질문	답변
CORS가 뭔가요?	CORS는 **웹 브라우저에서 다른 도메인(Origin) 간에 자원을 공유할 수 있도록 해주는 보안 메커니즘**입니다. 웹 애플리케이션은 **보안상**의 이유로 **동일 출처 정책(Same-Origin Policy)**에 따라 **다른 출처의 리소스에 접근할 수 없습니다.** 출처는 프로토콜, 도메인, 포트 번호를 기준으로 결정됩니다. CORS는 이러한 **동일 출처 정책을 우회하여 다른 도메인 간에 데이터를 공유하기 위한 방법을 제공**합니다. 웹 애플리케이션의 **서버와 브라우저 간에 정의된 규칙**을 따라, 다른 도메인의 **리소스 요청을 허용하거나 거부**할 수 있습니다. 이를 통해 **보안상의 이슈나 데이터 유출을 방지**하면서도 **필요한 데이터 교환**을 할 수 있게 됩니다. CORS는 **HTTP 헤더를 사용하여 구현**되며, 주로 서버 측에서 설정됩니다. 주요한 CORS 관련 헤더는 다음과 같습니다. Access-Control-Allow-Origin: 허용할 출처(도메인) 목록을 지정합니다. "*"를 사용하여 모든 출처를 허용할 수도 있습니다. Access-Control-Allow-Methods: 허용할 HTTP 메서드 목록을 지정합니다. Access-Control-Allow-Headers: 허용할 HTTP 헤더 목록을 지정합니다. Access-Control-Expose-Headers: 클라이언트가 접근할 수 있는 헤더 목록을 지정합니다. Access-Control-Allow-Credentials: 쿠키와 같은 인증 정보를 요청에 포함시킬지 여부를 나타냅니다. Access-Control-Max-Age: 사전 요청(Preflight)의 유효 시간을 지정합니다. CORS는 보안 및 데이터 공유의 중요한 측면을 다루므로 웹 개발자들은 이를 이해하고 올바르게 구성하여 웹 애플리케이션이 원활하게 작동하고 안전하게 운영될 수 있도록 해야 합니다.
CORS를 겪고 직접 해결해 본 경험이 있으면 말해주세요	React를 사용한 프로젝트에서 CORS 문제가 발생했을때 **proxy middleware를 사용하여 해결**했습니다. 클라이언트 코드에서 API 요청을 보낼 때, /api 경로를 사용하여 프록시 미들웨어를 통해 API 서버로 요청을 보냅니다. 이렇게 설정하면 **http-proxy-middleware**를 사용하여 프록시 미들웨어를 통해 API 요청을 보내게 되며, 브라우저에서의 CORS 문제를 우회하면서 API 데이터를 가져올 수 있게 됩니다.
SOP가 무엇이고 SOP를 지키는 이유	**웹 보안을 강화**하기 위해서입니다. 만약 모든 출처에서 **자유롭게 리소스에 접근**할 수 있다면, 해커가 웹 페이지를 통해 사용자의 **데이터에 액세스**하거나 **악의적인 스크립트를 실행**시키는 등의 공격이 더 쉬워질 것입니다. 예를 들어, **Cross-Site Request Forgery (CSRF)나 Cross-Site Scripting (XSS)**와 같은 공격이 발생할 수 있습니다.
쿠키가 뭔가요?	쿠키는 **브라우저 로컬에 저장**되는 **키와 값이 들어있는 작은 데이터 파일**이다. 웹사이트 방문 시에 **브라우저에 저장**되고, 그 이후 해당 웹사이트를 방문할 때 **서버로 전송**됩니다. 쿠키는 **클라이언트와 서버 간에 상태를 유지하고 정보를 저장하**는 데 사용됩니다. **사용자 인증**이 **유효한 시간을 명시**할 수 있으며, **유효 시간이 정해지면 브라우저가 종료되어도 인증이 유지**된다는 특징이 있다. 클라이언트에 300개까지 쿠키저장 가능, 하나의 도메인당 20개의 값만 가질 수 있음, 하나의 쿠키값은 4KB까지 저장한다. Response Header에 Set-Cookie 속성을 사용하면 클라이언트에 쿠키를 만들 수 있다. 쿠키는 사용자가 따로 요청하지 않아도 브라우저가 Request시에 Request Header를 넣어서 자동으로 서버에 전송한다.
쿠키, 세션을 왜 쓰나요?	**HTTP 프로토콜의 특성**이자 약점을 **보완**하기 위해서 **쿠키 또는 세션을 사용**한다. 기본적으로 HTTP 프로토콜 환경은 **비연결성, 무상태성** 이라는 특성을 가지기 때문에 **서버는 클라이언트가 누구인지 매번 확인**해야한다. 이 특성을 보완하기 위해서 쿠키와 세션을 사용하게된다. * connectionless: 클라이언트가 **요청을 한 후 응답**을 받으면 그 **연결을 끊어** 버리는 특징 * stateless: **통신이 끝**나면 **상태를 유지하지 않는** 특징
쿠키의 구성 요소	이름 : 각각의 쿠키를 구별하는 데 사용되는 이름 값 : 쿠키의 이름과 관련된 값 **유효시간** : 쿠키의 유지시간 도메인 : 쿠키를 전송할 도메인 경로 : 쿠키를 전송할 요청 경로

세션이 뭔가요?	**세션은 쿠키를 기반**하고 있지만, 사용자 정보 파일을 **브라우저에 저장하는 쿠키와 달리 세션은 서버 측에서 관리**한다. 서버에서는 클라이언트를 **구분하기 위해 세션 ID**를 부여하며 웹 브라우저가 서버에 접속해서 **브라우저를 종료할 때까지 인증상태를 유지**한다. 물론 접속 시간에 제한을 두어 일정 시간 응답이 없다면 정보가 유지되지 않게 설정이 가능하다. 사용자에 대한 정보를 서버에 두기 때문에 **쿠키보다 보안에 좋지**만, 사용자가 많아질수록 **서버 메모리를 많이 차지**하게 된다. 즉 동접자 수가 많은 웹 사이트인 경우 **서버에 과부하를 주게 되므로 성능 저하의 요인**이 된다. 클라이언트가 Request를 보내면, 해당 서버의 엔진이 클라이언트에게 유일한 ID를 부여하는 데 이것이 세션 ID이다.
쿠키와 세션의 차이는 어떤 점이 있을까요?	가장 큰 차이점은 **사용자의 정보가 저장되는 위치**다. 쿠키는 서버의 자원을 전혀 사용하지 않으며, 세션은 서버의 자원을 사용한다. **보안 면에서 세션이 더 우수**하며, **요청 속도는 쿠키**가 세션보다 더 빠르다. 그 이유는 세션은 **서버의 처리가 필요**하기 때문이다. 보안, 쿠키는 **클라이언트 로컬에 저장**되기 때문에 **변질**되거나 **request에서 스니핑** 당할 우려가 있어서 **보안에 취약**하지만 **세션은 쿠키를 이용해서 sessionid 만 저장**하고 그것으로 구분해서 **서버에서 처리하기 때문에 비교적 보안성이 좋다.** 라이프 사이클, **쿠키도 만료시간**이 있지만 **파일로 저장**되기 때문에 브라우저를 **종료해도 계속해서 정보**가 남아 있을 수 있다. 또한 **만료기간을 넉넉**하게 잡아두면 **쿠키삭제를 할 때 까지** 유지될 수도 있다. 반면에 **세션도 만료시간**을 정할 수 있지만 **브라우저가 종료**되면 만료시간에 상관없이 **삭제**됩니다. 예를 들어, 크롬에서 **다른 탭을 사용해도 세션이 공유**된다. 다른 브라우저를 사용하게 되면 다른 세션을 사용할 수 있다. 속도, 쿠키에 정보가 있기 때문에 서버에 요청시 속도가 빠르고 세션은 정보가 서버에 있기 때문에 처리가 요구되어 비교적 느린 속도를 가진다.
캐시란 무엇인가요?	데이터나 값을 **임시로 저장**하는 **고속 저장소**를 의미합니다. 캐시는 주로 **빠른 데이터 액세스를 위해 사용**되며, 주로 느린 주 메모리나 데이터베이스 등의 백엔드 **저장소로부터 데이터**를 가져와서 저장해 두고 **빠르게 액세스**할 수 있도록 도와줍니다. 이로써 시스템의 **전반적인 성능을 향상**시킬 수 있습니다. 캐시는 다양한 컴퓨터 시스템과 구성요소에서 사용됩니다.
하드웨어 캐시	하드웨어 캐시는 주로 **CPU와 메모리 사이**에서 데이터를 효율적으로 전달하고 처리 속도를 높이기 위해 사용됩니다. CPU가 **메모리에서 데이터를 가져올 때 시간이 더 오래** 걸리므로, 이를 보완하기 위해 **빠른 속도로 액세스할 수 있는 캐시 메모리**를 사용합니다. 보통 레벨 1(L1), 레벨 2(L2), 레벨 3(L3)과 같이 다중 계층으로 구성되며, 각 계층별로 크기와 액세스 속도 등이 다를 수 있습니다. 작은 용량의 캐시 메모리가 빈번하게 사용되는 데이터나 명령어를 저장하고, 이를 이용해 CPU의 작업 속도를 향상시킵니다.
소프트웨어 캐시	소프트웨어 캐시는 주로 **소프트웨어에서 데이터나 결과를 임시로 저장**하는 데 사용됩니다. 예를 들어 웹 브라우저에서는 이미지, 스타일 시트, 자바스크립트 파일 등을 로드할 때 이를 로컬 캐시에 저장하여 다음에 같은 자원을 요청할 때 서버에서 다시 내려받지 않고 로컬에서 바로 가져와 사용합니다. 이로써 웹 페이지 **로딩 속도가 향상**됩니다. 캐시의 원리는 데이터의 지역성 원리에 기반합니다. 데이터 액세스 패턴에서는 특정 데이터에 대한 액세스가 집중되는 경향이 있으며, 이를 이용해 빠른 액세스를 위해 해당 데이터를 캐시에 저장합니다. 캐시는 데이터 일관성과 관련된 문제를 일으킬 수 있습니다. 데이터가 **주 메모리와 캐시 메모리 간에 변화할 때 일관성을 유지하는 것이 중요**하며, 이를 위해 **캐시 일관성 프로토콜**과 같은 메커니즘이 사용됩니다.
캐싱	캐싱은 데이터나 정보를 **미리 저장하여 나중에 빠르게 접근하거나 재사용할 수 있는 메커니즘**을 말합니다. 주로 데이터나 리소스에 접근하는 시간을 줄이기 위해 사용되며, 여러 컴퓨팅 분야에서 널리 활용됩니다. 캐싱은 다양한 시스템과 환경에서 사용되며, 주요 목적은 성능 향상과 리소스 효율성을 높이는 것입니다.
웹 브라우저 캐싱	브라우저 캐싱은 **웹 브라우저가 웹 페이지의 자원을 로컬 컴퓨터에 임시로 저장**하여, 이후에 같은 자원을 요청할 때 서버로부터 다시 다운로드하지 않고 **로컬에서 가져와서** 더 **빠르게** 웹 페이지를 표시하는 기술입니다. 이는 웹 페이지 **로딩 속도를 개선**하고 네트워크 트래픽을 줄이는데 도움을 줍니다. 브라우저 캐싱은 주로 **정적인 리소스**에 적용됩니다. 정적인 리소스는 웹 페이지를 구성하는데 사용되지만, **변경 빈도**가 낮거나 거의 없는 리소스입니다. 이에는 이미지, 스타일 시트(CSS), 자바스크립트 파일 등이 포함됩니다. 이런 리소스들은 한 번 다운로드되면 여러 페이지에서 재사용될 수 있으므로 브라우저 캐싱을 통해 **다운로드 횟수를 최소화**할 수 있습니다.

브라우저 캐싱은 크게 두 가지 유형	브라우저 캐시 (Browser Cache) 브라우저는 웹 페이지의 **정적인 자원을 로컬 저장소에 캐싱**합니다. 웹 페이지를 방문할 때, 브라우저는 이전에 다운로드한 자원 중 최신 버전을 로컬 캐시에서 찾아서 사용합니다. 웹 페이지의 **<link> 태그나 HTTP 응답 헤더를 통해 리소스의 캐싱 정책**을 지정할 수 있습니다. 프록시 캐시 (Proxy Cache) **프록시 서버를 통해 웹 페이지에 접근**하는 경우, 프록시 서버는 **원격 서버로부터 받은 리소스를 로컬 캐시에 저장**하여 다음에 동일한 리소스가 요청될 때 원격 서버로부터 다시 내려받지 않도록 합니다. 이는 기업이나 조직에서 **내부 네트워크에서 공유하는 인터넷 연결을 효율적으로 관리**하기 위해 사용됩니다.