## **RESTful API**

(아마존 웹 서비스 설명문서를 참조함)

## **RESTful API란 무엇인가요?**

RESTful API는 **두 컴퓨터 시스템이 인터넷을 통해 정보를 안전하게 교환하기 위해 사용하는 인터페이스**입니다. 대부분의 비즈니스 애플리케이션은 다양한 태스크를 수행하기 위해 다른 내부 애플리케이션 및 서드 파티 애플리케이션과 통신해야 합니다. 예를 들어 월간 급여 명세서를 생성하려면 인보이스 발행을 자동화하고 내부의 근무 시간 기록 애플리케이션과 통신하기 위해 내부 계정 시스템이 데이터를 고객의 뱅킹 시스템과 공유해야 합니다. RESTful API는 안전하고 신뢰할 수 있으며 효율적인 소프트웨어 통신 표준을 따르므로 이러한 정보 교환을 지원합니다.

## **API란 무엇인가요?**

애플리케이션 프로그래밍 인터페이스(API)는 **다른 소프트웨어 시스템과 통신하기 위해 따라야 하는 규칙을 정의**합니다. 개발자는 다른 애플리케이션이 프로그래밍 방식으로 애플리케이션과 통신할 수 있도록 API를 표시하거나 생성합니다. 예를 들어, 근무 시간 기록 애플리케이션은 직원의 전체 이름과 날짜 범위를 요청하는 API를 표시합니다. 이 정보가 수신되면 내부적으로 직원의 근무 시간 기록을 처리하고 해당 날짜 범위에서 근무한 시간을 반환합니다.

웹 API는 클라이언트와 웹 리소스 사이의 게이트웨이라고 생각할 수 있습니다.

### 

### **클라이언트**

클라이언트는 **웹에서 정보에 액세스하려는 사용자**입니다. 클라이언트는 API를 사용하는 사람이거나 소프트웨어 시스템일 수 있습니다. 예를 들어 개발자는 날씨 시스템에서 날씨 데이터에 액세스하는 프로그램을 작성할 수 있습니다. 또는 사용자가 날씨 웹 사이트를 직접 방문할 때 브라우저에서 동일한 데이터에 액세스할 수 있습니다.

### **리소스**

리소스는 다양한 애플리케이션이 **클라이언트에게 제공하는 정보**입니다. 리소스는 이미지, 동영상, 텍스트, 숫자 또는 모든 유형의 데이터일 수 있습니다. 클라이언트에 리소스를 제공하는 시스템을 서버라고도 합니다. 조직은 API를 사용하여 리소스를 공유하고 보안, 제어 및 인증을 유지하면서 웹 서비스를 제공합니다. 또한 API는 특정 내부 리소스에 액세스할 수 있는 클라이언트를 결정하는 데 도움이 됩니다.

## **REST란 무엇인가요?**

Representational State Transfer(REST)는 **API 작동 방식에 대한 조건을 부과하는 소프트웨어 아키텍처**입니다. REST는 처음에 인터넷과 같은 복잡한 네트워크에서 통신을 관리하기 위한 지침으로 만들어졌습니다. REST 기반 아키텍처를 사용하여 대규모의 고성능 통신을 안정적으로 지원할 수 있습니다. 쉽게 구현하고 수정할 수 있어 모든 API 시스템을 파악하고 여러 플랫폼에서 사용할 수 있습니다.

API 개발자는 여러 아키텍처를 사용하여 API를 설계할 수 있습니다. REST 아키텍처 스타일을 따르는 API를 REST API라고 합니다. REST 아키텍처를 구현하는 웹 서비스를 RESTful 웹 서비스라고 합니다. RESTful API라는 용어는 일반적으로 RESTful 웹 API를 나타냅니다. 하지만 REST API와 RESTful API라는 용어는 같은 의미로 사용할 수 있습니다.

다음은 REST 아키텍처 스타일의 몇 가지 원칙입니다.

### **균일한 인터페이스**

균일한 인터페이스는 모든 RESTful 웹 서비스 디자인의 기본입니다. 이는 서버가 표준 형식으로 정보를 전송함을 나타냅니다. 형식이 지정된 리소스를 REST에서 표현이라고 부릅니다. 이 형식은 서버 애플리케이션에 있는 리소스의 내부 표현과 다를 수 있습니다. 예를 들어, 서버는 데이터를 텍스트로 저장하되, HTML 표현 형식으로 전송할 수 있습니다.

균일한 인터페이스에는 4가지 아키텍처 제약 조건이 있습니다.

1. 요청은 리소스를 식별해야 합니다. 이를 위해 균일한 리소스 식별자를 사용합니다.
2. 클라이언트는 원하는 경우 리소스를 수정하거나 삭제하기에 충분한 정보를 리소스 표현에서 가지고 있습니다. 서버는 리소스를 자세히 설명하는 메타데이터를 전송하여 이 조건을 충족합니다.
3. 클라이언트는 표현을 추가로 처리하는 방법에 대한 정보를 수신합니다. 이를 위해 서버는 클라이언트가 리소스를 적절하게 사용할 수 있는 방법에 대한 메타데이터가 포함된 명확한 메시지를 전송합니다.
4. 클라이언트는 작업을 완료하는 데 필요한 다른 모든 관련 리소스에 대한 정보를 수신합니다. 이를 위해 서버는 클라이언트가 더 많은 리소스를 동적으로 검색할 수 있도록 표현에 하이퍼링크를 넣어 전송합니다.

### 

### **무상태**

REST 아키텍처에서 무상태는 서버가 이전의 모든 요청과 독립적으로 모든 클라이언트 요청을 완료하는 통신 방법을 나타냅니다. 클라이언트는 임의의 순서로 리소스를 요청할 수 있으며 모든 요청은 무상태이거나 다른 요청과 분리됩니다. 이 REST API 설계 제약 조건은 서버가 매번 요청을 완전히 이해해서 이행할 수 있음을 의미합니다.

### **계층화 시스템**

계층화된 시스템 아키텍처에서 클라이언트는 클라이언트와 서버 사이의 다른 승인된 중개자에게 연결할 수 있으며 여전히 서버로부터도 응답을 받습니다. 서버는 요청을 다른 서버로 전달할 수도 있습니다. 클라이언트 요청을 이행하기 위해 함께 작동하는 보안, 애플리케이션 및 비즈니스 로직과 같은 여러 계층으로 여러 서버에서 실행되도록 RESTful 웹 서비스를 설계할 수 있습니다. 이러한 계층은 클라이언트에 보이지 않는 상태로 유지됩니다.

### **캐시 가능성**

RESTful 웹 서비스는 서버 응답 시간을 개선하기 위해 클라이언트 또는 중개자에 일부 응답을 저장하는 프로세스인 캐싱을 지원합니다. 예를 들어 모든 페이지에 공통 머리글 및 바닥글 이미지가 있는 웹 사이트를 방문한다고 가정해 보겠습니다. 새로운 웹 사이트 페이지를 방문할 때마다 서버는 동일한 이미지를 다시 전송해야 합니다. 이를 피하기 위해 클라이언트는 첫 번째 응답 후에 해당 이미지를 캐싱하거나 저장한 다음 캐시에서 직접 이미지를 사용합니다. RESTful 웹 서비스는 캐시 가능 또는 캐시 불가능으로 정의되는 API 응답을 사용하여 캐싱을 제어합니다.

### 

### **온디맨드 코드**

REST 아키텍처 스타일에서 서버는 소프트웨어 프로그래밍 코드를 클라이언트에 전송하여 클라이언트 기능을 일시적으로 확장하거나 사용자 지정할 수 있습니다. 예를 들어, 웹 사이트에서 등록 양식을 작성하면 브라우저는 잘못된 전화번호와 같은 실수를 즉시 강조 표시합니다. 서버에서 전송한 코드로 인해 이 작업을 수행할 수 있습니다.

## **RESTful API를 사용하면 어떤 이점이 있나요?**

RESTful API에는 다음과 같은 이점이 있습니다.

### **확장성**

REST API를 구현하는 시스템은 REST가 클라이언트-서버 상호 작용을 최적화하기 때문에 효율적으로 크기 조정할 수 있습니다. 무상태는 서버가 과거 클라이언트 요청 정보를 유지할 필요가 없기 때문에 서버 로드를 제거합니다. 잘 관리된 캐싱은 일부 클라이언트-서버 상호 작용을 부분적으로 또는 완전히 제거합니다. 이러한 모든 기능은 성능을 저하시키는 통신 병목 현상을 일으키지 않으면서 확장성을 지원합니다.

### **유연성**

RESTful 웹 서비스는 완전한 클라이언트-서버 분리를 지원합니다. 각 부분이 독립적으로 발전할 수 있도록 다양한 서버 구성 요소를 단순화하고 분리합니다. 서버 애플리케이션의 플랫폼 또는 기술 변경은 클라이언트 애플리케이션에 영향을 주지 않습니다. 애플리케이션 함수를 계층화하는 기능은 유연성을 더욱 향상시킵니다. 예를 들어, 개발자는 애플리케이션 로직을 다시 작성하지 않고도 데이터베이스 계층을 변경할 수 있습니다.

### **독립성**

REST API는 사용되는 기술과 독립적입니다. API 설계에 영향을 주지 않고 다양한 프로그래밍 언어로 클라이언트 및 서버 애플리케이션을 모두 작성할 수 있습니다. 또한 통신에 영향을 주지 않고 양쪽의 기본 기술을 변경할 수 있습니다.

## **RESTful API는 어떻게 작동하나요?**

RESTful API의 기본 기능은 인터넷 브라우징과 동일합니다. 클라이언트는 리소스가 필요할 때 API를 사용하여 서버에 접속합니다. API 개발자는 서버 애플리케이션 API 문서에서 클라이언트가 REST API를 어떻게 사용해야 하는지 설명합니다. 다음은 모든 REST API 호출에 대한 일반 단계입니다.

1. 클라이언트가 서버에 요청을 전송합니다. 클라이언트가 API 문서에 따라 서버가 이해하는 방식으로 요청 형식을 지정합니다.
2. 서버가 클라이언트를 인증하고 해당 요청을 수행할 수 있는 권한이 클라이언트에 있는지 확인합니다.
3. 서버가 요청을 수신하고 내부적으로 처리합니다.
4. 서버가 클라이언트에 응답을 반환합니다. 응답에는 요청이 성공했는지 여부를 클라이언트에 알려주는 정보가 포함됩니다. 응답에는 클라이언트가 요청한 모든 정보도 포함됩니다.

REST API 요청 및 응답 세부 정보는 API 개발자가 API를 설계하는 방식에 따라 약간씩 다릅니다.

## 

## **RESTful API 클라이언트 요청에는 무엇이 포함되어 있나요?**

RESTful API에는 다음과 같은 주요 구성요소를 포함하는 요청이 필요합니다.

### **고유 리소스 식별자**

서버는 고유한 리소스 식별자로 각 리소스를 식별합니다. REST 서비스의 경우 서버는 일반적으로 URL(Uniform Resource Locator)을 사용하여 리소스 식별을 수행합니다. URL은 리소스에 대한 경로를 지정합니다. URL은 웹페이지를 방문하기 위해 브라우저에 입력하는 웹 사이트 주소와 유사합니다. URL은 요청 엔드포인트라고도 하며 클라이언트가 요구하는 사항을 서버에 명확하게 지정합니다.

### **메서드**

개발자는 종종 Hypertext Transfer Protocol(HTTP)을 사용하여 RESTful API를 구현합니다.   
HTTP 메서드는 리소스에 수행해야 하는 작업을 서버에 알려줍니다.   
다음은 4가지의 일반적인 HTTP 메서드입니다.

***GET***

클라이언트는 GET을 사용하여 서버의 지정된 URL에 있는 리소스에 액세스합니다. GET 요청을 캐싱하고 RESTful API 요청에 파라미터를 넣어 전송하여 전송 전에 데이터를 필터링하도록 서버에 지시할 수 있습니다.

***POST***

클라이언트는 POST를 사용하여 서버에 데이터를 전송합니다. 여기에는 요청과 함께 데이터 표현이 포함됩니다. 동일한 POST 요청을 여러 번 전송하면 동일한 리소스를 여러 번 생성하는 부작용이 있습니다.

***PUT***

클라이언트는 PUT을 사용하여 서버의 기존 리소스를 업데이트합니다. POST와 달리, RESTful 웹 서비스에서 동일한 PUT 요청을 여러 번 전송해도 결과는 동일합니다.

***DELETE***

클라이언트는 DELETE 요청을 사용하여 리소스를 제거합니다. DELETE 요청은 서버 상태를 변경할 수 있습니다. 하지만 사용자에게 적절한 인증이 없으면 요청은 실패합니다.

### **HTTP 헤더**

요청 헤더는 클라이언트와 서버 간에 교환되는 메타데이터입니다. 예를 들어, 요청 헤더는 요청 및 응답의 형식을 나타내고 요청 상태 등에 대한 정보를 제공합니다.

***데이터***

REST API 요청에는 POST, PUT 및 기타 HTTP 메서드가 성공적으로 작동하기 위한 데이터가 포함될 수 있습니다.

***파라미터***

RESTful API 요청에는 수행해야 할 작업에 대한 자세한 정보를 서버에 제공하는 파라미터가 포함될 수 있습니다. 다음은 몇 가지 파라미터 유형입니다.

* URL 세부정보를 지정하는 경로 파라미터.
* 리소스에 대한 추가 정보를 요청하는 쿼리 파라미터.
* 클라이언트를 빠르게 인증하는 쿠키 파라미터.

## **RESTful API 인증 방법이란 무엇인가요?**

RESTful 웹 서비스는 응답을 보내기 전에 먼저 요청을 인증해야 합니다. 인증은 신원을 확인하는 프로세스입니다. 예를 들어, 신분증이나 운전면허증을 제시하여 신원을 증명할 수 있습니다. 마찬가지로 RESTful 서비스 클라이언트는 신뢰를 구축하기 위해 서버에 자신의 신원를 증명해야 합니다.

RESTful API에는 4가지의 일반적인 인증 방법이 있습니다.

### **HTTP 인증**

HTTP는 REST API를 구현할 때 직접 사용할 수 있는 일부 인증 체계를 정의합니다. 다음은 이러한 체계 중 두 가지입니다.

***기본 인증***

기본 인증에서 클라이언트는 요청 헤더에 사용자 이름과 암호를 넣어 전송합니다. 안전한 전송을 위해 이 페어를 64자의 세트로 변환하는 인코딩 기술인 base64로 인코딩합니다.

***전달자 인증***

전달자(bearer) 인증이라는 용어는 토큰 전달자에 대한 액세스 제어를 제공하는 프로세스를 나타냅니다. 일반적으로 전달자 토큰은 서버가 로그인 요청에 대한 응답으로 생성하는 암호화된 문자열입니다. 클라이언트는 리소스에 액세스하기 위해 요청 헤더에 토큰을 넣어 전송합니다.

### **API 키**

API 키는 REST API 인증을 위한 또 다른 옵션입니다. 이 접근 방식에서 서버는 고유하게 생성된 값을 최초 클라이언트에 할당합니다. 클라이언트는 리소스에 액세스하려고 할 때마다 고유한 API 키를 사용하여 본인을 검증합니다. API 키의 경우 클라이언트가 이 키를 전송해야 해서 네트워크 도난에 취약하기 때문에 덜 안전합니다.

### **OAuth**

OAuth는 모든 시스템에 대해 매우 안전한 로그인 액세스를 보장하기 위해 암호와 토큰을 결합합니다. 서버는 먼저 암호를 요청한 다음 권한 부여 프로세스를 완료하기 위해 추가 토큰을 요청합니다. 특정 범위와 수명으로 언제든지 토큰을 확인할 수 있습니다.

## 

## **RESTful API 서버 응답에는 무엇이 포함되어 있나요?**

REST 원칙에 따라 서버 응답에 다음과 같은 주요 구성 요소를 포함해야 합니다.

### **상태 표시줄**

상태 표시줄에는 요청 성공 또는 실패를 알리는 3자리 상태 코드가 있습니다. 예를 들어, 2XX 코드는 성공을 나타내고 4XX 및 5XX 코드는 오류를 나타냅니다. 3XX 코드는 URL 리디렉션을 나타냅니다.

다음은 몇 가지 일반적인 상태 코드입니다.

* 200: 일반 성공 응답
* 201: POST 메서드 성공 응답
* 400: 서버가 처리할 수 없는 잘못된 요청
* 404: 리소스를 찾을 수 없음

### 

### **메시지 본문**

응답 본문에는 리소스 표현이 포함됩니다. 서버는 요청 헤더에 포함된 내용을 기반으로 적절한 표현 형식을 선택합니다. 클라이언트는 데이터 작성 방식을 일반 텍스트로 정의하는 XML 또는 JSON 형식으로 정보를 요청할 수 있습니다. 예를 들어, 클라이언트가 John이라는 사람의 이름과 나이를 요청하면 서버는 다음과 같이 JSON 표현을 반환합니다.

'{"name":"John", "age":30}'

### **헤더**

응답에는 응답에 대한 헤더 또는 메타데이터도 포함됩니다. 이는 응답에 대한 추가 컨텍스트를 제공하고 서버, 인코딩, 날짜 및 콘텐츠 유형과 같은 정보를 포함합니다.

((끝))