1. Angular 프로젝트 만들고 실행해보기

★ App.html 에 routerLink tag 속성을 넣었는데 작동을 하지 않음

>> app.ts import RouterLink, RouterLinkActive 추가

>> router-outlet 및 route 개념 파악 필요

1. Angular 배포하는 방법 정리하기
2. Angular에서 Asp.net Core Web API 사용하기

★ CORS오류가 나서 API 수정

★ http Get, Post 사용해보기

★ D/B 데이터 및 SAP 데이터

1. 메뉴 구성해보기
2. Angular 폴더 구조 물어보기
3. Angular component및 여러 개체 만들어 보기
4. 화면 구성
5. 사설 IP에서 Https 구현해보기

☎ 추천 폴더 구조

Angular 앱의 폴더 구조는 정해진 규칙이 없지만, **확장성과 유지보수성**을 위해 몇 가지 모범 사례가 권장됩니다. 핵심은 기능별로 모듈을 나누고, 관련된 파일들을 함께 묶는 것입니다.

**일반적인 권장 폴더 구조**

다음은 중간 규모 이상의 Angular 프로젝트에 널리 사용되는 폴더 구조입니다.

src/

├── app/ # 앱의 핵심 로직

│ ├── core/ # 앱 전체에서 공유되는 서비스 (ex: 인증 서비스, 유틸리티)

│ ├── shared/ # 여러 모듈에서 재사용되는 컴포넌트, 모듈, 파이프 (ex: 버튼 컴포넌트, 공유 UI 모듈)

│ ├── features/ # 기능별로 분리된 폴더 (Lazy Loading을 위해 분리)

│ │ ├── dashboard/ # 대시보드 기능 모듈

│ │ │ ├── components/

│ │ │ ├── services/

│ │ │ ├── dashboard-routing.module.ts

│ │ │ └── dashboard.module.ts

│ │ ├── users/ # 사용자 관리 기능 모듈

│ │ │ ├── components/

│ │ │ ├── services/

│ │ │ └── ...

│ ├── app-routing.module.ts # 전체 앱 라우팅

│ └── app.module.ts # 최상위 앱 모듈

└── assets/ # 이미지, 폰트 등 정적 파일

└── environments/ # 개발, 운영 등 환경 설정 파일

**각 폴더의 역할**

* **core/**: 앱의 모든 곳에서 사용되는 **싱글턴(Singleton) 서비스**를 포함합니다. 예를 들어, AuthService나 LoggerService와 같이 애플리케이션 수명 주기 동안 단일 인스턴스만 필요한 서비스가 여기에 속합니다. 이 모듈은 **최상위 AppModule에서만 한 번** 가져와야 합니다.
* **shared/**: 여러 기능 모듈에서 재사용되는 UI 컴포넌트, 파이프, 디렉티브 등을 포함합니다. 예를 들어, 커스텀 버튼이나 로딩 스피너와 같은 컴포넌트를 여기에 두고 필요한 곳에서 **여러 번** 가져다 쓸 수 있습니다. 이 모듈은 @NgModule({ exports: [...] })에 컴포넌트를 내보내야 합니다.
* **features/**: 앱의 주요 기능들을 담는 곳입니다. **각 기능은 별도의 모듈**로 분리하여 \*\*지연 로딩(Lazy Loading)\*\*을 구현하는 것이 일반적입니다. 이는 초기 로딩 시간을 줄여 앱의 성능을 향상시키는 데 큰 도움이 됩니다.

**폴더 구조 원칙**

1. **기능별 응집**: 기능(예: dashboard, users)과 관련된 모든 파일(컴포넌트, 서비스, 모듈, 라우팅)은 하나의 폴더에 함께 둡니다.
2. **재사용성 분리**: core와 shared 폴더를 사용하여 재사용 가능한 코드를 명확하게 분리합니다.
3. **지연 로딩**: 대규모 앱에서는 features 폴더를 사용하여 지연 로딩을 구현하고, 이는 app-routing.module.ts에서 설정합니다.

이러한 구조를 따르면, 프로젝트의 규모가 커져도 특정 기능을 찾거나 수정하기가 쉬워져 장기적인 **유지보수성**이 크게 향상됩니다.

☎ 사설 IP에서 Https 구현해보기

네, 사설 IP에서도 HTTPS 서비스를 구축할 수 있습니다.

**사설 IP와 HTTPS**

사설 IP 주소는 **네트워크 내부에서만 유효한 IP 주소**입니다. 이는 공인 IP 주소와 달리 인터넷을 통해 직접 접근할 수 없습니다. 따라서 외부에서 사설 IP로 호스팅되는 HTTPS 서비스에 직접 접속하는 것은 불가능합니다. 하지만, 내부 네트워크 사용자들은 사설 IP를 이용해 HTTPS 서비스에 접속할 수 있습니다.

**외부 접속 방법**

만약 외부에서 사설 IP의 HTTPS 서비스에 접속하고 싶다면, 다음과 같은 방법들을 활용할 수 있습니다.

* **포트 포워딩 (Port Forwarding):** 공유기(라우터) 설정을 통해 외부에서 공유기의 공인 IP로 들어오는 특정 포트의 요청을 내부 네트워크의 사설 IP 주소와 포트로 전달하도록 설정하는 방법입니다.
* **VPN (Virtual Private Network):** 외부 사용자가 VPN을 통해 내부 네트워크에 접속하면, 사설 IP 주소에 직접 접근할 수 있습니다.
* **리버스 프록시 (Reverse Proxy):** 공인 IP를 가진 서버를 리버스 프록시로 설정하고, 이 서버가 외부 요청을 내부의 사설 IP 서버로 전달하는 방식입니다.

**SSL/TLS 인증서**

HTTPS 서비스를 구축하려면 **SSL/TLS 인증서**가 필요합니다. 사설 IP 주소를 사용하는 경우, **자체 서명(Self-Signed) 인증서**를 사용하거나, \*\*내부 CA(Certificate Authority)\*\*를 구축하여 인증서를 발급받을 수 있습니다.

* **자체 서명 인증서:** 신뢰할 수 있는 기관이 발급한 인증서가 아니기 때문에, 브라우저에서 **"안전하지 않은 연결"** 경고가 나타날 수 있습니다.
* **내부 CA:** 회사나 기관 등 내부 네트워크에서 자체적으로 인증서를 발급하고 관리하는 방식입니다. 내부 사용자들에게 CA를 신뢰하도록 설정하면 경고 없이 HTTPS 연결을 사용할 수 있습니다.

요약하자면, 사설 IP에서 HTTPS 서비스 구축은 가능하며, 내부 네트워크에서 주로 사용되거나 포트 포워딩, VPN 등을 통해 외부에서 접속할 수 있습니다.