PNU Mini Bootcamp 백엔드&클라우드 과정

1일차 - Python 기초 및 FastAPI 개요

송준우

2025년 2월 3일

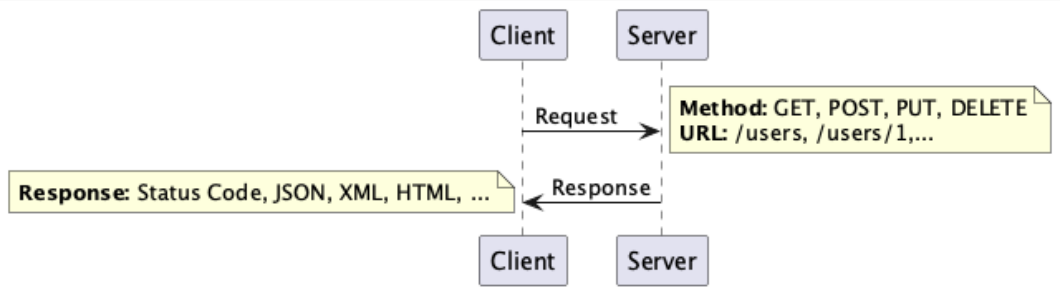
멋쟁이사자처럼

목차

3. RESTful API 설계

3.1 RESTful API

Representational State Transfer(REST)



3.1 RESTful API 특징

1. Resource

https://aws.amazon.com/ko/what-is/restful-api/

리소스를 URI(Uniform Resource Identifier)로 식별

- /users: 사용자 목록
- /users/1: ID가 1인 사용자
- /users/1/posts: ID가 1인 사용자의 글 목록
- /products: 상품 목록
- /products/1: ID가 1인 상품
- /products/1/photos: ID가 1인 상품의 사진 목록

3.1 RESTful API 특징

2. Method

HTTP Method를 사용하여 리소스에 대한 작업 정의

- GET: 리소스 조회(예: GET /users)
- POST: 리소스 생성(예: POST /users)
- PUT: 리소스 수정(예: PUT /users/1)
- DELETE: 리소스 삭제(예: DELETE /users/1)

3. Stateless

모든 요청은 독립적이며 요청간의 의존성이 존재하지 않음

- 요청헤더에 인증정보를 포함하여 사용자 식별
- 클라이언트단에서 세션을 관리

요청 예시

- Method: POST
- URL: /users
- Content-Type: application/json
- Body:

```
{
    "name": "John Doe",
    "email": "",
    "phone": "010-1234-5678",
    "age": 30
}
```

응답 예시

- Status: 201 Created
- URL: /users
- Content-Type: application/json
- Body:

```
{
    "user": {
        "id": 1,
        "name": "John Doe",
        "email": "",
        "phone": "010-1234-5678",
        "age": 30
    }
}
```

3.2 Best practices for REST API design

1. 요청 페이로드와 응답 페이로드는 JSON 형식을 사용

```
{
    "name": "John Doe",
    "email": "",
    "phone": "010-1234-5678",
    "age": 30
}
```

https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

2. Endpoint 주소는 복수형 명사를 사용

- GET /users
- GET /products/1
- PUT /users/1
- DELETE /users/1
- POST /products/1/comments

3. HTTP Method를 사용하여 리소스에 대한 작업 정의

get, create와 같은 기능은 HTTP Method로 구분

예시

작업	동사표현	REST
ID가 1인 사용자 가져오기	/users/get/1	GET /users/1
댓글 작성	/comments/create	POST /comments
ID가 1인 사용자 수정	/users/update/1	PUT /users/1
ID가 1인 사용자 삭제	/users/delete/1	DELETE /users/1

4. 서버의 응답은 HTTP 표준 상태 코드를 사용

예시

상태	코드
성공	200 OK
리소스 생성 성공	201 Created
클라이언트의 입력 Payload 유효성 검사 실패	400 Bad Request
접근 권한 없음	403 Forbidden
리소스를 찾을 수 없음	404 Not Found
서버 오류	500 Internal Server Error

https://developer.mozilla.org/en-US/docs/Web/HTTP/Statusclient_error_responses

5. 필터링, 정렬, 페이징 기능 지원

필터링, 정렬, 페이징 예시

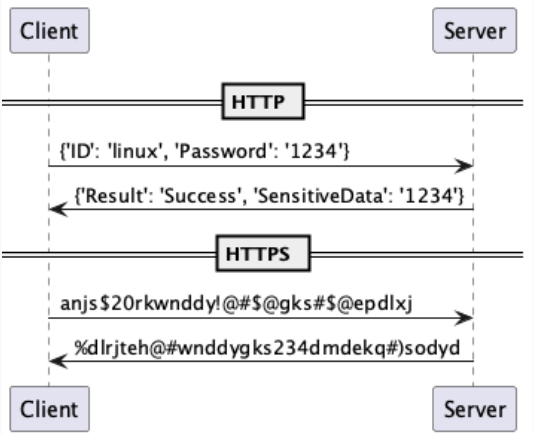
예시

Case	Endpoint	Filter	Sort	Paging
모든 사용자	/users		sort=name	page=1&limit=10
대한민국 거주자	/users	country=KR	sort=name	page=1&limit=10
나이가 30세 이상인 사용자	/users	age_gte=30	sort=name	page=1&limit=10

limit 주의시항: limit의 최대값을 설정하여 서버 부하 방지 필요

6. 보안을 위해 HTTPS 사용

https://로 시작하는 URL을 사용하여 통신 내용 암호화



7. 캐시 데이터

동일한 URI에 대한 요청을 캐시하여 응답 속도 향상

- 데이터베이스 쿼리는 CPU, 메모리, 디스크 I/O를 사용하며 잦은 요청은 비용과 부하 발생
- Redis와 같은 인메모리 DB를 사용하여 캐시 데이터 저장
- 응답 헤더에 Cache-Control 정보를 포함

8. API 버전 관리

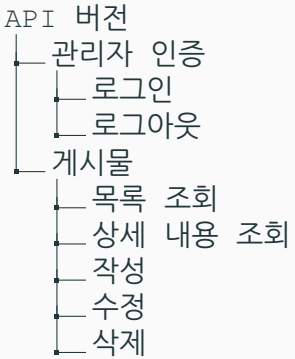
서비스 운영 중 기능 추가, 버그 수정 등 변경사항이 발생할 수 있음
요청과 응답이 변경되면 기존 클라이언트에 심각한 영향을 줄 수 있음
API 주소에 버전 정보를 포함하여 설계
오래된 버전의 API는 점진적으로 폐기

- API Version 1 - https://api.example.com/v1/users
- API Version 2 - https://api.example.com/v2/users

3.3 RESTful API 설계 실습

1. 블로그 API 요구사항

블로그 웹 사이트를 구축하기 위한 REST API를 설계해봅시다. 블로그에는 다음과 같은 기능들이 필요합니다.



2. 데이터 모델링

서비스를 구성하는 데이터 모델을 정의해봅시다.

- User
- Post

2. 데이터 모델링

User

```
{
    "id": 1,
    "username": "admin",
    "password": "password",
    "login_count": 1,
    "last_login": "2025-02-03T00:00:00",
    "created_at": "2025-02-03T00:00:00"
}
```

2. 데이터 모델링

Post

```
{
    "id": 1,
    "title": "Hello, World!",
    "content": "This is my first post.",
    "tags": ["Python", "FastAPI"],
    "created_at": "2025-02-03T00:00:00"
}
```

3. URI 설계

기능별 Method, URI, 인증 방식을 정의해봅시다.

기능	Method	URI	인증
로그인	POST	/auth/login	X
로그아웃	POST	/auth/logout	JWT
게시물 목록 조회	GET	/posts	X
게시물 상세 조회	GET	/posts/:id	X
게시물 작성	POST	/posts	JWT
게시물 수정	PUT	/posts/:id	JWT
게시물 삭제	DELETE	/posts/:id	JWT

4. 로그인 API

사용자의 아이디와 패스워드를 검증하여 JWT 토큰을 발급

Request

1. Method: POST
2. URL: /v1/auth/login
3. Content-Type: application/json
4. Body:

```
{
    "username": "admin",
    "password": "password"
}
```

Response

1. Content-Type: application/json
2. Body:

```
{
    "idToken": "eyJhbG...",
    "expiresIn": 3600,
    "error": null
}
```

https://jwt.io/

5. 로그아웃 API

사용자의 JWT 토큰을 검증 후 만료 처리

Request

1. Method: POST
2. URL: /v1/auth/logout
3. Content-Type: application/json
4. Authorization : Bearer JWT

```
{
}
```

Response

1. Content-Type: application/json
2. Body:

```
{
    "error": null
}
```

https://jwt.io/

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

6. 게시물 목록 조회 API

Request

1. Method: GET
2. URL: /v1/posts
3. Content-Type: application/json
4. Query Parameter:
   - page: 1
   - limit: 10
   - order: desc
   - tag: Python

Response

1. Content-Type: application/json
2. Body:

```
{
    "posts": [
        {
            "id": 1,
            "title": "Hello, World!",
            "tags": ["Python"],
            "created_at": "2025-02-03.."
        }, ...
    ],
    "error": null
}
```

7. 게시물 상세 조회 API

Request

1. Method: GET
2. URL: /v1/posts/:id
3. Content-Type: application/json
4. URI Parameter:
   - id: 게시물의 id

Response

1. Content-Type: application/json
2. Body:

```
{
    "posts": [
        {
            "id": 1,
            "title": "Hello, World!",
            "content": "This is my ...",
            "tags": ["Python"],
            "created_at": "2025-02-03.."
        }
    ], "error": null
}
```

8. 게시물 작성 API

Request

1. Method: POST
2. URL: /v1/posts
3. Content-Type: application/json
4. Authorization : Bearer JWT
5. Body:

```
{
    "title": "Hello, World!",
    "content": "This is my ...",
    "tags": ["Python"],
}
```

Response

1. Content-Type: application/json
2. Body:

```
{
    "posts": [
        {
            "id": 1,
            "title": "Hello, World!",
            "content": "This is my ...",
            "tags": ["Python"],
            "created_at": "2025-02-03.."
        }
    ], "error": null
}
```

9. 게시물 수정 API

Request

1. Method: PUT
2. URL: /v1/posts/:id
3. Content-Type: application/json
4. Authorization : Bearer JWT
5. Body:

```
{
    "title": "Hello, World!",
    "content": "This is my ...",
    "tags": ["Python"],
}
```

Response

1. Content-Type: application/json
2. Body:

```
{
    "posts": [
        {
            "id": 1,
            "title": "Hello, World!",
            "content": "This is my ...",
            "tags": ["Python"],
            "created_at": "2025-02-03.."
        }
    ], "error": null
}
```

10. 게시물 삭제 API

Request

1. Method: DELETE
2. URL: /v1/posts/:id
3. Content-Type: application/json
4. Authorization : Bearer JWT
5. Body:

```
{
}
```

Response

1. Content-Type: application/json
2. Body:

```
{
    "error": null
}
```