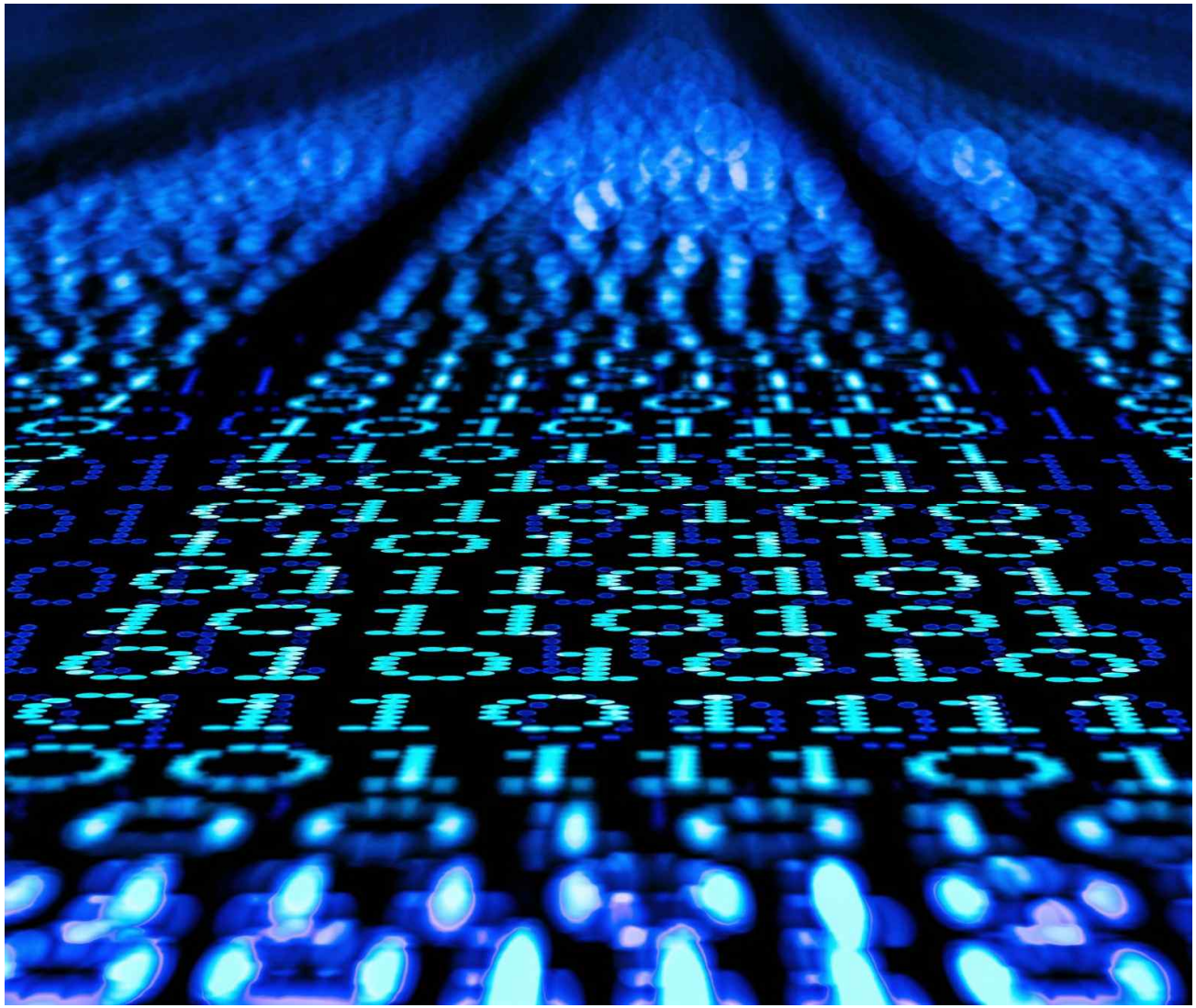

REST API 기초

HTTP 기초



HTTP 기초

[HTTP 기초]

1. HTTP란?

- **HTTP(HyperText Transfer Protocol)**는 클라이언트(예: 브라우저)와 서버 간의 데이터를 주고받기 위한 프로토콜입니다.
- 주로 **웹 페이지**, **이미지**, **JSON 데이터** 등의 리소스를 요청하고 응답받는 데 사용됩니다.
- **요청(Request)**과 **응답(Response)**이라는 두 가지 기본 구조로 이루어져 있습니다.

HTTP 기초

1. REST란?

- **REST(Representational State Transfer)**는 **웹 애플리케이션 설계 원칙** 중 하나로, HTTP 프로토콜을 기반으로 리소스(Resource)를 처리하기 위한 방법론입니다.
- **API(Application Programming Interface)**와 결합되어, 서버와 클라이언트 간의 데이터 통신 방식을 정의합니다.

HTTP 기초

2. REST의 핵심 요소

- **리소스(Resource):**
  - URL(엔드포인트)로 리소스를 식별합니다.
  - 예: `https://example.com/users` → 사용자 리소스를 나타냄.
- **표현(Representation):**
  - 리소스의 상태를 JSON, XML 등의 형식으로 표현합니다.
  - 예: 사용자 정보를 JSON 형태로 제공 → `{ "id": 1, "name": "Jane" }`
- **HTTP 메서드(Method):**
  - 리소스에 대해 어떤 작업을 할지 정의합니다. (아래에서 자세히 설명)

HTTP 기초

HTTP 메서드와 CRUD 연산:

- **GET:** 리소스 조회 (READ)
- **POST:** 새로운 리소스 생성 (CREATE)
- **PUT:** 기존 리소스 전체 업데이트 (UPDATE)
- **PATCH:** 기존 리소스 부분 업데이트 (PARTIAL UPDATE)
- **DELETE:** 리소스 삭제 (DELETE)

HTTP 기초

[HTTP 메서드 이해]

HTTP 메서드는 클라이언트가 서버에 요청할 작업의 유형을 나타냅니다. 주로 REST API와 함께 사용됩니다.

1. GET

- **용도:** 리소스를 조회.
- **특징:** 서버에 데이터를 변경하지 않음(읽기 전용).
- **예:**
  - 요청: `GET /users/1`
  - 응답: `{ "id": 1, "name": "Jane" }`

HTTP 기초

2. POST

- **용도:** 리소스를 생성.
- **특징:** 서버에 데이터를 추가합니다.
- **예:**
  - 요청: `POST /users`

```json
{
  "name": "Jane",
  "email": "jane@example.com"
}
```

  - 응답: `{ "id": 1, "name": "Jane" }`

HTTP 기초

3. PUT

- **용도:** 리소스를 완전히 업데이트.
- **특징:** 리소스 전체를 교체합니다.
- **예:**
  - 요청: `PUT /users/1`

```json
{
  "name": "Jane Doe",
  "email": "jane.doe@example.com"
}
```

  - 응답: `{ "id": 1, "name": "Jane Doe" }`

HTTP 기초

4. PATCH

- **용도:** 리소스의 일부를 업데이트.
- **특징:** 변경하려는 데이터만 전송합니다.
- **예:**
  - 요청: `PATCH /users/1`

```json
{
  "name": "Jane Doe"
}
```

  - 응답: `{ "id": 1, "name": "Jane Doe" }`

HTTP 기초

5. DELETE

- **용도:** 리소스를 삭제.
- **특징:** 서버에서 특정 리소스를 제거합니다.
- **예:**
  - 요청: `DELETE /users/1`
  - 응답: `{ "message": "User deleted successfully." }`

HTTP 기초

REST의 주요 원칙:

- **무상태성(Stateless):** 각 요청은 독립적으로 처리되며, 서버는 이전 요청의 상태를 저장하지 않습니다.
- **클라이언트-서버 구조:** 클라이언트와 서버가 분리되어 있으며, 클라이언트는 API를 통해 서버의 리소스에 접근합니다.
- **캐시 가능성(Cacheable):** 응답 데이터는 클라이언트가 캐싱할 수 있으며, 이를 통해 성능을 최적화할 수 있습니다.
- **계층화 시스템(Layered System):** 클라이언트는 서버와 직접 통신하지 않고, 중간 계층(프록시, 게이트웨이 등)을 거쳐 통신할 수 있습니다.
- **인터페이스 일관성(Uniform Interface):** 리소스에 접근하는 표준화된 방법을 제공합니다.

HTTP 기초

URI(Uniform Resource Identifier)는 리소스를 식별하는 경로입니다. RESTful API는 **명확하고 일관된 URI**를 사용하여 리소스를 식별해야 합니다.

- **URI 구조:**
  - 고유한 리소스를 식별하기 위한 주소
  - 예: `https://api.example.com/users/123`

1. 명사 사용

- URI에는 명사를 사용하여 리소스를 나타냅니다.
- **예시:** `/users`, `/orders`

2. 복수형 사용

- 리소스는 복수형으로 표현하여 일관성을 유지합니다.
- **예시:** `/posts`, `/comments`

3. 계층적 구조 사용

- 리소스 간의 관계를 나타내기 위해 계층적 구조를 사용합니다.
- **예시:** `/posts/{post_id}/comments`

REST API 호출

REST API 호출

1. 파이썬으로 REST API 사용 준비

- **환경 설정:**
  - Python 설치 및 버전 확인 (3.7 이상 권장)
  - 필수 라이브러리 설치: `pip install requests`
- **주요 라이브러리:**
  - **Requests:** HTTP 요청을 쉽게 처리하기 위한 라이브러리.
  - **JSON:** 응답 데이터를 파싱하고 처리.
- **개발 도구 추천:**
  - PyCharm, Visual Studio Code, Jupyter Notebook 등

REST API 호출

2. HTTP 요청의 종류와 사용법

- **GET 요청:**
  - 서버에서 데이터 가져오기
  - 코드 예시:

```python
import requests
response = requests.get('https://jsonplaceholder.typicode.com/posts')
print(response.json())
```

REST API 호출

- **POST 요청:**
  - 새로운 데이터 생성
  - 코드 예시:

python 복사 편집

```python
data = {'title': 'New Post', 'body': 'This is a new post.', 'userId': 1}
response = requests.post('https://jsonplaceholder.typicode.com/posts', json=data)
print(response.json())
```

REST API 호출

- **오류 처리:**
  - 상태 코드 확인: `response.status_code`
  - 예: 404 (Not Found), 500 (Internal Server Error)

REST API 호출

3. 응답 데이터 처리 및 활용

- **JSON 데이터 파싱:**
  - API 응답은 일반적으로 JSON 형식.
  - 예제:

```python
response = requests.get('https://jsonplaceholder.typicode.com/posts/1')
data = response.json()
print(data['title'])
```

REST API 호출

4. REST API의 주요 HTTP 응답 코드

2xx: 성공(Success)

- **200 (OK):** 요청이 성공적으로 처리되었고, 응답에 요청한 데이터가 포함됨.
    - 예: GET 요청으로 데이터 가져오기 성공.
- **201 (Created):** 요청이 성공적으로 처리되었고, 새로운 리소스가 생성됨.
    - 예: POST 요청으로 데이터 생성 성공.

REST API 호출

4xx: 클라이언트 오류(Client Error)

- **400 (Bad Request):** 잘못된 요청으로, 서버가 요청을 이해할 수 없음.
  - 예: 필수 파라미터가 누락되거나 데이터 형식이 잘못됨.
- **401 (Unauthorized):** 인증이 필요하지만 인증 자격 증명이 없거나 잘못됨.
  - 예: 로그인하지 않거나 토큰이 유효하지 않을 때.
- **403 (Forbidden):** 요청이 서버에서 거부됨(권한 없음).
  - 예: 권한이 없는 리소스에 접근하려고 할 때.
- **404 (Not Found):** 요청한 리소스를 서버에서 찾을 수 없음.
  - 예: 잘못된 URL 또는 존재하지 않는 리소스를 요청.
- **429 (Too Many Requests):** 클라이언트가 너무 많은 요청을 짧은 시간에 보냄.
  - 예: API 호출 제한을 초과했을 때.

REST API 호출

5xx: 서버 오류(Server Error)

- **500 (Internal Server Error):** 서버에서 요청을 처리하는 중에 오류가 발생함.
  - 예: 서버 코드에 문제가 있거나 예외가 처리되지 않은 경우.
- **502 (Bad Gateway):** 서버가 잘못된 응답을 다른 서버로부터 수신함.
  - 예: 게이트웨이 서버의 연결 문제.
- **503 (Service Unavailable):** 서버가 현재 사용 불가능한 상태(과부하 또는 유지보수 중).
  - 예: API 서비스가 다운되었을 때.
- **504 (Gateway Timeout):** 서버가 다른 서버로부터 응답을 기다리는 동안 시간 초과 발생.
  - 예: 네트워크 지연 또는 서버 간 연결 문제.

REST API 호출

HTTP 응답 코드 요약 테이블

상태 코드	설명	예시
200	요청 성공	GET 요청으로 데이터 조회 성공
201	리소스 생성 성공	POST 요청으로 데이터 추가 성공
400	잘못된 요청	필수 파라미터 누락
401	인증 실패	로그인하지 않음
403	권한 없음	접근 권한이 없는 리소스 요청
404	리소스를 찾을 수 없음	잘못된 URL 또는 삭제된 리소스 요청
429	너무 많은 요청	API 호출 횟수 제한 초과
500	서버 내부 오류	서버 코드에서 처리되지 않은 예외 발생
502	잘못된 게이트웨이 응답	서버 간 연결 문제 발생
503	서비스 불가능	서버 유지보수 또는 과부하
504	게이트웨이 시간 초과	서버 간 네트워크 지연

REST API 호출

- **주요 HTTP 메서드 테이블:**

HTTP 메서드	설명	예제 코드
GET	서버에서 데이터 요청	`requests.get('https://jsonplaceholder.typicode.com/posts')`
POST	서버에 데이터 추가	`requests.post('https://jsonplaceholder.typicode.com/posts', json=data)`
PUT	서버의 데이터 수정	`requests.put('https://jsonplaceholder.typicode.com/posts/1', json=data)`
DELETE	서버에서 데이터 삭제	`requests.delete('https://jsonplaceholder.typicode.com/posts/1')`

REST API 호출

- **핵심 요약:**
  - REST API는 현대 웹 개발에서 중요한 데이터 교환 방식.
  - 파이썬은 `requests` 라이브러리를 사용해 간단하고 효율적으로 REST API와 통신 가능.
  - HTTP 메서드(GET, POST 등)와 응답 데이터를 이해하는 것이 중요.

REST API 학습 자료 : **https://restfulapi.net/**

The End