

Angular SPA Development

soongon@hucloud.co.kr

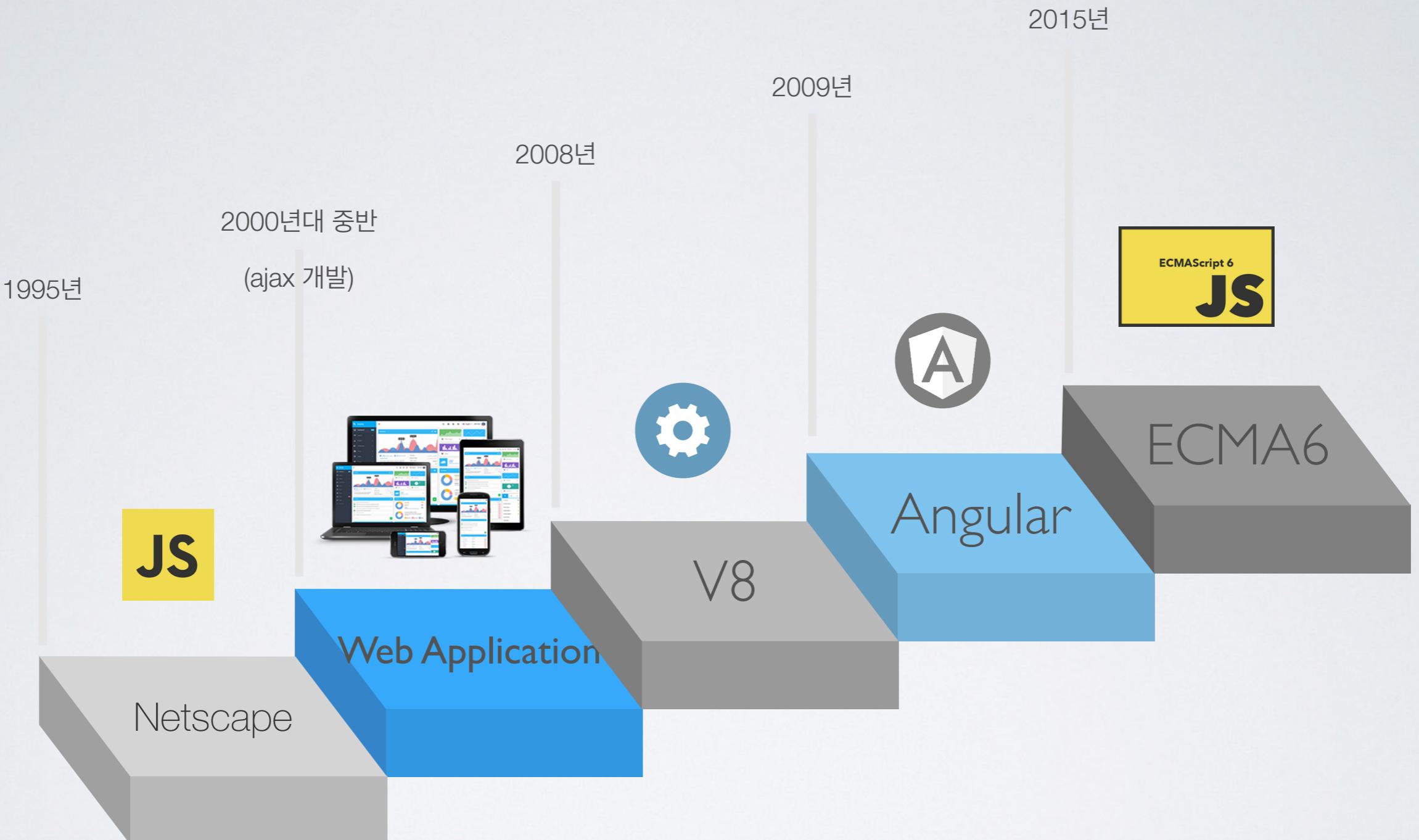
김 순곤

배울 내용

- Web Application (Framework)
- HTML / CSS / JavaScript
- EcmaScript6 / TypeScript / Node.js
- Angular CLI / NPM
- Angular 5.x (6.x)
 - Components
 - Router
 - Service (Dependency Injection / Reactive Programming)
 - Forms (Angular Form / Reactive Form)

0장 : 사전 지식

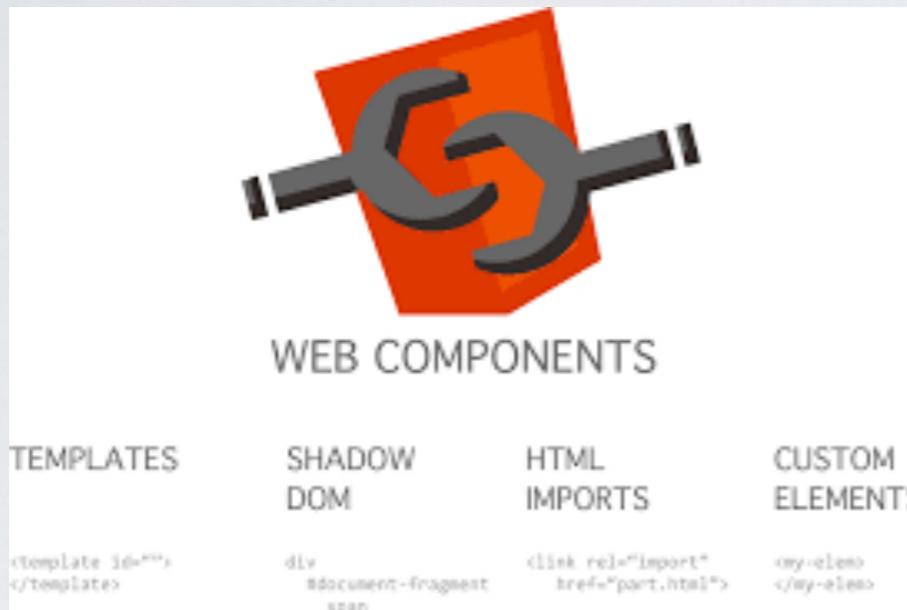
웹의 발전



웹표준

- 2009년 - W3C
 - HTML5 / CSS3 / EcmaScript5

Web Components



- <http://d2.naver.com/helloworld/188655>
- <https://www.webcomponents.org/>

웹 컴포넌트

- 웹 페이지를 특정 뷰 단위로 분리 가능하게 해 줌
 - Custom Elements
 - HTML Imports
 - Template
 - Shadow DOM
 - DOM 트리 내에 독립된 환경의 새로운 DOM 트리를 구축하여 외부로 부터 감춰진 요소

ECMA 스크립트에 대하여

ECMA 스크립트는 ECMA International의 표준

최초 ECMA 스크립트는 브라우저 언어인 Javascript와 Jscript간 차이를 줄이기 위한
공통 스펙 제안으로 출발 (1997, ECMA-262)

ECMA International은 전세계적인 표준기관

유럽 컴퓨터 제조협회로부터 기원함 (*ECMA*는 옛이름)

ECMA(European Computer Manufacturers Association)

C#, JSON, Dart을 포함한 많은 언어표준을 관리함

ECMA Script 의 진화

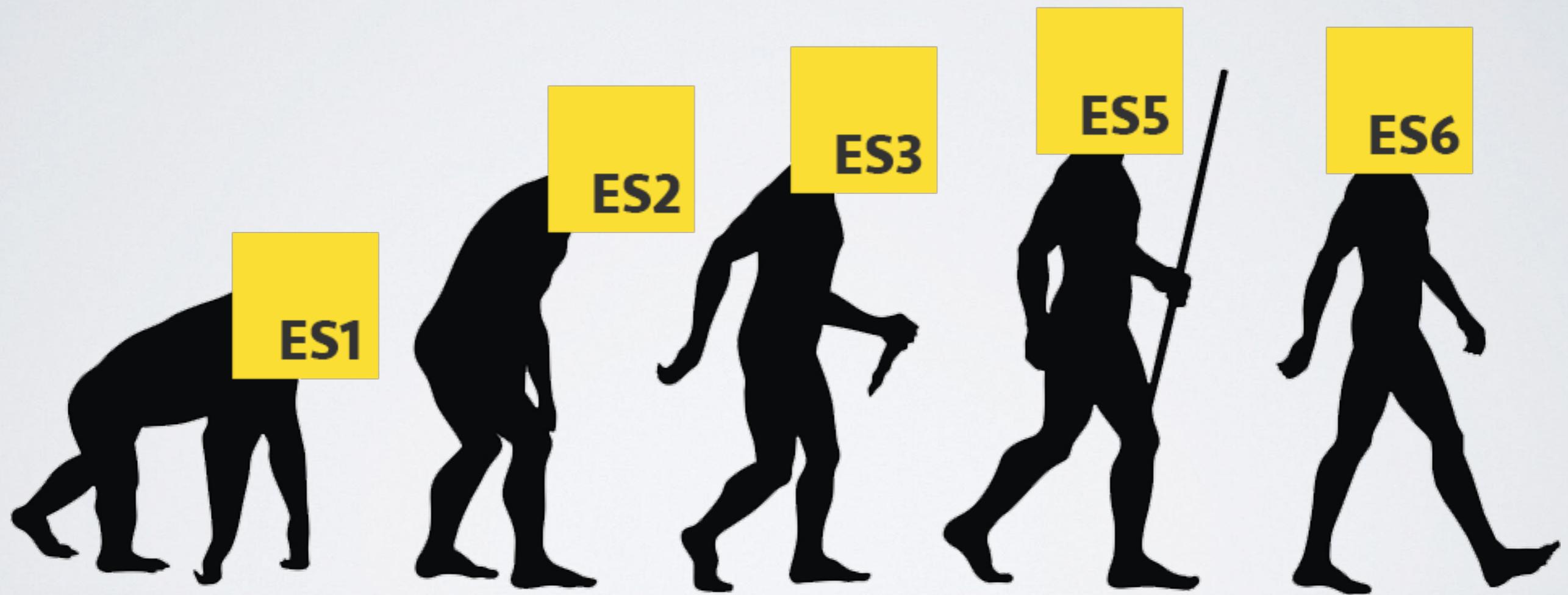
1997

1998

1999

2009

2015



ECMAScript 2015, 2016 소개

- Modules - 언어레벨에서 지원
- Classes - 프로토타입에서 클래스로
- Arrow Functions (=>) - 람다식 문법 지원
- Let and Const - 새로운 변수 선언 키워드
- Default, Rest, Spread - 함수 파라미터 세 가지 새로운 기능

TypeScript : www.typescriptlang.org



The screenshot shows the official TypeScript website at <https://www.typescriptlang.org>. The page features a large, dark blue header with the "TypeScript" logo and navigation links for Documentation, Samples, Download, Connect, and Playground. A banner at the top right encourages users to "Fork me on GitHub". Below the header is a large, stylized illustration of a city skyline with a Ferris wheel and a bridge, all in shades of blue. The main title "TypeScript" is prominently displayed in a large, white, sans-serif font. Below it, the tagline "JavaScript that scales." is written in a smaller, orange font. A paragraph explains that TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It also highlights the platform's compatibility ("Any browser. Any host. Any OS. Open source.") and provides download and documentation links. At the bottom, three icons represent different aspects of the language: a brace icon for "Starts and ends with JavaScript", a gear icon for "Strong tools for large apps", and a code editor icon for "State of the art JavaScript".

TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Any browser. Any host. Any OS. Open source.

Download Documentation

{JS} Starts and ends with JavaScript

Gears Strong tools for large apps

Code editor State of the art JavaScript

TypeScript의 역사



2012년 10월 첫 타입스크립트 버전 0.8 발표

2013년 6월 18일 타입스크립트 버전 0.9 발표

2014년 2월 25일 visual Studio 2013 빌트인 지원

2014년 4월 2일 타입스크립트 1.0 발표

2014년 7월 타입스크립트 컴파일러 발표, Github 이전

2016년 5월 타입스크립트 2.0 발표

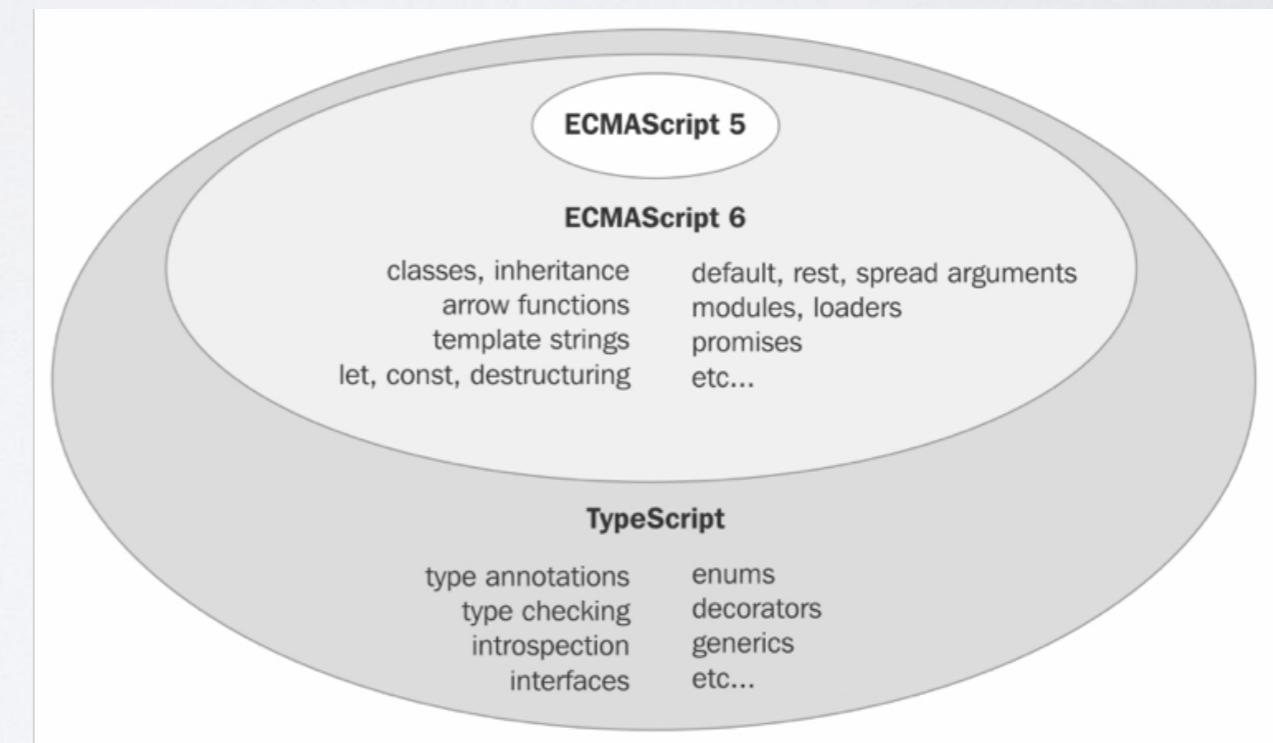
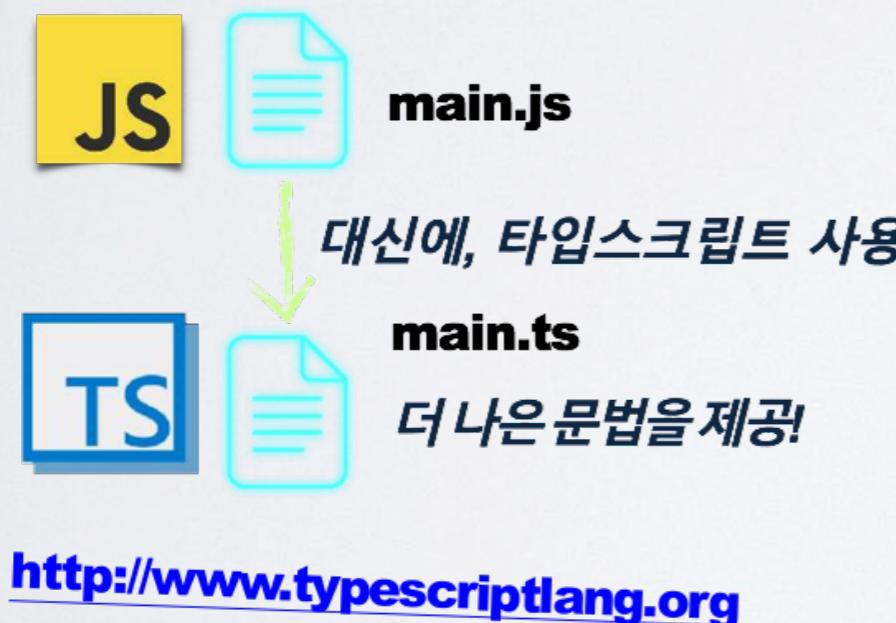
2017년 타입스크립트 2.3

TypeScript 소개



TypeScript는 마이크로소프트(**Anders Hejlsberg**)에서 개발한 자바스크립트의 확장된 언어이며, **ES2015**의 기능에 추가적으로 엄격한 타입체크와 객체지향적 기능 등을 추가한 자바스크립트의 **Superset**이다.

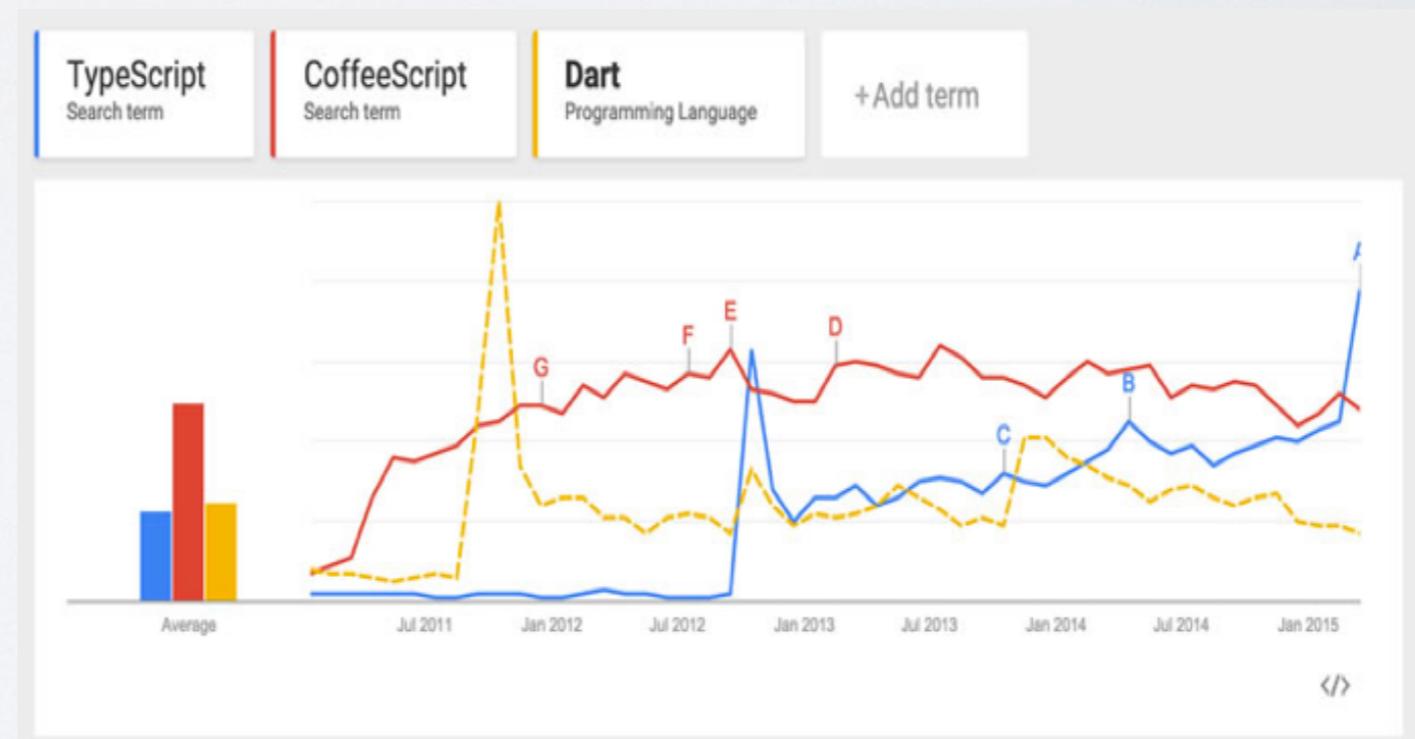
Angular 2 소스는 TypeScript로 개발되었다.



TypeScript 개요



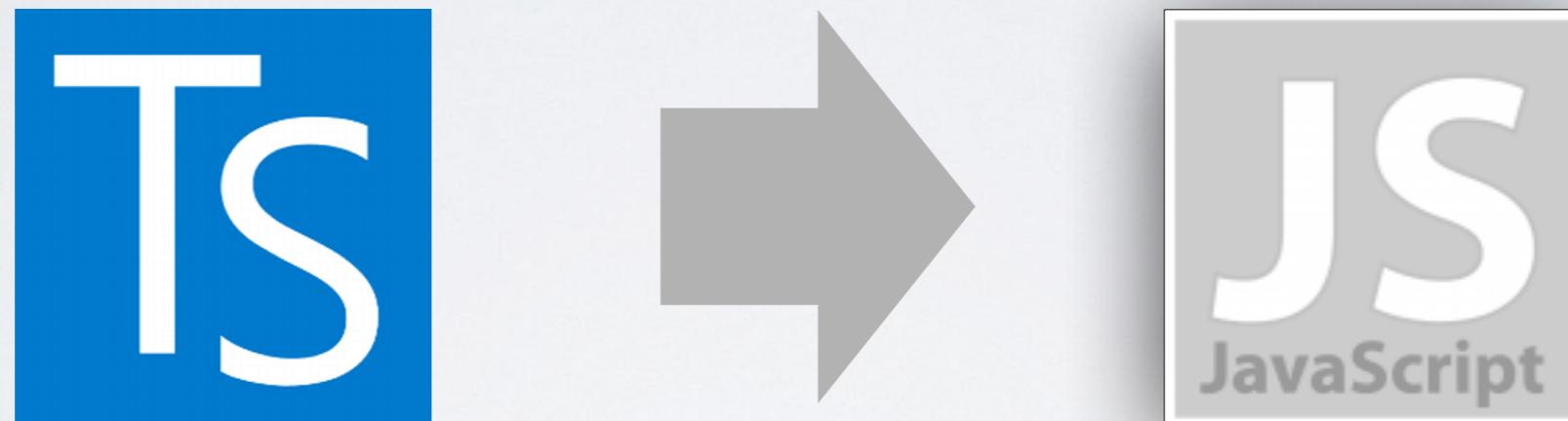
- 자바스크립트 —> ES2015(ES6) —> ES2016(ES7) —> TypeScript(?)
- 타입스크립트는 자바스크립트의 미래다?
- 자바스크립트(ES6, ES7) 기능에 타입스크립트의 필요한 기능을 추가하면 된다. (새로운 언어가 아님)
 - 클래스 관련기능
 - 정적 타이핑(static typing)



트랜스파일러 : tsc



TSC는 타입스크립트를 자바스크립트로 변환(transpiling)해주는 도구이다.



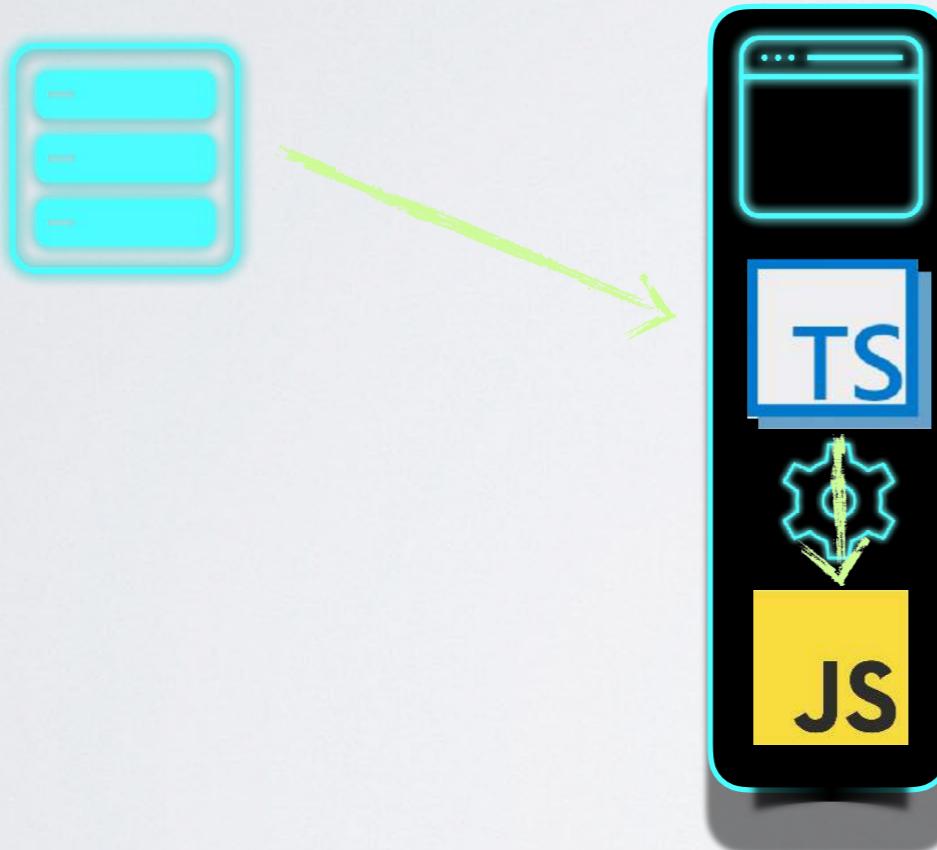
트랜스파일링

TypeScript : 트랜스파일 위치

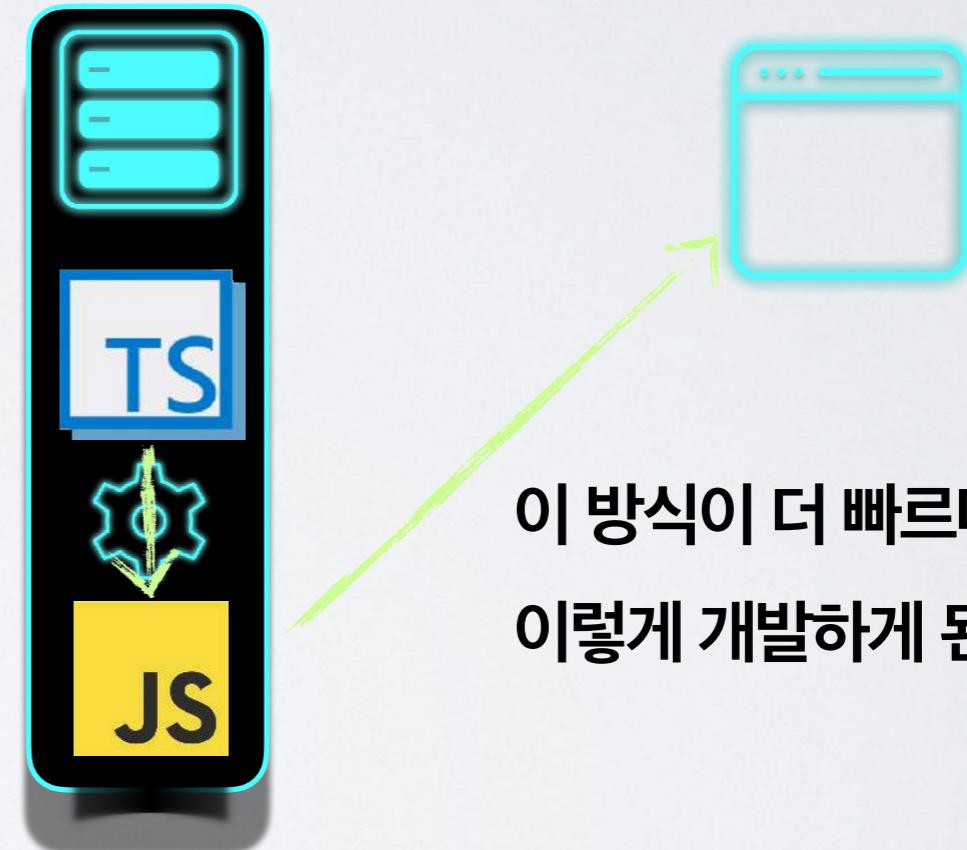


브라우저는 타입스크립트를 해석할 수 없으며, 자바스크립트로 변환하여 브라우저에서 처리되어야 한다. 다음 두 가지 방식이 사용되고 있다.

브라우저에서 자바스크립트로 변환



자바스크립트로 변환 후 브라우저로 로딩



이 방식이 더 빠르며, 주로
이렇게 개발하게 된다.

TypeScript : Playground



TS TypeScript - JavaScript t x TS Playground - TypeScript x

www.typescriptlang.org/play/index.html

TypeScript Documentation Samples Download Connect Playground Fork me on GitHub

TypeScript 2.2 is now available. Download our latest version today!

Select... TypeScript Share Options Run JavaScript

```
1 let myAdd2 = function(x: number, y: number)
2
3 let myAdd3: (baseValue:number, increment:number) => number
4   function(x: number, y: number): number
5
6 let myAdd: (x: number, y: number) => number
7   function(x: number, y: number): number
8 console.log(myAdd(10, 20));
9
```

```
1 var myAdd2 = function (x, y) { return x + y }
2 var myAdd3 = function (x, y) { return x + y }
3 var myAdd = function (x, y) { return x + y };
4 console.log(myAdd(10, 20));
5
```

TypeScript : Download



Get TypeScript

Node.js

The command-line TypeScript compiler can be installed as a Node.js package.

INSTALL

```
npm install -g typescript
```

COMPILE

```
tsc helloworld.ts
```

Visual Studio



Visual Studio 2017



Visual Studio 2015



Visual Studio Code

And More...



Sublime Text



Emacs



Atom



WebStorm



Eclipse



Vim

TypeScript 주요 문법

- Type

```
let firstName: string = "John";
let lastName = 'Smith';
let height: number = 6;
let isDone: boolean = false;
```

```
var numbers:number[] = [1, 2, 3];
var names:Array<string> = ['Alice', 'Helen', 'Claire'];
```

- Interface

```
interface IVehicle {
  wheels: number;
  engine: string;
  drive();
}
```

- Decorator

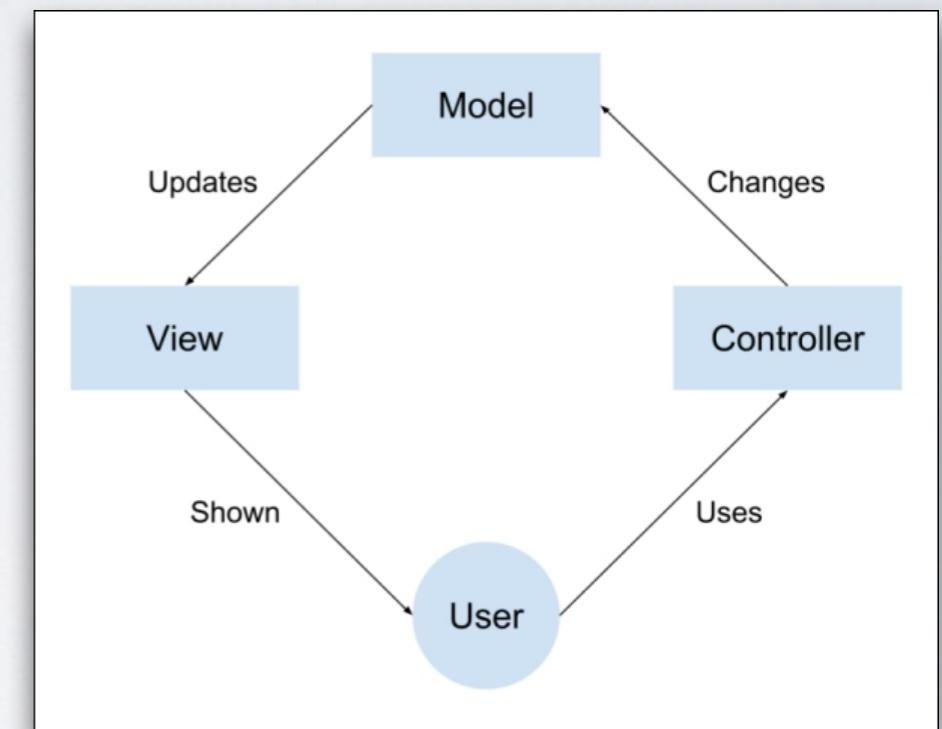
```
function Decorator(target: any) {
}
@Decorator
class MyClass {
}
```

```
function DecoratorWithArgs(options: Object) {
  return (target: Object) => {
    ...
  }
}

@DecoratorWithArgs({ type: 'SomeType' })
class MyClass {
}
```

3-tier 웹 개발

- 대부분 웹 애플리케이션은 3-tier 아키텍처로 구성됨
 - 데이터 - 로직 - 화면
 - 데이터베이스 - 서버 - 클라이언트
 - 데이터베이스 - 서버 로직 - 클라이언트 로직, 클라이언트 UI

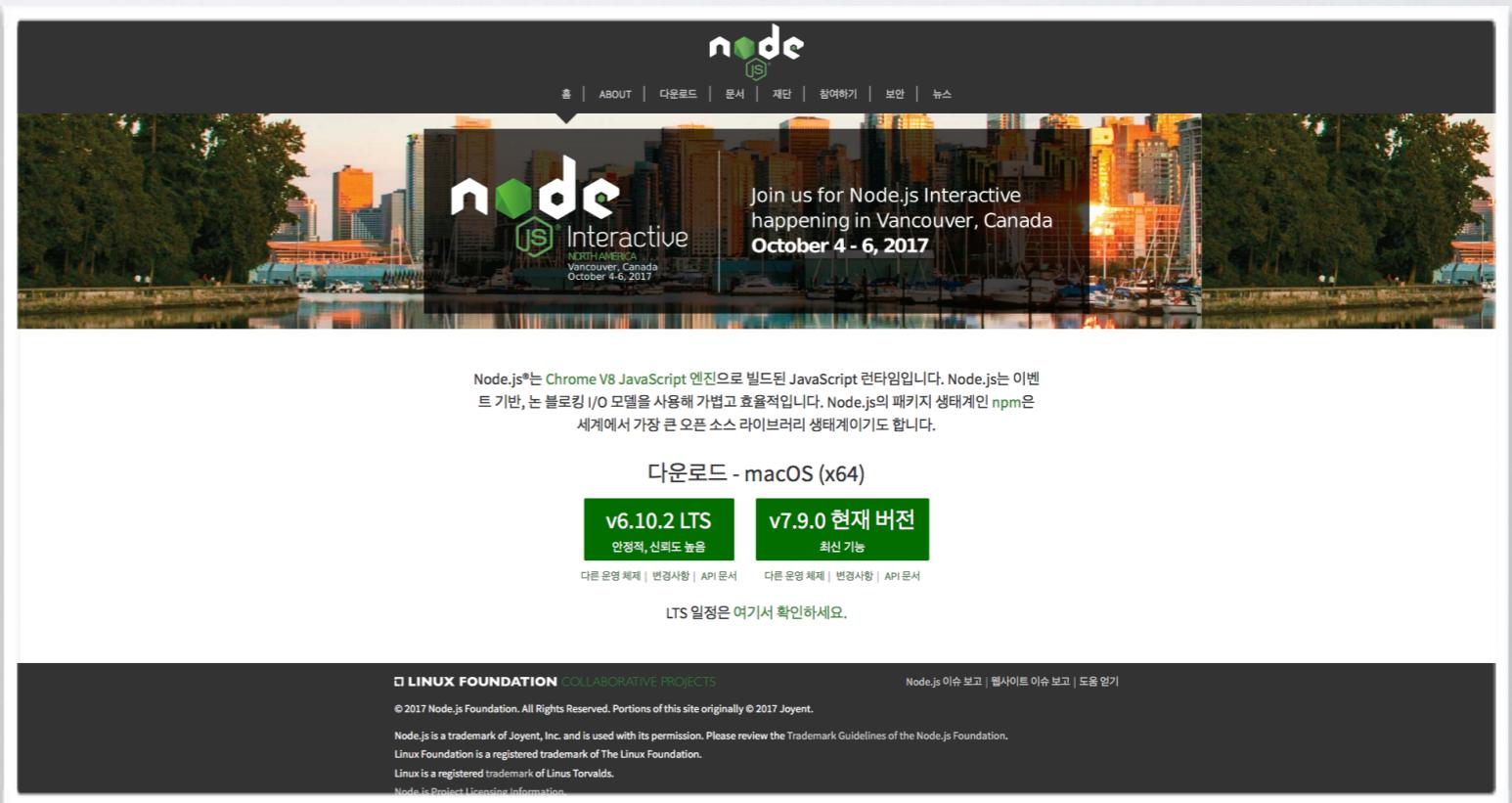


NodeJS 설치

- Node.js 와 npm 설치 : <http://nodejs.org>
 - Node 설치 시 NPM 기본으로 같이 설치 됨
- Git 설치 : <https://git-scm.com/>
 - 특정 모듈 설치 시 필요

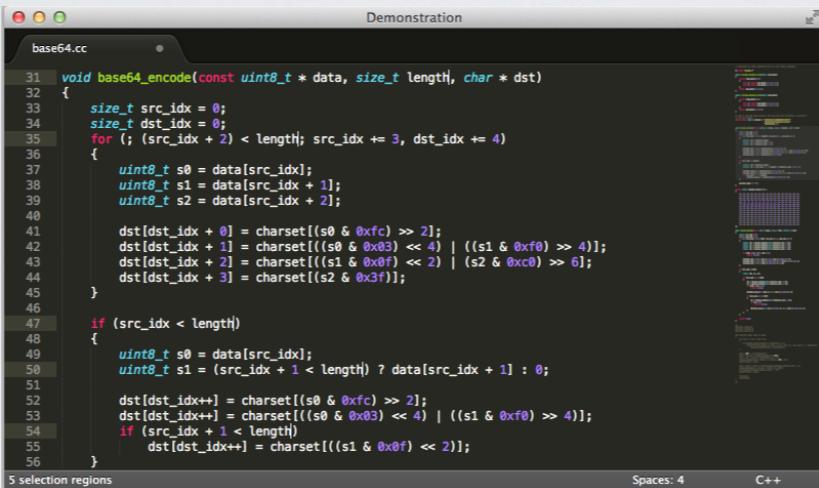
- 설치 확인
 - **node --version**
 - **npm --version**
 - **git --version**

위 세 가지
터미널에서 설치 확인

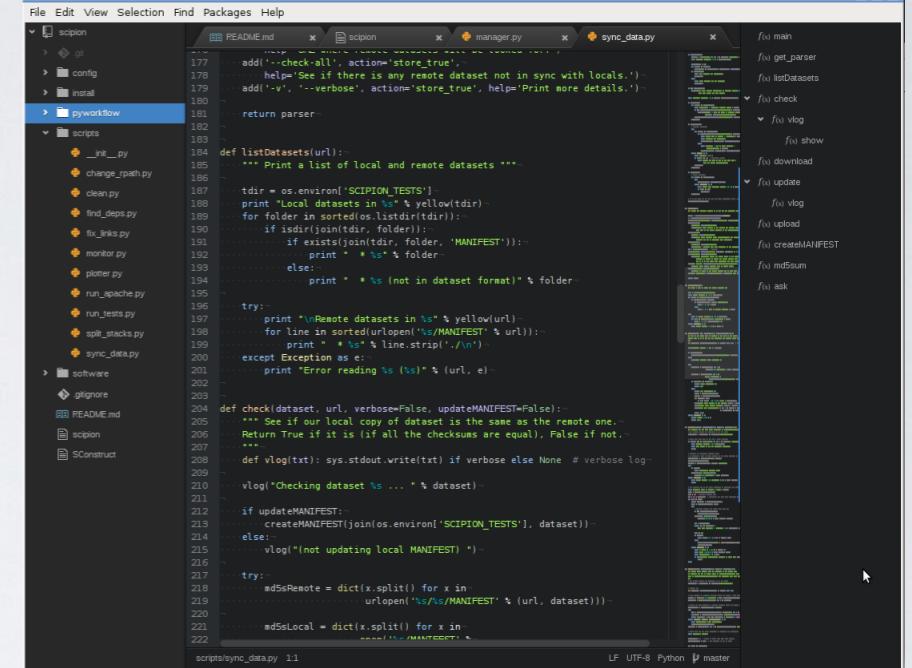


IDE 또는 Text Editor

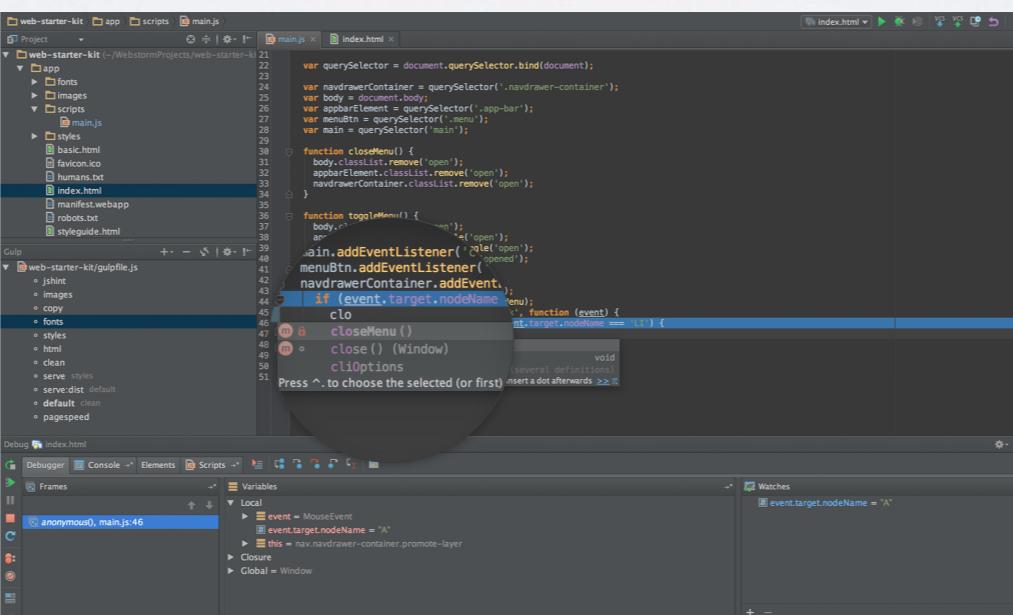
- sublime text



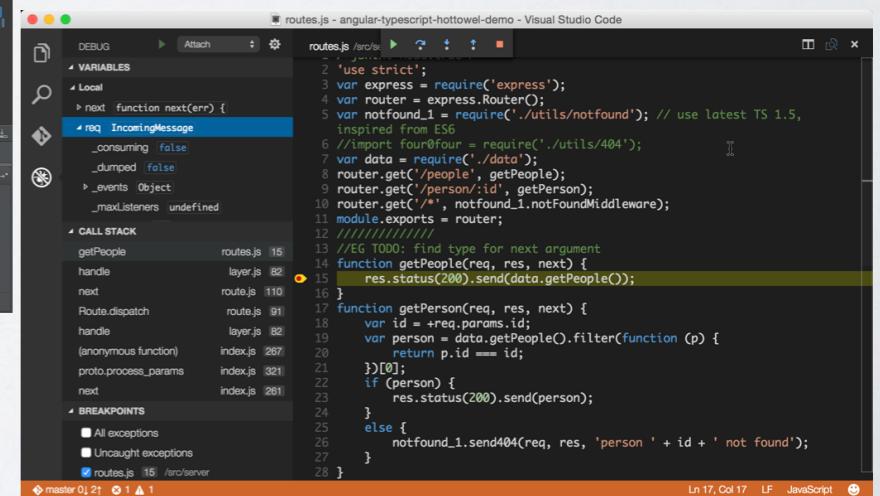
- Atom



- # • WebStorm



- VS Code



NPM (Node Package Manager)

- NodeJS 를 위한 써드파티 모듈 저장소
- 로컬, 글로벌 모듈을 관리하기 위한 CLI 툴

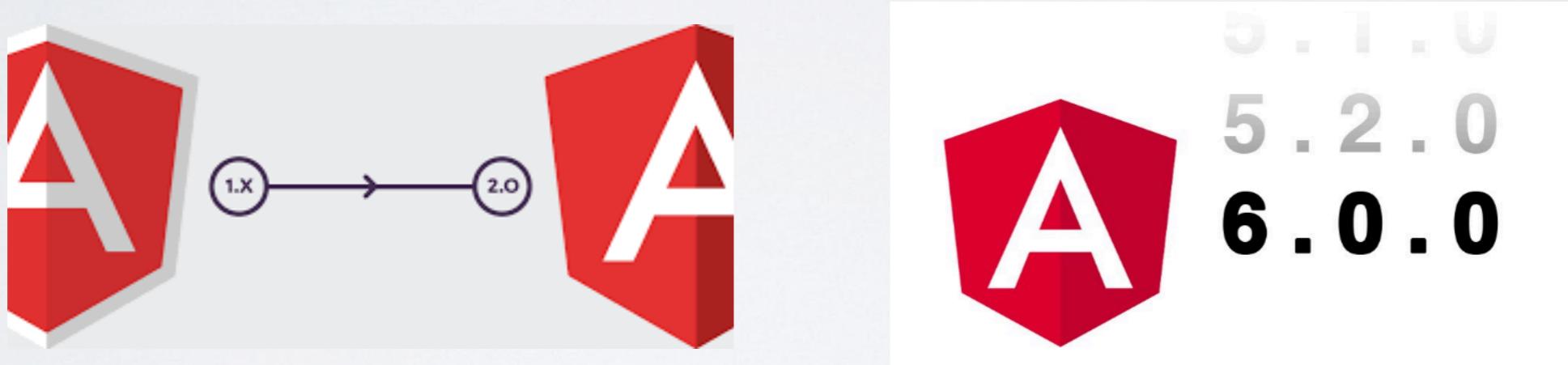
<https://www.npmjs.com/>

NPM 사용

- 로컬과 글로벌 모듈을 설치
 - 로컬에 설치 : node_modules
 - 글로벌 : C:\Users\%USERNAME%\AppData\Roaming\npm\node_modules
- 패키지(모듈) 설치
 - npm install <패키지 명>
 - npm install -g <패키지 명>
- 의존성 관리를 위한 : package.json
 - npm init - package.json 파일 생성
 - npm install - package.json 파일의 모든 의존 모듈 설치
 - npm install —save express - package.json 갱신

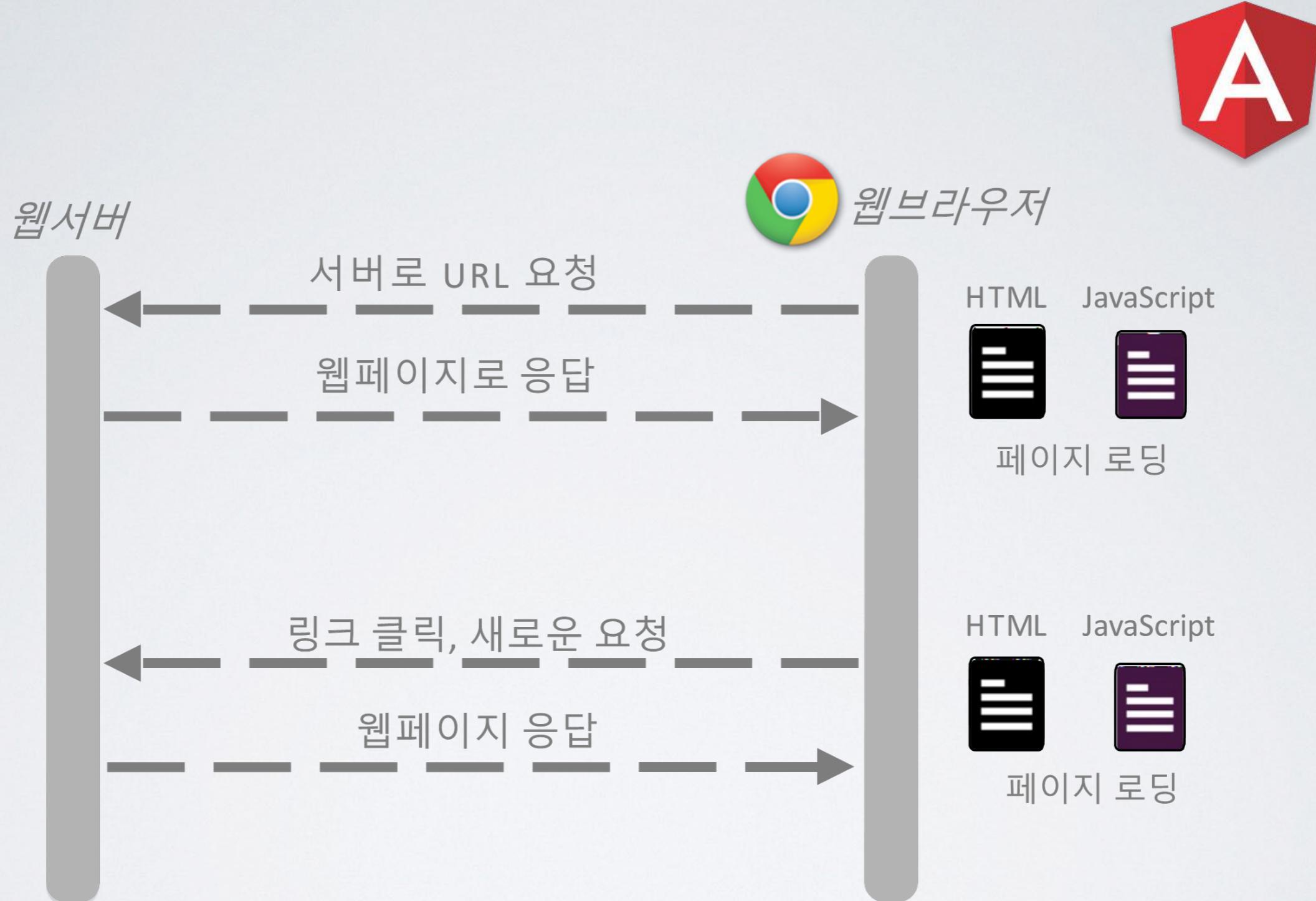
Angular 의 세계로

- 2014.09.18 : 구글 앵귤러 새로운 버전 최초 커밋
- 2017.03 : Angular 4 발표
- 2018.05 : Angular 6

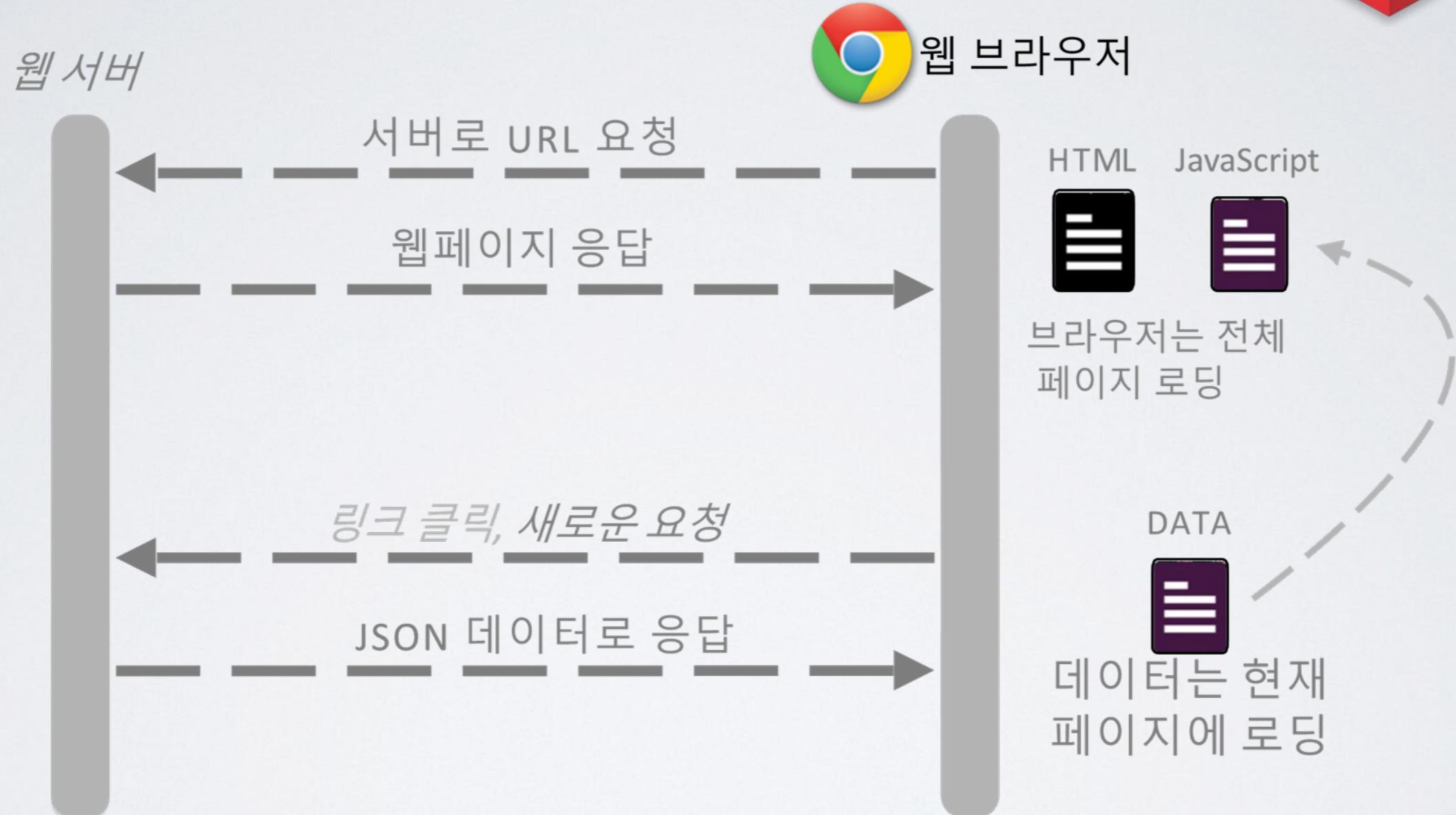


<https://www.madewithangular.com/#/categories/google>

기존 웹 어플리케이션 구조



Data-Driven 웹 아키텍처



I 장 : ANGULAR

Angular 주요내용 - 1

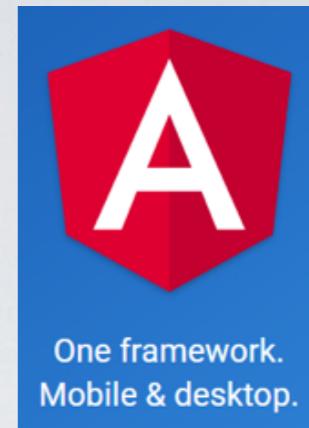
- Anuglar 소개
- Angular 구성 요소
- TypeScript와 Angular 설치와 설정
- Angular 애플리케이션 작성 및 구성
- Angular 컴포넌트 사용

Angular 주요내용 - 2

- Angular Data Binding
- Routing
- Service
 - Dependency Injection
 - Reactive Programming
- Forms

Angular 소개

- SPA 를 위한 프론트-엔드 자바스크립트 프레임워크
 - MVC 와 유사한 아키텍처를 사용
 - 프론트엔드 코드를 단순하고 명료하게 하기위해서,
 - HTML, JavaScript, 그리고 CSS 를 하나로 조합하는 방법을 제공.



Google

The slide features four distinct cards arranged in a grid-like layout, each highlighting a different aspect of the Angular framework:

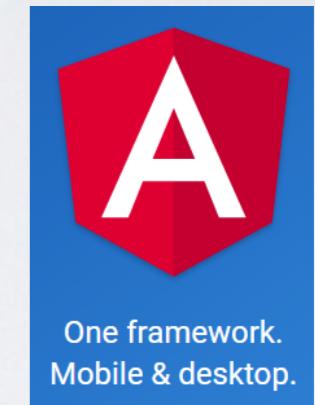
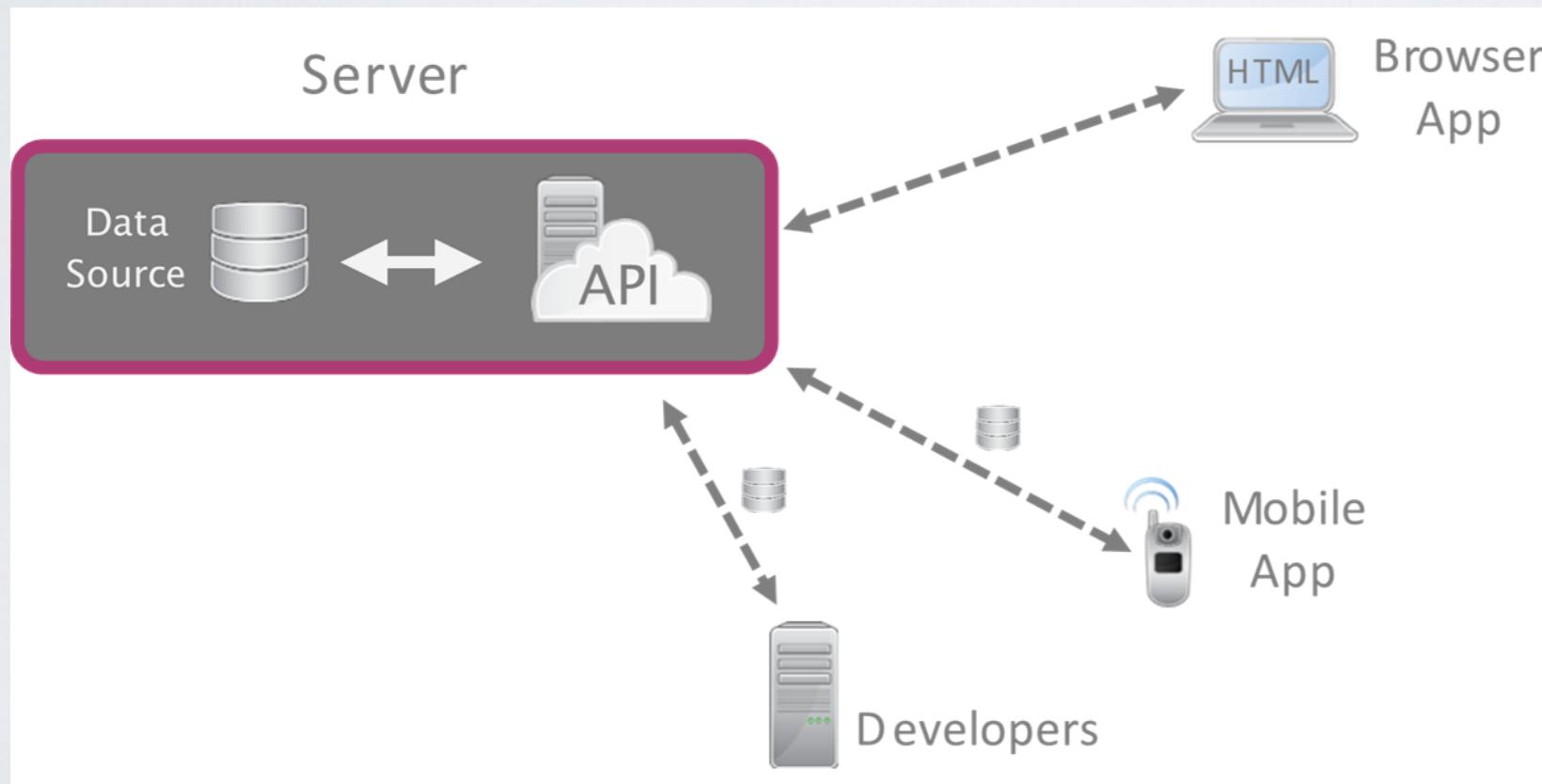
- Develop Across All Platforms**: This card shows icons of a laptop, a desktop monitor, and a smartphone, all displaying the Angular logo (a red hexagon with a white 'A'), symbolizing the platform-agnostic nature of the framework.
- Incredible Tooling**: This card displays a screenshot of an IDE or code editor showing a file named 'app.ts'. The code includes basic Angular syntax like imports and component declarations. A tooltip or code completion dropdown is visible over the word 'font' in a template, showing options like 'fontFamily', 'fontFeatureSettings', 'fontSize', and 'fontSizeAdjust', demonstrating the powerful tooling support provided by the Angular ecosystem.
- Loved by Millions**: This card presents a world map where numerous locations are marked with red hexagons containing a white 'A', representing the global community of users and developers who have adopted the Angular framework.

Angular



Google

- Google이 만든 오픈소스 다이나믹 웹 어플리케이션 프레임워크.
- 프론트엔드 코드를 단순하고 명료하게 하기위해서,
HTML, JavaScript, 그리고 CSS 를 하나로 조합하는 방법을 제공.



Angular 1 & 2 차이점?



속도 – Angular 2가 더 빠름.

컴포넌트 – Controller와 Scope 대신에 Component를 사용, 더 심플한 컨셉.

단순해진 디렉티브 – 사용자 정의 Directive의 작성이 단순해짐.

직관적 데이터 바인딩 – 데이터를 HTML에 바인딩 할 때나 버튼 클릭 이벤트를 부여할 때 더 직관적이고 단순한 문법을 제공.

서비스는 단순 클래스로 바뀜.

라우터 개선 및 재설계

많은 사소한 개선사항이 있음.

AngularJS vs. Angular

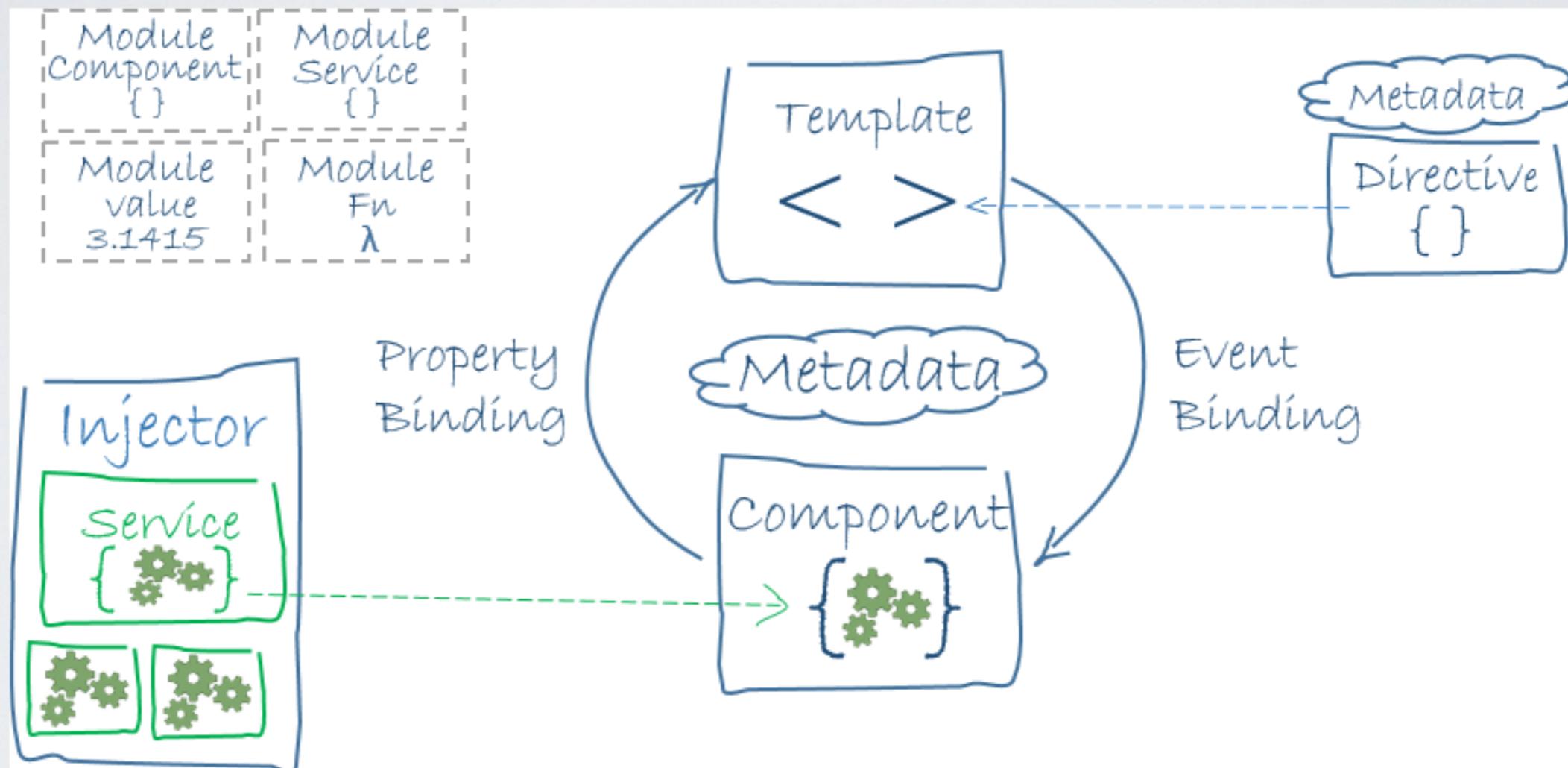
```
module.directive('myConfirmation', function() {
  return {
    scope: {},
    bindToController: {
      message: '=',
      onOk: '&'
    },
    controller: function() { },
    controllerAs: 'ctrl',
    template: `
      <div>
        {{ctrl.message}}
        <button ng-click="ctrl.onOk()">
          OK
        </button>
      </div>
    `
  }
});
```

```
@Component({
  selector: 'my-confirmation',
  inputs: ['message'],
  outputs: ['ok']
})
@View({
  template: `
    <div>
      {{message}}
      <button (click)="ok()">OK</button>
    </div>
  `
})
class MyConfirmation {
  okEvents = new EventEmitter();
  ok() {
    this.okEvents.next();
  }
}
```

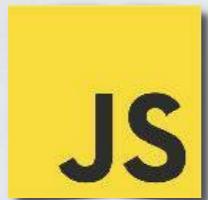
Angular2의 아키텍쳐



Component를 이용한 Template과 Service 로직 관리



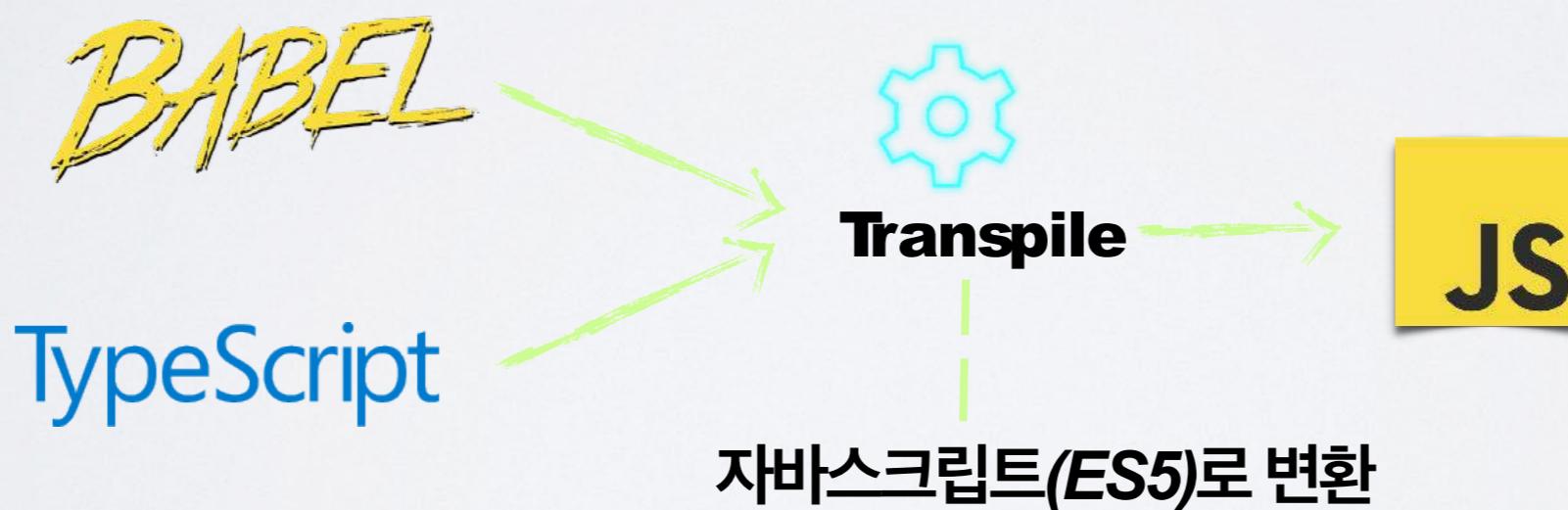
Angular 2에서 사용되는 언어는?



JavaScript

최신 자바스크립트 버전을 모든 브라우저가 지원하지 않음.

다음과 같은 방법으로 해결 가능:



빌드(?)가 필요!

- 개발환경과 배포환경이 서로 다름
- *.ts 로 개발 후 배포 때는 *.js 로 변환
- 여러 파일 (html, css, img, ts(js) 등) 을 하나로 묶어 다운로드 시간 단축

webpack 을 사용

HELLO ANGULAR

Angular 시작하기

Step 1. 개발 환경 설정

1. Angular CLI 글로벌 설치

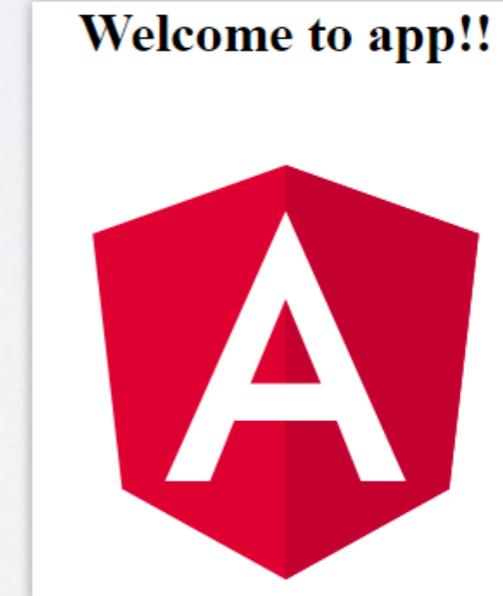
```
npm install -g @angular/cli
```

2. 새로운 프로젝트 생성

```
ng new hello-angular
```

3. 어플리케이션 실행

```
cd hello-angular  
ng serve --open
```



Angular CLI : <https://cli.angular.io>

- Angular 를 위한 커맨드라인 인터페이스
 - **ng new**
새로운 프로젝트 스캐폴딩 (Scaffolding)
 - **ng generate**
컴포넌트, 라우트, 서비스, 파이프 등 생성
 - **ng serve**
개발 테스트 환경 제공
 - 기타 : Test, Lint, Format

실습 : HELLOWORLD

ANGULAR CLI

Angular CLI ?

- Angular 프로젝트 스캐폴딩을 생성, 실행, 빌드 가능
- 다양한 구성 요소를 선별적으로 추가할 수 있는 커맨드-라인 인터페이스(command line interface)
- Angular CLI가 지원하는 기능
 - Angular 프로젝트 생성
 - Angular 프로젝트에 컴포넌트, 디렉티브, 파이프, 서비스, 클래스, 인터페이스 등의 구성 요소 추가
 - LiveReload를 지원하는 내장 개발 서버를 사용한 Angular 프로젝트 실행
 - Unit/E2E(end-to-end) 테스트 환경 지원
 - 배포를 위한 Angular 프로젝트 패키징

cli.angular.io

The screenshot shows a web browser window displaying the Angular CLI homepage at <https://cli.angular.io>. The page has a red header bar with the Angular CLI logo, navigation links for 'DOCUMENTATION' and 'GITHUB', and a 'GET STARTED' button. Below the header is a large orange section featuring a terminal-like box showing command-line instructions:

```
> npm install -g @angular/cli  
> ng new my-dream-app  
> cd my-dream-app  
> ng serve
```

To the right of the terminal box, the text 'Angular CLI' is displayed in large white font, followed by the subtitle 'A command line interface for Angular' and another 'GET STARTED' button.

Below the orange section, there are two sections with rounded corners:

- ng new**
The Angular CLI makes it easy to create an application that already works, right out of the box. It already follows our best practices!
- ng generate**

각종 명령어와 옵션, Angular CLI의 설정 파일인 `.angular-cli.json`에 대한 상세한 문서를 제공

Angular CLI 설치

- \$ npm install -g @angular/cli

```
$ npm uninstall -g angular-cli
```

```
$ npm cache clean
```

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

- 설치 후 버전 확인

- \$ ng version

- 도움말 : \$ ng help

프로젝트 생성

- \$ ng new <project-name>
 - 프로젝트가 생성되고 기본 패키지 매니저 npm 이 설정
- \$ cd <project-name>
- \$ ng serve
 - 프로젝트 실행 - <http://localhost:4200>
 - 다른 포트에서 실행 : \$ ng serve --port 4201

프로젝트 구성요소 추가

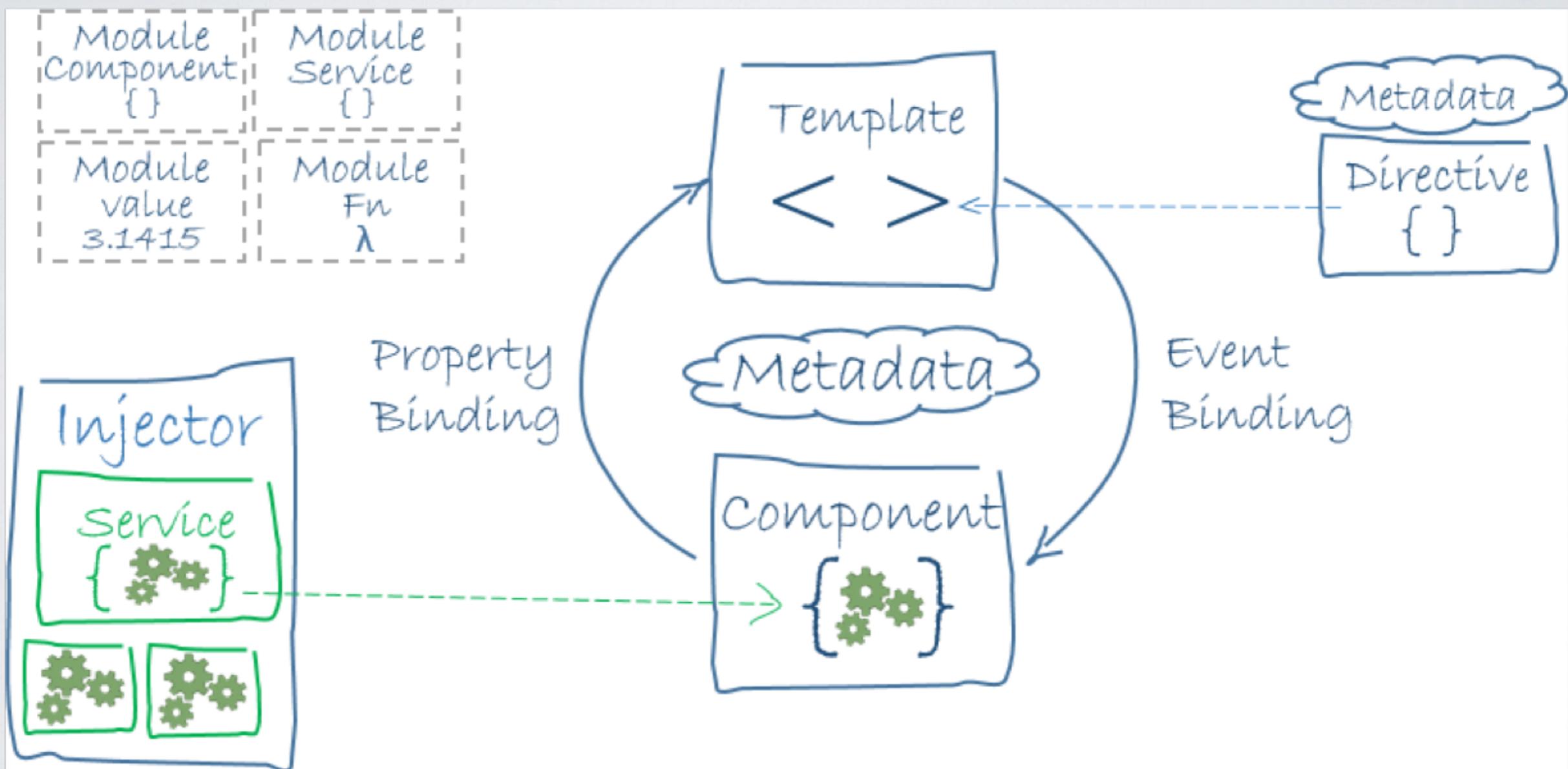
| 추가 대상 구성요소 | 명령어 | 축약형 |
|------------|--------------------------------------|-----------------------|
| 컴포넌트 | ng generate component component-name | ng g c component-name |
| 디렉티브 | ng generate directive directive-name | ng g d directive-name |
| 파이프 | ng generate pipe pipe-name | ng g p pipe-name |
| 서비스 | ng generate service service-name | ng g s service-name |
| 모듈 | ng generate module module-name | ng g m module-name |
| 가드 | ng generate guard guard-name | ng g g guard-name |
| 클래스 | ng generate class class-name | ng g cl class-name |
| 인터페이스 | ng generate interface interface-name | ng g i interface-name |
| Enum | ng generate enum enum-name | ng g e enum-name |

아키텍처

Angular Architecture

Angular 아키텍처

- Component 를 이용한 Template 과 Service 로직 관리



Angular 아키텍처

- 목표
 - HTML과 JavaScript 를 통합하여
 - 관리하기 쉽고 확장성 있는
 - 애플리케이션 개발
- 컴포넌트 기반 접근법
- 런타임 시 서비스 인젝션 사용

컴포넌트

Angular Components

Component : 컴포넌트 중심의 개발

Component 정의

기능 명세에 의한 개발

명세 따른 배포, 조립 가능한 독립 구성단위

컴포넌트는 인터페이스만을 통해서 접근

Component 중심 개발의 특징

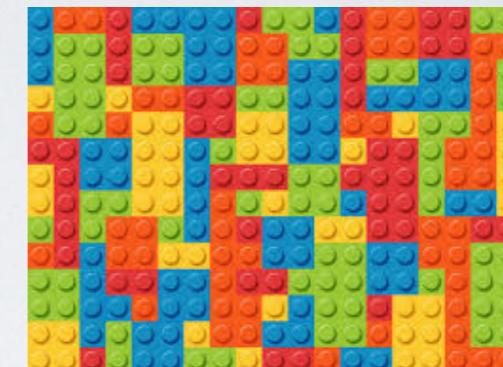
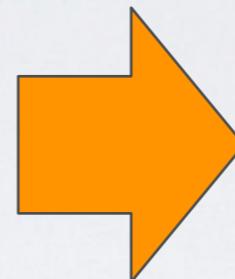
관심사 분리의 개발 방법으로 컴포넌트간 연결이 느슨하다(loosely coupled).

컴포넌트 재활용에 초점을 맞춘다.

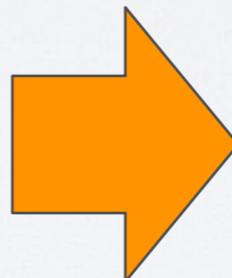
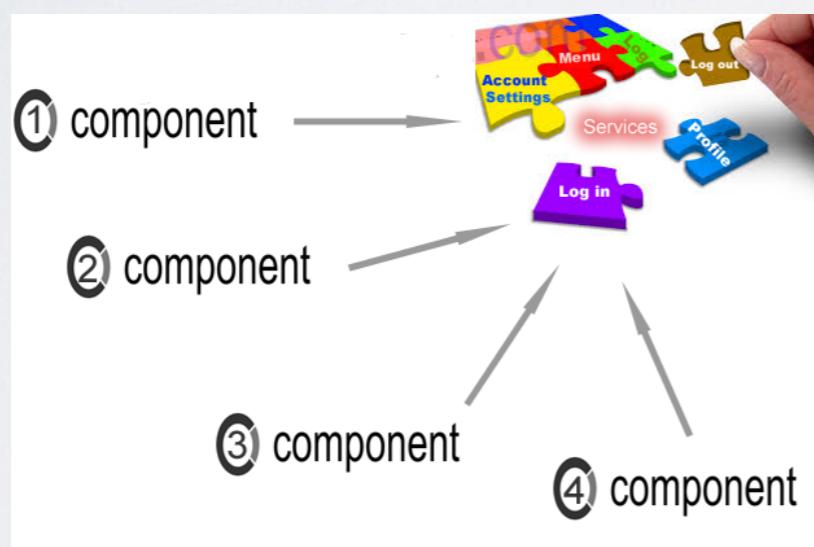
Component : Angular 를 활용한 컴포넌트 중심의 개발



Component



Component-Based System



Component-Based
Angular2 Application

Angular 컴포넌트

- Angular 애플리케이션의 Essential Building Block
 - 사용자 인터페이스 (UI) 의 한 부분
 - 대부분 애플리케이션은 최소 하나의 루트 컴포넌트로 구성
 - 다른 뷰를 다루기 위해 여러개의 컴포넌트를 사용
 - @Component 데코레이터를 사용하여 정의

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

@Component({
  selector: 'mean-app',
  template: '<h1>I AM AN APPLICATION COMPONENT</h1>'
})
export class AppComponent { }
```

Angular 템플리트

- 템플리트는 컴포넌트의 뷰를 렌더링 하는데 사용
 - 기본 HTML과 Angular 전용 속성이나 태그로 구성
 - 뷰를 외부 파일로 별도로 분리

AppComponent 클래스

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

@Component({
  selector: 'mean-app',
  templateUrl: 'app/app.template.html',
  styleUrls: ['app/app.component.css'],
})
export class AppComponent { }
```

Angular 디렉티브

1. 컴포넌트 �렉티브 (@Component) :

- @Directive 를 확장하여 작성됨 (템플리트가 추가)

2. Attribue �렉티브 : DOM 요소의 동작이나 노출을 변경

- ngClass : 엘레먼트에 CSS클래스 적용
- ngStyle : 엘레먼트에 인라인 스타일 제공
- ngModel : 폼 요소의 two-way 바인딩 생성

3. Structural �렉티브 : 코드의 로직을 수행하기 위해 제공

- ngIf : 분기
- **Angular 의 기본 작동원리 : 디렉티브를 사용하여 뷰를 다이나믹하게 생성하는 것**
- ngFor : 반복
- ngSwitch : 분기

@Component

- Angular는 컴포넌트를 조합해서 애플리케이션을 작성
 - 컴포넌트는 @Component 디렉티브로 작성
 - 하나 이상의 컴포넌트가 필요 : AppComponent
 - 컴포넌트는 계층적 구조로 구성됨

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

@Component({
  selector: 'sample-component',
  template: '<h1>I'm a component</h1>'
})
export class SampleComponent { }
```

```
import { Component } from '@angular/core';
import { SampleComponent } from 'sample.component';

@Component({
  selector: 'mean-app',
  template: '<sample-component></sample-component>',
  directives: [SampleComponent]
})
export class AppComponent { }
```

@Component : 컴포넌트 구성 예

- 컴포넌트는 Angular App의 구성요소 (Building blocks)
 - 각 컴포넌트는 서로 포함 될 수 있다.



각 컴포넌트는 다음을 포함한다.

- **class file**
- **html file**
- **css file**

컴포넌트 기반 접근법

- 웹 컴포넌트
 - HTML, CSS, Javascript 를 하나로 묶어 재사용 가능한 모듈로 만드는 것
 - Angular 에서의 컴포넌트란?
 - 커스텀 엘리먼트(HTML5)로 볼수 있다.
 - 특정 컴포넌트는 특정 Element를 말한다.
 - 사용예) <my-component></my-component>

Angular 의 컴포넌트

- 컴포넌트 중심의 Angular 개발 모델
 - Angular 는 컴포넌트 중심으로 개발을 진행
 - 컴포넌트는 Template과 Logic 을 포함
 - 컴포넌트는 다른 컴포넌트나 서비스를 주입 받을 수 있음

Angular에서 컴포넌트 란?

- 컴포넌트 = 화면
 - 화면은 여러 개의(심지어는 수백개 이상) 컴포넌트로 구성
 - 최상위 컴포넌트가 기본 제공 (App 컴포넌트라고 함)
 - 컴포넌트는 계층적 구조로 이루어짐
 - 커스텀 엘레먼트 <my-comp></my-comp>
- Angular에서 컴포넌트는 다음 세 가지 파일로 이루어져 있음
 - html / css / ts

실습 : 컴포넌트 나누기

온라인 옵션 어플리케이션

The screenshot shows a web application for an auction platform named 'ngAuction'. The top navigation bar includes links for Home, About, Services, and a dropdown menu. The main content area is divided into two sections: a search form on the left and a product list on the right.

Search Form (Left):

- Product title: An input field labeled 'Title'.
- Product price: A dropdown menu labeled 'Price'.
- Product category: A dropdown menu labeled 'Category'.
- A blue 'Search' button.

Product List (Right):

- A large placeholder text area with dimensions '800 x 300' and arrows indicating scrollability.
- A descriptive text below the placeholder: 'We'll render several ProductItem components here'.

Page Footer:

Copyright © ngAuction 2018

데이터 바인딩

Angular Data Binding

Angular 데이터 바인딩

I. Interpolation 바인딩

- HTML에 변수를 바인딩

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

@Component({
  selector: 'mean-app',
  template: '<h1>{{title}}</h1>'
})
export class AppComponent {
  title = 'MEAN Application';
}
```

2. Property 바인딩

- isButtonDisabled 속성이 true 이면
- 버튼이 disable 됨

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

@Component({
  selector: 'mean-app',
  template: '<button [disabled]="isButtonDisabled">My Button</button>'
})
export class AppComponent {
  isButtonDisabled = true;
}
```

3. Event 바인딩

- AppComponent의 showMessage()

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

@Component({
  selector: 'mean-app',
  template: '<button (click)="showMessage()">Show Message</button>'
})
export class AppComponent {
  showMessage() {
    alert('This is a message!')
  }
}
```

4. Two-way 바인딩

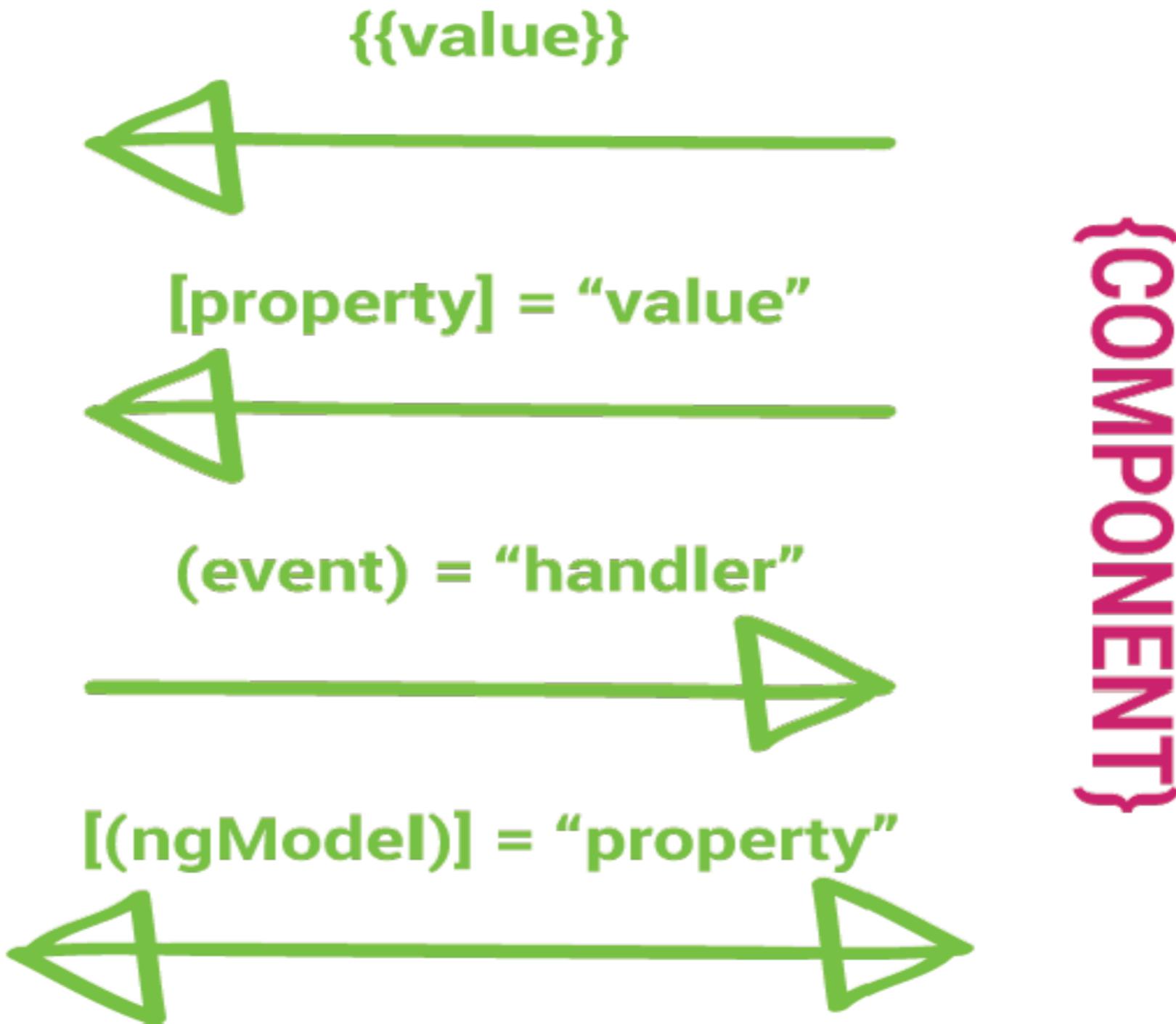
- [(ngModel)]로 구현

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

@Component({
  selector: 'mean-app',
  template: '<h1>Hello {{name}}</h1><br><input [(ngModel)]="name">'
})
export class AppComponent {
  name = ''
}
```

Angular 데이터 바인딩

<TEMPLATE>



{COMPONENT}

Interpolation : {{ }}

이중 브레이스 '{{ }}' 사용하여 컴포넌트 속성을 로딩 — **interpolation** 이라 함.

app/app.component.ts **TypeScript**

```
...  
@Component({  
  selector: 'my-app',  
  template: '<h1>{{title}}</h1>'  
})  
class AppComponent {  
  title = 'Ultra Racing';  
}  
  
..
```

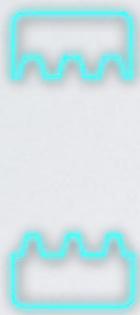


Interpolation : {{ }}

자바스크립트 객체를 화면에 출력하는 방법?

app/app.component.ts

TypeScript



```
...
@Component({
  selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
class AppComponent {
  title = 'Ultra Racing';
  carPart = {
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5
  };
}

...
```



Property Binding

모든 DOM 엘리먼트에 바인딩 가능하다.

```
<div hidden>secret</div>
```

```
<button disabled>Purchase</button>
```

```
<img alt="Image Description">
```

```
<div [hidden]="!user.isAdmin">secret</div>
```

```
<button [disabled] = "isDisabled" >Purchase</button>
```

```
<img [alt] ="image.description">
```

이벤트 바인딩

표준 DOM 이벤트를 괄호에 감싸서 사용.

일반적으로 **on** 접두어를 생략하여 사용한다.

```
<div (mouseover)="call()">
```

```
<input (blur)="call()">
```

```
<input (focus)="call()">
```

```
<input type="text" (keydown)="call()">
```

```
<form (submit)="call()">
```

Event 처리 : \$event 객체

\$event 객체를 메소드 내부로 전달 할 수 있다.

이벤트가 발생한 **target** 객체에 접근하려면 **\$event** 객체를 사용하면 된다.

```
<input type="text" (keydown)="showKey($event)">
```

```
showKey(event) {  
    alert(event.keyCode);  
}
```

```
<h2 (mouseover)="getCoord($event)">Hover Me</h2>
```

```
getCoord(event) {  
    console.log(event.clientX + ", " + event.clientY);  
}
```

Two-way binding : ngModel 문법

```
[ (ngModel) ]="<component property>"
```

주로 폼 필드에서 사용된다.

component properties:

```
[ (ngModel) ]="user.age"
```



```
[ (ngModel) ]="firstName"
```



에러 발생:

```
[ (ngModel) ]="fullName () "
```



Angular 데이터 바인딩의 종류



파이프

Angular Pipe

Pipe

- Template에서 출력을 변경하기 위해서 pipe를 사용
- Pipe는 데이터를 입력 받아 특정 포맷으로 출력해 주는 기능

API List

The screenshot shows a search interface with filters for 'TYPE: Pipe' and 'STATUS: All', and a 'Filter' search bar. Below the filters, the word 'common' is highlighted in blue. A grid of pipe components is listed under 'common':

| Type | Component Name |
|------|------------------------|
| P | AsyncPipe |
| P | DecimalPipe |
| P | DeprecatedDecimalPipe |
| P | I18nSelectPipe |
| P | PercentPipe |
| P | UpperCasePipe |
| P | CurrencyPipe |
| P | DeprecatedCurrencyPipe |
| P | DeprecatedPercentPipe |
| P | JsonPipe |
| P | SlicePipe |
| P | DatePipe |
| P | DeprecatedDatePipe |
| P | I18nPluralPipe |
| P | LowerCasePipe |
| P | TitleCasePipe |

사용자 정의 Pipe

- @Pipe - 메타데이터를 사용
- PipeTransform 인터페이스의 transform 메소드를 구현
- @angular/core 모듈의 Pipe를 import

```
import { Pipe, PipeTransform } from '@angular/core';
/*
 * Raise the value exponentially
 * Takes an exponent argument that defaults to 1.
 * Usage:
 *   value | exponentialStrength:exponent
 * Example:
 *   {{ 2 | exponentialStrength:10}}
 *   formats to: 1024
 */
@Pipe({name: 'exponentialStrength'})
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent: string): number {
    let exp = parseFloat(exponent);
    return Math.pow(value, isNaN(exp) ? 1 : exp);
  }
}
```

<https://angular.io/docs/ts/latest/guide/pipes.html>

라우터

Angular Router

라우터 주요 내용

- 라우터 설정
- 페이지 전환하며 데이터 전달하기
- 한 페이지에 내비게이션 영역(outlet) 여러 개 만들기
- 모듈 자연 로딩

Angular 라우팅

- 클라이언트 라우팅 : router 컴포넌트
 - 브라우저 URL에 따라 화면 컴포넌트를 정의 (화면을 구성)
- Setup : 라우터 파일을 각각 별도로 로딩 (모듈러 설계)
 - main HTML 파일 헤드 섹션 : <base href="/"> 추가
- Routes : 모든 애플리케이션은 하나의 라우터를 가짐
 - Routes 클래스 (배열과 유사) : URL과 컴포넌트 매핑 정보를 포함

```
import { Routes } from '@angular/router';
import { HomeComponent } from './home.component';

export const HomeRoutes: Routes = [
  {
    path: '',
    component: HomeComponent,
  },
];
```

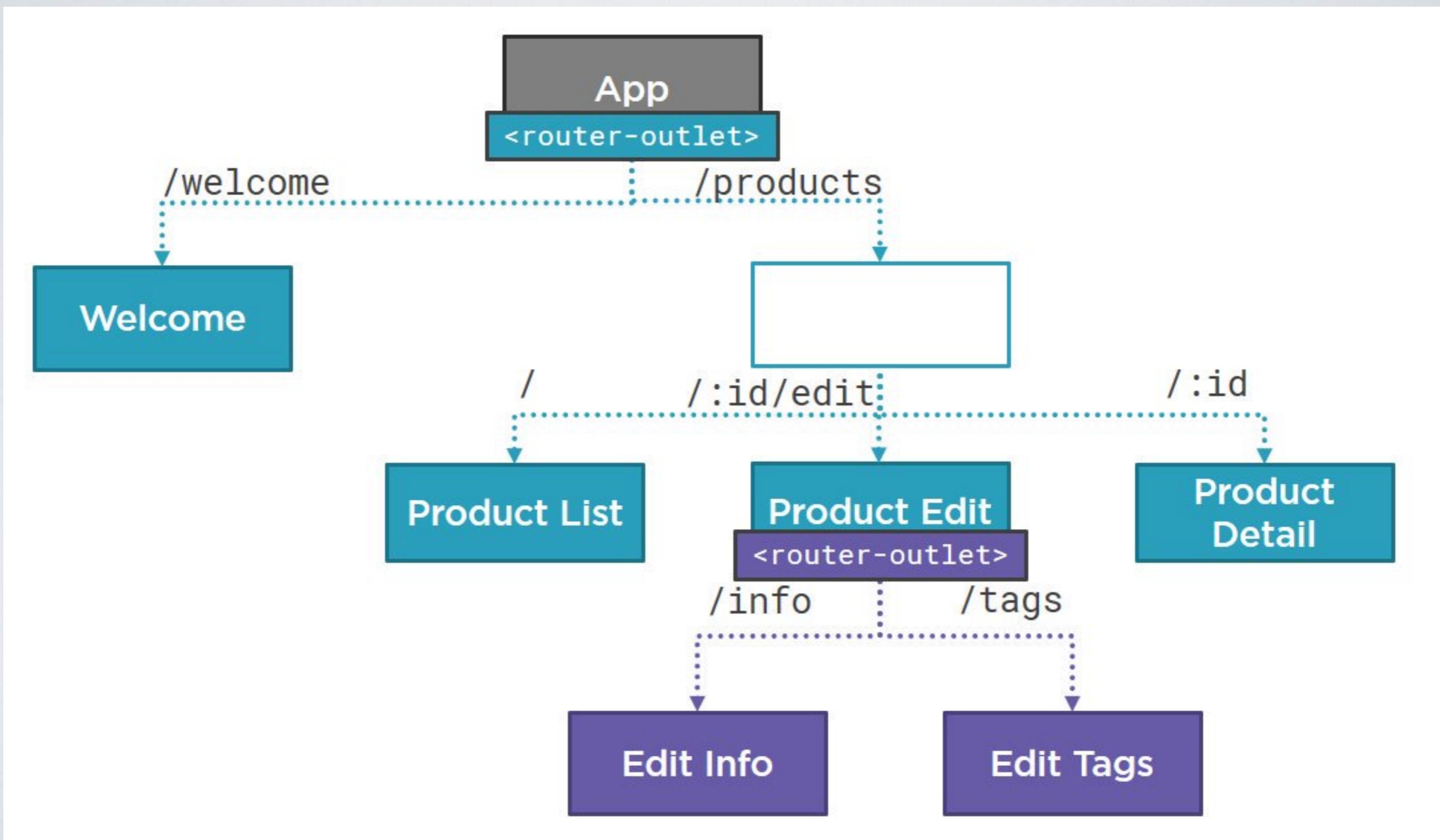
- Route outlet

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

@Component({
  selector: 'mean-app',
  template: '<h1>Application Title</h1>
            <br>
            <router-outlet></router-outlet>'
})
export class AppComponent { ... }
```
- 라우터 링크

```
<a [routerLink]="'[ '/about ' ]'">Some</a>
```

Routing



라우팅 구현

- AppRoutingModule 추가
 - 라우터는 별도의 top-level 모듈로 만든다.
 - src/app 디렉토리에 app-routing.module.ts 파일 추가

```
ng generate module app-routing --flat --module=app
```

- **--flat** : src/app 디렉토리에 파일을 만든다.
- **--module=app** : AppModule 에 등록

AppRoutingModule 추가

- RouterModule 을 @NgModule.exports 배열에 추가

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes} from '@angular/router';
import {MenuOneComponent} from './menu-one/menu-one.component';
import {MenuTwoComponent} from './menu-two/menu-two.component';

const routes: Routes = [
  { path: '', redirectTo: '/menu1', pathMatch: 'full' },
  { path: 'menu1', component: MenuOneComponent },
  { path: 'menu2', component: MenuTwoComponent }
];

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

라우트 추가

- Angular Routes 의 두가지 속성
 - path : 브라우저 주소창에서 URL 과 매칭
 - component : 해당 경로를 선택했을 때 생성되는 화면(컴포넌트)

```
const routes: Routes = [
  { path: 'menu1', component: MenuOneComponent },
  { path: 'menu2', component: MenuTwoComponent }
];
```

네비게이션 링크

- routerLink : 속성의 값으로 연결된 컴포넌트가 호출됨

```
<nav>
  <a routerLink="/menu1">메뉴 1</a>
  <a routerLink="/menu2">메뉴 2</a>
</nav>
```

Hash 주소 정책

<http://localhost:4200/#/menu2>

```
import ...

@NgModule({
  declarations: [
    AppComponent,
    MenuOneComponent,
    MenuTwoComponent
  ],
  imports: [
    BrowserModule,
    RouterModule,
    AppRoutingModule
  ],
  providers: [
    {provide: LocationStrategy, useClass: HashLocationStrategy}
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

navigate() 함수

- routerLink 대신 코드를 통해 라우팅 구현

```
<nav>
  <a routerLink="/menu1">메뉴 1</a>
  <a routerLink="/menu2">메뉴 2</a>
  <button (click)="goMenuThree()">메뉴 3</button>
</nav>
```

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

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

  constructor(private router: Router) {}

  goMenuThree() {
    this.router.navigate(['/menu3']);
  }
}
```

라우터로 데이터 전달

- URL 을 이용한 데이터 전달 localhost:4200/#/menu2/35

```
const routes: Routes = [
  { path: '', redirectTo: '/menu1', pathMatch: 'full' },
  { path: 'menu1', component: MenuOneComponent },
  { path: 'menu2/:id', component: MenuTwoComponent },
  { path: 'menu3', component: MenuThreeComponent }
];
```

```
<nav>
  <a routerLink="/menu1">메뉴 1</a>
  <a routerLink="/menu2/{{ testId }}">메뉴 2</a>
  <button (click)="goMenuThree()">메뉴 3</button>
</nav>
```

```
export class MenuTwoComponent implements OnInit {
  testId: string;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.testId = this.route.snapshot.paramMap.get('id');
  }
}
```

MODELS

Model 클래스

Model class

- 타입스크립트 장점인 객체지향적 기능을 활용
- 모델 클래스 생성

```
export class Product {  
  constructor(  
    public id: number,  
    public title: string,  
    public price: number,  
    public rating: number,  
    public description: string,  
    public categories: string[])  
  {}  
}
```

```
export class Product {  
  id: number;  
  title: string;  
  price: number;  
  rating: number;  
  description: string;  
  categories: string[];  
}
```

정적인 타입 (**static type**) : 컴파일 타임에 변수의 타입을 체크해서 더욱 정확한 값을 세팅하고 사용할 수 있게 해준다.

Model class 적용

```
export class ProductService {
  getProducts(): Product[] {
    return products.map(p => {
      return new Product(
        p.id, p.title, p.price, p.rating,
        p.description, p.categories);
    });
  }

  getProductById(productId: number): Product {
    return products.find(p => p.id === productId);
  }
}
```

```
const products: Product[] = [
  {
    'id': 0,
    'title': 'First Product',
    'price': 24.99,
    'rating': 4.3,
    'description': 'This is a ... elit.',
    'categories': ['electronics', 'hardware']
  },
  ...
  {
    'id': 5,
    'title': 'Sixth Product',
    'price': 54.99,
    'rating': 4.6,
    'description': 'This is a ... elit.',
    'categories': ['books']
  }
];
```

모델 객체를 도입

SERVICES

서비스 클래스 작성

DI(Dependency Injection)

Angular 서비스

- 서비스는 Angular 의 필수 요소
 - 클래스로 작성되며, 하나의 목적이나 기능을 포함
 - 컴포넌트에 포함되지 않는 기능을 포함
 - 예) 데이터 관리, 로깅, 애플리케이션 설정
 - Dependency Injection 을 통해 컴포넌트에 주입됨

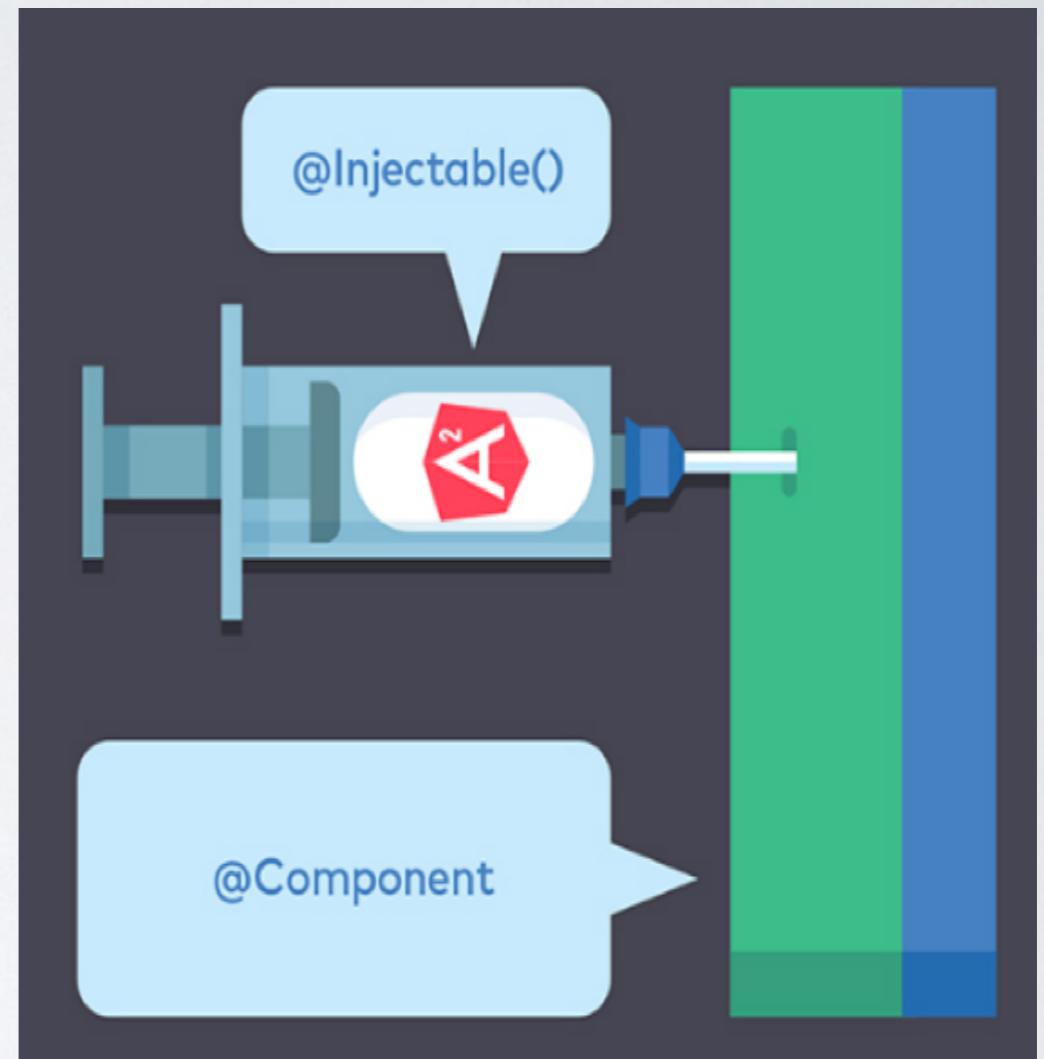
```
import { Component } from '@angular/core';
import { SomeService } from '../users/services/some.service';

@Component({
  selector: 'some-component',
  template: 'Hello Services',
  providers: [SomeService]
})
export class SomeComponent {
  user = null;
  constructor (private _someService: SomeService) {
    this.user = _someService.user;
  }
}
```

서비스는 컴포넌트 생성자를 통해 컴포넌트로 주입됨

Dependency Injection 사용 방법

- 컴포넌트 providers 에 등록되어 사용
- Service 사용하기 위한 세가지 절차 :
 - Service 에 @Injectable() 데코레이터 추가
 - app 모듈에 providers 로 등록
 - xxx.component.ts 에 의존성 주입



Service 객체 사용

- Data 를 Component 에 공급
- 컴포넌트들과 서버(데이터 공급자)의 중계자 역할
- 기능을 포함

Service 객체



some.component.ts



데이터 요청을 서비스 객체에 함



some-data.service.ts



실제 데이터를 가져옴



(server)

Dependancy Injection

- 의존성 주입(Dependency Injection, DI)
 - 프로그래밍에서 구성요소간의 의존 관계가 소스코드 내부가 아닌 외부의 설정파일 등을 통해 정의되게 하는 디자인 패턴 중의 하나
- 의존성 주입(Dependency Injection, DI)의 장점
 - 의존 관계 설정이 컴파일시점이 아닌 실행시점에 이루어져 모듈들간의 결합도를 낮출 수 있다.
 - 코드 재사용을 높여서 작성된 모듈을 여러 곳에서 소스코드의 수정 없이 사용할 수 있다.
 - mock 객체 등을 이용한 단위 테스트의 편의성을 높여준다.

서비스 객체는 주로 DI를 통해 사용됨

Angular 와 Dependency Injection

- 모듈 설정 내 'providers' 에 등록되어 사용

서비스 객체 - 주로 Injector 로 사용됨

Service 클래스 작성 및 사용하기 위한 세 가지 절차 :

1. Service를 생성하고 **@Injectable** 데코레이터를 추가.
2. app 모듈에 providers 로 등록.
3. some.component.ts에 의존성을 주입.

Angular 서비스 작성

- public/app/articles : article.service.ts

- rxjs/Observable 임포트
- @Injectable() 데코레이터 사용

```
import 'rxjs/Rx';
import {Observable} from 'rxjs/Observable';

import {Injectable} from '@angular/core';
import {Http, Headers, Request, RequestMethod, Response} from '@angular/http';

@Injectable()
export class ArticlesService {
  private _baseURL = 'api/articles';

  constructor (private _http: Http) {}

  create(article: any): Observable<any> {
    return this._http
      .post(this._baseURL, article)
      .map((res: Response) => res.json())
      .catch(this.handleError);
  }

  read(articleId: string): Observable<any> {
    return this._http
      .get(`${this._baseURL}/${articleId}`)
      .map((res: Response) => res.json())
      .catch(this.handleError);
  }

  update(article: any): Observable<any> {
    return this._http
      .put(`${this._baseURL}/${article._id}`, article)
      .map((res: Response) => res.json())
      .catch(this.handleError);
  }

  delete(articleId: any): Observable<any> {
    return this._http
      .delete(`${this._baseURL}/${articleId}`)
      .map((res: Response) => res.json())
      .catch(this.handleError);
  }

  list(): Observable<any> {
    return this._http
      .get(this._baseURL)
      .map((res: Response) => res.json())
      .catch(this.handleError);
  }

  private handleError(error: Response) {
    return Observable.throw(error.json().message || 'Server error');
  }
}
```

서버 연동

HttpClient 모듈

서버로 데이터 요청!



some.component.ts

서비스 객체에 데이터 요청



data.service.ts

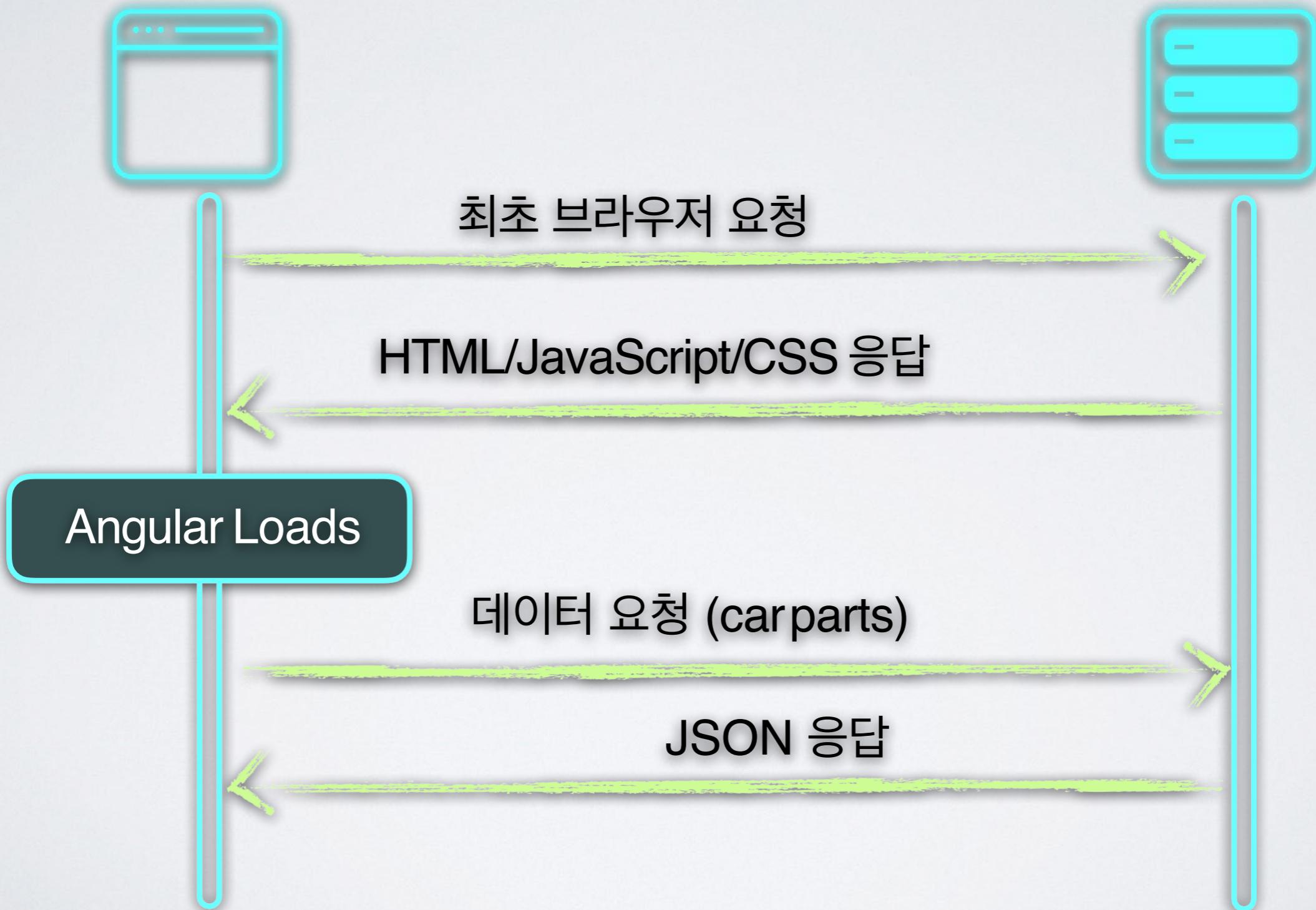
인터넷으로 데이터를 요청해서 가져옴



Internet

JSON 데이터 형태

서버로 데이터 요청!

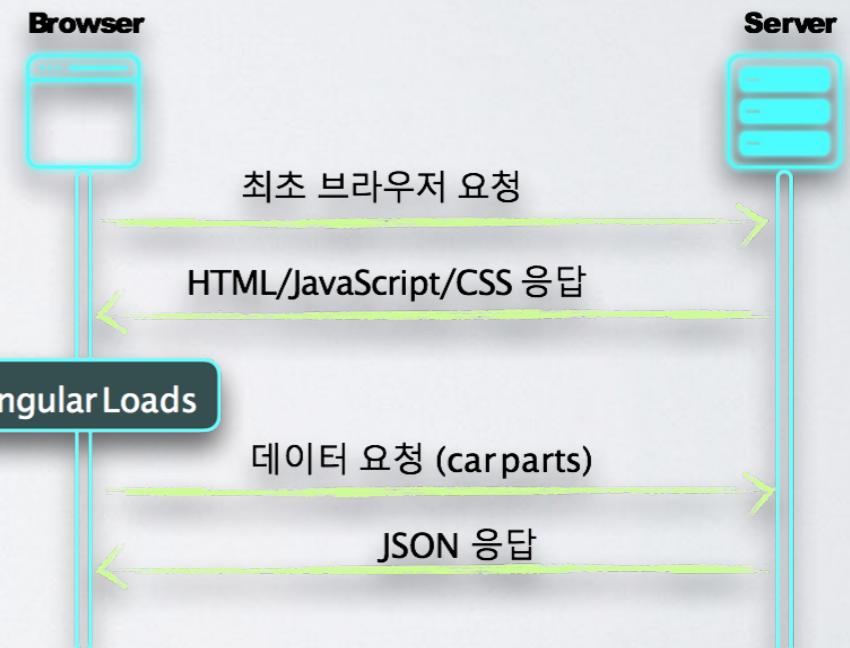


HttpClient

- http 모듈 : Angular 애플리케이션과 백엔드 API 와의 통신
 - SPA(Single Page App) : 최초 페이지 로딩 후 Ajax 를 통해 데이터만 가져옴
- Reactive Programming :

```
...  
"rxjs": "5.0.0-beta.12",  
...
```

```
signin(credentials: any): Observable<any> {  
  let body = JSON.stringify(credentials);  
  let headers = new Headers({ 'Content-Type': 'application/json' });  
  let options = new RequestOptions({ headers: headers });  
  
  return this.http.post(this._signinURL, body, options)  
    .map(res => this.user = res.json())  
    .catch(this.handleError)  
}
```



HTML FORMS

Template Forms vs. Reactive Forms

Angular Forms

- 템플릿 기반 Form vs. 반응형(Reactive) Form

I. 템플릿 기반 Form 방식

- ngModel Directive를 사용하여, 템플릿에 Form Model을 정의
- HTML 문법으로 Form을 구성하므로, 템플릿 기반 Form은 화면이 단순한 경우에 사용

2. 반응형(Reactive) Form 방식

- Typescript 코드로 Form Model을 정의하고, HTML에 각 항목을 연결
- Model은 @angular/form 모듈의 FormControl, FormGroup, FormArray 객체를 사용

Angular Forms

- Form API 활성화 하기
 - FormsModule 과 ReactiveFormsModule 추가 (app.module.ts)
 - 템플릿 기반 Form을 사용: FormsModule을 추가
 - 반응형 Form을 사용: ReactiveFormsModule을 추가

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

import { AppComponent } from './app.component';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';

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

ANGULAR 모듈

앱을 여러개의 모듈로 작성 가능

Angular 모듈

- Angular는 모듈로 구성된 모듈러 애플리케이션
 - 여러개의 모듈로 구성되고 각 모듈은 하나의 작업을 수행
 - 개발자는 필요한 모듈을 임포트 해서 사용
 - ES2015 모듈 문법을 사용함(ESM)

```
import { NgModule }      from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule }  from '@angular/router';

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

@NgModule({
  imports: [
    CommonModule,
    RouterModule.forRoot(AppRoutes),
  ],
  declarations: [
    AppComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Angular 모듈 설정

- public/app : articles.module.ts

```
import { NgModule }      from '@angular/core';
import { CommonModule }  from '@angular/common';
import { FormsModule }   from '@angular/forms';
import { RouterModule }  from '@angular/router';

import { ArticlesRoutes } from './articles.routes';
import { ArticlesComponent } from './articles.component';
import { CreateComponent } from './create/create.component';
import {ListComponent } from './list/list.component';
import { ViewComponent } from './view/view.component';
import { EditComponent } from './edit/edit.component';

@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    RouterModule.forChild(ArticlesRoutes),
  ],
  declarations: [
    ArticlesComponent,
    CreateComponent,
    ListComponent,
    ViewComponent,
    EditComponent,
  ]
})
export class ArticlesModule {}
```

ANGULAR MATERIAL

FIRESTORE 연동

파이어베이스 설정

- 파이어베이스 프로젝트 생성
- Firestore 데이터베이스 설정

파이어베이스 Angular 설정 - 1

```
$ npm install --save firebase @angular/fire
```

```
export const environment = {  
  production: false,  
  firebaseConfig : {  
    apiKey: "YOUR_API_KEY",  
    authDomain: "YOUR_AUTH_DOMAIN",  
    databaseURL: "YOUR_DATABASE_URL",  
    projectId: "YOUR_PROJECT_ID",  
    storageBucket: "YOUR_STORAGE_BUCKET",  
    messagingSenderId: "YOUR_MESSAGING_SENDER_ID"  
  }  
};
```

environments/environment.ts

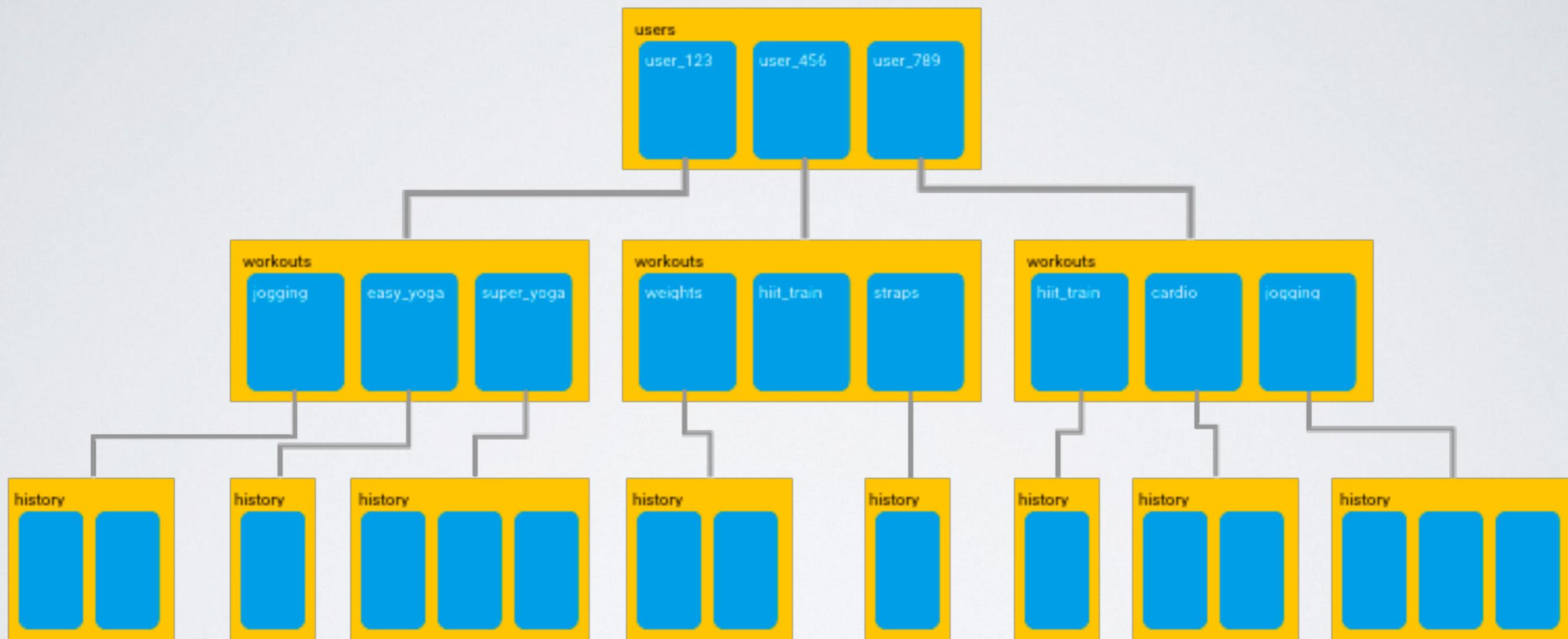
파이어베이스 Angular 설정 - 2

```
import { AngularFireModule } from '@angular/fire';
import { AngularFirestoreModule } from '@angular/fire/database';
import { environment } from '../environments/environment';

@NgModule({
    // [...]
    imports: [
        // [...]
        AngularFireModule.initializeApp(environment.firebaseioConfig),
        AngularFirestoreModule
    ],
})
```

src/app/app.module.ts

데이터 모델



모델 클래스

```
$ ng g class user --type=model
```

```
export class User {  
    id: string;  
    name: string;  
    age: number;  
    email: string;  
    hpNum: string;  
}
```

CRUD 작업 - service 클래스

```
import { Injectable } from '@angular/core';
import {AngularFirestore} from '@angular/fire/firestore';
import {User} from '../models/user';

@Injectable({
  providedIn: 'root'
})
export class UserService {

  constructor(private firestore: AngularFirestore) { }

  getUsers() {
    return this.firestore.collection('users').snapshotChanges();
  }

  createUser(user: User) {
    return this.firestore.collection('users').add(user);
  }

  updateUser(user: User) {
    delete user.id;
    this.firestore.doc('users/' + user.id).update(user);
  }

  deleteUser(userId: string) {
    this.firestore.doc('users/' + userId).delete();
  }
}
```

참고

- - Angular CHANGE LOG : <http://bit.ly/IH4QjC>
- - Angular 블로그 : <https://blog.angular.io/>
- - Angular 링크모음: <http://goo.gl/ctXUDo> (추가할 링크는 Pull Request)
- - Angular 질문/답변 <http://stackoverflow.com/search?q=angular>
- - Angular 컨퍼런스 영상 : <https://www.youtube.com/user/ngconfvideos>
- - Angular 레딧 <https://www.reddit.com/r/Angular2/>
- - Angular 튜토리얼 : <https://angular.io/tutorial>
- - 타입스크립트 2.1 <https://goo.gl/SOCxXP>
- - 타입스크립트 트위터 : <https://twitter.com/typescriptlang>
- - Angular 오픈소스 트렌딩 <https://github.com/trending/angular>

감사합니다.