\* Rest API가 무엇인 지 학습하고 Github에 올립니다.

1) HTTP 통신에 관하여

2) 브라우저에 URL을 입력 후 요청하여 서버에서 응답하는 과정까지에 대해

학습 한 내용을 작성

**REST(Representational State Transfer)**

* 네트워크 상에서 Client와 Server사이의 통신 방식 중 하나로, 자원을 이름(자원의 표현)으로 구분하여 해당 자원의 상태(정보)를 주고 받는 모든 것.
* 자원(resource)의 표현(representation)에 의한 상태 전달이며, ROA(Resource Oriented Architecture)를 따르는 웹 서비스 아키텍처.
* HTTP URI(Uniform Resource Identifier)를 통해 자원(Resource)을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD Operation을 적용.

**REST CRUD Operation​**

* CRUD : 대부분의 컴퓨터 소프트웨어가 가지는 기본적인 데이터 처리 기능(Create, Read, Update, Delete를 묶어서 일컫는 말.
* URI와 HTTP 메소드를 이용해 객체화된 서비스에 접근.
* 직관적인 URI를 이용해서 오브젝트의 멤버를 추적하는 것처럼 리소스에 접근하여 HTTP 메소드를 이용해 조작.

**REST의 특징**

* Server-Client(서버-클라이언트 구조)
  + 자원이 있는 쪽이 Server, 자원을 요청하는 쪽이 Client.
  + REST Server는 API를 제공해 비즈니스 로직 처리 및 저장을 책임지고, Client는 사용자 인증이나 context(세션, 로그인 정보) 등을 직접 관리하고 책임.
  + REST를 사용하면 서버와 클라이언트 간 의존성이 줄어듦.
* Stateless(무상태)
  + 이전, 이후에 대한 직접적인 정보가 필요없이 직관적인 오브젝트에의 접근으로 서비스를 처리.
  + 세션정보를 보관할 필요가 없기 때문에, 서비스의 자유도 또한 높아지고 로드밸런싱이라든지 기타 유연한 아키텍처의 적용이 가능.
  + 쿠키와 세션이 불필요.
* Cacheable(캐시 처리 가능)
  + 웹 표준 HTTP 프로토콜을 그대로 사용하므로 웹에서 사용하는 기존의 인프라를 그대로 활용 가능.
  + 캐시 사용을 통해 응답시간이 빨라지고 REST Server 트랜잭션이 발생하지 않기 때문에 전체 응답시간, 성능, 서버의 자원 이용률이 향상.
* Layered System(계층화)
  + Client는 REST API Server만 호출.
  + REST Server는 다중 계층으로 구성 가능.
  + PROXY, 게이트웨이 같은 네트워크 기반의 중간 매체를 사용 가능..
* Addressability(URI 접근)
  + REST는 모든 유일한 오브젝트에 대해 유일하고 직관적인 URI을 통해 접근하도록 함.
  + 웹 사이트의 이미지, 텍스트, DB 내용 등의 모든 자원에 고유한 ID인 HTTP URI를 부여.
* Uniform Interface(인터페이스 일관성)
  + URI로 지정한 리소스 조작을 통일되고 한정적인 인터페이스로 수행.

**REST의 장점**

* HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구축 불필요.
* HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 사용 가능.
* HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용 가능.
* Hypermedia API의 기본을 충실히 지키면서 범용성을 보장.
* REST API 메시지가 의도하는 바를 명확하게 나타내므로 의도하는 바를 쉽게 파악 가능.
* 여러가지 서비스 디자인에서 생길 수 있는 문제를 최소화.
* 서버와 클라이언트의 역할을 명확하게 분리.

**REST의 단점**

* 표준이 없음.
* 사용할 수 있는 메소드가 4가지가 전부.
* HTTP Method 형태가 제한적.
* 구형 브라우저가 지원해주지 못하는 부분(PUT, DELETE, pushState)이 존재.

**REST가 필요한 시점**

▶ 애플리케이션 분리 및 통합을 해야할 때

▶ 다양한 클라이언트들이 등장 했을 때

▶ 다양한 브라우저, 안드로이폰, 아이폰과 같은 모바일 디바이스에서도 통신 가능한 서버 프로그램이 필요할 때

▶ 멀티 플랫폼에 대한 지원을 위해 서비스 자원에 대한 아키텍처가 필요할 때

**REST API**

* REST 기반으로 서비스 API를 구현한 것.
* 최근 OpenAPI(누구나 사용할 수 있도록 공개된 API: 구글 맵, 공공 데이터 등), 마이크로 서비스(하나의 큰 애플리케이션을 여러 개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처) 등을 제공하는 업체 대부분은 REST API를 제공.

**REST API의 특징**

* 사내 시스템들도 REST 기반으로 시스템을 분산해 확장성과 재사용성을 높여 유지보수 및 운용이 편리.
* REST는 HTTP 표준을 기반으로 구현하므로, HTTP를 지원하는 프로그램 언어로 클라이언트, 서버를 구현 가능.
* REST API를 제작하면 델파이 클라이언트 뿐 아니라, 자바, C#, 웹 등을 이용해 클라이언트를 제작 가능.

**RESTful**

* REST라는 아키텍처를 구현하는 웹 서비스를 나타내기 위해 사용되는 용어.
* ‘REST API’를 제공하는 웹 서비스를 ‘RESTful’하다고 표현.
* 즉, REST 원리를 따르는 시스템은 RESTful이란 용어로 지칭.

**RESTful의 목적**

* 이해하기 쉽고 사용하기 쉬운 REST API를 만드는 것.
* 성능 향상에 있는 것이 아니라 일관적인 컨벤션을 통한 API의 이해도 및 호환성을 높이는 것이 주 목적.

**RESTful하지 못한 경우**

* CRUD 기능을 모두 POST로만 처리하는 API.
* 라우트(route)에 resource, id 외의 정보가 들어가는 경우(/books/updateName).

**HTTP 통신이란?**

* HyperText Transfer Protocol의 약자로 HTML 파일을 전송하는 프로토콜.
* 웹브라우저에서 통신이 일어나며, 초기에는 HTML 파일을 전송하려는 목적으로 만들어졌으나 현재는 JSON, Image 파일 등 다양한 파일을 전송.

**HTTP 통신의 통신 방식**

* HTTP 통신은 클라이언트에서 서버로 요청을 보내고 서버가 응답하는 방식.
* 응답에는 클라이언트의 요청에 따른 결과를 반환.
* 단방향 통신으로서 클라이언트의 요청이 있을 때 서버가 응답하는 방식.
* 서버의 응답에는 응답 코드가 같이 전송되며, 사용자는 응답 코드와 메세지 응답으로부터 오는 메세지 바디를 통해 요청 값을 전달받음.
* 초기에는 서버는 응답한 후 클라이언트(사용자)의 Connection을 곧바로 끊어버렸으나, 최근에는 성능상의 이유(Connection을 맺고 끊는 비용이 비싸다)로 Keep Alive 옵션을 통해 일정 기간 동안 클라이언트와 Connection을 유지하는 방식으로 통신이 가능.