**서버 기초와 API 설계**

**서버의 기본 개념**

* ﻿﻿서버는 네트워크를 통해 클라이언트로부터 요청을 받아 처리하고, 그에 따른 결과를 클라이언트에게 제공하는 컴퓨터 시스템입니다.
* ﻿﻿데이터를 저장하고 관리하며, 클라이언트가 필요로 하는 서비스나 데이터를 제공하는 역할을 수행합니다.

**서버의 주요 역할**

* ﻿﻿**데이터 제공:** 클라이언트가 요청하는 데이터나 서비스를 제공합니다. 예를 들어, 웹 서버는 웹 페이지나 멀티미디어 파일을 클라이언트에게 전송합니다.
* ﻿﻿**서비스 제공:** 다양한 서비스를 제공하며, 이메일 서버는 이메일의 전송과 수신을 담당합니다.
* ﻿﻿**자원 관리:** 서버는 하드웨어 자원(예: CPU, 메모리, 디스크 공간)을 관리하고, 여러 클라이언트 간에 자원을 공유하거나 제어합니다.

﻿﻿**네트워크 통신 관리:** 클라이언트와의 네트워크 통신을 관리하여 안정적이고 효율적인 데이터 전송을 보장합니다.

* ﻿**서버 종류:** 다양한 종류의 서버가 있으며, 각각의 서버는 특정한 기능을 수행합니다.
* ﻿﻿**웹서버:** HTTP 프로토콜을 사용하여 웹 페이지를 제공하는 서버입니다. (예: Apache, Nginx, React)
* ﻿﻿**데이터베이스 서버:** 데이터베이스 관리 시스템(DBMS)을 통해 데이터 저장과 관리를 담당하는 서버입니다. (예: MySQL, PostgresQL)
* ﻿﻿**애플리케이션 서버:** 비즈니스 로직을 실행하고 데이터 처리를 수행하는 서버입니다. (예: Tomcat, JBoss, Node js, Java Spring)
* ﻿﻿**파일 서버:** 파일을 저장하고 다른 사람과 공유할 수 있는 서버입니다.

﻿﻿메일 서버: 이메일 송수신을 관리하는 서버입니다.

**웹 서버의 구성 요소**

* ﻿﻿**웹 서버 소프트웨어:** Apache, Nginx, IIS 등 클라이언트 요청을 처리하는 소프트웨어입니다.
* ﻿﻿**애플리케이션 서버:** 동적 콘텐츠를 생성하기 위해 서버사이드 스크립트를 실행하는 환경입니다. 예: Tomcat, Node.js.
* ﻿﻿**데이터베이스 서버:** 데이터를 저장하고 관리하는 시스템입니다. 웹 서버는 데이터베이스와 상호작용하여 클라이언트 요청에 필요한 데이터를 가져옵니다.

﻿﻿**파일 시스템:** 웹 서버는 파일 시스템을 통해 정적 콘텐츠를 클라이언트에게 제공하며, 로그 파일과 설정 파일 등을 저장합니다.

**서버 가용성과 성능**

서버 가용성과 성능은 웹 애플리케이션 및 서비스의 성공적인 운영에 있어 중요한 요소입니다. 두 개념은 상호 보완적이며, 각각의 특성과 중요성을 이해하고 관리하는 것이 필수적입니다.

* ﻿﻿**서버 가용성**
* ﻿﻿**정의:** 서버 가용성은 서버가 정상적으로 운영되고 사용자 요청에 응답할 수 있는 능력을 의미합니다.
* ﻿﻿**목표:** 최대한의 업타임을 유지하여 사용자가 언제든지 서비스에 접근할 수 있도록 하는 것.
* ﻿﻿**서버 성능**
* ﻿﻿**정의:** 서버 성능은 서버가 요청을 처리하는 속도와 효율성을 의미합니다.
* ﻿﻿**목표**: 요청에 대한 신속하고 일관된 응답을 제공하는 것.

﻿﻿두 요소는 균형을 이루어야 합니다. 높은 가용성만으로는 성능 저하를 막을 수 없고, 반대로 높은 성능만으로는 다운타임을 방지할 수 없습니다.

**서버 상태 모니터링 하는 방법 모니터링 도구 활용**

* ﻿﻿Prometheus & Grafana
* ﻿﻿Prometheus: 시계열 데이터베이스로, 서버 성능 지표를 수집하고 저장.
* ﻿﻿Grafana: 시각화 도구로, Prometheus 데이터를 기반으로 대시보드 작성.
* ﻿﻿Nagios
* ﻿﻿서버, 네트워크, 애플리케이션 모니터링을 위한 오픈소스 도구.
* ﻿﻿Zabbix
* ﻿﻿서버와 네트워크 장비 모니터링을 위한 오픈소스 플랫폼.
* ﻿﻿New Relic

﻿﻿애플리케이션 성능 모니터링 도구로, 서버 성능과 애플리케이션 상태를 모니터링.

**서버 상태 모니터링 하는 방법**

로그 분석

* ﻿﻿ELK Stack (Elasticsearch, Logstash, Kibana)
* ﻿﻿Elasticsearch: 로그 데이터를 저장하고 검색.
* ﻿﻿Logstash: 로그 데이터를 수집하고 변환.
* ﻿﻿Kibang: 로그 데이터를 시각화.
* ﻿﻿Graylog
* ﻿﻿실시간 로그 관리 및 분석 도구.

시스템 내장 도구 사용

* ﻿﻿top/htop: 실시간으로 CPU, 메모리 사용량, 프로세스 상태 모니터링.
* ﻿﻿vmstat: 시스템 성능과 관련된 다양한 메트릭 제공.
* ﻿﻿iostat: 디스크 1/0와 CPU 사용량 모니터링.

﻿﻿netstat: 네트워크 연결 상태와 통계 정보 제공.

**다양한 통신 프로토콜**

**클라이언트와 서버의 통신을 위한 프로토콜 종류**

* ﻿﻿**프로토콜의 정의**
* ﻿﻿통신 프로토콜은 네트워크에서 데이터를 주고받기 위한 규칙과 절차의 집합입니다.
* ﻿﻿**주요 기능**
* ﻿﻿**데이터 포맷 지정:** 데이터를 어떤 형식으로 전송할지 정의합니다.
* ﻿﻿**전송 절차 규정:** 데이터 전송 절차와 흐름을 규정합니다.
* ﻿﻿**에러 검출 및 수정:** 데이터 전송 중 발생할 수 있는 오류를 검출하고 수정합니다.
* ﻿﻿**동기화:** 송신자와 수신자 간의 동기화를 유지합니다.
* ﻿﻿**압축:** 데이터 전송을 효율적으로 하기 위해 데이터를 압축합니다.
* ﻿﻿**프로토콜 예시**
* ﻿﻿**HTTP/HTTPS:** 웹 브라우저와 서버 간 데이터 전송에 사용됩니다.
* ﻿﻿**Websocket:** 클라이언트와 서버 간 양방향 실시간 통신을 지원합니다.
* ﻿﻿**FTP:** 파일 전송을 위한 프로토콜입니다.

﻿﻿**SSH:** 원격 서버에 안전하게 접속하기 위한 프로토콜입니다.

**HTTP/HTTPS**

* **﻿﻿HTTP**
* ﻿﻿웹 서버와 웹 브라우저 사이에서 데이터를 전송하기 위한 프로토콜입니다.
* ﻿﻿클라이언트가 서버에 요청을 보내고, 서버가 이에 대해 응답을 반환합니다.
* ﻿﻿주로 웹 페이지, 이미지, 동영상 등의 리소스를 클라이언트에게 전달합니다.
* ﻿﻿HTTP 요청 메서드: GET, POST, PUT, DELETE, PATCH 등
* ﻿﻿**HTTPS**
* ﻿﻿HTTP의 보안 버전으로, 데이터 전송 시 TLS(Transport Layer Security) 또는 SSL(Secure Sockets Layer) 프로토콜을 사용하여 암호화합니다.
* ﻿﻿데이터 무결성, 기밀성, 인증을 보장하여 중간에서 데이터를 가로채거나 변조하는 것을 방지합니다.
* ﻿﻿HTTPS는 웹사이트의 신뢰성을 높이고 사용자 데이터를 보호하기 위해 사용됩니다.

﻿﻿HTTPS 프로토콜을 지원하지 않는 웹 페이지는 보안 위험이 있으므로 이용하지 않는 것이 좋습니다.

**HTTP/HTTPS: SSL/TLS**

* ﻿﻿SSL은 인터넷 통신의 보안을 위해 개발된 프로토콜입니다.
* ﻿﻿TLS는 SSL의 후속 프로토콜로, 보안성과 효율성을 개선한 버전입니다.
* ﻿﻿작동 방식

i. **핸드셰이크 과정:**

* ﻿﻿**클라이언트 헬로:** 클라이언트가 서버에 연결 요청을 보내며, 지원하는 암호화 알고리즘 목록을 전송합니다.
* ﻿﻿**서버 헬로:** 서버가 클라이언트의 요청에 응답하며, 사용할 암호화 알고리즘을 선택하고, 서버 인증서를 전송합니다.
* ﻿﻿**키 교환**: 클라이언트와 서버가 세션 키를 생성하기 위한 정보를 교환합니다.
* ﻿﻿**세션 키 생성**: 클라이언트와 서버는 교환한 정보를 바탕으로 동일한 세션 키를 생성합니다.

**ii. 암호화된 통신**: 핸드 셰이크가 완료된 후, 클라이언트와 서버 간의 모든 데이터는 세션 키를 사용하여 암호화됩니다.

**HTTP/HTTPS: 요청 구조**

**1. 요청 라인(Request Line)**

* ﻿﻿**메소드:** HTTP 요청의 종류를 나타냅니다.
* ﻿﻿**URL:** 요청 대상 리소스의 경로입니다.

﻿﻿**HTTP 버전:** 사용되는 HTTP 프로토콜의 버전입니다.

GET /users HTTP/1.1

**2. 헤더(Headers)**

* ﻿﻿요청에 대한 메타데이터를 포함합니다.
* ﻿﻿각 헤더는 이름: 값 형식으로 작성됩니다.
* ﻿﻿자주 사용되는 요청 헤더
* ﻿﻿Host: 요청하는 호스트의 도메인 이름과 포트 번호입니다.
* ﻿﻿User-Agent: 요청을 보낸 클라이언트의 소프트웨어 정보입니다.
* ﻿﻿Content-Type: 요청 본문의 데이터 타입을 지정합니다.
* ﻿﻿Authorization: 인증 토큰이나 자격 증명을 포함합니다.

Host: www.example.com

User-Agent: Mozilla/5.0

Content-Type: application/x-www-form-urlencoded

3. 바디(Body)

* ﻿﻿요청에 대한 데이터를 포함합니다.
* ﻿﻿GET 요청에는 바디가 없으며, POST, PUT 요청에서는 주로 사용됩니다.

﻿﻿예시: 폼 데이터, JSON, XML 등

name=John&age=30

**HTTP/HTTPS: 응답 구조**

**1. 상태 라인(Status Line)**

* ﻿﻿**HTTP 버전**: 사용되는 HTTP 프로토콜의 버전입니다.
* ﻿﻿**상태 코드**: 요청에 대한 서버의 응답 상태를 나타냅니다. (예: 200, 404, 500)
* ﻿﻿**상태 메시지:** 상태 코드에 대한 설명입니다.

(04: OK, Not Found, Internal Server Error)

HTTP/1.1 200 OK

**2. 헤더(Headers)**

﻿﻿응답에 대한 메타데이터를 포함합니다. 각 헤더는 "이름: 값" 형식으로 작성됩니다.

* ﻿﻿자주 사용되는 응답 헤더
* ﻿﻿Content-Type: 응답 본문의 데이터 타입을 지정합니다.

(0): text/html, application/json)

* ﻿﻿Content-Length: 응답 본문의 길이를 바이트 단위로 지정합니다.
* ﻿﻿Set-Cookie: 서버가 클라이언트에게 설정할 쿠키를 지정합니다.

Content-Type: text/html

Content-Length: 138

Set-Cookie: sessionId=abc123; HttpOnly

**3. 바디(Body)**

* ﻿﻿응답에 대한 데이터를 포함합니다.

﻿﻿웹 페이지 HTML, JSON, 이미지 등 다양한 형태로 전송될 수 있습니다.

{"result": true, "todos": [] }

**WebSocket**

* ﻿﻿WebSocket은 TCP 기반의 실시간 양방향 통신을 제공하는 컴퓨터 통신 프로토콜입니다.
* ﻿﻿웹 애플리케이션과 웹 서버 간의 지속적인 연결을 가능하게 하여 실시간 데이터 전송이 필요한 애플리케이션에 적합합니다.
* ﻿﻿통신 과정
* ﻿﻿**Handshake:** 클라이언트가 서버에 WebSocket 연결을 요청하고, 서버가 요청을 수락하여 WebSocket 연결을 설정합니다.
* ﻿﻿**양방향 데이터 전송:** 연결이 설정된 후에는 클라이언트와 서버 간 양방향으로 데이터를 전송할 수 있습니다.
* ﻿﻿**연결 유지:** 연결이 설정된 후에는 지속적으로 데이터를 주고받을 수 있어 실시간 게임, 채팅 등에 적합합니다.
* ﻿﻿실시간 데이터 전송이 필요한 다양한 웹 애플리케이션에 적합합니다.

﻿﻿WebSocket은 SSL/TLS를 통해 암호화할 수 있어 보안이 강화될 수 있습니다.

FTP

* ﻿﻿FTP는 파일을 네트워크를 통해 전송하기 위한 표준 프로토콜입니다.
* ﻿﻿**FTP 모드**
* ﻿﻿**액티브 모드**
* ﻿﻿클라이언트가 서버의 포트 21에 연결을 시작합니다.
* ﻿﻿클라이언트는 데이터 전송을 위해 자신의 임의의 포트를 서버에 알립니다.
* ﻿﻿서버는 클라이언트가 알린 포트로 데이터 연결을 시도합니다.
* ﻿﻿**장점:** 설정이 간단하고, NAT 이전 네트워크에서 주로 사용됩니다.
* ﻿﻿**단점:** 클라이언트의 방화벽이 서버의 연결 요청을 막을 수 있습니다.
* ﻿﻿**패시브 모드**
* ﻿﻿클라이언트가 서버의 포트 21에 연결을 시작합니다.
* ﻿﻿서버는 데이터 전송을 위해 자신의 임의의 포트를 클라이언트에 알립니다.
* ﻿﻿클라이언트는 서버가 알린 포트로 데이터 연결을 시도합니다.
* ﻿﻿**장점:** 클라이언트의 방화벽 문제를 피할 수 있습니다.

﻿﻿**단점:** 서버의 방화벽 설정이 필요할 수 있습니다.

**SSH**

* ﻿﻿SSH는 네트워크 상에서 안전하게 통신하기 위한 프로토콜이며, 원격 접속을 위한 프로토콜입니다.
* ﻿﻿주로 터미널 세션에서 보안된 방식으로 명령을 실행하거나 파일을 전송하는 데 사용됩니다.
* ﻿﻿통신 과정
* ﻿﻿**인증**: 클라이언트가 서버에 접속 요청을 보내면, 서버는 클라이언트의 신원을 확인합니다.
* ﻿﻿**암호화된 세션:** 접속이 승인되면 클라이언트와 서버 간의 데이터 전송은 암호화된 채널을 통해 이루어집니다.
* ﻿﻿**명령 실행:** SSH 세션을 통해 원격 서버에서 명령을 실행하거나 파일을 전송할 수 있습니다.

﻿﻿데이터 전송 과정에서 암호화를 사용하여 보안을 강화합니다.

**요청 플로우와 데이터 구조**

**일반적인 클라이언트 요청과 서버 응답 플로우**

**1. 클라이언트(Client)**

• 사용자 인터페이스를 통해 요청을 생성하고 서버로 전송합니다.

**2. 컨트롤러(Controller)**

* ﻿﻿클라이언트의 요청을 받아 처리합니다.
* ﻿﻿필요한 데이터를 서비스로 전달하고, 서비스에서 반환된 데이터를 클라이언트에게 전달합니다.

**3. 서비스(Service)**

* ﻿﻿비즈니스 로직을 처리합니다.
* ﻿﻿데이터베이스와의 상호작용을 담당하며, 데이터 처리와 관련된 모든 작업을 수행합니다.

**4. 모델(Model)**

* ﻿﻿데이터의 구조를 정의하고, 데이터베이스에서 읽거나 저장할 때 사용하는 객체입니다.

﻿﻿데이터의 유효성 검사를 포함할 수 있습니다.

모듈식구조**스크린샷, 소프트웨어, 멀티미디어 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명**

계층식구조**스크린샷, 소프트웨어, 멀티미디어 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명**

**데이터 전달 구조**

**DTO (Data Transfer Object)**

* ﻿﻿클라이언트와 서버 간 데이터 교환을 위한 객체입니다.
* ﻿﻿주로 데이터 전송에 필요한 필드와 메서드를 포함하며, 데이터 전송의 효율성과 보안을 고려하여 설계됩니다.
* ﻿﻿DTO는 클라이언트가 전송한 데이터를 서버에서 사용할 수 있도록 변환하거나, 서버가 클라이언트에게 응답할 데이터를 구성하는 데

﻿﻿데이터 전송의 효율성과 일관성을 유지하는데 이점이 있습니다.

**Entity와 Schema**

* ﻿﻿데이터베이스에 저장되는 데이터를 표현하는 객체입니다.
* ﻿﻿주로 데이터베이스 테이블의 구조와 일치하며, 데이터베이스의 각 행을 객체로 표현합니다.
* ﻿﻿서버에서 데이터를 처리할 때 데이터의 유효성을 검증하거나, 데이터의 구조적 변경을 반영하는 데 활용됩니다.
* ﻿﻿Schema는 데이터베이스 자료구조를 정의하는 개념이고, Entity는 소프트웨어에서 데이터 모델을 표현하는 객체입니다.

﻿﻿따라서 어떤 이름을 사용해도 큰 문제 없지만, 일반적으로 ORM을 사용할 경우 ORM에 맞추어 파일명을 작성하는 것이 좋습니다.

**API 엔드포인트 설계**

* ﻿﻿API 엔드포인트는 클라이언트가 서버의 리소스에 접근하기 위한 URL을 말합니다.
* ﻿﻿클라이언트와 서버 간의 통신 인터페이스를 제공하는 역할을 합니다.
* ﻿﻿즉, API 엔드포인트를 설계한다는 것은 클라이언트와 어떤 방식으로 통신을 주고 받을지를 정의하는 과정을 나타냅니다.
* ﻿﻿설계 시 고려사항
* ﻿﻿RESTful 원칙 준수 및 명확하고 일관된 명명 규칙
* ﻿﻿보안 및 성능 최적화

﻿﻿버전 관리 및 하위 호환성

**리소스 기반 설계: Restful API**

* ﻿﻿ApI 엔드포인트는 주로 리소스(resource)를 기반으로 설계됩니다.
* ﻿﻿예를 들어, 사용자, 제품, 주문과 같은 도메인 개체를 리소스로 정의하고 이를 기준으로 엔드포인트를 구성합니다.
* ﻿﻿엔드포인트의 경로는 명확하고 일관성 있게 설계되어야 합니다.
* ﻿﻿일반적으로 리소스를 나타내는 경로와 HTTP 메서드를 조합하여 정의합니다.

﻿﻿리소스 간 관계를 URL 구조에 반영하여 계층적으로 표현합니다.

**파라미터**

* ﻿﻿파라미터는 URL에 포함되는 변수로, 특정 리소스에 접근하거나 식별하기 위해 사용됩니다.
* ﻿﻿주로 RESTful API에서 리소스의 고유 식별자를 전달하기 위해 사용됩니다.
* ﻿﻿파라미터를 통해 리소스 계층 구조를 명확히 표현하고 정확한 리소스를 식별할 수 있습니다.
* ﻿﻿잘못된 파라미터를 처리할 수 있는 로직이 필요할 수 있습니다. (예. 404 에러 등)
* router.get("/users/:id")i

GET /users/123

**쿼리 스트링**

* ﻿﻿쿼리 스트링은 URL의 끝에 ? 문자와 함께 추가되며, key=value 쌍으로 이루어져 있습니다.
* ﻿﻿여러 개의 쿼리 스트링 파라미터는 & 기호로 구분됩니다.
* ﻿﻿주로 필터링, 정렬, 페이지네이션 등의 데이터를 전달할 때 사용됩니다.

﻿﻿쿼리 스트링은 주로 HTTP GET 요청에서 사용됩니다. GET /users?role=admin&sort=name // role이 'admin'인 모든 사용자를 이름(name)으로 정렬하여 조회합니다. GET /products?page=2&limit=10 // 두 번째 페이지(page=2)에 해당하는 10개의 제품(limit=10)을 조회합니다.

**파라미터와 쿼리 스트링**

* ﻿﻿**비교**
* ﻿﻿파라미터는 URL 경로의 일부로, 리소스 식별자와 연결됩니다. = 쿼리 스트링은 선택적인 매개변수로, 요청에 추가적인 정보를 제공합니다.
* ﻿﻿**인코딩**
* ﻿﻿URL에서 사용되는 파라미터와 쿼리 스트링은 UTF-8로 인코딩되어야 합니다.
* ﻿﻿특수 문자나 공백 등은 URL 인코딩을 통해 안전하게 전달되어야 합니다.

﻿﻿자바스크립트에서 encodeURIComponent 혹은 ancodeURIComponent 함수를 활용해 보세요.

**API 엔드포인트 설계 순서**

* ﻿﻿﻿**리소스 식별:** API가 제공할 주요 리소스를 식별합니다. 이는 사용자, 제품, 주문 등과 같은 비즈니스 개체를 포함합니다.
* ﻿﻿﻿**HTTP 메소드 정의:** 각 리소스에 대해 지원할 HTTP 메서드를 정의합니다.
* ﻿﻿﻿**URI 설계:** 각 리소스와 연결된 URI를 설계합니다.
* ﻿﻿﻿**파라미터 및 쿼리 스트링 정의:** 필요한 경우 경로 파라미터와 쿼리 스트링을 정의합니다.
* ﻿﻿﻿**응답 포맷 정의:** API가 반환할 데이터의 형식(JSON, XML 등)을 정의하고, 각 엔드포인트의 예상 응답과 상태 코드를 정의합니다.
* ﻿﻿﻿**개발:** 설계한 내용에 따라 라우터를 작성합니다.

﻿﻿﻿**문서화:** API를 사용하는 개발자들이 이해하기 쉽도록 각 엔드포인트의 사용 방법, 요청 및 응답 예시를 포함한 문서를 작성합니다.

**API 문서화 방법**

**API 문서 작성의 목적**

* ﻿﻿**ApI 이해 돕기:** API 문서는 API가 무엇을 하는지, 어떤 기능을 제공하는지를 명확히 설명하여 개발자들이 API의 작동 원리와 사용 방법을 이해할 수 있도록 합니다.
* ﻿﻿**사용법 제공:** API 문서는 각 엔드포인트의 경로, 사용 가능한 HTTP 메서드, 필요한 파라미터와 쿼리 스트링 등을 포함하여 API를 어떻게 호출해야 하는지에 대한 구체적인 사용법을 제공합니다.
* ﻿﻿**오류 예방:** API 호출 시 발생할 수 있는 오류 상황과 오류 코드를 설명하여 개발자가 문제를 예방하고 해결하는 데 도움을 줍니다.

﻿﻿**버전 관리:** API의 버전 관리 정보를 포함하여, 새로운 버전이 출시될 때 기존 사용자들이 변경 사항을 쉽게 파악하고 업데이트할 수 있도록 돕습니다.

**API 문서 작성 방법: Swagger / OpenAPI 활용**

* ﻿﻿Swagger는 API를 설계, 빌드, 문서화할 수 있는 도구와 스펙이며, OpenApI Specification(OAS)을 사용하여 API를 설명하는 표준입니다.
* ﻿﻿자동화된 문서 생성: Swagger ui를 통해 API 엔드포인트, 요청 및 응답 예시를 시각적으로 표현할 수 있습니다.
* ﻿﻿API 테스트: Swagger uI에서 API를 직접 테스트하고 결과를 확인할 수 있습니다.

﻿﻿표준화된 형식: OpenApI Specification을 통해 API 설계의 일관성과 가독성을 유지할 수 있습니다.

**API 문서 작성 방법: Code to Swagger/OpenAPI**

* ﻿﻿프로젝트에 작성된 코드로부터 Swagger 또는 OpenApI 문서를 자동으로 생성할 수 있습니다.
* ﻿﻿이러한 자동 생성은 대부분의 언어에서 라이브러리로 제공되는 기능입니다.
* ﻿﻿Java Spring: SpringFox , SpringDoc
* ﻿﻿Node js: swagger-jsdoc
* ﻿﻿Python Flask: flask\_restplus
* ﻿﻿**장점**
* **﻿﻿일관성 유지**: 코드와 문서의 일관성을 유지하여 문서화의 정확성을 높일 수 있습니다.
* ﻿﻿자동화: 문서화를 자동화하여 개발자가 수동으로 문서를 작성하고 유지보수하는 부담을 줄일 수 있습니다.
* ﻿﻿**고려 사항**

﻿﻿자동 생성된 문서가 모든 세부 사항을 포괄하지 않을 수 있으므로, 추가적인 설명이나 예시를 수동으로 보완할 필요가 있습니다.

**JSON 과 XML 데이터 포맷**

**JSON (JavaScript Object Notation)**

* ﻿﻿JSON은 경량의 데이터 교환 형식으로, 사람과 기계가 모두 이해하기 쉽도록 설계된 텍스트 기반의 데이터 형식입니다.
* ﻿﻿가독성: 텍스트 기반으로, 사람이 쉽게 읽고 쓸 수 있습니다.
* ﻿﻿간결성: 불필요한 태그가 없고, 구조가 단순하여 데이터 크기가 작습니다.
* ﻿﻿편의성: 다양한 프로그래밍 언어에서 지원하며, 파싱이 용이합니다.

﻿﻿데이터 타입: 문자열, 숫자, 배열, 객체, 불리언, null 등 다양한 데이터 타입을 지원합니다.

**XML (eXtensible Markup Language)**

* ﻿﻿XML은 다목적 마크업 언어로, 데이터를 구조화하고 전송하는 데 사용되는 텍스트 기반의 형식입니다.
* ﻿﻿확장성: 사용자가 자신만의 태그를 정의하여 데이터를 구조화할 수 있습니다.
* ﻿﻿복잡성: 더 많은 태그를 필요로 하며, 데이터 크기가 상대적으로 크고 파싱이 복잡할 수 있습니다.

﻿﻿메타데이터: 태그를 사용하여 데이터에 메타데이터를 추가할 수 있습니다.