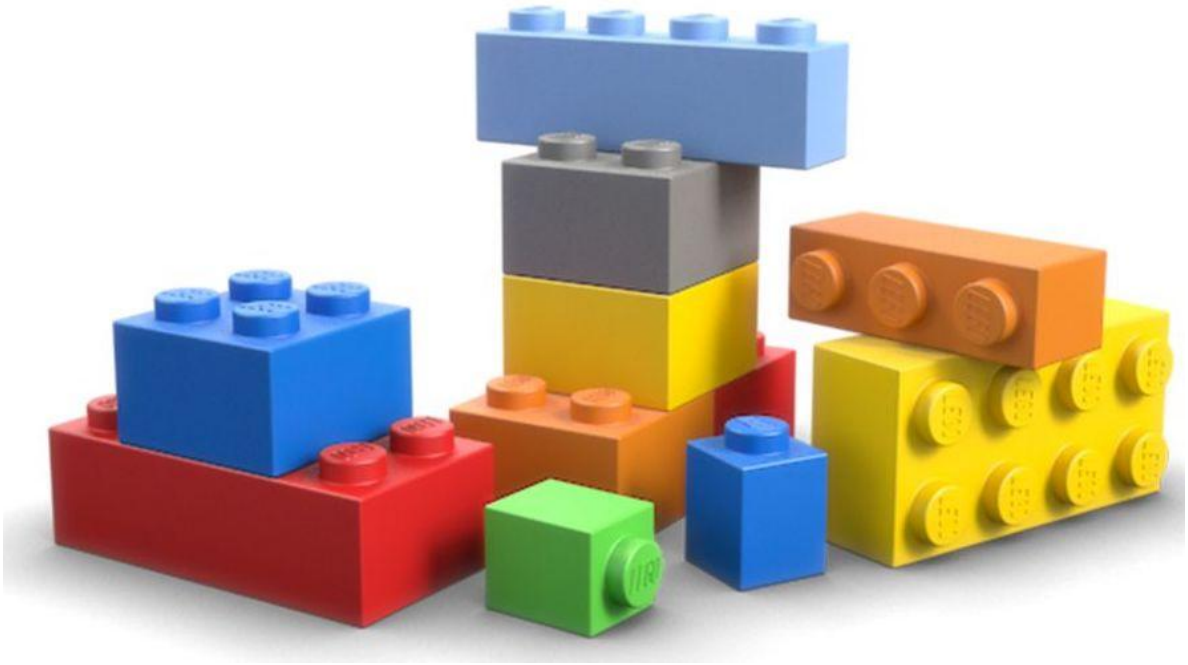
PoiemaWeb

13.23 Angular NgModule

모듈

#1. 모듈(NgModule)이란?

Angular의 모듈(NgModule)은 관련이 있는 구성요소(컴포넌트, 디렉티브, 파이프, 서비스 등)를 하나의 단위로 묶는 메커니즘을 말한다. 다시 말해 모듈은 관련이 있는 구성요소들로 구성된 응집된 기능 블록으로 애플리케이션을 구성하는 하나의 단위를 말한다. 모듈은 다른 모듈과 결합할 수 있으며 Angular는 여러 모듈들을 조합하여 애플리케이션을 구성한다.



Angular 애플리케이션은 모듈들의 조합이다.

Angular 애플리케이션은 하나의 모듈로 구성할 수도 있고 여러 개의 모듈로 구성할 수도 있다. 하지만 애플리케이션은 적어도 하나의 모듈(루트 모듈)을 소유하여야 한다. 즉, 모든 애플리케이션은 루트 모듈을 가져야 하고 이 루트 모듈은 애플리케이션의 최상위에 존재하며 애플리케이션의 시작점이 된다. 아래는 루트 모듈의 예이다.

TYPESCRIPT

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

import { AppComponent } from './app.component';
import { SomeDirective } from './some.directive';
import { SomeComponent } from './some/some.component';
import { SomePipe } from './some.pipe';
import { SomeService } from './some.service';

@NgModule({
  declarations: [
    AppComponent,
    SomeDirective,
    SomeComponent,
    SomePipe
  ],
  imports: [BrowserModule],
  providers: [SomeService],
```

```
  bootstrap: [AppComponent]
})
export class AppModule { }
```

NgModule은 @NgModule 데코레이터로 장식된 클래스이다. @NgModule는 메타데이터 객체를 인자로 전달받아서 Angular에 모듈 코드를 어떻게 컴파일하면 되는지, 또 어떻게 실행하면 되는지를 Angular에게 설명한다.

모듈은 다른 모듈을 import할 수 있다. Angular에서 제공하는 라이브러리 모듈(BrowserModule, FormsModule, HttpClientModule, RouterModule 등) 또는 서드 파티 라이브러리(Angular Material, Ionic, AngularFire2 등)도 import하여 사용할 수 있다.

애플리케이션 개발에 있어서 모듈성(Modularity)은 중요한 의미를 갖는다. 간단한 애플리케이션이라면 하나의 모듈, 즉 루트 모듈만으로 애플리케이션을 구성하여도 문제가 없으나 애플리케이션에 대한 요구사항이 많아지면서 코드의 복잡도가 커짐에 따라 루트 모듈(Root module), 기능 모듈(Feature module), 공유 모듈(Shared module), 핵심 모듈(Core module)로 모듈을 분리하여 애플리케이션을 구성한다. 이것은 모듈 간의 결합을 최소화하고 모듈의 응집성을 극대화한 애플리케이션, 즉 모듈성을 갖춘 애플리케이션을 개발하기 위한 바람직한 어프로치이다.

2. @NgModule 데코레이터

모듈은 @NgModule 데코레이터로 장식된 클래스이다. @NgModule 데코레이터는 함수이며 모듈의 설정 정보가 기술된 메타데이터 객체를 인자로 전달받아 모듈을 생성한다. 메타데이터는 아래와 같다.

TYPESCRIPT

```
@NgModule({
  providers?: Provider[]
  declarations?: Array<Type<any>|any[]>
  imports?: Array<Type<any>|ModuleWithProviders|any[]>
  exports?: Array<Type<any>|any[]>
  entryComponents?: Array<Type<any>|any[]>
  bootstrap?: Array<Type<any>|any[]>
  schemas?: Array<SchemaMetadata|any[]>
  id?: string
})
```

메타데이터 객체의 중요한 프로퍼티에 대해 살펴보도록 하자.

프로퍼티	내용
providers	주입 가능한 객체(injectable object) 즉 서비스의 리스트를 선언한다. 루트 모듈에 선언된 서비스는 애플리케이션 전역에서 사용할 수 있다.
declarations	컴포넌트, 디렉티브, 파이프의 리스트를 선언한다. 모듈에 선언된 구성 요소는 모듈에서 사용할 수 있다.
imports	의존 관계에 있는 Angular 라이브러리 모듈, 기능 모듈(Feature module)이라 불리는 하위 모듈, 라우팅 모듈, 서드 파티 모듈을 선언한다.
bootstrap	루트 모듈에서 사용하는 프로퍼티로서 애플리케이션의 진입점(entry point)인 루트 컴포넌트를 선언한다.

3. 라이브러리 모듈

라이브러리 모듈은 Angular가 제공하는 빌트인 모듈이다. Angular CLI를 통해 생성된 프로젝트의 package.json을 살펴보면 @angular 라이브러리 모듈 패키지가 포함되어 있음을 확인할 수 있다.

JSON

```
...
  "dependencies": {
    "@angular/animations": "^6.0.0",
    "@angular/common": "^6.0.0",
    "@angular/compiler": "^6.0.0",
    "@angular/core": "^6.0.0",
    "@angular/forms": "^6.0.0",
    "@angular/http": "^6.0.0",
    "@angular/platform-browser": "^6.0.0",
    "@angular/platform-browser-dynamic": "^6.0.0",
    "@angular/router": "^6.0.0",
    "core-js": "^2.5.4",
    "rxjs": "^6.0.0",
    "zone.js": "^0.8.26"
  },
...
```

Angular의 라이브러리 모듈 패키지는 모듈의 집합체이다. 따라서 라이브러리 모듈 패키지에서 필요한 모듈만을 선택하여 임포트한다. 예를 들어 @angular/platform-browser 패키지에서 BrowserModule 모듈을 임포트하는 경우, 아래와 같이 기술한다.

TYPESCRIPT

```
// app.module.ts
import { BrowserModule } from '@angular/platform-browser';
```

BrowserModule은 브라우저 환경에서 동작하는 애플리케이션을 위한 필수 기능을 제공하는 모듈로서 브라우저에서 동작하는 웹 애플리케이션의 경우, 반드시 BrowserModule을 임포트하여야 한다. BrowserModule은 NgIf 및 NgFor와 같은 빌트인 디렉티브와 빌트인 파이프를 제공하는 CommonModule을 내부에서 import한다. 따라서 BrowserModule을 import하면 별도의 추가적인 import없이 CommonModule을 사용할 수 있게 되어 모든 애플리케이션의 컴포넌트 템플릿에서 빌트인 디렉티브와 빌트인 파이프를 사용할 수 있다.

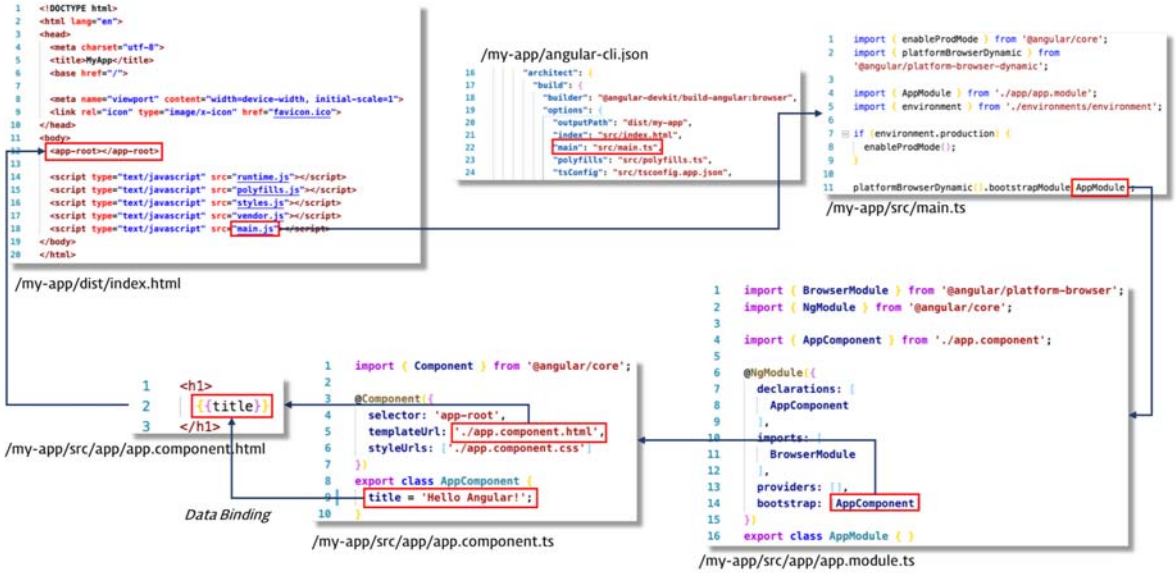
만약 ngModel 디렉티브를 사용하려면 FormsModule을, HttpClient 서비스를 사용하려면 HttpClientModule을 import할 필요가 있다.

4. 루트 모듈

Angular 애플리케이션은 적어도 하나 이상의 모듈을 소유하여야 한다. 루트 모듈은 애플리케이션의 최상위에 존재하는 유일한 모듈로 애플리케이션 레벨의 컴포넌트, 디렉티브, 파이프, 서비스를 선언하거나 의존 라이브러리 모듈과 기능 모듈(Feature module)이라 불리는 하위 모듈을 포함(import)할 수 있다.

웹 애플리케이션의 경우, 루트 모듈은 반드시 BrowserModule을 임포트하여야 한다. 루트 모듈을 제외한 다른 모듈은 CommonModule을 임포트하여야 한다.

모든 애플리케이션은 루트 모듈을 가져야 하고 이 루트 모듈은 애플리케이션의 시작점이 된다. 즉, Angular 애플리케이션은 모듈 단위로 동작하여 루트 모듈이 부트스트랩 되는 것에 의해 애플리케이션이 동작하게 된다.



Angular 애플리케이션의 흐름

루트 모듈은 일반적으로 AppModule이라는 이름으로 생성하며 Angular CLI를 통해 프로젝트를 생성하면 src/app/app.module.ts 파일 내에 생성된다.

TYPESCRIPT

```
// src/app/app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

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

@NgModule 데코레이터는 함수이며 모듈에 대한 설정 정보를 담고 있는 메타 데이터를 인자로 전달받는다. 이 설정 정보를 바탕으로 AppModule 클래스는 모듈을 생성한다. 메타 데이터는 declarations, imports, providers, bootstrap 등의 프로퍼티로 구성된다.

#5. 모듈의 분리

애플리케이션이 커짐에 따라 루트 모듈에 등록된 컴포넌트, 디렉티브, 파이프, 서비스도 늘어나게 된다. 이때 기능적으로 관련도가 떨어지는 구성요소가 하나의 모듈에 혼재되어 있으면 관리가 어려워지고 구성 요소의 이름이 중복되어 충돌할 가능성 또한 커진다. Angular는 기능 모듈, 핵심 모듈, 공유 모듈로 모듈을 분리한다.

모듈	개요	대상
기능 모듈	관심사가 유사한 구성 요소로 구성한 모듈	특정 화면을 구성하는 구성 요소
공유 모듈	애플리케이션 전역에서 공유할 구성 요소들로 구성한 모듈로서 기능 모듈에 의해 임포트된다.	애플리케이션 전역에서 사용하는 컴포넌트, 디렉티브, 파이프 등
핵심 모듈	애플리케이션 전역에서 공통 사용할 구성 요소들로 구성한 모듈로서 루트 모듈에 등록하여 싱글턴으로 사용한다.	애플리케이션 전역에서 사용하는 데이터 서비스, 인증 서비스, 인증 가드 등

사용자의 정보를 표시하는 간단한 애플리케이션을 통해 모듈을 분리해보자.

BASH

```
$ ng new module-exam -t -s -S
$ cd module-exam
$ ng generate component header
$ ng generate component home
$ ng generate service user
$ ng generate interface user
```

생성된 애플리케이션의 src/app의 파일 구성은 아래와 같다.

CODE

```
src
└── app
    ├── header
    │   └── header.component.ts
```

```
        ├── home
        │   └── home.component.ts
        ├── app.component.ts
        ├── app.module.ts
        ├── user.service.ts
        └── user.ts
```

header.component.ts는 애플리케이션의 헤더에 타이틀과 사용자 이름을 나타내기 위한 컴포넌트로서 모든 뷰에 공통으로 사용한다. header.component.ts를 아래와 같이 작성한다.

TYPESCRIPT

```
// header/header.component.ts
import { Component, OnInit, Input } from '@angular/core';

import { UserService } from '../user.service';
import { User } from '../user';

@Component({
  selector: 'app-header',
  template: `
    <nav>
      <span>{{title}}</span>
      <a class="user" href="#">{{user.name}}</a>
    </nav>
  `,
  styles: [`
    nav {
      background-color: #4a4c88;
      overflow: hidden;
    }
    .title, .user {
      line-height: 50px;
      margin: 0 30px;
      color: #fff;
      text-decoration: none;
      font-weight: bold;
      text-transform: uppercase;
      opacity: 0.7;
    }
```

```
      .title {
        float: left;
      }
      .user {
        float: right;
        font-style: italic;
      }
    `]
})
export class HeaderComponent implements OnInit {
  @Input() title: string;
  user: User;

  constructor(private userService: UserService) {}

  ngOnInit() {
    this.user = this.userService.getUser();
  }
}
```

home.component.ts는 home 페이지를 위한 컴포넌트로서 사용자의 정보를 표시하며 header.component.ts를 사용한다. home.component.ts를 아래와 같이 작성한다.

TYPESCRIPT

```
// home/home.component.ts
import { Component, OnInit } from '@angular/core';

import { UserService } from '../user.service';
import { User } from '../user';

@Component({
  selector: 'app-home',
  template: `
    <app-header [title]="title"></app-header>
    <ul>
      <li>id : {{ user.id }}</li>
      <li>name : {{ user.name }}</li>
      <li>admin : {{ user.admin }}</li>
    </ul>
```

```
  `
})
export class HomeComponent implements OnInit {
  title = 'User Information';
  user: User;

  constructor(private userService: UserService) {}

  ngOnInit() {
    this.user = this.userService.getUser();
  }
}
```

user.service.ts는 사용자 정보를 제공하는 서비스로서 애플리케이션 전역에서 공통으로 사용한다. user.service.ts를 아래와 같이 작성한다.

TYPESCRIPT

```
// user.service.ts
import { Injectable } from '@angular/core';
import { User } from './user';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  getUser(): User {
    // 임의의 사용자를 반환한다. 실제 환경에서는 서버의 데이터를 취득하여 반환할 것이다.
    return { id: 1, name: 'Lee', admin: true };
  }
}
```

user.ts는 사용자 정보를 나타내는 User 타입 인터페이스이다. user.ts를 아래와 같이 작성한다.

TYPESCRIPT

```
// user.ts
export interface User {
  id: number;
```

```
  name: string;
  admin: boolean;
}
```

루트 컴포넌트를 아래와 같이 작성한다.

TYPESCRIPT

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

@Component({
  selector: 'app-root',
  template: '<app-home></app-home>'
})
export class AppComponent {}
```

루트 모듈을 아래와 같이 작성한다.

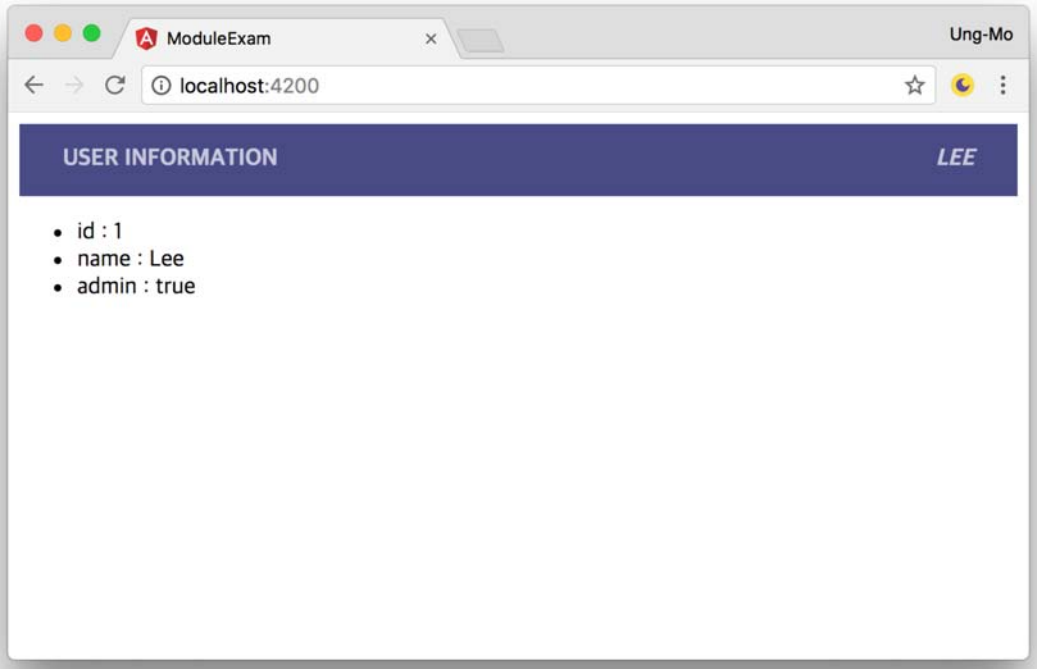
TYPESCRIPT

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

import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { HomeComponent } from './home/home.component';

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

위 예제의 실행 결과는 아래와 같다.



위 예제의 구성 요소를 기능 모듈, 핵심 모듈, 공유 모듈로 분리해보고 그 분리 기준에 대해 살펴보도록 하자.

5.1 기능 모듈(Feature module)

루트 모듈에 여러 기능이 혼재되면 관리가 어려워지고 분업 또한 곤란해진다. 따라서 관심사가 유사한 구성 요소들을 그룹화한 하위 모듈을 생성할 필요가 있다. 이렇게 생성된 하위 모듈을 상위 모듈은 임포트하여 사용한다. 이때 상위 모듈은 하위 모듈이 외부로 공개한 구성 요소를 사용할 수 있다.

기능 모듈은 관심사가 유사한 구성 요소로 구성한 모듈이다. **일반적으로 기능 모듈은 특정 화면 단위를 기준으로 구성한다.** 기능 모듈은 루트 모듈과 마찬가지로 @NgModule 데코레이터와 메터데이터로 구성한다.

위의 예제에서 home.component.ts는 home 페이지를 위한 컴포넌트로서 사용자의 정보를 표시한다. 이 컴포넌트는 특정 화면을 담당하므로 기능 모듈로 분리할 수 있다. 기능 모듈인 home 모듈을 작성해 보자.

BASH

```
$ ng generate module home
```

위 명령어를 실행하면 home 폴더에 home.module.ts이 생성된다. 생성된 home.module.ts는 아래와 같다.

TYPESCRIPT

```
// home/home.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [CommonModule],
  declarations: []
})
export class HomeModule { }
```

생성된 HomeModule은 루트 모듈이 아니므로 CommonModule을 임포트하여야 한다. Angular CLI을 사용하여 모듈을 생성하면 CommonModule이 자동 등록된다.

이제 HomeModule에 HomeComponent를 등록하고 HomeComponent를 외부로 공개하자.

TYPESCRIPT

```
// home/home.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

/* HomeComponent 임포트 */
import { HomeComponent } from './home.component';

@NgModule({
  imports: [CommonModule],
  declarations: [HomeComponent], /* HomeComponent 선언 */
  providers: [],
  exports: [HomeComponent] /* HomeComponent 공개 */
})
export class HomeModule { }
```

HomeModule이 완성되었다. 이제 루트 모듈에 HomeModule을 등록하도록 하자.

TYPESCRIPT

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

/* HomeModule 임포트 */
import { HomeModule } from './home/home.module';

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

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent
  ],
  imports: [
    BrowserModule,
    HomeModule /* HomeModule 임포트 */
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

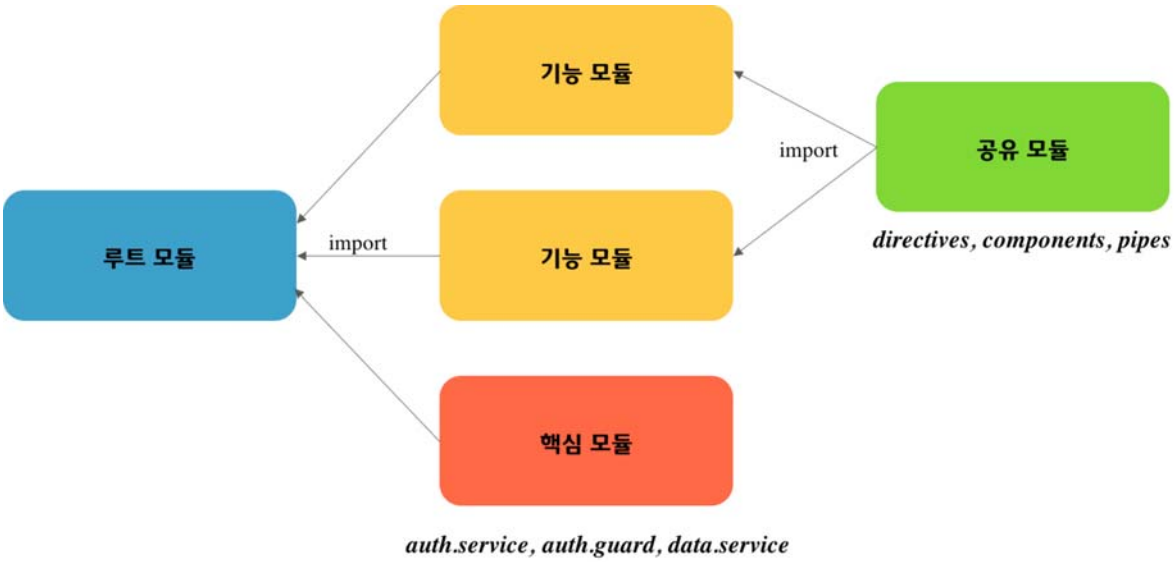
HomeComponent는 HomeModule에 등록되었으므로 더이상 루트 모듈의 관리 대상이 아니다. 따라서 루트 모듈의 declarations 프로퍼티에 선언되어 있던 HomeComponent를 제외시킨다. 그리고 HomeModule을 루트 모듈의 imports 프로퍼티에 선언한다.

5.2 공유 모듈(Shared module)

공유 모듈은 애플리케이션 전역에서 공유할 구성 요소들로 구성한 모듈로서 다른 모듈(주로 기능 모듈)에서 공통적으로 사용된다. 예를 들어 애플리케이션 전역에서 사용하는 컴포넌트, 디렉티브, 파이프 등이 대

상이 된다.

루트 모듈은 기능 모듈을 임포트하고 기능 모듈은 공유 모듈을 임포트하여 사용한다. 이렇게 모듈을 구성하여 기능 모듈의 중복을 제거하여 모듈 선언을 간소화한다. 다시 말해 공유 모듈은 루트 모듈에 직접 임포트되지 않고 기능 모듈에 의해 임포트되어 사용된다.



공유 모듈

위의 예제에서 header.component.ts는 해당 뷰의 타이틀과 사용자 이름을 나타내는 애플리케이션의 헤더를 위한 컴포넌트이다. 이 컴포넌트는 화면 단위의 기능 모듈에 의해 공통적으로 사용된다. 따라서 이 컴포넌트는 공유 모듈로 분리할 수 있다. 공유 모듈인 shared 모듈을 작성해 보자.

BASH

```
$ ng generate module shared
```

위 명령어를 실행하면 shared 폴더에 shared.module.ts이 생성된다. 생성된 SharedModule은 루트 모듈이 아니므로 CommonModule을 임포트하여야 한다. Angular CLI을 사용하여 모듈을 생성하면 CommonModule이 자동 등록된다. 생성된 shared.module.ts는 아래와 같다.

TYPESCRIPT

```
// shared/shared.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [CommonModule],
  declarations: []
```

```
})
export class SharedModule { }
```

먼저 공유 모듈의 구성 요소인 header.component.ts 파일을 shared 폴더로 이동시킨다. 그리고 SharedModule에 HeaderComponent를 등록하고 HeaderComponent를 외부로 공개하자.

TYPESCRIPT

```
// shared/shared.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

/* HeaderComponent 임포트 */
import { HeaderComponent } from './header.component';

@NgModule({
  imports: [CommonModule],
  declarations: [HeaderComponent], /* HeaderComponent 선언 */
  providers: [],
  exports: [HeaderComponent] /* HeaderComponent 공개 */
})
export class SharedModule { }
```

SharedModule이 완성되었다. 공유 모듈은 기능 모듈에 의해 사용되므로 기능 모듈인 HomeModule에 SharedModule을 등록하도록 하자.

TYPESCRIPT

```
// home/home.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

/* SharedModule 임포트 */
import { SharedModule } from '../shared/shared.module';

/* HomeComponent 임포트 */
import { HomeComponent } from './home.component';

@NgModule({
```

```
  imports: [
    CommonModule,
    SharedModule, /* SharedModule 임포트 */
  ],
  declarations: [HomeComponent], /* HomeComponent 선언 */
  providers: [],
  exports: [HomeComponent] /* HomeComponent 공개 */
})
export class HomeModule { }
```

이제 HeaderComponent는 공유 모듈인 SharedModule에 등록되었으므로 루트 모듈의 declarations 프로퍼티에 선언되어 있던 HeaderComponent를 제거하도록 한다.

TYPESCRIPT

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

/* HomeModule 임포트 */
import { HomeModule } from './home/home.module';

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

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    HomeModule  /* HomeModule 임포트 */
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5.2 핵심 모듈(Core module)

핵심 모듈은 애플리케이션 전역에서 공통 사용할 구성 요소들로 구성한 모듈로서 루트 모듈에 등록한다. 애플리케이션 전역에서 사용된다는 의미에서 공유 모듈과 유사하지만 핵심 모듈은 루트 모듈에 등록하여 싱글턴으로 사용하고 공유 모듈은 기능 모듈에 의해 사용된다. 예를 들어 애플리케이션 전역에서 사용하는 데이터 서비스, 인증 서비스, 인증 가드 등이 대상이 된다.

핵심 모듈은 루트 모듈의 구성을 보다 간결하게 관리할 목적으로 어떤 모듈에도 포함되지 않는 독립적인 요소를 루트 모듈에서 분리하여 구성한 모듈이다. 핵심 모듈을 도입하면 루트 모듈에는 라이브러리 모듈, 기능 모듈, 공유 모듈, 핵심 모듈, 라우트 모듈만 등록되어 간결한 관리가 가능하다.

위의 예제에서 user.service.ts는 사용자 정보를 제공하는 서비스로서 애플리케이션 전역에서 공통으로 사용한다. 따라서 이 서비스는 핵심 모듈로 분리할 수 있다. 핵심 모듈인 core 모듈을 작성해 보자.

BASH

```
$ ng generate module core
```

위 명령어를 실행하면 core 폴더에 core.module.ts이 생성된다. 생성한 CoreModule은 서비스만을 제공하고 있기 때문에 CommonModule이 필요 없다.

먼저 핵심 모듈의 구성 요소인 user.service.ts 파일을 core 폴더로 이동시킨다. 이때 User 인터페이스의 경로가 변경되므로 user.service.ts 파일을 아래와 같이 수정한다.

TYPESCRIPT

```
// core/user.service.ts
import { Injectable } from '@angular/core';
import { User } from '../user'; /* 패스 변경 */
...
```

그리고 UserService의 패스가 변경되었으므로 UserService를 import하는 HomeComponent와 HeaderComponent의 패스도 수정하도록 하자. HomeComponent, HeaderComponent의 import 패스를 아래와 같이 변경한다.

TYPESCRIPT

```
// home/home.component.ts와 share/header.component.ts
import { Component, OnInit } from '@angular/core';

import { UserService } from '../core/user.service';
import { User } from '../user';
```

```
...
```

이제 UserService를 CoreModule의 구성요소로 등록하기 위해 프로바이더를 변경하도록 하자.

TYPESCRIPT

```
// core/user.service.ts
import { Injectable } from '@angular/core';
import { CoreModule } from './core.module';

import { User } from '../user';

@Injectable({
  providedIn: CoreModule /* UserService를 CoreModule의 구성요소로 등록*/
})
export class UserService {
  getUser(): User {
    // 임의의 사용자를 반환한다. 실제 환경에서는 서버의 데이터를 취득하여 반환할 것이다.
    return { id: 1, name: 'Lee', admin: true };
  }
}
```

@Injectable의 메타데이터 객체의 providedIn 프로퍼티 값으로 서비스가 등록될 모듈의 타입을 등록한다. 또는 CoreModule의 프로바이더에 UserService를 등록할 수도 있다.

TYPESCRIPT

```
// core/core.module.ts
import { NgModule } from '@angular/core';

/* UserService 임포트 */
import { UserService } from './user.service';

@NgModule({
  imports: [],
  declarations: [],
  providers: [UserService], /* UserService 등록 */
  exports: []
```

```
})
export class CoreModule { }
```

CoreModule의 프로바이더에 UserService를 등록할 경우, 서비스의 @Injectable 데코레이터에 전달할 메타데이터 객체는 삭제한다.

TYPESCRIPT

```
// core/user.service.ts
…
@Injectable()
…
```

CoreModule에 등록된 서비스가 여러 개인 경우, CoreModule의 프로바이더에 서비스를 관리하는 것이 일관된 관리 측면에서 유리할 수 있으므로 이 예제에서는 CoreModule에 프로바이더에 서비스를 등록하도록 하자.

이제 CoreModule이 완성되었다. 이제 루트 모듈의 imports 프로퍼티에 CoreModule을 등록하자.

TYPESCRIPT

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

/* HomeModule 임포트 */
import { HomeModule } from './home/home.module';
/* CoreModule 임포트 */
import { CoreModule } from './core/core.module';

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

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    HomeModule,  /* HomeModule 임포트 */
    CoreModule   /* CoreModule 임포트 */
  ],
```

```
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

기능 모듈, 공유 모듈, 핵심 모듈을 도입하여 모듈을 분리하였다. 루트 모듈에는 루트 컴포넌트와 모듈의 선언만이 존재하고 보다 간결한 구성이 되었다. 프로젝트 폴더 구조 또한 아래와 같이 간결하게 정리되었다.

CODE

```
src
└── app
    ├── core
    │   ├── core.module.ts   # 핵심 모듈
    │   └── user.service.ts
    ├── home
    │   ├── home.component.ts
    │   └── home.module.ts   # 기능 모듈
    ├── shared
    │   ├── header.component.ts
    │   └── shared.module.ts # 공유 모듈
    ├── app.component.ts     # 루트 컴포넌트
    ├── app.module.ts        # 루트 모듈
    └── user.ts
```

지금까지 살펴본 기능 모듈, 공유 모듈, 핵심 모듈로 반드시 애플리케이션을 구성해야 한다는 제약이 있는 것은 아니다. 모듈은 애플리케이션의 코드를 공유하여 재사용 하는 것에 관심이 있다. 적절한 모듈의 분리와 재사용은 애플리케이션을 세련되게 만들며 코드의 양 또한 줄여주는 효과가 있다. 모듈의 규모나 구분 방식은 프로젝트의 규모와 재사용성, 협업 등을 고려하여 결정해야 한다. 따라서 정답이란 있을 수 없다. 이 장의 내용은 애플리케이션에 적절한 구조를 구성할 때 하나의 참고로 이해하기 바란다.

Reference

- Angular NgModules
- @NgModule
- Introducing Angular Modules - Root Module