GDSC Mount - 실습 Step 1

1. 파일 업로드, 삭제 구현사항

파일 업로드

< 구현할 사항 >

- 파일을 업로드 할 경우, 관련된 메타데이터를 DB에 저장해야 합니다. 필수적으로 저장해야 할 데이터를 고려하여 DB에 저장하도록 해주세요.
- 로그인은 구현하지 않을 것 입니다. 따라서 업로드한 유저의 정보도 같이 전달해 주도록 하세요.
- 서로 다른 사용자가 같은 이름의 파일을 업로드 할 수 있습니다. 이 경우, 서버에 저장되는 파일 이름은 어떻게 해야할까요?

< 구현 방향 >

우선 기본적인 메타데이터는 file_size, original_filename, stored_filename, id, uploader 로 정했다. 서로 다른 사용자가 같은 이름의 파일을 업로드 할 것을 대비해서 stored_filename 에는 유저가 업로드한 시간과 uuid 로 만든 고유한 값이 같이 저장되도록 했다.

그리고, 파일 삭제를 구현하다가, 한 사람이 여러 파일을 올리면 어떻게 특정 파일을 삭제하지 ? 라는 생각을 했다. 그래서 count 를 같이 도입했다. 만약 같은 유저가 동일한 파일을 업로드 하면, count 가 default = 1 에서 +1 씩 올라간다.

- 내가 저장한 메타데이터 : file_size, original_filename, stored_filename, id, uploader, count
- id : primary_key

ison

```
[
  {
    "file_size": 18,
    "original_filename": "test.txt",
    "count": 1,
    "stored_filename": "20230912T014320_test.txt_275fc1742c3c42cc93f645e64e868857",
    "id": 1,
    "uploader": "soeun"
  },
  {
    "file_size": 18,
    "original_filename": "test.txt",
    "count": 2,
    "stored_filename": "20230912T014411_test.txt_4db6a03d1a2041cc9b9c0486edcef26b",
    "id": 2,
    "uploader": "soeun"
  }
]
```

파일 삭제

< 구현할 사항 >

- 당연하지만 파일이 존재하지 않으면 오류를 반환해야 합니다.
- 물리적으로도 삭제되도록 해 주세요.

< 구현 방향 >

처음에 내가 delete 를 구현했을 때는 아래처럼 구현했다. 일단 is_user 함수는, original_filename 을 입력 받으면 그 파일을 업로드 한 유저 중에 내가 입력한 username 이 있는지 찾는다.
delete 함수는 만약 유저 이름과 유저가 업로드한 original_filename 이 일치한다면, 그 유저가 업로드한 첫번째 파일을 가져와서 삭제 한다.

```
def is_user(original_filename : str, username : str, db : db_dependency):
    uploaded_files = db.query(Files).filter(Files.original_filename == original_filename).all()
    for uploaded_file in uploaded_files:
        if uploaded_file.uploader == username:
            return True
    return False


@router.delete("/deletefile/{original_filename}", status_code = status.HTTP_204_NO_CONTENT)
async def delete_file(db:db_dependency, username : str, original_filename : str):
    if not is_user(original_filename, username, db ):
        raise HTTPException(status_code = 403, detail = "Permission denied")
    uploaded_file = db.query(Files).filter(Files.original_filename == original_filename).filter(Files.uploader == username).first()
    if not uploaded_file:
        raise HTTPException(status_code = 404, detail = "File not found")

    stored_filename = uploaded_file.stored_filename
    try:
        os.remove(stored_filename)
    except FileNotFoundError:
        pass

    db.delete(uploaded_file)
    db.commit()
```

하지만 내가 간과했던 부분이, 만약 유저가 동일한 파일명을 여러 번 올린다면 ? 만약 3번 업로드 했다면 첫번째, 두번째로 올린 파일을 삭제하고 싶다면 ? 오로지 original_filename 과 username 으로는 이 기능을 수행하지 못한다.

그렇다면 .. 결국 stored_filename 을 사용해야 하는데.. 사용자가 직접 이 stored_filename 을 입력으로 받는 것은 무리다. 자신이 올린 정확한 시간을 기억하지 못할 것이다. 그렇다면 올린 id 번호로 ..?

이 문제를 고민하다가 upload 함수에 count 를 추가했다. 유저는 자신이 몇 번째로 올린 파일을 삭제할 것인지 input 을 받는다. (default = 1 이다) 최종적으로는 아래와 같이 구현했다.

```
def is_user(original_filename : str, username : str, db : db_dependency):
    uploaded_files = db.query(Files).filter(Files.original_filename == original_filename).all()
    for uploaded_file in uploaded_files:
        if uploaded_file.uploader == username:
            return True
    return False


@router.delete("/deletefile/{original_filename}", status_code = status.HTTP_204_NO_CONTENT)
async def delete_file(db:db_dependency, username : str, original_filename : str, count : int = 1):
    if not is_user(original_filename, username, db ):
        raise HTTPException(status_code = 403, detail = "Permission denied")
    uploaded_file = db.query(Files).filter(
        Files.original_filename == original_filename)\
            .filter(Files.uploader == username)\
                .filter(Files.count == count).first()
    if not uploaded_file:
        raise HTTPException(status_code = 404, detail = "File not found")

    stored_filename = uploaded_file.stored_filename
    try:
        os.remove(stored_filename)
    except FileNotFoundError:
        pass

    db.delete(uploaded_file)
    db.commit()
```

파일 다운로드

< 구현할 사항 >

- 우선은 본인이 만든 파일이 아니면 다운이 불가능하게 설정해주세요.

< 구현 방향 >

일단 유저, 파일 이름, count 를 입력받아서 파일을 다운로드 하도록 구현했다.

```
@router.get("/download/file/{file_id}")
async def download_file(db:db_dependency, username : str, original_filename : str, count: int = 1):
    uploaded_file = db.query(Files).filter(Files.original_filename == original_filename)\
        .filter(Files.uploader == username)\
            .filter(Files.count == count)\
                .first()
    if not uploaded_file:
        raise HTTPException(status_code = 404, detail = "File not found")
    if not is_user(original_filename, username, db ):
        raise HTTPException(status_code = 403, detail = "Permission denied")
    file_path = f"{uploaded_file.original_filename}"
    print("File Path:", file_path)
    return FileResponse(file_path, headers={"Content-Disposition": f"attachment; filename={uploaded_file.original_filename}"
```

하지만 또 문제가 생겼다. 나는 DB 에 시간별로 파일을 저장하고 있지만, test.txt 파일을 만들어서 내용을 다르게 변경한 후 다운로드 해봤다.

test1.txt > Test 1
test1.txt > Test1, Test2

이렇게 내용을 변경해서, 동일 파일명으로 업로드 해봤다. 그랬더니 한 파일 명이 덮어 씌워 진다 ...
만약 첫번째 파일을 다운 받고 싶어도 그럴 수 없다 . 이거는 어떻게 고칠까 생각을 해봤는데.. 도저히 모르겠다.

근데 DB에는 파일의 메타 데이터만 저장하지, 파일 안의 내용은 저장하지 않는다. 데이터베이스의 특징이 파일의 실제 데이와, 파일의 메타 데이터를 따로 저장하는 것인데, 그러면 파일의 실제 데이터는 어디에 저장하지 ?