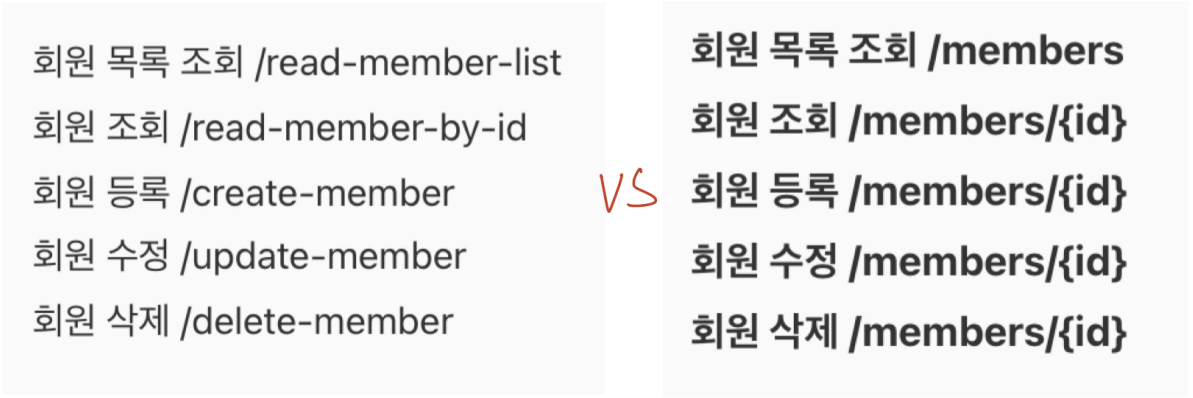
RESTFUL



두 API중 어느게 더 좋은 API 일까?

API에서 URI를 설계할 때 가장 중요한 것은 **리소스 식별**이다. 리소스란 회원을 등록하고 수정하고 조회하는 행위가 아니다. 회원이라는 개념 자체가 바로 리소스다. 즉 URI에서 기능은 모두 빼버리고 회원이라는 리소스만 식별할 수 있도록 설계하는게 더 좋은 API 설계라 할 수 있다. 그렇다면 리소스와 행위는 어떻게 구분할까?? 바로 HTTP 메서드를 이용하면 된다.

내 생각: 일관성과 문서화 비용 감소

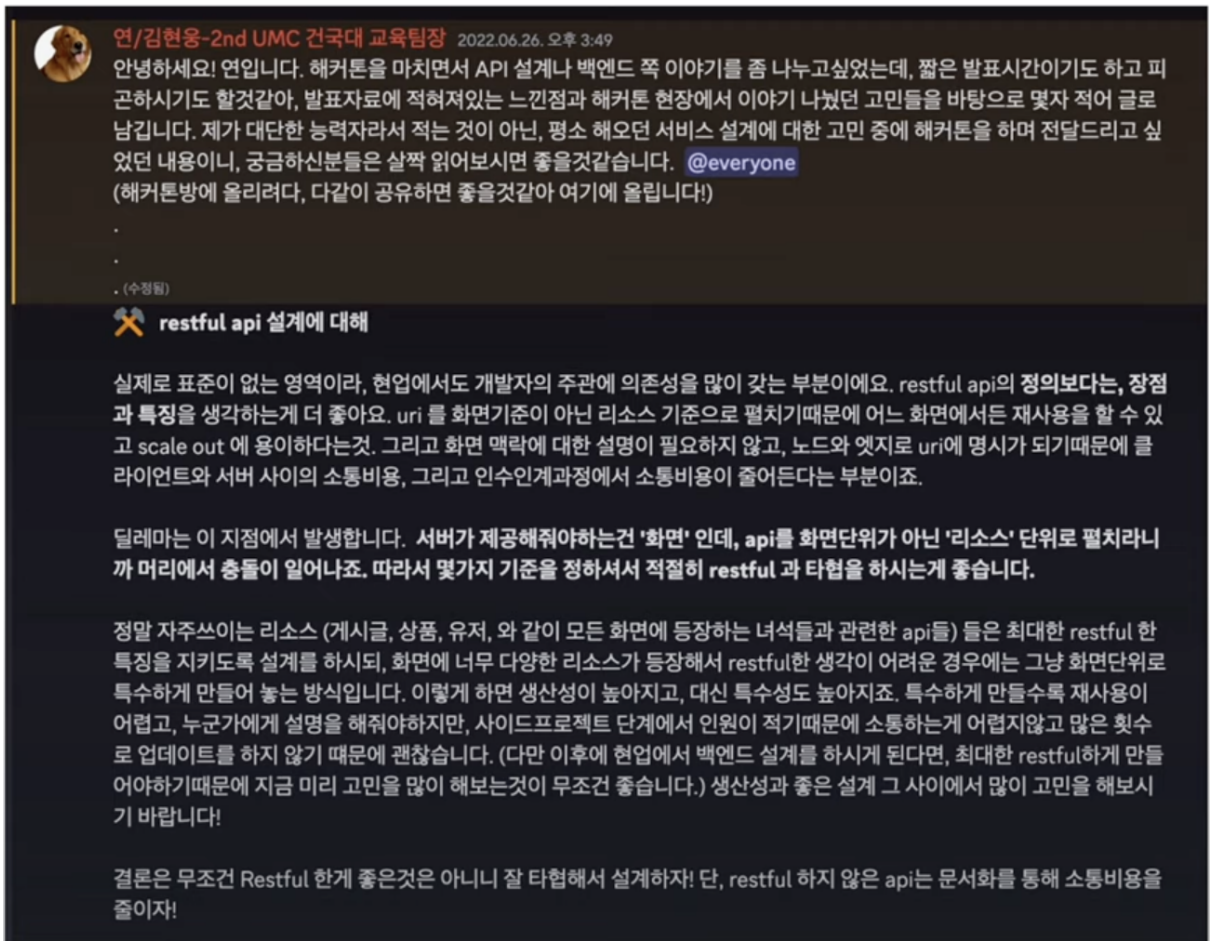
REST API 기본 개념

REST API는 Representational State Transfer API의 약자로 웹 서비스를 위한 소프트웨어 아키텍처 스타일 중 하나이다. 클라이언트-서버 모델을 따르며, 클라이언트가 서버에 요청을 보내면 서버가 데이터를 반환하는 방식으로 작동한다.

REST API는 자원(resource)을 표현하는 URI(Uniform Resource Identifier)를 사용한다. 클라이언트는 URI를 통해 자원에 접근하며, 서버는 HTTP 메서드(GET, POST, PUT, DELETE 등)를 사용하여 클라이언트 요청을 처리한다. REST API는 상태를 유지하지 않고, 요청과 응답 간에 상태 정보가 없는 상태를 유지한다. 따라서 REST API는 확장성과 유연성이 높아, 다양한 웹 서비스에서 사용되며, 모바일 앱, 웹 애플리케이션 등에서도 많이 활용된다. REST API는 클라이언트와 서버 간의 통신을 표준화하고, 상호 운용성(interoperability)을 제공하므로, 서로 다른 플랫폼에서도 쉽게 사용할 수 있다.

REST API의 장점 및 특징

1. REST API는 HTTP 메서드를 사용하기 때문에 HTTP 메서드가 가지고 있는 특징들을 가진다. REST API는 HTTP 메서드(GET, POST, PUT, DELETE 등)를 사용하여 자원에 대한 요청을 처리한다. 각 메서드는 특정한 의미를 가지며, 일반적으로 GET은 자원을 조회하는 데, POST는 자원을 생성하는 데, PUT은 자원을 수정하는 데, DELETE는 자원을 삭제하는 데 사용된다.
2. 자원 지향: REST API는 자원(resource)을 중심으로 설계된다. 각 자원은 고유한 URI(Uniform Resource Identifier)를 가지며, 클라이언트는 URI를 사용하여 자원에 접근한다.
3. Stateless(무상태성): REST API는 클라이언트와 서버 간에 상태 정보를 유지하지 않는다. 클라이언트가 요청을 보내면 서버는 요청에 대한 응답을 반환하고, 클라이언트와 서버 간의 연결은 끊어진다.
4. 캐시 가능: REST API는 HTTP 프로토콜을 기반으로 하므로, 캐싱을 활용하여 성능을 향상시킬 수 있다.
5. 자체 설명(표현 구조): REST API는 자원에 대한 설명 정보를 포함하고 있어서, 클라이언트는 이 정보를 사용하여 자원에 대해 쉽게 이해할 수 있다.
6. 다양한 데이터 형식 지원: REST API는 다양한 데이터 형식을 지원한다. 현재 가장 일반적인 형식은 JSON이지만 XML, CSV, YAML 등의 형식도 지원된다.


연/김현웅-2nd UMC 건국대 교육팀장 2022.06.26. 오후 3:49
안녕하세요! 연입니다. 해커톤을 마치면서 API 설계나 백엔드 쪽 이야기를 좀 나누고싶었는데, 짧은 발표시간이기도 하고 피곤하시기도 할것같아, 발표자료에 적혀져있는 느낀점과 해커톤 현장에서 이야기 나눴던 고민들을 바탕으로 몇자 적어 글로 남깁니다. 제가 대단한 능력자라서 적는 것이 아닌, 평소 해오던 서비스 설계에 대한 고민 중에 해커톤을 하며 전달드리고 싶었던 내용이니, 궁금하신분들은 살짝 읽어보시면 좋을것같습니다. @everyone
(해커톤방에 올리려다, 다같이 공유하면 좋을것같아 여기에 올립니다!)
.
.
. (수정됨)
restful api 설계에 대해
실제로 표준이 없는 영역이라, 현업에서도 개발자의 주관에 의존성을 많이 갖는 부분이에요. restful api의 정의보다는, 장점과 특징을 생각하는게 더 좋아요. uri 를 화면기준이 아닌 리소스 기준으로 펼치기때문에 어느 화면에서든 재사용을 할 수 있고 scale out 에 용이하다는것. 그리고 화면 맥락에 대한 설명이 필요하지 않고, 노드와 엣지로 uri에 명시가 되기때문에 클라이언트와 서버 사이의 소통비용, 그리고 인수인계과정에서 소통비용이 줄어든다는 부분이죠.
딜레마는 이 지점에서 발생합니다. 서버가 제공해줘야하는건 '화면' 인데, api를 화면단위가 아닌 '리소스' 단위로 펼치라니까 머리에서 충돌이 일어나죠. 따라서 몇가지 기준을 정하셔서 적절히 restful 과 타협을 하시는게 좋습니다.
정말 자주쓰이는 리소스 (게시글, 상품, 유저, 와 같이 모든 화면에 등장하는 녀석들과 관련한 api들) 들은 최대한 restful 한 특징을 지키도록 설계를 하시되, 화면에 너무 다양한 리소스가 등장해서 restful한 생각이 어려운 경우에는 그냥 화면단위로 특수하게 만들어 놓는 방식입니다. 이렇게 하면 생산성이 높아지고, 대신 특수성도 높아지죠. 특수하게 만들수록 재사용이 어렵고, 누군가에게 설명을 해줘야하지만, 사이드프로젝트 단계에서 인원이 적기때문에 소통하는게 어렵지않고 많은 횟수로 업데이트를 하지 않기 때문에 괜찮습니다. (다만 이후에 현업에서 백엔드 설계를 하시게 된다면, 최대한 restful하게 만들어야하기때문에 지금 미리 고민을 많이 해보는것이 무조건 좋습니다.) 생산성과 좋은 설계 그 사이에서 많이 고민을 해보시기 바랍니다!
결론은 무조건 Restful 한게 좋은것은 아니니 잘 타협해서 설계하자! 단, restful 하지 않은 api는 문서화를 통해 소통비용을 줄이자!