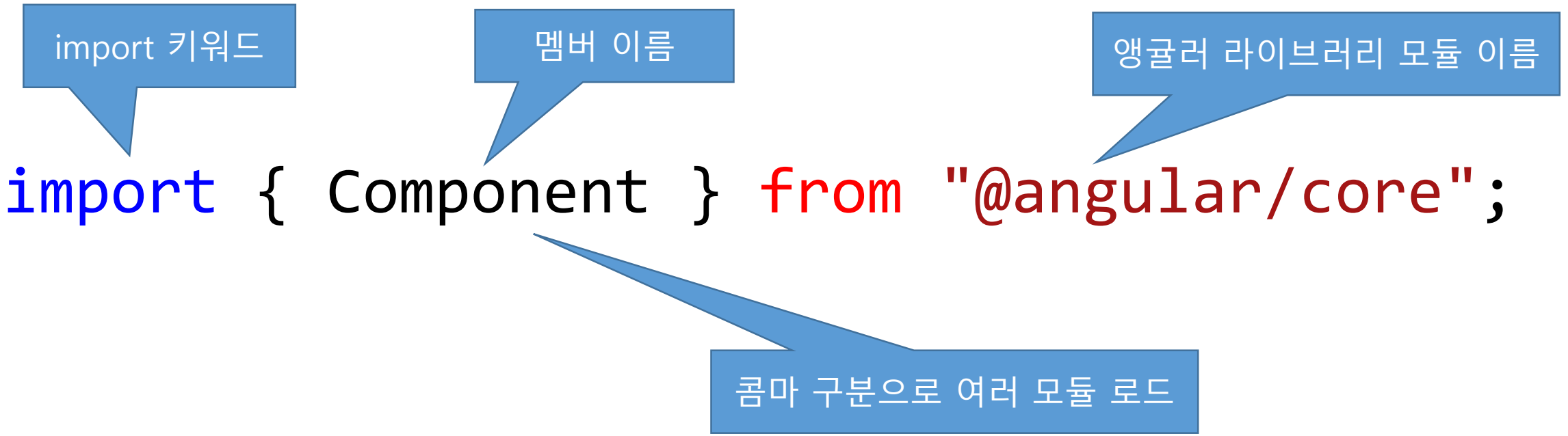
클래스
(export)

컴포넌트 이름

import -필요한 모듈 포함하기

- 외부 함수 또는 클래스를 현재 페이지에서 사용하기 위한 절차
- import 구문
  - ES2015
  - C#의 using 구문과 의도가 비슷
- 앵귤러는 모든 기능이 모듈화 됨
  - @angular/core
  - @angular/http
  - @angular/forms
  - @angular/route

import 구문


import 키워드
멤버 이름
앵귤러 라이브러리 모듈 이름
import { Component } from "@angular/core";
콤마 구분으로 여러 모듈 로드

기본 제공 AppComponent 예

app.component.html

```
<ul>
    <li *ngFor="let t of teches">
        {{t.id}} - {{t.title}}
    </li>
</ul>
```

app.component.ts

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

@Component({
    selector: 'my-app',
    templateUrl: 'app/app.component.html'
})
export class AppComponent {
    title: string = 'Angular 퀵스타트'; // 속성(필드)
    description: string = 'Angular2 퀵스타트 예제입니다.';
    name: string = 'RedPlus';

    // 컬렉션 형태의 데이터: *ngFor 구문 사용 출력
    teches = [
        {id: 1, title:'Angular'},
        {id: 2, title:'ASP.NET Core'},
        {id: 3, title:'Azure Web App'}
    ];

}
```

메타데이터 정의(컴포넌트 설명)

- import { Component } from '@angular2/core'; 식 필요

- 데코레이터(Decorator) 함수
  - @ 접두사 사용
  - 클래스와 멤버들에 메타데이터를 주는 함수
  - ES2016의 특징
- @Component() 데코레이터
  - 기본으로 내장된 데코레이터
  - selector 속성 : HTML에서 사용하는 태그(선택자(지시자;디렉티브)) 이름
  - template 속성: 뷰 레이아웃(HTML)
    - templateUrl: 뷰 페이지를 따로 HTML 파일로 지정
      - Webpack 방식에서는 template: require('./*.html') 형태를 사용
  - {{}} : 바인딩 식

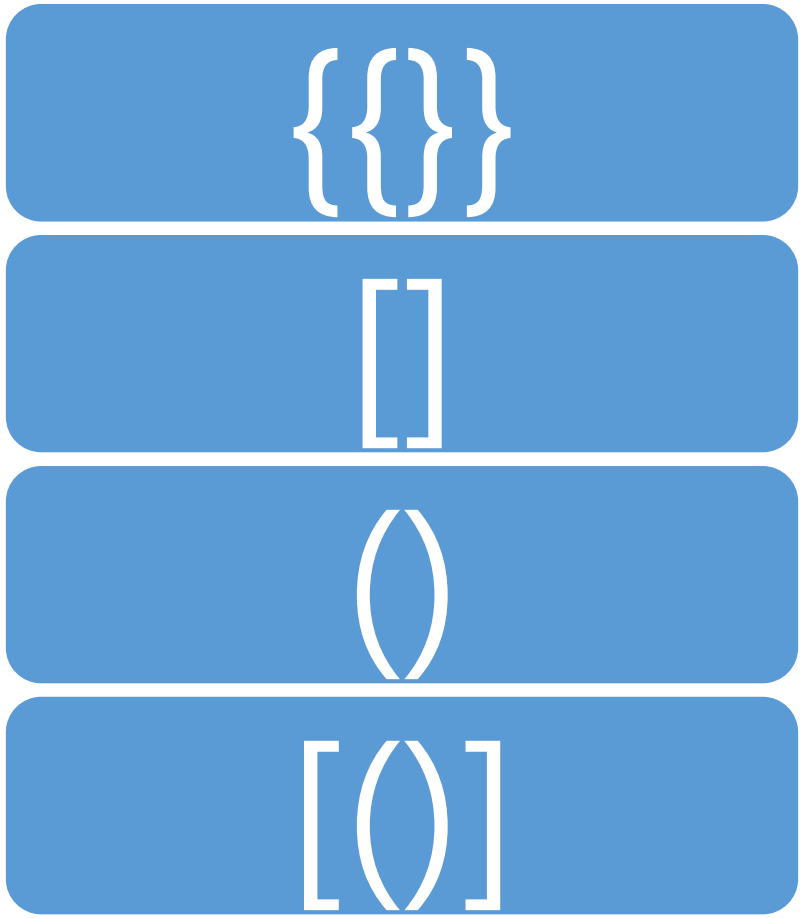
템플릿(Template) : View 페이지(HTML)

- 인라인 템플릿
  - 싱글 라인
    - template: “”
  - 멀티 라인: ES 2015 백 틱(`) 기호 사용
    - template: ` … `
- 링크드 템플릿
  - templateUrl: ‘*.html’

```
template: '<div><h1>싱글라인 템플릿</h1></div>'
```

```
template: `
<div>
    <h1>멀티라인 템플릿</h1>
</div>
`
```

템플릿(HTML 페이지)에 데이터 바인딩



표현식(Expressions)

- 표현식(바인딩 표현식, 핸들바, 데이터 바인딩)
  - {{ }} 로 사용됨
  - JavaScript의 모든 문법을 지원하지 않음
    - Subset
  - 배열 등 선언 가능
- 사용 예
  - {{ 표현식 }}
  - {{ name }}
  - {{ 3 % 5 }}
  - {{ “안녕” + “하세요” }}

```
{{ expression }}
{{ name }}
{{ amount * 123 + 4 }}
{{ number in [1, 3, 5] }}
```

{{}}

- 바인딩 표현식
  - 할당 : =, +=, ++ 등
  - new 키워드
  - 세미콜론으로 여러 개 지정
  - 전역 변수
  - 단방향 바인딩
  - 빠름
  - 간단함
  - 컴포넌트의 값 출력
    - null값인 경우 ?. 형태 사용 가능: {{note?.title}}
  - 출력 전용으로만 쓰자!!!

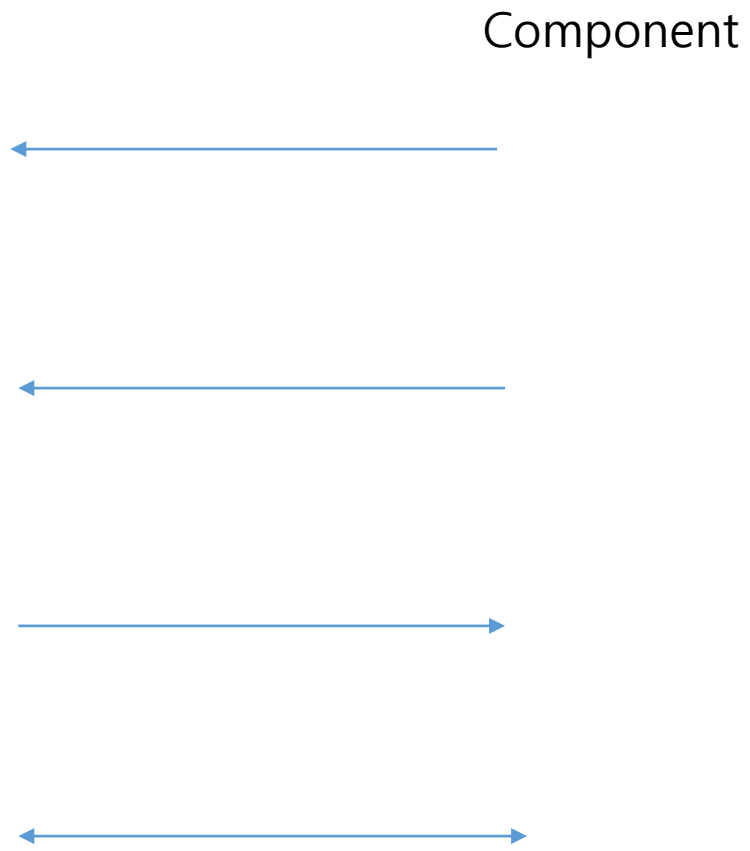
moduleId: module.id

- @Component({ moduleId: module.id })
- WebPack 방식에서는 필요없음

Angular 2 데이터 바인딩

데이터 바인딩

- 바인딩(문자열 보간법(String Interpolation))
  - {{}}
    - {{ 속성 }}
    - {{ title }}, {{ vm.note }}
    - {{ 표현식 }}, {{ 3 + 5 }}
- 프로퍼티(Property) 바인딩: One Way 바인딩
  - [속성]='구문'
  -
- 이벤트(Event) 바인딩
  - (이벤트)='메서드($event)'
  - <button (click)='' />
- 양방향 바인딩: Two Way 바인딩
  - [(ngModel)]='속성'
    - FormsModule 참조



바인딩 표현식(보간법)(핸들바): {{ }}

Property 바인딩


```
<img [src] = ’maxims.image’>

<img src = {{maxims.image}}>
```

public 속성과 private 필드

- ES2015와 TypeScript에서 클래스 멤버들은 기본값으로 public 이다.

속성 바인딩(Property Binding) : [ ]

- 프로퍼티(속성) 바인딩: []
  - 요소: [src]
  - 컴포넌트 속성: [m_field]
  - 디렉티브 속성: [ngClass], [ngStyle]
- 컴포넌트 => DOM
  - 속성의 값을 DOM의 영역에 출력
  - 컴포넌트의 값을 DOM에 출력

- <img [src]='member.photoUrl'>
- <img [src]={{member?.photoUrl}}>
-

단방향 바인딩

Angular 1

- <h1 ng-bind="vm.car.name"></h1>

Angular 2

- <h1 [innerText]="car.name"></h1>
- <div [style.color]="color">{{ Title}}

단방향 바인딩

- [attr.aria-label]
  - 특성 바인딩
  - 알려진 속성이 아닌 경우에는 이 방식을 사용
- [class.btn]
  - 클래스 속성 바인딩
  - [class.black] : 조건이 맞으면 class="black" 추가됨
- [style.color], [style.background]="'#C0FFEE'"
  - 스타일 속성 바인딩

속성 바인딩과 이벤트 바인딩

- Property Binding
- Event Binding

app.component.html

```
1   <h1>{{ title }}</h1>
2   <div>
3       안녕하세요. {{ name }} 님.
4       <br />
5       {{ description }}
6       <hr />
7       <input
8           type="text"
9           [value]="name"
10          (keyup)="name = $event.target.value"
11      />
12  </div>
```

속성 바인딩

이벤트 바인딩

이벤트 바인딩

- DOM에 있는 것을 컴포넌트로 가져옴
  - <button (click)='btnClick_Click()' />
    - () : 타겟 이벤트
    - '' : 템플릿 구문
  - <button (click)="btnSave_Click()">Save</button>

- 주요 이벤트 바인딩
  - (click)=''
  - (focus)=''
  - (blur)=''
  - (keyup)=''
  - (mouseenter)="color = 'red'"
  - (mouseleave)="color = 'blue'"

이벤트 바인딩(Event Binding) : ( )

Angular 1

- <button ng-click="vm.check('click')"></button>

Angular 2

- <button (click)="check('click')"></button>

```
<button (click)="btnToggle()">토글</button>
<div [hidden]="flg">
  보였다. 안 보였다.
</div>
```

```
16    flg = true;
17    btnToggle() {
18      this.flg = !this.flg;
19    }
```

#이름지정

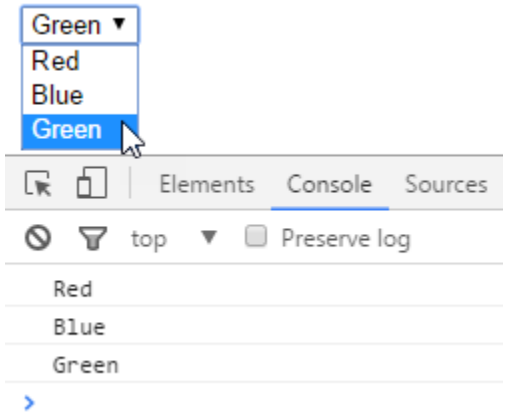
- 특정 폼 컨트롤에 #별칭 형태로 ID 속성을 지정 가능
  - <input #txtName />
    - txtName.value로 값 접근 가능

app.component.html

```
<h1>
  {{title}}
</h1>

<select #lstSelect
  (change)="selection(lstSelect.value)">
  <option>Red</option>
  <option>Blue</option>
  <option>Green</option>
</select>
```

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = '#으로 컨트롤 참조';

  selection(v) {
    console.log(v);
  }
}
```



$event.target.value

- (blur)="name = $event.target.value"

```
<select
  (change)="selection($event.target.value)">
  <option>Red</option>
  <option>Blue</option>
  <option>Green</option>
</select>
```

ngModel과 Two-Way 데이터 바인딩

- FormsModule에 들어있는 기본 제공 디렉티브
  - AppModule에 등록 필요
- 바인딩 전: 반드시 name 특성 필요
  - <input name="txtName" ngModel />
- 단방향 바인딩
  - <input name="txtName" [ngModel]="txtName" />
- 양방향 바인딩: [()] 표현식 사용, 박스에 든 바나나?
  - <input name="txtName" [ngModel]="txtName" (ngModelChange)="txtName=$event" />
  - <input name="txtName" [(ngModel)]="txtName" />

양방향 바인딩(Two-way Binding)

- [(ngModel)] = '속성'

app.module.ts

```ts
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';

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

@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

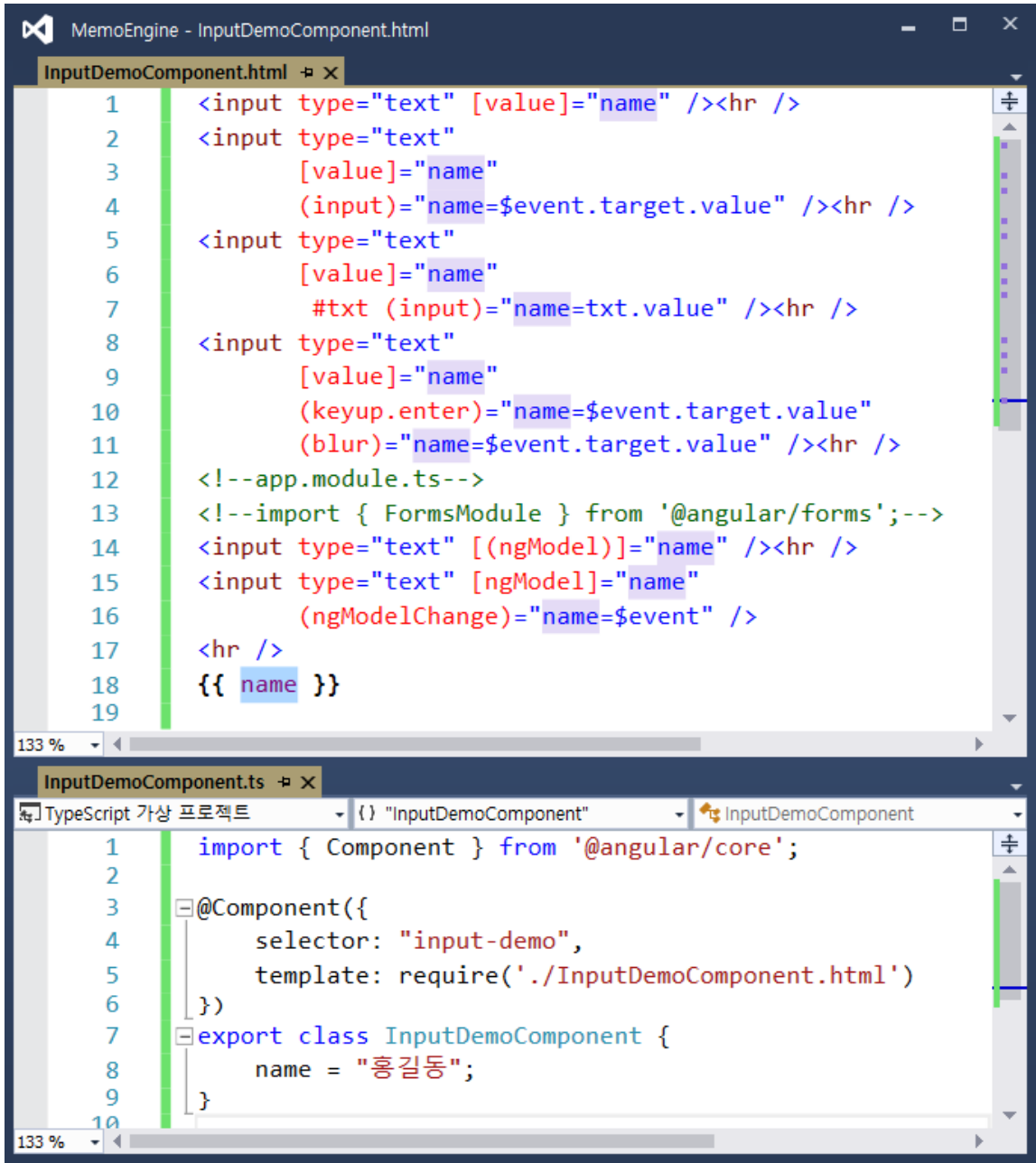
app.component.html

```html
<h1>{{ title }}</h1>
<div>
    안녕하세요. {{ name }} 님.
    <br />
    {{ description }}
    <hr />
    <input
        type="text"
        [value]="name"
        (keyup.enter)="name = $event.target.value"
    />

    <input
        type="text"
        [(ngModel)]="name"
    />

</div>
```

텍스트박스

- 텍스트박스 양방향 바인딩
  - [ngModel]="name"
  - (ngModelChange)="name=$event"

```
<input type="text" [(ngModel)]="name" />
```



앵귤러4 기본 지시자

if문

for문

switch문

기타(CSS 관련)

바인딩 복습: {{}}과 null 가드

- {{ 식 }}
  - {{ product.name }}
  - {{ 3 + 5 }}
- {{ model?.name }}
- 입력 바인딩: [속성]="값"
- 출력 바인딩: (이벤트)="이벤트처리기()"
- 양방향 바인딩: [(속성)]="값"

지시자(Directive;디렉티브)

- *
  - 애스터리스크 기호를
  - 사용하여 지시자 호출
    - *ngFor
    - *ngIf
    - *ngSwitch

IfThenElse.component.ts

```
import * as core from '@angular/core';

@core.Component({
    selector: 'if-then-else',
    template: `
        <button (click)="toggle()">토글</button>
        <div *ngIf="isShow">첫번째 뷰</div>
        <div *ngIf="!isShow">두번째 뷰</div>
        <div *ngIf="isShow;else secondView">첫번째 뷰</div>
        <ng-template #secondView>두번째 뷰</ng-template>

        <div *ngIf="isShow;then fView else sView"></div>
        <ng-template #fView>첫번째 뷰</ng-template>
        <ng-template #sView><div>두번째 뷰</div></ng-template>
    `
})
export class IfThenElseComponent {
    isShow: boolean = true;
    toggle(): void {
        this.isShow = !this.isShow;
    }
}
```

지시자(Directive;디렉티브)

- *ngIf 디렉티브 : if문
- *ngFor 디렉티브 : for문(foreach문),
  - 컬렉션(리스트) 형태 데이터 바인딩
  - *ngFor = “let member of members, let I = index”
    - let : 로컬 변수 선언
- *ngSwitch 디렉티브 : switch문
- BrowserModule 모듈에 포함된 CommonModule에 정의됨

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

@Component({
    selector: 'blog-category',
    templateUrl: './blog-category.component.html'
})
export class BlogCategoryComponent {
    categories = ['Angular', 'ASP.NET', 'Azure'];
}
```

```
<div class="panel panel-default">
  <div class="panel-heading">카테고리 리스트</div>
  <div class="panel-body" [hidden]="false">
    <ul>
      <li *ngFor="let c of categories">{{c}}</li>
    </ul>
  </div>
</div>

```

내장 지시자: BrowserModule

- BrowserModule(CommonModule 포함)
- *ngIf
  - If문
    - Angular 1.X => ng-if
- *ngFor
  - for문
    - Angular 1.X => ng-repeat

```
export class ProductListComponent
{
    // title 속성(Property)
    public Title:string = "상품 관리 - 상품 리스트";
    public Products: any[] = [
        { "productId": 1, "productName": "좋은 책", "price": 1000 },
        { "productId": 2, "productName": "좋은 강의", "price": 3000 },
        { "productId": 3, "productName": "좋은 컴퓨터", "price": 5000 }
    ];
}
```

```
                2000
            </td>
        </tr>
        <!-- {{ 문자열 보간법 }} -->
        <tbody *ngIf='Products && Products.length'>
            <tr *ngFor='let p of Products'>
                <td>{{ p.productId }}</td>
                <td>{{ p.productName }}</td>
                <td>{{ p.price }}</td>
            </tr>
        </tbody>
    </table>
```

*ngIf, *ngFor => 구조적 지시자

- 구조적 지시자(Structural Directives) : * 접두사 사용
  - 구조를 변경

```
@Component({
    selector: 'about-about',
    templateUrl: "app/about/about.component.html"
})
export class AboutComponent {
    public Title: string = "정보.";
    public Sites: any[] = [
        { "siteName": "데브렉", "siteUrl": "http://www.devle.
        { "siteName": "박용준", "siteUrl": "http://www.youtub
    ];
}
```

```
<h3>추천 사이트</h3>
<ul *ngIf='Sites.length'>
    <li *ngFor='let s of Sites'>
        <a [href]='s.siteUrl'>
            {{ s.siteName }}
        </a>
    </li>
</ul>
```

기본 내장 지시자 사용법 예

Angular 1

- ng-repeat=“”
- ng-if=“”

Angular 2, 4

- *ngFor=“”
  - let 키워드: 지역 변수 선언
- *ngIf=“”

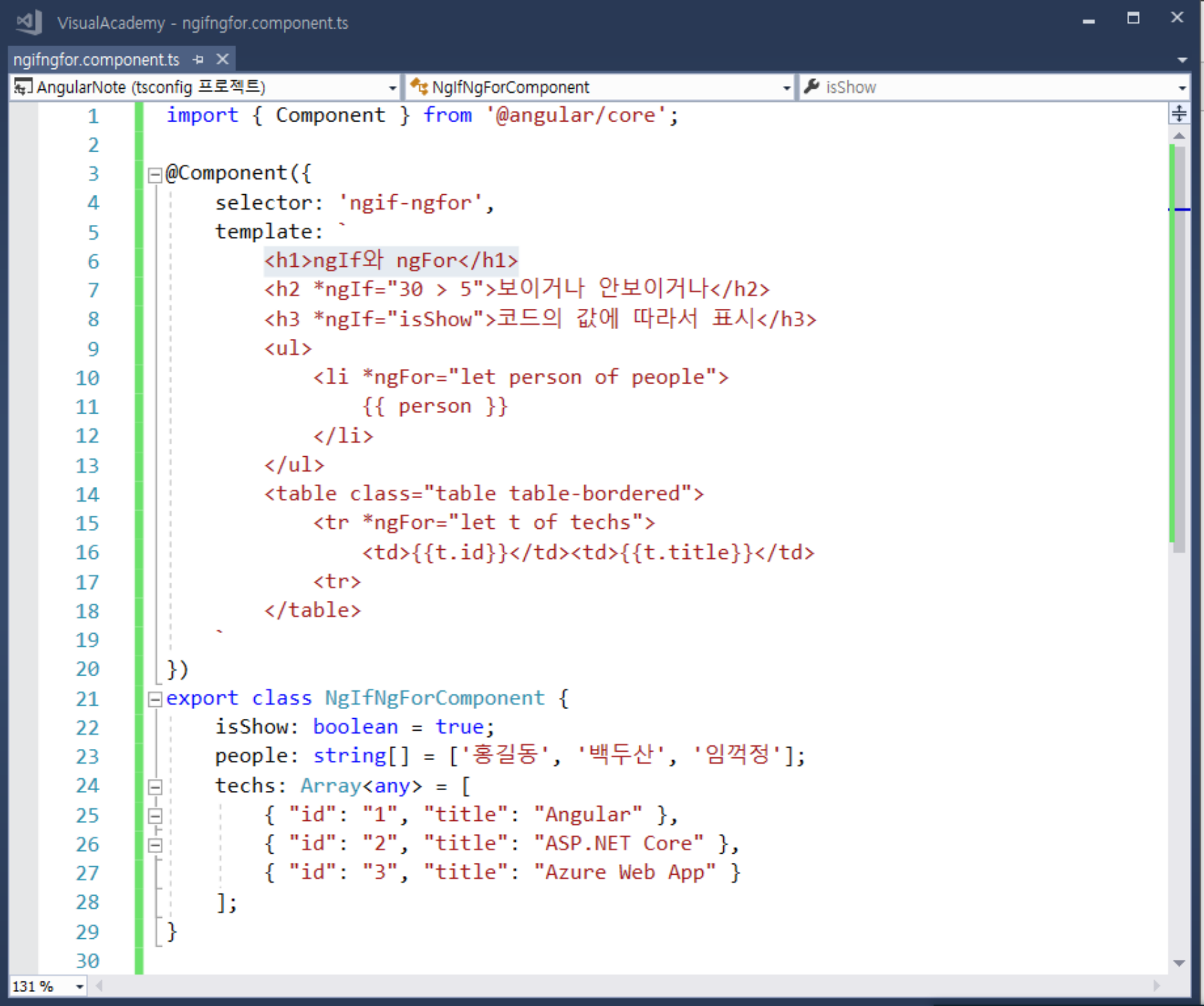
*ngFor를 사용하여 컬렉션 데이터 출력

app.component.html

```
23  <h2>기술 리스트</h2>
24
25  <ul>
26      <li>
27          {{ teches[0].id }} - {{ teches[0].title }}
28      </li>
29      <li>
30          {{ teches[1].id }} - {{ teches[1].title }}
31      </li>
32      <li>
33          {{ teches[2].id }} - {{ teches[2].title }}
34      </li>
35  </ul>
36
37  <ul>
38      <li *ngFor="let t of teches">
39          {{t.id}} - {{t.title}}
40      </li>
41  </ul>
```

app.component.ts

```
1   import { Component } from '@angular/core';
2
3   @Component({
4       selector: 'my-app',
5       templateUrl: 'app/app.component.html'
6   })
7   export class AppComponent {
8       title: string = 'Angular 퀵스타트'; // 필드
9       description: string = 'Angular2 퀵스타트 예제입니다.';
10      name: string = 'RedPlus';
11
12      // 컬렉션 형태의 데이터: *ngFor 구문 사용 출력
13      teches = [
14          {id: 1, title:'Angular'},
15          {id: 2, title:'ASP.NET Core'},
16          {id: 3, title:'Azure Web App'}
17      ];
18
19  }
```


VisualAcademy - ngifngfor.component.ts
ngifngfor.component.ts
AngularNote (tsconfig 프로젝트)
NgIfNgForComponent
isShow
import { Component } from '@angular/core';
@Component({
selector: 'ngif-ngfor',
template: `
<h1>ngIf와 ngFor</h1>
<h2 *ngIf="30 > 5">보이거나 안보이거나</h2>
<h3 *ngIf="isShow">코드의 값에 따라서 표시</h3>

<li *ngFor="let person of people">
{{ person }}

<table class="table table-bordered">
<tr *ngFor="let t of techs">
<td>{{t.id}}</td><td>{{t.title}}</td>
<tr>
</table>
`
})
export class NgIfNgForComponent {
isShow: boolean = true;
people: string[] = ['홍길동', '백두산', '임꺽정'];
techs: Array<any> = [
{ "id": "1", "title": "Angular" },
{ "id": "2", "title": "ASP.NET Core" },
{ "id": "3", "title": "Azure Web App" }
];
}
131 %


Home Page - AngularN
localhost:2279/ngifngfor
ngIf와 ngFor
보이거나 안보이거나
코드의 값에 따라서 표시
홍길동
백두산
임꺽정
1 Angular
2 ASP.NET Core
3 Azure Web App

어트리뷰트 지시자(Attribute Directive)

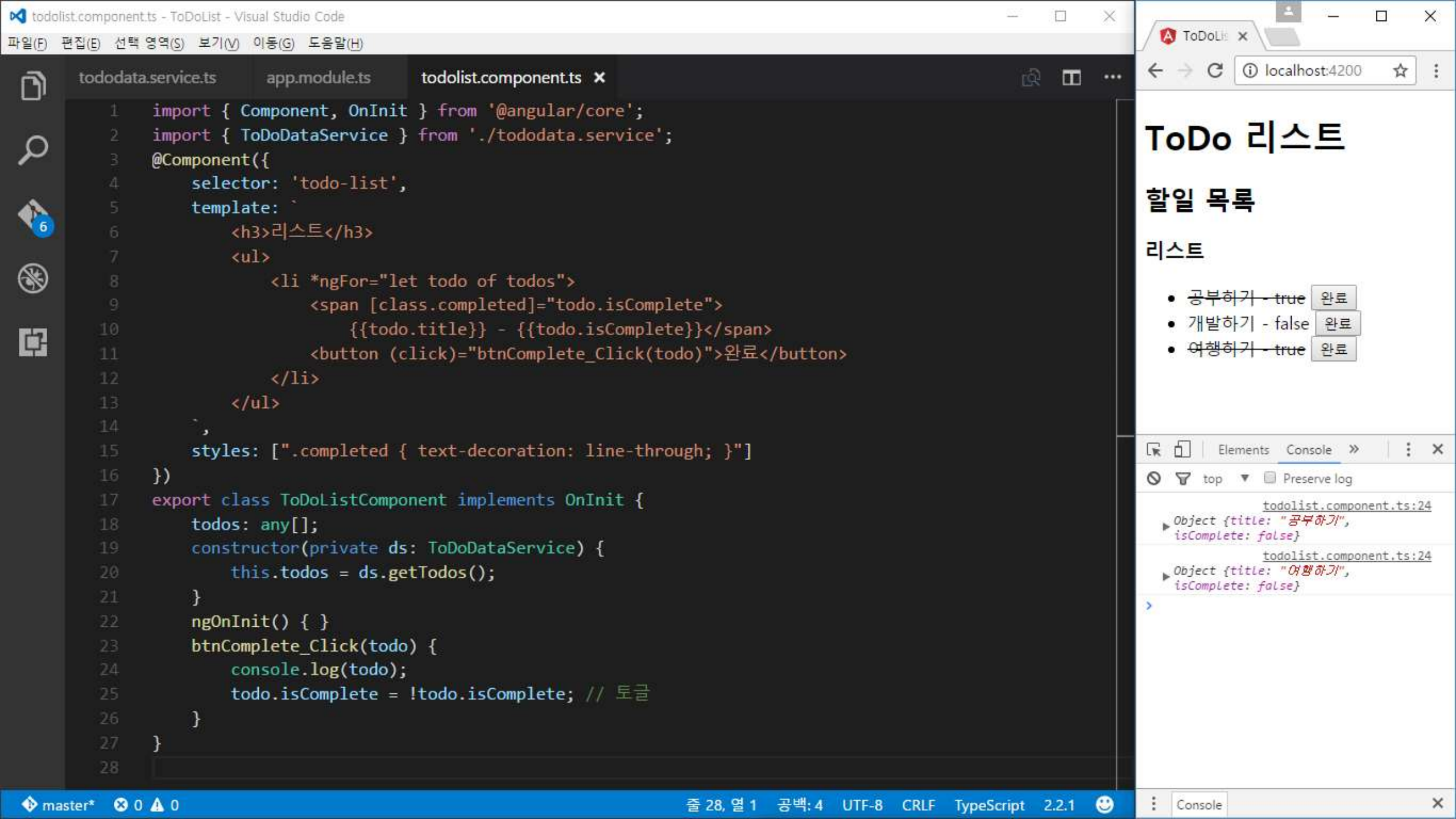
- [class.blue] = “true”
  - blue 스타일을 적용할건지
- [ngClass] = “”
- [ngStyle]= “”

DOM 숨기기

- DOM 숨기기
  - [hidden]="true or false“
  - *ngIf를 사용해도 됨

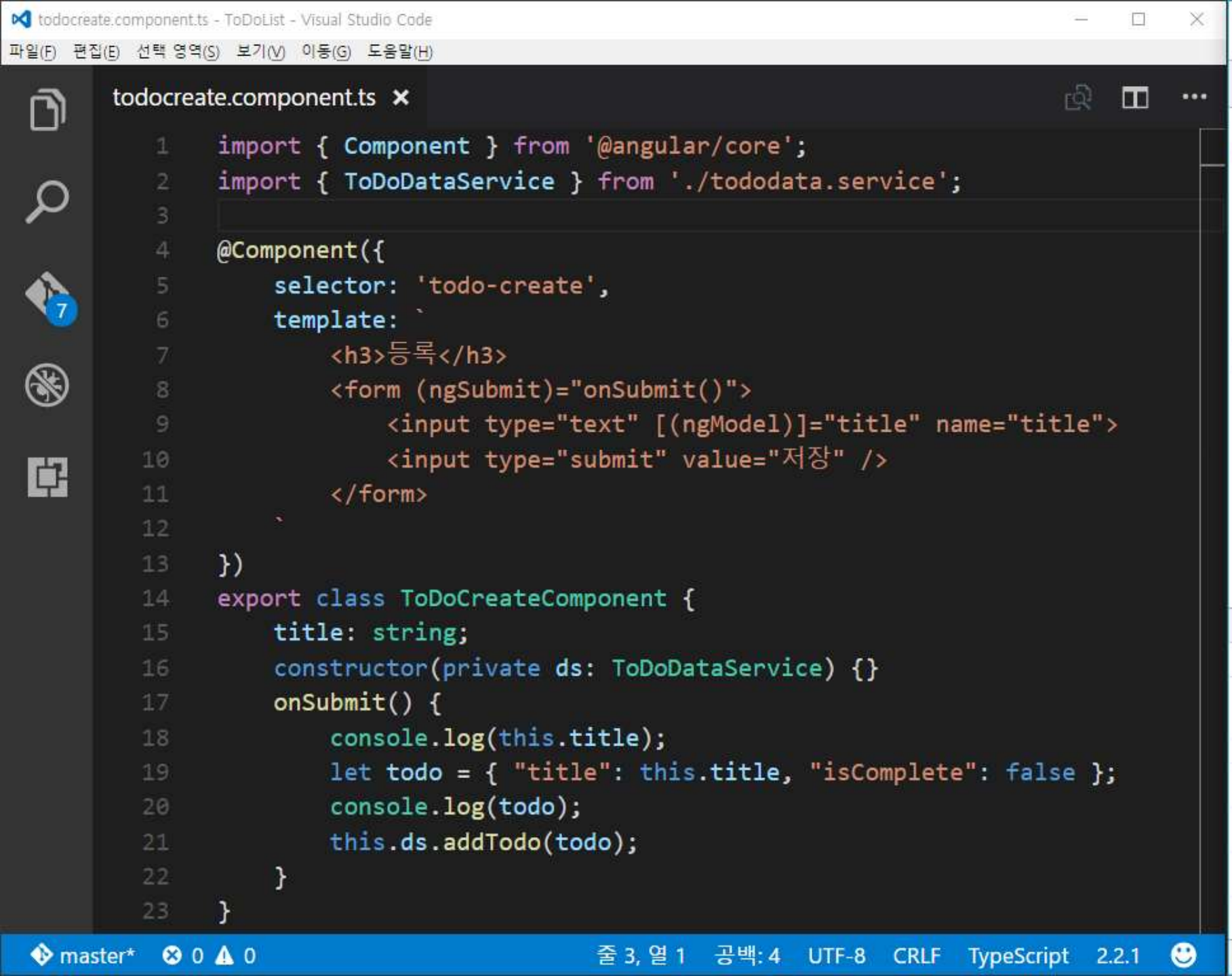
ngStyle, ngClass

- ngStyle
  - [style.스타일명]
  - [ngStyle]="setStyles()"
- ngClass
  - [class.클래스명]
  - [ngClass]="setClasses()"
    - [ngClass]="{cssClassName: true}"
      - <div [ngClass]="{a:true, b:false}" />
      - <div [ngClass]="funcMyFunction()" />
        - 개체 리터럴 형태로 반환
          - {a:true, b:false}
        - 또는 문자열 배열로 직접 반환
          - ["a b"] 또는 []


todolist.component.ts - ToDoList - Visual Studio Code
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 도움말(H)
tododata.service.ts
app.module.ts
todolist.component.ts
import { Component, OnInit } from '@angular/core';
import { ToDoDataService } from './tododata.service';
@Component({
selector: 'todo-list',
template: `
<h3>리스트</h3>

<li *ngFor="let todo of todos">
<span [class.completed]="todo.isComplete">
{{todo.title}} - {{todo.isComplete}}
<button (click)="btnComplete_Click(todo)">완료</button>

`,
styles: [".completed { text-decoration: line-through; }"]
})
export class ToDoListComponent implements OnInit {
todos: any[];
constructor(private ds: ToDoDataService) {
this.todos = ds.getTodos();
}
ngOnInit() { }
btnComplete_Click(todo) {
console.log(todo);
todo.isComplete = !todo.isComplete; // 토글
}
}
master*
0 0
줄 28, 열 1
공백: 4
UTF-8
CRLF
TypeScript
2.2.1
ToDoLi
localhost:4200
ToDo 리스트
할일 목록
리스트
공부하기 - true 완료
개발하기 - false 완료
여행하기 - true 완료
Elements
Console
top
Preserve log
todolist.component.ts:24
Object {title: "공부하기", isComplete: false}
todolist.component.ts:24
Object {title: "여행하기", isComplete: false}
Console


todocreate.component.ts - ToDoList - Visual Studio Code
파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 도움말(H)
todocreate.component.ts
import { Component } from '@angular/core';
import { ToDoDataService } from './tododata.service';
@Component({
selector: 'todo-create',
template: `
<h3>등록</h3>
<form (ngSubmit)="onSubmit()">
<input type="text" [(ngModel)]="title" name="title">
<input type="submit" value="저장" />
</form>
`
})
export class ToDoCreateComponent {
title: string;
constructor(private ds: ToDoDataService) {}
onSubmit() {
console.log(this.title);
let todo = { "title": this.title, "isComplete": false };
console.log(todo);
this.ds.addTodo(todo);
}
}
master* 0 0 줄 3, 열 1 공백: 4 UTF-8 CRLF TypeScript 2.2.1


ToDoList
localhost:4200
ToDo 리스트
할일 목록
등록
놀기
저장
리스트
공부하기 - false 완료
개발하기 - true 완료
여행하기 - true 완료
놀기 - false 완료
Elements Console Sources
top Preserve log
Angular is running in the development mode. Call enableProdMode() to enable the production mode. lang.js:130
놀기 todocreate.component.ts:18
todocreate.component.ts:20
Object {title: "놀기", isComplete: false}
todolist.component.ts:24
Object {title: "개발하기", isComplete: false}
todolist.component.ts:24
Object {title: "여행하기", isComplete: false}
Console

*ngSwitch

- [ngSwitch]=“”
  - *ngSwitchCase=“'aaa'”
  - *ngSwitchCase=“'bbb'”
  - *ngSwitchDefault

*ngSwitch 디렉티브

```
@Component({
    selector: 'TopRanking',
    template: `

<div [ngSwitch]="rank" class="text-center">
    <img src="/images/One.gif" *ngSwitchCase="1" />
    <img src="/images/Two.gif" *ngSwitchCase="2" />
    <img src="/images/Three.gif" *ngSwitchCase="3" />
    <span *ngSwitchDefault>{{ rank }}</span>
</div>

    `
})
export class TopRankingComponent {
    @Input()
    public rank: any;
}
```

배지
1
2
3
4

모델과 컴포넌트

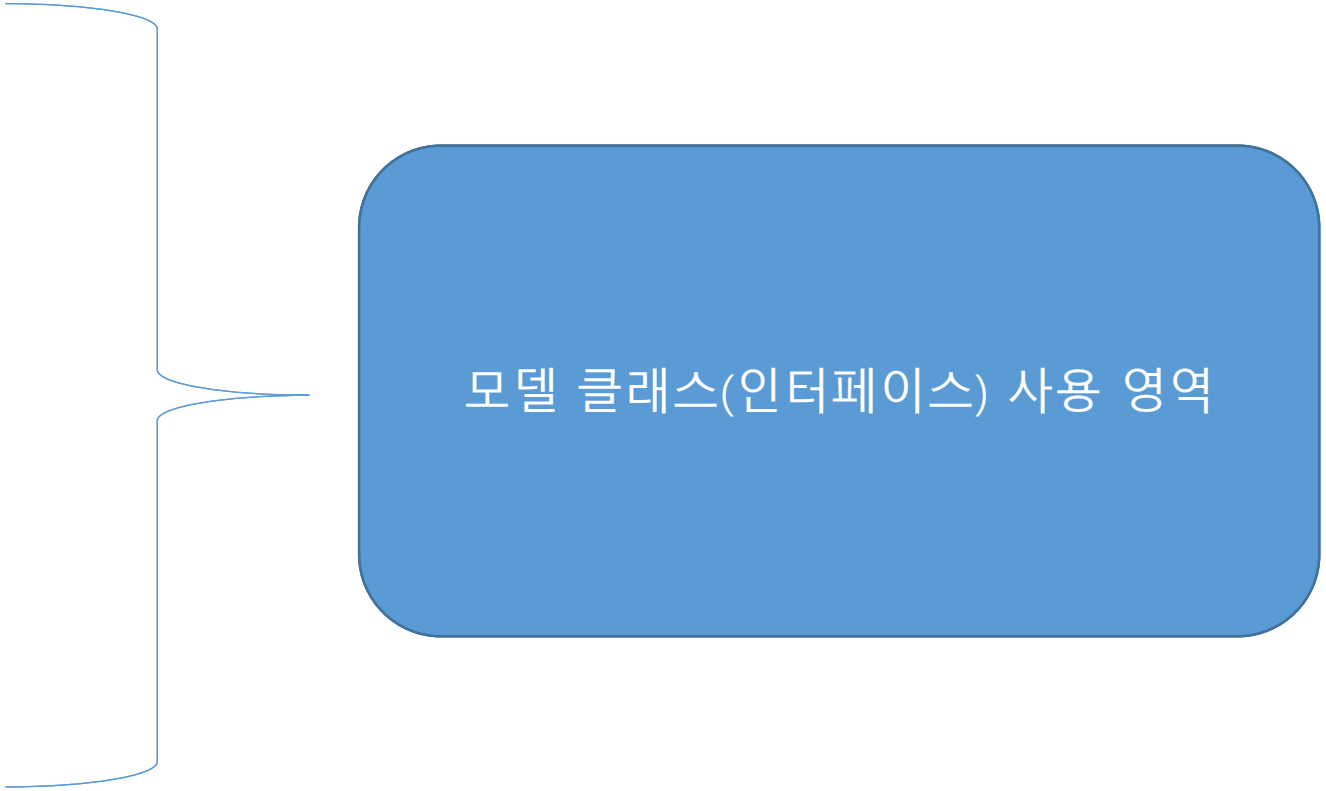
모델 클래스

인터페이스

컴포넌트 라이프사이클

앵귤러 모델

- 디렉티브
  - 컴포넌트
  - 어트리뷰트
  - HTML
- 데이터 흐름
  - 문자열 보간법
  - 이벤트 바인딩
  - 양방향 바인딩
- 프로바이더
  - 서비스
    - 재 사용 가능한 로직
    - 데이터 저장 및 CRUD
  - 라이브러리



export class 클래스명 { 멤버; }

blog.post.mainsummary.ts

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

//import { PostViewModel, DateViewModel } from './blog.model';

@Component({
    selector: 'blog-post-main-summary',
    template: require('./blog.post.mainsummary.html'),
    styles: [require('./blog.post.mainsummary.css')]
})
export class BlogPostMainSummaryComponent {
    posts : PostViewModel[] = [
        { postId: 1, "title": "Angular 2", "name": "박용준", date: {month: "Jan", day: 25}},
        { postId: 2, "title": "ASP.NET Core", "name": "한상훈", date: {month: "Jan", day: 26}}
    ];
}

export class PostViewModel {
    postId: number;
    title: string;
    name: string;
    date: DateViewModel;
}

export class DateViewModel {
    month: string;
    day: number;
}
```

blog.post.mainsummary.html

```html
<h3>최근 글 리스트</h3>
<ul class="notice">
    <li *ngFor="let post of posts">
        <div class="date">
            <div class="month">{{post.date.month}}</div>
            <div class="day">{{post.date.day}}</div>
        </div>
        <div class="details">
            <span class="title">
                <a href="/Blog/Post/{{post.postId}}">
                    {{post.title}}
                </a>
            </span>
            <span class="name">{{post.name}}</span>
        </div>
    </li>
</ul>
```

@Component 데코레이터

- selector
- template
- directives
- providers
  - 특정 서비스 클래스를 사용하겠다고 지정
  - providers: [BlogService]

컴포넌트 고급

- 강력한 형식과 인터페이스
- 스타일 캡슐화
- 라이프 사이클
- 커스텀 파이프
- 중첩된 컴포넌트

강력한 형식과 인터페이스

- 타입스크립트의 인터페이스
- 개발 시점에서만 사용
- 속성과 메서드에 대한 시그니처 제공
- 제약
  - 속성 및 메서드 이름 잘못 입력시 에러 제공

IProduct.ts

```
1   export interface IProduct
2   {
3       id: number;
4       modelName: string;
5       price: number;
6   }
```

app.component.ts

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

// 인터페이스: 모델/스펙
interface ITech
{
    id: number;
    title: string;
}

@Component(
    selecto: 'my-app',
    templateUrl: 'app/app.component.html'
})
export class AppComponent {
    title: string = 'Angular 퀵스타트'; // 속성(필드)
    description: string = 'Angular2 퀵스타트 예제입니다.';
    name: string = 'RedPlus';

    // 컬렉션 형태의 데이터: *ngFor 구문 사용 출력
    teches: ITech[] = [
        {id: 1, title:'Angular'},
        {id: 2, title:'ASP.NET Core'},
        {id: 3, title:'Azure Web App'},
        {id: 4, title:'TypeScript'}
    ];

}
```

```ts
// 컬렉션 형태의 데이터: *ngFor 구문 사용 출력
teches: ITech[] = [
    {id: 1, title:'Angular'},
    {id: 2, title:'ASP.NET Core'},
    {id: 3, titles:'Azure Web App'},
    {id: 4, title:'TypeScript'}
];
```

컴포넌트 라이프사이클

컴포넌트 라이프사이클

- ngOnInit
- ngOnChanges
- ngAfterViewInit
- ngOnDestroy
- …

컴포넌트 라이프사이클 잡기

- OnInit
  - 초기화
- OnChanges
  - 입력 속성의 값이 변경되었을 때
    - @Input 데코레이터로 지정된 값
- OnDestroy
  - 마무리

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

export class HeroesComponent implements OnInit {

    ngOnInit(): void {
      this.getHeroes();
    }
```

앵귤러 서비스

```
// 서비스 클래스
import { Injectable } from '@angular/core';

@Injectable()
export class AppService {
    getCourses() {
        return COURSES;
    }
}

const COURSES = [
    {id: 1, title: "Angualr2"},
    {id: 2, title: "ASP.NET Core"},
    {id: 3, title: "Azure Web App"},
    {id: 4, title: "Bootstrap"}
];
```

- 1 - Angualr2
- 2 - ASP.NET Core
- 3 - Azure Web App
- 4 - Bootstrap

서비스

- 특정 비즈니스 로직을 다른 파일에서 관리
- 여러 서비스를 만들고 놓고 필요할 때마다 가져다(Inject) 사용
  - 생성자에 매개변수 주입 방식의 DI
- Shared 폴더
- XXXService.ts
- XXXDataService.ts

서비스 클래스

- 다른 클래스에게 특정 서비스(기능)을 부여하는 목적

- Angular 1.X의 서비스
  - 팩토리
  - 서비스
  - 프로바이더
  - 상수
  - 값
- Angular 2의 서비스
  - Angular 1.X의 모든 서비스 개념이 클래스로 포함됨

```
1   // 서비스 클래스 생성
2   import { Injectable } from '@angular/core';
3
4   @Injectable()
5   export class MissionService {
6       getMissions() {
7           return MISSIONS;
8       }
9   }
10
11  const MISSIONS = [
12      { id: 1, name: "홍길동" },
13      { id: 2, name: "백두산" },
14      { id: 3, name: "임꺽정" },
15  ];
16
```

서비스와 인젝터

- 서비스 클래스
  - 주로 HTTP 데이터 서비스용 클래스
  - 앵귤러 모듈에 등록 필요
    - 컴포넌트와 달리 providers 섹션에 등록

```
import { AppService } from './app.service';

@NgModule({
  declarations: [ AppComponent ],
  imports: [
    BrowserModule, FormsModule, HttpModule
  ],
  providers: [
    AppService // 서비스 클래스 등록
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- 컴포넌트 클래스
  - 서비스를 사용하는 클래스
  - 직접 데이터 서비스 코드를 구현해도 무관

의존성 주입(Dependency Injection)

- @Injectable() 데코레이터
  - Injectable 모듈 인클루드
    - import { Injectable } from '@angular/core';
  - 서비스 클래스 만들 때 사용

- 컴포넌트 클래스의 생성자에 private으로 주입

```
export class AppComponent {
  title = 'app works!';
  public courses = [];
  constructor(private appService: AppService) {
    this.courses = appService.getCourses();
  }
}
```

서비스 클래스 주입

- 생성자에 주입
  - 컴포넌트 클래스의 생성자에 서비스 클래스 주입

{provide: 클래스, useClass: 클래스}

- services.AddTransient<클래스, 클래스>(); 와 동일

- useClass
- useValue
- useExisting
- useFactory: fac()

부모 컴포넌트와 자식 컴포넌트

- 모든 컴포넌트는 자식 컴포넌트 포함
  - 손자 컴포넌트(?)

```
// 상단 메뉴
import { TopMenuComponent } from './layouts/_top-menu.component';
// 사이드바 헤더
import { SidebarHeaderComponent } from './layouts/_sidebar-header.component';
// 사이드 메뉴
import { AsideMenuComponent } from './layouts/_aside-menu.component';
```

MyRanking.component.ts

TypeScript 가상 프로젝트 | "MyRanking.component"

```
41      showSubClicked(msg) {
42          console.log(msg);
43      }
44  }
```

MyRanking.component.html

```
13          <td>
14              <TopRanking
15                  [rank]="r.rank"
16                  (eventEmitter)="showSubClicked($event)"></TopRanking>
17          </td>
18      </tr>
```

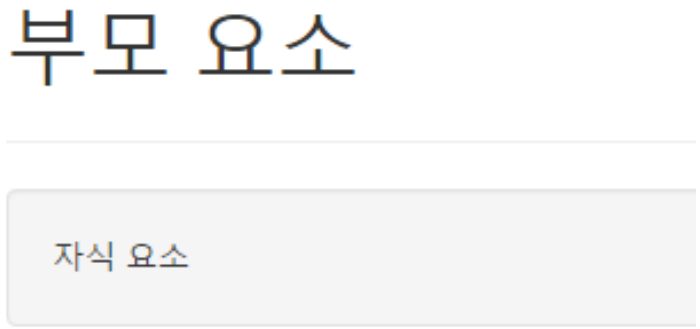
TopRanking.component.ts

TypeScript 가상 프로젝트 | TopRankingComponent

```
1   import { Component, Input, Output, EventEmitter } from '@angular/core';
2
3   @Component({
4       selector: 'TopRanking',
5       template: `
6   <div [ngSwitch]="rank" class="text-center" (click)="boxClick()">
7       <img src="/images/One.gif" *ngSwitchCase="1" />
8       <img src="/images/Two.gif" *ngSwitchCase="2" />
9       <img src="/images/Three.gif" *ngSwitchCase="3" />
10      <span *ngSwitchDefault>{{ rank }}</span>
11  </div>
12      `
13  })
14  export class TopRankingComponent {
15      @Input()
16      public rank: any;
17
18      @Output()
19      public eventEmitter = new EventEmitter();
20
21      public boxClick() {
22          this.eventEmitter.emit(this.rank + " 클릭됨"); // 부모에게 전달
23      }
24  }
```

Input 데코레이터와 Output 데코레이터

- @Input과 @Output
- 모듈 임포트
  - Input
    - 부모에서 자식으로
    - 컴포넌트에게 데이터 전달
  - Output
    - 자식에서 부모로
  - EventEmitter
    - 자식의 값을 부모에서 사용하도록
    - **새로운 이벤트를 정의**
- EventEmitter<T>
  - 제네릭으로 number, string 형 정해서 전달



@Output 데코레이터

- @Output pageChanged = new EventEmitter()

MemoEngine - OutputChild.html

OutputChild.ts

TypeScript 가상 프로젝트 | "OutputChild"

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

@Component({
    selector: "output-child",
    template: require('./OutputChild.html')
})
export class OutputChild {
    pages = [1, 2, 3];

    clickPage(page: number) {
        this.pageChanged.emit(page);
    }

    @Output()
    pageChanged: EventEmitter<number> = new EventEmitter();
}
```

121 %

OutputChild.html

```html

<h2>Output 데모 - 자식</h2>

<ul class="pagination">
    <li *ngFor="let page of pages" (click)="clickPage(page)">
        <a>{{page}}</a>
    </li>
</ul>
```

MemoEngine - OutputParent.ts

OutputParent.ts

TypeScript 가상 프로젝트 | "OutputParent"

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

@Component({
    selector: "output-parent",
    template: require('./OutputParent.html')
})
export class OutputParent {
    page: number;
    showPage(page: number) {
        this.page = page;
    }
}

```

121 %

OutputParent.html

```html

<h1>Output 데모 - 부모</h1>

{{page}}

<hr />

<output-child (pageChanged)="showPage($event)"></output-child>

```

자식요소의 #id를 사용하여 접근

- #id
  - 템플릿 참조 변수
- <부모템플릿>
- 　　<자식요소 #id></자식요소>
- 　　{{id.자식요소의속성}}
- </부모템플릿>

자식 컴포넌트에도 *ngFor 사용 가능

- <자식 *ngFor="let note of notes" [note]="note" />

라우팅

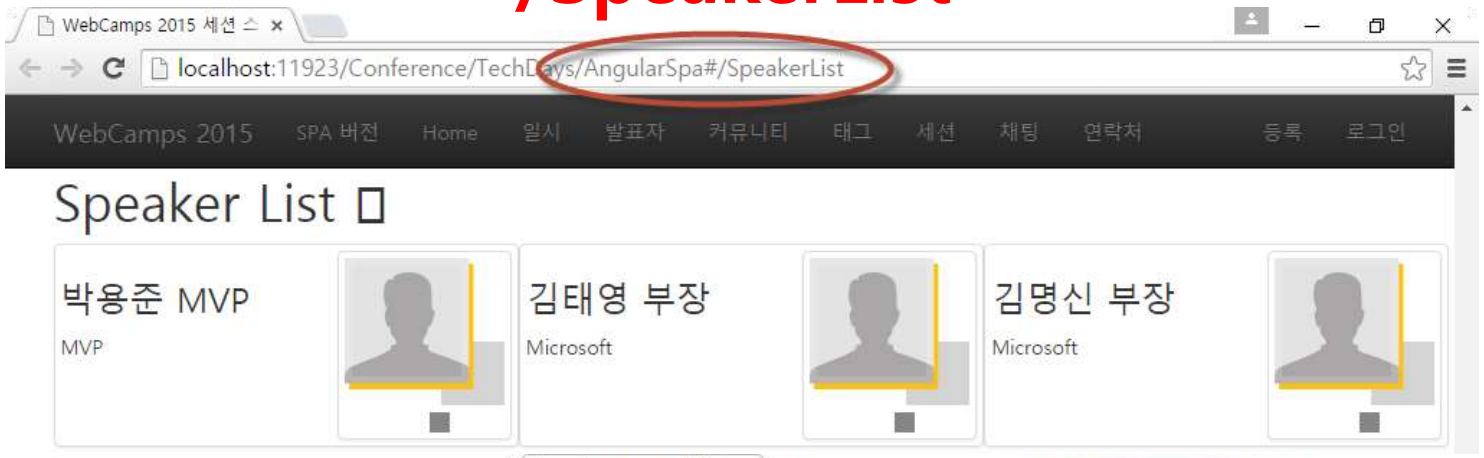
<router-outlet>

@RenderBody()

<ng-view />

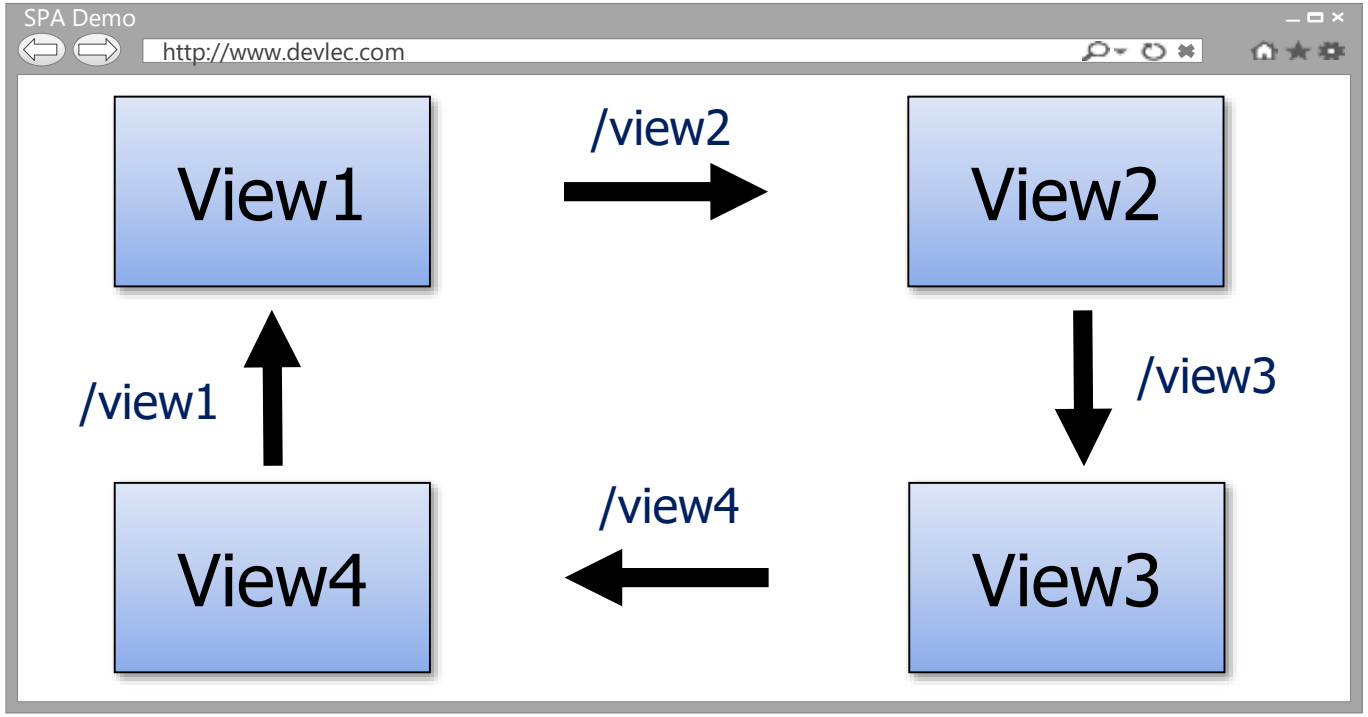
라우팅 의미

#/SpeakerList



#/SpeakerDetails/0

라우트(Route)의 역할



컨텐츠 영역만 로딩

라우팅, 라우트

- 앵귤러 앱에서 페이지 이동 관련 기능의 집합체
- 라우팅은 컴포넌트 기반

- BoardList 에서 BoardView?id=1234 식으로 이동
  - 콜론 기호를 사용하여 파라미터 지정
- 라우트 파라미터 가져오기
- 코드로 페이지 이동
- #(해시) 스타일 라우팅 가능

라우터/라우팅

- 메인 뷰 페이지에
  - <base href="/"> 적용
    - Index.htmI 페이지가 루트 페이지임을 확인
- 라우팅 적용 순서
  - RouterModule 포함
    - {useHash:true}
  - @angular/router
  - 라우터 정의
  - <router-outlet> 지정
    - 모듈에 RouterModule imports 필요
  - [routerLink] 디렉티브로 링크 지정
  - [routerLinkActive] 디렉티브로 CSS 하이라이트 주기

```ts
import { RouterModule } from '@angular/router'; // 모듈 추가

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    AboutComponent,
    ContactComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    RouterModule.forRoot([
      { path: '', redirectTo: 'home', pathMatch: 'full' },

      { path: 'home', component: HomeComponent },
      { path: 'about', component: AboutComponent },
      { path: 'contact', component: ContactComponent },

      { path: '**', redirectTo:'home' }
    ])
  ],
```

```html
<a [routerLink]="['/home']">home</a>
<a [routerLink]="['/about']">about</a>
<a [routerLink]="['/contact']">contact</a>
<hr />

<h1>
  {{title}}
</h1>

<div class='container'>
  <router-outlet></router-outlet>
</div>
```

Angular 라우터

app.module.ts에 라우터 모듈 등록

```
import { RouterModule } from '@angular/router';
```

```
declarations: [
    AppComponent,
    NavMenuComponent,
    CounterComponent,
    FetchDataComponent,
    HomeComponent,
    RoutingList, RoutingView // 라우팅 데모 관련
],
```

```
import { RouterModule } from '@angular/router';

//[!] RoutingDemo 관련 컴포넌트 등록
import { RoutingList }
    from './components/routingdemo/routing-list.component';
import { RoutingView }
    from './components/routingdemo/routing-view.component';
```

```
// 라우팅 데모 관련
{ path: 'routinglist', component: RoutingList },
{ path: 'routingview/:id', component: RoutingView },
```

라우팅

- app.module.ts

```
RouterModule.forRoot([
    { path: '', redirectTo: 'home', pathMatch: 'full' },
    { path: 'home', component: HomeComponent },
    { path: 'about', component: AboutComponent },
    { path: 'contact', component: ContactComponent },
    { path: 'counter', component: CounterComponent },
    { path: 'fetch-data', component: FetchDataComponent },
    { path: '**', redirectTo: 'home' }
])
```

기본 라우트

기타: 404 방지

pathMatch: ‘full’ 또는 ‘prefix’

- ‘full’
  - 경로 매치되는 경로로 이동
- ‘prefix‘
  - 특정 URL로 시작하는 경로로 이동

[routerLink]로 이동

- [routerLink]=“”
  - 라우터에 등록된 링크로 이동

```
<ul>
    <li><a href="/">Home</a></li>
    <li><a [routerLink]="['/home']">Home</a></li>
    <li><a [routerLink]="['/routingview', 1]">상세보기 1</a></li>
    <li><a [routerLink]="['/routingview', 1234]">상세보기 1234</a></li>
</ul>
```

/home

```
<li [routerLinkActive]="['link-active']">
    <a [routerLink]="['/home']">
        <span class='glyphicon glyphicon-home'></span>
        Home
    </a>
</li>
<li [routerLinkActive]="['link-active']">
    <a [routerLink]="['/about']">
        <span class='glyphicon glyphicon-home'></span>
        About
    </a>
</li>
<li [routerLinkActive]="['link-active']">
    <a [routerLink]="['/contact']">
        <span class='glyphicon glyphicon-home'></span>
        Contact
    </a>
</li>
```

RouteOutlet 지시자

- 라우터 컴포넌트 뷰가 출력될 위치 설정
- 특정 라우팅에 해당하는 컴포넌트가 실행될 위치
  - <router-outlet></router-outlet>
    - Angular 1.X의 <ng-view>와 동일
    - ASP.NET Core의 RenderBody()와 동일

라우트 파라미터

- <a [routerLink]="['/BoardView', id]">{{ title }}

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
@Component(...)
export class HomeDetailComponent implements OnInit {
    constructor(private route: ActivatedRoute, private router: Router) { }
    id: number;
    //[1] 페이지 로드시 넘어온 쿼리스트링 파라미터 받기
    ngOnInit() {
        this.id = +this.route.snapshot.params["id"];
    }
    //[2] 뒤로가기
    goBack(): void {
        this.router.navigate(['/home']);
    }
}
```

라우터 파라미터

- 콜론(:) 기호를 사용하여 파라미터 정의
- { path: ‘courses’ }
- { path: ‘courses/:id’ }
  - :id
    - 플레이스 홀더
    - 라우트 매개 변수

import { Router } from @angular/router

- constructor(private router:Router) {}

- this.router.navigate(['/home'])
  - /home 경로로 이동
  - this.router.navigate('/home');
    - 위 형태도 사용 가능
- this.router.navigate(['BlogPost'])

- this.router.navigateByUrl('/home');

ActivatedRoute: 라우터 파라미터

- +this.route.snapshot.params['id']

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

@Component({
    selector: 'routing-view',
    template: `
<h1>상세보기</h1>
<p>넘어온 매개변수 id값: {{id}}</p>
    `
})
export class RoutingView implements OnInit {
    id: number;

    constructor(private route: ActivatedRoute) { }

    // 페이지 로드
    ngOnInit() {
        this.id = +this.route.snapshot.params["id"];
    }
}
```

Router

```
import { Component, OnInit } from '@angular/core';
// ActivatedRoute 서비스 클래스
import { ActivatedRoute, Router } from '@angular/router';

@Component({
    selector: 'routing-view',
    template: `
<h1>상세보기</h1>
<p>넘어온 매개변수 id값: {{id}}</p>
<button (click)="goBack()">뒤로</button>
    `
})
export class RoutingView implements OnInit {
    id: number;
    constructor(private route: ActivatedRoute
        , private router: Router) { }
    // 페이지 로드
    ngOnInit() {
        this.id = +this.route.snapshot.params["id"];
    }
    goBack(): void { this.router.navigate(['/routinglist']); }
}
```

라우팅을 통한 데이터 전달

- Snapshot
  - ActivatedRoute 주입
  - this.router.snapshot.params[‘id’]
- Observable
  - ActivatedRoute 주입
  - this.route.params.map(params => params[‘id’]).do().subscribe();
    - ngOnInit() { this.route.params.map(params => parmas['id']).do(id => this.id = parsetInt(id)).subscribe(…); }
- Resolvers
  - 컴포넌트가 로드하기 전에 가져오기
  - 라우트에 resolve: {} 섹션 구현

라우팅 가드(Guard)

- Resolve
- CanActivate
- CanDeactivate
- CanActivateChild
- CanLoad
  - 비동기 라우팅 금지

CanActivate

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, Router }
    from '@angular/router';

@Injectable()
export class RoutingViewGuard implements CanActivate {
    constructor(private router: Router) { }

    canActivate(route: ActivatedRouteSnapshot): boolean {
        let id = +route.url[1].path;

        if (isNaN(id)) {
            this.router.navigate(['/routinglist']);
            return false;
        }

        return true;
    }
}
```

자식 라우트(Child Route)

- children 섹션
- forChild()

```
const routes: Routes = [
  { path: '', component: DashboardComponent, data: { title: '대시보드' }},
  { path: 'dashboardother', component: DashboardOtherComponent, data: { title: '대시보드 참고' }},
  { path: 'missiondetail/:id', component: DashboardMainMissionDetailComponent, data: { title: '미션' }}
];

@NgModule({
  imports: [RouterModule.forChild(routes)], // app.module.ts와 다르게 forChild() 메서드로 자식 요소 등록
  exports: [RouterModule] // 어떤 모듈이 사용되는지
})
export class DashboardRoutingModule {}
```

라우트 링크 스타일 적용

- <a [routeLinke]="" routerLinkActive="active">...
  - routerLinkActive="스타일"

- routerLinkeActiveOptions="{exact:true}"
  - 정확히 매치되는 라우터 경로만 스타일 주기

```
<a class="nav-link"
    routerLinkActive="active"
    [routerLink]="['/dashboard']"
    [routerLinkActiveOptions]='{exact:true}'>
        <i class="icon-speedometer"></i> 대시보드
        <span class="badge badge-info">HOME</span>
</a>
```

앵귤러 폼

템플릿 기반 폼과 모델 기반 폼

- 템플릿 기반 폼
  - <input name="txtNote" ngModel /> // 바인딩 없음
  - <input name="note" [ngModel]="note" /> // 단방향
  - <input name="note" [ngModel]="note" (ngModelChange)="note=$event" /> // 양방향
    - $event : 텍스트박스의 값
  - <input name="note" [(ngModel)]="note" />

- 모델 기반 폼
  - 리액티브 폼

폼과 컴포넌트

- 폼(템플릿)
  - 폼 요소
  - 입력 요소
  - 데이터 바인딩
  - 유효성 검사

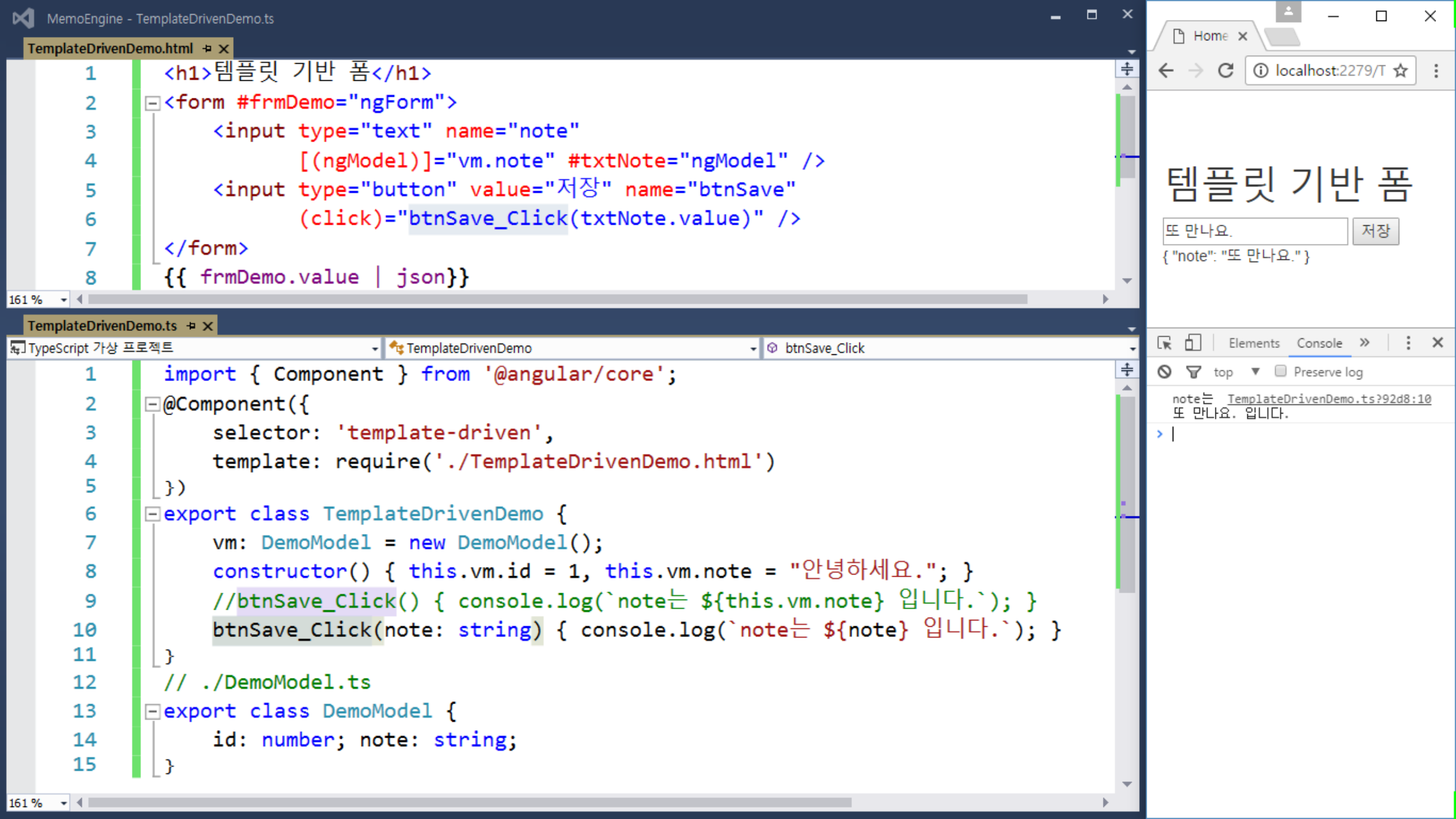
- 컴포넌트(코드)
  - 속성
    - 데이터 바인딩
  - 메서드
    - 전송된 값 받기

폼 상태를 나타내는 속성들

값 변경	유효성 검사	컨트롤 선택(터치)
• pristine • dirty	• valid • invalid • errors	• touched • untouched

템플릿 기반 폼 관련 지시자(Directive)

- FormsModule(FormGroup, FormControl)
  - ngForm
  - ngModel
  - ngModelGroup
  - ngSubmit
- # 기호를 사용하여 템플릿에서 폼의 구성요소 참조 가능
  - #frm
  - #frmRegister
  - #txtName, #txtEmail


MemoEngine - TemplateDrivenDemo.ts
TemplateDrivenDemo.html
<h1>템플릿 기반 폼</h1>
<form #frmDemo="ngForm">
<input type="text" name="note"
[(ngModel)]="vm.note" #txtNote="ngModel" />
<input type="button" value="저장" name="btnSave"
(click)="btnSave_Click(txtNote.value)" />
</form>
{{ frmDemo.value | json}}
161 %
TemplateDrivenDemo.ts
TypeScript 가상 프로젝트
TemplateDrivenDemo
btnSave_Click
import { Component } from '@angular/core';
@Component({
selector: 'template-driven',
template: require('./TemplateDrivenDemo.html')
})
export class TemplateDrivenDemo {
vm: DemoModel = new DemoModel();
constructor() { this.vm.id = 1, this.vm.note = "안녕하세요."; }
//btnSave_Click() { console.log(`note는 ${this.vm.note} 입니다.`); }
btnSave_Click(note: string) { console.log(`note는 ${note} 입니다.`); }
}
// ./DemoModel.ts
export class DemoModel {
id: number; note: string;
}
161 %
Home
localhost:2279/T
템플릿 기반 폼
또 만나요.
저장
{ "note": "또 만나요." }
Elements
Console
top
Preserve log
note는 TemplateDrivenDemo.ts?92d8:10
또 만나요. 입니다.

<input [value]="postForm.title" />

- 약간의 양방향 바인딩? 아래 2개를 함께 주기
  - [value]="postForm.title"
  - (input) = "postForm.title = $event.target.value"

- (input)
  - (input)="txtUsername=$event.target.value"
  - 텍스트박스에 값이 입력(input)되면 이벤트가 발생하여 지정된 속성(txtUsername)의 값을 업데이트
  - (input) 방식은 사용하기 복잡해서 [(ngModel)]을 더 추천

<select #lst (evt)="color(lst.value)" />

- #lst => changeColor(lst.value)
  - -또는-
- changeColor($event.target.value)

selectoption.component.html

```
1  <h1>드롭다운리스트 데모</h1>
2  <select #ctl (change)="printSelect(ctl.value)">
3      <option>Angular</option>
4      <option>ASP.NET Core</option>
5      <option>Azure</option>
6  </select>
7  {{tech}}
8  <hr />
9  <select (change)="printWeb($event.target.value)">
10     <option>Angular</option>
11     <option>ASP.NET Core</option>
12     <option>Azure</option>
13 </select>
14 {{web}}
```

selectoption.component.ts

TypeScript 가상 프로젝트 | {} "selectoption.component"

```
1  import { Component } from '@angular/core';
2
3  @Component({
4      selector: "selectoption",
5      template: require('./selectoption.component.html')
6  })
7  export class SelectOptionComponent {
8      tech = "-";
9      printSelect(sv) {
10         this.tech = sv;
11     }
12     web = '-';
13     printWeb(sv) {
14         this.web = sv;
15     }
16 }
17
```

<select><option>

- <select [(ngModel)]="note.encoding" name="encoding">
  - <option [ngValue]="note.id" *ngFor="let n of notes">

pristine과 dirty: {{ form.pristine }}

- #form.pristine
  - 폼이 처음 상태이면 true
  - 폼의 상태 또는 모델(ngModel) 내용이 변경되면 false

TextBoxName.className

- 현재 개체에 지정된 CSS class 속성 리스트 출력

라디오 버튼 선택

```
<div class="radio">
    <label>
        <input type="radio" name="isPinned"
               value="Yes"
               [(ngModel)]="vm.isPinned"
               >
        공지글 올리기
    </label>
</div>
<div class="radio">
    <label>
        <input type="radio" name="isPinned"
               value="No"
               [(ngModel)]="vm.isPinned"
               >
        공지글 내리기
    </label>
</div>
```

```
// 게시판 글 모델 클래스
export class Note {
    constructor(
        public name: string,
        public title: string,
        public content: string,
        public isRemember: boolean,
        public isPinned: string
    ) {
    }
}
```

```
public vm = new Note("이름", "제목", "내용", true, "No");
```

ngModel 속성들

- ng-untouched
- ng-touched
  - <div *ngIf="txtName.invalid && txtName.touched" ...
- ng-pristine : 폼이 처음 상태이면 true
- ng-dirty : 폼의 내용이 변경된 상태이면 true
  - {{ txt.dirty }}
- ng-valid : 유효성을 통과하였으면 true
- ng-invalid : 유효성을 통과하지 않았으면 true

MemoEngine - user.component.html

user.component.ts

TypeScript 가상 프로젝트 | "user.component" | UserComponent

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

@Component({
    selector: "user", template: require('./user.component.html'),
    styles: [`
        span { color: tomato; }
    `]
})
export class UserComponent {
    btnLogin_Click(args) {
        console.log(args); // OK
        console.log(JSON.stringify(args)); // JSON => POST 방식 사용

        console.log(args.username);
        console.log(args.password);
    }
}
```

100 %

user.component.html

```html
<h1>로그인</h1>
<hr />
<div class="col-md-4">
    <form autocomplete="off" #frmLogin="ngForm" novalidate
          (ngSubmit)="btnLogin_Click(frmLogin.value)">
        <div class="form-group">
            <label for="username">아이디: </label>
            <span *ngIf="frmLogin.controls.username?.touched
                  && frmLogin.controls.username?.invalid">
                * 필수
            </span>
            <input type="text" id="username" name="username"
                   [(ngModel)]="username" required
                   placeholder="아이디..." class="form-control" />
        </div>
        <div class="form-group">
            <label for="password">암호: </label>
            <span *ngIf="pwd.touched && pwd.invalid">
                * 필수
            </span>
            <input type="password" id="password" name="password"
                   #pwd="ngModel"
                   [(ngModel)]="password" required
                   placeholder="암호..." class="form-control" />
        </div>
        <button type="submit" class="btn btn-primary"
                [disabled]="frmLogin.invalid">
            로그인
        </button>
        <button type="button" class="btn btn-default">취소</button>
    </form>
    <hr />
    {{ frmLogin.value | json }}
</div>
```

100 %

[class.CSS클래스명]=“true”

- [class.has-error]=“true”

write.component.html

```
<h1>입력</h1>

<div class="container">
    <div class="row">
        <div class="col-md-12">
            <form class="form-horizontal" role="form" #frm="ngForm">
                <div class="form-group">
                    <label class="col-sm-2 control-label" for="title">문제</label>
                    <div class="col-sm-10">
                        <input type="text" class="form-control" id="title" placeholder="문제"
                               name="title" [(ngModel)]="model.title">
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-sm-offset-2 col-sm-10">
                        <div class="checkbox">
                            <label>
                                <input type="checkbox"
                                       name="isAgree" [(ngModel)]="isAgree"> 입력하시겠습니까?
                            </label>
                        </div>
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-sm-offset-2 col-sm-10">
                        <button type="submit" class="btn btn-primary" name="btnSave"
                                (click)="btnSave_Click(frm.value)">
                            저장
                        </button>
                        <a [routerLink]="['/question']" class="btn btn-default">취소</a>
                    </div>
                </div>
            </form>
            <hr />
            {{ frm.value | json }}
        </div>
    </div>
</div>
```

write.component.ts

```
import { Component, OnInit } from '@angular/core';
import { QuestionDataService } from './dataservice';
import { IQuestion, QuestionModel } from './model';
import { Router } from '@angular/router';

@Component({
    selector: 'question-write',
    template: require('./write.component.html')
})
export class QuestionWriteComponent implements OnInit {
    model: QuestionModel = new QuestionModel(0, "");
    // 동의 체크박스
    isAgree: boolean = false;
    constructor(private ds: QuestionDataService, private router: Router) { }
    ngOnInit() { }
    // 저장(전송) 버튼 클릭
    btnSave_Click(frmValue) {
        if (this.isAgree === true) {
            this.ds.add(frmValue as IQuestion)
                .subscribe(
                // 성공
                (model: IQuestion) => {
                    if (model) {
                        console.log("저장 완료..." + JSON.stringify(model));
                        // 저장 후 리스트 페이지로 이동
                        // this.router.navigate(['/four']);
                    }
                    else {
                        console.log("저장되지 않음...");
                    }
                },
                // 에러
                (err: any) => console.log(err),
                () => { this.router.navigate(['/question']); }
                );
        }
        else {
            console.log("체크박스에 체크해야만 저장이 됩니다.");
        }
    }
}
```

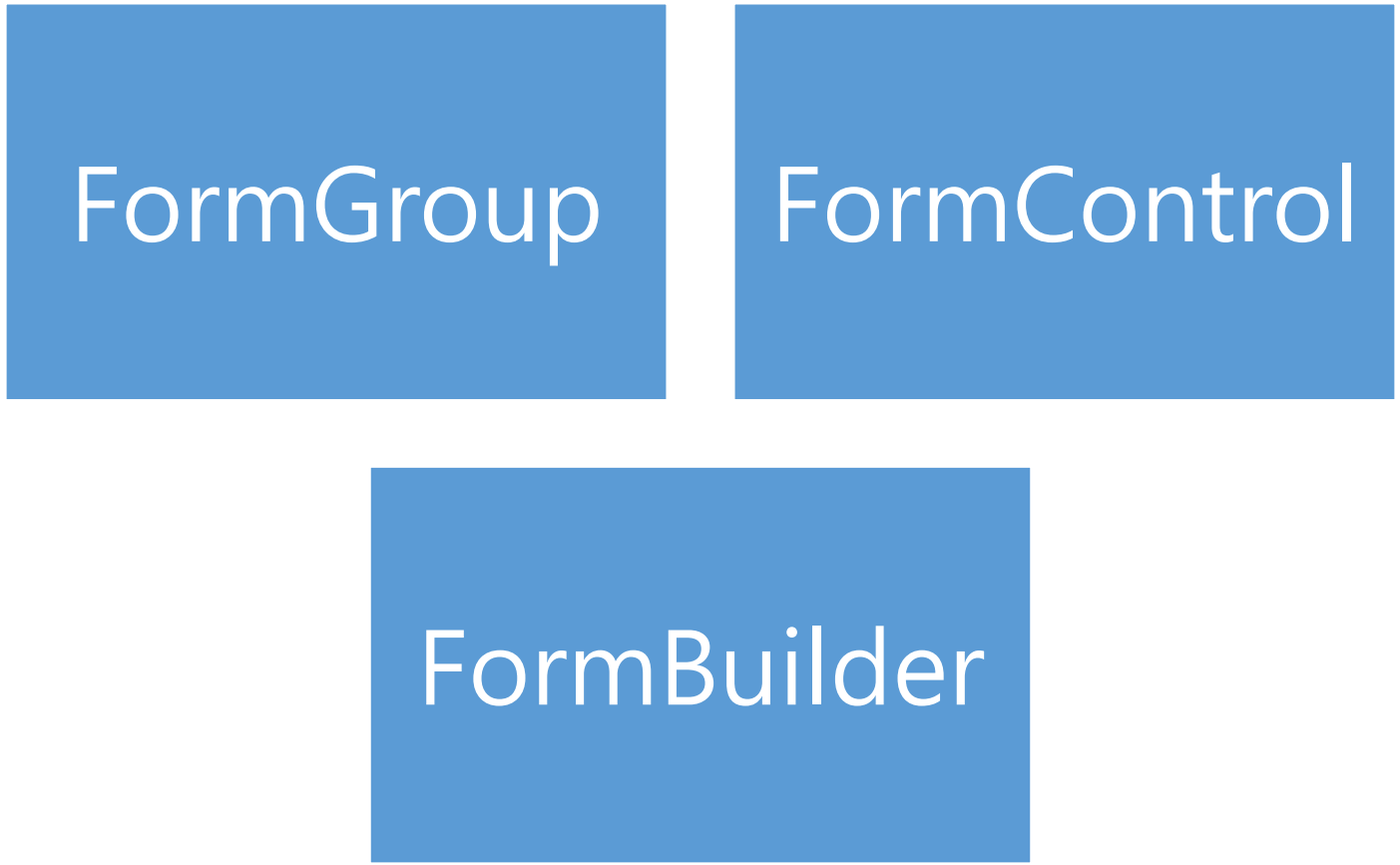
리액티브 폼

리액티브 폼

- 컴포넌트(코드)
  - 코드에서 폼 관련 요소 생성
  - 폼 모델
  - 유효성 검사 규칙
  - 속성에 데이터 바인딩
  - 메서드 생성(전송되는 데이터 가져오기)

- 폼(템플릿)
  - 폼 요소
  - 입력 요소
  - 컴포넌트에서 생성한 개체 바인딩

리액티브 폼 관련 개체



리액티브 폼

```
<form [formGroup]="editForm" (ngSubmit)="save($event)">
    <input type="text" formControlName="id" />
    <input type="text" formControlName="title" />
    <input type="submit" name="btnEdit" value="수정" />
</form>
```

```
import { FormBuilder, FormGroup, Validators, FormControl } from '@angular/forms'; //

@Component(...)
export class QuestionEditReactiveComponent implements OnInit {
    id: number;
    model: IQuestion = { id: 0, title: '' }; //[!] 기본값으로 초기화

    public editForm: FormGroup; //[!]

    constructor(private route: ActivatedRoute,
        private ds: QuestionDataService, private router: Router,
        public fb: FormBuilder) {
    }

    ngOnInit() {
        let id = +this.route.snapshot.params["id"]; // id 매개변수 받기
        this.id = id;
        this.getData(id);
        //[!]
        this.formCreate(); // 페이지 로드시 FormGroup 생성
    }

    //[!]
    formCreate() {
        this.editForm = this.fb.group({
            id: [this.model.id, Validators.required],
            title: [this.model.title, Validators.required]
        });
        //console.log("값" + this.editForm.get("title").value);
    }
```

코드에서 FormGroup 외 참조

- import { FormGroup } from '@angular/forms';

- FormModel
  - 상태, 값, 컨트롤(자식 컨트롤 포함)들 포함
  - Form Group
    - FormControl

템플릿의 리액티브 폼 관련 지시자

- ReactiveFormsModule
  - formGroup
    - 최상위 폼 그룹 지시자
  - formControlName
    - 각 컨트롤
  - formControl
  - formGroupName
    - 여러 컨트롤을 묶어서 관리
      - frm.controls.groupName.controls.ctlName.touched
      - frm.get('groupName.ctlName').valid
  - formArrayName

- [formGroup]=''과 formControlName=''

폼 요소 접근: frm.controls.txtName.valid

- 폼의 모델에 접근하는 방식 2가지
  - frm.controls.txtName.valid
  - frm.get('txtName').valid

리액티브 폼의 내용 수정: patchValue()

- 폼.patchValue()
  - 업데이트할 요소만 업데이트
  - 폼.setValue()
    - 모든 폼의 요소가 필요

FormBuilder

- 폼 모델 생성
  - 폼 빌더 개체 임포트
  - 생성자 매개변수로 폼 빌더 주입
  - 폼빌더.group({})으로 폼 모델 생성

Validators: 리액티브 폼 관련 유효성 검사

- https://angular.io/ Docs 참조
  - Validators.required
  - Validators.pattern('')
  - Validators.minLength(*n*)

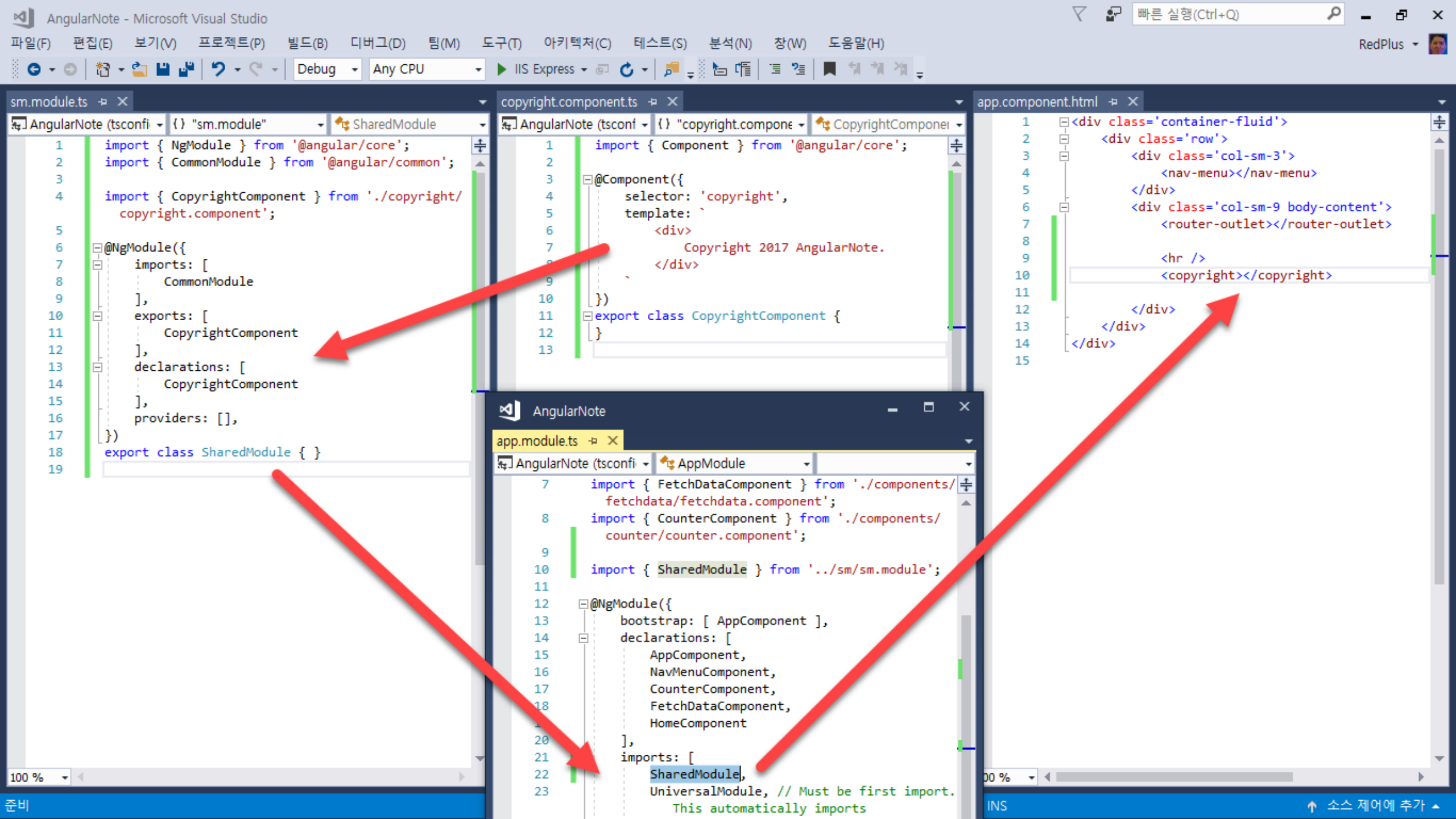
setValidators(), clearValidator()

- ctl.setValidators()
- ctl.clearValidators()
- ctl.updateValueAndValidity()

앵귤러 모듈

앵귤러 모듈

- NgModule 데코레이터가 사용된 클래스
- 큰 프로젝트를 모듈 단위로 나누어 관리
  - Feature 모듈
- Declarations
  - 컴포넌트


AngularNote - Microsoft Visual Studio
빠른 실행(Ctrl+Q)
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 도구(T) 아키텍처(C) 테스트(S) 분석(N) 창(W) 도움말(H)
RedPlus
Debug
Any CPU
IIS Express
sm.module.ts
AngularNote (tsconfi
"sm.module"
SharedModule
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { CopyrightComponent } from './copyright/
copyright.component';
@NgModule({
imports: [
CommonModule
],
exports: [
CopyrightComponent
],
declarations: [
CopyrightComponent
],
providers: [],
})
export class SharedModule { }
copyright.component.ts
AngularNote (tsconf
"copyright.compone
CopyrightCompone
import { Component } from '@angular/core';
@Component({
selector: 'copyright',
template: `

Copyright 2017 AngularNote.

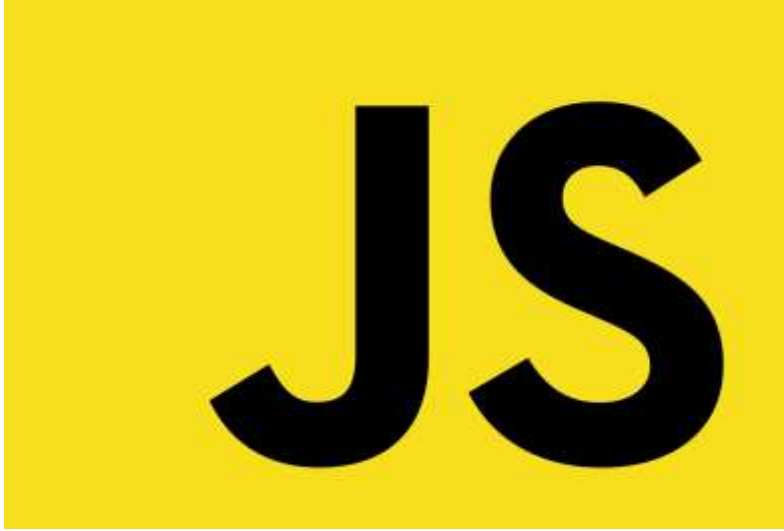
`
})
export class CopyrightComponent {
}
app.component.html

<nav-menu></nav-menu>

<router-outlet></router-outlet>
<hr />
<copyright></copyright>

AngularNote
app.module.ts
AngularNote (tsconfi
AppModule
import { FetchDataComponent } from './components/
fetchdata/fetchdata.component';
import { CounterComponent } from './components/
counter/counter.component';
import { SharedModule } from '../sm/sm.module';
@NgModule({
bootstrap: [AppComponent],
declarations: [
AppComponent,
NavMenuComponent,
CounterComponent,
FetchDataComponent,
HomeComponent
],
imports: [
SharedModule,
UniversalModule, // Must be first import.
This automatically imports
100 %
준비
INS
소스 제어에 추가

ES6 모듈



math.ts

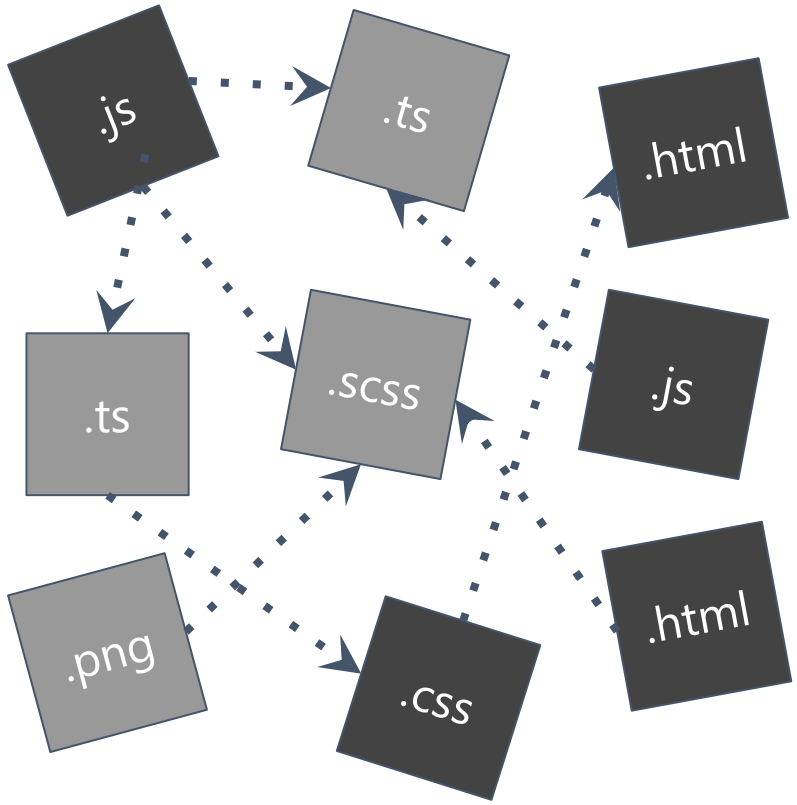
```
export function sum(x, y)
  return x + y;
}
```

calculator.ts

```
import {sum} from './math';

sum(1, 2);
```

앵귤러 모듈: 컴포넌트, 서비스, 모듈

```
@NgModule({
      declarations:[HomeComponent],
      imports:     [FormsModule],
      providers:[HomeService],
})
export class HomeModule { }
```



의존성 그래프

모듈
번들러

모듈

- 모듈
  - 네임스페이스
  - 코드 조직화
  - ES 2015 모듈
    - export 키워드를 붙인 클래스
      - export class Note {}
    - import 구문으로 포함시켜 사용
      - import { Note } from ‘./Note’

Angular 시작하기

Angular 1

- <html ng-app="app">

Angular 2

- import { bootstrap } from 'angular2/...';
- import { MyComponent } from './app';

- bootstrap(MyComponent);

Angular 2 구성

- 응용 프로그램 = 컴포넌트들과 서비스들의 집합
- 컴포넌트 = 템플릿 + 클래스(멤버들) + 메타데이터

- 템플릿
  - HTML 영역, 바인딩, 지시자
- 클래스
  - 생성자, 속성, 메서드
- 메타데이터
  - 데코레이터, 특성

모듈(Modules)

- Export
  - member.ts
    - export class Member {}
- Import
  - member-list.ts
    - import { Member } from ‘./member’
      - 확장자 필요없음

Lazy Loading과 Preload

- loadChildren

- data: { preload: true }

라우팅 고급

라우팅 기본

- <base href="/">

- <a [routerLink]="['/home']">Home
  - <a [routerLink]="['/home', this.id, 'list']">Home

RouterModule.forRoot()

- 앵귤러 앱에 단 하나 존재
- RouterModule.forChild()
  - Feature 모듈 당 하나씩 구현

브라우저 URL 스타일

- 기본 스타일
- 해시 스타일
  - /#/
  - RouterModule.forRoot(, { useHash: true })

라우팅 모듈로 분리

- <a [routerLink]="['/board', 1234, 'edit']">게시물 수정

쿼리 파라미터

라우트 리졸버(Route Resolver)

- 라우트 리졸버 서비스 생성 및 등록
- 라우트 설정에 리졸버 추가
- ActivatedRoute로부터 데이터 읽기

라우트 리졸버 === 서비스 클래스

```
import { Injectable } from '@angular/core';
import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs/Observable';

import { IHero } from './model';

@Injectable()
export class HeroesResolverService implements Resolve<IHero> {
    resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<IHero> {

        return null;
    }
}
```

```typescript
import { Injectable } from '@angular/core';
import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/observable/of';
import 'rxjs/add/operator/map';
import { IQuestion } from './model';
import { QuestionDataService } from './dataservice';

@Injectable()
export class QuestionsResolverService implements Resolve<IQuestion> {
    constructor(private ds: QuestionDataService, private router: Router) { }
    resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<IQuestion> {
        let id = route.params["id"];

        // 예외 처리
        if (isNaN(id)) {
            console.log(`문제 번호는 숫자값이어야 합니다. : ${id}`);
            this.router.navigate(['/question']);
            return Observable.of(null);
        }

        return this.ds.getById(+id)
            .map(question => {
                if (question) {
                    return question;
                }
                console.log(`문제를 찾을 수 없습니다. : ${id}`);
                this.router.navigate(['/question']);
                return null;
            }).catch(error => {
                console.log(`에러가 발생했습니다. : ${error}`);
                this.router.navigate(['/question']);
                return Observable.of(null);
            });
    }
}
```

라우트 모듈에 라우트 리졸버 적용

- resolve: { question: QuestionResolverService }

라우트 리졸버를 사용하는 컴포넌트

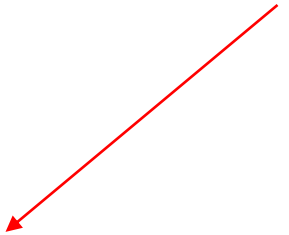
```
ngOnInit() {
    let id = this.route.snapshot.params["id"]; // id 매개변수 받기
    this.id = +id;

    //[!] 라우트 리졸버 서비스 클래스 사용하기
    //this.model = this.route.snapshot.data["hero"];
    this.route.data.subscribe((data) => {
        this.model = data["hero"];
    });
}
```

이름이 있는 <router-outlet>

- <router-outlet name="aside">

파이프(Pipe)

- 출력 전에 속성 변경
- 많은 수의 내장 파이프 제공



```
<pre>value: {{ postForm.value | json }}</pre>
```

Pipe 예제

- {{ board.title | uppercase }}
  - uppercase
    - 대문자
  - lowercase
  - date
    - date:"yMMd"
    - date: 'medium'
    - date: 'yMMMd'
  - number
  - decimal
  - percent
  - currency
    - currency:'USD':true
  - json
  - slice

날짜형식

- {{note?.postdate | date:'yyyy-MM-dd'}}

import { Pipe, PipeTransform }

- '@angular/core'

@Pipe 데코레이터

사용자 정의 파이프

- @Pipe 데코레이터
- transform(value, args)

컴포넌트 스타일 캡슐화

- 해당 컴포넌트에서만 사용되는 스타일 필요
  - 인라인 스타일
  - 외부 스타일
- @Component 데코레이터에 styles와 styleUrls를 사용해서 스타일 적용
  - styles: []
    - 내부에 직접 정의
  - styleUrls: []
    - 외부 css 파일에 정의

```
@Component({
  moduleId: module.id,
  selector: 'my-heroes',
  templateUrl: 'heroes.component.html',
  styleUrls: ['heroes.component.css']
})
export class HeroesComponent implements OnInit {
```

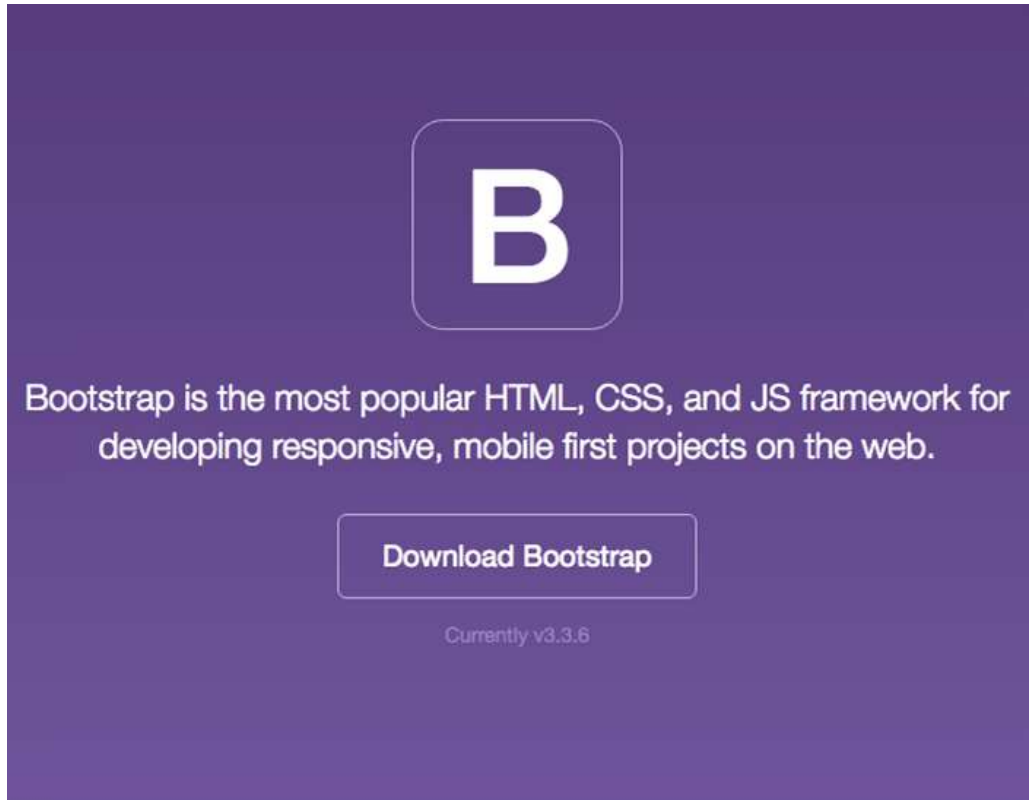
styles: []

- styles: ['li { cursor: pointer; }']


Ultimate Web UI Elements
RADIO BUTTONS
TOGGLE SWITCHES
CHECK BOXES
TABS & ACCORDIANS
Disabled
Enabled
Lorem ipsum dolor sit amet
TAB 01
TAB 02
TAB 03
BUTTONS
Normal
Hover
Active
FORM FIELDS
Search
Ultimate Web
Ultimate Web Graphics
Ultimate Web UI Elements
Ultimate Web User Interface
Normal Input Field
Normal dropdown Field
BUTTONS
Danger | Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Success | Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Warning | Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Info | Lorem ipsum dolor sit amet, consectetur adipiscing elit.
PROGRESS BARS
54 %
60 %
LOGIN FORM
User Name
ERROR
AUDIO PLAYER
01:50:45
VIDEO PLAYER
Good UX requires great components
Once you choose a framework, it's hard to change

Material 2

Bootstrap



뷰 캡슐화 모드

- ViewEncapsulation
  - None
  - Native

커스텀 파이프

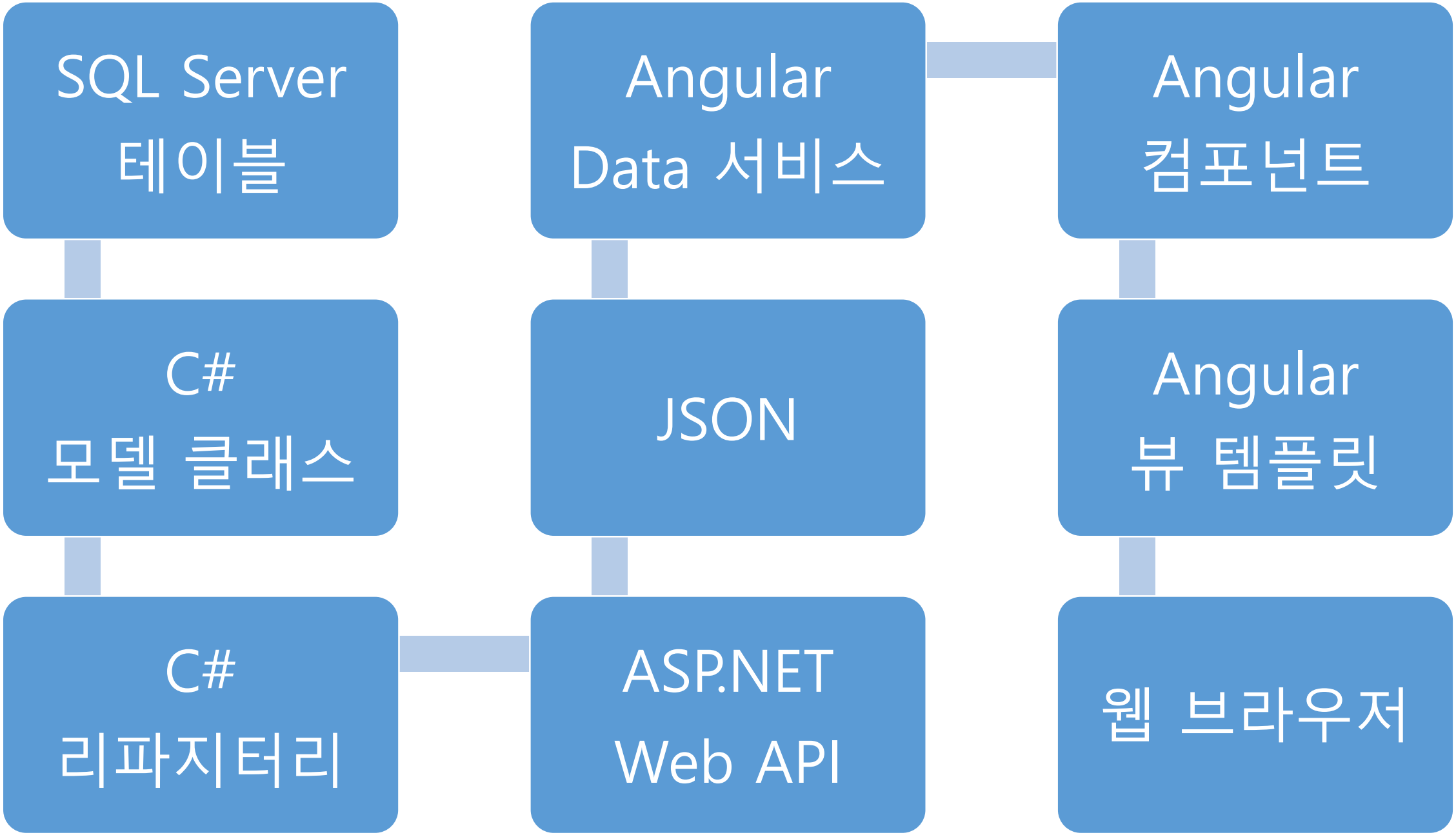
- 직접 특정 기능을 사용자 정의
- PipeTransform
  - transform()
- Pipe
- @Pipe()

<ng-content>

- 부모에서 던져 준 내용을 포함하는 영역
  - 여러 개 사용 가능
    - <ng-content select=“.title” />
      - class=”title”이 적용된 div 영역
    - <ng-content select=“.content” />
    - <ng-content select=‘[attr이름]’ />

@ViewChild와 @ViewChildren

- 부모 컴포넌트에서 자식 컴포넌트의 멤버 호출
  - #id로 지정된 개체를 ViewChild 데코레이터에 지정
  - 자식 컴포넌트의 특정 메서드를 부모 컴포넌트에서 호출

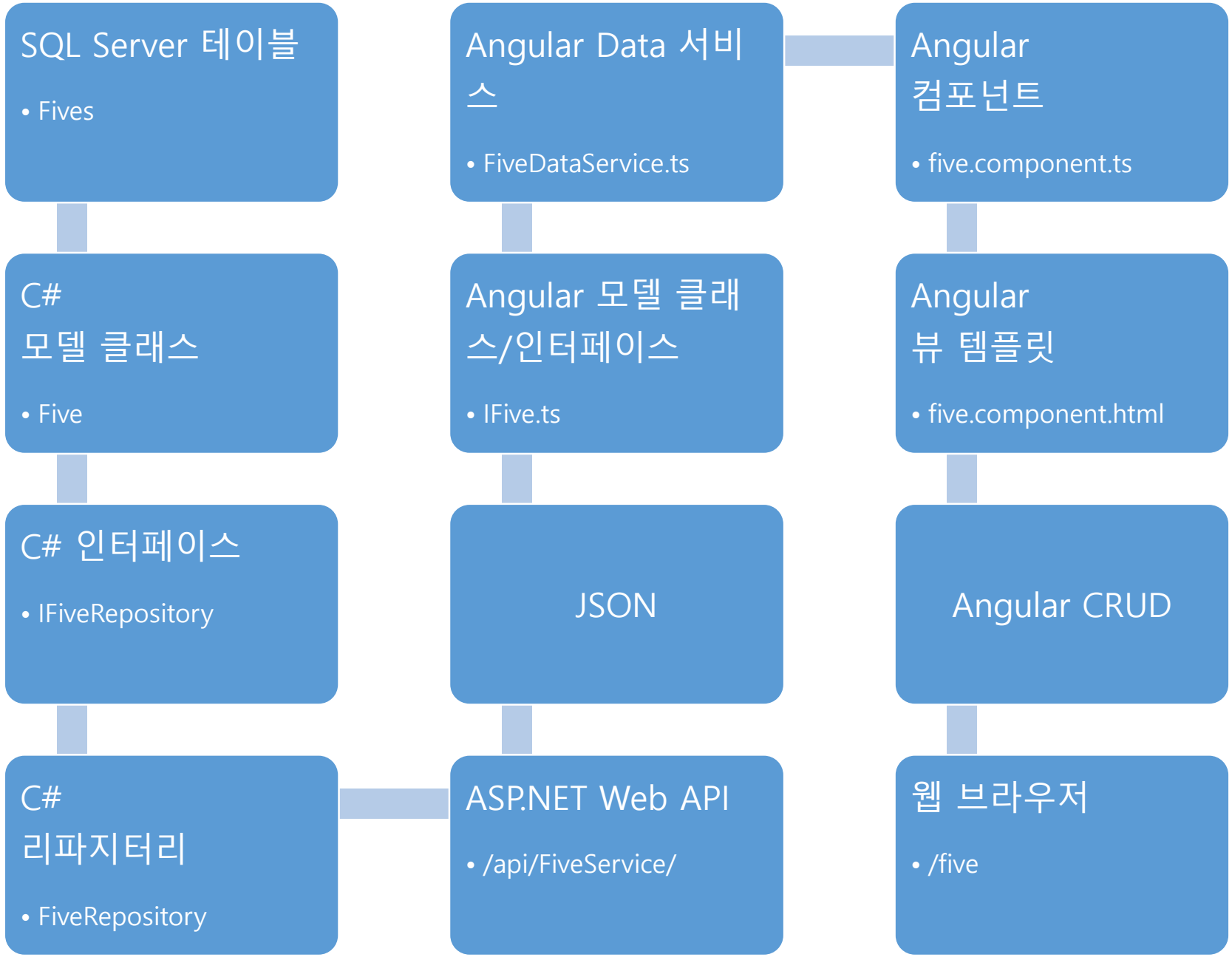

SQL Server
테이블
Angular
Data 서비스
Angular
컴포넌트
C#
모델 클래스
JSON
Angular
뷰 템플릿
C#
리파지터리
ASP.NET
Web API
웹 브라우저

아이디어 관리자

- 1 - Angular
- 2 - ASP.NET Core
- 3 - Azure
- 9 - Bootstrap
- 10 - C#
- 11 - Dapper
- 12 - EF Core

아이디어:

저장


SQL Server 테이블
• Fives
Angular Data 서비스
• FiveDataService.ts
Angular 컴포넌트
• five.component.ts
C# 모델 클래스
• Five
Angular 모델 클래스/인터페이스
• IFive.ts
Angular 뷰 템플릿
• five.component.html
C# 인터페이스
• IFiveRepository
JSON
Angular CRUD
C# 리파지터리
• FiveRepository
ASP.NET Web API
• /api/FiveService/
웹 브라우저
• /five

HTTP

- HttpModule 임포트
- http.get()
- subscribe()

Angular 2 HTTP 서비스

- Http는 HttpModule이 필요
  - app.module.ts
  - @anular/http

- import { Http, Response } from '@angular2/http';

- _http.get(URL).map((response: Response) => response.json());

MaximComponent.html

```
<h3>명언 관리</h3>
<h2>
    <span class="glyphicon glyphicon-thumbs-up"></span>
    {{ title }}
</h2>
<form #form="ngForm" novalidate>
    <input type="text" name="name" ngModel />
    <input type="text" name="content" ngModel />
    <input type="button" value="저장" (click)="submit(form.value)" />
</form>
<hr />
<p *ngIf="!maxims"><em>Loading...</em></p>
<div class="row" *ngIf="maxims">
    <div style="height:150px;"
         class="col-lg-3 col-md-4 col-sm-6 col-xs-12 thumbnail"
         *ngFor="let maxim of maxims">
        <a href="#" class="pull-right">
            <img src="http://placeimg.com/100/100/any"
                 class="img-thumbnail"
                 alt="{{ maxim.name }}" />
        </a>
        <h3>{{ maxim.name }}</h3>
        <p><em>{{ maxim.content }}</em></p>
        <div class="clearfix visible-xs"></div>
    </div>
</div>
```

MaximComponent.ts

TypeScript 가상 프로젝트 | "MaximComponent"

```
})
export class MaximComponent {
    private API_URI: string = "/api/MaximService";
    maxims = [];
    constructor(private _http: Http) {
        _http.get(this.API_URI).subscribe(result => {
            this.maxims = result.json();
        });
    }
    submit(val) {
        var headers = new Headers();
        headers.append('Accept', 'application/json');
        headers.append('Content-Type', 'application/json');
        this._http.post(
            this.API_URI,
            JSON.stringify(val),
            { headers: headers }).subscribe();
    }
}
```

비동기

toPromise() => then()

observable() => subscribe()

Promise와 Observable

Promise

- 단일 값 반환
- 최소 불가
- 기존 웹 브라우저 지원

Observable

- 다중 값 반환
- 취소 가능
- 표준 배열 함수 제공
  - (map, filter, reduce, 등)
- RxJS 라이브러리에 의존

앵귤러 서비스와 컴포넌트 클래스

Observable

(앵귤러 서비스)

Subscriber

(앵귤러 컴포넌트)

Promise와 Observable 코드 차이

- Promise
  - ~.then(값처리, 에러처리)
- Observable
  - ~.subscribe(값처리, 에러처리)
  - ~.subscribe(값처리, 에러처리, 완료처리)
  - toPromise()
    - Obsevable을 Promise로 변경

new Promise<T>(resol => { resol(...); });

- ().then(data => { this.data = data; });

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

@Injectable()
export class PromiseDemoService {
    getData() {
        //return [{ id: 1, name: "홍길동" }, { id: 2, name: "백두산" }];

        //return new Promise<any[]>(resolve => {
        //    return resolve(
        //        [{ id: 1, name: "홍길동" }, { id: 2, name: "백두산" }]
        //    );
        //});

        return new Promise<any[]>(resolve => {
            setTimeout(() => {
                resolve(
                    [{ id: 1, name: "홍길동" }, { id: 2, name: "백두산" }]
                );
            }, 2000);
        });
    }
}
```

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

@Component({
    selector: 'service-demo',
    template: require('./PromiseDemoComponent.html')
})
export class PromiseDemoComponent {
    isBusy = false;
    names: any[];
    constructor(private svc: PromiseDemoService) {
        this.getData();
    }

    getData() {
        //this.names = svc.getData();
        this.svc.getData().then(n => {
            this.isBusy = true;
            this.names = n;
            this.isBusy = false;
        });
    }
}
```

```html
<h1 *ngIf="isBusy">로딩중...</h1>
<h1 *ngIf="!isBusy">Promise 데모</h1>

<ul>
    <li *ngFor="let n of names">
        {{n.id}} - {{n.name}}
    </li>
</ul>
```

로딩중...

Promise 데모

- 1 - 홍길동
- 2 - 백두산

RxJS

- RxJS
  - Reactive Extensions for JavaScript
    - http://reactivex.io/rxjs
  - 비동기 처리를 위한 자바스크립트 라이브러리

```typescript
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import { of } from 'rxjs/observable/of';
import 'rxjs/add/operator/delay';
import 'rxjs/add/operator/do';

@Injectable()
export class ObservableDemoService {
    getData(): Observable<any[]> {
        let data = [{ id: 1, name: "홍길동" }, { id: 2, name: "백두산" }];

        return of(data)
            .delay(2000)
            .do(() => {
                console.log('가져오기');
        });
    }
}
```

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

@Component({
    selector: 'service-demo',
    template: require('./ObservableDemoComponent.html')
})
export class ObservableDemoComponent {
    isBusy = true;
    names: any[];
    constructor(private svc: ObservableDemoService) {
        this.getData();
    }

    getData() {
        this.svc.getData().subscribe(n => {
            this.isBusy = true;
            this.names = n;
            this.isBusy = false;
        });
    }
}
```

```html
<h1 *ngIf="isBusy">로딩중...</h1>
<h1 *ngIf="!isBusy">Observable 데모</h1>

<ul>
    <li *ngFor="let n of names">
        {{n.id}} - {{n.name}}
    </li>
</ul>
```

로딩중...

Observable 데모

- 1 - 홍길동
- 2 - 백두산

http.get().delay().map().catch()

참고자료

- 앵귤러2 공식 사이트
- 채널9
  - Microsoft Ignite
- 마이크로소프트 버추얼 아카데미

마무리

- Angular 기반의 웹 사이트 제작

감사합니다

감사합니다.
박용준

데브렉(http://www.devlec.com/) : 쉽게 배우는 Angular