

1. 앵귤러(Angular) 시작하기

sang-gyeong · 2021 년 8 월 1 일

0

1. Angular 소개

Angular 의 장점

2. Angular CLI

지원기능

기본 사용 방법

프로젝트 구성요소 생성

3. Angular 의 파일 구조와 처리 흐름

3-1. Angular 애플리케이션의 파일 구조

3-2. Angular 애플리케이션의 처리 흐름

index.html

main.ts

app.module.ts

app.component.ts

Angular 학습내용 정리

목록 보기

1/4



1. Angular 소개

Angular 는 SPA 개발을 위한 구글의 오픈소스 프레임워크입니다.

정적 타입을 제공하는 TypeScript 를 주력 언어로 채택하여,
대규모 애플리케이션 개발에 보다 적합한 환경을 제공합니다.

Angular 의 장점

1) 컴포넌트 기반 개발

컴포넌트 기반 개발(CBD)은 개발 생산성을 높이며 대규모 애플리케이션에 적합한 구조입니다.

2) TypeScript 채택

Angular가 기본언어로 채택하고 있는 TypeScript의 명시적 타입 지정은 코드의 가독성을 높이고 예측할 수 있게 하며, 컴파일 단계에서 오류를 포착할 수 있는 장점이 있습니다.

3) 개발도구의 통합 및 개발 환경 구축 자동화

Angular CLI를 통해 간편한 개발 환경 구축을 지원합니다.

프레임워크를 도입할 때 장벽으로 손꼽히는 것 중 하나가 개발 환경 구축임을 미루어보면 이는 큰 장점입니다.

간단한 명령어를 사용하여 프로젝트 생성에서 빌드, 테스트, 구성요소 추가 등을 간편하게 할 수 있으며, 개발용 서버를 내장하고 있어 실행 또한 가능합니다.

4) 성능의 향상

• AoT 컴파일

AoT 컴파일(Ahead of Time compilation)은 사전 컴파일 방식을 의미합니다.

예를 들어, ngIf, ngFor와 같은 구조 디렉티브를 브라우저가 실행 가능한 코드로 변환해야 하는데 이러한 과정을 런타임에서 실시하지 않고, 사전에 컴파일하여 실행 속도를 향상시키는 기법입니다.

• Lazy Loading

SPA의 단점을 극복하기 위한 대안으로,

애플리케이션 실행시점에 모든 모듈을 한꺼번에 로딩하지 않고 필요한 시점에 필요한 모듈만을 로딩하는 방식입니다. 현재 페이지에서 불필요한 모듈까지 로딩하는 낭비를 방지함으로써 페이지 로딩 속도를 높입니다.

- 코드 최적화

'Mobile First'를 지향하는 고성능 프레임워크를 표방하는 만큼

Angular 코드는 지속적으로 최적화되고 있습니다.

2. Angular CLI

Angular CLI는 Angular 프로젝트 스캐폴딩을 생성, 실행, 빌드할 수 있으며,

Angular의 다양한 구성요소를 선별적으로 추가할 수 있는 강력한 커맨드-라인 인터페이스입니다.

개발용 서버를 내장하고 있어서 간단히 프로젝트를 실행시켜서 동작을 확인할 수 있습니다.

지원기능

- Angular 프로젝트 생성
- Angular 프로젝트에 컴포넌트, 디렉티브, 서비스, 파이프, 클래스, 인터페이스 등의 구성요소 추가
- LiveReload를 지원하는 내장 개발 서버를 이용한 Angular 프로젝트 실행
- Unit/E2E 테스트 환경 지원

- 배포를 위한 Angular 프로젝트 패키징

기본 사용 방법

npm 패키지 매니저를 사용하여 간단하게 Angular CLI 를 설치할 수 있습니다.

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

@angular/cli 는 퍼스트 파티 패키지로, SPA 개발에 필요한 모든 도구를 포함하고 있습니다.

컴포넌트, 서비스, 모듈 등 모든 파일들을 명령어로 쉽게 추가할 수 있고 기존 코드를 읽어 import 문도 작성해줍니다.

새로운 앵귤러 프로젝트를 생성하고, 빌드한 뒤에 실행하려면 다음과 같이 명령어를 실행하면 됩니다.

```
$ ng new <프로젝트 이름>  
# 프로젝트 실행  
$ cd <프로젝트 이름>  
$ ng serve (--open, --port ..)
```

ng serve 는 로컬 개발환경에서 애플리케이션을 빌드한 후에 서버를 실행하는 명령어입니다.

소스 코드가 변경되면 자동으로 재빌드하고 환경을 갱신합니다.

앵귤러에서 기본 포트는 **4200** 번 입니다.
브라우저에서 <http://localhost:4200> 으로 접근하면
애플리케이션이 실행되는 것을 확인할 수 있습니다.

프로젝트 구성요소 생성

```
$ ng generate (ng g)
```

컴포넌트, 디렉티브, 서비스, 파이프, 모듈, 가드, 클래스,
인터페이스, Enum 등을 생성할 수 있습니다.

3. Angular 의 파일 구조와 처리 흐름

3-1. Angular 애플리케이션의 파일 구조

1) src 폴더

Angular 프로젝트는 컴포넌트, 디렉티브, 서비스, 모듈
등 Angular 구성 요소와 각종 설정 파일로 구성됩니다.
src 폴더는 이러한 Angular 의 모든 구성요소, 공통 CSS,
이미지나 폰트와 같은 정적파일, 설정 파일 등
애플리케이션 필수 파일을 담고 있습니다.

src/app 폴더에는 Angular 구성요소가 위치하며, 개발자가
작성하는 대부분의 파일은 이곳에 포함됩니다.

```
- app/app.components. (ts, html, css, spec.ts) :
```

모든 컴포넌트의 부모 컴포넌트인 루트 컴포넌트를

구성하는 컴포넌트 클래스, 템플릿, CSS 유닛
테스트용 스펙 파일

- app/app.module.ts : Angular 구성요소를 등록하는 루트 모듈
- assets/ : 이미지나 폰트와 같은 정적 파일을 위한 폴더
- environments/ : 프로젝트 빌드 시에 사용될 프로덕션용/개발용 환경설정 파일이 담겨있는 폴더
- browserslist: Autoprefixer, babel 과 같은 프론트엔드 도구 간에 적용 대상 브라우저를 공유하는 browserslist 라이브러리 설정 파일
- favicon.ico: 파비콘 파일

- index.html : 웹 애플리케이션 방문 시 처음으로 로딩되는 디폴트 페이지.

루트 컴포넌트의 선택자인 <app-root>에 의해 루트 컴포넌트의 뷰가 로드되어 브라우저에 표시됩니다.

빌드 시에는 번들링된 자바스크립트 파일이 자동 추가된 index.html 이 /dist 폴더에 생성됩니다

- karma.conf.js : Karma test runner 를 위한 설정 파일로. ng test 명령어 실행 시 참조됩니다

- `main.ts` : 프로젝트의 메인 진입점. 루트 모듈을 사용하여 애플리케이션을 부트스트랩(기동)합니다
- `polyfills.ts`: 크로스 브라우징을 위한 폴리필들을 임포트하는 역할을 합니다. (browser support 참고)
- `styles.css`: 애플리케이션 전역에 적용되는 글로벌 css 파일
- `test.ts` : 유닛 테스트를 위한 메인 진입점
- `tsconfig. (app|spec). json`: TypeScript 컴파일 옵션 설정 파일
- `typings. d. ts` : TypeScript 를 위한 타입 선언 파일

2) 기타 설정 파일

src 폴더 밖의 파일들은 테스트, 빌드, 배포 등을 위한 각종 설정 파일입니다.

- `e2e/`: e2e(end-to-end) 테스트 관련 파일을 위한 폴더. e2e 테스트를 위해 Protractor 가 사용하는 설정 파일인 `protractor.conf. js` 가 담겨있습니다. `ng e2e` 명령어 실행 시 참조됩니다.
- `node_modules/` : `package. json` 에 등록된 의존 모듈이 패키지 매니저에 의해 설치되는 의존 모듈 저장소

- .editorconfig: 코드 에디터 기본 설정 파일
- .gitignore : Git 소스 관리 제외 대상을 위한 설정 파일
- angular.json : Angular CLI 를 위한 설정 파일
- package.json : 의존 모듈 관리를 위해 패키지 매니저가 사용하는 모듈 관리 파일
- README.md : 프로젝트의 개요를 기술한 README 파일. Angular CLI 가 기본적인 내용을 자동 생성합니다
- tsconfig.json : TypeScript 컴파일 옵션 설정 파일
- tslint.json : TSLint 가 사용하는 linting(구문 체크) 설정 파일. ng lint 명령어 실행 시 참조됩니다.

3-2. Angular 애플리케이션의 처리 흐름

index.html

웹 브라우저가 가장 먼저 로딩하는 프로젝트 파일은 /my-app/dist/index.html 입니다.

이것은 ng build 명령어로 프로젝트를 빌드했을 때 /my-app/dist/index.html 에 번들링된 자바스크립트 파일이 추가되어 자동으로 생성되는 파일입니다.

index.html 은 빌드의 결과물로 실제 배포시에는 서버로 이관됩니다.

ng serve 명령어에 의해 내장 개발 서버를 이용하여 로컬 환경에서 프로젝트를 실행(preview)하는 경우, Angular CLI 내부적으로 빌드를 자동 수행하므로 빌드를 별도 실행하여 index.html 를 생성할 필요는 없습니다.

자동으로 빌드되어 가상 드라이브에 저장되어있는 index.html 을 내장 개발 서버가 로드한다고 이해하면 됩니다.

Angular 애플리케이션을 가동하기 위해서는 수많은 의존성 모듈(@angular/*, core-js, zone.js, RxJs 등)과 TypeScript 파일의 컴파일 결과물인 자바스크립트 파일을 로드할 필요가 있습니다.

Angular 는 모듈 번들러 Webpack 을 사용하여 의존성 모듈과 자바스크립트 파일을 번들링 한 후, 수작업 없이 간편하게 로드할 수 있도록 자동화 기능을 제공합니다.

번들링의 결과물로 생성된 자바스크립트 파일들이 로드되어 실행되면서

Angular 애플리케이션은 동작하기 시작합니다. 번들링된 자바스크립트 파일은 아래와 같습니다.

- main.js
- polyfills.js
- styles.js
- vendor.js
- runtime.js

main.ts

main.ts 는 프로젝트의 메인 진입점 입니다.
루트 모듈을 사용하여 애플리케이션을
부트스트랩(기동)합니다.

```
// src/main.ts  
  
...  
platformBrowserDynamic().bootstrapModule(AppModule)  
  .catch(err => console.log(err));
```

main.ts 는 angular.json 의 main 프로퍼티의 설정에 의해
로드됩니다.

```
// angular.json  
{  
  ...  
  "architect": {  
    "build": {  
      "builder": "@angular-  
devkit/build-angular:browser",  
      "options": {  
        "outputPath": "dist/my-app",  
        "index": "src/index.html",  
        "main": "src/main.ts",  
        ...  
      }  
    }  
  }  
}
```

app.module.ts

@NgModule 데코레이터의 인자로 전달되는 메타데이터에 애플리케이션 전체의 설정정보를 기술한 루트 모듈.

루트 모듈은 (/src/app/app.component.ts)를 부트스트랩합니다.

app.component.ts

모든 컴포넌트의 부모 역할을 담당하는 루트 컴포넌트입니다.

my-app 프로젝트의 경우 /dist/index.html 의 app-root 에 의해 루트 컴포넌트의 뷰가 로드되어 app-root 의 콘텐츠로 브라우저에 표시됩니다.

아무생각 없이 "구글"에서 앵글러를 사용한다고 하여
무턱대고 앵글러를 익히기 위해 책을 한권 샀었다.

그냥 단순히 자바스크립트로 이루어진 JQuery 처럼
쓰면 될 줄 알았는데, 웬걸..듣도보도 못한

Typescript?? 컴파일??

책을 한권 사 놓고 바로 환불해야 되는지 엄청 고민
을 했었던 것 같았다.

아무튼 앵귤러는 버전에 따라서 2 가지 스타일로 나
뉘는데, 버전 2.0 이하와 2.0 이상에서의 코딩 방식
이 아주 다르다.

2.0 이하의 1 버전은 소위 말하는 JQuery 처럼 자바스
크립트를 활용한 규칙을 토대로 코딩하면된다.

그런데, 2.0 이상부터는 Typescript 를 활용해서 코딩
을 한 뒤에 Javascript 로 컴파일(?)을 해 주어야 한다.

여기까지만 보더라도 앵귤러가 우리나라에서 유행을 잘 안타는 이유가 접근성이 어렵기 때문이라 할 수 있다. 웬 타입스크립트란 말인가.

아무튼 앵귤러를 설치하여 보자. 앵귤러를 편리하게 개발 하려면 node.js 가 설치되어 있어야한다.

개발도구는 비주얼 스튜디오 코드를 사용하자. (무료다!!!)

설치순서이다.

1. Node.js 설치한다.

2. Node.js 설치 이후 타입스크립트와 앵귤러 빌드 툴을 설치한다.

(DemoAngular 폴더를 만들고 아래와 같이 실행한다.

22/1/21 일 체크)

- cmd 창에서 명령을 입력하자 : **npm install -g**

typescript

- cmd 창에서 명령을 입력하자 : **npm i @angular/cli**

-g

3. 비주얼 스튜디오 코드를 설치한다.

아래 단계에 따라서 천천히 진행하여 보도록하자.

* 주의 : cmd 개발 환경과 자바스크립트의 기본을 전혀 알지 못하면 앵귤러는 사실 접근하기 매우 어렵다.

1. Node.js 설치한다.



다운로드

최신 LTS 버전: **10.15.3** (includes npm 6.4.1)

플랫폼에 맞게 미리 빌드된 Node.js 인스톨러나 소스코드

LTS

대다수 사용자에게 추천



Windows Installer

node-v10.15.3-x86.msi

macOS

node-v10.15.3-darwin.pkg

Windows Installer (.msi)

Windows Binary (.zip)

Node.js 를 다운받는다. 운영체제를 잘 보도록 하자.

Node.js Setup

Welcome to the Node

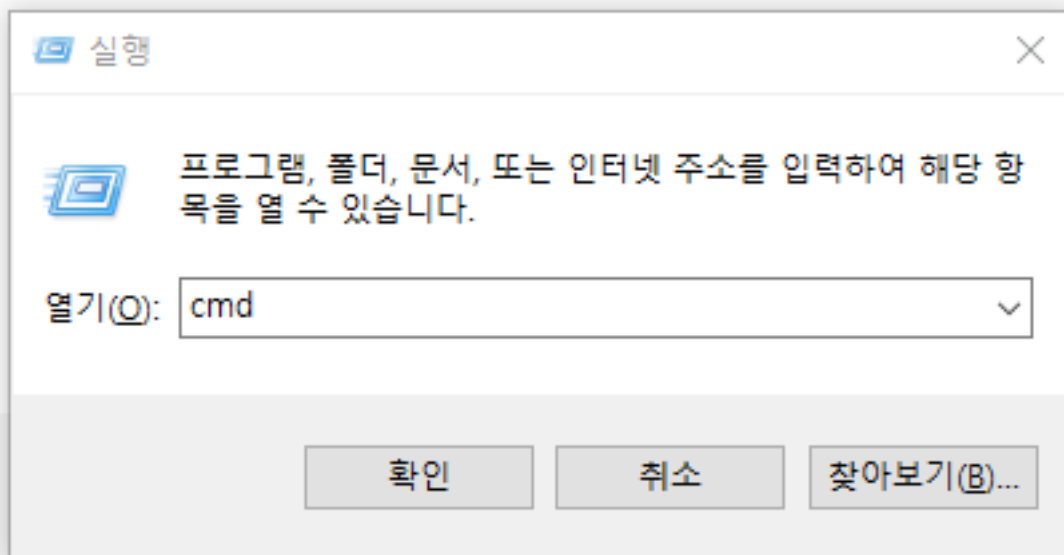
Please wait while the Setup V
through the installation.

Computing space requirement

Back

받은다음에..설치하는 모습이다. Next 버튼만 눌러서
기본세팅으로 해도 상관없다.

2. Node.js 가 설치가 완료되면 cmd 창에서 아래 명령어를 입력한다.cmd 창은 window + R 키를 누른뒤에 cmd 를 입력한다.



window + R 키를 눌러서 CMD 를 입력하면 검은색 화면이 나온다.

```
C:\>npm install -g typescript
C:\Users\sro\AppData\Roaming\npm\tsserver -> C:\Users\sro\AppData\Roaming\npm\tsc -> C:\Users\sro\AppData\Roaming\npm\tsc
+ typescript@3.4.5
updated 1 package in 4.406s

C:\>npm i @angular/cli -g
C:\Users\sro\AppData\Roaming\npm\ng -> C:\Users\sro\AppData\Roaming\npm\ng
npm WARN rollback Rolling back node-pre-gyp@0.12.0 failed (this is not an error)
stat 'C:\Users\sro\AppData\Roaming\npm\node_modules\@angular\cli\node_modules\node-pre-gyp' ENOENT: no such file or directory, stat 'C:\Users\sro\AppData\Roaming\npm\node_modules\@angular\cli\node_modules\node-pre-gyp'
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\@angular\cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: {"current": {"os": "win32", "arch": "x64"}, "wanted": {"os": "darwin", "arch": "x64"}}
+ @angular/cli@7.3.9
added 6 packages from 5 contributors, removed 5 packages, updated 1 package and audited 1 package in 1.411s
C:\>
```

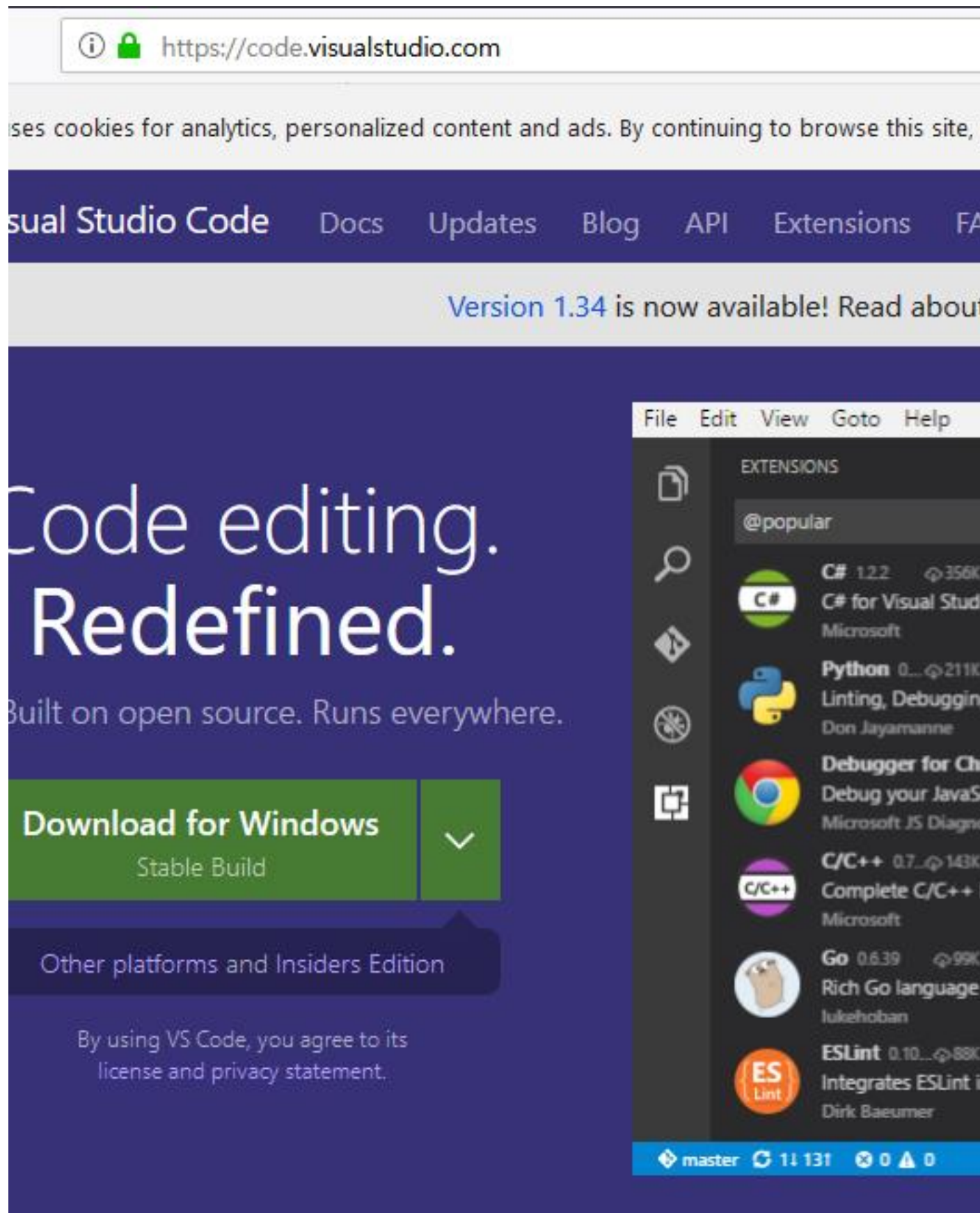
위 2 개의 명령어를 차례대로 입력하자. 지금화면은 이미 설치되어 있어서..업데이트 하는 화면이다..업데이트 무서운데..

* 첫번째 명령어 : npm install -g typescript

* 두번째 명령어 : npm i @angular/cli -g

위 2 개의 명령어로 설치가 되면 개발도구를 설치하여보자.

3. 비주얼 스튜디오 코드를 다운받는다. 설치는 NEXT를 그냥 눌러도 된다.



개발툴인 vs 코드를 받는 곳이다. 마찬가지로 설치는 Next, Next...

개발도구로는 **VS Code(비주얼 스튜디오 코드)**를 사용
하겠다. 해당툴은 한글지원도 되고(한글팩 설치)

타입스크립트 및 앵귤러 언어지원 팩도 설치 할 수
있어서 자동완성 및 디버깅에 좋다.

VS 코드를 설치하고 개발에 도움을 주는 확장팩(모듈)
을 설치하자.

Angular 8 Snippets 를 설치하면 타입스크립트랑 앵귤
러에 관한 지원팩이 설치된다.. (가운데 숫자는 버전)

코드 자동완성, 오류검사 등을 지원하여 주므로 꼭 설치하도록 하자.

4. 설치 완료 후 VS 코드를 실행시킨 뒤 확장팩부터 설치하자.



파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G)



확장: 마켓플레이스



15



korean



검색어 입력



Korean Language Pack for Visual Studio

한국어

Microsoft



Korean-translator 0.4.5

한->영 번역기

dinner



vscode-spellchecker-korean 0.0.1

This extension is a spell checker for ko

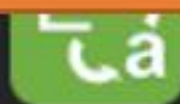
William Kim



Gaji - Gherkin Feature Indentation

Gaji is a formatter for Gherkin(cucumber)

clarekang



translator 1.1.0

translate for Korean

sculove



단디 - 한국어 맞춤법 검사기 0.1.1

fallroot



BLACKPINK 0.0.2



클릭

아랫부분을 클릭후에 korean 이라 입력하면 한글 확장팩을 설치 할 수 있다.



파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G)



확장: 마켓플레이스

검색어



Angular



15



Angular 8 Snippets - TypeScript, HTML

242 Angular Snippets (TypeScript, HTML)

Mikael Morlund



Angular Snippets (Version 8) 8.1.1

Angular version 8 snippets by John Papa

John Papa



Angular Language Service 0.800.0

Editor services for Angular templates

Angular



Angular Schematics 1.23.0

Angular schematics (CLI commands) for

Cyrille Tuzi

클릭



Angular Console 8.0.3

Angular Console for Visual Studio Code

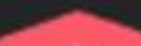
nrwl



Angular Essentials 0.6.3

Essential extensions for Angular development

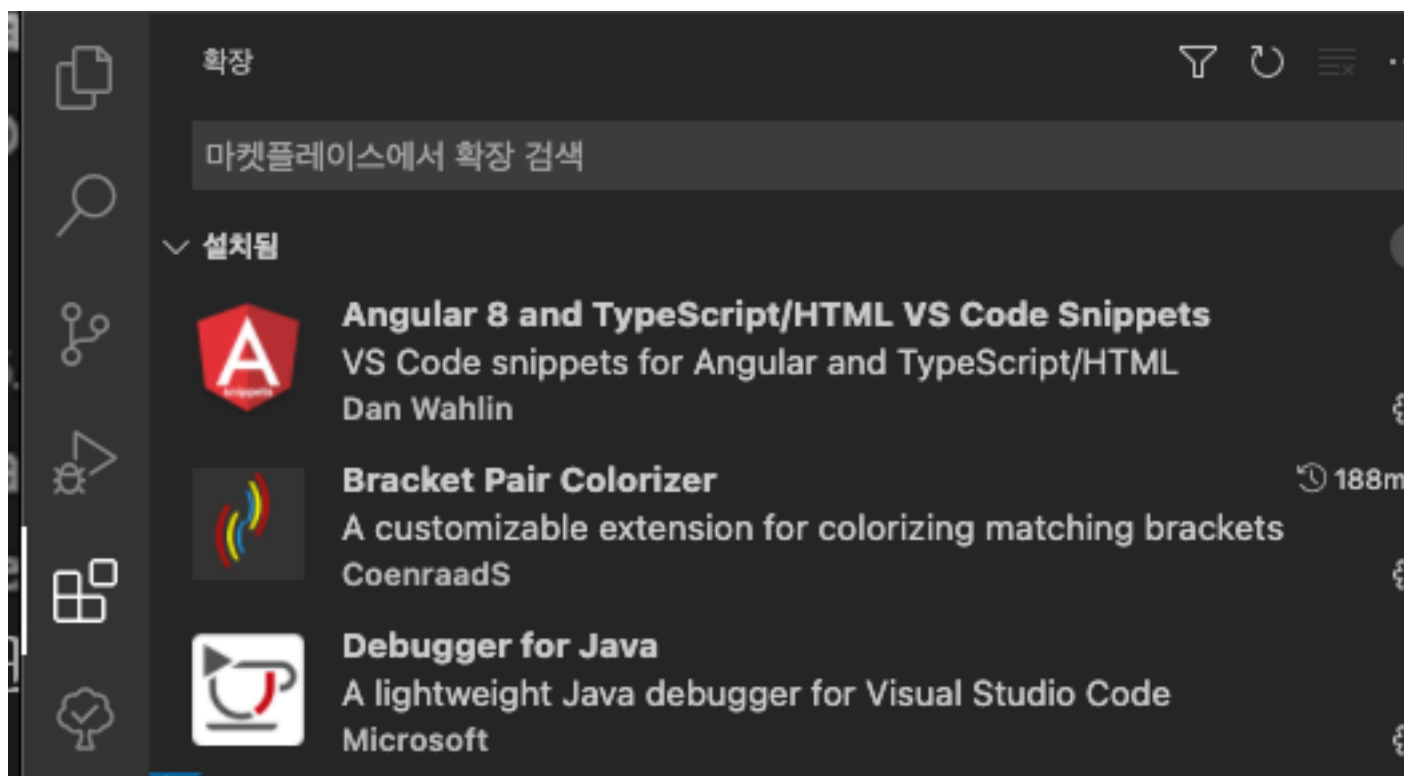
John Papa



Angular Files 1.6.2

Angular 를 입력하고 확장팩을 설치하자. 맨 처음 나온거를 설치하면 된다!! 자동완성, 문법검사 기능을 지원하여 준다.

(VS Code 에 아래와 같이 Angular 8 and TypeScript
를 설치함. 22/1/21 일 체크)



방금 설치한 앵귤러를 살펴보자. 주로 사용하게 될 명령어는 2 개 이다.

ng new 프로젝트 명

ng serve

첫번째 명령어는 프로젝트를 만드는 명령어이며 두 번째 명령어는 만든 프로젝트를 임시 서버로 돌리는 역할을 한다.

그리고 나중에 살펴볼 **ng build** 라는 명령어를 통해 코드를 javascript 로 컴파일하여 일반 프로젝트에 추가 할 수 있다.

그러면 이제 한번 시작 해 보자. 앞선 앵귤러와 vs 코드가 설치 되었다는 상황이다.

cmd 를 다시 열도록 하자. 만들고자 하는 프로젝트가 위치 할 디렉토리를 정한 뒤에 아래 명령어를 입력하자.

명령어를 입력하면 입력과 선택해야되는 부분이 나오는데 사진처럼 입력하고 선택하자.

* 명령어 : **ng new study**

* 디렉토리 : 여기서 사용한 디렉토리는 E 드라이브에 04_NODE 폴더로 하였다. (E:/04_NODE)

```
E:\#04_NODE>ng new study
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE study/angular.json (3798 bytes)
CREATE study/package.json (1304 bytes)
CREATE study/README.md (1022 bytes)
CREATE study/tsconfig.json (435 bytes)
CREATE study/tslint.json (1621 bytes)
CREATE study/.editorconfig (246 bytes)
CREATE study/.gitignore (629 bytes)
CREATE study/src/favicon.ico (5430 bytes)
CREATE study/src/index.html (292 bytes)
CREATE study/src/main.ts (372 bytes)
CREATE study/src/polyfills.ts (2841 bytes)
CREATE study/src/styles.css (80 bytes)
CREATE study/src/test.ts (642 bytes)
CREATE study/src/browserslist (388 bytes)
CREATE study/src/karma.conf.js (1018 bytes)
```

N 입력 후 첫번째 CSS 선택

해당 명령어를 입력하면 **study** 라는 폴더도 같이 만들어진다.

따라서 폴더를 굳이 안만들어주어도 된다.

해당 명령어를 입력하면 뭐라 묻는내용이 나오는데...

첫번째는 N 을 입력하고 두번째는 CSS 를 선택하여
주자!

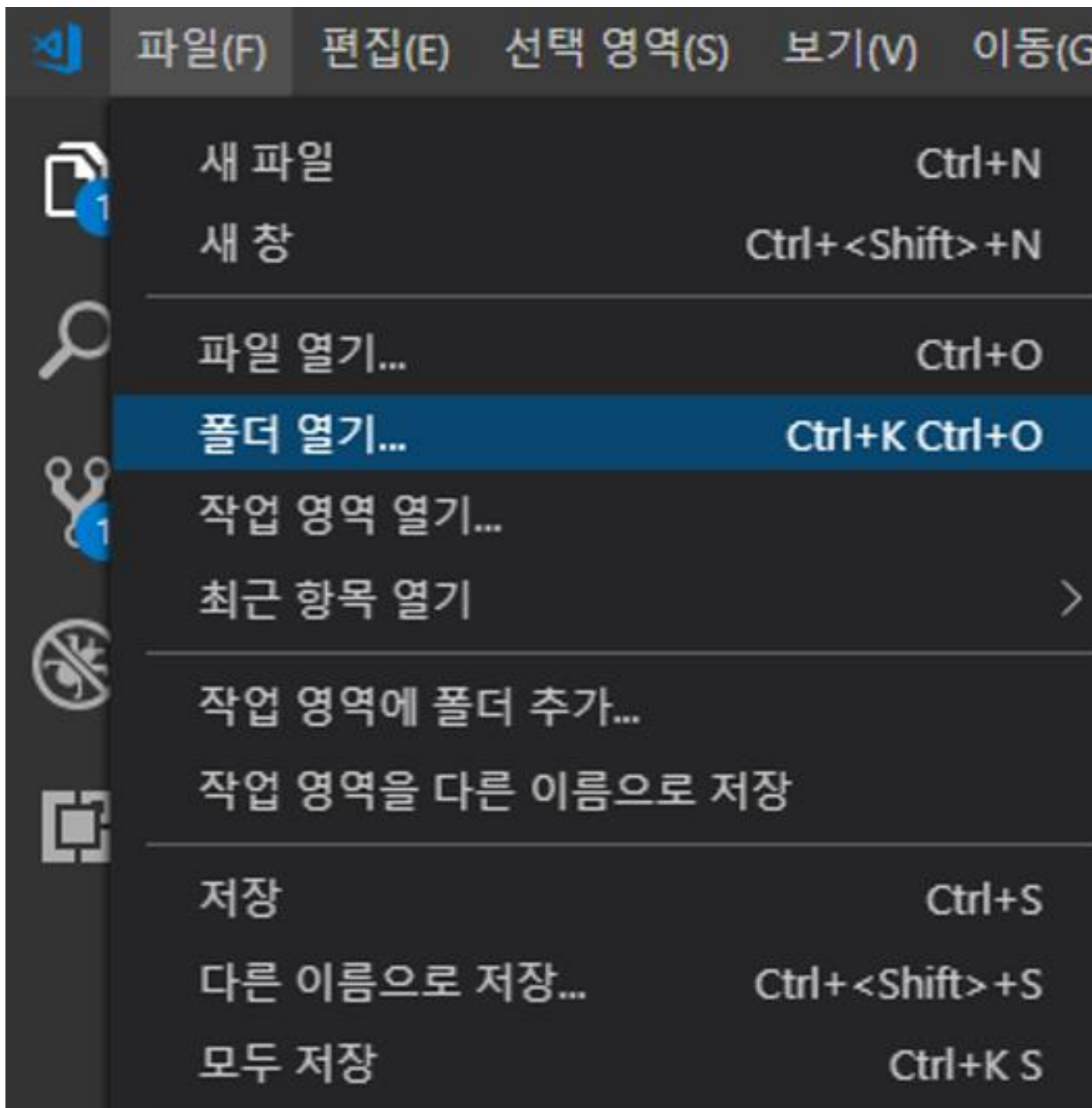
E:/04_NODE 라는 디렉토리에 study 라는 폴더가 생기
면서 앵글러와 관련된 기본 모듈이 설치된다.

설치하는데 시간이 좀 걸린다..뭐라뭐라 글씨도 엄청
나오고..

vs 코드를 선택해서 방금 만들어준 디렉토리를(study
디렉토리까지) 선택하여주자.

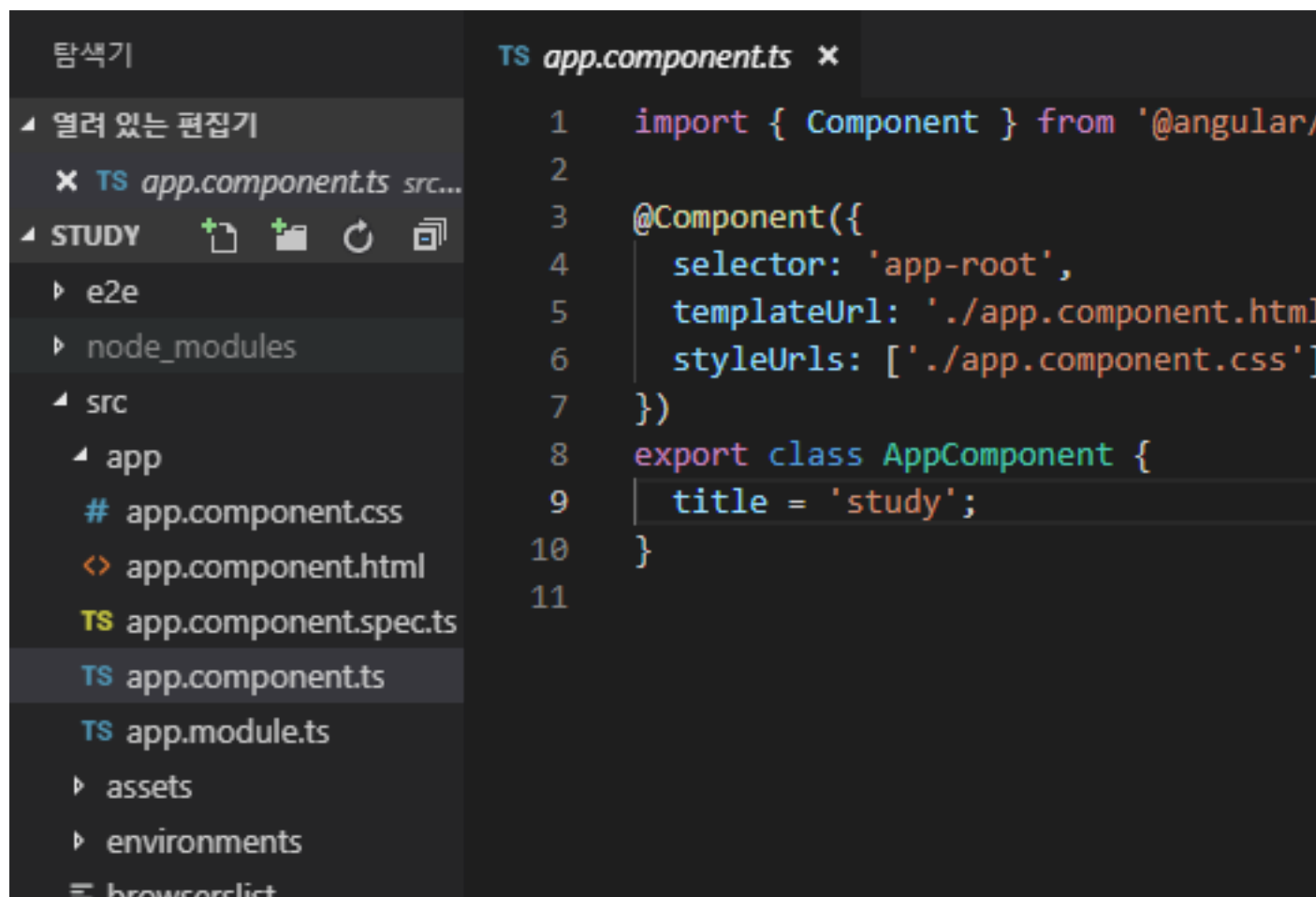
여기서는 E:/04_NODE/study 까지 선택해야 된다.

드라이브 및 디렉토리명이 다르더라도 study 까지만
선택하자.



폴더열기를 클릭해서 해당 디렉토리(directory) 까지만 선택하자.

이후에 엄청난 디렉토리가 생기는데 전부 이해하는 것은 불가능하다. 일단 src - app 디렉토리까지 이동하자.



src 를 누른다음 app 을 눌러보자!!

해당 디렉토리에서 `app.component.ts` 를 클릭하자.

`app.component.ts` 파일을 줄여서 메인 컴포넌트라 하겠다.

메인 컴포넌트는 당분간 이해하기로는 Java 에서의 메인 메소드, Javascript 에서는 `<script></script>` 태그 정도로 생각하면 된다. 무언가 입력하면 실행되는 곳이라 보면 될 것 같다.

다음과 같이 입력하여 보자.

```
import { Component } from
```

```
'@angular/core';
```

```
@Component ({
```

```
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  title = 'study';
```

```
  constructor() { //생성자...?
```

```
    console.log('Working'); //콘솔.로
```

```
    그..?
```

```
  }
```

```
}
```

(study 폴더로 이동하고 ng serve 를 실행한다.

22/1/21 일 체크)

입력한 뒤에 ng serve 명령어를 cmd 창에서 입력하자.

추가로, CMD 를 VS 코드에서 사용 할 수 있다.

VS 코드를 사용하면 단축키 **Ctrl + ~** 를 누르면 아래 사진처럼 "터미널" 항목에 **CMD** 창이 등장하게 된다.

단축키를 눌러서 터미널창이 나왔는데 아무것도 표시가 안되면 엔터키를 몇번 입력하면 **CMD** 가 잘 나오게 된다.

탐색기

열려 있는 편집기

× TS app.component... M

STUDY

▸ e2e

▸ node_modules

▸ src

▸ app

app.component.css

<> app.component.html

TS app.component.spec.ts

TS app.component.ts M

TS app.module.ts

▸ assets

▸ environments

≡ browserslist

★ favicon.ico

<> index.html

K karma.conf.js

TS main.ts

TS polyfills.ts

styles.css

TS test.ts

{ } tsconfig.app.json

{ } tsconfig.spec.json

{ } tslint.json

⚙ .editorconfig

🔒 .gitignore

{ } angular.json

{ } package-lock.json

{ } package.json

📖 README.md

{ } tsconfig.json

{ } tslint.json

TS app.component.ts ×

```
1 import { Component } from '@angular/
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'study';
10
11   constructor() {
12     console.log('Working');
13   }
14 }
15
```

문제 출력 디버그 콘솔 터미널

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

PS E:\04_NODE\study> ng serve

** Angular Live Development Server is listening on 4200, browser at http://localhost:4200

Date: 2019-05-21T06:36:04.860Z

Hash: ba0b37b9780646e4b80a

Time: 21514ms

chunk {main} main.js, main.js.map (main) 9.86 KB

chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 1.12 KB

chunk {runtime} runtime.js, runtime.js.map (runtime) 1.12 KB

chunk {styles} styles.js, styles.js.map (styles) 1.12 KB

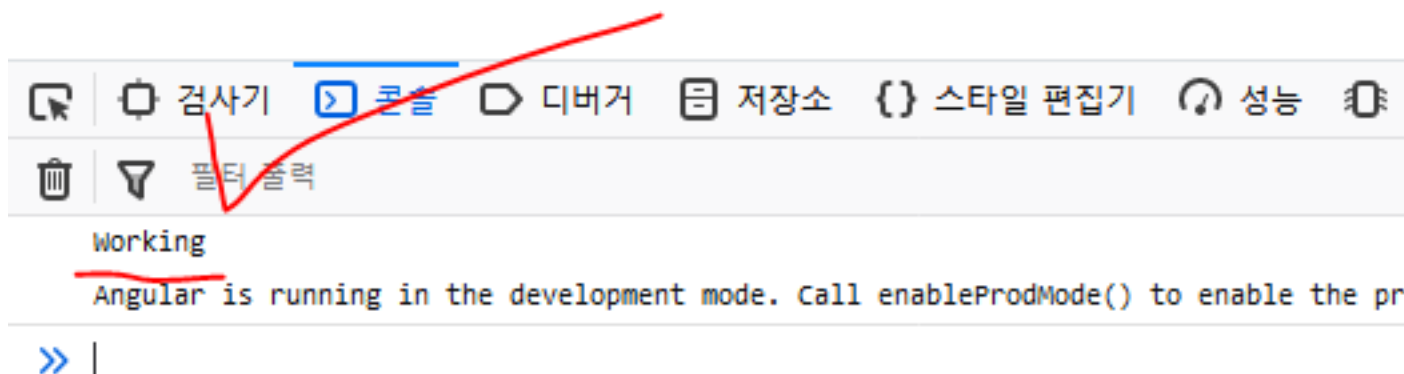
chunk {vendor} vendor.js, vendor.js.map (vendor) 1.12 KB

i [wdm]: Compiled successfully.

□

cmd 창을 열어서 사용하는 모습. cmd 는 Ctrl + ~ 키를 누르면 나온다.

이제 `http://localhost:4200` 으로 접속하여 보자. 접속한 뒤에 개발도구를 켜 보면 반가운 문구가 보인다.



워킹이 보인다?

constructor 로 입력을 한 뒤에 console.log 가 실행이 되었다.

일단 여기까지 하였다면 1 차목표는 성공 한 것이다.
그러면 아래 코드를 한번 입력하여보자.

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

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



```
constructor() {
```

```
    console.log('Working');
```

```
    var number = 1;    //자바스크립트에서 자
```

```
주보던 var
```

```
    console.log('number -> ', number);
```

```
    console.log(number * 10);    //연산..?
```

```
}
```

```
}
```

저장버튼을 누르는 순간 4200 번으로 동작하는 서버
가 무언가 행동을 한다. 그리고선 아무것도 안했는데
페이지가 새로고침되면서 입력한 값이 출력이 된다.

탐색기

열려 있는 편집기

× TS app.component.ts src\app M

STUDY

▸ e2e

▸ node_modules

▾ src ●

▾ app ●

app.component.css

<> app.component.html

TS app.component.spec.ts

TS app.component.ts M

TS app.module.ts

▸ assets

▸ environments

≡ browserslist

★ favicon.ico

<> index.html

K karma.conf.js

TS main.ts

TS polyfills.ts

styles.css

TS test.ts

{ } tsconfig.app.json

{ } tsconfig.spec.json

{ } tslint.json

⚙ .editorconfig

💎 .gitignore

{ } angular.json

{ } package-lock.json

{ } package.json

ⓘ README.md

{ } tsconfig.json

{ } tslint.json

TS app.component.ts ×

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'study';
10
11    constructor() {
12      console.log('Working');
13      var number = 1;
14      console.log('number -> ' + number);
15      console.log(number * 10);
16    }
17  }
18
```

문제 출력 디버그 콘솔 터미널

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

PS E:\04_NODE\study> ng serve

** Angular Live Development Server is running on port 4200.

Date: 2019-05-21T06:45:40.634Z

Hash: ba0b37b9780646e4b80a

Time: 14873ms

chunk {main} main.js, main.js.map (main)

chunk {polyfills} polyfills.js, polyfills.js.map (polyfills)

chunk {runtime} runtime.js, runtime.js.map (runtime)

chunk {styles} styles.js, styles.js.map (styles)

chunk {vendor} vendor.js, vendor.js.map (vendor)

i [wdm]: Compiled successfully.

i [wdm]: Compiling...

타입스크립트 파일이 변경되면 자동으로 재컴파일을 해 준다.

이 4200 번으로 동작중인 서버는 앵글러를 개발하기

위한 임시 서버라고 생각하면 된다. 즉 사용자가 코

딩을 하면 해당 소스코드의 변경사항을 감지하여

Javascript 로 컴파일 해 준 뒤에 화면에 보여주는 역

할을 한다.

동작 원리, 프로세스, 실제 서버와의 연동, 데이터베

이스 사용 등등 궁금한 내용이 정말 많지만, 여기서

는 이정도까지 만족하도록 하자.

정리하여보면,

1. Node.js 설치

2. Node.js 설치 이후 타입스크립트 설치, cmd 창에서

명령을 입력하자 : `npm install -g typescript`

3. cmd 창에서 명령을 입력하자 : `npm i @angular/cli -`

`g`

4. VS 코드(비주얼 스튜디오 코드)를 설치하자.

5. Angular 8 Snippets 를 설치하자.

6. 프로젝트를 만들 디렉토리를 정한 뒤 `ng new` 프로젝트명(여기서는 `study` 로하였다)을 입력하자.

7. 프로젝트명으로 디렉토리가 만들어지는데 해당 디렉토리로 이동해서 `ng serve` 명령어를 입력하자.

8. `src/app` 디렉토리에서 `app.component.ts` 파일을 클릭하자

9. 클릭한 파일에 `constructor(){}` 를 입력하고
`console.log` 등을 입력하여 보자.

다음장에서는 기초 부분인 자료형과 변수에 대해서
알아보자.

* 앵귤러 버전은 6.0 기준으로 되어있습니다.

* 상위 버전(9.0 이상)에서는 동작과 관련하여 소소한
오류가 있을 수 있습니다.

* 가장 어려운 것이 초기 환경구성입니다.

* 기본 세팅이 어렵거나 이해되지 않으시면 언제든지
연락주세요. ^.^

* 앞서 이야기드린바와 같이 **CMD** 를 전혀 사용하지 못하거나 자바스크립트기초가 부족하면 이해하기가 불가능 합니다.

전시간에는 기본적인 환경 세팅에 대해 살펴 보았다.

뭐가뭔지 전부 알기에는 방대하므로 아직까지는 `app.component.ts` 만 사용해서 차근차근 알아가도록 하자.

기존에는 window 키와 R 키를 누른뒤에 CMD 를 입력해서 실행하였다면,

비주얼 스튜디오 코드에서는 **Ctrl + ~** 키를 눌러서 **cmd** 를 실행 할 수 있다.

명령어는 해당 프로젝트 디렉토리에서 **ng serve** 를 입력하자.

요 **ng serve** 라는 명령어는 앵귤러를 개발할 때 필요한 테스트 서버를 동작시켜주는 역할을 한다.

일단..앵귤러를 개발하려면 **ng serve** 를 입력해서 개발용 서버를 동작시켜주는 것으로 이해하자.

열려 있는 편집기 1이(가) 저장되지 않음

● TS app.component.ts src\app M

STUDY

▸ e2e

▸ node_modules

▾ src ●

▾ app ●

app.component.css

<> app.component.html

TS app.component.spec.ts

TS app.component.ts M

TS app.module.ts

▸ assets

▸ environments

≡ browserslist

★ favicon.ico

<> index.html

K karma.conf.js

TS main.ts

TS polyfills.ts

styles.css

TS test.ts

{ } tsconfig.app.json

{ } tsconfig.spec.json

{ } tslint.json

⚙ .editorconfig

💎 .gitignore

{ } angular.json

{ } package-lock.json

{ } package.json

① README.md

{ } tsconfig.json

{ } tslint.json

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'study';
10
11    constructor() {
12      console.log('Working');
13      var number = 1;
14      console.log('number -> ' + number);
15      console.log(number * 10);
16    }
17  }
18
```

문제 출력 디버그 콘솔 터미널

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

PS E:\04_NODE\study> ng serve

** Angular Live Development Server is running on port 4200. Open your browser at http://localhost:4200/

Date: 2019-05-21T07:27:32.629Z

Hash: 624d51d4ff5f0aca6481

Time: 22481ms

chunk {main} main.js, main.js.map (main)

chunk {polyfills} polyfills.js, polyfills.js.map (polyfills)

chunk {runtime} runtime.js, runtime.js.map (runtime)

chunk {styles} styles.js, styles.js.map (styles)

만약 위와 같은 화면을 만나지 못하였다면 걱정할 필요 없다.

비주얼 스튜디오코드를 실행한 뒤에 맨 위의 파일 - 폴더열기를 클릭하여주자.

아래 사진처럼 폴더열기를 클릭한 뒤에 이전시간에 만든 **study** 폴더까지만 선택하여주면 된다.



폴더 열기에서 study가 있는 곳 까지만 선택 하여야

한다!!!

실행이 되었다면 이제 자료형에 대해서 먼저 살펴보자.

Typescript 기반으로 만들어진 앵귤러는 Typescript 언어의 자료형을 그대로 지원한다. 타입스크립트를 조금 살펴보려면 아래 링크에서 타입스크립트와 관련된 연습을 해 보아도 좋다.

<https://lts0606.tistory.com/17>

TypeScript 시작

Javascript 문법이 강력해지고 규모가 커지면서 코드의 관리가 중요해졌다. 자바스크립트는 객체지향 언어의 개념보다는 동적타입언어, 느슨한타입 언어로써 코드량이 많아짐에 따라 규모가 큰 프로젝트에서는 관..

이번에도 역시 **constructor** 내부에 코딩을 통해서 자료형의 종류를 살펴 보겠다.

앵귤러에서는 string, number, boolean 및 기타 배열 등 자바스크립트에서 사용되던 대부분의 자료형이 사용 가능하며,

자바처럼 접근제어자(private, public)의 기능도 지원한다.

예제코드를 살펴보자.

```
constructor() {
```

```
    console.log('Working');
```

```
var number = 1;
```

```
console.log('number -> ', number);
```

```
console.log(number * 10);
```

```
var trueFalse = true;
```

```
console.log(trueFalse);
```

```
}
```

constructor 부분만 보면 그리 어렵지가 않다.

콘솔 출력은 브라우저에서 볼 수 있다

<http://localhost:4200/>

문법적인 오류를 방지하기 위해서 자료형을 명시할 수 있게 지원해 준다.

아래 코드를 보자.

```
constructor() {
```

```
var num : number = 1;
```

```
console.log(num);
```

```
var text : string = 'text';
```

```
console.log(text);
```

```
var all : any = 'abcd';
```

```
all = 1;
```

```
}
```

"var 변수 : 자료형태 = 값" 이러한 규칙이 있다는게 놀라울 따름이다.

즉 해당 변수에 다른 자료형이 들어가면 오류가나게 된다.

자세히 보면 any 라는 녀석을 볼 수 있는데 해당 자료형은 모든지 받는, 즉 Object 개념을 지니고 있다.

swift 가 생각나는건 뭘까..

다음 코드를 살펴보자. (args?는 에러 발생 22/1/19 일
체크)

```
constructor() {
```

```
    var text : string = 'hello world';
```

```
    console.log(text);
```

```
    var num : number = 1234;
```

```
    console.log(num);
```

```
    var bool : boolean = false;
```

```
    console.log(bool);
```

```
    var method = (arg = " ") :
```

```
void=>{ console.log(arg);} //void 로 선언
```

```
    // 당연히 var method : void 는 안된다.
```

```
즉 함수에서 리턴값이 없을 때 붙인다.
```

```
method('print');
```

```
var data : any = null; //객체값이 비
```

어있음

```
console.log(data);
```

```
}
```

(arg = ""이렇게 하면 에러 없이 실행됨. 22/1/21 일
체크)

가운데 method 라는 녀석이 있다. 요녀석은 람다표
현식으로 이루어 졌으며 자바스크립트로 비교하자면
아래처럼 표현 할 수 있다.

```
var method = function(arg=""){console.log(arg);}
```

```
method('print');
```

이미 눈치챘을지 모르겠지만 여태껏 사용하던 **constructor** 는 생성자를 의미한다.

즉, 해당 `app.component.ts` 가 실행될 때 **딱 한번만 최초에 무조건 같이 실행해 주는 역할을 한다.**

굳이 왜 쓰는지 는 나중에 알게 되므로 일단..무조건 딱 한번만 최초에 실행되는 함수라고 이해하자.

사용자가 어떠한 이벤트나 명령을 따로 주지 않아도 실행되는 함수라고 생각하면된다!!

그러면, 생성자에 전부 코딩을 할 수 없으니 다른 함수를 만들어 보겠다.


```
import { Component } from
```

```
'@angular/core';
```

```
@Component({
```

```
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  title = 'study';
```

```
  public printAll(arg="") {
```

```
    console.log(arg);
```

```
  }
```

```
  private print(arg="") {
```

```
    console.log(arg);
```

```
  }
```

```
  constructor() {
```

```
    var text : string = 'plain text';
```

```
    this.print(text);
```

```
this.printAll(text);  
}  
}
```

자바를 경험한 사람이라면 `private` 와 `public` 을 본 순간 이해했을 것이다.

`public` 이라는 접근 제어자는 외부 다른 곳에서도 해당 함수를 실행 할 수 있게 해주는 의미이며, `private` 는 해당 `app.component.ts` 내부에서만 사용 가능하다는 의미이다.

원소리나면..

`private` 가 붙은 함수는 다른 곳에서 호출을 못하고, `public` 이 붙은 함수는 다른 곳에서 불러올 수 있다는 의미이다.

일반 자바스크립트에 비유하면..내가 만든 function 이 private 가 붙으면 특정한 곳에서만 쓸 수있고, public 이 붙으면 아무곳에서나 호출 가능하다고 보면 된다.

* 앵귤러 9.0 이상의 버전에서는 private 접근 제어자가 붙을 경우 해당 컴포넌트에서만 접근이 가능하다.

* 다른 어떠한 곳에서도 private 가 붙은 곳에는 접근이 불가능 하다 (2020-04-14 작성)

지금이야 파일 1 개로 연습하지만 나중에 규모가 커지고 파일 갯수가 많아지게 된다면 private 와 public 의 의미가 잘 다가 올 것이다.

조금 특이한 것은 함수를 사용할 때 **this** 를 붙여서
쓴 점이다.

this 를 삭제하면 에러가 난다. 일종의 규칙이라 볼 수
있다.

모든 글로벌한 함수 및 변수는 반드시 this 를 사용
해 주어야 한다. (개발도구가 오류난다고 먼저 알려준
다. ^^)

마찬가지로 멤버변수에게도 자료형과 접근제어자를
쓸 수 있다.

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

```
@Component ({  
  selector: 'app-root',
```

```
templateUrl: './app.component.html',
```

```
styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
public title : string = 'study';
```

```
public num : number = 10;
```

```
public all : any = new Array();
```

```
public printAll(arg="") {
```

```
    console.log(arg);
```

```
}
```

```
private print(arg="") {
```

```
    console.log(arg);
```

```
}
```

```
constructor() {
```

```
    var text : string = 'plain text';
```

```
    this.print(text);    //규칙! this 가 불
```

```
어야 한다!
```

```
this.printAll(text);
```

```
console.log(this.title, this.num,  
this.all); //멤버들도 this 를 써 주어야 한  
다.  
}  
}
```

지금 것 살펴본 app.component.ts 는 일종의 백그라운드(화면을 구성하는 이외의 기능)의 개념이라 볼 수 있다.

즉, 화면을 구성하는 기능은 따로 있고 데이터 처리 변수 정의 및 함수를 app.component.ts 곳에서 한다고 생각하면 된다.

음..Javascript 로 치면 html 파일에서 <script></script>의 역할을 하는 곳으로 생각하면 된다.

위 상단의 @Component 라는 곳을 살펴 보면 내부에 templateUrl 이라는 곳이 존재한다. 해당 파일 또한 src - app 폴더에 존재하는데 해당파일을 클릭해보자.

클릭하면 여태껏 동작한 화면내용을 볼 수 있다.

탐색기

열려 있는 편집기

TS app.component.ts src\app M

✕ <> app.component.html src\app

STUDY

▶ e2e

▶ node_modules

▶ src

▶ app

app.component.css

<> app.component.html

TS app.component.spec.ts

TS app.component.ts M

TS app.module.ts

▶ assets

▶ environments

≡ browserslist

★ favicon.ico

<> index.html

📄 karma.conf.js

TS main.ts

TS polyfills.ts

styles.css

TS test.ts

{ } tsconfig.app.json

{ } tsconfig.spec.json

{ } tslint.json

⚙️ .editorconfig

📄 .gitignore

{ } angular.json

{ } package-lock.json

{ } package.json

📖 README.md

{ } tsconfig.json

{ } tslint icon

TS app.component.ts

<> app.component.html

1 <!--The content below is o

2 <div style="text-align:cent

3 <h1>

4 | Welcome to {{ title }}>

5 </h1>

6 <img width="300" alt="Ang

7 </div>

8 <h2>Here are some links to

9

10

11 | <h2><a target="_blank"

12

13

14 | <h2><a target="_blank"

15

16

17 | <h2><a target="_blank"

18

19

20

21

문제

출력

디버그 콘솔

터미널

chunk {polyfills} polyfills.js, polyfills.js.map (m

chunk {runtime} runtime.js, runtime.js.map (m

chunk {styles} styles.js, styles.js.map (m

chunk {vendor} vendor.js, vendor.js.map (m

i [wdm]: Compiled successfully.

i [wdm]: Compiling...

Date: 2019-05-21T07:42:44.746Z - Hash: 44444444

4 unchanged chunks

chunk {main} main.js, main.js.map (m

i [wdm]: Compiled successfully.

i [wdm]: Compiling...

여기있었군!!! html 파일내용이다.

자, 그럼 이 두개의 개념을 살펴보면 아래와 같은 정의를 낼 수 있다.

1. app.component.ts 는 자바스크립트의

document.ready 이후 실행할 *.js 파일 같은 개념이다.

2. app.component.ts 내부 @Component 안에

templateUrl 은 화면을 구성하는 부분이다.

다음장에서는 이러한 app.component.html 과

app.component.ts 와의 관계에 대해서 살펴보도록 하자.

*** 추가(2020-04-14)**

- **private** 와 관련된 속성이 해당 컴포넌트에서만 사용하도록 되어있습니다.

- 앵글러 버전을 9.0 이상으로 사용중이신 분들은 외부에서 접근하는 변수나 함수에 대해서는 **private** 부분을 제거하셔야 합니다.

```

src/app/app.component.ts:5:16
5   templateUrl: './app.component.html',
                        ~~~~~
Error occurs in the template of component AppComponent.
src/app/app.component.html:5:43 - error TS2341: Property 'im_text'
5   <input type="text" [(ngModel)]="im_text" /> <!--이건 처음보
                                           ~~~~~

src/app/app.component.ts:5:16
5   templateUrl: './app.component.html',
                        ~~~~~
Error occurs in the template of component AppComponent.

```

private는 해당 컴포넌트에서만 접근하도록 바뀌었습니다.

전 시간에는 ts 파일에 메소드와 자료형을 토대로 콘솔이 동작하는 것 까지 확인 해 보았다.

app.component.ts 는 아직 자세히 알 수는 없지만,
웹 화면이 동작할 때 어떠한 행동을 만들어주는, 마

치 Javascript 의 `<script></script>`내부에 쓰여진 코드 같은 역할을 하는 것으로 볼 수 있다.

그리고 잠깐 살펴본 `app.component.html` 은 `*.html`, `*.jsp`, `*.ejs`, `*.php` 등등..자바스크립트 노드(예를들면 `div` 태그, `input type` 태그 등등)가 포함 된 사용자가 실제로 보는 화면을 구성하는 것으로 볼 수 있다.

여기까지만 놓고 본다면, 타입스크립트 기반의 앵귤러는 화면을 구성하는 코드 / 동작에 대한 정의를 하는 코드가 2 가지로 구분되어 관리된다고 볼 수 있다.

조금 더 앞선 이야기지만, 이러한 `app.component.ts` 파일에 `ajax` 처럼 데이터를 주고받는 행위를 추가한다면 데이터베이스에 접속하는 개념의 행동도 할 수 있는 것이다.

전시간에 만든 파일을 살펴보자.

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

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

```
title : string = 'study'; //요녀석을 주
```

목하자

```
public num : number = 10; //요녀석도 주
```

목하자

```
public all : any = new Array();
```

```
public printAll(arg="") {
```

```
    console.log(arg);
```

```
}
```

```
private print(arg="") {
```

```
    console.log(arg);
```

```
}
```

```
constructor() {
```

```
    var text : string = 'plain text';
```

```
    this.print(text);
```

```
    this.printAll(text);
```

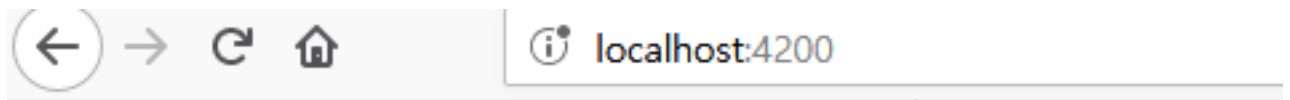
```
console.log(this.title, this.num,  
this.all);  
}  
}
```

app.component.html 파일을 클릭해서 전부 지우고 아래
두줄만 남겨본다.

```
<div>{{ title }}</div>  
<div>{{ num }}</div>
```

개발용 서버를 구동하여 보자. 구동방법은 프로젝트
디렉토리에서 `ng serve` 라는 명령어를 입력하면 된다.

접속은 `http://localhost:4200` 이다.



study
10

짜잔! 값이 나왔다.

이럴수가..ts 파일에 쓴 내용이(변수가) 표출 되었다.
app.component.ts 파일을 열어서 아래코드를 입력하
여 보자.

```
import { Component } from
```

```
'@angular/core';
```

```
@Component ({
```

```
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
```



```
styleUrls: ['./app.component.css']
```

```
}))
```

```
export class AppComponent {
```

```
  title : string = 'study';
```

```
  public num : number = 10;
```

```
  public all : any = new Array();
```

```
  public printAll(arg) {
```

```
    console.log(arg);
```

```
  }
```

```
  private print(arg) {
```

```
    console.log(arg);
```

```
  }
```

```
  constructor() {
```

```
    var text : string = 'plain text';
```

```
    this.print(text);
```

```
    this.printAll(text);
```

```
        console.log(this.title, this.num,  
this.all);  
    }
```

```
    public ifIwereGoToServer() : void  
{    //요 메소드이다.  
    //어떤 특정 데이터를 주는 서버에 갔다왔다  
    고 하자.  
    //예를들어 ajax 를 실행하고 나서 result  
    값이 아래와 같은 배열이 나왔다고 가정하자.  
    this.all.push('Result');  
    this.all.push(12345);  
    this.all.push(23456);  
    this.all.push(78923);  
    }  
}
```

다음으로 app.component.html 파일을 수정하여 보자.

```
<div>{{ title }}</div>
```

```
<div>{{ num }}</div>
```

```
<input type='button' value='서버에 갔다왔
```

```
다' (click)='ifIwereGoToServer()' />
```

```
<br><br>
```

```
<div *ngIf="all.length > 0">
```

```
  <div *ngFor="let item of all">서버에 갔
```

```
다왔다고 한 데이터 : {{item}}</div>
```

```
</div>
```

에..처음보는 녀석들이 추가되었다.html 에서 onclick
은 봤어도 (click)은 못봤는데 말이다..

(click)은 앵귤러에서 ts 파일에 존재하는 메소드 등을
실행 할 때 사용되는 의미이다.

즉, 내가 만든 함수를 클릭해서 실행하고 싶을때 위
처럼 써주면 된다.

여기까지는 직관적이라 사실 어렵지 않는데..

*ngIf 랑 *ngFor 는 뭐란말인가;; 대충 느낌적으로는
뭔가 if 는 조건문인거 같고, for 는 반복문인거 같은데
말이다.

여기서는 일단 우리가 app.component.ts 에서 선언한
all 이라는 변수를 출력하기 위한 용도로만 이해하자.

버튼을 누르면 빈 화면에서 갑자기 내용이 나타난다.

study

10

서버에 갔다왔다

버튼까지는 뭐.. 해했는데..
에...내용이 추가되었다??

study

10

서버에 갔다왔다

서버에 갔다왔다고 한 데이터 : Result
서버에 갔다왔다고 한 데이터 : 12345
서버에 갔다왔다고 한 데이터 : 23456
서버에 갔다왔다고 한 데이터 : 78923

버튼을 계속 누르면 계속 내용이 추가됨을 알 수 있
다.

우리가 확인해보아야 할 것은 `ifIwereGoToServer()` 함수이다.

해당 함수는 어떤 서버에서 데이터를 가져왔다고 가정한 메소드인데, 코드를 보면 데이터를 `push` 하고 있다. ***Javascript 에서 배열에 `push` 하는 기능과 똑같다.**

맨 처음에 `push` 를 하는 대상인 `all` 이라는 변수는 아무런 값을 가지고 있지 않다.

단지 `public all : any = new Array();` 로 새로 만들었을 뿐이었다.

ifIwereGoToServer() 메소드가 실행되면 push 를 통해서 데이터를 넣어준 것 뿐인데 신기하게도 화면에 넣은 데이터가 생겼다.

app.component.html 파일에서 아직은 알 수 없는 ***ngif** 와 ***ngfor** 라는 녀석이 all 이라는 변수에 데이터가 들어오자마자 화면에 태그가 추가된 것을 알 수 있다.

그렇다면 이렇게 생각 할 수 있다.

app.component.ts 파일에서 데이터를 받아오면 미리 코딩한 ***ngIf** 나 ***ngFor** 가 해당 데이터가 없으면 아무행동도 안하고 있다가 데이터가 추가되면 화면에 표출해 주는 것이다 라고 말이다.

사실, 위 내용은 전문용어로는 "양방향 바인딩"이라 불리우는데...그냥 app.component.ts 에서 선언한 데이터는 app.component.html 에서 바로바로 표현 할 수 있다 정도로만 이해하면 될 것 같다.

마지막으로 그렇다면 1 가지 샘플을 더 봐보자.ts 파일에 메소드 1 개를 추가한다.

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

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

```
export class AppComponent {
```



```
title : string = 'study';
```

```
public num : number = 10;
```

```
public all : any = new Array();
```

```
public printAll(arg) {
```

```
    console.log(arg);
```

```
}
```

```
private print(arg) {
```

```
    console.log(arg);
```

```
}
```

```
constructor() {
```

```
    var text : string = 'plain text';
```

```
    this.print(text);
```

```
    this.printAll(text);
```

```
    console.log(this.title, this.num,
```

```
    this.all);
```

```
}
```

```
public ifIwereGoToServer() : void{
```

```
//어떤 특정 데이터를 주는 서버에 갔다왔다  
고 하자.
```

```
this.all.push('Result');
```

```
this.all.push(12345);
```

```
this.all.push(23456);
```

```
this.all.push(78923);
```

```
}
```

```
public changeTitle(){ //새로 추가한 메
```

```
소드
```

```
this.title = '바꾼다..title';
```

```
}
```

```
}
```

이어서 html 파일에도 기능을 추가하자.

```
<div>{{ title }}</div>
```

```
<div>{{ num }}</div>
```

```
<input type='button' value='서버에 갔다왔  
다' (click)='ifIwereGoToServer()' />
```

```
<br><br>
```

```
<div *ngIf="all.length > 0">
```

```
  <div *ngFor="let item of all">서버에 갔  
다왔다고 한 데이터 : {{item}}</div>
```

```
</div>
```

```
<br><br>
```

```
<input type='button' value='바뀌어라 타이  
틀' (click)='changeTitle()' />
```

예상한대로 새로 등록한 버튼을 누르면 화면의 title
로 선언한 변수값이 변경이 된다.

앵귤러의 코딩방식은 위의 방식으로 이루어 지는 것이 거의 전부라 할 수 있다.

어떠한 ts 파일에서 각종 데이터에 대한 행동을 정의하고,

html 파일에서는 ngIf, ngFor 같은 기호들을 통해서 데이터를 표출하는 것이다.

그렇다면 아래처럼 정리가 가능하다.

1. ts 로 불리는 파일은 Javascript 에서의

<script></script> 역할을 한다.

2. ts 로 불리는 파일은 각종 데이터, 변수 및 함수를 보관하고 있다.

3. 화면을 구성하는 것은 html 파일로 이루어진다.

4. 특수한 기호를 활용하면 ts 파일에서 정의한 내용을 가져올 수 있다. -> 예: {{blabla}}, ngFor, NgIf...

다음시간에는 ngFor, ngIf 같은 녀석이 뭐하는 녀석인지와 app.component.ts 와 app.component.html 관계에 대해서 조금 더 깊게 알아보자.

이번에는 component.ts 와 component.html 의 관계에 대해서 조금 더 알아보도록 하자.

화면에서 사용자가
시킬 수 있는 명령들



app.component.ts



화면 구성에 필요한
정보들

먼저 `app.component.ts` 라는 파일은 앞서 이야기한바와 같이 화면을 구성하기 전 파일이다.

`html` 파일이 생성되기 전에 해야될 일을 적어두고, 적어놓은 각종 규칙, 설정 및 기능에 대해서 준비를 한 다음에 `html` 파일을 만들어서 사용자에게 보여주는 역할을 담당한다.

계속 살펴 보았던 `app.component.ts` 파일을 열어보자.

```
import { Component } from
```

```
'@angular/core';
```

```
@Component({
```

```
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
//생략..
```

```
}
```

해당 컴포넌트파일에서 @Component()라는 부분을 살펴보자.

해당 부분은 앵귤러의 데코레이터 중 1개로 해당 파일이 컴포넌트임을 명시하는 역할을 한다. (네..? 뭔 소리??)

쉽게 말해서, 해당 파일은 *.html 파일이 만들어지기 전에 앵귤러가 읽어들이는 설정파일 같은 개념이다.

해당 데코레이터에 어떠한 내용도 없다면 앵귤러는 동작을 하지 않는 다는 것 이다.

단순한 개념으로 본다면 데코레이터 = json 형식 기반
의 앵귤러 설정 방법이라 볼 수 있다.

간단한 예제로 조금 더 살펴보자.

아래처럼 코드를 변경하자.

```
@Component ({  
  selector: 'app-root',  
  //templateUrl: './app.component.html',  
  //기존내용을 주석처리하고  
  template : '<div>뽕미</div><div>이것은  
title : {{title}}</div>', //요 부분만 추  
가 해 보자.  
  styleUrls: ['./app.component.css']  
})
```

놀랍게도 해당 내용을 적용하면 기존에

app.component.html 에 쓴 내용이 하나도 반영이 안

되고...template 라는 내용이 전부 반영 된 것을 알 수 있다.

사용자가 어떠한 내용을 추가하던지 간에

app.component.ts 파일이 먼저 읽혀서 화면을 구성

할 준비를 하고 그 이후에 *.html 파일을 구성하는

것을 볼 수 있다.

만약 기존 html 파일을 보기 운한다면 template 를

주석(삭제)하고 기존 templateUrl 을 주석해제 하도록

하자.

앵귤러에서 화면구성 = 컴포넌트 임을 기억하자.

앵귤러를 통해서 html 을 구성 할 때는 "컴포넌트에 의해서 화면이 구성된다." 라는 개념을 잊지 말도록 하자.

이러한 컴포넌트는 컴포넌트 데코레이터 를 통해서 사용할 수 있다는 점도 기억하자.

그럼 @Component 밑의 export class 라는 기호는 뭘까?

export 는 내용 그대로 방출, 보내는 뜻이다. export 를 하지 않으면 내부에서만 사용되는 의미가 되며, 해당 내용을 지우고 실행하여보면 아래 사진처럼 오류를 보게 된다.

```
❗ Error: Unexpected value 'undefined' declared by the module 'AppModule'
⚠ [WDS] Warnings while compiling.
⚠ ./src/app/app.module.ts 11:16-28
  "export 'AppComponent' was not found in './app.component'
⚠ ./src/app/app.module.ts 17:24-36
  "export 'AppComponent' was not found in './app.component'
>>
```

뭐..뭐라구요..?

컴포넌트를 구성 할 때 export 를 반드시 붙인다고만
기억하자.

export 앞에 class 는 각종 메소드, 변수등을 묶는 꾸
러미 같은 개념으로 이해하면된다.

맨 위 상단의 import 는 javascript 에서의 <script
src='파일.js' /> 와 동일한 기능이다.

원하는 js 파일을 가져올 때 처럼 마찬가지로 앵귤러에서의 import 도 동일한 기능이다.

정리하여보면,

1. 컴포넌트 파일은 화면을 구성하는 기초 파일 이다.

2. 컴포넌트 파일에 쓰여진 코드가 읽혀지고 나서,
이후에 화면을 구성 한다.

3. @Component 는 앵귤러의 데코레이터 일종이다.

4. 컴포넌트 파일을 구성 할 때는 @Component 데코레이터와 export class 이름{} 라는 부분이 필요하다.

여기까지 이해를 하였다면 이제 ngIf 와 ngFor 에 대해서도 이해하기가 쉬워진다.

컴포넌트 파일에서 컴포넌트 데코레이터의 template 를 지우고(주석) templateUrl 로 바꾸자.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  //template : '<div>뿔미</div><div>이것은 title : {{title}}</div>',
  styleUrls: ['./app.component.css']
})
```

요렇게..다시 원래대로 돌립시다.

app.component.html 파일을 살펴보자. 컴포넌트 데코레이터에서 templateUrl 을 명시 하였기 때문에

app.component.html 파일이 읽힐 것이다.

ngIf 는 컴포넌트.ts 파일에서 쓰여진 변수들이 실제 존재 하는지를 지속적으로 감시(?)하는 **디렉티브**라고 한다.

사용법은 컴포넌트.ts 파일에서 선언한 변수에 대해서 조건을 주는 방식으로 쓰여진다.

조건문은 일반 자바스크립트와 거의 동일하다.

마찬가지로 ngFor 는 컴포넌트.ts 파일에서 쓰여진 변수에 대해서 해당 변수가 배열형태의 반복문이 실행 가능한 데이터 이면 해당 데이터에 맞추어서 실행되는 개념이다.

```
<div *ngFor="let item of all">서버에 갔다  
왔다고 한 데이터 : {{item}}</div>
```

let item 은 all 이라는 변수에서 반복문을 통해 나오는
1 개의 데이터를 의미한다.

{{item}}이라고 쓰여진 부분은 all 이라는 변수에서 반
복문을 통해 나온 각각의 데이터를 의미한다.

```
<div *ngFor="let item of all;let idx =  
index;">서버에 갔다왔다고 한 데이터 : {{idx  
+ 1}}, {{item}}</div>
```

idx 는 순서, 반복문이 동작할 때의 순서를 의미한다.
콜론; 을 통해서 내용을 구분지을 수 있다.

앵글러가 편한 것은, 이러한 내용을 선언만 해 두고 데이터가 변하는 내용에 따라서 동적으로 적용시켜 준다는 점이다. 즉, 지금은 데이터가 없지만 데이터가 바뀌거나, 생기거나, 삭제되는 모든 행동이 서로 통신하듯 적용되어 바로바로 바뀐다는 점이다. (양방향 데이터바인딩....)

여기까지 이해가 되지 않는다면 다음부터의 내용이 많이 어려울 수 있다.

최대한 쉽게 풀어서 내용을 기술하였기에 명칭, 용어가 난해할 수도 있다.

다음장에서는 데이터를 통한 화면구성에 대해 조금 더 알아보도록 하자.

앵귤러의 전체 구조, 실제 서버에 적용시키는 방법,
데이터베이스와의 관계 등등에 대해서는 일단 잠시
넘어가도록 하자.

간단하게 내용을 정리하여보면,

1. 데코레이터(@로 시작하는)는 앵귤러의 설정을 담당하여준다.
2. 컴포넌트 데코레이터(@Component)는 화면구성을 담당하는 파일이다.

안녕하세요.

앵귤러에 대해서 알아보기위해 이곳을 찾아주신 분들

께 감사의 말씀 드립니다.^^

천천히, 초심자도 조금 더 쉽게 접근할 수 있도록 내용을 구성하여 보겠습니다.

어려운 부분이나 이해가 가지 않는 부분은 언제든지 연락주세요.

* 기본적으로 CLI(커멘드 라인 인터페이스)를 전혀 모르시거나, 자바스크립트를 최소한 1 년이상 하지 않으셨다면 접근하기는 매우 어렵습니다.

* 난이도를 위해 ES6(ECMA6) 이상의 문법을 최대한 사용하지 않고 소개하고 있습니다. :)

앵귤러는 웹 화면을 구성하는 데 사용되는 프레임워크입니다.

Javascript, JQuery 처럼 각종 명령어와 규칙을 통해서 웹 화

면에 이벤트와 기능을 부여하고, 사용자가 원하는 행동을 하기 위해 만들어졌습니다.

이와 비슷한 유형의 프레임워크, 라이브러리 형태가 바로 리액트, 뷰 js 등이 있습니다.

html 의 내용과 기능을 만들기 위해서 사용하는 프레임워크(JS) 라고 단순하게 생각하시면 될 것 같습니다.

앵귤러는 2020 년 5 월 기준 버전 9 까지 나왔습니다.

앵귤러 버전 1 은 소위 말하는 JQuery 처럼 라이브러리를 등록해서 사용하는 방식 이였다면,

앵귤러 버전 2 부터는 타입스크립트를 활용하여 개발한 뒤에 컴파일 하여 주는 방식으로 변경이 되었습니다.

타입스크립트는 사실 아직까지는 웹 개발자들이 많이 사

용하지 않는 언어입니다.

타입스크립트는 자바스크립트를 조금 더 엄격하게, 규칙 있게 사용하기 위해서 만들어진 언어입니다.

여기까지 소개했는데..무언가 어렵고 복잡하여 보입니다.
간단하게 압축해서 설명드리면,

앵귤러는 "타입스크립트 기반으로 앵귤러의 문법대로 개발하는 언어" 라고 생각하시면 될 것 같습니다. ^^

사실 타입스크립트는 자바스크립트와 거의 동일한 구조입니다.

그러므로 굳이 걱정하지 않으셔도 됩니다!

가끔 이런 이야기를 들어본 것 같습니다.

"앵귤러가 리엑트, 뷰 js 보다 익히기 어렵다."

솔직히 전부 해 보았지만 개인적 취향차이 일 뿐 전
~부 어렵습니다..^-^;

이번시간에는 **기본 개발 환경구성**에 대해서 살펴보겠습니다.

환경 구성에 필요한 내용으로는,

1. Node.js 를 설치 해야 합니다.
2. 개발 도구를 설치 해야 합니다.
3. 타입스크립트를 Node.js 의 라이브러리 설치명령어의 도움을 받아 설치합니다.
4. 앵귤러를 Node.js 의 라이브러리 설치명령어의 도움을 받아 설치합니다.

Node.js 를 설치하는 이유는 앵귤러를 개발하는데 필요한 라이브러리 설치와, 작업에 필요한 테스트서버를 사용하기 위해서 입니다.

npm 이라는 라이브러리를 관리하는 명령어를 통해서 우리는 앞으로 각종 앵귤러에 필요한 라이브러리를 일일이 다운로드 받지 않고 몇번의 명령어를 통해서 쉽게 받도록 할 것 입니다.

여기서 개발도구는 비주얼 스튜디오코드를 사용하겠습니다. 다른 개발도구를 사용해도 상관 없습니다.

먼저 Node.js 를 다운받은 뒤에 자신의 운영체제에 맞는 버전을 설치하도록 합니다.

<https://nodejs.org/ko/>

Node.js

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
nodejs.org

다음으로, 개발에 필요한 개발도구인 비주얼 스튜디오 코드를받습니다.

<https://code.visualstudio.com/>

Visual Studio Code - Code Editing. Redefined

Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications. Visual Studio Code is free and available on your favorite platform - Linux, macOS, and Windows.
code.visualstudio.com

* 설치방법은 매우 쉬우므로 생략하겠습니다.

비주얼 스튜디오 코드를 받게 되면 앵글러 개발에 필요한 확장팩을 설치하여 코드 문법검사, 자동완성 등을 사용할 수 있습니다.

또한 한글도 지원합니다!

아래처럼 필요한 기능을 추가하여 설치하도록 합니다.



파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G)



확장: 마켓플레이스



15



korean



검색어 입력



Korean Language Pack for Visual Studio

한국어

Microsoft



Korean-translator 0.4.5

한->영 번역기

dinner



vscode-spellchecker-korean 0.0.1

This extension is a spell checker for ko

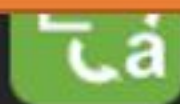
William Kim



Gaji - Gherkin Feature Indentation

Gaji is a formatter for Gherkin(cucumber)

clarekang



translator 1.1.0

translate for Korean

sculove



단디 - 한국어 맞춤법 검사기 0.1.1

fallroot

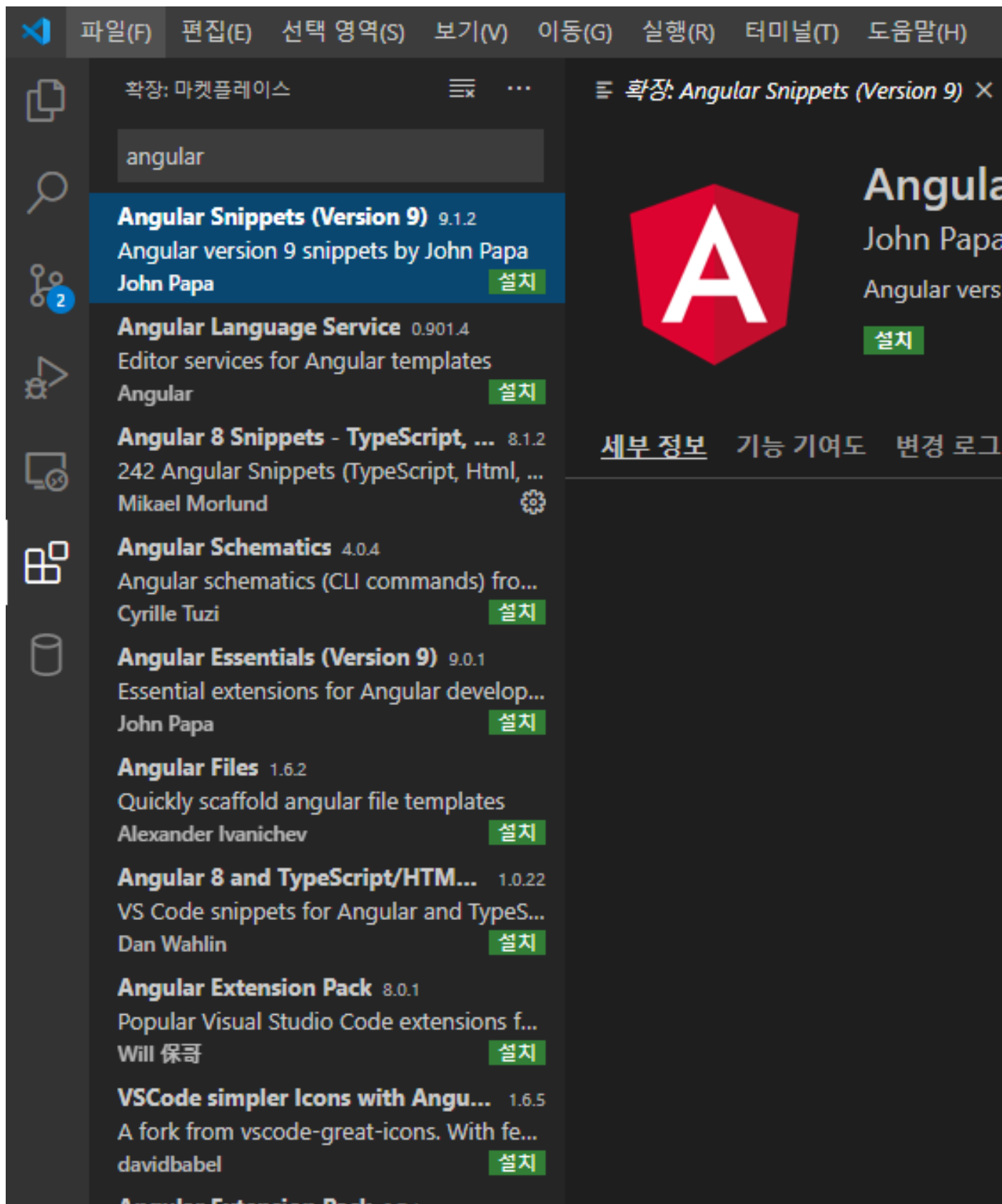


BLACKPINK 0.0.2



클릭

원하는 각종 기능을 검색해서 추가 할 수 있습니다.
korean 이라고 검색하면 비주얼 스튜디오코드를 한글로 바꾸어줍니다.



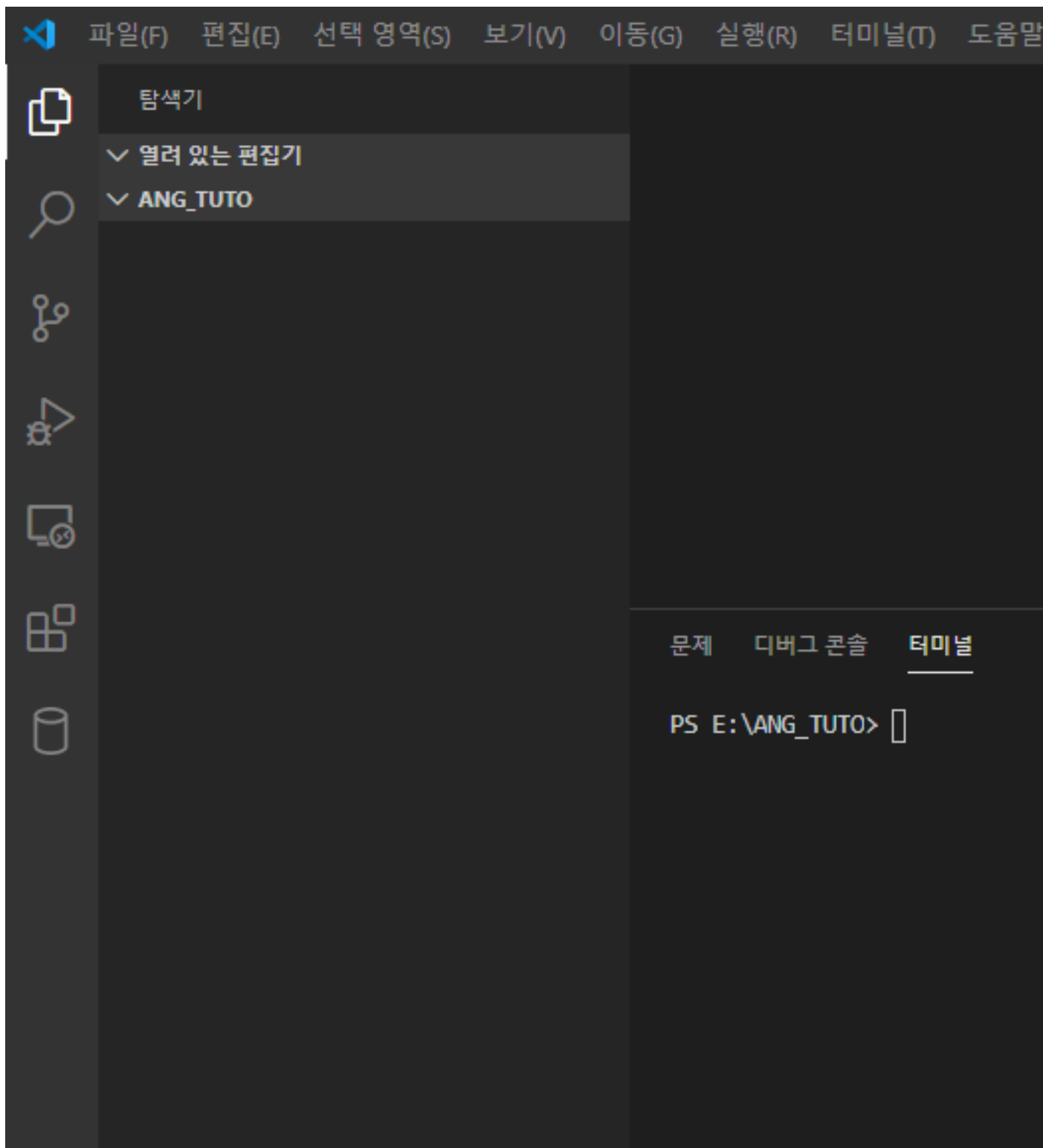
angular 라고 검색하면 최 상단에 검색되는 프로그램을

추가하여 코드 문법검사, 자동완성 기능을 제공받도록 합니다.

비주얼 스튜디오 코드를 실행하여봅니다.

맨 위의 상단에 파일(F) 탭을 눌러 앞으로 프로젝트를 담은 폴더를 만든 뒤 선택하여줍니다.

* 여기서는 E 디렉토리에 **ANG_TUTO** 를 선택하였습니다.



ANG_TUTO 라는 디렉토리를 만든뒤에 비주얼 스튜디오코드로 폴더를 선택하였습니다.

키보드 `Ctrl + ~` 를 활용해서 `cmd` 를 실행하도록 합니다.

키보드 `Ctrl + ~` 를 활용해서 `cmd` 가 실행되었는데 사진처럼 커서가 보이지 않는다면 엔터키를 몇번 입력하면 위 모습처럼 깜빡이는 커서가 나타나게 됩니다.

`Node.js` 를 설치하였으므로 우리는 앵귤러에 필요한 라이브러리를 이제 `npm` 명령어를 통해서 쉽게 사용할 수 있습니다.

만약 `npm` 명령어를 쓰지 않고 앵귤러에 필요한 라이브러리를 일일이 다운받는다면 그것은 무척 힘든 일이 될 것입니다.

cmd 내부에 아래 명령어를 입력하여 앵귤러 개발에 필요한 라이브러리를 설치합니다.

타입스크립트를 사용하기 위해서 관련된 라이브러리를 npm 명령어를 통해 설치 합니다.

```
npm install -g typescript
```


문제 디버그 콘솔 터미널

```
PS E:\ANG_TUTO> npm install -g typescript  
[.....] - rollbackFailedOptional: verb npm-session c9011db1
```

타입스크립트 언어를 설치합니다.

다음으로 앵귤러를 사용하기 위해서 아래 명령어를
입력합니다.

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

문제 디버그 콘솔 터미널

```
PS E:\ANG_TUTO> npm i @angular/cli -g  
[.....] / loadIdealTree:loadAllDepsIntoIdealTree: sill in
```

앵귤러 프레임워크를 설치합니다.

npm 명령어를 통해 타입스크립트와 앵귤러에 관련한 라이브러리를 전부 설치하였습니다!

이제 그러면 기본 연습을 위해 프로젝트를 만들어 보겠습니다.

아래 명령어를 입력합니다.

```
ng new firstStudy
```

문제 디버그 콘솔 터미널

```
PS E:\ANG_TUTO> ng new firstStudy
```

```
? Would you like to add Angular routing? (y/N) █
```

라우팅 할껀지 물어봅니다..

무언가 물어보는데 여기서 N 을 눌러주도록 합니다.

그리고나서 선택하는 모양이 나오는데, 사진처럼 css

를 선택하여줍니다.

문제 디버그 콘솔 터미널

```
PS E:\ANG_TUTO> ng new firstStudy
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS  [ https://sass-lang.com/documentation/syntax#scss
  Sass  [ https://sass-lang.com/documentation/syntax#the-index
  Less   [ http://lesscss.org
  Stylus [ http://stylus-lang.com
```

css 라고 보이는 탭을 선택한 뒤에 엔터!

뭔가 설치되고있는 모습입니다.

시간이 좀 걸리므로 느긋하게 기다려줍니다.

문제 디버그 콘솔 터미널

```
PS E:\ANG_TUTO> ng new firstStudy
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE firstStudy/angular.json (3598 bytes)
CREATE firstStudy/package.json (1287 bytes)
CREATE firstStudy/README.md (1027 bytes)
CREATE firstStudy/tsconfig.json (489 bytes)
CREATE firstStudy/tslint.json (3125 bytes)
CREATE firstStudy/.editorconfig (274 bytes)
CREATE firstStudy/.gitignore (631 bytes)
CREATE firstStudy/browserslist (429 bytes)
CREATE firstStudy/karma.conf.js (1022 bytes)
CREATE firstStudy/tsconfig.app.json (210 bytes)
CREATE firstStudy/tsconfig.spec.json (270 bytes)
CREATE firstStudy/src/favicon.ico (948 bytes)
CREATE firstStudy/src/index.html (296 bytes)
CREATE firstStudy/src/main.ts (372 bytes)
CREATE firstStudy/src/polyfills.ts (2835 bytes)
CREATE firstStudy/src/styles.css (80 bytes)
CREATE firstStudy/src/test.ts (753 bytes)
CREATE firstStudy/src/assets/.gitkeep (0 bytes)
CREATE firstStudy/src/environments/environment.prod.ts (51 bytes)
CREATE firstStudy/src/environments/environment.ts (662 bytes)
CREATE firstStudy/src/app/app.module.ts (314 bytes)
CREATE firstStudy/src/app/app.component.html (25725 bytes)
CREATE firstStudy/src/app/app.component.spec.ts (954 bytes)
CREATE firstStudy/src/app/app.component.ts (214 bytes)
CREATE firstStudy/src/app/app.component.css (0 bytes)
CREATE firstStudy/e2e/protractor.conf.js (808 bytes)
CREATE firstStudy/e2e/tsconfig.json (214 bytes)
CREATE firstStudy/e2e/src/app.e2e-spec.ts (643 bytes)
CREATE firstStudy/e2e/src/app.po.ts (301 bytes)
/ Installing packages...[]
```

시간이 좀 걸립니다..5 분?

설치가 완료되고 나면 firstStudy 라는 디렉토리가 만들어진 것을 볼 수 있습니다.

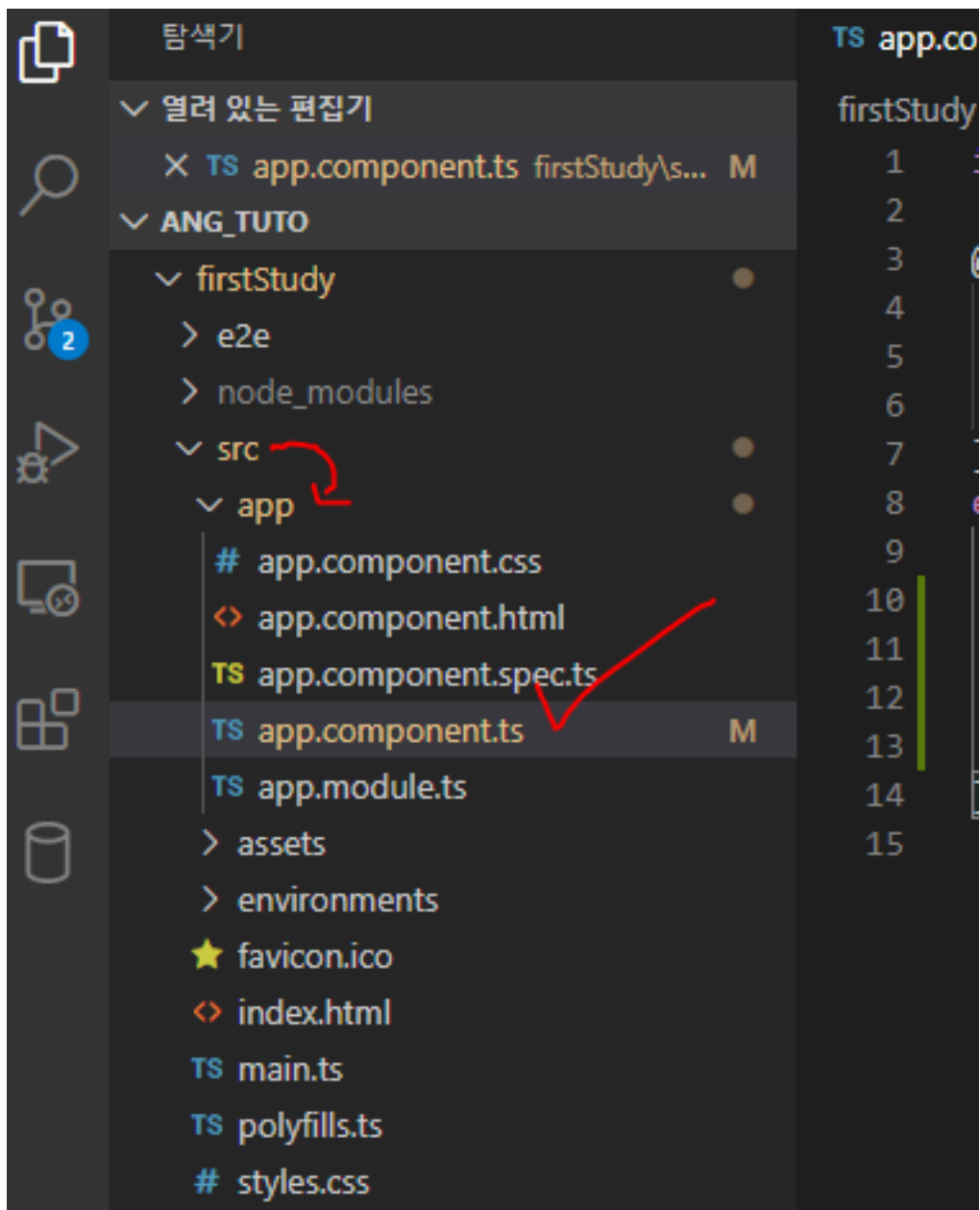
ng 라는 명령어는 앵귤러에서 사용하는 명령어입니다.

ng new "대상" 을 입력하면 새로운 앵귤러 프로젝트를 만드는데, "대상" 이름으로 만들어 달라는 내용입니다.

프로젝트를 새로 만들기위한 명령어 입니다.

아직 전부 알 수는 없지만 많은 내용의 목록이 보입니다.

여기서 src 의 app 탭까지 이동하여 app.component.ts 파일을 더블클릭하여 봅니다.



app.component.ts 파일에 아래 내용으로 변경합니다.

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

```
@Component ({
```

```
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  title = 'firstStudy';
```

```
  constructor () { //constructor 라는 함
```

```
수를 추가합니다.
```

```
    console.log('동작');
```

```
  }
```

```
}
```

constructor 라는 함수를 추가해보았습니다.

constructor 는 생성자의 의미로 최초 단 1 번만 동작하는 함수라고 일단 이해하시면 될 것 같습니다.

firstStudy 디렉토리로 이동 한 뒤에 아래 명령어를 입력하
여봅니다.

반드시 새로 만들어진 디렉토리로 이동한 뒤에 명령
어를 입력해야 합니다!

```
ng serve
```



The screenshot shows a terminal window with a dark background. At the top, there are three tabs: '문제' (Problem), '디버그 콘솔' (Debug Console), and '터미널' (Terminal), with '터미널' being the active tab. The terminal content shows the following commands and output:

```
PS E:\ANG_TUTO> cd .\firstStudy\  
PS E:\ANG_TUTO\firstStudy> ng serve  
? Would you like to share anonymous usage data about this project with  
Google under Google's Privacy Policy at https://policies.google.com/  
details and how to change this setting, see http://angular.io/analyt
```

무언가 물어보네요~

해당 프로젝트에 대한 내용을 공유할건지 물어보는데..원하는 데로 입력합니다.

여기서는 n 을 입력하였습니다. ^^;

아래 사진과 같은 모양이나왔다면 성공입니다!

문제 디버그 콘솔 터미널

```
M Compiling @angular/core : es2015 as esm2015
Compiling @angular/animations/browser : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/core/testing : es2015 as esm2015
Compiling @angular/common/testing : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/animations/browser/testing : es2015 as esm2015
Compiling @angular/common/http : es2015 as esm2015
Compiling @angular/forms : es2015 as esm2015
Compiling @angular/platform-browser/testing : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
Compiling @angular/common/http/testing : es2015 as esm2015
Compiling @angular/router : es2015 as esm2015
Compiling @angular/platform-browser/animations : es2015 as esm2015
Compiling @angular/platform-browser-dynamic/testing : es2015 as esm2015
Compiling @angular/router/testing : es2015 as esm2015

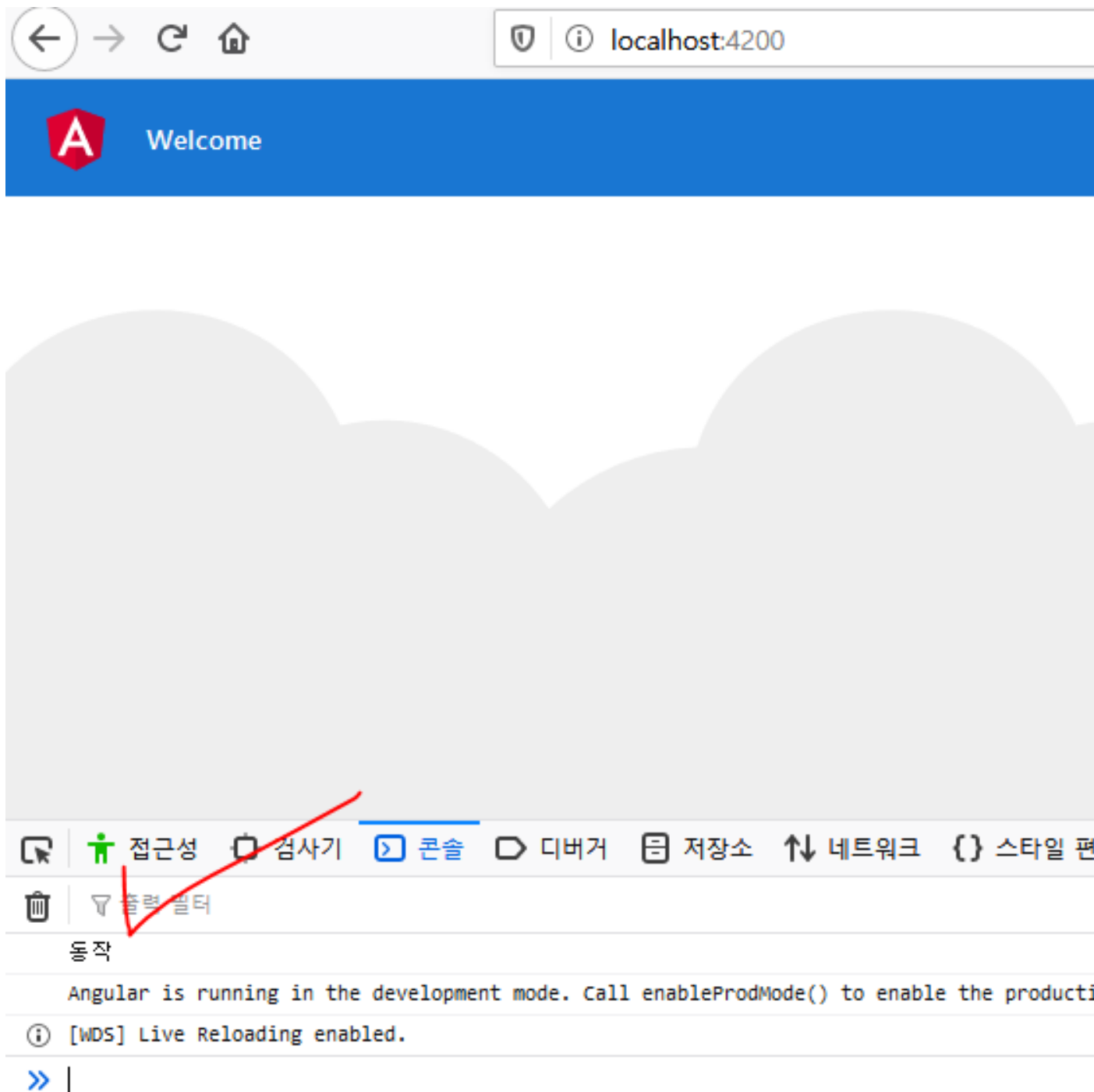
chunk {main} main.js, main.js.map (main) 57.8 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [initial] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 12.4 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 2.71 MB [initial] [rendered]
Date: 2020-05-06T01:25:27.587Z - Hash: c989b85bdacfe11e7f2e - Time: 10000ms
** Angular Live Development Server is listening on localhost:4200
: Compiled successfully.
```

맨 아래 문구에 컴파일 성공이라고 하는 내용이
보입니다.

친절하게도 콘솔에서 `http://localhost:4200/` 로 접속하라
고 안내해주고 있습니다.

브라우저에 <http://localhost:4200/> 로 접속하여봅니다.

개발도구를 실행하여보면 `constructor` 에 입력하였던 "동작" 이라는 내용이 출력되고있음을 볼 수 있습니다.



"동작" 이라는 문구가 잘 보입니다. 해당 문구는

constructor 라는 함수에 있었던 console.log 명령어입니다.

여기까지가 앵귌러를 개발하기 위한 환경구성입니다.

많이 복잡하고 어려운 내용이 나온 것 같습니다.

정리하여보겠습니다!

1. 앵귌러는 타입스크립트 기반으로 구성된 프레임 워크 입니다.

2. 앵귌러를 개발하기 위해서는 Node.js 를 반드시 설치하여야합니다.

* 각종 라이브러리를 다운받기 위해서는 npm 명령어가 매우 쉽고 편리하기 때문 입니다.

* 또한 앵귌러 테스트 서버를 동작시키기 위해서도 필요 합니다.

3. Node.js 가 설치되면 타입스크립트 언어를 설치하고, 앵귤러 프레임워크를 설치합니다.

* `npm install -g typescript`

* `npm i @angular/cli -g`

4. `ng new "대상"` 을 입력하면 해당 디렉토리에 "대상" 이름으로 새로운 앵귤러 프로젝트를 만들어 줍니다.

5. 해당 디렉토리에서 `ng serve` 라는 명령어를 실행하면 테스트 서버가 4200 번 포트를 활용하여 동작 합니다.

* 만약 4200 번 포트가 사용중이라면 `--port` 라는 옵션을 주어 실행하도록 합니다. "`ng serve --port 포트`"

이번시간에는 각 파일이 어떤것을 의미하는지, 어떠한 내용인지 알 필요가 없습니다.

"동작" 이라는 내용이 4200 번 포트에서 정확하게 출력되기만 한다면 성공입니다!

다음시간에는 왜 앵귤러를 사용하는지, 앵귤러를 통해서 어떠한 방식으로 개발하는지 등에 대해서 살펴보겠습니다!

* 추가

앵귤러, 리엑트, 뷰js 등 프론트 프레임워크(라이브러리)는 대체적으로 버전이 자주 올라갑니다.

이에 따라 기존에 사용되었던 기능이 버전업에 따라 동작을 하지 않을수도 있습니다.

만약 동작을 원활하게 되지 않는다면 아래

package.json 내용에서의 의존성(dependencies) 버전을 맞추어 주세요.

```
{
```

```
  "dependencies": {
```

```
    "@angular/animations": "~9.1.4",
```

```
    "@angular/common": "~9.1.4",
```

```
    "@angular/compiler": "~9.1.4",
```

```
    "@angular/core": "~9.1.4",
```

```
    "@angular/forms": "~9.1.4",
```

```
    "@angular/platform-browser":
```

```
      "~9.1.4",
```

```
    "@angular/platform-browser-dynamic":
```

```
      "~9.1.4",
```

```
    "@angular/router": "~9.1.4",
```

```
    "rxjs": "~6.5.4",
```

```
    "tslib": "^1.10.0",
```

```
    "zone.js": "~0.10.2"
```

```
},
```

```
"devDependencies": {
```

```
  "@angular-devkit/build-angular":
```

```
    "~0.901.4",
```

```
  "@angular/cli": "~9.1.4",
```

```
  "@angular/compiler-cli": "~9.1.4",
```

```
  "@angular/language-service":
```

```
    "~9.1.4",
```

```
  "@types/node": "^12.11.1",
```

```
  "@types/jasmine": "~3.5.0",
```

```
  "@types/jasminewd2": "~2.0.3",
```

```
  "codelyzer": "^5.1.2",
```

```
  "jasmine-core": "~3.5.0",
```

```
  "jasmine-spec-reporter": "~4.2.1",
```

```
  "karma": "~5.0.0",
```

```
  "karma-chrome-launcher": "~3.1.0",
```

```
  "karma-coverage-istanbul-reporter":
```

```
    "~2.1.0",
```

```
  "karma-jasmine": "~3.0.1",
```

```
"karma-jasmine-html-reporter":  
  "^1.4.2",  
  "protractor": "~5.4.3",  
  "ts-node": "~8.3.0",  
  "tslint": "~6.1.0",  
  "typescript": "~3.8.3"  
}  
}
```

앵귤러 환경구성은 문제없이 잘 끝내셨는지요!

이번시간에는 앵귤러를 왜 쓰는지, 어떻게 사용해서 실제로 적용하는지에 대해서 먼저 살펴보도록 하겠습니다.

예전의 HTML 에서의 개발방식은 Javascript, JQuery 를 사용하여 소위 돔(Dom) 이라는 객체를 직접 제어하

고 데이터를 가져와서 서버로 전달해 주는 방식이었습니다.

그러다 보니 조금 개발적 지식이 있는 사람들은 브라우저를 실행한 뒤에 개발자 도구를 켜서 직접 객체에 접근해서 데이터를 변조해서 서버로 전송할 수도 있었습니다.

또한 HTML5 로 기능이 업그레이드 되면서 엄청난 웹 기술발전이 이루어짐에 따라 많은량의 프론트 코드가 늘어가게 되었습니다.

서버언어가(자바의 jstl, php 의 echo 나 node.js 의 표현식 등) html 페이지에 존재하는 경우도 있었습니다. 그러하다보니 유지보수나 개발이 어려워지고 복잡해지는 경우가 많이 생겨났습니다.

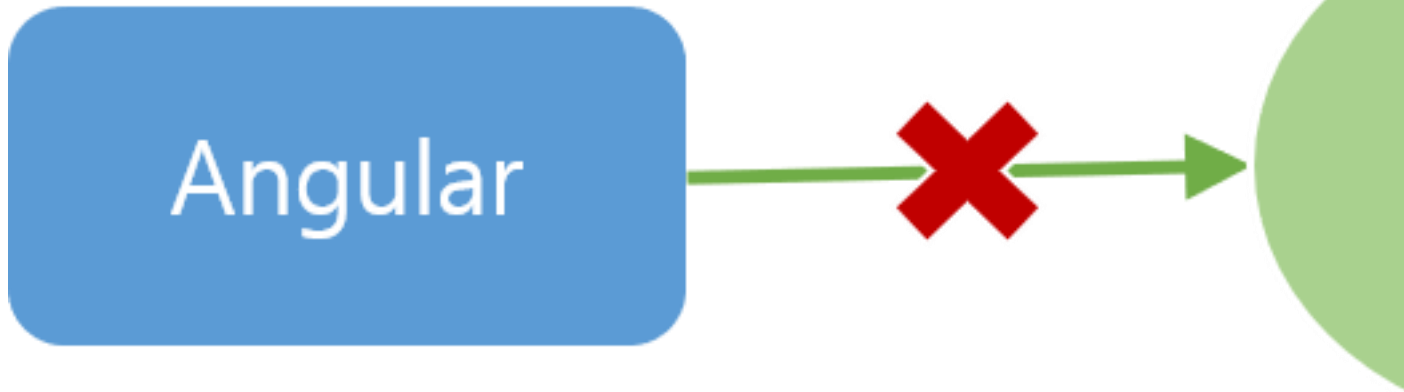
프론트 코드의 보안과 복잡하게 얽혀있는 코드를 개선해
보고자 만들어진 프레임워크가 바로 앵귤러 입니다.

앵귤러는 순수 100% 클라이언트 코드입니다.

**여기서 클라이언트 코드란, 웹 브라우저에서만 동작하
는 코드를 말합니다.**

그러므로 데이터베이스 연동, 다른서버와의 직접 연
동 등 이러한 행위는 직접 할 수가 없습니다.

웹 브라우저



앵귤러는 100% 브라우저에서만 동작하는 코드입니다!

앵귤러관련 프로젝트를 하는 경우 일반적으로 2 가지 타
입의 개발자(역할)가 필요하게 됩니다.

1. 앵귤러 개발자.

2. 서버개발자.

* 물론 혼자서 2 가지 다 하는 경우도 있겠습니다...

ㅠ

앵귤러 개발자는 소위 ajax 형식으로 서버 개발자에게 데

이터 CRUD 와 관련된 내용을 전달하고, 그에 따른 결과를 화면에 나타내는 작업만 합니다.

백그라운드 요청을 담당하는 개발자는 자신의 서버언어 (Node, php, Java, 파이썬 등)를 활용해서 앵글러에서 요청한 데이터를 처리하는 작업만 합니다.

두 개발자가 작업이 끝나게 되면, 앵글러를 통해서 화면을 만들던 개발자가 자신이 만들던 코드를 서버를 만든 개발자에게 전달하게 되고, 해당 프로젝트가 최종적으로 합쳐지게 되는 것이 앵글러를 통한 개발방식의 모습 입니다.

개발중



앵귤러는 결국에는 백그라운드서버에 합치게 되어 1개의 완성된 프로젝트 모양이 됩니다.

그러므로 백그라운드와 프론트의 분리가 완벽하게 일어나게 되며, 조금 더 각각의 역할과 기능에 집중할 수 있게 되는 것입니다.

조금 이르긴 하지만, 첫번째 시간에서 만든 프로젝트를 완성본이라 생각하고 합치는 작업을 한다고 가정해보겠습니다.

기존에 만든 firstStudy 프로젝트 디렉토리로 이동하여봅시다.

그리고 아래명령어를 입력합니다.

```
ng build
```

ANG_TUTO

- firstStudy
 - dist
 - e2e
 - node_modules
 - src
 - app
 - # app.component.css
 - <> app.component.html
 - TS app.component.spec.ts
 - TS app.component.ts M
 - TS app.module.ts
 - assets
 - environments
 - ★ favicon.ico
 - <> index.html
 - TS main.ts
 - TS polyfills.ts
 - # styles.css
 - TS test.ts
 - .editorconfig
 - .gitignore
 - { } angular.json M
 - ≡ browserslist
 - 📄 karma.conf.js
 - { } package-lock.json
 - { } package.json
 - 📖 README.md
 - { } tsconfig.app.json
 - 📄 tsconfig.json
 - { } tsconfig.spec.json
 - { } tslint.json

```

3  @Component({
4      selector: 'app-root',
5      templateUrl: './app.component.html',
6      styleUrls: ['./app.component.css'],
7  })
8  export class AppComponent {
9      title = 'firstStudy';
10
11      constructor() {
12          console.log('AppComponent');
13      }
14  }
15

```

문제 디버그 콘솔 터미널

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

새로운 크로스 플랫폼 PowerShell

```

PS E:\ANG_TUTO> cd .\firstStudy
PS E:\ANG_TUTO\firstStudy> ng build
Generating ES5 bundles for development...
ES5 bundle generation complete.

```

```

chunk {polyfills} polyfills-es2015.js
chunk {runtime} runtime-es2015.js
chunk {runtime} runtime-es5.js
chunk {styles} styles-es2015.js
chunk {styles} styles-es5.js
chunk {main} main-es2015.js, main-es5.js
chunk {main} main-es5.js, main-es2015.js
chunk {polyfills-es5} polyfills-es5.js
chunk {vendor} vendor-es2015.js
chunk {vendor} vendor-es5.js
Date: 2020-05-06T05:18:16.211Z
PS E:\ANG_TUTO\firstStudy>

```

원지는 몰라도 build라고 하니까...만드는게 아닌가 싶습니다. (이것도 시간이 좀 걸립니다!!!)

dist 라는 디렉토리가 만들어지면서 알 수 없는 파일이 가득 들어왔습니다.

자세히보면 dist 에 index.html 파일이 만들어진 것을 볼 수 있습니다.

해당 파일들이 바로 우리가 ng new 로 만들어 주었던 거대한 프로젝트 파일이 최종적으로 적용 할 백그라운드 서버에서 사용가능한 파일로 변환된 모습입니다!



Welcome



firstStudy app is running!

Resources

Here are some links to help you get started:



[Learn Angular](#) >



[CLI Documentation](#) >



[Angular Blog](#) >

Next Steps

What do you want to do next with your app?



[New Component](#)



[Angular Material](#)



[Add PWA Support](#)



[Add Dependency](#)



[Run and Watch Tests](#)

와....dist에 있는 파일이 ng serve와 한 내용과 같습니다!!

해당파일을 위 사진처럼 간단하게 Node.js 에서 제공하는 http 서버로 동작시켜 실행하여 보았습니다.

아무런 문제없이 잘 동작하는 것을 볼 수 있습니다.

기능과 역할의 분리가 앵귤러의 최대 관심사가 아닌가 싶습니다. * 개인적 생각입니다!!

이처럼 서로의 역할과 기능을 분리하여 개발이 가능하도록 만든 프론트 프레임워크가 바로 앵귤러 입니다.

* 비슷한 기능을 가진 리엑트, 뷰 Js 도 이런 방식으로 개발과 협업을 하게 되어 있습니다.

이번시간에는 앵귤러를 왜 사용하는지, 또한 어떠한 방식으로 개발 및 적용하는지에 대해서 간단히 살펴 보았습니다.

전반적인 흐름에 대해서 꼭 이해를 하여야 합니다.

해당 페이지가 이해되지 않는다면 댓글 또는 메일로 연락주세요. ^^

다음시간에는 본격적으로 앵귤러의 파일이 어떠한 역할을 하는지, 어떻게 사용하는지에 대해서 차근차근 알아보겠습니다!

저번시간까지는 앵귤러 개발에 대한 환경구성, 그리고 사용하는 목적과 이유등에 대해서 살펴보았습니다.

이번시간에는 처음 설치한 firstStudy 프로젝트를 기반으로 앵귤러의 기본 구조에 대해 단계적으로 살펴보려 합니다

다.


앵귤러는 타입스크립트 기반으로 구성되어 있습니다.

타입스크립트는 자바스크립트와 거의 동일하며, 큰 특징으로는 데이터의 형태(type)가 존재하여 조금 더 엄격한 코드관리가 이루어 질 수 있게 합니다.

타입스크립트에 대한 내용을 살펴보시려면 아래 사이트에서 연습을 해보시는 것도 좋습니다.

<https://lts0606.tistory.com/17>

TypeScript 시작

Javascript 문법이 강력해지고 규모가 커지면서 코드의 관리가 중요해졌다. 자바스크립트는 객체지향 언어의 개념보다는 동적타입언어, 느슨한타입 언어로써 코드량이 많아짐에 따라 규모가 큰 

lts0606.tistory.com

가장 먼저 살펴볼 것은 **모듈(Module) - 컴포넌트(Component)**의 관계입니다.

앵귤러에서 "모듈과 컴포넌트만 익히면 앵귤러는 끝!" 이라고 할 정도로 두 기능이 매우 중요합니다.

여기서 모듈이란, 각종 설정과 관련된 내용이 존재하는 파일입니다.

컴포넌트는 모듈이 만들어준 환경에서 동작하여 실제 화면에 관한 내용을 직접적으로 수행하는 파일입니다.

우리가 일반적으로 자바스크립트로 작업을하면 html 파일에 `<script/>` 태그를 선언한 뒤에 다양한 함수, 객체등을 활용해서 html 태그의 이벤트를 처리합니다. 그러나 앵귤러에서는,

모듈이라는 파일을 활용하여 각종 기능에 대한 규칙, 필요한 라이브러리 및 정보등을 관리하도록 합니다.

컴포넌트라는 파일에서는 html 파일에서의 각종 이벤트에 대한 내용, 데이터에 대한 처리를 실시 합니다.

html 파일에서의 `<script/>`에 기록된 다양한 함수, 객체등을 앵귤러에서 역할과 종류를 분리해 놓은 개념이 바로 모듈과 컴포넌트라고 생각하시면 되겠습니다.

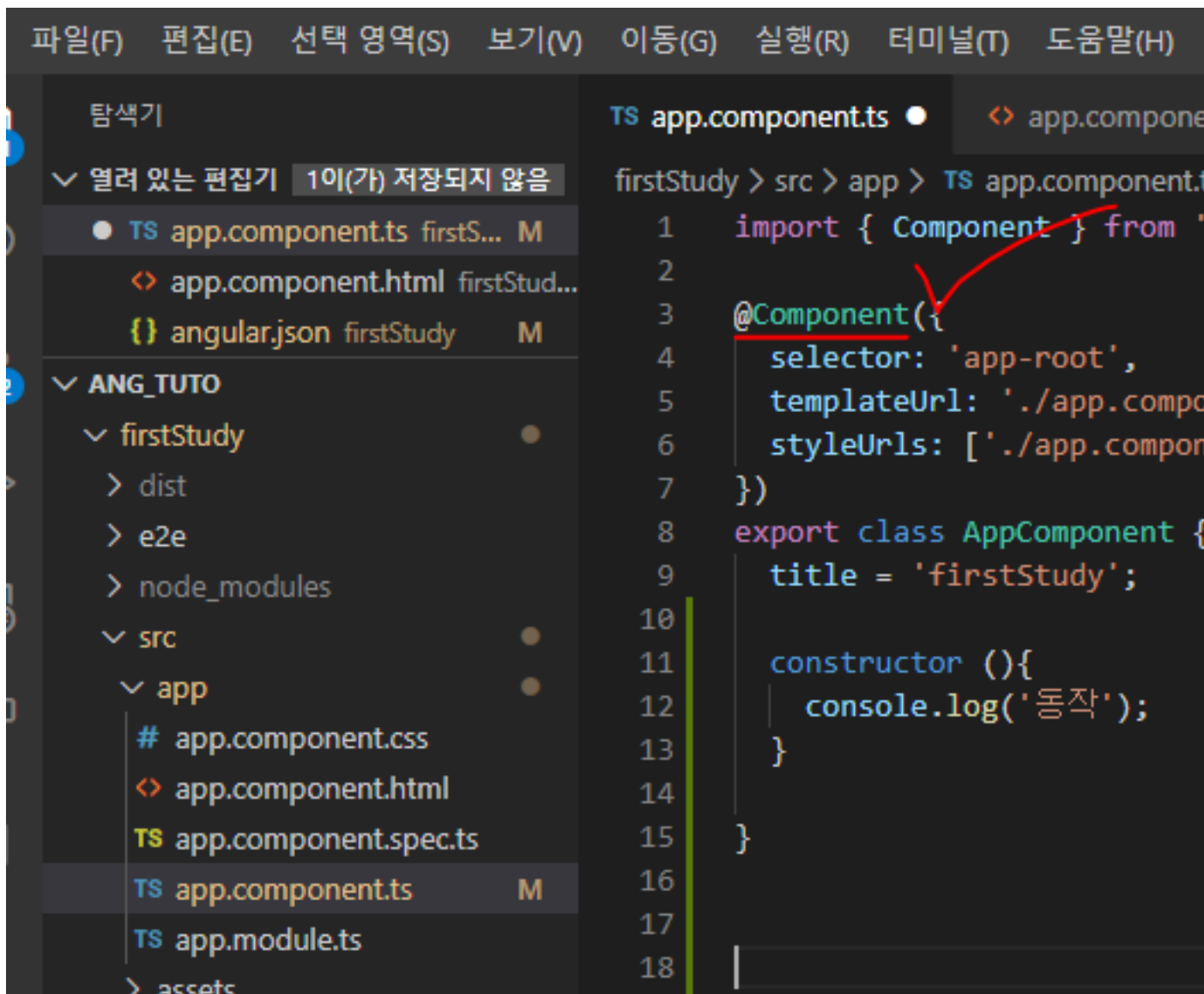
이에 대해 먼저 알아볼 내용이 바로 컴포넌트입니다.

컴포넌트는 앵귤러에서 화면구성 및 각종 이벤트에 대해서 일을 수행하는 일꾼입니다.

컴포넌트를 찾아보려면 프로젝트 firstStudy 에서 src - app 탭까지 이동한 뒤에 **app.componnt.ts** 파일을 선택

택하면 볼 수 있습니다.

해당 파일은 첫 번째 시간에서 constructor 라는 함수를 붙여넣어 주었던 파일입니다.



```
파일(F)  편집(E)  선택 영역(S)  보기(V)  이동(G)  실행(R)  터미널(T)  도움말(H)

탐색기
  ▼ 열려 있는 편집기 1이(가) 저장되지 않음
    • TS app.component.ts firstS... M
      <> app.component.html firstStud...
      {} angular.json firstStudy M
  ▼ ANG_TUTO
    ▼ firstStudy
      > dist
      > e2e
      > node_modules
      ▼ src
        ▼ app
          # app.component.css
          <> app.component.html
          TS app.component.spec.ts
          TS app.component.ts M
          TS app.module.ts
          > assets

TS app.component.ts
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'firstStudy';
10
11    constructor () {
12      console.log('동작');
13    }
14
15  }
16
17
18
```

Component 라는 문구가 보이시나요?

우리는 app.componnt.ts 파일에서 @Component 라는

부분을 살펴볼 필요가 있습니다.

앵귤러에서의 컴포넌트는 class 파일에 @Component 라는 꾸밈말(데코레이터)이 붙여진 파일을 의미합니다.

데코레이터는, 앵귤러에서 사용되는 일종의 명령어(예약어)이며, 해당 파일이 앵귤러에서 특수한 기능으로 사용되는 파일임을 알려줍니다.

* class 형태는 앵귤러의 기능이라고 하기 보다 ES6의 클래스 개념 or 타입스크립트의 클래스 개념이라 할 수 있습니다. ^^

* class 가 무엇인지 처음 보시는 분들은 아까 알려드린 링크 또는 Javascript class 라고 검색하신뒤 연습해 보시는게 좋습니다!

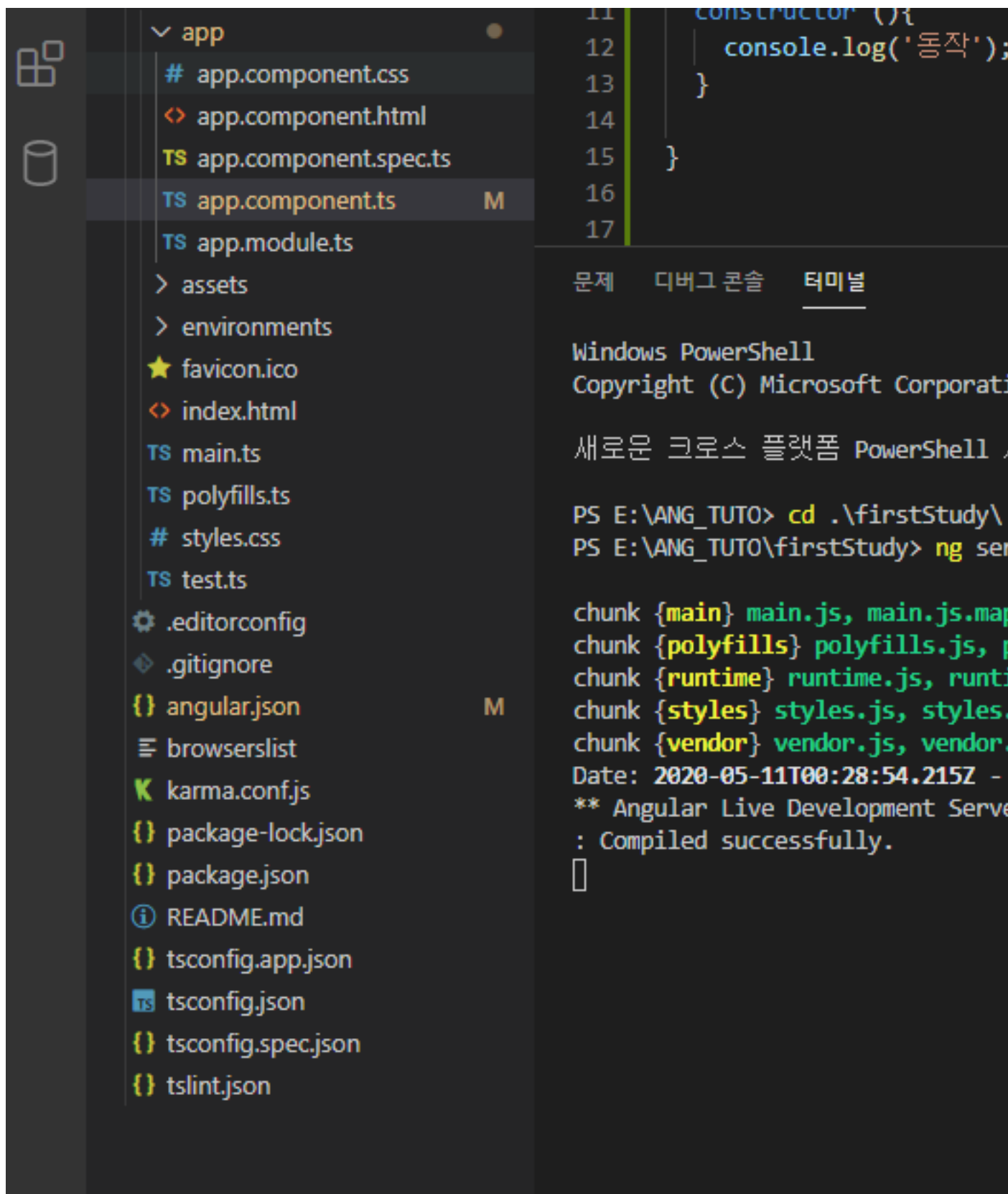
그러므로 @Component 기호는 "컴포넌트 데코레이터" 라고 합니다.

다시 정리를하면 @Component 기호가 붙어있는 class 파일은 앵귤러에서 컴포넌트를 의미합니다.

해당 파일이 어떻게 화면을 구성하는지 살펴보겠습니다.

"Ctrl + ~" 단축키를 누른뒤에 firstStudy 디렉토리로 이동합니다.

그리고 예전에 보았던 **ng serve** 라는 명령어를 입력 합니다.



구동하는데 역시 시간이 좀 걸리는군요! 차분하게
기다려주세요 ^^

app.component.ts 파일에서 title 이라는 값 부분을 원하는 값 아무렇게 입력하여봅니다.

그리고 웹 주소 **http://127.0.0.1:4200/** 를 입력하여 브라우저에서 실행하여 보면 아래 사진과같이 내용이 바뀌어서 출력됨을 볼 수 있습니다.

파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) 도움말(H)

탐색기

▼ 열려 있는 편집기

× TS app.component.ts firstS... M

<> app.component.html firstStud...

{ } angular.json firstStudy M

▼ ANG_TUTO

▼ firstStudy ●

> dist

> e2e

> node_modules

▼ src ●

▼ app ●

app.component.css

<> app.component.html

TS app.component.spec.ts

TS app.component.ts M

TS app.module.ts

> assets

> environments

★ favicon.ico

<> index.html

TS main.ts

TS polyfills.ts

styles.css

TS test.ts

⚙ .editorconfig

💎 .gitignore

{ } angular.json M

≡ browserslist

📄 karma.conf.js

{ } package-lock.json

{ } package.json

① README.md

{ } tsconfig.app.json

TS app.component.ts X

<> app.component.ts

firstStudy > src > app > TS app.component.ts

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = '아무값이나 입력해도 됩니다';
10
11    constructor() {
12      console.log('동작');
13    }
14  }
15
16
17
```

문제 디버그 콘솔 터미널

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

새로운 크로스 플랫폼 PowerShell 사용

PS E:\ANG_TUTO> cd .\firstStudy\

PS E:\ANG_TUTO\firstStudy> ng serve

```
chunk {main} main.js, main.js.map (main)
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills)
chunk {runtime} runtime.js, runtime.js.map (runtime)
chunk {styles} styles.js, styles.js.map (styles)
chunk {vendor} vendor.js, vendor.js.map (vendor)
Date: 2020-05-11T00:28:54.215Z - Hash: 1a2b3c4d
** Angular Live Development Server is running on port 4200.
: Compiled successfully.
```

```
Date: 2020-05-11T00:30:09.283Z - Hash: 1a2b3c4d
4 unchanged chunks
```

title 이라는 곳의 값을 바꾸었는데 html 내용이 바뀌었습니다!!

html 파일을 만진 것도 아닌데 내용이 바뀌어서 출력되는 것을 볼 수 있습니다.

바꾸고 싶은 값을 계속해서 입력해 보면, 입력한 내용이 상단에 출력됨을 알 수 있습니다.

이해를 위해 button 태그를 만든 뒤에 이벤트를 부여하여 보겠습니다.

먼저 버튼이 눌릴 경우 console 을 통해서 내용이 출력되는 함수를 만들어 보겠습니다.

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

```
@Component ({  
  selector: 'app-root',
```



```
templateUrl: './app.component.html',
```

```
styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  title = 'firstStudy';
```

```
  constructor () {
```

```
    console.log('동작');
```

```
  }
```

```
  public clickAfterPrint () : void{
```

```
    console.log('출력');
```

```
  }
```

```
}
```

조금 익숙하지 않는 void 라는 기호와 public 이라는 기호가 나타났습니다.

public 은 접근 제어자의 의미로 해당 함수가 어느 파일에

서든시간에 사용 할 수 있음을 의미합니다.

`void` 라는 타입은 해당 함수는 아무런 값을 반환하지 않는 것을 의미합니다.

접근제어자, 타입에 대한 내용은 다음시간에 다루어 보겠습니다.

일단 중요한 것은, `click` 이벤트를 만들었다는 점 입니다.

그러면 `src - app` 탭안에 존재하는 `app.component.html` 이라는 파일을 더블클릭하여 봅니다.

그리고 `html` 파일 내용을 전부지우고 아래처럼 바꾸어 줍니다.

```
<button
```

```
(click)='clickAfterPrint()'>{{title}} <-
```

```
당신이 입력한 변수 title </button>
```

click 이라는 기능은 아마도 클릭했을 때 이벤트를 의미하는 것 같습니다.

마치 html 에서의 onclick 처럼 기능으로 보입니다.

그리고 저장한 뒤 화면에서 버튼을 클릭하면 개발 도구 콘솔에 정상적으로 "출력" 이라는 단어가 나오는 것을 볼 수 있습니다.

편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) 도움말(H)

는 편집기

app.component.html fir... M

app.component.ts firstS... M

UTO

Study

t

e

de_modules

pp

app.component.css

app.component.html M

app.component.spec.ts

app.component.ts M

app.module.ts

assets

environments

favicon.ico

index.html

main.ts

polyfills.ts

styles.css

test.ts

tsconfig

tsconfig

angular.json M

browserslist

tsconfig.conf.js

package-lock.json

package.json

README.md

<> app.component.html X

firstStudy > src > app > <> app.component.html > button

```
1 | <button (click)='clickAfterPrint()'>{{t
```

TS app.component.ts X

firstStudy > src > app > TS app.component.ts > AppComponent

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'firstStudy';
10
11    constructor () {
12      console.log('동작');
13    }
14
15    public clickAfterPrint () : void {
16      console.log('출력');
17    }
18  }
```

문제 디버그 콘솔 터미널

"출력"이라는 단어가 잘 나오고 있습니다.

내용이 조금 어렵지만 핵심 부분은 "컴포넌트를 변경하니 웹 화면이 바뀌었다"라는 점입니다.

그러면, 다시 이제 `app.component.ts` 파일로 되돌아가 보겠습니다.

컴포넌트 데코레이터 안에 내용을 자세히 살펴보면 `templateUrl`이라는 문구가 보입니다.

파일(F) 편집(E) 선택 영역(S) 보기(V) 이동(G) 실행(R) 터미널(T) 도움말(H)

탐색기

▼ 열려 있는 편집기

1 그룹

× TS app.component.ts firstS... M

2 그룹

<> app.component.html fir... M

▼ ANG_TUTO

▼ firstStudy ●

> dist

> e2e

> node_modules

▼ src ●

▼ app ●

app.component.css

<> app.component.html M

TS app.component.spec.ts

TS app.component.ts M

TS app.module.ts

> assets

> environments

★ favicon.ico

<> index.html

TS main.ts

TS polyfills.ts

styles.css

TS test.ts

⚙ .editorconfig

💎 .gitignore

{ } angular.json M

≡ browserslist

⚡ karma.conf.js

{ } package-lock.json

{ } package.json

① README.md

{ } tsconfig.app.json

TS tsconfig.json

{ } tsconfig.spec.json

TS app.component.ts ×

firstStudy > src > app > TS app.component.ts > App

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'firstStudy';
10
11   constructor () {
12     console.log('동작');
13   }
14
15   public clickAfterPrint () : void {
16     console.log('출력');
17   }
18 }
```

문제 디버그 콘솔 터미널

Date: 2020-05-11T00:33:22.196Z - Hash: 687a423

5 unchanged chunks

아까 우리가 button 태그로 변경한 html 파일과 이름이 같습니다!

Component 데코레이터가 존재하는 파일에는 화면에서 보여줄 html 파일의 위치를 지정 할 수 있습니다.

이와 같이 화면구성에 사용할 html 파일이 무엇인지, css는 무엇인지, 또는 사용되어질 데이터는 무엇인지 등등 다양한 내용등을 관리 하도록 기능을 부여한 것이 바로 컴포넌트입니다.

여기까지! 컴포넌트에 대해서 간단하게 알아보았습니다.

내용이 역시 쉽지가 않습니다..ㅠ

그래도 이번장에서 반드시 이해해야될 점은, "컴포넌트의 역할" 입니다.

화면구성과 관련된 내용은 컴포넌트라는점을 꼭 이해하

여야 합니다.

다음시간에는 접근제어자, 타입에 대한 설명을 컴포넌트를 통해서 이어나가보도록 하겠습니다.

궁금한점, 이해가 되지 않는부분은 언제든지 연락주세요!

* 컴포넌트 파일(app.component.ts)

```
import { Component } from
```

```
'@angular/core';
```

```
@Component ({
```

```
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
```



```
})
```

```
export class AppComponent {
```

```
  title = 'firstStudy';
```

```
  constructor () {
```

```
    console.log('동작');
```

```
  }
```

```
  public clickAfterPrint () : void{
```

```
    console.log('출력');
```

```
  }
```

```
}
```

* Html 파일(app.component.html)

```
<button
```

```
  (click)='clickAfterPrint()'>{{title}} <-
```

```
당신이 입력한 변수 title </button>
```

* 추가

앵귤러, 리엑트, 뷰 js 등 프론트 프레임워크(라이브러리)는 대체적으로 버전이 자주 올라갑니다.

이에 따라 기존에 사용되었던 기능이 버전업에 따라 동작을 하지 않을수도 있습니다.

만약 동작을 원활하게 되지 않는다면 아래

package.json 내용에서의 의존성(dependencies) 버전을 맞추어 주세요.

```
"dependencies": {
```

```
  "@angular/animations": "~9.1.4",
```

```
  "@angular/common": "~9.1.4",
```

```
  "@angular/compiler": "~9.1.4",
```

```
  "@angular/core": "~9.1.4",
```

```
  "@angular/forms": "~9.1.4",
```

```
"@angular/platform-browser":  
  "~9.1.4",  
  "@angular/platform-browser-dynamic":  
    "~9.1.4",  
  "@angular/router": "~9.1.4",  
  "rxjs": "~6.5.4",  
  "tslib": "^1.10.0",  
  "zone.js": "~0.10.2"  
},  
"devDependencies": {  
  "@angular-devkit/build-angular":  
    "~0.901.4",  
  "@angular/cli": "~9.1.4",  
  "@angular/compiler-cli": "~9.1.4",  
  "@angular/language-service":  
    "~9.1.4",  
  "@types/node": "^12.11.1",  
  "@types/jasmine": "~3.5.0",  
  "@types/jasminewd2": "~2.0.3",  
  "codelyzer": "^5.1.2",
```

```
"jasmine-core": "~3.5.0",  
"jasmine-spec-reporter": "~4.2.1",  
"karma": "~5.0.0",  
"karma-chrome-launcher": "~3.1.0",  
"karma-coverage-istanbul-reporter":  
"~2.1.0",  
"karma-jasmine": "~3.0.1",  
"karma-jasmine-html-reporter":  
"^1.4.2",  
"protractor": "~5.4.3",  
"ts-node": "~8.3.0",  
"tslint": "~6.1.0",  
"typescript": "~3.8.3"  
}
```

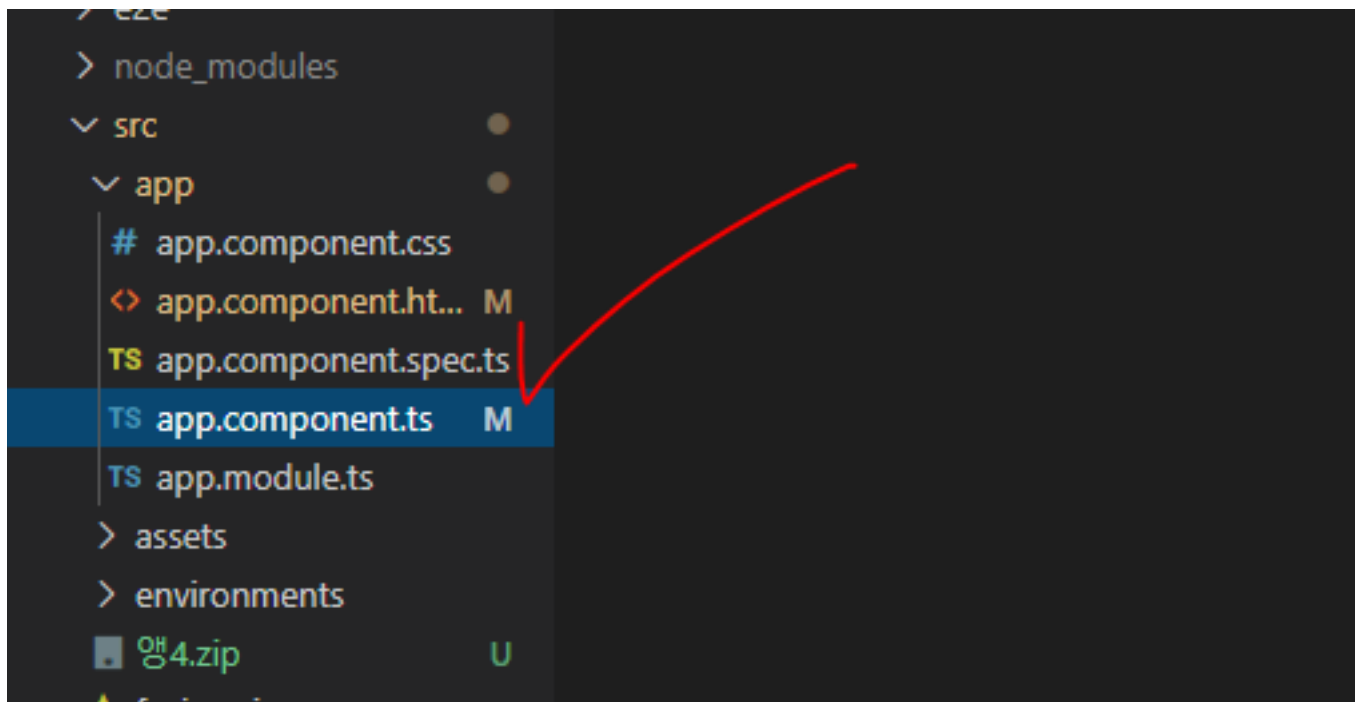
이번시간까지는 앵귤러에서의 컴포넌트의 역할에 대해서 간단하게 살펴 보았습니다.

컴포넌트는 앵귤러에서 화면구성, 이벤트에 대한 행위를 담당 합니다.

우리가 html 파일에서 여러 tag 로 만든 화면, <script> 테 그 안에 선언한 다양한 이벤트, 변수 등에 대한 행위 이러한 사용자 ui 와 직접적으로 동작하는 것이 바로 컴포넌트 입니다.

그러면 컴포넌트에 대해 몇가지 예제를 통해서 좀 더 알아보겠습니다.

firstStudy 프로젝트로 이동한 뒤에 src - app 으로 이동합니다.



경로 헛갈리지마세요!

그리고 **app.component.ts** 파일을 더블클릭합니다.

배열과 함수를 추가하여보았습니다.

아래 내용으로 수정하여주세요.

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

```
const array : Array<string> =  
['data0','data1','data2']
```

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

```
export class AppComponent {  
  title : string = 'firstStudy';
```

```
  constructor () {
```

```
  }
```

```
  public clickAfterPrint () : void{
```

```
    this._innerFunc();
```

```
    console.log(array);
```

```
    console.log(this.title);
```

```
  }
```

```
  private _innerFunc() {
```

```
array.push('data' + array.length);
```

```
}
```

```
}
```

app.component.html 파일 내용을 아래처럼 바꾸어주세요.

(html 파일입니다!!)

기존내용을 전부 지우거나 주석처리해주세요.

```
<button
```

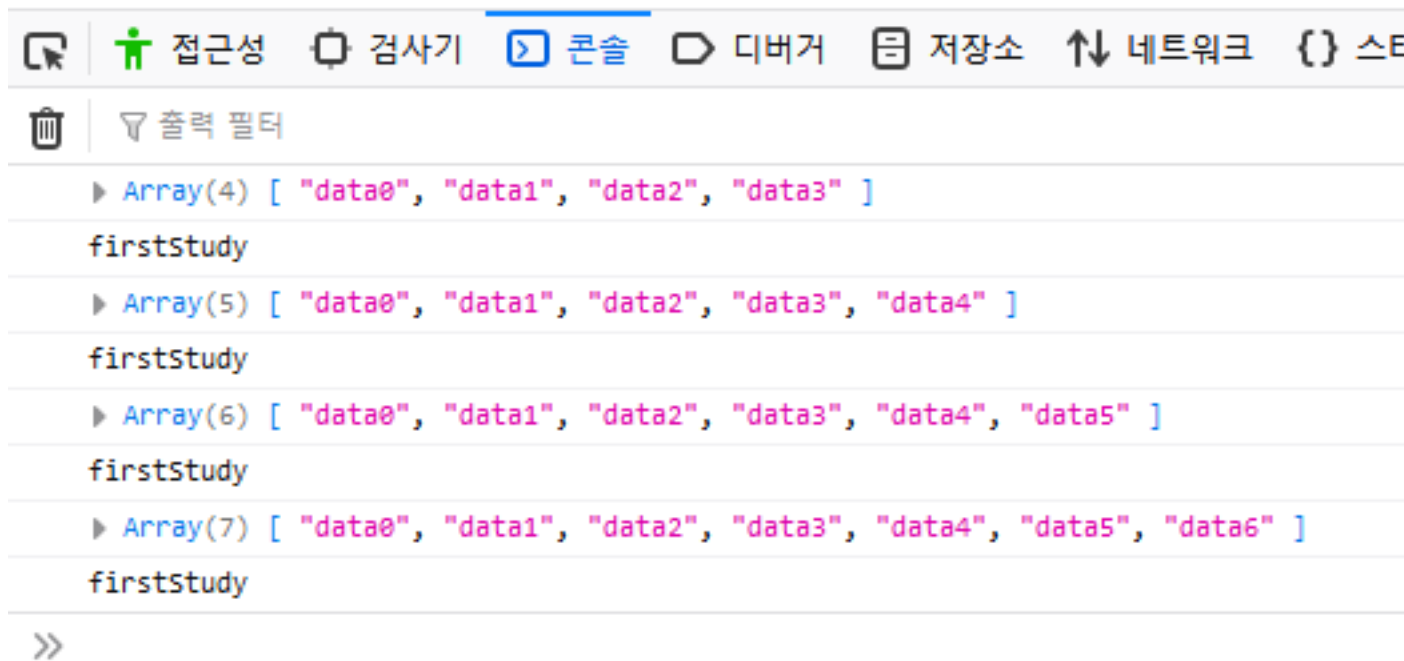
```
(click)='clickAfterPrint()'>{{title}} <-
```

```
당신이 입력한 변수 title </button>
```

그런 다음 아래처럼 작업용 서버를 구동합니다.

ng serve

버튼을 누를 때 마다 배열 array 의 값이 추가되어 개발자
도구에 출력되는 것을 볼 수 있습니다.



데이터가 계속해서 추가되어 출력됩니다!

내용을 천천히 살펴보겠습니다.

파일 이름은 `app.component.ts` 이며, 클래스 명칭은 `AppComponent` 클래스 입니다.

컴포넌트 데코레이터(`@Component`)가 사용되었기 때문에 해당 파일은 컴포넌트입니다.

따라서 해당파일은 아래처럼 부를 수 있습니다.

1. AppComponent 클래스

2. AppComponent 컴포넌트

통상적으로 첫번째 보다는 두번째 경우로 불려지는 것 같습니다.

*** 이것은 어디까지나 주관적인 입장입니다!!**

다음으로 `array` 라는 변수를 살펴보겠습니다.

`array` 라는 변수는 `class` 바깥에 선언한 변수로 해당 컴포넌

트 파일에서 글로벌하게 사용되는 변수 입니다.

array 라는 변수는 class 에 속하지 않기 때문에 class 안에
서 사용하는 변수 및 함수처럼 this 를 붙여서 사용하지 않
습니다.

* 클래스 안에 사용하는 함수 또는 클래스변수는 this 를 사
용하지만, 클래스 외부에서 선언한 변수 및 함수는 this
를 사용하지 않습니다.

```

2
3  const array : Array<string> = ['data0','data1','data2']
4
5  @Component({
6      selector: 'app-root',
7      templateUrl: './app.component.html',
8      styleUrls: ['./app.component.css']
9  })
10 export class AppComponent {
11     title : string = 'firstStudy';
12
13     constructor () {
14
15     }
16

```

꺅은 괄호<>에는 데이터 형태를 지정 해 줄수 있습니다.

array 변수 뒤에 Array<string> 라는 기호가 붙여있습니다.

자바스크립트와 달리 타입스크립트에서는 데이터의 형태를 지정 해 줄 수 있습니다.

변수에 대입해야 될 데이터 형태를 지정하므로써 개발자의 형 변환과 같은 실수를 사전에 방지 할 수 있습니다.

사용법은 매우 직관적이며, 콜론(:)뒤에 지정하고자 하는 자료형을 선언하여 주면 됩니다.

`const array : Array<string>` 의 의미는 변수명은 `array` 이면서 `array` 가 가질 수 있는 자료형태는 문자(`string`)로 이루어진 `Array`(배열)을 의미합니다.

꺾은 괄호<>는 제네릭을 의미하며, Java 에서의 기능과 동일합니다.

제네릭을 간단하게 설명하면 `Array` 에서 올 수 있는 데이터의 형태를 의미합니다.

`array` 에 단순한 숫자, 날짜 형태의 데이터를 `push` 하여보세요. 오류가 발생할 것 입니다.

* 타입스크립트를 가볍게라도 한번 꼭 살펴보셔야 합니다.

다음으로 살펴볼 클래스 안에서 만들어진 함수 `_innerFunc` 입니다.

앞에 `private` 라는 기호가 붙어 있습니다.

private 라는 기호는 "내부" 의 의미로 해당 함수는 다른 곳에서 사용 할 수 없는 함수를 의미합니다.

```
9      })
10     export class AppComponent {
11         title : string = 'firstStudy';
12
13         constructor () {
14
15         }
16
17         public clickAfterPrint () : void {
18             this._innerFunc();
19             console.log(array);
20             console.log(this.title);
21         }
22
23         private _innerFunc(){
24             array.push('data' + array.length);
25         }
26     }
```

private, public 이보입니다. 아무것도 없으면 protected 입니다!

private 와 protected, public 3 가지의 형태의 접근 제어자를 타입스크립트는 제공합니다.

다른 곳에서 사용 못하고 오직 해당 클래스에서만 사용하

게 하려면 `private`,

해당 클래스와 직접적으로 관계된 곳에서만 사용하게 하려면 `protected`,

누구든지 사용 가능하게 하려면 `public` 을 붙여주면 됩니다.

앵귤러 기반에서의 클래스의 접근제어 기본 값은 **`protected`** 입니다.

앵귤러에서는 아무것도 명시하지 않으면 암묵적으로 `protected` 로 지정되어 패키지 레벨로 공개됩니다.

만약 순수 **`Typescript`** 로만 이루어진 프로젝트인 경우라면 접근제어자 기본은 `public` 으로 선언이 됩니다.

여기서 `_innerFunc` 함수는 다른 곳에서 사용할 수 없는 함

수이며 오직 해당 파일에서만 사용 할 수 있습니다.

`_innerFunc` 함수는 `this` 를 붙여서 `clickAfterPrint` 함수에
서 사용되었습니다.

`clickAfterPrint` 뒤에 사용된 `void` 라는 형태는 값이 없음
을 의미합니다.

```
}  
  
public clickAfterPrint () : void{  
    this._innerFunc();  
    console.log(array);  
    console.log(this.title);  
}
```

콜론뒤의
내용은 반환 값 입니다. `void` 는 "없다" 입니다.

타입스크립트에서는 함수의 반환 값도 형태를 지
정 할 수 있습니다.

변수에 데이터 형태를 지정한 것과 동일하게 함수의 소괄

호 뒤에 콜론(:)을 사용한 뒤에 반환값을 지정 해 주면 됩니다.

여기까지 간단하게 컴포넌트의 내부에 쓰여진 코드를 살펴 보았습니다.

조금 더 이해를 돕기위해 다시 코드를 조금 변경하여 보겠습니다.

app.component.ts (AppComponent 컴포넌트) 파일을 아래와 같이 수정하여봅니다.

```
import { Component } from
```

```
'@angular/core';
```

```
const array : Array<string> =
```

```
['data0', 'data1', 'data2']
```

```
@Component ({
```

```
selector: 'app-root',
```

```
templateUrl: './app.component.html',
```

```
styleUrls: ['./app.component.css']
```

```
})
```

```
export class AppComponent {
```

```
  title : string = 'firstStudy';
```

```
  showArray : Array<string>; //아직 초기화
```

되지 않았으며 protected 한 변수 입니다.

```
  constructor () { //생성자는 오직 1 번만 클
```

래스파일이 생성 될 때 동작 합니다.

```
    this.showArray = array; //외부 array
```

의 변수 값을 showArray 게 참조하게 하였습니다.

```
  }
```

```
  public clickAfterPrint () : void{
```

```
    this._innerFunc();
```

```
    console.log(array);
```

```
console.log(this.title);
```

```
}
```

```
private _innerFunc() {
```

```
array.push('data' + array.length);
```

```
//배열값을 변화시킵니다.
```

```
}
```

```
}
```

이어서 app.component.html 파일을 아래처럼 바꾸어 봅
니다. (html 파일입니다!!)

```
<button
```

```
(click)='clickAfterPrint()'>{{title}} <-
```

```
당신이 입력한 변수 title </button>
```

```
<div *ngFor="let item of
```

```
showArray">{{item}}</div>
```

clickAfterPrint 함수 실행을 위해 버튼을 눌러봅니다!

```
firstStudy <- 당신이 입력한 변수 title
```

```
data0  
data1  
data2  
data3  
data4  
data5
```

접근성 | 🔍 검사기 | **콘솔** | 디버거 | 저장소 | 네트워크 | 스타일

출력 필터

```
▶ Array(4) [ "data0", "data1", "data2", "data3" ]
```

```
firstStudy
```

```
▶ Array(5) [ "data0", "data1", "data2", "data3", "data4" ]
```

```
firstStudy
```

```
▶ Array(6) [ "data0", "data1", "data2", "data3", "data4", "data5" ]
```

```
firstStudy
```

```
>>
```

누를때 마다 html 파일의 내용도 같이 변합니다.

외부에 선언한 array 값이 클래스 변수인 showArray 가 참조하게 하였습니다.

click 이벤트를 통해 clickAfterPrint 가 호출되면

서 _innerFunc 함수가 array 변수에 데이터를 push 하게 하였습니다.

array 를 참조하는 showArray 값이 변화를 할 때마다 동기화게도 화면의 데이터가 늘어나는 것을 볼 수 있습니다.

app.component.ts 파일의 내용이 지금 우리는 무엇인지 알 필요는 없습니다.

다만 컴포넌트 내부의 데이터가 변화하였을 뿐 인데 html 파일에서의 내용이 변경되었다는 점 입니다.

만약에 `clickAfterPrint` 함수가 마치 `ajax` 처럼 특정 주소에서 데이터를 등록, 수정, 삭제 또는 가져오는 기능이였다면 어땠을까요?

이처럼 컴포넌트에서는 직접적으로 화면과 관련된 일을 수행합니다.

데이터의 변화, 이벤트 등등 이러한 일을 컴포넌트에서 수행하며 처리합니다.

여기까지 앵귤러의 컴포넌트에 대해서 알아보았습니다.

타입스크립트를 어느정도 익혀야 앵귤러에 조금 더 쉽게 접근 할 수 있습니다.

사실, 타입스크립트는 앵귤러에서만 사용되는 것이 아니므로 익혀둘 경우 다양하게 사용 할 수 있습니다.

* 추가

앵귤러, 리엑트, 뷰js 등 프론트 프레임워크(라이브러리)는 대체적으로 버전이 자주 올라갑니다.

이에 따라 기존에 사용되었던 기능이 버전업에 따라 동작을 하지 않을수도 있습니다.

만약 동작을 원활하게 되지 않는다면 아래 `package.json` 내용에서의 의존성(dependencies) 버전을 맞추어 주세요.

```
"dependencies": {
```

```
  "@angular/animations": "~9.1.4",
```

```
  "@angular/common": "~9.1.4",
```

```
  "@angular/compiler": "~9.1.4",
```

```
  "@angular/core": "~9.1.4",
```

```
  "@angular/forms": "~9.1.4",
```

```
  "@angular/platform-browser":
```

```
    "~9.1.4",
```



```
"@angular/platform-browser-dynamic":
  "~9.1.4",
  "@angular/router": "~9.1.4",
  "rxjs": "~6.5.4",
  "tslib": "^1.10.0",
  "zone.js": "~0.10.2"
},
"devDependencies": {
  "@angular-devkit/build-angular":
    "~0.901.4",
  "@angular/cli": "~9.1.4",
  "@angular/compiler-cli": "~9.1.4",
  "@angular/language-service":
    "~9.1.4",
  "@types/node": "^12.11.1",
  "@types/jasmine": "~3.5.0",
  "@types/jasminewd2": "~2.0.3",
  "codelyzer": "^5.1.2",
  "jasmine-core": "~3.5.0",
  "jasmine-spec-reporter": "~4.2.1",
```

```
"karma": "~5.0.0",
```

```
"karma-chrome-launcher": "~3.1.0",
```

```
"karma-coverage-istanbul-reporter":
```

```
"~2.1.0",
```

```
"karma-jasmine": "~3.0.1",
```

```
"karma-jasmine-html-reporter":
```

```
"^1.4.2",
```

```
"protractor": "~5.4.3",
```

```
"ts-node": "~8.3.0",
```

```
"tslint": "~6.1.0",
```

```
"typescript": "~3.8.3"
```

```
}
```