RESTful API

제출일	2022.10.10	**전공**	컴퓨터공학과
과목	컴퓨터네트워크(02)	**학번**	201602037
담당교수	이영석 교수님	**이름**	이규정

① RESTful API with SQLite3 코드

 # 코드에 대한 설명은 주석으로 달아 놓았습니다.

이번 과제는 SQLite3 의 python 문법과 cursor class의 함수들을 검색하여 해결할 수 있었고, POST 함수 구현 부분에서 execute의 매개변수 부분에 알맞은 변수를 넣는 부분에서 막히는 부분이 있었지만 여러가지 변수를 넣어가면서 테스트 해본 결과 paste.content를 매개변수에 넣어주어서 해결 할 수 있었습니다.

1. __Init__

main.py

```python
import sqlite3
from fastapi import FastAPI
from pydantic import BaseModel

# POST : INSERT
# GET : SELECT
# PUT : UPDATE
# DELETE : DELETE

app = FastAPI()
conn = sqlite3.connect('answer.db', check_same_thread=False)
cur = conn.cursor()
# DB 파일을 열고 cursor 생성

cur.execute('''CREATE TABLE IF NOT EXISTS Paste (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                content TEXT);''')
# 불러온 DB에 Paste 라는 TABLE이 존재 하지 않으면 Paste 테이블 생성
# id : INTEGER, content : TEXT

conn.commit()
# 테이블의 변경내용을 DB에 commit하는 부분     . uvicorn을 종료해도 데이터가 사라지지 않음.

# Paste 테이블의 content column에 접근하기 위한 class, content는 str로 선언.
class Paste(BaseModel):
    content: str
```

2. POST 함수

```python
# POST method
@app.post('/paste/')
def post_paste(paste: Paste):
    POST_SQL = '''INSERT INTO Paste(content) VALUES (?)'''
    # 테이블에 데이터를 삽입하기 위한 SQL 쿼리문

    res = cur.execute(POST_SQL, (paste.content,))
    # Cursor class의 sql 쿼리를 실행하는 함수(execute)
    # param 에는 위에서 작성한 POST_SQL과 FastAPI 에서 content = "..." 에 접근하기 위해 paste.content로 선언.

    print("DATA INSERT, ID:", cur.lastrowid)
    # 입력한 content data에 접근하기 위한 paste_id를 알기 위해 출력문으로 작성.
    # cursor class의 lastrowid() 함수로 마지막으로 입력된 row의 번호가 출력.
    cur.fetchone()
    conn.commit()
    # commit로 변경사항 저장.

    return {'paste_id': cur.lastrowid,
            'paste': paste.content}
```

3. GET 함수

```python
# GET method
@app.get('/paste/{paste_id}')
def get_paste(paste_id: int):
    GET_SQL = '''SELECT id, content FROM Paste WHERE id = ?'''
    # 테이블의 데이터 조회 SQL 쿼리문

    res = cur.execute(GET_SQL, (paste_id,))
    # execute로 sql 문장 실행, 조회하고자 하는 paste_id로 접근
    data = res.fetchone()

    # 단지 테이블을 조회 하는것 뿐이기에 commit는 할 필요없음.
    if data is not None:
        paste = Paste(content=data[1])
        return {'paste_id': data[0],
                'paste': paste}
    else:
        return {'paste_id': paste_id,
                'paste': None}
```

4. PUT 함수

```python
# PUT method
@app.put('/paste/{paste_id}')
def put_paste(paste_id: int, paste: Paste):
    PUT_SQL = '''UPDATE Paste SET content = ? WHERE id = ?'''
    res = cur.execute(PUT_SQL, (paste.content, paste_id))
    # UPDATE 하고자 하는 변수 : paste.content, id = paste_id 로 접근
    res.fetchone()
    conn.commit()

    return {'paste_id': paste_id,
            'paste': None}
```

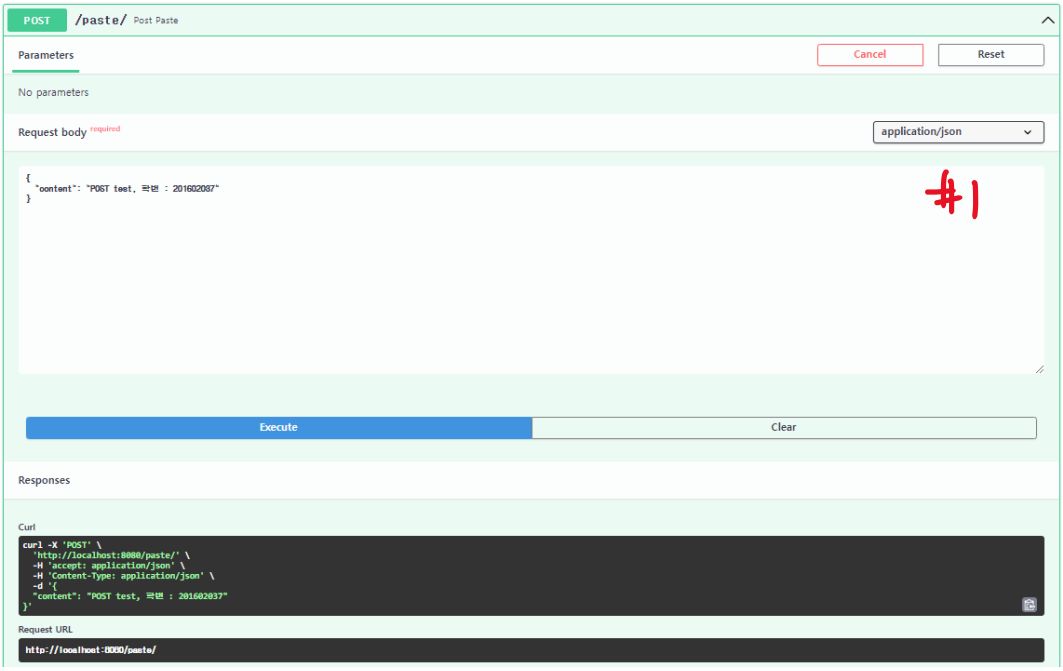
5. DELETE 함수

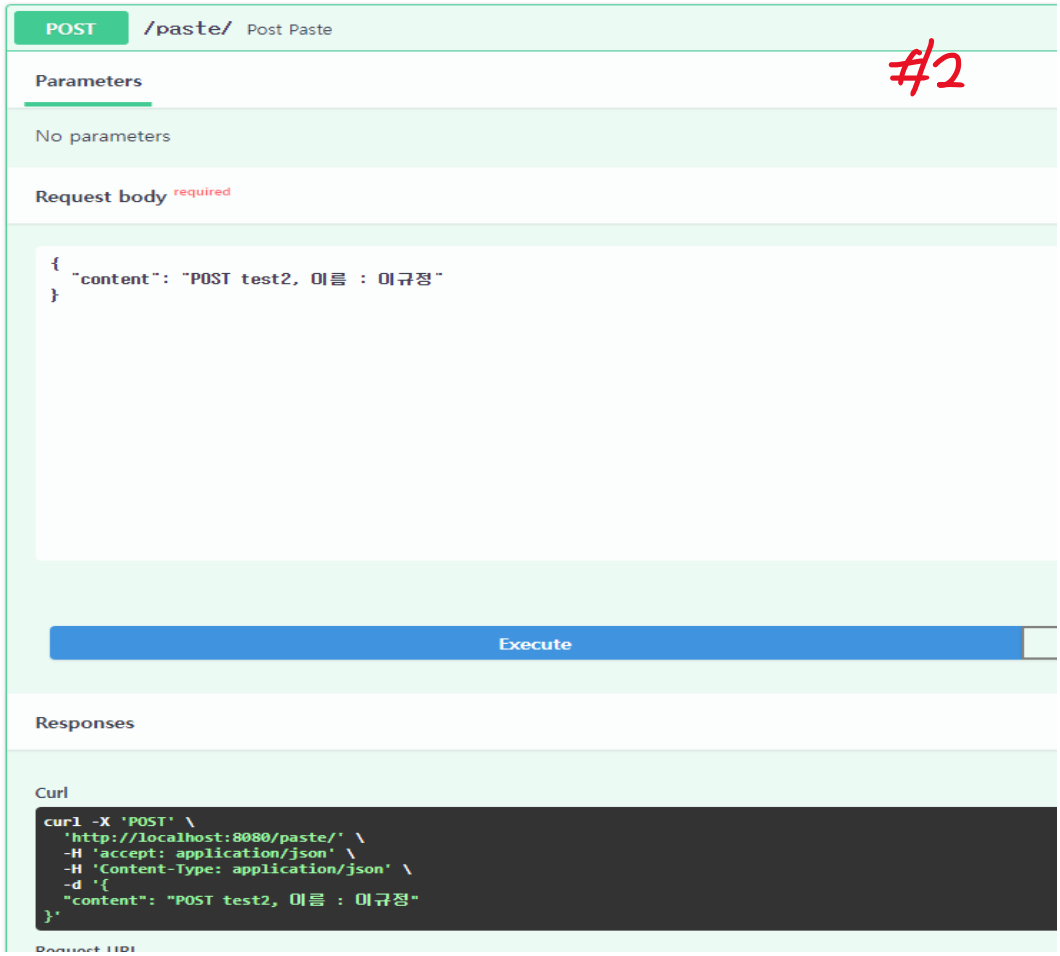
```python
# DELETE method
@app.delete('/paste/{paste_id}')
def delete_paste(paste_id: int):
    DELETE_SQL = '''DELETE FROM Paste WHERE id = ?'''
    res = cur.execute(DELETE_SQL, (paste_id,))
    # DELETE : 입력한 paste_id 삭제
    res.fetchone()
    conn.commit()

    return {'paste_id': paste_id,
            'paste': None}
```

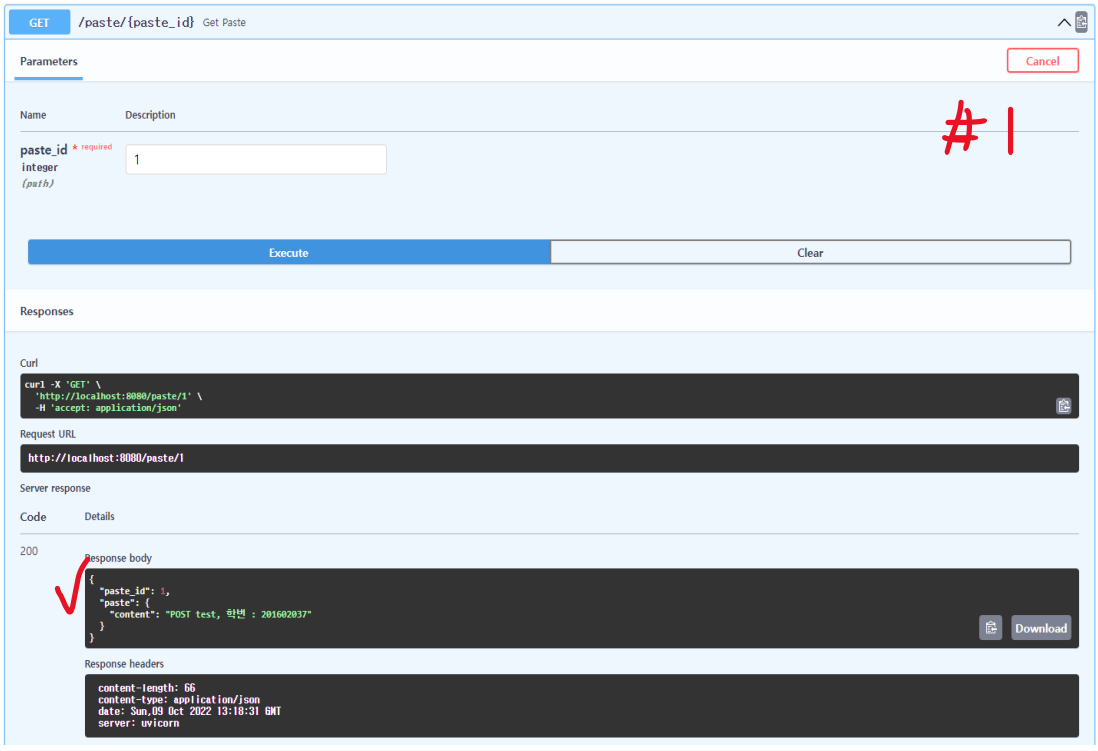
② 실행 결과 (docs)

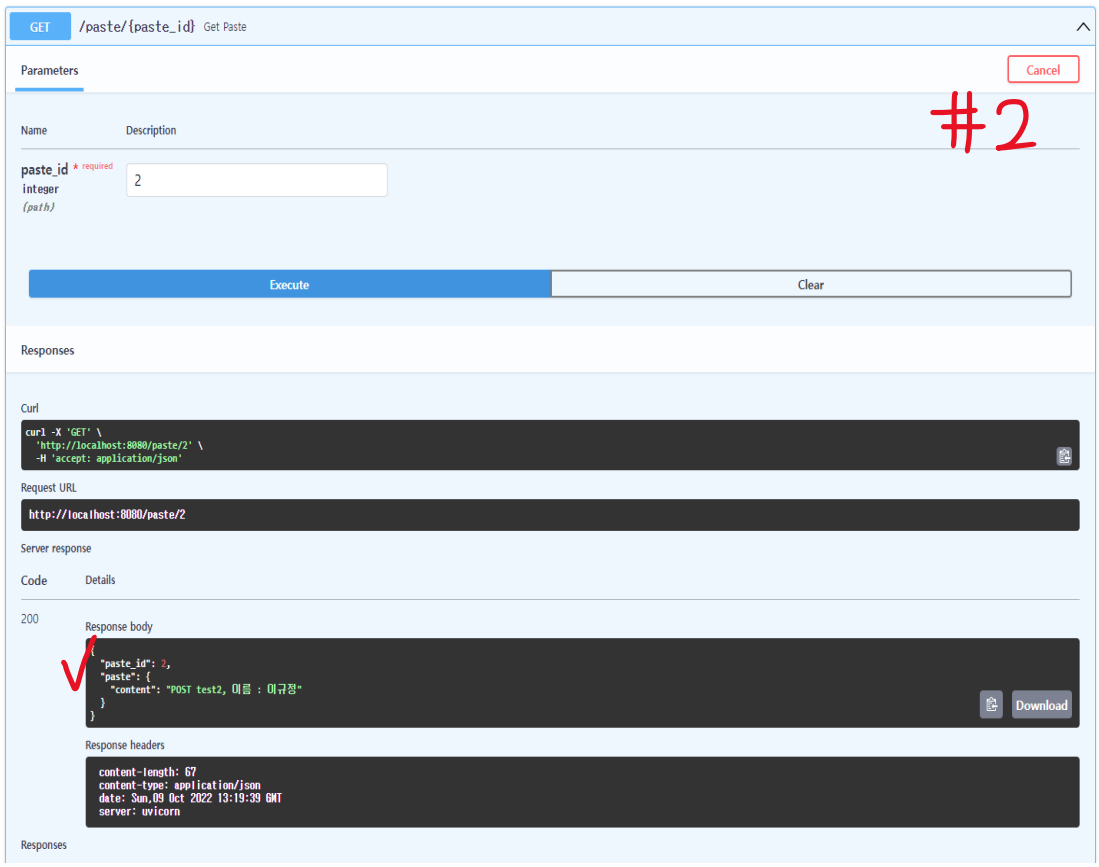
1. POST(Data 2개 삽입)



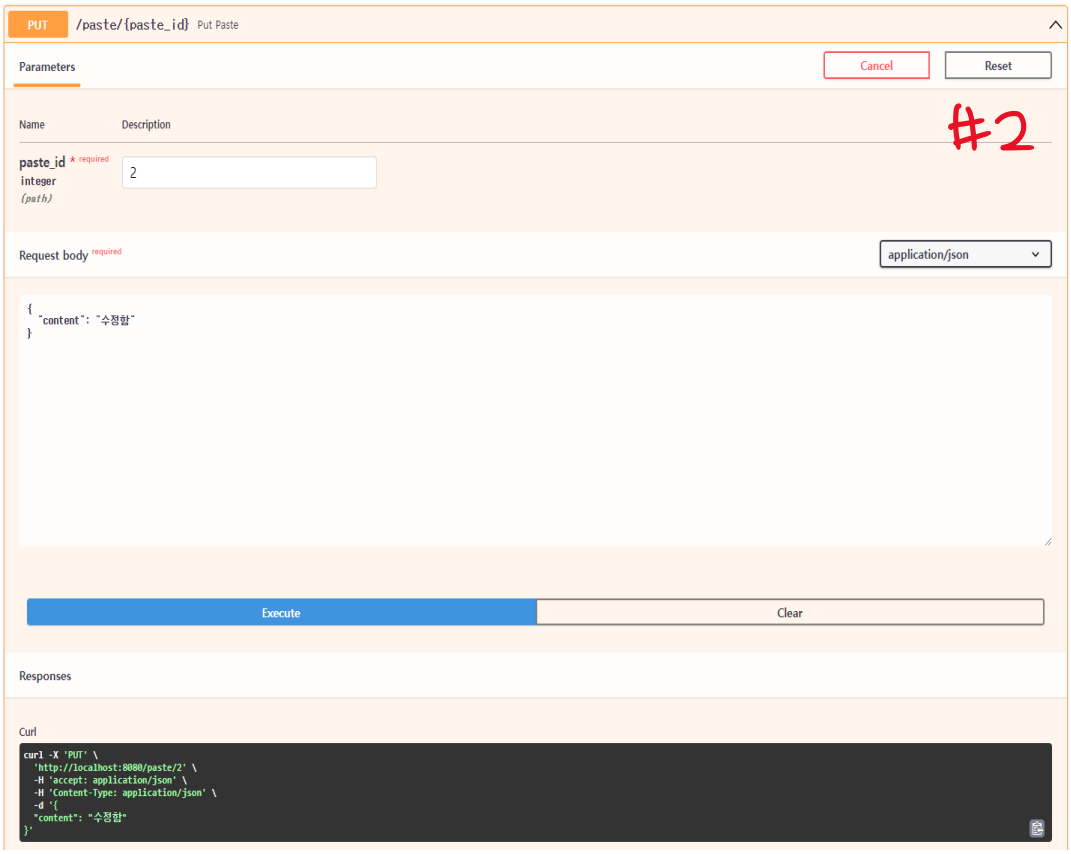


2. GET(위에서 삽입한 Data 2개 조회)

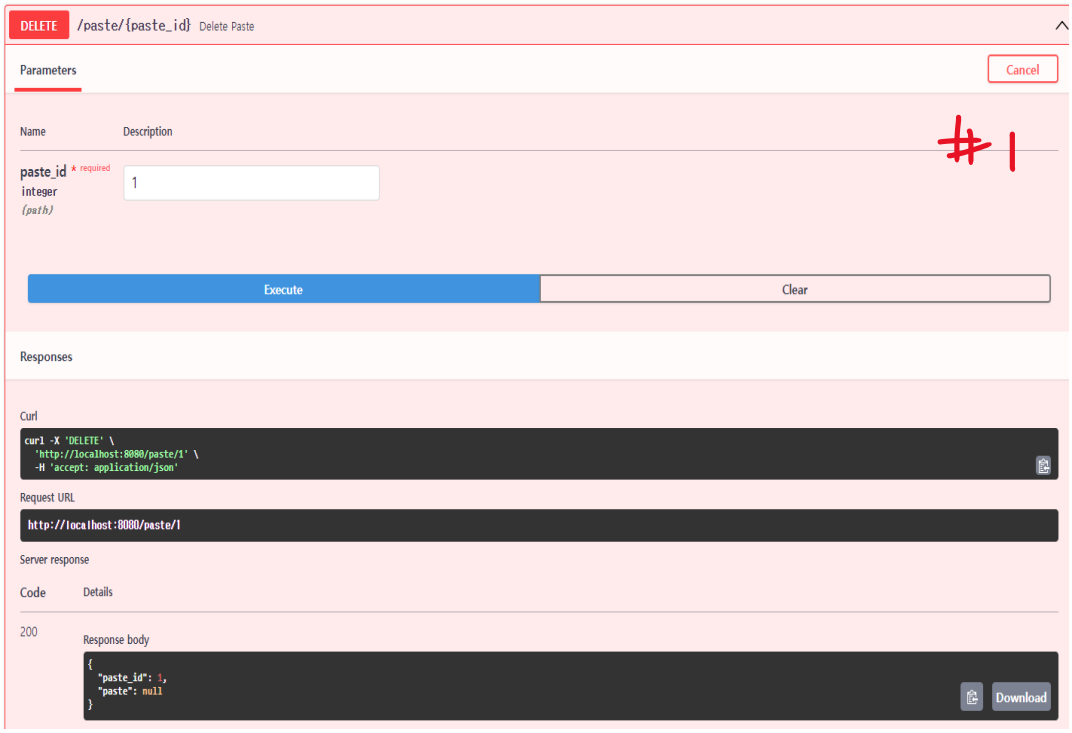




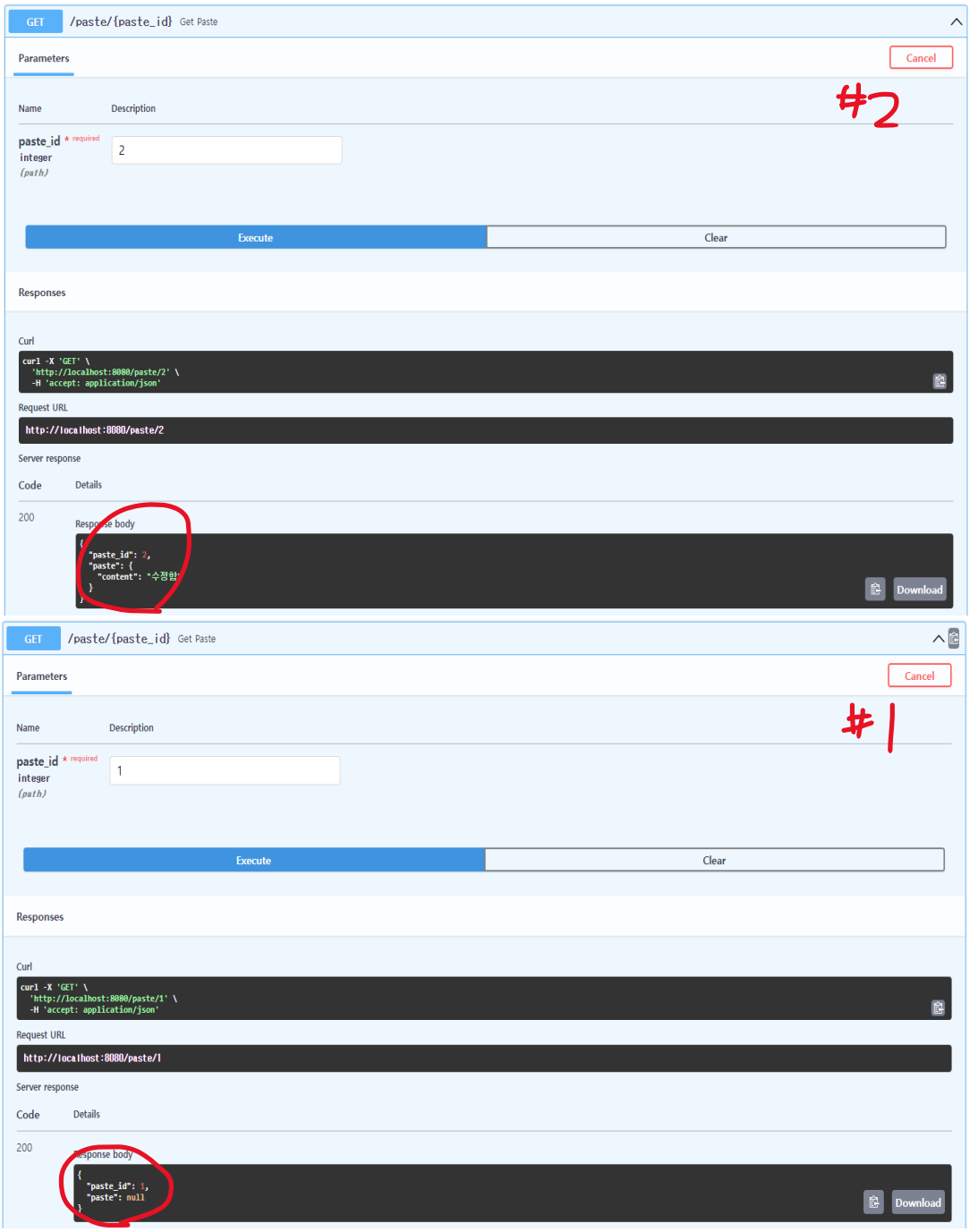
3. PUT(2번 Data 수정 -> content : “수정함”)



4. DELETE(1번 Data 삭제)



5. GET(3번에서 PUT한 Data, 4번에서 DELETE한 Data 다시 GET으로 조회)



③ 서버 재가동 후 docs GET 결과

 # 단순한 스크린샷으로 증명이 어려워 영상으로 첨부합니다.