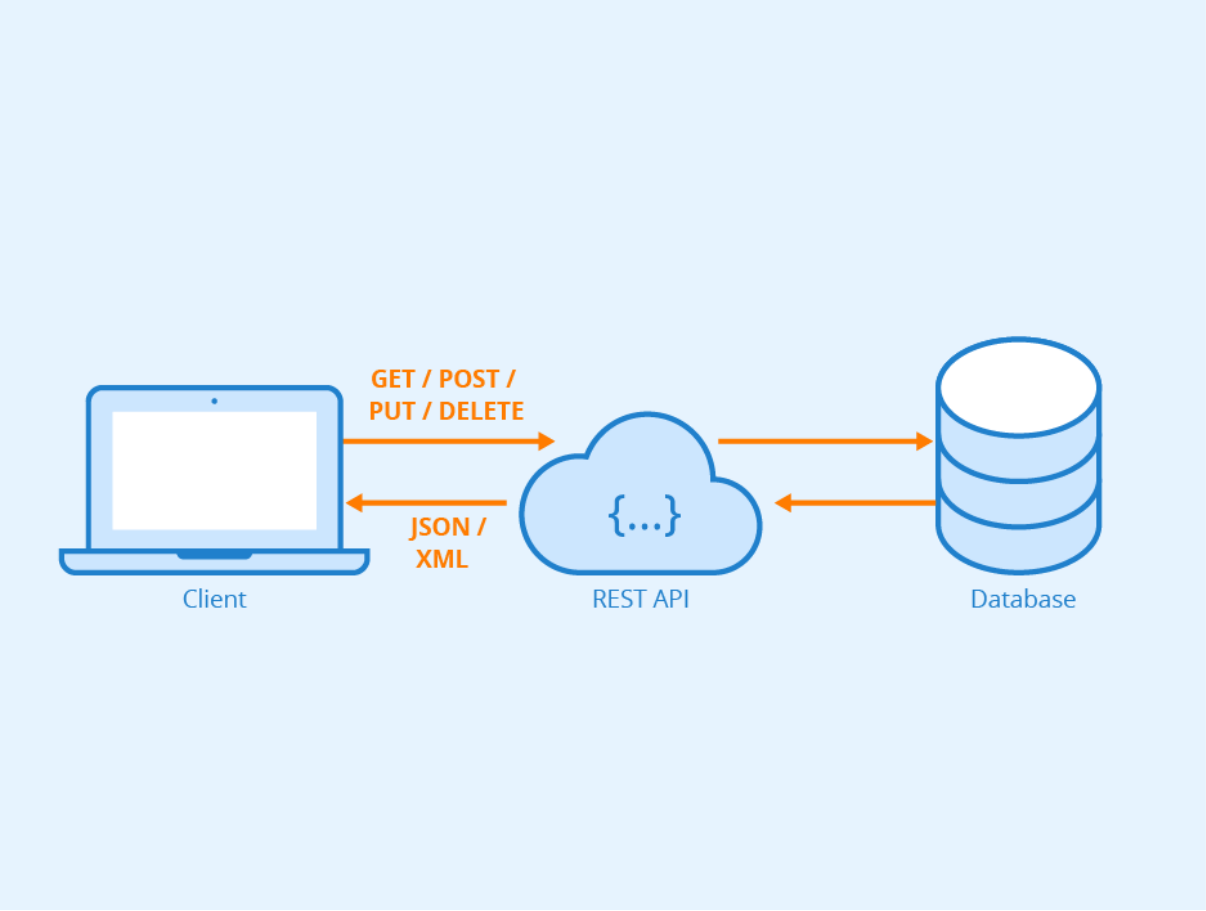
RESTful

태그	NETWORK
주차	7주차

목차

1. REST API



- Representation State Transfer
- 소프트웨어 프로그램 아키텍쳐의 한 형식
- 자원의 이름(자원의 표현)으로 구분하여 해당 자원의 상태(정보)를 주고받는 모든 것을 의미
- 월드 와이드 웹(WWW)과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 개발 아키텍처의 한 형식
- 웹의 기존 기술과 HTTP 프로토콜을 그대로 활용하기 때문에 웹의 장점을 최대한 활용할 수 있는 아키텍처 스타일

1-1. RESTful API를 만드는 이유

- Client Side를 정형화된 플랫폼이 아닌 모바일, PC, 어플리케이션 등 플랫폼에 제약을 두지 않는 것을 목표로 했기 때문

1-2. 구성

1) 자원(Resource) URL

- 모든 자원에 고유한 ID가 존재하고, 이 자원은 서버에 존재
- 자원을 구별하는 ID는 `/orders/order_id/1`과 같은 HTTP URI

2) 행위(Verb) - HTTP Method

- HTTP 프로토콜의 Method 사용
  - `GET`, `POST`, `PUT`, `DELETE`, `PATCH`와 같은 메소드 제공

3) 표현(Representation of Resource)

- 클라이언트가 자원의 상태(정보)에 대한 조작을 요청하면 서버는 이에 적절한 응답(Representation)을 보냄
- REST에서 하나의 자원은 JSON, XML, TEXT, RSS 등 여러 형태의 Representation으로 나타낼 수 있음
- 현재는 대부분 JSOn

1-3. 특징

1) 클라이언트-서버 구조

- 클라이언트는 유저와 관련된 처리를, 서버는 REST API를 제공함으로써 각각의 역할이 확실하게 구분되고 일괄적인 인터페이스로 분리되어 작동할 수 있게 함
- 서로 간 의존성이 줄어듦
- **REST Server**
  - API를 제공하고 비지니스 로직 처리 및 저장을 맡음
- **Client**
  - 사용자 인증이나 context (세션, 로그인 정보) 등을 직접 관리하고 책임진다.

2) 무상태성(Stateless)

- REST는 HTTP의 특성을 이용하기 때문에 무상태성을 가짐
- 즉 서버에서 어떤 작업을 하기 위해 상태정보를 기억할 필요가 없고 들어온 요청에 대해 처리만 해주면 되기 때문에 구현이 쉽고 단순

3) 캐시 처리 가능(Cacheable)

- HTTP라는 기존 웹표준을 사용하는 REST의 특징 덕분에 기본 웹에서 사용하는 인프라를 그대로 사용 가능

- 대량의 요청을 효율적으로 처리하기 위해 캐시가 요구
- 캐시 사용을 통해 응답시간이 빨라지고 REST Server 트랜잭션이 발생하지 않기 때문에 전체 응답시간, 성능, 서버의 자원 이용률 향상

4) 자체 표현 구조(Self-descriptiveness)

- JSON을 이용한 메시지 포멧을 이용하여 직관적으로 이해할 수 있고 REST API 메시지만으로 그 요청이 어떤 행위를 하는지 알 수 있음

5) 계층화(Layered System)

- 클라이언트와 서버가 분리되어 있기 때문에 중간에 프록시 서버, 암호화 계층 등 중간매체를 사용할 수 있어 자유도가 높음

6) 유니폼 인터페이스(Uniform)

- Uniform Interface는 Http 표준에만 따른다면 모든 플랫폼에서 사용이 가능하며, URI로 지정한 리소스에 대한 조작을 가능하게 하는 아키텍쳐 스타일
- URI로 지정한 Resource에 대한 조작을 통일되고 한정적인 인터페이스로 수행
- 즉, 특정 언어나 기술에 종속되지 않음

1-4. 규칙

1) 중심 규칙

- REST에서 가장 중요하며 기본적인 규칙은 아래 두 가지
  1. URI는 정보의 자원을 표현해야 함
  2. 자원에 대한 행위는 HTTP Method(GET, POST, PUT, DELETE 등)으로 표현

2) 세부 규칙

1. 슬래시 구분자(`/`)는 계층 관계를 나타내는 데 사용
2. URI 마지막 문자로 슬래시(`/`)를 포함하지 않음
   - 즉, URI에 포함되는 모든 글자는 리소스의 유일한 식별자로 사용돼야 하며, URI가 다르다는 것은 리소스가 다르다는 것
   - 역으로 리소스가 다르면 URI도 달라져야 함
3. 하이픈(`-`)은 URI 가독성을 높이는데 사용함
4. 밑줄(`_`)은 URI에 사용하지 않음

5. URI 경로에는 소문자가 적합
    - URI에 대문자 사용은 피하도록 함
6. 파일 확장자에는 URI에 포함하지 않음
    - REST API에서는 메세지 바디 내용의 포맷을 나타내기 위한 파일 확장자를 URI에 포함시키지 않음
    - 대신 Accept Header 사용
    - ex) `GET: http://restapi.exam.com/orders/2/Accept: image/jpg`
7. 리소스 간에 연관 관계가 있는 경우
    - /리소스명/리소스ID/관계가 있는 다른 리소스명
    - ex) `GET: /users/2/orders` (일반적으로 소유의 관계를 표현할 때 사용)

1-5. 설계 목표

1. 컴포넌트들간의 유연한 (쉽게 확장가능한) 상호 연동성 확보
2. 범용 인터페이스
3. 각 컴포넌트들의 독립적인 배포
4. 지연 감소, 보안강화, 레거시 시스템을 `인캡슐레이션` 하는 중간 컴포넌트로의 역할