

Angular 입문과 활용

2021.5.31 ~ 6.2

백명숙



과정개요

■ Angular Framework의 개념과 Architecture를 이해하고, Angular CLI을 사용하여 Angular Project 생성 방법을 살펴본다. Nodejs기반 Express를 이용하여 Restful 어플리케이션을 작성하여 Angular App과 연동한다. Angular 주요 컴포넌트(Service, Routing, HttpClient)를 사용하여 서버와 연동하는 Angular App을 작성해 본다.

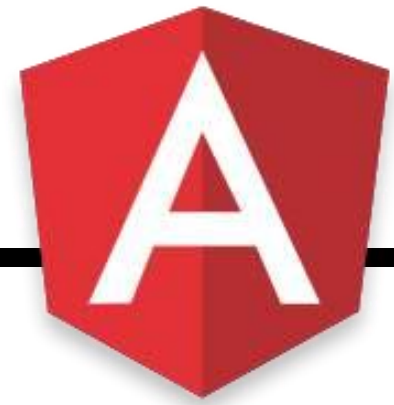


LO	커리큘럼
Angular 소개	<ul style="list-style-type: none"> - Angular 소개 - ECMAScript6 와 TypeScript - Angular CLI의 사용
Angular 아키텍처와 주요 컴포넌트들	<ul style="list-style-type: none"> - Module - Component와 Template - Data Binding, 이벤트 처리, Directive - Service - Angular HTTP - Angular Router - Angular Form
Spring Boot와 Angular 연동	<ul style="list-style-type: none"> - Spring Boot 어플리케이션 작성 - Tour of Hero 어플리케이션 작성 - User CRUD 어플리케이션 작성

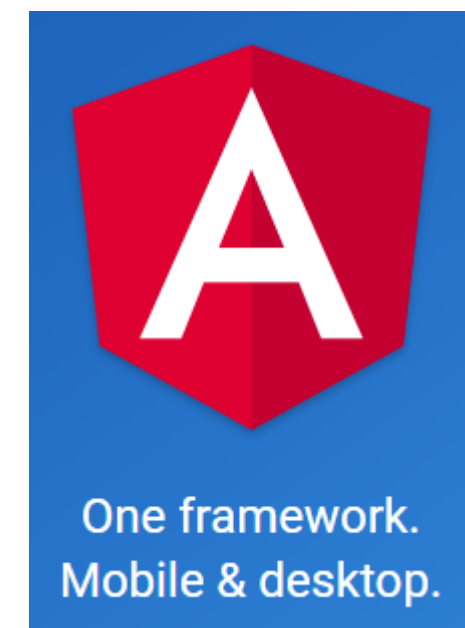
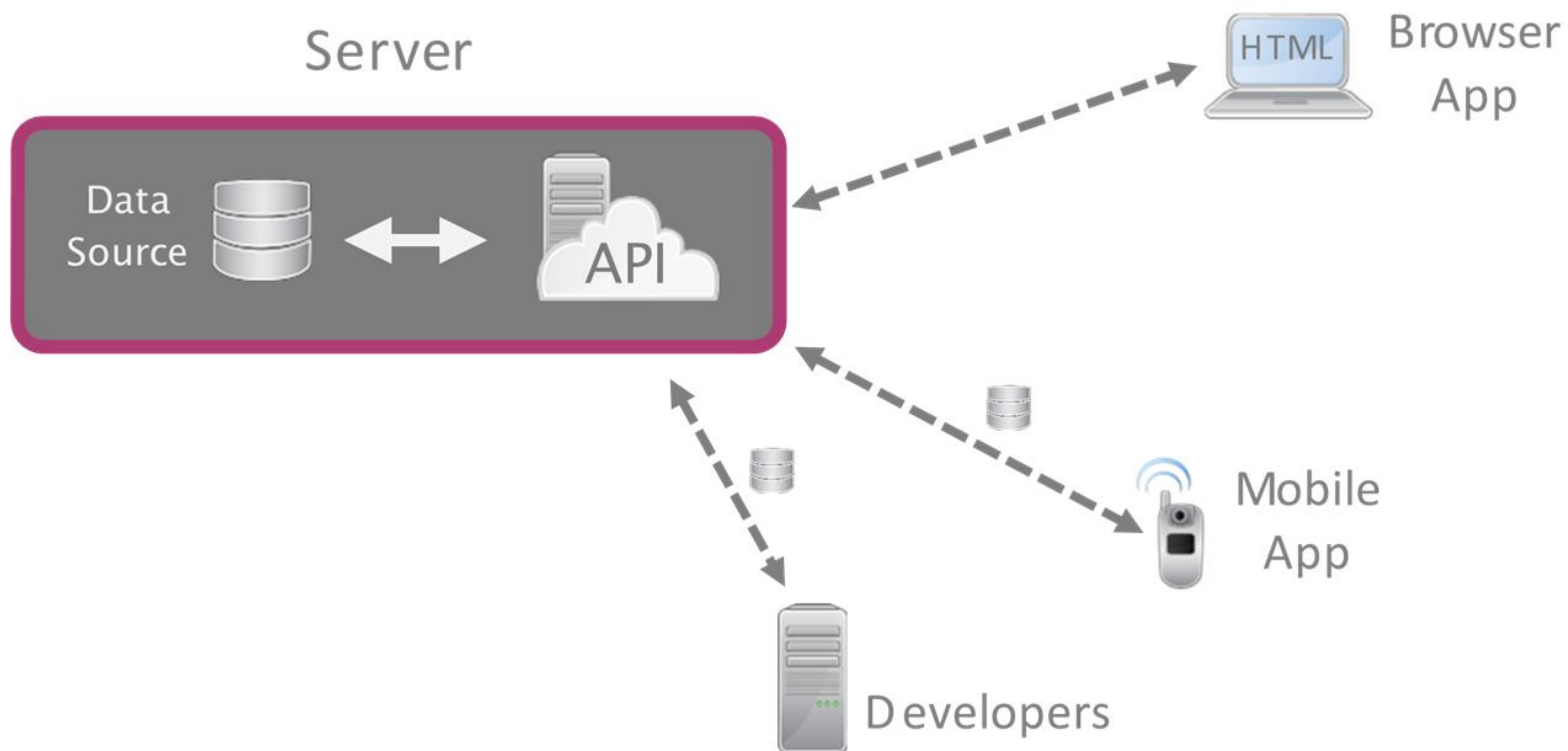
Angular 입문과 활용



Angular



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



Angular의 History

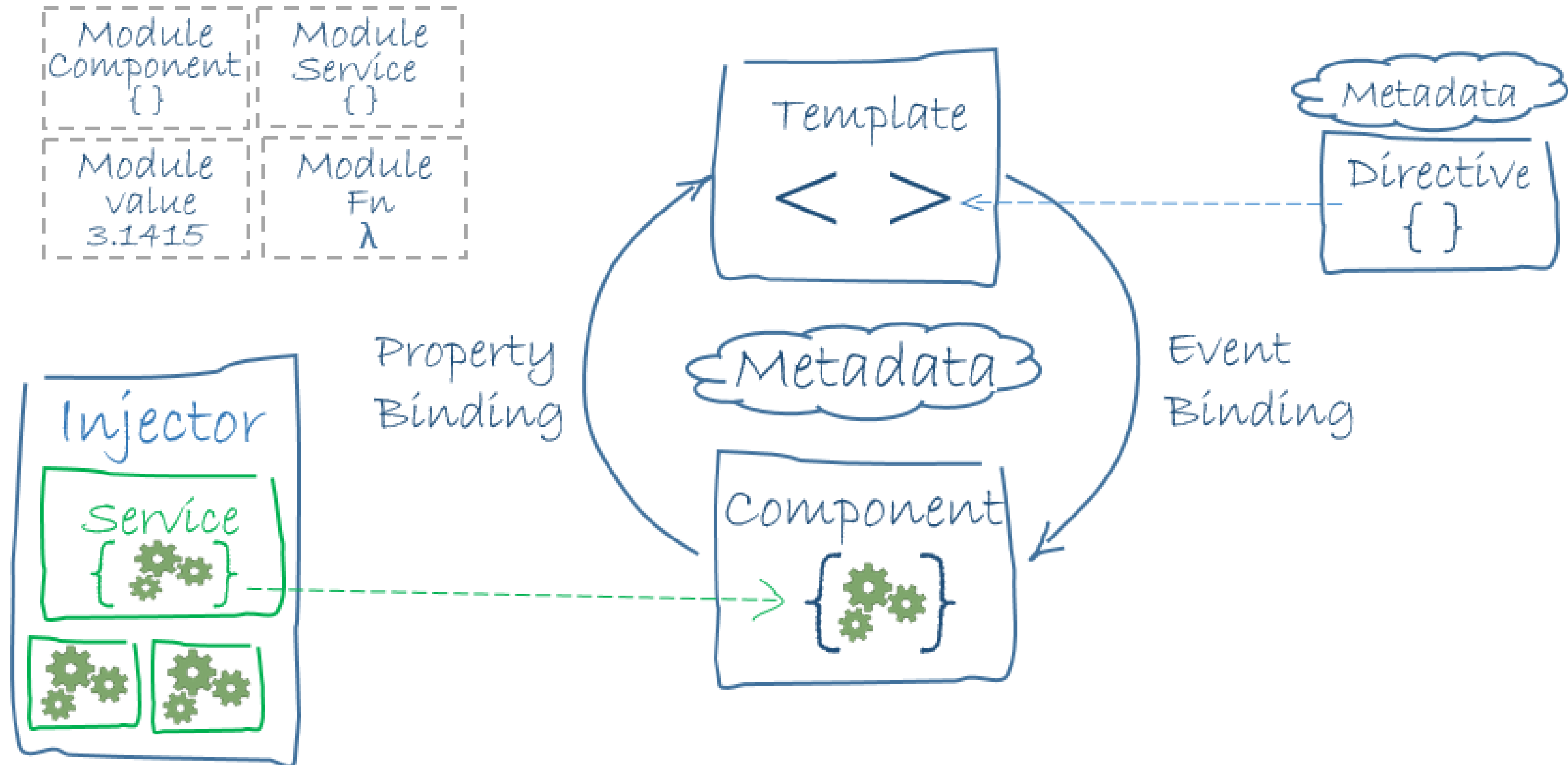


- 2014년 10월
ngEurope conference에서 첫소개
- 2015년 4월 30일 (Alpha Version), 12월 (Beta Version)
<https://angular.io/> 에서 다운로드 받을 수 있게 됨
- 2016년 5월
처음 angular2 release candidate가 되어 출시함
- 2017년 3월 : v2.4.10
- 2017년 10월 : v4.4.4
- 2018년 3월 : v5.2.7
- 2018년 6월 : v6.0.4
- 2019년 6월 : v7.2.11
- 2021년 5월 : v12.0.2

Angular의 아키텍처



- Component를 이용한 Template과 Service 로직 관리



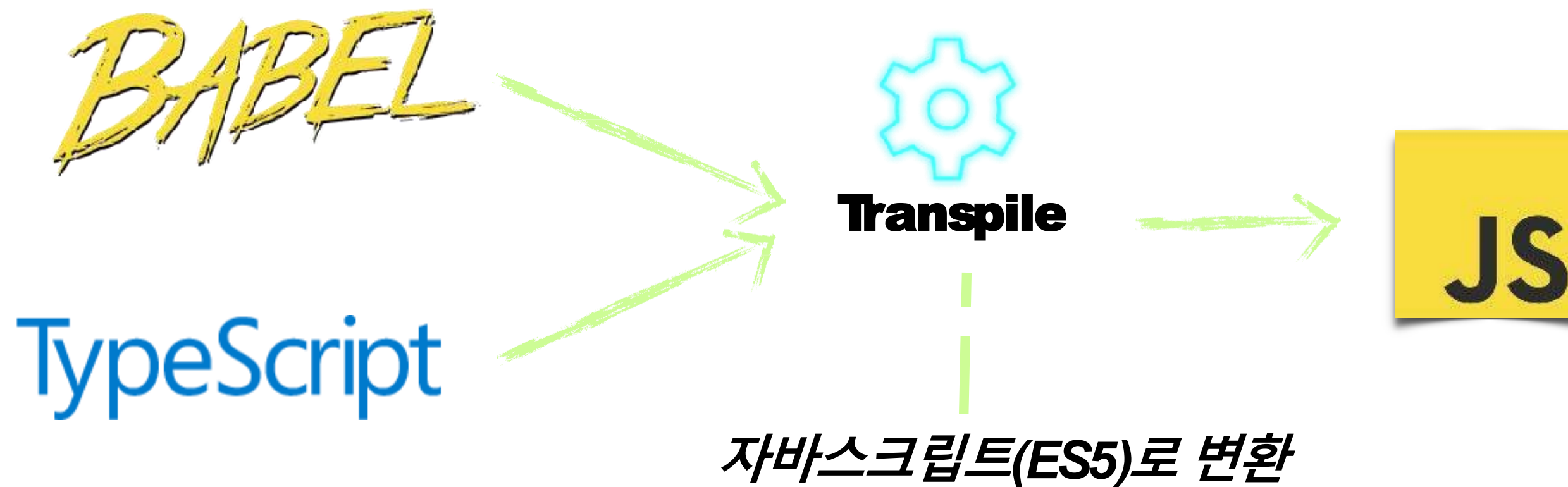
Angular에서 사용되는 언어는?



JavaScript

최신 자바스크립트 버전을 모든 브라우저가 지원하지 않음.

다음과 같은 방법으로 해결 가능:



Angular 개발을 위한 IDE



Angular IDE by Webclipse

Built first and foremost for Angular. Turnkey setup for beginners; powerful for experts.

IntelliJ IDEA

Capable and Ergonomic Java * IDE

Visual Studio Code

VS Code is a Free, Lightweight Tool for Editing and Debugging Web Apps.

Webstorm

Lightweight yet powerful IDE, perfectly equipped for complex client-side development and server-side development with Node.js

Angular 개발을 위한 TOOL



Angular CLI

The official Angular CLI makes it easy to create and develop applications from initial commit to production deployment. It already follows our best practices right out of the box!

Angular Universal

Server-side Rendering for Angular 2 apps.

Augury

A Google Chrome Dev Tools extension for debugging Angular 2 applications.

Celerio Angular Quickstart

Generate an Angular 2 CRUD application from an existing database schema

Codelyzer

A set of tslint rules for static code analysis of Angular 2 TypeScript projects.

Lite-server

Lightweight development only node server

Angular 프로젝트 환경 설정

- 1. Node.js 설치하기

Node.js 를 현재 기준 LTS 버전인 v10 버전을 설치하세요. [노드 공식 홈페이지](#)

- 2. Visual Studio Code 설치 및 Plug In 설치하기

VS Code 설치는 [Visual Studio Code](#) 에서 하실 수 있습니다.

Angular 프로젝트 환경 설정

- 3. Visual Studio Code Plug In 설치하기

1. Angular Essentials

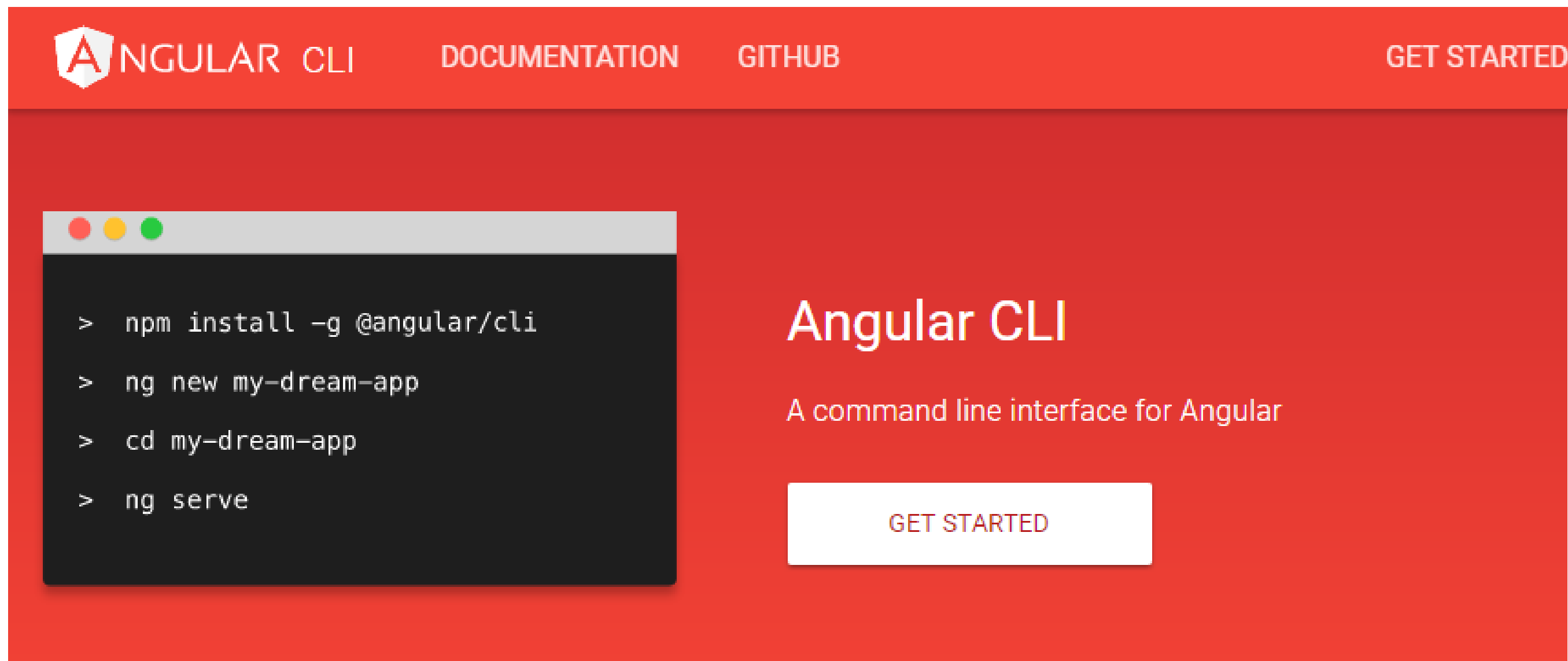
(<https://marketplace.visualstudio.com/items?itemName=johnpapa.angular-essentials>)

2. IntelliSense for CSS, SCSS class names

(<https://marketplace.visualstudio.com/items?itemName=gencer.html-slim-scss-css-class-completion>)

Angular CLI 개요

- [Angular CLI](#) 는 기본 구조, 컴포넌트 생성, 빌드, 유닛테스트, 개발서버, 배포를 관리 할 수 있도록 도와주는 커맨드 라인 인터페이스 입니다.



Angular CLI 개요

- 필수 사항

: Angular CLI는 Node 6.9.0이상, NPM 3 이상에서 가능합니다.

- Angular CLI 설치

`npm install -g @angular/cli`

- 프로젝트 생성 (ng new)

`ng new [프로젝트명]`

: 기본 구조, 컴포넌트 생성, 빌드, 유닛 테스트, 개발서버, 배포에 관련된 모든 의존성 라이브러리를 설치합니다.

Angular CLI 개요

- **개발서버 실행 (ng serve)**

`cd [프로젝트명]`

`ng serve`

: Webpack 빌드과정이 끝나면 <http://localhost:4200> 접속하면 개발서버를 확인할 수 있습니다.

- **컴포넌트, 디렉티브, 서비스, 파이프 생성(ng generate)**

`ng generate component [컴포넌트명]` 또는

`ng g component [컴포넌트명]`

: src/app/[컴포넌트명] 아래 4개의 파일이 생성됩니다.

[컴포넌트명].component.css

[컴포넌트명].component.html

[컴포넌트명].component.spec.ts

[컴포넌트명].component.ts

Angular CLI 개요

- **컴포넌트, 디렉티브, 서비스, 파이프 생성(ng generate)**

: 디렉티브, 서비스, 파이프, 클래스 등 같은 형태로 쉽게 폴더 구조 및 파일을 생성할 수 있습니다.

Scaffold	Usage
Component	<code>ng g component my-new-component</code>
Directive	<code>ng g directive my-new-directive</code>
Pipe	<code>ng g pipe my-new-pipe</code>
Service	<code>ng g service my-new-service</code>
Class	<code>ng g class my-new-class</code>
Guard	<code>ng g guard my-new-guard</code>
Interface	<code>ng g interface my-new-interface</code>
Enum	<code>ng g enum my-new-enum</code>
Module	<code>ng g module my-module</code>

Angular CLI 개요

- 빌드(**ng build**)

ng build

: dist/ 폴더에 실제 서비스를 위한 파일이 생성됩니다.

- 웹 문서 열기 (**ng doc**)

ng doc [keyword]

: CLI 를 통해서 Angular 2 웹 사이트 내에 API Reference 의 검색결과 페이지를 열어줍니다.

ECMAScript6와 TypeScript



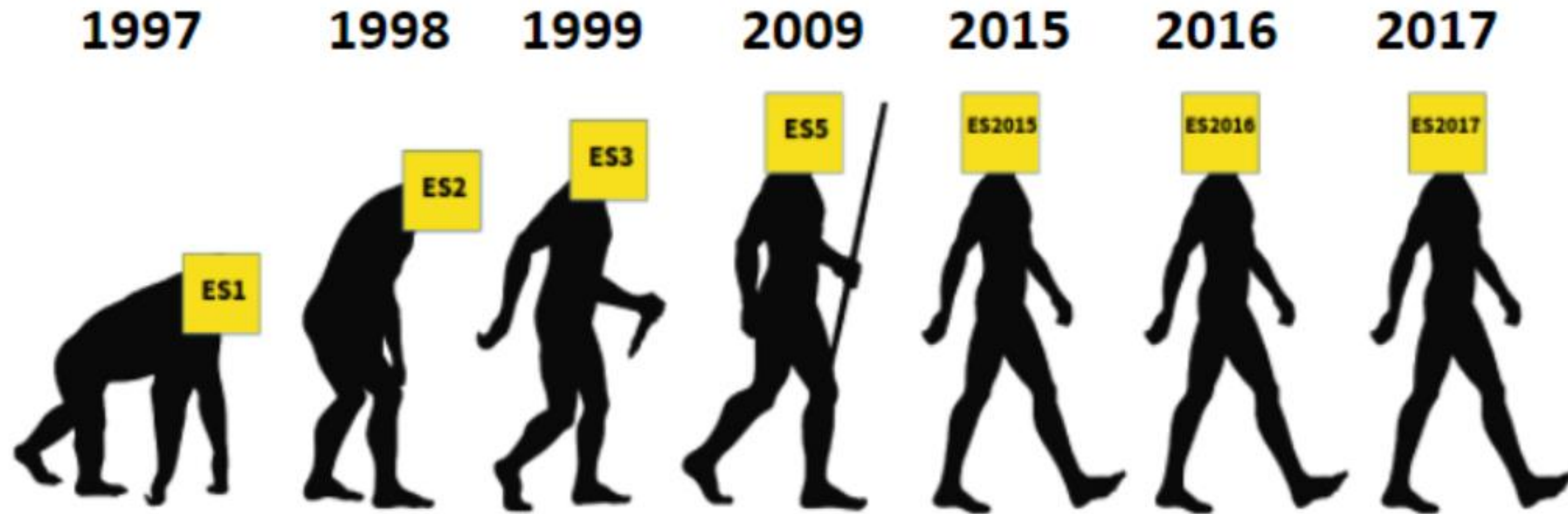
<http://www.typescriptlang.org>

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 (ECMA는 옛이름)
ECMA(European Computer Manufacturers Association)
C#, JSON, Dart을 포함한 많은 언어 표준을 관리함

ECMA Script (ES) 히스토리



ECMAScript 6 소개

- ECMA(European Computer Manufacturers Association) Script는 JavaScript 프로그래밍 언어를 정의하는 국제 표준의 이름입니다.
- ECMA의 Technical Committee39(TC39)에서 논의 되었습니다.
- 현재 사용하는 대부분의 JavaScript는 2009년에 처음 제정되어 2011년에 개정된 ECMAScript 5.1 표준에 기반하고 있습니다.
- 이후 클래스 기반 상속, 데이터 바인딩(Object.observe), Promise 등 다양한 요구사항들이 도출 되었고 그 결과 2015년 6월에 대대적으로 업데이트된 ECMAScript6 가 발표 되었고, 매년 표준을 업데이트하는 정책에 따라 2016년 6월에 ECMAScript7 까지 발표 되었습니다.
- ES6 = ECMAScript6 = ECMAScript 2015 = ES2015

ES6

- var / let, const

- const는 ES6에 도입된 키워드로서, 한번 선언하고 바뀌지 않는 값을 설정 할 때 사용됩니다.
변경 될 수 있는 값은 let 을 사용하여 선언합니다.
- var는 scope이 함수 단위이고, 반면 const와 let은 scope이 블록 단위 입니다.
- ES6 에서는 var 를 사용하지 않고, 값을 선언 후 바뀌야 할 땐 let, 그리고 바뀌지 않을 땐 const를 사용하면 됩니다.

src/App.js

```
function foo() {  
  var a = 'hello';  
  if (true) {  
    var a = 'bye';  
    console.log(a); // bye  
  }  
  console.log(a); // bye  
}
```

src/App.js

```
function foo() {  
  let a = 'hello';  
  if (true) {  
    let a = 'bye';  
    console.log(a); // bye  
  }  
  console.log(a); // hello  
}
```

ES6 축약코딩기법

• 1. 삼항 조건 연산자 (The Ternary Operator)

기존

```
const x = 20;
let answer;
if (x > 10) {
  answer = 'greater than 10';
} else {
  answer = 'less than 10';
}
```

축약기법

```
const answer = x > 10 ? 'greater than 10' : 'less than 10';
```

React에서 사용

```
//react에서 특정 버튼을 state 값에 따라 보여지게 할 경우에 이렇게 사용할 수 있음
{editable ? (
  <a onClick={() => this.save(record.key)}> </a>
) : (
  <a onClick={() => this.edit(record.key)}> </a>
)}
```


ES6 축약코딩기법

• 2. 간략 계산법 (Short-circuit Evaluation)

- 기존의 변수를 다른 변수에 할당하고 싶은 경우, 기존 변수가 null, undefined 또는 empty 값이 아닌 것을 확인해야 합니다. (위 세가지 일 경우 예러가 뜹니다) 이를 해결 하기 위해서 긴 if문을 작성 하거나 축약 코딩으로 한 줄에 끝낼 수 있습니다

기존

```
if (variable1 != null || variable1 !== undefined || variable1 !== '') {  
    let variable2 = variable1;  
}
```

축약기법

```
const variable2 = variable1 || 'new';
```

Console에서 확인

```
let variable1;  
let variable2 = variable1 || '';  
console.log(variable2 === ''); //print true  
  
let variable3 = 'foo';  
let variable4 = variable3 || 'foo';  
console.log(variable4 === 'foo'); //print true
```

ES6 축약코딩기법

• 3. 변수 선언

- 함수를 시작하기 전 먼저 필요한 변수들을 선언하는 것은 현명한 코딩 방법입니다. 축약 기법을 사용하면 여러 개의 변수를 더 효과적으로 선언함으로 시간과 코딩 스페이스를 줄일 수 있습니다.

기존

```
let x;  
let y;  
let z = 3;
```

축약기법

```
let x,y,z = 3;
```

ES6 축약코딩기법

• 4. For 루프

기존

```
for (let i=0; i < msgs.length; i++)
```

축약기법

```
for (let value of msgs)
```

Array.forEach 축약기법

```
function logArrayElements(element, index, array) {  
    console.log('a[' + index + '] = ' + element);  
}  
[2,5,9].forEach(logArrayElements);  
//a[0] = 2  
//a[1] = 5  
//a[2] = 9
```

ES6 축약코딩기법

• 5. 간략 계산법 (Short-circuit Evaluation)

- 기본 값을 부여하기 위해 파라미터의 null 또는 undefined 여부를 파악할 때 short-circuit evaluation 방법을 이용해서 한줄로 작성하는 방법이 있습니다.
- Short-circuit evaluation 이란?
두가지의 변수를 비교할 때, 앞에 있는 변수가 false 일 경우 결과는 무조건 false 이기 때문에 뒤의 변수는 확인하지 않고 return 시키는 방법.
- 아래의 예제에서는 process.env.DB_HOST 값이 있을 경우 dbHost 변수에 할당하지만, 없으면 localhost를 할당합니다.

기존

```
let dbHost;  
if (process.env.DB_HOST) {  
  dbHost = process.env.DB_HOST;  
} else {  
  dbHost = 'localhost';  
}
```

축약기법

```
Const dbHost = process.env.DB_HOT || 'localhost';
```

ES6 축약코딩기법

• 6. 객체 프로퍼티

- 객체 리터럴 표기법은 자바스크립트 코딩을 훨씬 쉽게 만들어 줍니다. 하지만 ES6는 더 쉬운 방법을 제안합니다. 만일 프로퍼티 이름이 key 이름과 같을 경우, 축약 기법을 활용할 수 있습니다.

기존

```
const obj = {x:x, y:y}
```

축약기법

```
const obj = {x, y}
```


ES6 축약코딩기법

• 7.Arrow(화살표) 함수

- Arrows 는 => 함수를 짧게 표현하는 방식(람다식)을 말합니다.
- 이는 C#, Java8 이나 CoffeeScript와 문법적으로 유사합니다.

기존

```
function sayHello(name) {  
  console.log('Hello', name);  
}  
  
setTimeout(function() {  
  console.log('Loaded')  
}, 2000);  
  
list.forEach(function(item) {  
  console.log(item);  
});
```

축약기법

```
sayHello = name => console.log('Hello', name);  
  
setTimeout(() => console.log('Loaded'), 2000);  
  
list.forEach(item => console.log(item));
```

ES6: Arrow Function

- 7.1 Arrow(화살표) 함수
- Arrow 함수와 기존 함수는 참조하고 있는 `this`의 값이 다릅니다.

기존 함수

```
function BlackDog() {  
  this.name = '흰둥이';  
  return {  
    name: '검둥이',  
    bark: function() {  
      console.log(this.name + ' 멍멍!');  
    }  
  }  
}  
  
const blackDog = new BlackDog();  
blackDog.bark(); // 검둥이 멍멍!
```

arrow 함수

```
function whiteDog() {  
  this.name = '흰둥이';  
  return {  
    name: '검둥이',  
    bark: () => {  
      console.log(this.name + ' 멍멍!');  
    }  
  }  
}  
  
const whiteDog = new whiteDog();  
whiteDog.bark(); // 흰둥이 멍멍!
```

ES6 축약코딩기법

• 8. 묵시적 반환(Implicit Return)

- return 은 함수 결과를 반환 하는데 사용되는 명령어 입니다.
- 한 줄로만 작성된 arrow 함수는 별도의 return 명령어가 없어도 자동으로 반환 하도록 되어 있습니다.
- 다만, 중괄호({})를 생략한 함수여야 return 명령어도 생략 할 수 있습니다
- 한 줄 이상의 문장(객체 리터럴)을 반환 하려면 중괄호({})대신 괄호(())를 사용해서 함수를 묶어야 합니다.
이렇게 하면 함수가 한 문장으로 작성 되었음을 나타낼 수 있습니다.

기존

```
function calcCircumference(diameter) {  
  return Math.PI * diameter  
}
```

축약기법

```
calcCircumference = diameter => (  
  Math.PI * diameter;  
)
```

ES6 축약코딩기법

• 9. 파라미터 기본 값 지정하기(Default Parameter Values)

- 기존에는 if 문을 통해서 함수의 파라미터 값에 기본 값을 지정해 줘야 했습니다. ES6에서는 함수 선언문 자체에서 기본값을 지정해 줄 수 있습니다.

기존

```
function volume(l, w, h) {  
  if (w === undefined)  
    w = 3;  
  if (h === undefined)  
    h = 4;  
  return l * w * h;  
}
```

축약기법

```
volume = (l, w = 3, h = 4) => (l * w * h);  
volume(2) //output: 24
```

ES6 축약코딩기법

• 10. 템플릿 리터럴 (Template Literals)

- 백틱(backtick) 을 사용해서 스트링을 감싸고, `${}`를 사용해서 변수를 담아 주면 됩니다.

기존

```
const welcome = 'You have logged in as ' + first + ' ' + last + '.'  
const db = 'http://' + host + ':' + port + '/' + database;
```

축약기법

```
const welcome = `You have logged in as ${first} ${last}`;  
const db = `http://${host}:${port}/${database}`;
```

ES6 축약코딩기법

• 11. 비구조화 할당 (Destructuring Assignment)

- 데이터 객체가 컴포넌트에 들어가게 되면, unpack 이 필요합니다.

Destructuring Assignment 기본 문법

```
let a, b, rest;  
[a, b] = [1, 2];  
[a, b, ...rest] = [10, 20, 3, 4, 5];  
  
let foo = ["one", "two", "three"];  
let [foo1, foo2, foo3] = foo;
```

값 교환 (Swapping)

```
let a = 1;  
let b = 3;  
  
[a, b] = [b, a];
```

객체 분해

```
let obj = {p: 42, q: true};  
let {p, q} = obj;
```

기존

```
const observable = require('mobx/observable');  
const action = require('mobx/action');  
const runInAction = require('mobx/runInAction');  
  
const store = this.props.store;  
const form = this.props.form;  
const loading = this.props.loading;  
const errors = this.props.errors;  
const entity = this.props.entity;
```



축약기법

```
import { observable, action, runInAction } from 'mobx';  
const { store, form, loading, errors, entity } = this.props;
```

ES6 축약코딩기법

• 12. 전개연산자 (Spread Operator) #1

- ES6에서 소개된 전개 연산자는 자바스크립트 코드를 더 효율적으로 사용 할 수 있는 방법을 제시합니다. 간단히는 배열의 값을 변환하는데 사용할 수 있습니다. 전개 연산자를 사용하는 방법은 점 세개(...)를 붙이면 됩니다.

기존

```
// joining arrays
const odd = [1, 3, 5];
const nums = [2, 4, 6].concat(odd);

// cloning arrays
const arr = [1, 2, 3, 4];
const arr2 = arr.slice();
```

축약기법

```
// joining arrays
const odd = [1, 3, 5 ];
const nums = [2, 4, 6, ...odd];
console.log(nums); // [ 2, 4, 6, 1, 3, 5 ]

// cloning arrays
const arr = [1, 2, 3, 4];
const arr2 = [...arr];
```


ES6 축약코딩기법

• 12. 전개연산자 (Spread Operator) #2

- `concat()` 함수와는 다르게 전개 연산자를 이용하면 하나의 배열을 다른 배열의 아무 곳이나 추가할 수 있습니다.

축약기법

```
const odd = [1, 3, 5];  
const nums = [2, ...odd, 4, 6];
```

- 전개 연산자는 ES6의 구조화 대입법(destructuring notation)와 함께 사용할 수도 있습니다.

축약기법

```
const { a, b, ...z } = { a: 1, b: 2, c: 3, d: 4 };  
console.log(a) // 1  
console.log(b) // 2  
console.log(z) // { c: 3, d: 4 }
```

ES6 축약코딩기법

• 13. 필수(기본) 파라미터 (Mandatory Parameter)

- 기본적으로 자바스크립트는 함수의 파라미터 값을 받지 않았을 경우, undefined로 지정합니다. 다른 언어들은 경고나 에러 메시지를 나타내기도 하죠. 이런 기본 파라미터 값을 강제로 지정하는 방법은 if 문을 사용해서 undefined일 경우 에러가 나도록 하거나, 'Mandatory parameter shorthand'을 사용하는 방법이 있습니다.

기존

```
function foo(bar) {  
  if(bar === undefined) {  
    throw new Error('Missing parameter!');  
  }  
  return bar;  
}
```

축약기법

```
let mandatory = () => {  
  throw new Error('Missing parameter!');  
}  
  
let foo = (bar = mandatory()) => {  
  return bar;  
}
```

• 14 Object.assign() 함수

- Object.assign() 함수는 첫 번째 Object에 그 다음 Object (들)을 병합해 줍니다.

Object.assign(target, ...sources)

- object.assign 메서드의 첫 번째 인자는 target 객체입니다. 두 번째 인자부터 마지막 인자까지는 source 객체입니다. source 객체는 target 객체에 병합됩니다. 그리고 리턴값으로 타겟오브젝트를 반환합니다.

Object.assign()

//target에 빈 객체를 주고 source 객체를 한 개만 주면 해당 source 객체를 복제하는 것이 됩니다.

```
var obj = {a:1};  
var copy = Object.assign({}, obj);  
console.log(copy); // {a: 1}
```

//obj1, obj2, obj3를 각각 {}안에 병합합니다.

```
var obj1 = {a:1};  
var obj2 = {b:2};  
var obj3 = {c:3};  
var newObj = Object.assign({}, obj1, obj2, obj3);  
console.log(newObj); // {a: 1, b: 2, c: 3}
```

ES6 축약코딩기법

• 15 Promise 객체

- “A promise is an object that may produce a single value some time in the future”
- Promise는 자바스크립트 비동기 처리에 사용되는 객체입니다. 여기서 자바스크립트의 비동기 처리란 ‘특정 코드의 실행이 완료될 때까지 기다리지 않고 다음 코드를 먼저 수행하는 자바스크립트의 특성’을 의미합니다.
- `new Promise(function(resolve, reject) { ... });`
- Promise 객체를 생성하면 `resolve`와 `reject`의 함수를 전달받는다.
- 작업이 성공하면 `resolve`함수를 호출하여 `resolve`의 인자값을 `then`으로 받게 되고
- 작업에 실패하면 `reject` 함수를 호출하여 `reject`의 인자값을 `catch`로 받게 된다.
- 성공, 실패 여부에 관계없이 항상 처리 되게 하려면 `finally`로 받아서 처리할 수도 있다.

Promise

```
new Promise(function (resolve, reject) {  
  }).then(function (resolve) {  
    //resolve 값 처리  
  }).catch(function (reject) {  
    //reject 값 처리  
  }).finally(function(){  
    //항상 처리  
  });
```

ES6 축약코딩기법

• 16. import / export

- ES6(ECMA2015)부터는 import / export 라는 방식으로 모듈을 불러오고 내보낸다.
- Export는 내부 스크립트 객체를 외부 스크립트로 모듈화 하는 것이며, export 하지 않으면 외부 스크립트에서 import를 사용할 수 없다.

app.js (export)

```
export const myNumbers = [1, 2, 3, 4];
const animals = ['Panda', 'Bear', 'Eagle'];

export default function myLogger() {
  console.log(myNumbers, pets);
}

export class Alligator {
  constructor() {
    // ...
  }
}
```

import

```
//Importing with alias
import myLogger as Logger from 'app.js';

//Importing all exported members
import * as Utils from 'app.js';

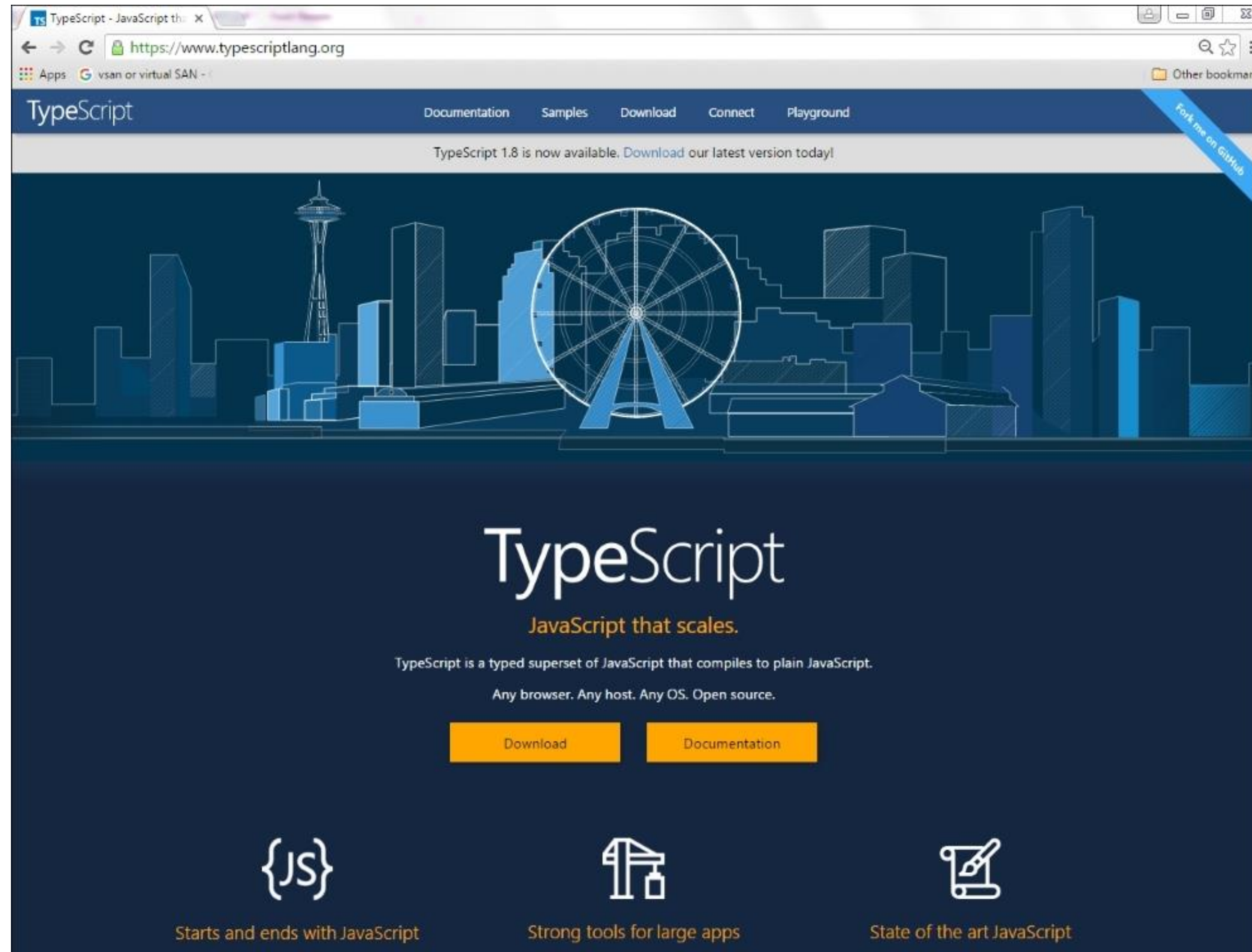
Utils.myLogger();

//Importing a module with a default member
import myLogger from 'app.js';

import myLogger, { Alligator, myNumbers } from 'app.js';
```

- <https://www.digitalocean.com/community/tutorials/js-modules-es6>

TypeScript ■ www.typescriptlang.org



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 2.0 발표
- 2017년 6월 2.5
- 2018년 1월 2.6
- 2018년 6월 2.9
- 2019년 현재 3.5

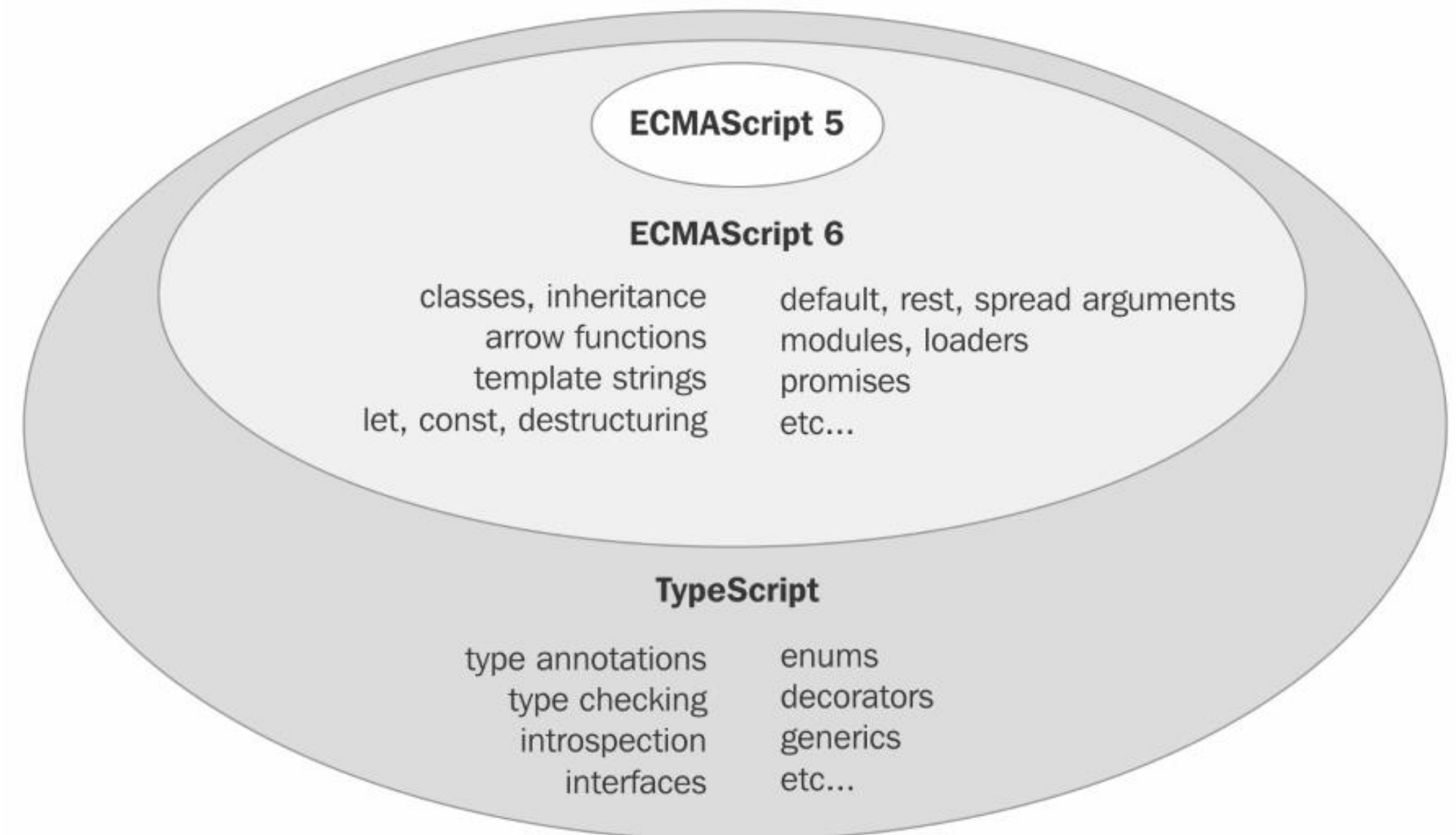


TypeScript의 개요

TypeScript 또한 자바스크립트 대체 언어의 하나로써 자바스크립트(ES5)의 Superset(상위확장)이다. C#의 창시자인 덴마크 출신 소프트웨어 엔지니어 Anders Hejlsberg가 개발을 주도한 TypeScript는 Microsoft에서 2012년 발표한 오픈소스로, 정적 타이핑을 지원하며 ES6(ECMAScript 2015)의 클래스, 모듈 등과 ES7의 Decorator 등을 지원한다.



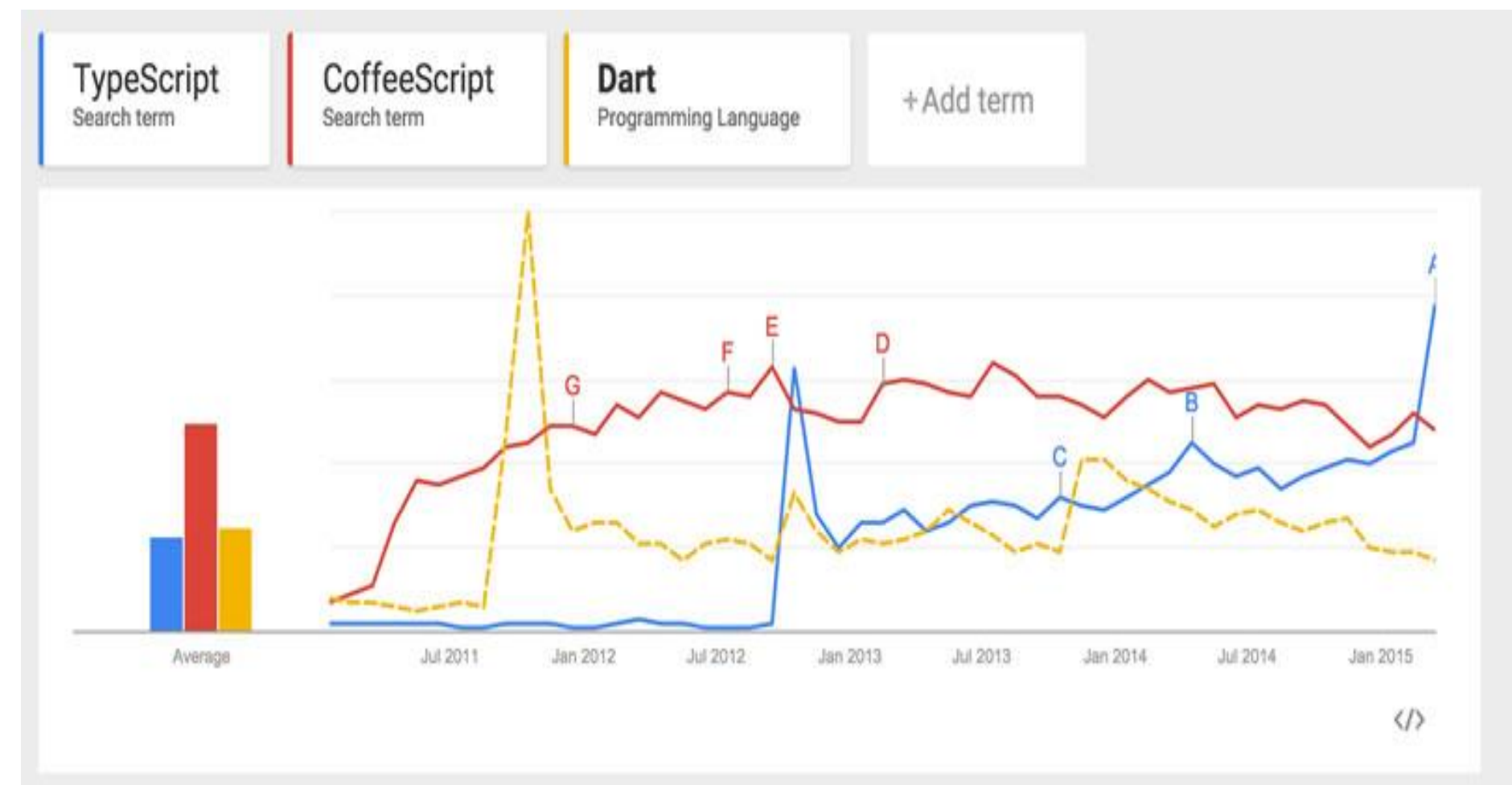
<http://www.typescriptlang.org>



TypeScript의 개요



- 자바스크립트 → ES2015(ES6) → ES2016(ES7) → TypeScript
- AngularJS의 후속 버전인 Angular의 TypeScript 정식 채용으로 TypeScript에 대한 관심이 커졌다.
- 자바스크립트(ES6, ES7) 기능에 타입스크립트의 필요한 기능을 추가 하면 된다. (새로운 언어가 아님)
 - 클래스 관련기능
 - 정적 타이핑 (static typing)



TypeScript 특징



- 컴파일 언어, 정적 타입 언어이다. JS는 인터프리터 언어지만, TypeScript는 컴파일 언어로 코드 수준에서 미리 타입을 체크하여 오류를 체크해낸다. 단 전통적인 컴파일 언어와는 다르게, 링킹(Linking) 과정이 생략되어 있다.
- 타입 기반 언어로써
 - ✓ 타입 스크립트 = Javascript + Type
 - ✓ 컴파일 단계에서 타입 오류를 잡아낼 수 있고, 코드 어시스트 기능도 지원받을 수 있다.
 - ✓ 이것을 통해 암묵적 형변환, 호이스팅 문제를 해결할 수 있다.
 - ✓ JS의 var와 같은 자료형 대신, string, number 같은 자료형을 지정함으로써 안정성을 확보한다.
 - ✓ tsc는 컴파일 과정에서 타입 검사를 통해 어려없이 안정성이 확보되면 타입을 제거하고 최종적으로 JS코드를 생성한다.

TypeScript의 차별점



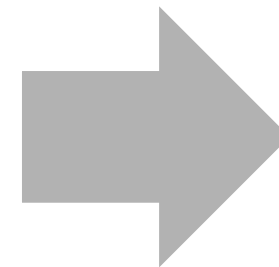
- 명시적인 자료형 선언가능

JS

```
var a="10";  
var b=10;  
var sum=a+b;  
console.log(sum);
```

결과 : 1010

해석: 20이 출력 되길 기대했
지만, 자바스크립트의 암묵적
(implicit) 형변환은 예측할 수
없는 오류를 만들어 냄.



TS

```
function add_error(){  
  let a: string = "10";  
  let b: number = 10;  
  
  let sum: number = a + b;  
  //error!  
  console.log(sum);  
}  
add_error();
```

결과 : 타입이 다른 경우 더하기 에
러가 발생함! (오류가능성 사전제
거)

TypeScript의 차별점



- 명시적인 자료형 선언가능

```
function add(){  
  let a: number = 10;  
  let b: number = 10;  
  let sum: number = a + b;  
  console.log(sum);  
}  
  
add();
```

TS

결과 : 20

- 명시적인 자료형 선언으로 가독성이 향상됨
 - 자료형을 명시적으로 정의함으로써 오류를 사전에 감지함
- 예 : 자료형이 다르면 비교나 할당이 불가능함

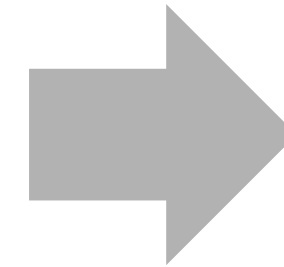
TypeScript의 차별점



- 객체지향 프로그래밍 지원

JS

```
var car = (function(){  
    function car(){};  
    car.prototype.getNumTier=function(){};  
    ...  
})();
```



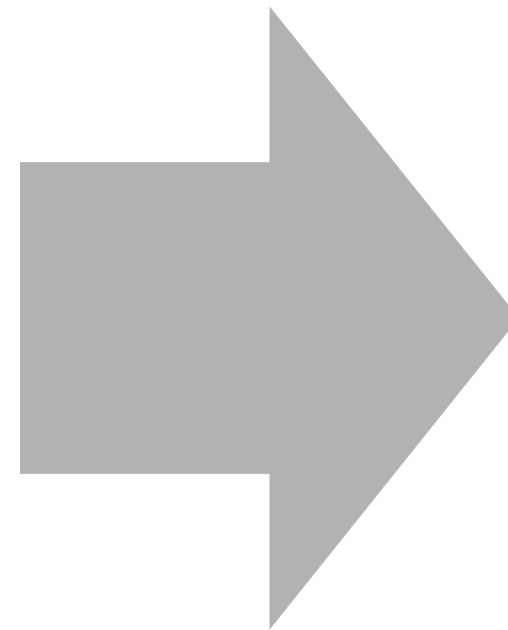
TS

```
class car {  
    numTier: number;  
    constructor(){}  
    getNumTier(){}  
    ...  
}
```

트랜스파일러 : tsc



- TSC는 타입스크립트를 자바스크립트로 변환(transpiling)해주는 도구이다.



트랜스파일링

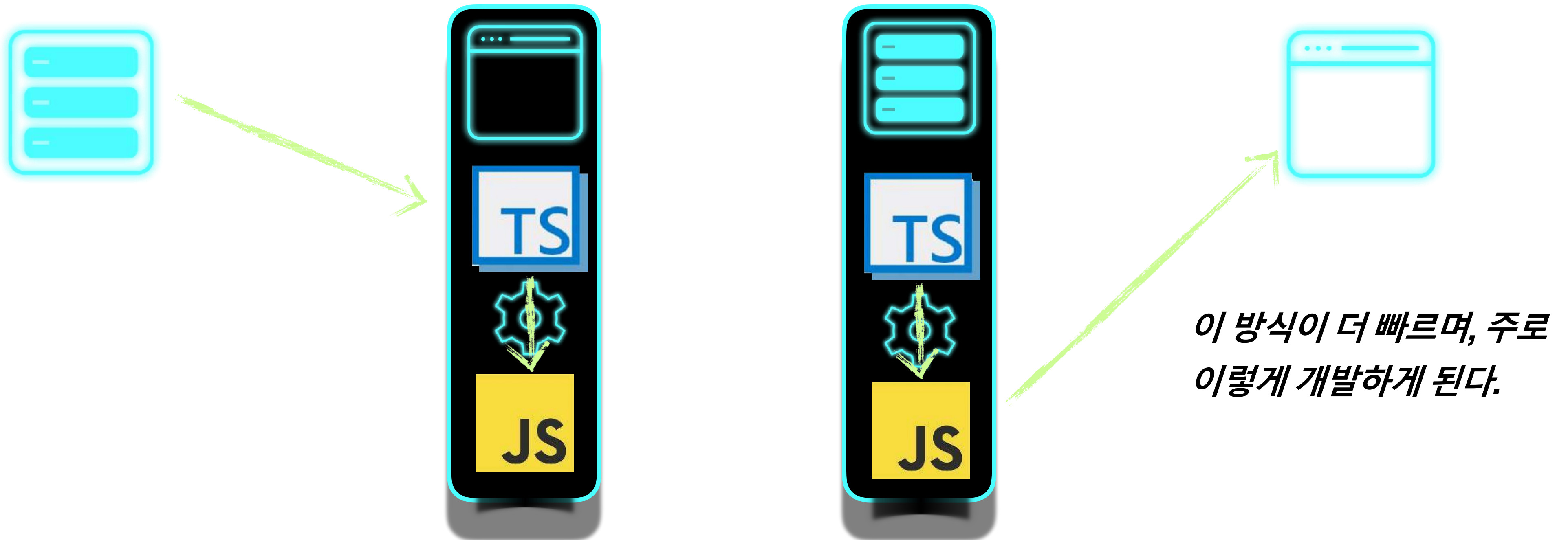
TypeScript : 트랜스파일 위치



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

브라우저에서 자바스크립트로 변환

자바스크립트로 변환 후 브라우저로 로딩



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:nu
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



Atom



Eclipse



Emacs



WebStorm



Vim

TypeScript 구성요소



Strongly
Typed

Classes

Interfaces

Generics

Modules

Type
Definitions

Compiles to
JavaScript

EcmaScript 6
Features

TypeScript : 타입 시스템



- **String**
- **Number**
- **Boolean**
- **Array**
- **Dynamic Typing**
- **Enum**
- **Void**

TypeScript : 클래스



- class 키워드를 이용하여, 클래스 정의

```
class car {  
  numTier: number;  
  carName: string;  
  
  constructor(  
    carName: string, numTier: number){  
    this.carName = carName;  
    this.numTier = numTier;  
  }  
  getNumTier(){  
    return this.numTier;  
  }  
  getCarName(){  
    return this.carName;  
  }  
}
```

```
let myCar = new car("해피카",4);  
console.log(myCar.getCarName()+"의 타이어  
개수는 "+myCar.getNumTier()+"개 입니다.");
```

[결과]
해피카의 타이어 개수는 4개 입니다.

TypeScript : 상속



- extends 키워드를 이용하여 부모 클래스를 상속

```
class HappyCar {
  numTier: number;
  carName: string;
  speed: number;

  constructor(carName: string, numTier: number){
    this.carName = carName;
    this.numTier = numTier;
  }

  setSpeed(speed: number){
    this.speed=speed;
  }

  getSpeed(){
    return this.speed;
  }
}
```

```
class bus extends HappyCar {
  constructor(carName: string, numTier: number) {
    super(carName,numTier);
  }
  setSpeed(speed = 0) {
    super.setSpeed(speed);
  }
}

class truck extends HappyCar {
  constructor(carName: string, numTier: number) {
    super(carName,numTier);
  }
  setSpeed(speed = 0) {
    super.setSpeed(speed);
  }
}
```

TypeScript : 상속



- 인스턴스 생성 후 테스트

```
let myBus = new bus("myBus",6);  
let myTruck: HappyCar = new truck("myTruck",10);  
  
myBus.setSpeed(100);  
myTruck.setSpeed(120);  
console.log("현재 버스속도 : "+myBus.getSpeed());  
console.log("현재 트럭속도 : "+myTruck.getSpeed());
```

[결과]

현재 버스속도 : 100

현재 트럭속도 : 120

TypeScript : 인터페이스



- Interface에 선언된 변수나 메서드에 대한 사용을 강제함

```
interface AddressInterface {  
    addressBookName:string;  
    setBookName(addressBookName: string);  
    getBookName();  
}  
  
class AddressBook implements AddressInterface {  
  
    addressBookName:string;  
    setBookName(addressBookName: string) {  
        this.addressBookName = addressBookName;  
    }  
    getBookName(){  
        return this.addressBookName;  
    }  
    constructor() { }  
}
```

```
let myAddressBook = new AddressBook();  
myAddressBook.setBookName("나의 주소록");  
console.log(myAddressBook.getBookName());
```

[결과]
나의 주소록

TypeScript : module



- ECMAScript 2015에서의 module 개념, export 와 import

Validation.ts

```
export interface StringValidator {  
  isAcceptable(s: string): boolean;  
}
```

ZipCodeValidator.ts

```
import { StringValidator } from "./Validation";  
  
export const numberRegexp = /^[0-9]+$/;  
export class ZipCodeValidator implements StringValidator {  
  isAcceptable(s: string) {  
    return s.length === 5 && numberRegexp.test(s);  
  }  
}
```

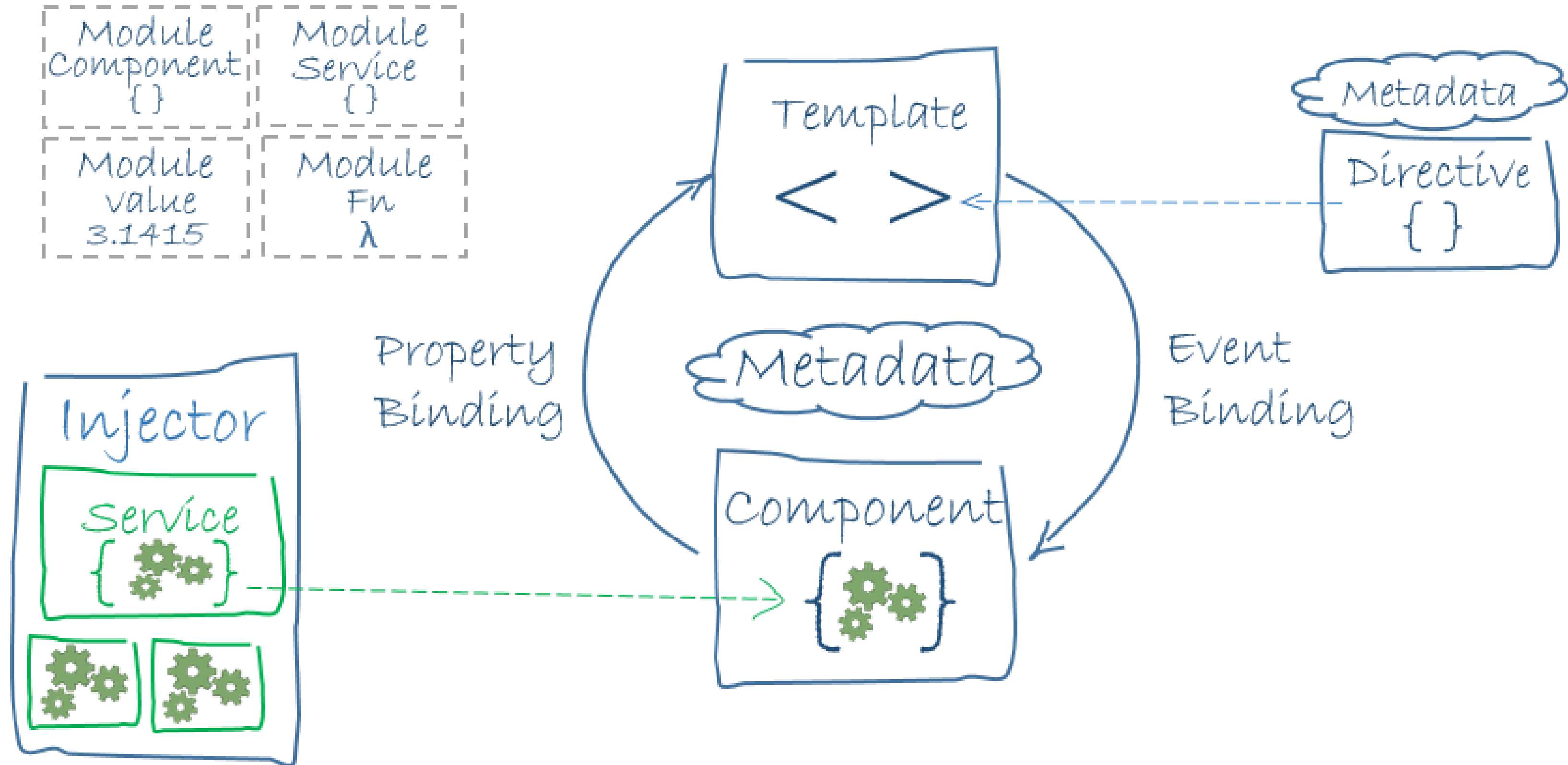
Test.ts

```
import { ZipCodeValidator } from "./ZipCodeValidator";  
let myValidator = new ZipCodeValidator();
```

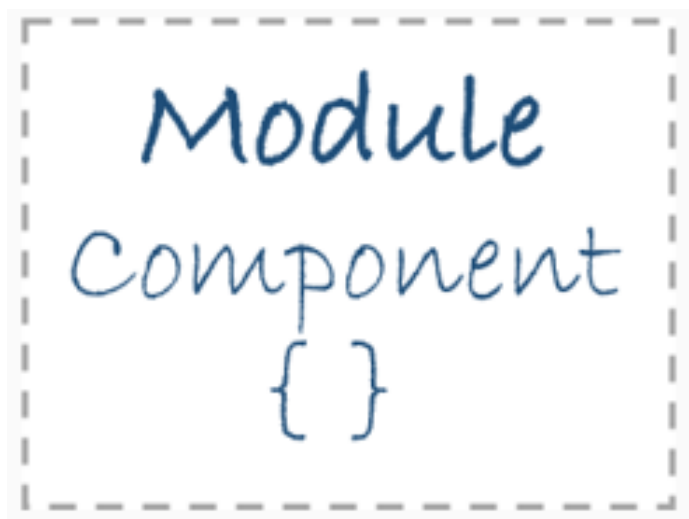

Angular 아키텍처와 주요 컴포넌트

Angular Architecture

- Component를 이용한 Template과 Service 관리



1. Modules



Module
Component
{ }

- 모든 Angular App은 1개의 NgModule(root module) 을 가지고 있다.
- Root module의 이름은 일반적으로 AppModule 이다.
- Module은 @NgModule Decorator를 선언한다.

@NgModule Decorator의 속성들.

declarations	module에 속한 view 클래스들(component, directive, pipe)을 선언한다.
exports	declaration에서 정의한 클래스들이 다른 module에서 사용되는 경우에 export 해야 한다.
imports	필요해서 import 하는 다른 module들을 선언한다.
providers	모든 App에 접근 가능한 Service를 선언한다.
bootstrap	root component 역할을 하는 main component view를 선언한다.

1. Modules

src/app/app.module.ts

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

src/app/main.ts

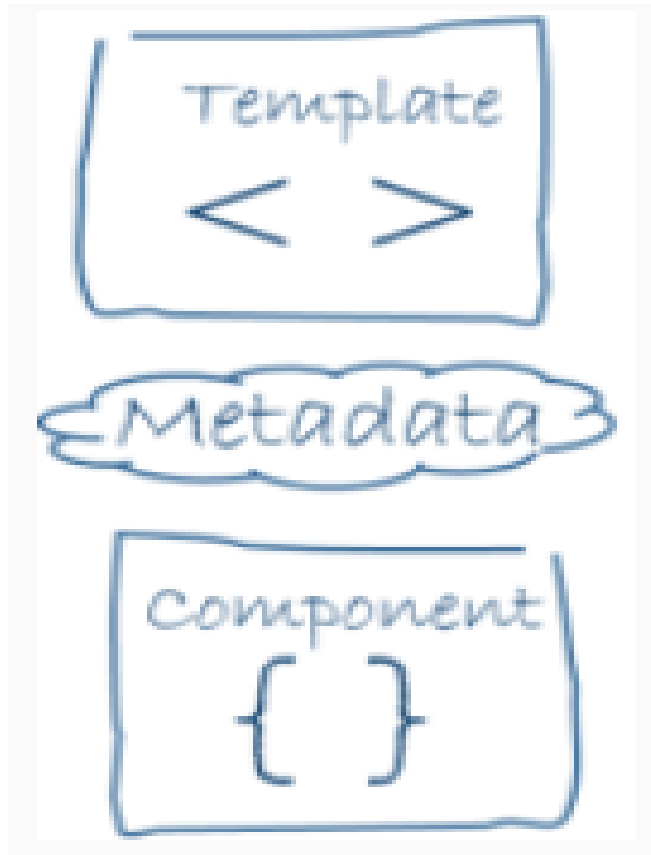
```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

2. Component와 Template



- Component는 View(Template)을 제어하는 역할을 한다.
- View를 support 하기 위해 Component내에 application logic을 정의할 수 있다.
- Component는 `@Component` Decorator를 선언한다.

@Component Decorator의 속성들.

selector	View에서 사용되는 Tag를 설정한다. 이 Tag를 만나면 Angular component가 생성되어진다.
template	Component 내에 View역할을 하는 HTML Code를 포함한다.
templateUrl	View 역할을 하는 HTML Code를 별도의 html 파일로 정의하고, 그 파일명을 지정한다.
styleUrls	Style을 별도의 CSS 파일로 정의하고, 그 파일명을 지정한다.
providers	특정 component에서만 접근 가능한 Service를 선언한다

2. Component와 Template

src/app/hero-list.component.ts

```
import { Component } from '@angular/core';
import { Hero } from './hero';
import { HeroService } from './hero.service';

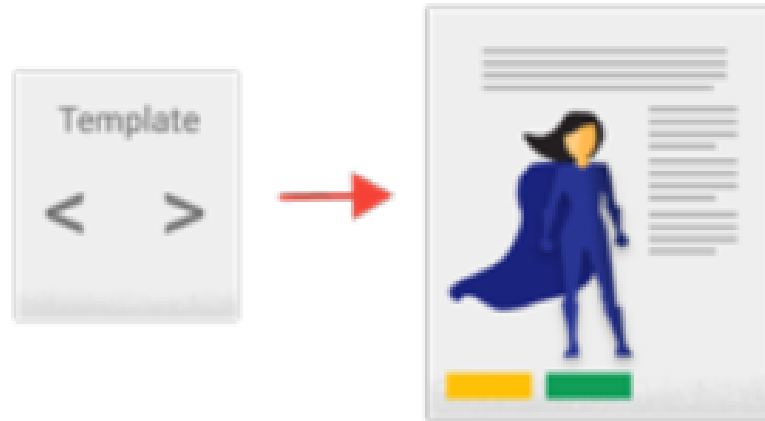
@Component({
  selector: 'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ HeroService ]
})
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private service: HeroService) { }

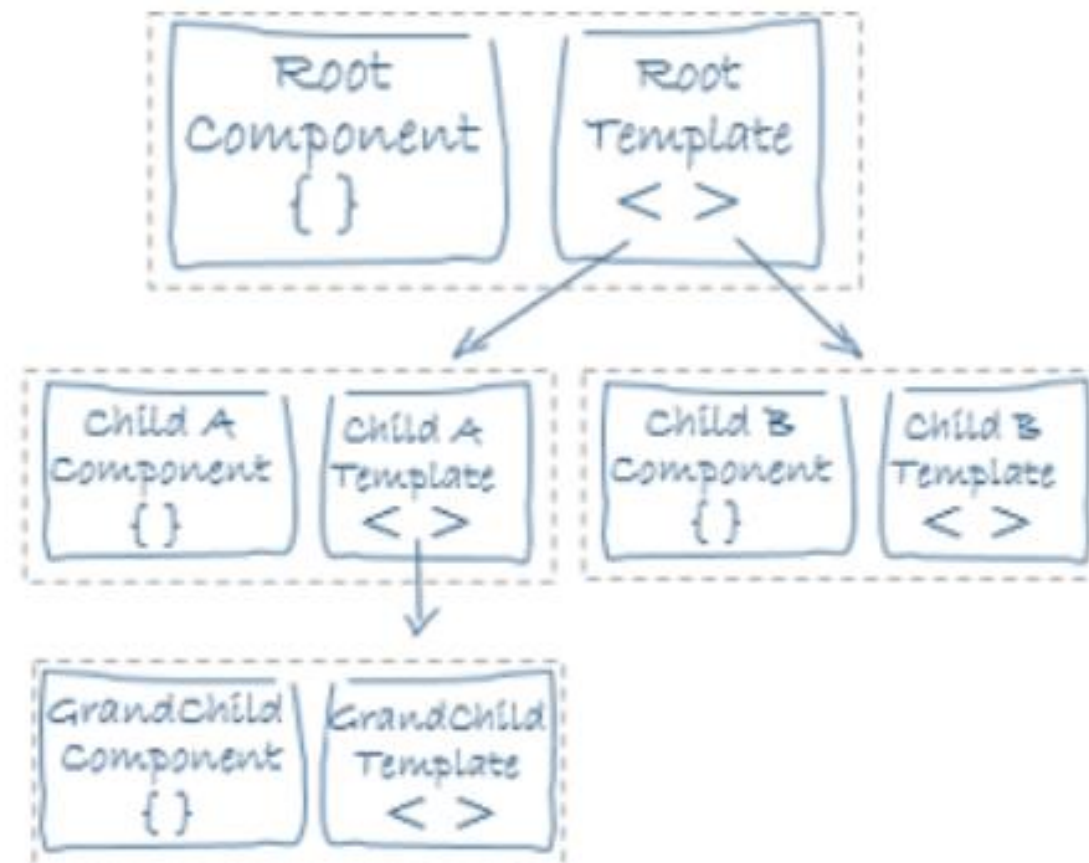
  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }

  selectHero(hero: Hero) { this.selectedHero = hero; }
}
```

2. Component와 Template



- Template으로 Component의 View를 정의합니다.
- Template은 Component를 렌더링 하는 HTML 형식입니다.
- Template은 해당 Component의 Host View를 정의하며, 다른 Component가 호스팅 하는 View를 포함하는 View 계층을 정의 할 수도 있습니다.
- 뷰 계층 구조에는 동일한 NgModule에 있는 구성 요소의 뷰가 포함될 수 있지만 다른 NgModule에 정의 된 구성 요소의 뷰를 포함 할 수 있습니다.



2. Component와 Template

src/app/hero-list.component.html

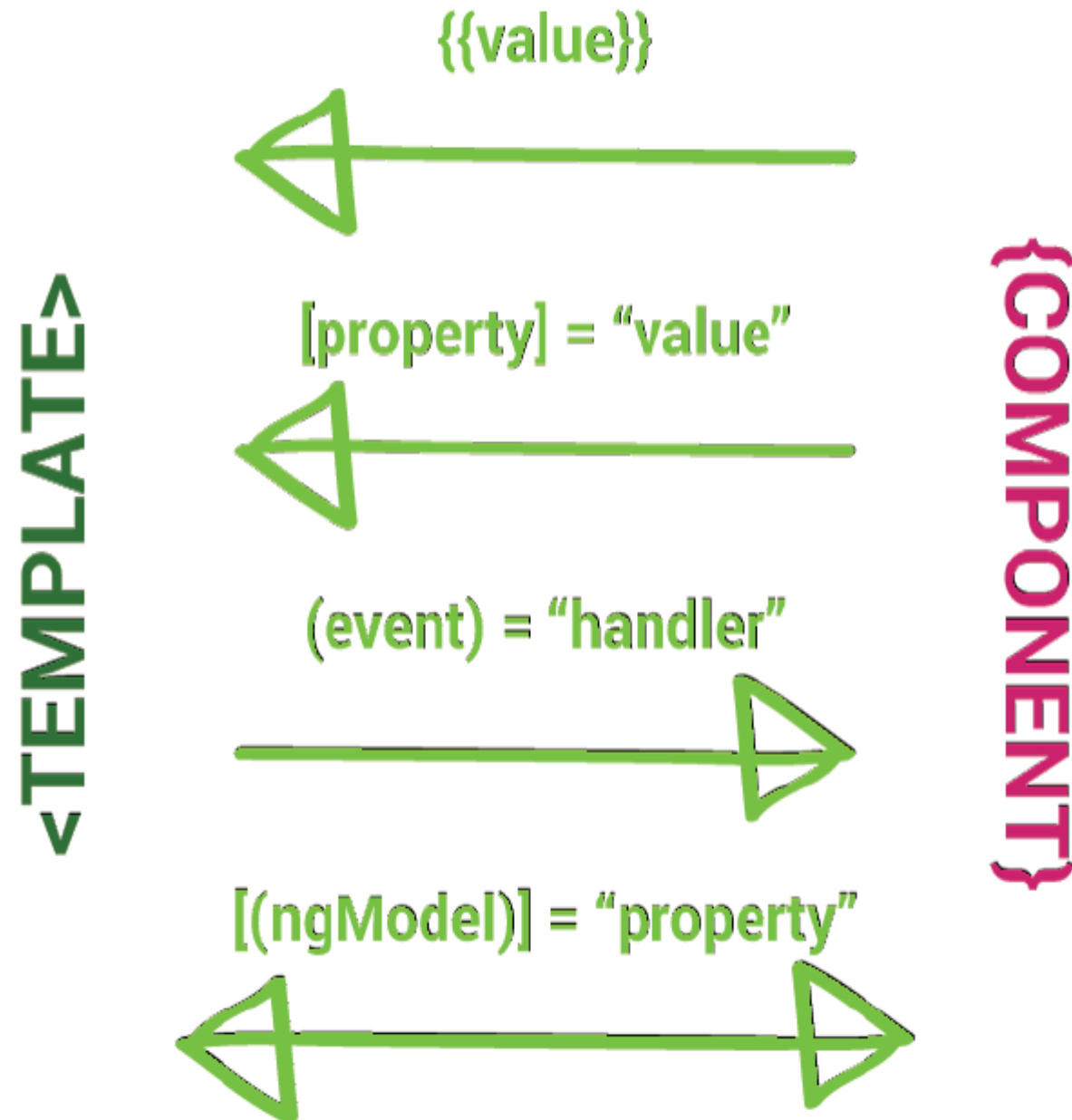
```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

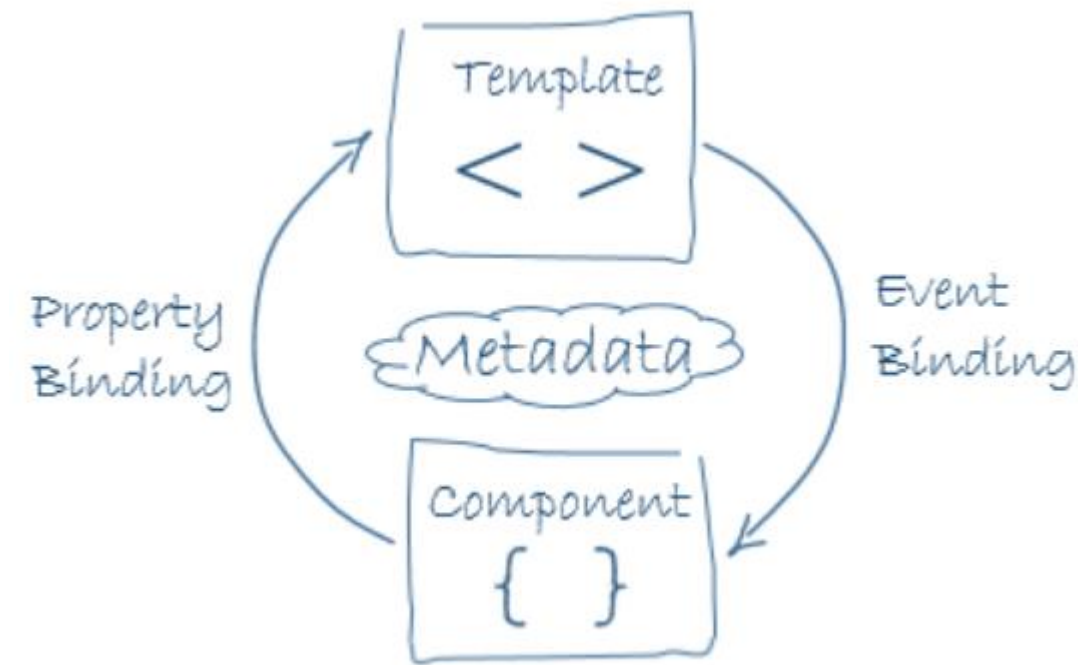
<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

- <app-hero-detail> tag 는 새로운 HeroDetailComponent 를 나타낸다.
- HeroDetailComponent 는 HeroListComponent의 child 이다.
- *ngFor, {{hero.name}}, (click), [hero], <app-hero-detail> 는 Angular의 Template Syntax 이다.

3. Data Binding



Data direction	Syntax	Type
One-way from data source to view target	<code>{{expression}}</code> <code>[target]="expression"</code> <code>bind-target="expression"</code>	Interpolation Property Attribute Class Style
One-way from view target to data source	<code>(target)="statement"</code> <code>on-target="statement"</code>	Event
Two-way	<code>[(target)]="expression"</code> <code>bindon-target="expression"</code>	Two-way



3. Data Binding 문법

Type	Target	Examples
Property	Element property Component property Directive property	<div>src/app/app.component.html</div> <pre> <app-hero-detail [hero]="currentHero"></app-hero-detail> <div [ngClass]="{'special': isSpecial}"></div></pre>
Event	Element event Component event Directive event	<div>src/app/app.component.html</div> <pre><button (click)="onSave()">Save</button> <app-hero-detail (deleteRequest)="deleteHero()"></app-hero-detail> <div (myClick)="clicked=\$event" clickable>Click me</div></pre>
Two-way	Event and property	<div>src/app/app.component.html</div> <pre><input [(ngModel)]="name"></pre>

3. Data Binding

Type	Target	Examples
Attribute	Attribute (the exception)	<div>src/app/app.component.html</div> <div><button [attr.aria-label]="help">help</button></div>
Class	Class Property	<div>src/app/app.component.html</div> <div><div [class.special]="isSpecial">Special</div></div>
Style	Style property	<div>src/app/app.component.html</div> <div><button [style.color]="isSpecial" ? 'red' : 'green'"></div>

3. Data Binding

src/app/hero-list.component.html (binding)

```
<li>{{hero.name}}</li>

<app-hero-detail [hero]="selectedHero"></app-hero-detail>

<li (click)="selectHero(hero)"></li>

<input [(ngModel)]="hero.name">
```

- {{hero.name}} interpolation 은 component의 hero.name property 값을 출력한다.
- [hero] property binding 은 parent HeroListComponent 가 child HeroDetailComponent 에게 selectedHero 의 값을 전달한다.
- (click) event binding 은 hero's name을 클릭했을 때 component의 selectHero method 을 호출한다.
- ngModel Directive를 사용해서 property binding과 event binding을 한번에 동시에 처리한다.

4. Directive



- Directive는 @Directive Decorator를 선언한다.
- Structural Directive와 Attribute Directive 두 가지 종류가 있다.
- Structural Directive는 DOM 엘리먼트를 추가,삭제, 갱신 할 수 있다.

1) Structural Directive : ngFor , ngIf

src/app/hero-list.component.html (structural)

```
<li *ngFor="let hero of heroes"></li>  
<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

2) Attribute Directive : ngModel

src/app/hero-detail.component.html (attribute)

```
<input [(ngModel)]="hero.name">
```

5. Pipe

- Pipe를 사용하면 템플릿 HTML에서 표시 값 변환을 선언 할 수 있습니다.
- Angular는 날짜 파이프 및 통화 파이프와 같은 다양한 파이프를 정의합니다.
- HTML 템플릿에서 값 변환을 지정하려면 파이프 연산자 (|)를 사용하십시오.
- `{{interpolated_value | pipe_name}}`
- `@Pipe` 데코레이터가 있는 클래스는 입력 값을 출력 값으로 변환하여 뷰에 표시하는 함수를 정의합니다.
- 사용자 정의 데코레이터 클래스를 작성할 수도 있습니다.

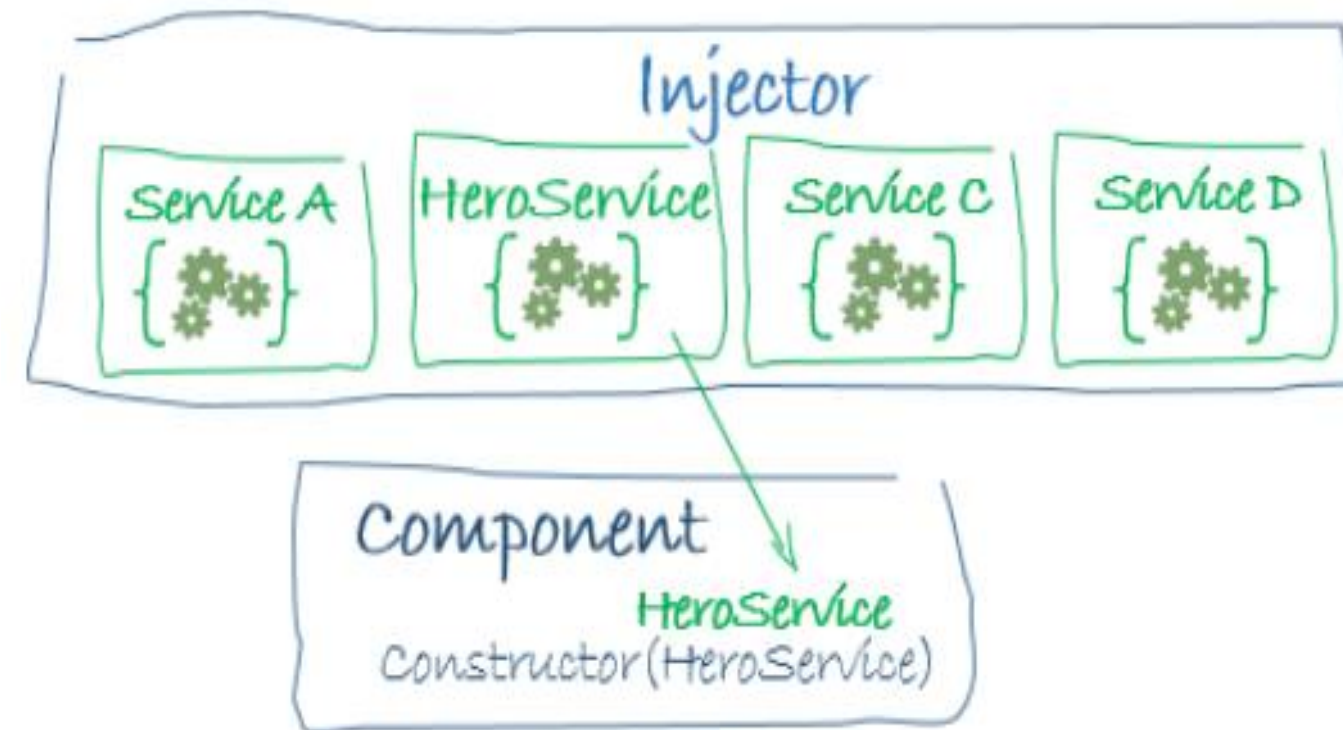
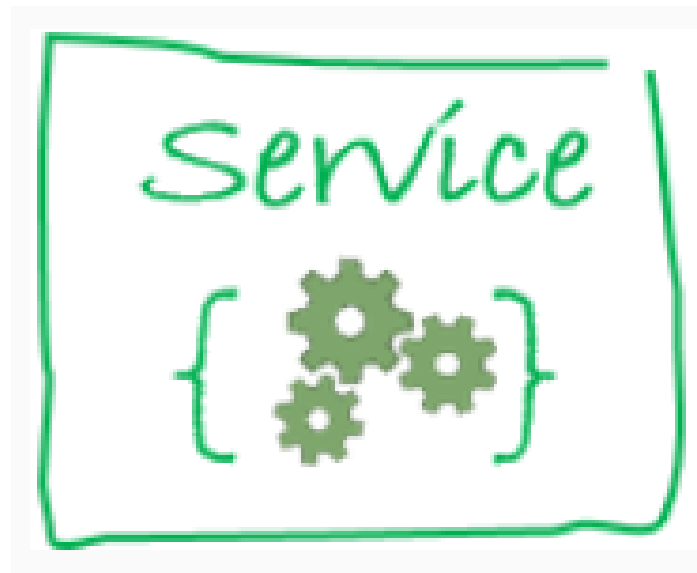
src/app/hero-detail.component.html (attribute)

```
<!-- Default format: output 'Jun 15, 2019'-->
<p>Today is {{today | date}}</p>

<!-- fullDate format: output 'Monday, June 15, 2019'-->
<p>The date is {{today | date:'fullDate'}}</p>

<!-- shortTime format: output '9:43 AM'-->
<p>The time is {{today | date:'shortTime'}}</p>
```

6. Service



- Service는 @Injectable Decorator를 선언한다.
- Service에는 다양한 기능들을 정의할 수 있다.
- 예) logging service, data service, message bus, tax calculator
- Component는 Service의 big consumer 이다.
- Angular는 Service를 Component에게 제공하기 위해 DI(dependency Injection) 사용한다.
- Dependency Injection은 의존관계에 있는 다른 객체에게 새로운 객체를 제공하기 위한 방법이다.

6. Service

src/app/hero.service.ts

```
@Injectable()
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService, private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

src/app/hero-list.component.ts

```
private heros: Hero[];

constructor(private service: HeroService) { }

ngOnInit() {
  this.heros = service.getHeroes();
}
```


6. Service

- Service 객체를 Root module의 providers로 등록하면 , 모든 Component 들이 접근할 수 있다.

src/app/app.module.ts

```
providers: [  
  BackendService,  
  Logger  
],
```

- Service 객체를 Component의 providers로 등록하면 , 특정 Component 에서만 접근할 수 있다.

src/app/hero-list.component.ts

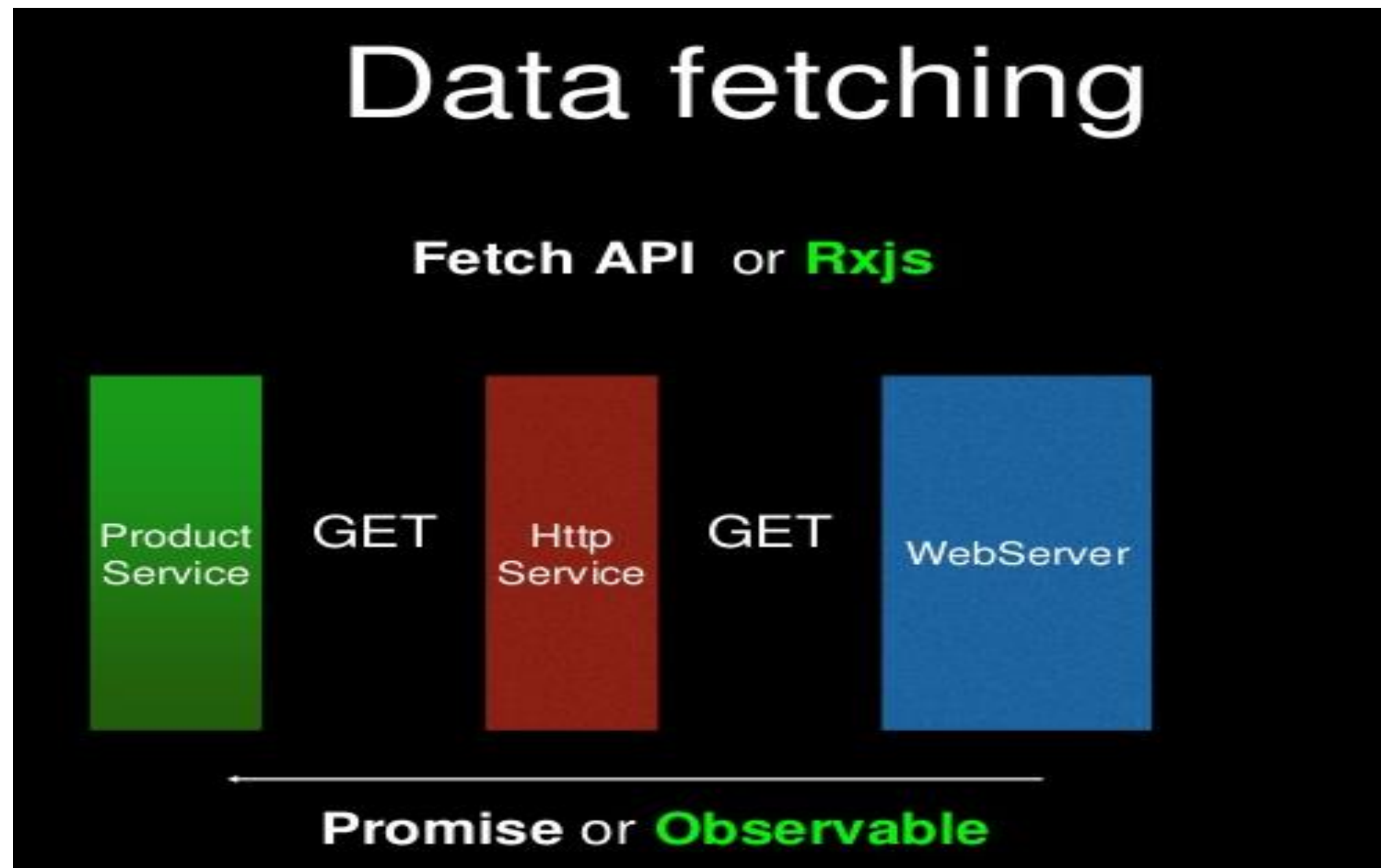
```
constructor(private service: HeroService) { }  
  
@Component({  
  selector: 'app-hero-list',  
  templateUrl: './hero-list.component.html',  
  providers: [ HeroService ]  
})
```

7. HTTP 와 RxJS 라이브러리 사용하기

HTTP 라이브러리는 원격서버에 접속하여 데이터를 주고 받기위한 라이브러리이다.

RxJS 라이브러리는 **Reactive Extensions** 의 약자이고, **http** 요청에 대한 여러가지 기능을 제공한다.

Angular http는 **RxJS**의 **Observable**과 **ES6**의 **Promise** 객체를 사용한다.

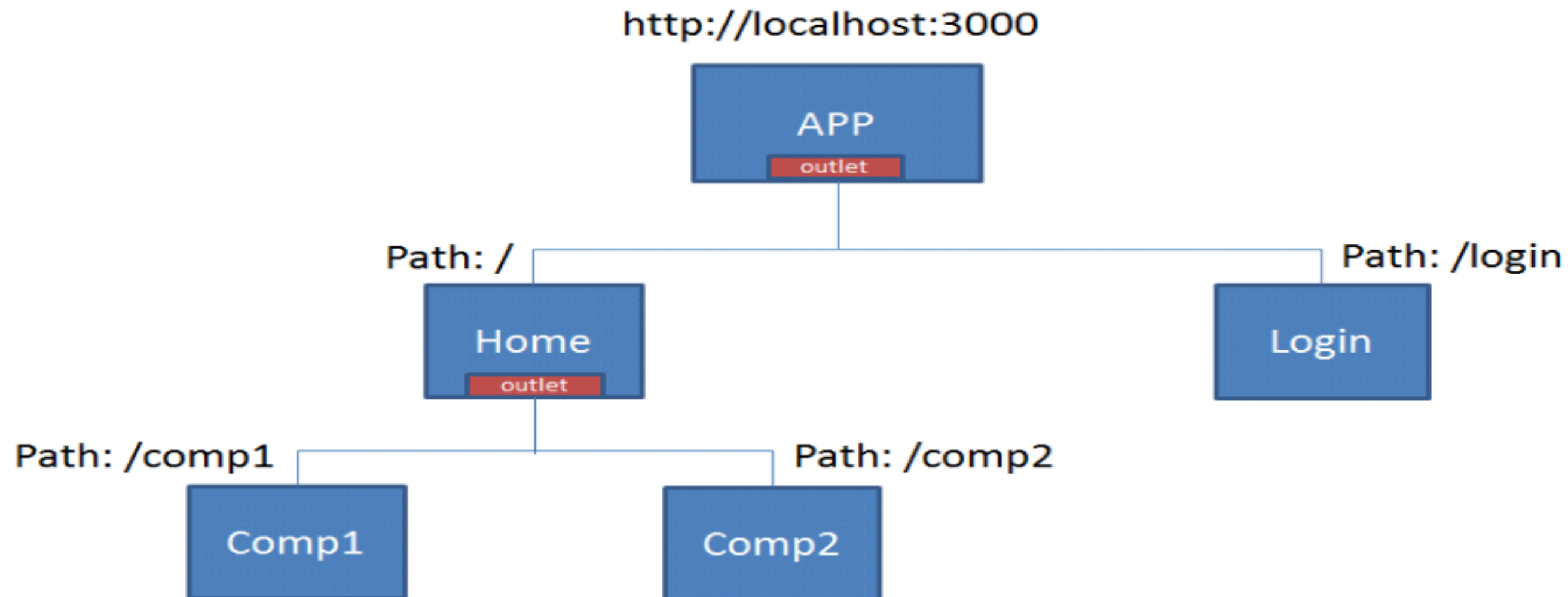


8. Router 개요

- Router 란?

: URI Hash 값을 이용하여 화면의 Navigation을 지원하는 기능이다.

: Router를 사용할 경우 구분된 여러 개의 Component를 교체하면서 화면에 보여줄 수 있다.



9. Angular Form : 2가지 Form-building 방식

- Angular의 두가지 Form-building 방식

- 1) Template-driven forms 와 Reactive forms가 있다.

- 2) 두 방식 모두 @angular/forms 라이브러리에 속한다.

- 3) Template-driven forms : FormsModule 사용

- Reactive forms : ReactiveFormsModule 사용

- 4) Template-driven forms의 특징

- Template-driven forms 방식은 form control object를 사용자가 만들지 않고, Angular가 form control object를 만들어 준다. 즉, ngModel를 제공 하여 Angular가 핸들링 한다.

- 이벤트가 발생하면 Angular는 mutable 데이터 모델을 업데이트 합니다.

9. Angular Form : 2가지 Form-building 방식

- Angular의 두가지 Form-building 방식

- 5) Reactive forms의 특징

- Reactive forms는 컴포넌트 클래스에 form control object를 만들고, 이를 컴포넌트 템플릿의 form control 엘리먼트에 바인딩 할 수 있다.
 - 컴포넌트 클래스는 데이터 모델과 form control 구조에 즉시 액세스 할 수 있으므로 데이터 모델 값을 form control로 보내고, 사용자가 변경 한 값을 다시 가져올 수 있다.
 - Reactive forms의 장점은 값 및 유효성 업데이트가 항상 동기적 이고 사용자가 제어 할 수 있다는 점입니다.

Spring Boot REST Service

Spring Boot REST Service : Spring Boot Project 생성

- Spring Boot Application 작성

1) Spring_INITIALIZER (<https://start.spring.io/>)

2) 생성된 프로젝트 zip 파일을 저장하고, 압축을 푼다.

3) STS에서 File -> Open Project From File System으로 프로젝트 폴더를 연다.

SPRING INITIALIZR bootstrap your application now

Generate a

Maven Project ▾

 with

Java ▾

 and Spring Boot

2.0.2 ▾

Project Metadata

Artifact coordinates

Group

myspringboot.user

Artifact

springboot-anular4

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web ✕

Generate Project alt + ⌘

Spring Boot REST Service : Model 클래스 작성

- 1-1) Hero Model 클래스 작성 #1

src/main/java/myspringboot/hero/Hero.java

```
package myspringboot.hero;

public class Hero {
    private Long id;
    private String name;

    public Hero() {

    }

    public Hero(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    getter()
    setter()
    toString()
}
```


Spring Boot REST Service : Controller 클래스 작성

- 2-1) HeroController 클래스 작성 #1

src/main/java/myspringboot/hero/HeroController.java

```
package myspringboot.hero;

@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping(value = "/heroes")
public class HeroController {
    private List<Hero> heroList = new ArrayList<>();
    HeroController() {
        buildHeroes();
    }
    void buildHeroes() {
        Hero hero1 = new Hero(11L, "Miss. Nice");
        Hero hero2 = new Hero(12L, "Narco");
        Hero hero3 = new Hero(13L, "Bombasto");
        Hero hero4 = new Hero(14L, "Celeritas");
        Hero hero5 = new Hero(15L, "Magneta");
        Hero hero6 = new Hero(16L, "RubberMan");
        Hero hero7 = new Hero(16L, "Dynamia");
        Hero hero8 = new Hero(18L, "Dr IQ");
        Hero hero9 = new Hero(19L, "Magma");
        Hero hero10 = new Hero(20L, "Tornado");
        heroList.add(hero1);
        ...
    }
}
```

Spring Boot REST Service : Controller 클래스 작성

- 2-2) HeroController 클래스 작성 #2

src/main/java/myspringboot/hero/HeroController.java

```
@RequestMapping(method = RequestMethod.GET)
public List<Hero> getHeros() {
    return this.heroList;
}
@RequestMapping(value = "/{id}", method = RequestMethod.GET)
public Hero getHero(@PathVariable("id") Long id) {
    return this.heroList.stream()
        .filter(hero -> hero.getId() == id)
        .findFirst()
        .orElse(null);
}
@RequestMapping(method = RequestMethod.POST)
public Hero saveHero(@RequestBody Hero hero) {
    Long nextId = 0L;
    if (this.heroList.size() != 0) {
        Hero lastHero = this.heroList.stream().skip(this.heroList.size() - 1)
            .findFirst().orElse(null);
        nextId = lastHero.getId() + 1;
    }
    hero.setId(nextId);
    this.heroList.add(hero);
    return hero;
}
```

Spring Boot REST Service : Controller 클래스 작성

- 2-3) HeroController 클래스 작성 #3

src/main/java/myspringboot/hero/HeroController.java

```
@RequestMapping(method = RequestMethod.PUT)
public Hero updateHero(@RequestBody Hero hero) {
    Hero modifiedHero = this.heroList.stream()
        .filter(u -> u.getId() == hero.getId()).findFirst().orElse(null);
    modifiedHero.setName(hero.getName());
    return modifiedHero;
}

@RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
public boolean deleteHero(@PathVariable Long id) {
    Hero deleteHero = this.heroList.stream()
        .filter(h -> h.getId() == id).findFirst().orElse(null);
    if (deleteHero != null) {
        this.heroList.remove(deleteHero);
        return true;
    } else {
        return false;
    }
}

@RequestMapping(value = "/name/{name}", method = RequestMethod.GET)
public List<Hero> searchHeroes(@PathVariable String name) {
    return this.heroList.stream()
        .filter(hero -> hero.getName().contains(name))
        .collect(Collectors.toList());
}
```

Spring Boot REST Service : Tomcat에 배포하기 위한 설정

- 3-1) application.properties 수정하기

Default port 번호는 8080 이다. Port 변경이 필요한 경우에 server.port 속성 추가

```
src/main/resources/application.properties
```

```
server.port=8087
```

- 3-2) Spring Boot App Start

<http://localhost:8087/heroes> 로 Json 데이터를 확인한다.

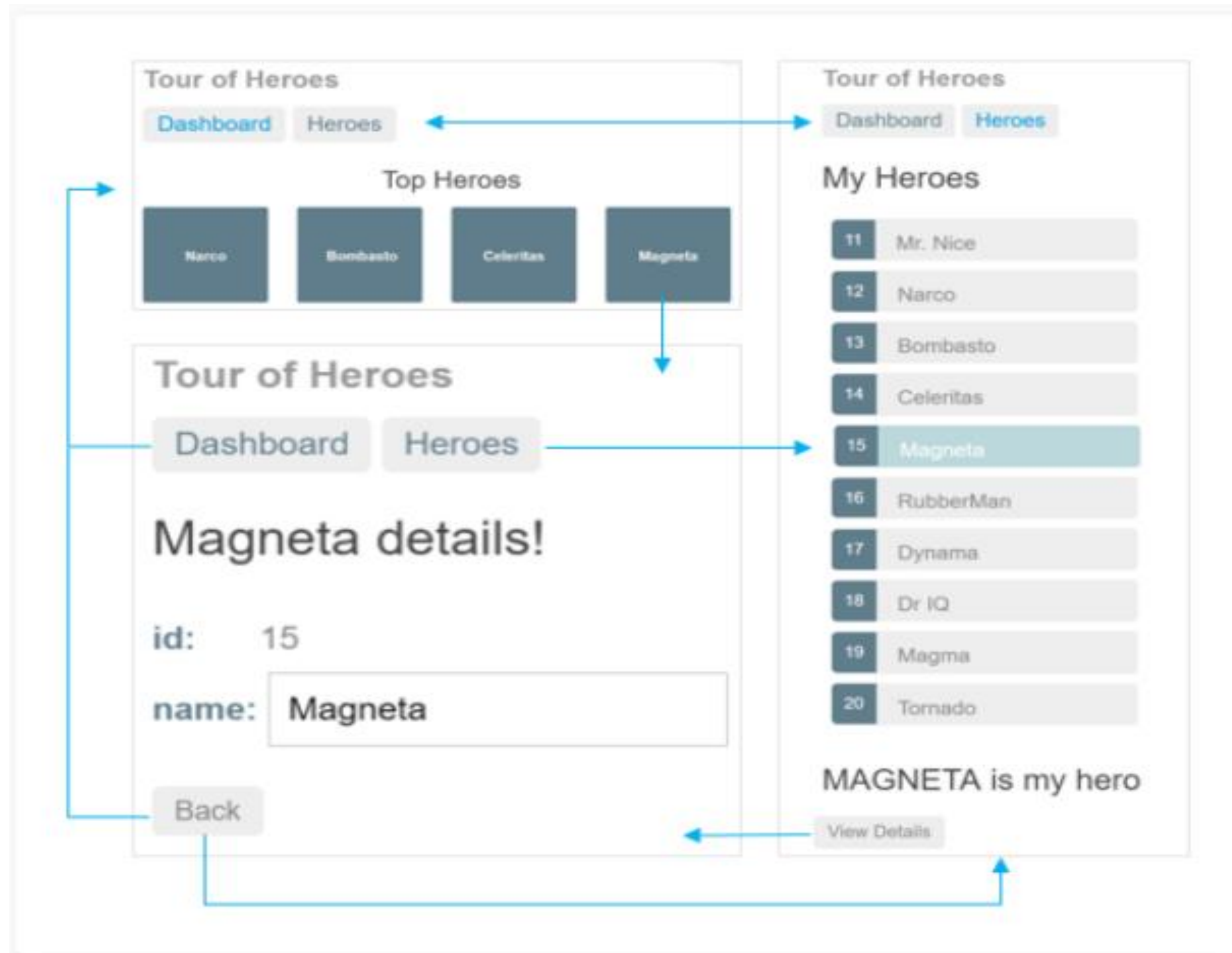
Angular App 작성

: Tour of Heroes App

Tour of Heroes App 소개

- App을 통해 구현되는 내용들
 - 1) Built-in Angular Directive를 사용하여 Hero data의 리스트를 보여준다.
 - 2) Angular 컴포넌트를 작성하여 Hero의 상세정보를 보여준다.
 - 3) Read-Only data를 위한 One-Way Data Binding을 사용한다.
 - 4) Model을 업데이트 하기 위해 Editable 필드를 추가하여 Two-Way Binding을 사용한다.
 - 5) Event를 처리할 때 Component의 메서드와 연결한다.
 - 6) Master List의 Element를 선택하면 Element의 detail 한 정보를 볼 수 있다.
 - 7) Pipe를 사용하여 데이터를 포매팅 한다.
 - 8) Service를 생성한다.
 - 9) 화면을 전환하기 위해서 Routing을 사용한다.

Tour of Heroes App 소개



Tour of Heroes App 작성

- 새로운 Angular Application Project 생성

```
ng new angular-tour-of-heroes
```

- Application 실행

```
cd angular-tour-of-heroes  
ng serve --open
```

- Application 구성요소

- 1) app.component.ts – TypeScript로 작성된 component class
- 2) app.component.html – HTML로 작성된 component template
- 3) app.component.css – 현재의 Component에만 적용되는 CSS style

Tour of Heroes App 작성 : (1) Application Shell

(1)-1. Root Component의 title 변경

src/app/app.component.ts

```
title = "Tour of Heroes"
```

src/app/app.component.html

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

src/styles.css

```
h1 { color: #369; font-family: Arial, Helvetica, sans-serif; font-size: 250%; }  
h2, h3 { color: #444; font-family: Arial, Helvetica, sans-serif; font-weight: lighter; }  
body { margin: 2em; }  
body, input[type="text"], button { color: #333; font-family: Cambria, Georgia, Serif; }  
* { font-family: Arial, Helvetica, sans-serif; }
```

Tour of Heroes App 작성 : (2) Hero Editor

(2)-1. Heroes Component 작성 (lifecycle hooks : <https://angular.io/guide/lifecycle-hooks>)

```
ng generate component heroes
```

src/app/heroes/heroes.component.ts

```
@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css'] })
export class HeroesComponent implements OnInit {
  hero = 'widnstrom';
  constructor() { }
  ngOnInit() { }
}
```

src/app/heroes/heroes.component.html

```
<h1> {{hero}} </h1>
```

src/app/app.component.html

```
<h1> {{title}} </h1>
<app-heroes></app-heroes>
```

Tour of Heroes App 작성 : (2) Hero Editor

(2)-2. Hero Model 클래스 작성 : Hero 클래스를 hero 변수의 타입으로 사용함

```
ng generate interface hero
```

src/app/hero.ts

```
export interface Hero {  
  id: number;  
  name: string;  
}
```

src/app/heroes/heroes.component.ts

```
@Component({  
  selector: 'app-heroes',  
  templateUrl: './heroes.component.html',  
  styleUrls: ['./heroes.component.css'] })  
export class HeroesComponent implements OnInit {  
  hero: Hero = { id: 1, name: 'Windstorm' };  
  constructor() { }  
  ngOnInit() { }  
}
```

Tour of Heroes App 작성 : (2) Hero Editor

(2)-3. Hero Model 클래스 작성 : Hero 클래스의 property 사용, UppercasePipe 사용

src/app/heroes/heroes.component.html

```
<h2>{{ hero.name | uppercase }} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div><span>name: </span>{{hero.name}}</div>
```

(2)-4. Heroes의 property를 Edit 할 수 있는 기능 추가 :

Two-way Binding을 지원하는 ngModel Directive의 사용

src/app/heroes/heroes.component.html

```
<h2>{{ hero.name | uppercase }} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div>
  <label for="hero-name">Hero name: </label>
  <input id="hero-name" [(ngModel)]="hero.name" placeholder="name"> </label>
</div>
```

Template parse errors:

Can't bind to 'ngModel' since it isn't a known property of 'input'.

Tour of Heroes App 작성 : (2) Hero Editor

(2)-5. Heroes의 property를 Edit 할 수 있는 기능 추가 :

ngModel Directive가 포함된 FormsModule import 해야 한다.

새로 만든 HeroesComponent도 선언(declarations) 해야 한다.

src/app/app.module.ts

```
import { FormsModule } from '@angular/forms';
import { HeroesComponent } from './heroes/heroes.component';

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

Tour of Heroes App 작성 : (3) Displaying a List

(3)-1. Heroes List를 출력하는 기능 추가 : Mock heroes 클래스 작성, HEROES 상수 정의

src/app/mock-heroes.ts

```
import { Hero } from './hero';

export const HEROES: Hero[] = [
  { id: 11, name: 'Mr. Nice' }, { id: 12, name: 'Narco' }, { id: 13, name: 'Bombasto' },
  { id: 14, name: 'Celeritas' }, { id: 15, name: 'Magneta' }, { id: 16, name: 'RubberMan' },
  { id: 17, name: 'Dynamia' }, { id: 18, name: 'Dr IQ' }, { id: 19, name: 'Magma' }, { id: 20, name: 'Tornado' }
];
```

(3)-2. Heroes List를 출력하는 기능 추가 : HEROES 상수를 component에서 사용

src/app/heroes/heroes.component.ts

```
import { HEROES } from '../mock-heroes';
@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css'] })
export class HeroesComponent implements OnInit {
  heroes = HEROES;
  constructor() { }
  ngOnInit() { }
}
```

Tour of Heroes App 작성 : (3) Displaying a List

(3)-3. Heroes List를 출력하는 기능 추가 : ngFor repeater Directive 사용

src/app/heroes/heroes.component.html

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes"> <span class="badge">{{hero.id}}</span> {{hero.name}} </li>
</ul>
```

- 는 host element
- heroes는 HeroesComponent클래스에 정의된 heroes list에 해당 되는 변수이다.
- hero는 list를 for each iteration을 수행 하면서 현재의 hero object를 저장하는 임시변수이다.

Tour of Heroes App 작성 : (3) Displaying a List

(3)-4. Heroes List를 출력하는 기능 추가 : heroes.component의 css style

src/app/heroes/heroes.component.css

```
.heroes { margin: 0 0 2em 0; list-style-type: none; padding: 0; width: 15em; }

.heroes li { cursor: pointer; position: relative; left: 0;
  background-color: #EEE; margin: .5em; padding: .3em 0; height: 1.6em; border-radius: 4px; }

.heroes li:hover { color: #2c3a41; background-color: #e6e6e6; left: .1em; }

.heroes li.selected { background-color: black; color: white; }

.heroes li.selected:hover { background-color: #505050; color: white; }

.heroes li.selected:active { background-color: black; color: white; }

.heroes .badge { display: inline-block; font-size: small; color: white; padding: 0.8em 0.7em 0 0.7em;
  background-color: #405061; line-height: 1em; position: relative;
  left: -1px; top: -4px; height: 1.8em; margin-right: .8em; border-radius: 4px 0 0 4px;
}

input { padding: .5rem; }
```


Tour of Heroes App 작성 : (3) Displaying a List

(3)-5. Heroes Master/Detail 기능 추가 : click event binding 처리, onSelect() 메서드 추가

src/app/heroes/heroes.component.html

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)" >
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
```

src/app/heroes/heroes.component.ts

```
export class HeroesComponent implements OnInit {
  heroes = HEROES;
  selectedHero?: Hero;
  constructor() { }
  ngOnInit() { }

  onSelect(hero: Hero): void {
    this.selectedHero = hero;
  }
}
```

Tour of Heroes App 작성 : (3) Displaying a List

(3)-6. Heroes Master/Detail 기능 추가 : detail 정보를 출력하는 부분을 수정한다.

src/app/heroes/heroes.component.html

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)" >
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>

<h2>{{ selectedHero.name | uppercase }} Details</h2>
<div><span>id: </span>{{selectedHero.id}}</div>
<div>
  <label for="hero-name">Hero name: </label>
  <input id="hero-name" [(ngModel)]="selectedHero.name" placeholder="name">
</div>
```

```
HeroesComponent.html:3 ERROR TypeError: Cannot read property 'name' of undefined
```

Tour of Heroes App 작성 : (3) Displaying a List

(3)-7. Heroes Master/Detail 기능 추가 :

selectedHero가 존재할 때 만 선택된 hero의 상세정보를 출력하기 위해 **ngIf directive**를 사용함

src/app/heroes/heroes.component.html

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)" >
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>

<div *ngIf="selectedHero">
  <h2>{{ selectedHero.name | uppercase }} Details</h2>
  <div><span>id: </span>{{selectedHero.id}}</div>
  <div>
    <label for="hero-name">Hero name: </label>
    <input id="hero-name" [(ngModel)]="selectedHero.name" placeholder="name">
  </div>
</div>
```

Tour of Heroes App 작성 : (3) Displaying a List

(3)-8. Heroes Master/Detail 기능 추가 : selected Hero의 style 변경(background color 변경)

14 Celeritas

15 Magneta

16 RubberMan

class binding은 조건에 따라서 CSS class를 쉽게 추가하거나 제거할 수 있다.

class binding의 형식은 [class.some-css-class]="some-condition" 이다.

src/app/heroes/heroes.component.html

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes"
      [class.selected]="hero === selectedHero"
      (click)="onSelect(hero)" >
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
```

Tour of Heroes App 작성 : (4) Master/Detail 컴포넌트

(4)-1. HeroDetail Component 새로 작성하여 Master/Detail 기능 추가 :

```
ng generate component hero-detail
```

heroes.component.html에 있던 hero의 상세정보를 hero-detail.component.html로 옮긴다.

src/app/hero-detail/hero-detail.component.html

```
<div *ngIf="hero">
  <h2>{{ hero.name | uppercase }} Details</h2>
  <div><span>id: </span>{{hero.id}}</div>
  <div>
    <label for="hero-name">Hero name: </label>
    <input id="hero-name" [(ngModel)]="hero.name" placeholder="name">
  </div>
</div>
```

src/app/heroes/heroes.component.html

```
<app-hero-detail [hero]="selectedHero"> </app-hero-detail>
```

Tour of Heroes App 작성 : (4) Master/Detail 컴포넌트

(4)-2. HeroDetail Component 새로 작성하여 Master/Detail 기능 추가

src/app/hero-detail/hero-detail.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { Hero } from '../hero';

export class HeroDetailComponent implements OnInit {
  @Input() hero: Hero;
```

src/app/heroes/heroes.component.html

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes"
      [class.selected]="hero === selectedHero"
      (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>

<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

Tour of Heroes App 작성 : (5) Services

(5)-1. HeroService Component 작성하기

```
ng generate service hero
```

HeroService는 DI(Dependency Injection)를 사용하여 HeroesComponent에 주입(inject) 되어진다.

src/app/hero.service.ts

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

@Injectable( { providedIn: 'root' } )
export class HeroService {
  constructor() { }
}
```

Tour of Heroes App 작성 : (5) Services

(5)-2. HeroService Component 작성하기 : HeroService에 getHeroes() 메서드 추가

src/app/hero.service.ts

```
@Injectable()
export class HeroService {
  constructor() {}
  getHeroes(): Hero[] {
    return HEROES;
  }
}
```

src/app/heroes/heroes.component.ts

```
import { HeroService } from '../hero.service';
export class HeroesComponent implements OnInit {
  heroes: Hero[] = [];
  constructor(private heroService: HeroService) {}
  ngOnInit() {
    this.getHeroes();
  }
  getHeroes(): void {
    this.heroes = this.heroService.getHeroes();
  }
}
```


Tour of Heroes App 작성 : (5) Services

(5)-3. Observable HeroService Component 작성하기 : getHeroes() 메서드 수정하기

Observable은 RxJS Library(<http://reactivex.io/rxjs/>)의 중요한 클래스이다.

Angular의 HttpClient의 메서드들은 RxJS Observable 객체를 리턴 한다.

RxJS의 of() function은 Server로 부터 데이터를 가져오는 것 처럼 simulate 하는 함수이다.

(of() 대신 HttpClient의 get() 메서드 사용하기) 

src/app/hero.service.ts

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';

@Injectable()
export class HeroService {
  constructor() {}
  getHeroes(): Observable<Hero[]> {
    const heroes = of(HEROES);
    return heroes;
  }
}
```

Tour of Heroes App 작성 : (5) Services

(5)-4. Observable HeroService Component 작성 : heroes.component의 getHeroes() 메서드 수정
RxJS의 subscribe() 함수는 of() 함수에서 emit한 데이터를 subscribe(구독)하는 함수이다.
subscribe()는 콜백 함수의 인자로 받은 데이터를 this.heroes 변수에 저장해 준다.

src/app/heroes/heroes.component.ts

```
import { HeroService } from '../hero.service';
export class HeroesComponent implements OnInit {
  heroes: Hero[] = [];
  constructor(private heroService: HeroService) {
  }
  ngOnInit() {
    this.getHeroes();
  }
  getHeroes(): void {
    this.heroService.getHeroes().subscribe(heroes => this.heroes = heroes);
  }
}
```

Tour of Heroes App 작성 : (5) Services

(5)-5. Message Component 작성

```
ng generate component messages
```

src/app/messages/messages.component.ts

```
@Component({
  selector: 'app-messages',
  templateUrl: './messages.component.html',
  styleUrls: ['./messages.component.css'] })
export class MessagesComponent implements OnInit {
  constructor() { }
  ngOnInit() { }
}
```

src/app/app.component.html

```
<h1> {{title}} </h1>
<app-heroes></app-heroes>
<app-messages></app-messages>
```

Tour of Heroes App 작성 : (5) Services

(5)-6. MessageService Component 작성

ng generate service message

src/app/message-service.ts

```
@Injectable( { providedIn: 'root',} )

export class MessageService {
  messages:string[] = [];

  add(message:string) {
    this.messages.push(message);
  }
  clear() {
    this.messages = [];
  }
}
```

Tour of Heroes App 작성 : (5) Services

(5)-7. HeroService에서 MessageService 호출하기

"*service-in-service*" scenario

src/app/hero.service.ts

```
import { MessageService } from './message.service';

@Injectable({ providedIn: 'root',})
export class HeroService {
  constructor(private messageService: MessageService) { }

  getHeroes(): Observable<Hero[]> {
    //send the message _after_ fetching the heroes
    const heroes = of(HEROES);
    this.messageService.add('HeroService: fetched heroes');
    return heroes;
  }
}
```

Tour of Heroes App 작성 : (5) Services

(5)-8. MessageComponent에서 MessageService 호출하기

src/app/messages/messages.component.ts

```
import { MessageService } from './message.service'
@Component({
  selector: 'app-messages',
  templateUrl: './messages.component.html',
  styleUrls: ['./messages.component.css']
})
export class MessagesComponent implements OnInit {
  constructor(public messageService: MessageService) {}

  ngOnInit() {}
}
```

src/app/messages/messages.component.html

```
<div *ngIf="messageService.messages.length">
  <h2>Messages</h2>
  <button class="clear"
    (click)="messageService.clear()">Clear messages</button>
  <div *ngFor="let message of messageService.messages"> {{message}} </div>
</div>
```

Tour of Heroes App 작성 : (5) Services

(5)-9. MessageComponent에서 MessageService 호출하기

src/app/messages/messages.component.css

```
/* MessagesComponent's private CSS styles */
h2 {
  color: #A80000; font-family: Arial, Helvetica, sans-serif; font-weight: lighter;
}
.clear {
  color: #333; background-color: #eee;
  margin-bottom: 12px; padding: 1rem;
  border-radius: 4px; font-size: 1rem;
}
.clear:hover {
  color: white; background-color: #42545C;
}
```

Tour of Heroes App 작성 : (5) Services

(5)-4. HeroesComponent에서 MessageService 호출하기

src/app/heroes/heroes.component.ts

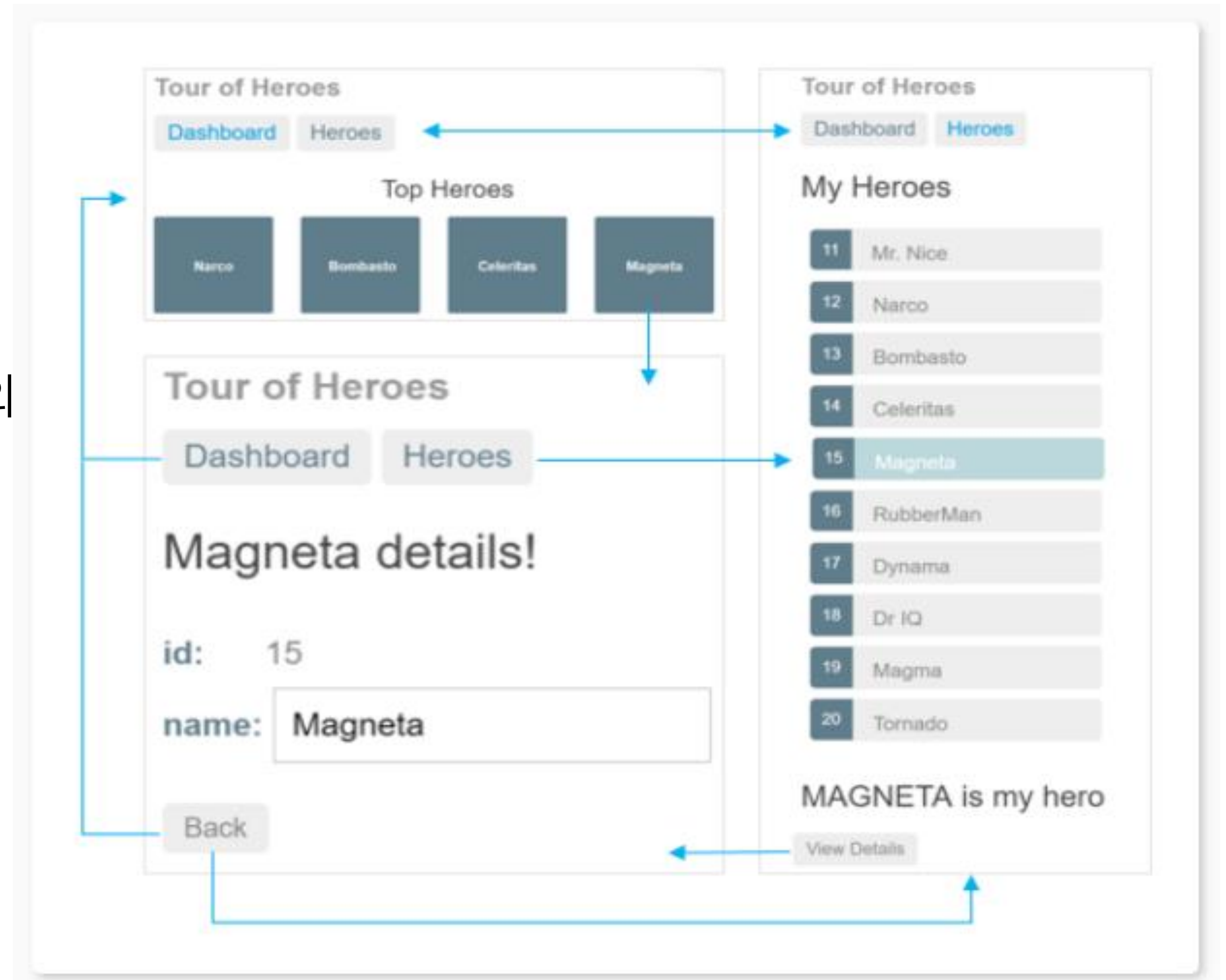
```
import { HeroService } from '../hero.service';
export class HeroesComponent implements OnInit {
  selectedHero?: Hero;
  heroes: Hero[] = [];

  constructor(private heroService: HeroService, private messageService: MessageService) {
  }
  ngOnInit() {
    this.getHeroes();
  }
  onSelect(hero: Hero): void {
    this.selectedHero = hero;
    this.messageService.add(`HeroesComponent: Selected hero id=${hero.id}`);
  }
  getHeroes(): void {
    this.heroService.getHeroes().subscribe(heroes => this.heroes = heroes);
  }
}
```


Tour of Heroes App 작성 : (6) Routing

(6)-1. 새로운 요구사항 추가

- a. Dashboard view 추가.
- b. Heroes와 Dashboard view 사이에 navigate 할 수 있는 기능 추가.
- c. hero name을 클릭하면 선택 된 hero의 detail view를 navigate 할 수 있다.



Tour of Heroes App 작성 : (6) Routing

(6)-1. AppRoutingModuleModule Component 작성

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

--flat 옵션은 디렉토리를 생성하지 않고 app-routing.module.ts 파일을 src/app 폴더에 생성 되도록 해준다.

--module=app 옵션은 AppModule의 imports array에 등록 되도록 해준다.

src/app/app-routing.module.ts (generated)

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
@NgModule ({
  imports: [ CommonModule ],
  declarations: []
})
export class AppRoutingModule { }
```

Tour of Heroes App 작성 : (6) Routing

(6)-1. AppRoutingModuleModule Component 작성

Angular Routes의 두 가지 property

- path: URL과 매칭되는 문자열
- component: path에서 지정한 URL이 선택 되었을 때 생성되는 Component 클래스의 이름

src/app/app-routing.module.ts (v1)

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

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

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

Tour of Heroes App 작성 : (6) Routing

(6)-1. AppRoutingModuleModule Component 작성

A 태그 routerLink 속성의 "/heroes" 문자열은 HeroesComponent와 매핑된다.

<app-heroes> Element를 <router-outlet> Element로 교체해야 한다.

HeroesComponent의 Template View가 <router-outlet> Element 부분에 출력이 된다.

src/app/app-component.html

```
<h1> {{title}} </h1>
<nav>
  <a routerLink="/heroes">Heroes</a>
</nav>

<!--<app-heroes></app-heroes>-->
<router-outlet> </router-outlet>
<app-messages></app-messages>
```

Tour of Heroes App 작성 : (6) Routing

(6)-2. dashboard view Component 작성

```
ng generate component dashboard
```

Routing을 적용하기 위해서 새로운 Component 가 필요하다.

src/app/dashboard/dashboard.component.ts

```
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: [ './dashboard.component.css' ]
})
export class DashboardComponent implements OnInit {
  heroes: Hero[] = [];
  constructor(private heroService: HeroService) { }
  ngOnInit() {
    this.getHeroes();
  }
  getHeroes(): void {
    this.heroService.getHeroes()
      .subscribe(heroes => this.heroes = heroes.slice(1, 5));
  }
}
```

Tour of Heroes App 작성 : (6) Routing

(6)-2. dashboard view Component 작성

src/app/dashboard/dashboard.component.html

```
<h3>Top Heroes</h3>
<div class="heroes-menu">
  <a *ngFor="let hero of heroes">
    {{hero.name}}
  </a>
</div>
```

Tour of Heroes App 작성 : (6) Routing

(6)-2. dashboard CSS Component 작성

src/app/dashboard/dashboard.component.css

```
/* DashboardComponent's private CSS styles */
h2 { text-align: center;}

.heroes-menu {
  padding: 0; margin: auto; max-width: 1000px;
  /* flexbox */
  display: flex; flex-direction: row; flex-wrap: wrap;
  justify-content: space-around; align-content: flex-start; align-items: flex-start;
}
a {
  background-color: #3f525c; border-radius: 2px; padding: 1rem;
  font-size: 1.2rem; text-decoration: none; display: inline-block;
  color: #fff; text-align: center; width: 100%;
  min-width: 70px; margin: .5rem auto; box-sizing: border-box;
  /* flexbox */
  order: 0; flex: 0 1 auto; align-self: auto;
}
@media (min-width: 600px) {
  a { width: 18%; box-sizing: content-box; }
}

a:hover { background-color: #000; }
```

Tour of Heroes App 작성 : (6) Routing

(6)-2. AppRoutingModule에 DashboardComponent와 default route 추가한다.

src/app/app-routing.module.ts (v2)

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HeroesComponent } from './heroes/heroes.component';
import { DashboardComponent } from './dashboard/dashboard.component';

const routes: Routes = [
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'heroes', component: HeroesComponent }
];

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


Tour of Heroes App 작성 : (6) Routing

(6)-2. app.component.html에 dashboard link 추가한다.

src/app/app-component.html

```
<h1> {{title}} </h1>
<nav>
  <a routerLink="/dashboard">Dashboard</a>
  <a routerLink="/heroes">Heroes</a>
</nav>

<!--<app-heroes></app-heroes>-->
<router-outlet></router-outlet>
<app-messages></app-messages>
```

Tour of Heroes App 작성 : (6) Routing

(6)-2. AppComponent CSS Component 작성

src/app/app-component.css

```
/* AppComponent's private CSS styles */
h1 {
  margin-bottom: 0;
}
nav a {
  padding: 1rem; text-decoration: none;
  margin-top: 10px; margin-left: 10px; display: inline-block;
  background-color: #e8e8e8; color: #3d3d3d;
  border-radius: 4px;
}
nav a:hover {
  color: white; background-color: #42545C;
}
nav a.active {
  background-color: black;
}
```

Tour of Heroes App 작성 : (6) Routing

(6)-3. AppRoutingModuleModule에 HeroDetailComponent route 추가한다.

src/app/app-routing.module.ts (v3)

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HeroesComponent } from './heroes/heroes.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { HeroDetailComponent } from './hero-detail/hero-detail.component';

const routes: Routes = [
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'heroes', component: HeroesComponent },
  { path: 'detail/:id', component: HeroDetailComponent }
];

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

Tour of Heroes App 작성 : (6) Routing

(6)-3. dashboard.component.html에 HeroDetailComponent를 호출하는 link 추가한다.

src/app/dashboard/dashboard-component.html

```
<a *ngFor="let hero of heroes"
  routerLink="/detail/{{hero.id}}"> {{hero.name}} </a>
```

(6)-3. heroes.component.html에 HeroDetailComponent를 호출하는 link 추가한다.

src/app/heroes/heroes-component.html (기존 : 변경 전 list with onSelect)

```
<ul class="heroes">
  <li *ngFor="let hero of heroes"
    [class.selected]="hero === selectedHero" (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
```

src/app/heroes/heroes-component.html (변경 후 list with links)

```
<ul class="heroes">
  <li *ngFor="let hero of heroes">
    <a routerLink="/detail/{{hero.id}}">
      <span class="badge">{{hero.id}}</span> {{hero.name}}
    </a>
  </li>
</ul>
```

Tour of Heroes App 작성 : (6) Routing

(6)-3. heroes.component.ts에 onSelect() 메서드를 제거한다.

src/app/heroes/heroes.component.ts (Cleaned up)

```
export class HeroesComponent implements OnInit {
  heroes: Hero[] = [];

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }

  getHeroes(): void {
    this.heroService.getHeroes()
      .subscribe(heroes => this.heroes = heroes);
  }
}
```

Tour of Heroes App 작성 : (6) Routing

(6)-4. Routable 한 HeroDetailComponent로 수정

~/detail/11 URL이 요청되었을 때 HeroDetailComponent 가 생성 되어진다.

HeroDetailComponent는 상세정보를 제공하기 위한 새로운 방법이 필요하다.

- 1) 생성자에서 주입 받은 ActivatedRoute는 HeroDetailComponent의 id 값을 가지고 있다.
- 2) ActivatedRoute로 부터 id 값을 추출해야 한다.

`this.route.snapshot.paramMap.get('id')`

- 3) 추출한 id로 생성자에서 주입 받은 HeroService에게 getHero(id) 메서드를 요청해야 한다.

Tour of Heroes App 작성 : (6) Routing

(6)-4. Routable 한 HeroDetailComponent로 수정

src/app/hero-detail/hero-detail.component.ts

```
import {ActivatedRoute} from '@angular/router';
import {Location} from '@angular/common';

@Component({
  selector: 'app-hero-detail',
  templateUrl: './hero-detail.component.html',
  styleUrls: [ './hero-detail.component.css' ]
})
export class HeroDetailComponent implements OnInit {
  @Input() hero: Hero;
  constructor( private route: ActivatedRoute, private heroService: HeroService, private location: Location ) { }

  ngOnInit(): void { this.getHero(); }

  getHero(): void {
    const id = Number(this.route.snapshot.paramMap.get('id'));
    this.heroService.getHero(id)
      .subscribe(hero => this.hero = hero);
  }
}
```

Tour of Heroes App 작성 : (6) Routing

(6)-4. HeroService에 getHero() 메서드 추가

src/app/hero.service.ts

```
import { MessageService } from './message.service'
@Injectable({ providedIn: 'root'})
export class HeroService {
  constructor(private messageService:MessageService) { }

  getHeroes(): Observable<Hero[]> {
    //send the message _after_ fetching the heroes
    const heroes = of(HEROES);
    this.messageService.add('HeroService: fetched heroes');
    return heroes;
  }
  getHero(id: number): Observable<Hero> {
    // Todo: send the message _after_ fetching the hero
    const hero = HEROES.find(h => h.id === id)!;
    this.messageService.add(`HeroService: fetched hero id=${id}`);
    return of(hero);
  }
}
```


Tour of Heroes App 작성 : (6) Routing

(6)-5. hero-detail.component.html에 back 버튼을 추가한다.

src/app/hero-detail/hero-detail.component.html (back button)

```
<button (click)="goBack()">go back</button>
```

src/app/hero-detail/hero-detail.component.ts

```
goBack(): void {  
  this.location.back();  
}
```

Tour of Heroes App 작성 : (6) Routing

(7)-2. hero-detail.component CSS Component 작성

src/app/hero-detail/hero-detail.component.css

```
/* HeroDetailComponent's private CSS styles */
label {
  color: #435960; font-weight: bold;
}
input {
  font-size: 1em; padding: .5rem;
}
button {
  margin-top: 20px; background-color: #eee;
  padding: 1rem; border-radius: 4px; font-size: 1em;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee; color: #ccc; cursor: auto;
}
```

Tour of Heroes App 작성 : (6) Routing

(7)-3. heroes.component의 변경된 css style

src/app/heroes/heroes.component.css

```
/* HeroesComponent's private CSS styles */
.heroes { margin: 0 0 2em 0; list-style-type: none; padding: 0; width: 15em;}
.heroes li { position: relative; cursor: pointer; }
.heroes li:hover { left: .1em;}
.heroes a {
  color: #333; text-decoration: none; background-color: #EEE; margin: .5em;
  padding: .3em 0; height: 1.6em; border-radius: 4px; display: block; width: 100%;
}
.heroes a:hover {
  color: #2c3a41; background-color: #e6e6e6;
}
.heroes a:active {
  background-color: #525252; color: #fafafa;
}
.heroes .badge {
  display: inline-block; font-size: small; color: white; padding: 0.8em 0.7em 0 0.7em;
  background-color:#405061; line-height: 1em; position: relative; left: -1px;
  top: -4px; height: 1.8em; min-width: 16px; text-align: right; margin-right: .8em; border-radius: 4px 0 0 4px;
}
input {
  display: block; width: 100%; padding: .5rem; margin: 1rem 0; box-sizing: border-box;
}
```

Tour of Heroes App 작성 : (7) HTTP

HttpClient는 HTTP를 통해 원격 서버와 통신할 수 있는 컴포넌트

(7)-1. HttpClient를 이용하여 구현해야 하는 기능

- 1) HeroService는 HTTP Requests를 사용하여 hero data를 가져올 수 있다.
- 2) HTTP를 통해 heroes를 추가, 수정, 삭제 할 수 있다.
- 3) HTTP를 통해 name(이름)으로 heroes를 검색할 수 있다.

(7)-1. HttpClient를 사용하기 위해 필요한 설정

src/app/app.module.ts

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [ HttpClientModule ],
})
export class AppModule { }
```

Tour of Heroes App 작성 : (7) HTTP

(7)-2. HeroService를 수정 : HttpClient와 MessageService를 주입 받기

서버를 호출하기 위해 private http 속성에 HttpClient를 생성자에서 주입(inject)합니다.

메시지를 출력하기 위해 private messageService 속성에 MessageService를 주입(inject)합니다.

src/app/hero.service.ts

```
import { HttpClient, HttpHeaders } from '@angular/common/http';

@Injectable()
export class HeroService {

  constructor(private http: HttpClient, private messageService: MessageService) { }

  /** Log a HeroService message with the MessageService */
  private log(message: string) {
    this.messageService.add('HeroService: ' + message);
  }
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-3. HeroService를 수정 : getHeroes() 메서드 수정 – HttpClient.get() 메서드 사용

- ① 수정하기 전의 HeroService.getHeroes()는 RxJS of() 함수를 사용 하였으나, 수정한 후의 getHeroes()에서는 HttpClient의 get() 함수를 사용한다.
- ② HttpClient.get()은 untyped JSON 객체로 response body를 반환합니다.
JSON 데이터의 shape는 서버의 데이터 API에 의해 결정됩니다.
Tour of Heroes 데이터 API는 hero data를 배열로 반환합니다.

(Observable의 subscribe() 메서드 사용하기)



src/app/hero.service.ts

```
private heroesUrl = 'http://localhost:8087/heroes'; // URL to web api

/** GET heroes from the server */
getHeroes (): Observable<Hero[]> {
  return this.http.get<Hero[]>(this.heroesUrl);
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-3-1. HeroService를 수정 : handleError() 메서드 추가

error를 직접 처리하는 대신 fail된 operation의 name과 return 값으로 구성된 catchError에 오류 처리기 함수를 반환합니다.

src/app/hero.service.ts

```
/**
 * Handle Http operation that failed.
 * @param operation - name of the operation that failed, @param result - optional value to return as the
observable result
 */
private handleError<T> (operation = 'operation', result?: T) {
  return (error: any): Observable<T> => {
    // TODO: send the error to remote logging infrastructure
    console.error(error); // log to console instead
    // TODO: better job of transforming error for user consumption
    this.log(`${operation} failed: ${error.message}`);
    // Let the app keep running by returning an empty result.
    return of(result as T);
  };
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-3-2. HeroService를 수정 : getHeroes() 메서드 수정

: Observable의 pipe(), tap(), catchError() 메서드 사용

- ① pipe() 메서드는 여러 개의 순수 함수들을 연결하고 쉽게 공유 논리를 다시 사용하여 훨씬 더 재사용 가능한 RxJS 코드를 작성할 수 있도록 해줍니다.
- ② tap() 메서드는 pipe 라인의 메시지에 대해 임의의 작업을 실행하기 위해 message stream을 가로 채서 쿼리 동작을 디버깅, 로깅하는 데 사용할 수 있습니다.

src/app/hero.service.ts

```
import { Observable, of } from 'rxjs';
import { catchError, tap } from 'rxjs/operators';
/** GET heroes from the server */
getHeroes (): Observable<Hero[]> {
  return this.http.get<Hero[]>(this.heroesUrl)
    .pipe(
      tap(_ => this.log(`fetched heroes`)),
      catchError(this.handleError<Hero[]>('getHeroes', []))
    );
}
```


Tour of Heroes App 작성 : (7) HTTP

(7)-3-3. HeroService를 수정 : getHero() 메서드 수정 – HttpClient.get() 메서드 사용

- ① getHero()에서는 HttpClient의 get() 함수를 사용한다.
- ② Observable의 pipe(), tap(), catchError() 메서드 사용

src/app/hero.service.ts

```
private heroesUrl = 'http://localhost:8087/heroes'; // URL to web api

/** GET hero by id. Will 404 if id not found */
getHero(id: number): Observable<Hero> {
  const url = `${this.heroesUrl}/${id}`;
  return this.http.get<Hero>(url).pipe(
    tap(_ => this.log(`fetched hero id=${id}`)),
    catchError(this.handleError<Hero>(`getHero id=${id}`))
  );
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-4. Hero를 Update 하는 기능 추가

- ① Hero detail 템플릿의 마지막 부분에 `save()` 메서드를 호출하는 저장 버튼을 추가합니다.

src/app/hero-detail/hero-detail.component.html (SAVE button)

```
<button (click)="save()">save</button>
```

- ② Hero detail 컴포넌트에 `save()` 메서드를 추가합니다.

src/app/hero-detail/hero-detail.component.ts

```
save(): void {  
  if (this.hero) {  
    this.heroService.updateHero(this.hero)  
      .subscribe() => this.goBack();  
  }  
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-4-1. Hero를 Update 하는 기능 추가 : HeroService에 updateHero() 메서드 추가

- ① updateHero() 메서드는 http.put()을 사용하여 변경된 hero 정보를 서버에 저장함
- ② HttpClient.put() 메서드의 세가지 파라미터는 URL, 변경할 데이터, options 이다.
- ③ 세번째 파라미터인 options 헤더는 httpOptions constant로 정의합니다.

src/app/hero.service.ts

```
httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};
/** PUT: update the hero on the server */
updateHero (hero: Hero): Observable<Hero> {
  const url = `${this.heroesUrl}/${hero.id}`;
  return this.http.put(url, hero, this.httpOptions).pipe(
    tap((hero: Hero) => this.log(`updated hero id=${hero.id}`)),
    catchError(this.handleError<Hero>('updateHero'))
  );
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-5. Hero를 Add 하는 기능 추가 : heroes.component.html 수정

① Hero를 추가하려면 hero의 name 만 있으면 됩니다. 컴포넌트의 add() 메서드 호출

src/app/heroes/heroes.component.html (add button)

```
<div>
  <label for="new-hero">Hero name: </label>
  <input id="new-hero" #heroName />

  <!-- (click) passes input value to add() and then clears the input -->
  <button class="add-button" (click)="add(heroName.value); heroName.value=""> Add hero</button>
</div>
```

src/app/heroes/heroes.component.css

```
.add-button {
  padding: .5rem 1.5rem; font-size: 1rem; margin-bottom: 2rem;
}
.add-button:hover {
  color: white; background-color: #42545C;
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-5-1. Hero를 Add 하는 기능 추가 : heroes.component.ts 수정

- ① 핸들러는 주어진 이름으로 hero 오브젝트를 생성하고 (id가 누락 된 것만) 서비스 addHero() 메소드에 전달합니다.
- ② addHero가 성공적으로 저장되면 subscribe 콜백은 새로운 hero를 subscribe하고 표시하기 위해 hero list에 push 합니다.

src/app/heroes/heroes.component.ts

```
add(name: string): void {  
  name = name.trim();  
  if (!name) { return; }  
  this.heroService.addHero({ name } as Hero)  
    .subscribe(hero => {  
    this.heroes.push(hero);  
  });  
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-5-2. Hero를 Add 하는 기능 추가 : hero.service.ts 에 addHero() 메서드 추가

- ① addHero()는 put() 대신 HttpClient.post()를 호출합니다.
- ② Observable <Hero>에서 caller에게 반환하는 새로운 hero에 대한 ID를 서버가 생성 할 것입니다.

src/app/hero.service.ts

```
/** POST: add a new hero to the server */
addHero(hero: Hero): Observable<Hero> {
  return this.http.post<Hero>(this.heroesUrl, hero, this.httpOptions).pipe(
    tap((newHero: Hero) => this.log(`added hero w/ id=${newHero.id}`)),
    catchError(this.handleError<Hero>('addHero'))
  );
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-6. Hero를 Delete 하는 기능 추가 : heroes.component.html 삭제

- ① hero list에있는 각 hero는 delete 버튼이 있어야 합니다.
- ② 반복 된 요소의 hero name 뒤에 delete 버튼 엘리먼트를 추가합니다.

src/app/heroes/heroes.component.html

```
<ul class="heroes">
  <li *ngFor="let hero of heroes">
    <a routerLink="/detail/{{hero.id}}">
      <span class="badge">{{hero.id}}</span> {{hero.name}}
    </a>
    <button class="delete" title="delete hero"
      (click)="delete(hero)">x</button>
  </li>
</ul>
```

Tour of Heroes App 작성 : (7) HTTP

(7)-6-1. Hero를 Delete 하는 기능 추가 : heroes.component.css 삭제

① hero 항목의 맨 오른쪽에 삭제 버튼을 배치하려면 heroes.component.css 수정

src/app/heroes/heroes.component.css

```
button.delete {  
  position: absolute; left: 210px; top: 5px;  
  background-color: white; color: #525252; font-size: 1.1rem;  
  padding: 1px 10px 3px 10px;  
}  
button.delete:hover {  
  background-color: #525252;  
  color: white;  
}
```


Tour of Heroes App 작성 : (7) HTTP

(7)-6-2. Hero를 Delete 하는 기능 추가 : heroes.component.ts 삭제

- ① HeroService에 hero 삭제를 위임 하더라도 자체 hero list를 업데이트해야 합니다.
- ② 구성 요소의 delete() 메서드는 HeroService가 서버에서 성공할 것으로 예상하여 해당 목록에서 삭제할 hero를 즉시 제거합니다.

src/app/heroes/heroes.component.ts

```
delete(hero: Hero): void {  
  this.heroes = this.heroes.filter(h => h !== hero);  
  this.heroService.deleteHero(hero.id).subscribe();  
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-6-3. Hero를 Delete 하는 기능 추가 : hero.service.ts 수정

- ① HttpClient.delete() 메서드 호출
- ② 첫번째 파라미터 URL은 hero resource URL과 삭제할 hero의 ID입니다

src/app/hero.service.ts

```
/** DELETE: delete the hero from the server */
deleteHero(id:number): Observable<Hero> {
  const url = `${this.heroesUrl}/${id}`;

  return this.http.delete<Hero>(url, this.httpOptions).pipe(
    tap(_ => this.log(`deleted hero id=${id}`)),
    catchError(this.handleError<Hero>('deleteHero'))
  );
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-7. Hero를 Search 하는 기능 추가 : heroes.service.ts에 searchHeroes() 메서드 추가

- ① 검색어가 없는 경우 빈 배열을 반환합니다. getHeroes()와 매우 유사합니다.
- ② 중요한 차이점은 검색어가 포함 된 검색어 문자열을 포함하는 URL입니다.

src/app/hero.service.ts

```
/* GET heroes whose name contains search term */
searchHeroes(term: string): Observable<Hero[]> {
  if (!term.trim()) {
    // if not search term, return empty hero array.
    return of([]);
  }
  return this.http.get<Hero[]>(`${this.heroesUrl}/?name=${term}`).pipe(
    tap(x => x.length ?
      this.log(`found heroes matching "${term}"`) :
      this.log(`no heroes matching "${term}"`)),
    catchError(this.handleError<Hero[]>('searchHeroes', []))
  );
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-7-1. Hero를 Search 하는 기능 추가 : Dashboard에 search 하는 기능 추가

- ① DashboardComponent 템플릿의 맨 아래에 hero 검색 요소 인 <app-hero-search>를 추가합니다.

src/app/dashboard/dashboard.component.html

```
<h2>Top Heroes</h2>
<div class="heroes-menu">
  <a *ngFor="let hero of heroes" routerLink="/detail/{{hero.id}}">
    {{hero.name}}
  </a>
</div>

<app-hero-search> </app-hero-search>
```

Tour of Heroes App 작성 : (7) HTTP

(7)-7-2. Hero를 Search 하는 기능 추가 : HeroSearchComponent 생성

- ① CLI는 3가지 HeroSearchComponent 생성하고, AppModule 선언에 컴포넌트를 추가함

```
ng generate component hero-search
```

```
src/app/hero-search/hero-search.component.html
```

```
<div id="search-component">
  <label for="search-box">Hero Search</label>

  <input #searchBox id="search-box" (input)="search(searchBox.value)" />

  <ul class="search-result">
    <li *ngFor="let hero of heroes$ | async" >
      <a routerLink="/detail/{{hero.id}}"> {{hero.name}} </a>
    </li>
  </ul>
</div>
```

Tour of Heroes App 작성 : (7) HTTP

(7)-7-2. Hero를 Search 하는 기능 추가 : hero-search.component.html

① AsyncPipe

```
<li *ngFor="let hero of heroes$ | async" >
```

- : heros\$에서 \$는 배열이 아닌 Observable임을 나타내는 관례입니다.
- : 파이프 문자(|) 뒤의 async는 Angular의 AsyncPipe를 나타냅니다.
- : AsyncPipe는 Observable에 자동으로 가입하므로 구성 요소 클래스에서 Observable을 수행 할 필요가 없습니다.

Tour of Heroes App 작성 : (7) HTTP

(7)-7-3. Hero를 Search 하는 기능 추가 : HeroSearchComponent에 CSS 추가

src/app/hero-search/hero-search.component.css

```
/* HeroSearch private styles */
label {
  display: block; font-weight: bold; font-size: 1.2rem; margin-top: 1rem; margin-bottom: .5rem;
}
input {
  padding: .5rem; width: 100%; max-width: 600px; box-sizing: border-box; display: block;
}
input:focus { outline: #336699 auto 1px;}
li { list-style-type: none;}
.search-result li a {
  border-bottom: 1px solid gray; border-left: 1px solid gray;
  border-right: 1px solid gray; display: inline-block; width: 100%; max-width: 600px;
  padding: .5rem; box-sizing: border-box; text-decoration: none; color: black;
}
.search-result li a:hover { background-color: #435A60; color: white; }
ul.search-result { margin-top: 0; padding-left: 0;}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-7-4. Hero를 Search 하는 기능 추가 : HeroSearchComponent 컴포넌트

src/app/hero-search/hero-search.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Observable, Subject } from 'rxjs';
import { debounceTime, distinctUntilChanged, switchMap } from 'rxjs/operators';
import { Hero } from '../hero';
import { HeroService } from '../hero.service';

@Component({
  selector: 'app-hero-search',
  templateUrl: './hero-search.component.html',
  styleUrls: [ './hero-search.component.css' ]
})
```


Tour of Heroes App 작성 : (7) HTTP

(7)-7-4. Hero를 Search 하는 기능 추가 : HeroSearchComponent 컴포넌트

- ① searchTerms 속성이 RxJS Subject로 선언 되었습니다.
- ② Subject는 observable values의 source이며 Observable 자체입니다.
Observable과 같이 Subject를 구독 할 수 있으며, next(value) 메서드를 호출하여 Observable로 값을 보낼 수 있습니다.

src/app/hero-search/hero-search.component.ts

```
export class HeroSearchComponent implements OnInit {  
  heroes$: Observable<Hero[]>;  
  private searchTerms = new Subject<string>();  
  
  constructor(private heroService: HeroService) {}  
  // Push a search term into the observable stream.  
  search(term: string): void {  
    this.searchTerms.next(term);  
  }  
}
```

Tour of Heroes App 작성 : (7) HTTP

(7)-7-4. Hero를 Search 하는 기능 추가 : HeroSearchComponent 컴포넌트

- ① `debounceTime(300)`은 300 밀리 초 동안 기다린 후 최신 문자열을 전달합니다.
- ② `distinctUntilChanged`는 필터 텍스트가 변경된 경우에만 요청을 보냅니다.
- ③ `switchMap()`은 `debounce` 및 `distinctUntilChanged`를 통해 검색하는 각 검색어에 대해 검색 서비스를 호출합니다

src/app/hero-search/hero-search.component.ts

```
ngOnInit(): void {  
  this.heroes$ = this.searchTerms.pipe(  
    // wait 300ms after each keystroke before considering the term  
    debounceTime(300),  
    // ignore new term if same as previous term  
    distinctUntilChanged(),  
    // switch to new search observable each time the term changes  
    switchMap((term: string) => this.heroService.searchHeroes(term)),  
  );  
}
```

Tour of Heroes App 작성 Summary

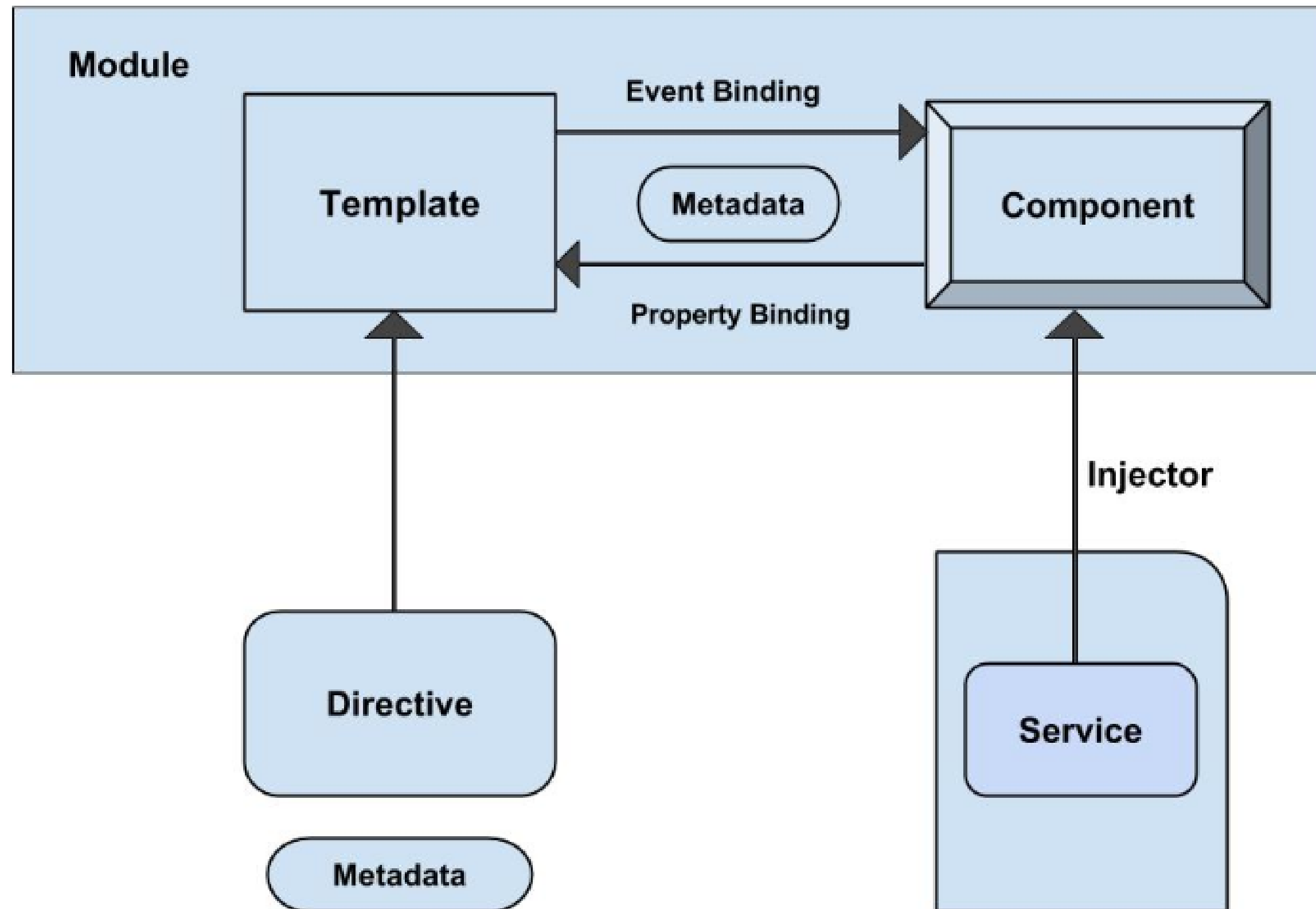
Tour of Heroes App 작성하면서 배운 사항들

- ① 앱에서 HTTP를 사용하기 위해 필요한 의존성을 추가 했습니다.
- ② 웹 API에서 hero를 로드 하기 위해 HeroService를 리팩터링 하였습니다.
- ③ post(), put() 및 delete() 메소드를 지원 하도록 HeroService를 확장했습니다.
- ④ hero을 adding, editing 및 deleting할 수 있도록 구성 요소를 업데이트 했습니다.
- ⑤ Observables 객체의 사용법을 배웠습니다.

Angular App 작성

: User CRUD App

Angular Architecture



Angular CRUD application 작성

- App을 통해 구현되는 내용들
 - 1) Spring Boot를 사용하여 REST 서비스를 작성한다.
 - 2) User(사용자) CRUD Angular 웹 애플리케이션을 작성한다.
 - 3) Angular CLI를 사용하여 클라이언트 웹 애플리케이션을 스켈레톤 코드를 생성한다.
 - 4) 화면 전환을 위해서 Routing 기능을 사용한다.
 - 5) 서버의 REST 서비스와 통신하기 위해 Angular의 HttpClient와 RxJS의 Observable 객체를 사용한다.
 - 6) 등록/수정 Form은 Angular의 Reactive Form를 사용한다.

Angular CRUD application 작성

User

Users

New User

#	First Name	Last Name	email	
1	John	Doe	john@email.com	<div>EditDelete</div>
2	Jon	Smith	smith@email.com	<div>EditDelete</div>
3	Will	Craig	will@email.com	<div>EditDelete</div>
4	Sam	Lernorad	sam@email.com	<div>EditDelete</div>
5	Ross	Doe	ross@email.com	<div>EditDelete</div>

Angular CRUD application 작성

First Name

Last Name

Email

```
{  
  "firstName": "John",  
  "lastName": "Doe",  
  "email": "john@email.com"  
}
```


Angular CRUD application 작성 : 프로젝트 생성

- 새로운 Angular Application Project 생성

```
ng new user-app --routing
```

- Application 실행

```
cd user-app  
ng serve --open
```

- Application 구성요소
 - 1) app.component.ts – TypeScript로 작성된 component class
 - 2) app.component.html – HTML로 작성된 component template
 - 3) app.component.css – 현재의 Component에만 적용되는 CSS style

Angular CRUD application 작성 : 컴포넌트 생성

- User Module 생성

```
ng generate module user --routing
```

- User Service 생성

```
ng generate service user/user
```

- User Component 생성

```
ng generate component user/user-list
```

```
ng generate component user/user-create
```

Angular CRUD application 작성 : Routing Module

(1). User Routing Module 작성하기

src/app/user/user-routing.module.ts

```
import { Routes, RouterModule } from '@angular/router';
import { UserListComponent } from '../user-list/user-list.component';
import { UserCreateComponent } from '../user-create/user-create.component';

const routes: Routes = [
  {path: 'user', component: UserListComponent},
  {path: 'user/create', component: UserCreateComponent},
  {path: 'user/edit/:id', component: UserCreateComponent}
];

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

Angular CRUD application 작성 : User Module

(2). User Module에 FormsModule과 ReactiveFormsModule, HttpClientModule 추가하기

src/app/user/user.module.ts

```
import {NgModule} from '@angular/core';
import {CommonModule} from '@angular/common';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';
import {HttpClientModule} from '@angular/common/http';

import {UserRoutingModule} from './user-routing.module';
import {UserListComponent} from './user-list/user-list.component';
import {UserCreateComponent} from './user-create/user-create.component';

@NgModule({
  imports: [
    CommonModule,
    UserRoutingModule,
    FormsModule,
    ReactiveFormsModule,
    HttpClientModule
  ],
  declarations: [UserListComponent, UserCreateComponent]
})
export class UserModule { }
```

Angular CRUD application 작성 : Root Module

(3). Root Module에 User Module 추가하기

src/app/app.module.ts

```
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { UserModule } from './user/user.module';

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

Angular CRUD application 작성 : Root Template

(4). Root Template 수정하기

src/app/app.component.html

```
<nav>
  <a routerLink="/user" routerLinkActive="active">Users</a>
  <a routerLink="/user/create" routerLinkActive="active">New User</a>
</nav>
<router-outlet></router-outlet>
```

(4-1). 실행하기

```
ng serve
```



Angular CRUD application 작성 : Styling

(5). Bootstrap 4 설치하기

```
npm install bootstrap font-awesome --save
```

(5-1). Bootstrap css import 하기

src/styles.css

```
@import "~bootstrap/dist/css/bootstrap.min.css";  
@import "~font-awesome/css/font-awesome.css";
```

Angular CRUD application 작성 : Model 클래스 작성

(6). User Model 클래스

src/app/user/user.ts

```
export class User {  
  
  id: number;  
  firstName: string;  
  lastName: string;  
  email: string;  
  
  constructor(id: number, firstName: string, lastName: string, email: string) {  
    this.id = id;  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.email = email;  
  }  
}
```


Angular CRUD application 작성 : 사용자 목록 조회 기능

(7-1). User Service 컴포넌트

src/app/user/user.service.ts

```
import { Injectable } from '@angular/core';
import { User } from '../user';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  private apiUrl = 'http://localhost:8087/users';

  constructor(private http: HttpClient) { }

  findAll(): Observable<User[]> {
    return this.http.get<User[]>(this.apiUrl);
  }
}
```

Angular CRUD application 작성 : 사용자 목록 조회 기능

(7-2). User-List UI 컴포넌트 작성

src/app/user/user-list/user-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from "../user";
import { UserService } from "../user.service";
@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css'],
  providers: [UserService]
})
export class UserListComponent implements OnInit {
  private users: User[];
  constructor(private userService: UserService) { }
  ngOnInit() { this.getAllUsers(); }
  getAllUsers() {
    this.userService.findAll().subscribe(
      users => {
        this.users = users;
      },
      err => {
        console.log(err);
      }
    );
  }
}
```

Angular CRUD application 작성 : 사용자 목록 조회 기능

(7-3). User-List UI 템플릿 작성 # 1

src/app/user/user-list/user-list.component.html

```
<div class="container">
  <div class="row">
    <div class="col">
      <section>
        <header class="header">
          <div class="row">
            <div class="col-md-4">
              <h1>Users</h1>
            </div>
            <div class="col-md-6">

            </div>
            <div class="col-md-2">

            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Angular CRUD application 작성 : 사용자 목록 조회 기능

(7-3). User-List UI 템플릿 작성 # 2

src/app/user/user-list/user-list.component.html

```
<section class="main">
  <table class="table">
    <thead>
      <tr>
        <th>#</th>
        <th>First Name</th><th>Last Name</th><th>email</th><th></th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let user of users">
        <th scope="row">{{user.id}}</th>
        <td>{{user.firstName}}</td>
        <td>{{user.lastName}}</td>
        <td>{{user.email}}</td>
        <td>
        </td>
      </tr>
    </tbody>
  </table>
</section>
</div>
</div>
</div>
```

Angular CRUD application 작성 : 사용자 등록 기능

(8-1). User-List UI 템플릿 작성 : 등록 버튼 추가

src/app/user/user-list/user-list.component.html

```
<div class="container">
  <div class="row">
    <div class="col">
      <section>
        <header class="header">
          <div class="row">
            <div class="col-md-4">
              <h1>Users</h1>
            </div>
            <div class="col-md-6">

              </div>
              <div class="col-md-2">
                <button type="button" class="btn btn-primary" (click)="redirectNewUserPage()">
                  New User</button>
              </div>
            </div>
          </div>
        </header>
      </section>
    </div>
  </div>
</div>
```

Angular CRUD application 작성 : 사용자 등록 기능

(8-2). User-List UI 컴포넌트 작성 : redirectNewUserPage() 메서드 추가

src/app/user/user-list/user-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from "../user";
import { UserService } from "../user.service";
import { Router } from '@angular/router';

@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css'],
  providers: [UserService]
})
export class UserListComponent implements OnInit {

  private users: User[];

  constructor(private router: Router,
               private userService: UserService) { }

  redirectNewUserPage() {
    this.router.navigate(['/user/create']);
  }
}
```

Angular CRUD application 작성 : 사용자 등록 기능

(8-3) User Service 컴포넌트 작성 : saveUser() 메서드 추가

src/app/user/user.service.ts

```
import { Injectable } from '@angular/core';
import { User } from '../user';
import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

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

  private apiUrl = 'http://localhost:8087/users';

  constructor(private http: HttpClient) {

  }

  saveUser(user: User): Observable<User> {
    return this.http.post<User>(this.apiUrl, user);
  }
}
```

Angular CRUD application 작성 : 사용자 등록 기능

(8-4). User-Create UI 템플릿 작성 : 등록 Form #1

src/app/user-create/user-create.component.html

```
<div class="container">
  <div class="row">
    <div class="col">
      <section>
        <form novalidate [formGroup]="userForm" (ngSubmit)="onSubmit()">
          <fieldset>
            <div class="form-group">
              <label>First Name</label>
              <input type="text" class="form-control" formControlName="firstName">
            </div>
            <div class="form-group">
              <label>Last Name</label>
              <input type="text" class="form-control" formControlName="lastName">
            </div>
          </fieldset>
        </form>
      </section>
    </div>
  </div>
</div>
```


Angular CRUD application 작성 : 사용자 등록 기능

(8-4). User-Create 템플릿 작성 : 등록 Form #2

src/app/user-create/user-create.component.html

```
<div class="form-group">
  <label>Email</label>
  <input type="email" class="form-control" formControlName="email">
</div>
<button type="submit" class="btn btn primary" [disabled]="!userForm.valid" >
  Submit</button>
<button type="button" class="btn btn-warning" (click)="redirectToUserPage()">
  Cancel</button>
<pre>{{userForm.value | json}}</pre>
</form>
</section>
</div>
</div>
</div>
```

Angular CRUD application 작성 : 사용자 등록 기능

(8-5). User-Create 컴포넌트 작성 : 등록 Form #1

src/app/user-create/user-create.component.ts

```
export class UserCreateComponent implements OnInit {  
  userForm: FormGroup;  
  
  constructor(private router: Router, private userService: UserService,  
    private location: Location) { }  
  
  ngOnInit() {  
    this.userForm = new FormGroup({  
      firstName: new FormControl('', validators.required),  
      lastName: new FormControl('', validators.required),  
      email: new FormControl('', [  
        validators.required,  
        validators.pattern('[^ @]*@[^ @]*')  
      ])  
    });  
  }  
}
```

Angular CRUD application 작성 : 사용자 등록 기능

(8-5). User-Create UI 컴포넌트 작성 : 등록 Form #2

src/app/user-create/user-create.component.ts

```
onSubmit() {  
  if (this.userForm.valid) {  
    const user: User = new User(null,  
      this.userForm.controls['firstName'].value,  
      this.userForm.controls['lastName'].value,  
      this.userForm.controls['email'].value);  
    this.userService.saveUser(user).subscribe(() => this.goBack());  
  }  
  this.userForm.reset();  
}  
  
goBack(): void {  
  this.location.back();  
}  
}
```

Angular CRUD application 작성 : 사용자 업데이트 기능

(9-1). User Service 컴포넌트 작성 : findById() / updateUser() 메서드 추가

src/app/user/user-service.ts

```
import { Injectable } from '@angular/core';
import { User } from '../user';
import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

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

  findById(id: number): Observable<User> {
    return this.http.get<User>(this.apiUrl + '/' + id);
  }

  updateUser(user: User): Observable<User> {
    return this.http.put<User>(this.apiUrl, user);
  }
}
```

Angular CRUD application 작성 : 사용자 업데이트 기능

(9-2). User-List UI 템플릿 작성 : 업데이트 버튼 추가

src/app/user/user-list/user-list.component.html

```
<section class="main">
  <table class="table">
    ..
    <tbody>
      <tr *ngFor="let user of users">
        <th scope="row">{{user.id}}</th>
        <td>{{user.firstName}}</td>
        <td>{{user.lastName}}</td>
        <td>{{user.email}}</td>
        <td>
          <button type="button" class="btn btn-success" (click)="editUserPage(user)">
            Edit</button>
        </td>
      </tr>
    </tbody>
  </table>
</section>
```

Angular CRUD application 작성 : 사용자 등록 기능

(9-3). User-List UI 컴포넌트 작성 : editUserPage() 메서드 추가

src/app/user/user-list/user-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from "../user";
import { UserService } from "../user.service";
import { Router } from '@angular/router';

@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css'],
  providers: [UserService]
})
export class UserListComponent implements OnInit {

  constructor(private router: Router,
               private userService: UserService) { }

  editUserPage(user: User) {
    if (user) {
      this.router.navigate(['/user/edit', user.id]);
    }
  }
}
```

Angular CRUD application 작성 : 사용자 업데이트 기능

(9-4). User-Create 컴포넌트 작성 : 업데이트 Form #1

src/app/user-create/user-create.component.ts

```
export class UserCreateComponent implements OnInit, OnDestroy {
  userForm: FormGroup;
  id: number;
  user: User;
  private sub: any;

  constructor(private route: ActivatedRoute, private router: Router, private userService:
    UserService, private location: Location) { }

  ngOnInit() {
  }

  ngOnDestroy(): void {
    this.sub.unsubscribe();
  }
}
```

Angular CRUD application 작성 : 사용자 업데이트 기능

(9-4). User-Create 컴포넌트 작성 : 업데이트 Form #2

src/app/user-create/user-create.component.ts

```
ngOnInit() {  
  this.sub = this.route.params.subscribe(params => {  
    this.id = params['id'];  
  });  
  
  if (this.id) {  
    this.userService.findById(this.id).subscribe(  
      user => {  
        this.id = user.id;  
        this.userForm.patchValue({  
          firstName: user.firstName,  
          lastName: user.lastName,  
          email: user.email,  
        });  
      }, error => {  
        console.log(error);  
      }  
    );  
  }  
}  
} //ngOnInit
```


Angular CRUD application 작성 : 사용자 업데이트 기능

(9-4). User-Create 컴포넌트 작성 : 업데이트 Form #4

src/app/user-create/user-create.component.ts

```
onSubmit() {  
  if (this.userForm.valid) {  
    if (typeof this.id !== 'undefined') {  
      const user: User = new User(this.id,  
        this.userForm.controls['firstName'].value,  
        this.userForm.controls['lastName'].value,  
        this.userForm.controls['email'].value);  
      this.userService.updateUser(user).subscribe(() => this.goBack());  
    } else {  
      const user: User = new User(null,  
        this.userForm.controls['firstName'].value,  
        this.userForm.controls['lastName'].value,  
        this.userForm.controls['email'].value);  
      this.userService.saveUser(user).subscribe(() => this.goBack());  
    }  
  
    this.userForm.reset();  
  }  
}
```

Angular CRUD application 작성 : 사용자 삭제 기능

(10-1). User Service 컴포넌트 작성 : deleteUserById() 메서드 추가

src/app/user/user-service.ts

```
import { Injectable } from '@angular/core';
import { User } from '../user';
import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

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

  deleteUserById(id: number): Observable<boolean> {
    return this.http.delete<boolean>(this.apiUrl + '/' + id);
  }
}
```

Angular CRUD application 작성 : 사용자 삭제 기능

(10-2). User-List UI 템플릿 작성 : 삭제 버튼 추가

src/app/user/user-list/user-list.component.html

```
<section class="main">
  <table class="table">
    ..
    <tbody>
      <tr *ngFor="let user of users">
        <th scope="row">{{user.id}}</th>
        <td>{{user.firstName}}</td>
        <td>{{user.lastName}}</td>
        <td>{{user.email}}</td>
        <td>
          <button type="button" class="btn btn-danger" (click)="deleteUser(user)">
            Delete</button>
        </td>
      </tr>
    </tbody>
  </table>
</section>
```

Angular CRUD application 작성 : 사용자 등록 기능

(10-3). User-List UI 컴포넌트 작성 : deleteUser () 메서드 추가

src/app/user/user-list/user-list.component.ts

```
export class UserListComponent implements OnInit {
  constructor(private router: Router,
               private userService: UserService) { }

  deleteUser(user: User) {
    const flag = confirm(user.firstName + ' ' + user.lastName + ' 사용자를 삭제하시겠습니까?');
    if (flag) {
      if (user) {
        this.userService.deleteUserById(user.id).subscribe(
          res => {
            this.getAllUsers();
            console.log('done');
          }
        );
      }
    } else {
      console.log('삭제 되지 않음');
    }
  }
}
```

Spring Boot REST Service

Spring Boot REST Service : Spring Boot Project 생성

- Spring Boot Application 작성

1) Spring_INITIALIZER (<https://start.spring.io/>)

2) 생성된 프로젝트 zip 파일을 저장하고, 압축을 푼다.

3) STS에서 File -> Open Project From File System으로 프로젝트 폴더를 연다.

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project ▾ with Java ▾ and Spring Boot 2.0.2 ▾

Project Metadata

Artifact coordinates

Group

myspringboot.user

Artifact

springboot-anular4

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web ×

Generate Project alt + ⌘

Spring Boot REST Service : Model 클래스 작성

- 1-1) User Model 클래스 작성 #1

src/main/java/myspringboot/user/User.java

```
package myspringboot.user;
public class User {
    private Long id;
    private String firstName;
    private String lastName;
    private String email;

    public User() { }

    public User(Long id, String firstName, String lastName, String email) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ",
email=" + email + "]\n";
    }
}
```

Spring Boot REST Service : Model 클래스 작성

- 1-2) User Model 클래스 작성 #2

src/main/java/myspringboot/user/User.java

```
public Long getId() { return id;}
public void setId(Long id) { this.id = id;}

public String getFirstName() { return firstName;}
public void setFirstName(String firstName) { this.firstName = firstName;}

public String getLastName() { return lastName;}
public void setLastName(String lastName) { this.lastName = lastName;}

public String getEmail() { return email; }
public void setEmail(String email) { this.email = email; }
}
```


Spring Boot REST Service : Controller 클래스 작성

- 2-1) UserController 클래스 작성 #1

src/main/java/myspringboot/user/UserController.java

```
package myspringboot.user;

@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping(value = "/users")
public class UserController {
    private List<User> users = new ArrayList<>();

    UserController() {
        buildUsers();
    }
    void buildUsers() {
        User user1 = new User(1L, "John", "Doe", "john@email.com");
        User user2 = new User(2L, "Jon", "Smith", "smith@email.com");
        User user3 = new User(3L, "Will", "Craig", "will@email.com");
        User user4 = new User(4L, "Sam", "Lernorad", "sam@email.com");
        User user5 = new User(5L, "Ross", "Doe", "ross@email.com");
        users.add(user1);
        users.add(user2);
        users.add(user3);
        users.add(user4);
        users.add(user5);
    }
}
```

Spring Boot REST Service : Controller 클래스 작성

- 2-2) UserController 클래스 작성 #2

src/main/java/myspringboot/user/UserController.java

```
@RequestMapping(method = RequestMethod.GET)
public List<User> getUsers() {
    return this.users;
}

@RequestMapping(value =("/{id}", method = RequestMethod.GET)
public User getUser(@PathVariable("id") Long id) {
    return this.users.stream().filter(user -> user.getId() == id).findFirst().orElse(null);
}

@RequestMapping(method = RequestMethod.POST)
public User saveUser(@RequestBody User user) {
    Long nextId = 0L;
    if (this.users.size() != 0) {
        User lastUser = this.users.stream()
            .skip(this.users.size() - 1).findFirst().orElse(null);
        nextId = lastUser.getId() + 1;
    }
    user.setId(nextId);
    this.users.add(user);
    return user;
}
```

Spring Boot REST Service : Controller 클래스 작성

- 2-3) UserController 클래스 작성 #3

src/main/java/myspringboot/user/UserController.java

```
@RequestMapping(method = RequestMethod.PUT)
public User updateUser(@RequestBody User user) {
    User modifiedUser = this.users.stream()
        .filter(u -> u.getId() == user.getId()).findFirst().orElse(null);
    modifiedUser.setFirstName(user.getFirstName());
    modifiedUser.setLastName(user.getLastName());
    modifiedUser.setEmail(user.getEmail());
    return modifiedUser;
}

@RequestMapping(value =("/{id}", method = RequestMethod.DELETE)
public boolean deleteUser(@PathVariable Long id) {
    User deleteUser = this.users.stream()
        .filter(user -> user.getId() == id).findFirst().orElse(null);
    if (deleteUser != null) {
        this.users.remove(deleteUser);
        return true;
    } else {
        return false;
    }
}
}
```

Angular Form

Template Driven Forms

Angular Form : 2가지 Form-building 방식

- Angular의 두가지 Form-building 방식

- 1) Template-driven forms 와 Reactive forms가 있다.

- 2) 두 방식 모두 @angular/forms 라이브러리에 속한다.

- 3) Template-driven forms : FormsModule 사용

- Reactive forms : ReactiveFormsModule 사용

- 4) Template-driven forms의 특징

- Template-driven forms 방식은 form control object를 사용자가 만들지 않고, Angular가 form control object를 만들어 준다. 즉, ngModel를 제공 하여 Angular가 핸들링 한다.

- 이벤트가 발생하면 Angular는 mutable 데이터 모델을 업데이트 합니다.

Angular Form : 2가지 Form-building 방식

- Angular의 두가지 Form-building 방식

- 5) Reactive forms의 특징

- Reactive forms는 컴포넌트 클래스에 form control object를 만들고, 이를 컴포넌트 템플릿의 form control 엘리먼트에 바인딩 할 수 있다.
 - 컴포넌트 클래스는 데이터 모델과 form control 구조에 즉시 액세스 할 수 있으므로 데이터 모델 값을 form control로 보내고, 사용자가 변경 한 값을 다시 가져올 수 있다.
 - Reactive forms의 장점은 값 및 유효성 업데이트가 항상 동기적 이고 사용자가 제어 할 수 있다는 점입니다.

Angular Form : Template Driven Forms

- Template-driven Forms

- 1) ngModel directive를 사용하여 Form을 작성할 수 있습니다.
- 2) 이 Form의 세가지 필드 중 두 개가 필수입니다. 녹색 막대는 필수 입력란입니다.
- 3) Hero Name 값을 입력하지 않으면 스타일의 유효성 검사 오류가 표시됩니다.

Hero Form

Name

Dr IQ

Alter Ego

Chuck Overstreet

Hero Power

Really Smart

Submit

Hero Form

Name

Name is required

Alter Ego

Chuck Overstreet

Hero Power

Really Smart

Submit

Angular Form : Template Driven Forms

- **Template-driven Forms 작성 절차**

- 1) Hero Model 클래스를 만듭니다.
- 2) Form을 제어하는 구성 요소를 만듭니다.
- 3) 초기 Form 레이아웃으로 템플릿을 만듭니다.
- 4) ngModel Two-Way 데이터 바인딩 구문을 사용하여 각 Form 컨트롤에 데이터 속성을 바인딩합니다.
- 5) 각 form 입력 컨트롤에 name 속성을 추가 하십시오.
- 6) 사용자 정의 CSS를 추가하여 시각적 피드백을 제공합니다.
- 7) 유효성 검사 오류 메시지를 표시하거나 숨깁니다.
- 8) ngSubmit으로 form 제출을 처리한다.
- 9) Form이 유효 할 때까지는 Form의 Submit 버튼을 disable 시킨다.

Angular Form : (1) angular-form 프로젝트와 Model 클래스

(1)-1. 새로운 프로젝트 생성 : angular-forms

```
ng new angular-forms
```

(1)-2. Hero Model 클래스 생성

Hero 클래스는 세 가지 필수 필드 (id, name, power)와 하나의 선택적 필드 (alterEgo)가 있다.

```
ng generate class Hero
```

src/app/hero-search/hero-search.component.ts

```
export class Hero {  
  constructor(  
    public id: number,  
    public name: string,  
    public power: string,  
    public alterEgo?: string  
  )  
}
```

Angular Form : (2) Form Component 생성

(2)-1. 새로운 컴포넌트 생성 : HeroForm

ng generate component HeroForm

src/app/hero-form/hero-form.component.ts (v1)

```
import { Component } from '@angular/core';
import { Hero } from '../hero';
@Component({
  selector: 'app-hero-form',
  templateUrl: './hero-form.component.html',
  styleUrls: ['./hero-form.component.css']
})
export class HeroFormComponent {
  powers = ['Really Smart', 'Super Flexible', 'Super Hot', 'Weather Changer'];
  model = new Hero(18, 'Dr IQ', this.powers[0], 'Chuck Overstreet');
  submitted = false;
  onSubmit() { this.submitted = true; }
  // TODO: Remove this when we're done
  get diagnostic() { return JSON.stringify(this.model); }
}
```

Angular Form : (3) app.module.ts 수정

(3)-1. app.module.ts 수정

src/app/app.module.ts (v1)

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { HeroFormComponent } from './hero-form/hero-form.component';

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

Angular Form : (4) 템플릿 수정, style 수정

(4)-1. app.component.html 수정

src/app/app.component.html

```
<app-hero-form> </app-hero-form>
```

(4)-2. initial HTML form template : hero-form.component.html 수정, style.css 수정

- ① hero 필드 인 name과 alterEgo를 두 개 제시합니다.
- ② Name <input> 컨트롤에는 HTML5 required 속성이 있습니다.
alterEgo는 선택 사항이므로 Alter Ego <input> 컨트롤은 사용하지 않습니다.
- ③ container, form-group, form-control 및 btn css 클래스는 Twitter Bootstrap에서 가져옵니다.

src/style.css

```
@import url('https://unpkg.com/bootstrap@3.3.7/dist/css/bootstrap.min.css');
```

Angular Form : (4) 템플릿 수정, style 수정

(4)-3. Initial HTML form template : hero-form.component.html 수정

src/app/hero-form/hero-form.component.html

```
<div class="container">
  <h1>Hero Form</h1>
  <form>
    <div class="form-group">
      <label for="name">Name</label>
      <input type="text" class="form-control" id="name" required>
    </div>
    <div class="form-group">
      <label for="alterEgo">Alter Ego</label>
      <input type="text" class="form-control" id="alterEgo">
    </div>
    <button type="submit" class="btn btn-success">Submit</button>
  </form>
</div>
```

Angular Form : (5) *ngFor 구문 사용

(5)-1. Add powers with *ngFor : hero-form.component.html 수정

- ① 폼에 select를 추가하고 ngFor를 사용하여 option을 power list에 바인딩합니다.
- ② pow 템플릿 입력 변수는 각 iteration에서 각 다른 power를 나타내며, interpolation 구문({{}})을 사용하여 출력합니다.

src/app/hero-form/hero-form.component.html

```
<div class="form-group">
  <label for="power">Hero Power</label>
  <select class="form-control" id="power" required>
    <option *ngFor="let pow of powers" [value]="pow">{{pow}}</option>
  </select>
</div>
```

Angular Form : (6) Two-way data binding과 NgForm

(6)-1. two-way data binding, ngform : hero-form.component.html 수정

- ① 사용자 입력은 이벤트 바인딩을 사용하여 DOM 이벤트를 수신하는 방법과 표시된 값으로 구성 요소 속성을 업데이트하는 방법을 보여줍니다.
- ② [(ngModel)]을 Form과 함께 사용하면 name 속성을 정의해야 합니다.
- ③ NgForm Directive는 ngModel directive와 name 속성을 사용하여 만든 엘리먼트들의 유효성을 포함하여 속성을 모니터링합니다.

src/app/hero-form/hero-form.component.html

```
<form #heroForm="ngForm">
  <div class="form-group">
    <label for="name">Name</label>
    <input type="text" class="form-control" id="name"
      required
      [(ngModel)]="model.name" name="name">
  </div>
```

Angular Form : (6) Two-way data binding과 NgForm

(6)-2. two-way data binding : hero-form.component.html 수정

- ① Angular는 FormControl 인스턴스를 만들고, <form>의 NgForm Directive를 등록합니다.
- ② 각 FormControl은 name 속성에 지정한 name으로 등록됩니다.

src/app/hero-form/hero-form.component.html

```
<div class="form-group">
  <label for="alterEgo">Alter Ego</label>
  <input type="text" class="form-control" id="alterEgo"
    [(ngModel)]="model.alterEgo" name="alterEgo">
</div>

<div class="form-group">
  <label for="power">Hero Power</label>
  <select class="form-control" id="power" required
    [(ngModel)]="model.power" name="power">
    <option *ngFor="let pow of powers" [value]="pow">{{pow}}</option>
  </select>
</div>
</form>
```


Angular Form : (7) ngModel로 상태 제어 및 유효성 추적

(7)-1. ngModel을 사용하여 state(상태) 제어 및 validity(유효성) 추적

- ① Form에서 ngModel을 사용하면 사용자가 컨트롤을 터치했는지, 값이 변경되었는지, 값이 유효하지 않은지 여부를 알려줍니다.
- ② NgModel Directive 상태를 반영하는 특별한 Angular CSS 클래스로 컨트롤을 업데이트합니다. 이러한 클래스 이름을 활용하여 컨트롤의 모양을 변경할 수 있습니다.

State	Class if true	Class if false
The control has been visited.	ng-touched	ng-untouched
The control's value has changed.	ng-dirty	ng-pristine
The control's value is valid.	ng-valid	ng-invalid

Angular Form : (7) ngModel로 상태 제어 및 유효성 추적

(7)-2. ng-valid와 ng-invalid 에 대한 custom CSS 추가하기

- ① ng-valid / ng-invalid 에 대한 CSS 클래스 추가
- ② 입력 왼쪽에 있는 색상 막대를 사용하여 필수 입력란과 잘못된 데이터를 동시에 표시한다.

<input type="text" value="Dr IQ"/>	Valid + Required
<input type="text" value="Chuck Overstreet"/>	Valid + Optional
<input type="text"/>	Invalid (required optional)

src/assets/form.css

```
.ng-valid[required], .ng-valid.required {  
  border-left: 5px solid #42A948; /* green */  
}  
.ng-invalid:not(form) {  
  border-left: 5px solid #a94442; /* red */  
}
```

src/index.html(styles)

```
<link rel="stylesheet" href="assets/forms.css">
```

Angular Form : (7) ngModel로 상태 제어 및 유효성 추적

(7)-3. 유효성 검사 오류 메시지 표시 및 숨기기

- ① name input box의 값이 없으면, 막대가 빨간색으로 바뀌고 유용한 메시지를 출력해야 한다.
- ② <div> 엘리먼트의 hidden 속성을 binding하여 name 오류 메시지의 가시성을 제어합니다.
- ③ "pristine"은 사용자가 이 양식으로 표시된 이후 값을 변경하지 않았 음을 의미합니다.

Name

Name is required

src/app/hero-form/hero-form.component.html

```
<label for="name">Name</label>
<input type="text" class="form-control" id="name"
      required
      [(ngModel)]="model.name" name="name" #name="ngModel">
<div [hidden]="name.valid || name.pristine"
      class="alert alert-danger">
  Name is required
</div>
```

Angular Form : (7) ngModel로 상태 제어 및 유효성 추적

(7)-4. New Hero 버튼 추가

- ① Form 아래쪽에 새로운 Hero 버튼을 놓고 click 이벤트를 newHero 컴포넌트 메서드에 바인딩 한다.
- ② newHero() 메서드를 호출 한 후, Form의 reset() 메서드를 호출하여 Form 컨트롤의 초기 상태가 되도록 합니다.

src/app/hero-form/hero-form.component.html (reset the form)

```
<button type="button" class="btn btn-default" (click)="newHero(); heroForm.reset()">New  
Hero</button>
```

src/app/hero-form/hero-form.component.ts

```
newHero() {  
  this.model = new Hero(42, "", "");  
}
```

Angular Form : (8) ngSubmit으로 Form submit 하기

(8)-1. Form submit 처리하기

- ① Form의 ngSubmit 이벤트 속성을 Hero Form 컴포넌트의 onSubmit() 메서드에 바인딩합니다.

```
src/app/hero-form/hero-form.component.html (ngSubmit)
```

```
<form (ngSubmit)="onSubmit()" #heroForm="ngForm">
```

- ② heroForm 변수를 통해 Form의 전체 유효성을 이벤트 바인딩을 사용하여 button의 disabled 속성에 바인딩합니다.

- ③ name input box의 값이 없으면 오류 메시지에 명시된 "required"규칙을 위반하게 되고, submit button도 사용 불가능합니다.

```
src/app/hero-form/hero-form.component.html (ngSubmit)
```

```
<button type="submit" class="btn btn-success" [disabled]="!heroForm.form.valid">Submit</button>
```

Angular Form : (9) Toggle two form regions

(9)-1. form 영역 토글링 하기

- ① 현저한 시각 효과를 위해 데이터 입력 영역을 숨기고 다른 것을 표시합니다.
- ② <div>로 form을 감싸고 hidden 속성을 HeroFormComponent.submitted 속성에 바인딩합니다.
- ③ submit button을 클릭하면 submit 된 flag 변수가 true가되고 form이 사라지게 됩니다.

src/app/hero-form/hero-form.component.html (ngSubmit)

```
<div [hidden]="submitted">
  <h1>Hero Form</h1>
  <form (ngSubmit)="onSubmit()" #heroForm="ngForm">

    <!-- ... all of the form ... -->
  </form>
</div>
```

src/app/hero-form/hero-form.component.ts

```
submitted = false;

onSubmit() { this.submitted = true; }
```

Angular Form : (9) Toggle two form regions

(9)-2. form 영역 토글링 하기

- ① interpolation({{ }}) 바인딩으로 read-only로 표시되고, 아래의 <div>는 컴포넌트가 submit 된 상태인 동안에 만 나타납니다.

src/app/hero-form/hero-form.component.html

```
<div [hidden]="!submitted">
  <h2>You submitted the following:</h2>
  <div class="row">
    <div class="col-xs-3">Name</div>
    <div class="col-xs-9 pull-left">{{ model.name }}</div>
  </div>
  <div class="row">
    <div class="col-xs-3">Alter Ego</div>
    <div class="col-xs-9 pull-left">{{ model.alterEgo }}</div>
  </div>
```

Angular Form : (9) Toggle two form regions

(9)-3. form 영역 토글링 하기

① edit button을 클릭하면 이 <div> 블록이 사라지고 편집 가능한 form이 다시 나타납니다.

src/app/hero-form/hero-form.component.html (ngSubmit)

```
<div class="row">
  <div class="col-xs-3">Power</div>
  <div class="col-xs-9 pull-left">{{ model.power }}</div>
</div>
<br>
<button class="btn btn-primary" (click)="submitted=false">Edit</button>
</div>
```


Angular Form 작성 Summary

Template Driven Forms 작성하면서 배운 사항들

- ① Angular HTML form template.
- ② @Component decorator를 사용한 form 컴포넌트 클래스
- ③ NgForm.ngSubmit 이벤트 속성에 바인딩 한 Form submission 처리.
- ④ two-way 데이터 바인딩을 위한 [(ngModel)] 구문
- ⑤ validation(유효성 검사) 및 form-element 변경 추적을 위해 name 속성 사용.
- ⑥ NgForm 유효성(validity)에 바인딩하여 submit button의 사용 가능 상태 제어.

Angular Form : [hero-form/hero-form.component.ts](#)

src/app/hero-form/hero-form.component.ts

```
import { Component } from '@angular/core';
import { Hero } from '../hero';
@Component({
  selector: 'app-hero-form',
  templateUrl: './hero-form.component.html',
  styleUrls: ['./hero-form.component.css']
})
export class HeroFormComponent {
  powers = ['Really Smart', 'Super Flexible',
    'Super Hot', 'Weather Changer'];

  model = new Hero(18, 'Dr IQ', this.powers[0], 'Chuck Overstreet');

  submitted = false;

  onSubmit() { this.submitted = true; }

  newHero() {
    this.model = new Hero(42, '', '');
  }
}
```

Angular Form : [hero-form/hero-form.component.html](#) – (1)

src/app/hero-form/hero-form.component.html

```
<div class="container">
  <div [hidden]="submitted">
    <h1>Hero Form</h1>
    <form (ngSubmit)="onSubmit()" #heroForm="ngForm">
      <div class="form-group">
        <label for="name">Name</label>
        <input type="text" class="form-control" id="name"
          required
          [(ngModel)]="model.name" name="name"
          #name="ngModel">
        <div [hidden]="name.valid || name.pristine"
          class="alert alert-danger">
          Name is required
        </div>
      </div>
    </div>

    <div class="form-group">
      <label for="alterEgo">Alter Ego</label>
      <input type="text" class="form-control" id="alterEgo"
        [(ngModel)]="model.alterEgo" name="alterEgo">
    </div>
```

Angular Form : [hero-form/hero-form.component.html](#) – (2)

src/app/hero-form/hero-form.component.html

```
<div class="form-group">
  <label for="power">Hero Power</label>
  <select class="form-control" id="power"
    required
    [(ngModel)]="model.power" name="power"
    #power="ngModel">
    <option *ngFor="let pow of powers" [value]="pow">{{pow}}</option>
  </select>
  <div [hidden]="power.valid || power.pristine" class="alert alert-danger">
    Power is required
  </div>
</div>

<button type="submit" class="btn btn-success" [disabled]="!heroForm.form.valid">Submit</button>
<button type="button" class="btn btn-default" (click)="newHero(); heroForm.reset()">New Hero</button>
</form>
</div>
```

Angular Form : hero-form/hero-form.component.html – (3)

src/app/hero-form/hero-form.component.html

```
<div [hidden]="!submitted">
  <h2>You submitted the following:</h2>
  <div class="row">
    <div class="col-xs-3">Name</div>
    <div class="col-xs-9 pull-left">{{ model.name }}</div>
  </div>
  <div class="row">
    <div class="col-xs-3">Alter Ego</div>
    <div class="col-xs-9 pull-left">{{ model.alterEgo }}</div>
  </div>
  <div class="row">
    <div class="col-xs-3">Power</div>
    <div class="col-xs-9 pull-left">{{ model.power }}</div>
  </div>
  <br>
  <button class="btn btn-primary" (click)="submitted=false">Edit</button>
</div>
</div>
```

3. Angular Form ■ hero.ts / app.component.html /app.component.ts

src/app/hero.ts

```
export class Hero {  
  constructor(  
    public id: number,  
    public name: string,  
    public power: string,  
    public alterEgo?: string  
  ) { }  
}
```

src/app/app.component.html

```
<app-hero-form> </app-hero-form>
```

src/app/app.component.ts

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

Angular Form : `app.module.ts`

`src/app/app.module.ts`

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

import { AppComponent } from './app.component';
import { HeroFormComponent } from './hero-form/hero-form.component';

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

Angular Form ■ [styles.css](#) / [angular-cli.json](#) / [forms.css](#) / [index.html](#)

src/styles.css

```
@import url('https://unpkg.com/bootstrap@3.3.7/dist/css/bootstrap.min.css');
```

angular-cli.json

```
"styles": [  
  "styles.css"  
],
```

src/assets/forms.css

```
.ng-valid[required], .ng-valid.required {  
  border-left: 5px solid #42A948; /* green */  
}  
.ng-invalid:not(form) {  
  border-left: 5px solid #a94442; /* red */  
}
```

src/index.html

```
<link rel="stylesheet" href="assets/forms.css">
```


Angular Form

Reactive Forms

Angular Form : Reactive Forms

- Angular의 두가지 Form-building 방식

- 1) Reactive forms의 특징

- Reactive forms는 컴포넌트 클래스에 form control object를 만들고, 이를 컴포넌트 템플릿의 form control 엘리먼트에 바인딩 할 수 있다.
 - 컴포넌트 클래스는 데이터 모델과 form control 구조에 즉시 액세스 할 수 있으므로 데이터 모델 값을 form control로 보내고 사용자가 변경 한 값을 다시 가져올 수 있다.
 - 컴포넌트는 Form control state의 변경 사항을 관찰하고 변경사항에 대응할 수
 - Reactive forms의 장점은 값 및 유효성 업데이트가 항상 동기적 이고 사용자가 제어 할 수 있다는 점입니다.

Reactive Forms App 작성 : Data Model 작성

- 새로운 Angular Application Project 생성

```
ng new angular-reactive-forms
```

- Data Model 생성

```
ng generate class data-model
```

src/app/data-model.ts

```
export class Hero {  
  id = 0;  
  name = "";  
  addresses: Address[];  
}  
export class Address {  
  street = "";  
  city = "";  
  state = "";  
  zip = "";  
}
```

Reactive Forms App 작성 : Data Model 작성

src/app/data-model.ts

```
export const heroes: Hero[] = [
  { id: 1,
    name: 'Whirlwind',
    addresses: [
      {street: '123 Main', city: 'Anywhere', state: 'CA', zip: '94801'},
      {street: '456 Maple', city: 'Somewhere', state: 'VA', zip: '23226'},
    ] },
  { id: 2,
    name: 'Bombastic',
    addresses: [
      {street: '789 Elm', city: 'Smallville', state: 'OH', zip: '04501'},
    ] },
  { id: 3,
    name: 'Magneta',
    addresses: [ ]
  },
];

export const states = ['CA', 'MD', 'OH', 'VA'];
```

Reactive Forms App 작성 : reactive form 컴포넌트

- 새로운 reactive forms 컴포넌트 생성

```
ng generate component HeroDetail
```

(1) HelloDetailComponent 클래스 수정

- FormControl은 FormControl 인스턴스를 직접 만들고 관리 할 수 있는 Directive이다.
- name변수는 템플릿의 HTML input box의 hero name에 바인딩 됩니다.

```
src/app/hero-detail/hero-detail.component.ts
```

```
import { FormControl } from '@angular/forms';

export class HeroDetailComponent1 {
  name = new FormControl();
}
```

Reactive Forms App 작성 : reactive form 템플릿

(2) HelloDetailComponent 템플릿 수정

- 클래스의 FormControl name에 연결되는 입력 임을 알려려면 <input>의 템플릿에 [formControl] = "name"이 필요합니다.

src/app/hero-detail/hero-detail.component.html

```
<h2>Hero Detail</h2>
<h3><i>Just a FormControl</i></h3>
<label class="center-block">Name:
  <input class="form-control" [formControl]="name">
</label>
```

(3) AppComponent 템플릿 수정

src/app/app.component.html

```
<div class="container">
  <h1>Reactive Forms</h1>
  <app-hero-detail> </app-hero-detail>
</div>
```

Reactive Forms App 작성 : **ReactiveFormsModule**

(4) AppModule 클래스 수정

- ReactiveFormsModule를 import 합니다.
- AppModule's imports list에 ReactiveFormsModule를 추가한다.

src/app/app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule }  from '@angular/platform-browser';
import { ReactiveFormsModule } from '@angular/forms'; // <-- #1 import module
import { AppComponent }   from './app.component';
import { HeroDetailComponent } from './hero-detail/hero-detail.component'; // #1 import component
@NgModule({
  imports: [ BrowserModule,
    ReactiveFormsModule // #2 add to @NgModule imports
  ],
  declarations: [ AppComponent, HeroDetailComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Reactive Forms App 작성 : reactive form API

- **Essential form classes**

- 1) AbstractControl은 FormControl, FormGroup 및 FormArray의 세 가지 concrete form control의 abstract base 클래스입니다. common behaviors와 property를 제공합니다.
- 2) FormControl은 개별 Form Control의 값과 유효성 상태를 추적합니다. <input> 또는 <select>와 같은 HTML 양식 컨트롤에 해당합니다.
- 3) FormGroup은 AbstractControl 인스턴스 그룹의 값과 유효성 상태를 추적합니다. 그룹의 속성에는 하위 컨트롤이 포함됩니다. FormGroup은 top-level form 컴포넌트이다.
- 4) FormArray는 숫자로 인덱싱 된 AbstractControl 인스턴스 배열의 값과 유효성 상태를 추적합니다.

Reactive Forms App 작성 : style the app

(5) Style 설정

AppComponent와 HeroDetailComponent의 템플릿 HTML에서 부트 스트랩 CSS 클래스를 사용하므로 styles.css 헤드에 bootstrap CSS 스타일 시트를 추가함.

```
src/styles.css
```

```
@import url('https://unpkg.com/bootstrap@3.3.7/dist/css/bootstrap.min.css');
```

Reactive Forms App 작성 : FormGroup 추가

(6) 컴포넌트에 FormGroup 추가하기

여러 개의 FormControl를 만들려고 하면 FormGroup을 FormControl의 Parent로 해야 한다.

src/app/hero-detail/hero-detail.component.ts

```
export class HeroDetailComponent2 {  
  heroForm = new FormGroup ({  
    name: new FormControl()  
  });  
}
```

src/app/hero-detail/hero-detail.component.html

```
<h2>Hero Detail</h2>  
<h3><i>FormControl in a FormGroup</i></h3>  
<form [formGroup]="heroForm">  
  <div class="form-group">  
    <label class="center-block">Name:  
      <input class="form-control" formControlName="name">  
    </label>  
  </div>  
</form>
```

Reactive Forms App 작성 : FormGroup 추가

(7) Form model에 입력한 값 확인하기

heroForm.value를 JsonPipe를 통해 piping하면 Model이 브라우저에서 JSON으로 렌더링 됩니다.

src/app/hero-detail/hero-detail.component.html

```
<form [formGroup]="heroForm">
  <div class="form-group">
    <label class="center-block">Name:
      <input class="form-control" formControlName="name">
    </label>
  </div>
</form>
<p>Form value: {{ heroForm.value | json }}</p>
```

Hero Detail

FormControl in a FormGroup

Name:

Form value: { "name": null }

Reactive Forms App 작성 : FormBuilder 추가

(8) 컴포넌트에 FormBuilder 추가하기

- Constructor에서 FormBuilder를 주입(injection) 받는다.
- Constructor에서 FormBuilder의 createForm() 메서드를 호출한다.

src/app/hero-detail/hero-detail.component.ts

```
export class HeroDetailComponent3 {
  heroForm: FormGroup; // <--- heroForm is of type FormGroup

  constructor(private fb: FormBuilder) { // <--- inject FormBuilder
    this.createForm();
  }

  createForm() {
    this.heroForm = this.fb.group({
      name: '', // <--- the FormControl called "name"
    });
  }
}
```

Reactive Forms App 작성 : FormBuilder 추가

(9) 컴포넌트에 Validators.required 사용하기

- FormGroup의 name 속성을 배열로 변경합니다. 첫 번째 항목은 name의 초기 값입니다.
- 두 번째는 필요한 유효성 검사기 Validators.required입니다.

src/app/hero-detail/hero-detail.component.ts

```
export class HeroDetailComponent3 {
  heroForm: FormGroup; // <--- heroForm is of type FormGroup

  constructor(private fb: FormBuilder) { // <--- inject FormBuilder
    this.createForm();
  }

  createForm() {
    this.heroForm = this.fb.group({
      name: ['', Validators.required ],
    });
  }
}
```