Report

HW#2 간단한 백엔드 프로그램

고급 프로그래밍 실습

Dpt. of Mobile System Engineering
32192530 양윤성

2023.12.14.

● Contents

1. Abstract

이번에 구현하고자 하는 프로그램은 백엔드 기술들을 활용하여 만든 간단한 to-do list 프로그램이다. 사용자는 구현된 인터페이스를 통해 to-do list를 실행할 수 있다. 프로그램에서 실행 가능한 기능은 10가지가 있는데, 사용자가 선택한 옵션에 따라 할 일을 추가하거나 수정, 삭제가 가능하며 할 일의 완료 상태나 우선순위에 따른 조회 또한 가능하다. 이번 과제에서는 to-do list의 기능들을 REST API, gRPC, 그리고 MySQL 데이터베이스가 모두 효율적인 상호작용을 통해 수행하는 것에 목표를 둔다. 본 Techinical Report의 순서는 다음과 같다. Part2에서는 프로그램의 원리를 이해하기 위해 사용된 기술들에 대한 지식을 설명하고, 왜 이런 기술들을 선택했는지에 대해 말할 것이다. Part3, 4에서는 프로그램의 전반적인 아키텍쳐와 소개하며, 이후 Part에서는 프로그램 컴파일 방법 및 예제 테스트 스크린샷, 프로젝트 Conclusion으로 마무리한다.

2. Introduction

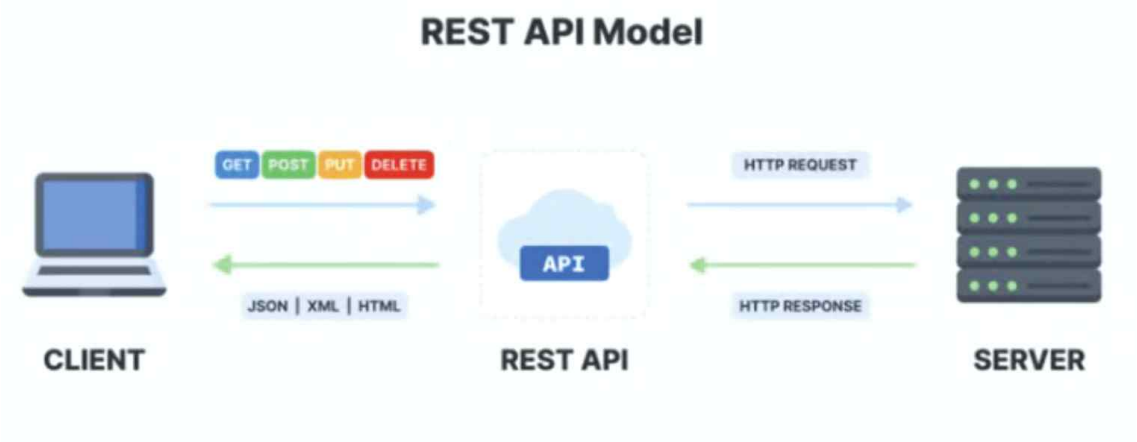


그림1. REST API

REST(Representational State Transfer) API는 웹 서비스를 위한 일반적인 인터페이스로, 웹 표준(HTTP)을 사용하여 서버와 클라이언트 간의 통신을 구현한다. REST API는 resource 중심의 아키텍처를 채택하여, resource 정보를 주고 받는 모든 것을 의미한다. URL을 통해 resource를 식별하고 HTTP method(GET, POST, PUT, PATCH, DELETE)를 사용하여 해당 resource에 대한 다양한 작업을 수행한다. 이러한 접근 방식은 간결하고 이해하기 쉬우며, HTTP 프로토콜을 지원하는 모든 플랫폼에서 적용 가능한 호환성을 갖고 있어 널리 사용된다.

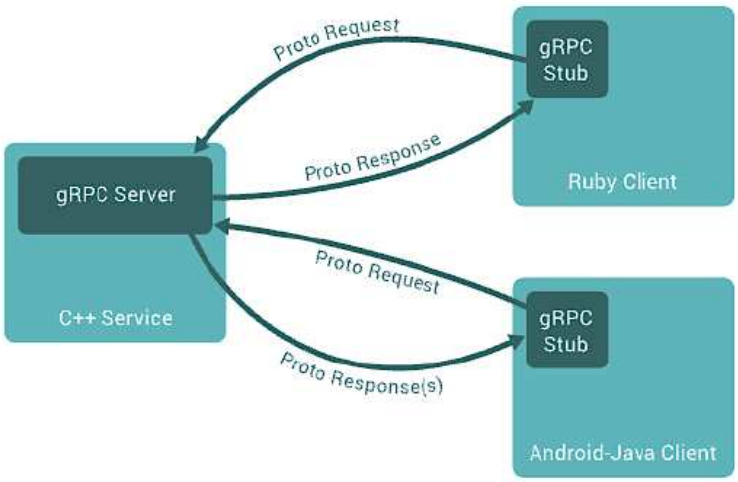


그림2. gRPC

gRPC(Google Remote Procedure Call)는 Google에서 개발한 고성능의 오픈소스 RPC 프레임워크다. gRPC는 클라이언트와 서버 간에 효율적인 양방향 통신을 가능하게 하며, Protocol Buffers(proto)로 인터페이스를 설계한다는 특징이 있다. 이는 데이터 전송에 필요한 대역폭을 최소화하고, 다양한 언어와 크로스 플랫폼이 가능하도록 한다. 또한 높은 처리량에 비해 낮은 latency을 가지고 있어 복잡한 서비스 아키텍처에 적합하다. 보통 REST API와 gRPC는 각자의 유용한 케이스에서 사용하는 경우가 많지만, 두 기술을 조합할 수 있다. REST API와 gRPC는 서로 다른 장점을 가지고 있고 이를 조합함으로써 다양한 요구 사항들을 효과적으로 충족시킬 수 있다. 그래서 이번 프로젝트에서는 REST API를 통해 간단한 인터페이스를 제공하고, gRPC를 사용하여 백엔드 서비스 간의 빠르고 효율적인 통신을 구현하고자 했다.

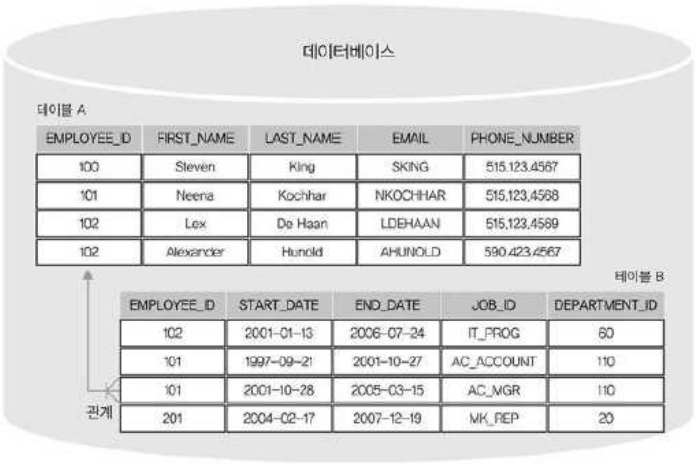


그림3. 관계형 데이터베이스

이번에 선택한 데이터베이스는 관계형(Relational) 데이터베이스 중 하나인 MySQL이다. To-do List 프로그램은 각 할 일별로 설명, 날짜, 완료여부, 우선순위 등과 같은 정형화된 데이터를 다뤄야 하는데, MySQL은 이런 데이터를 체계적으로 관리하고 효율적으로 액세스할 수 있는 구조를 제공한다. NoSQL 데이터베이스는 스키마가 유동적이기에 빠른 변화에는 잘 적응하지만, 데이터의 일관성과 정확성을 유지하는 측면에서는 관계형 데이터베이스가 유리하다고 생각했다. 또한 보편적으로 사용되는 SQL 쿼리 언어를 사용하여 데이터의 관리와 조작을 보다 쉽게 할 수 있기에 MySQL을 선택했다.

3. Design

현재 파트에서는 프로그램의 전체적인 구조를 간략히 소개하고, 구조에 대한 자세한 설명은 다음 파트인 Implemenation에서 살펴보도록 한다. 이전 과제와 비슷하게 API별로 상세한 Flow-chart가 있으면 좋겠지만, 이번에는 프로그램의 전체 구조를 통해 서비스의 상호작용에 집중하고자 한다.

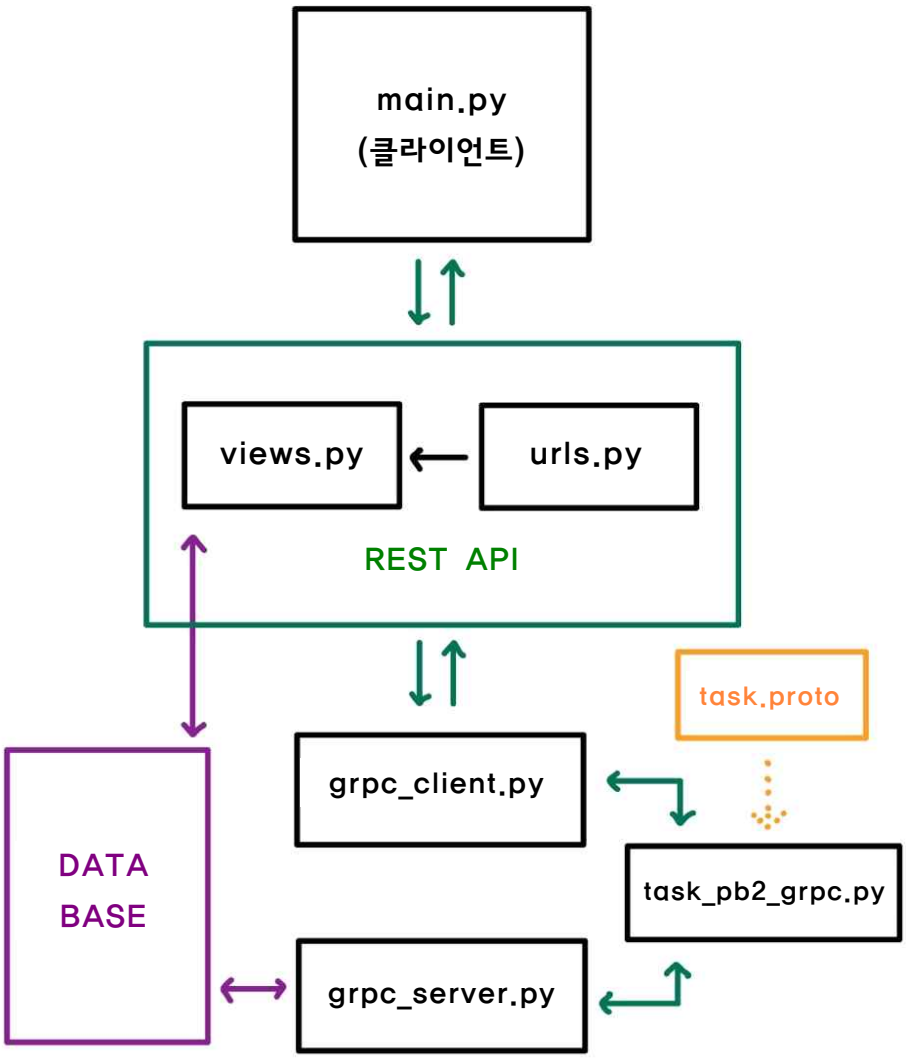


그림4. 프로그램 전체 구조도

4. Implementation

프로그램의 전체 구조를 크게 살펴보면 클라이언트 역할을 하는 main.py, REST API역할을 하는 urls.py와 views.py, gRPC 서비스 파트로 나누어진다. gRPC 부분에서는 grpc_client.py와 grpc_server.py가 있고 이 둘의 통신을 위한 task_pb2_grpc.py가 있다. 위 구조도에는 없지만 이 파일은 task_pb2.py와 같이 task.proto에서 정의되어 만들어진 코드이다. proto 파일은 Protobuf를 사용하여 데이터를 정의하는데 사용되는 파일로, 메세지 정의 파트는 pb2.py에, 통신에 관련된 gRPC skeleton과 Stub은 pb2_grpc.py로 구현된다. 이렇게 Protocol Buffers을 사용함으로써 구조화된 데이터를 직렬화할 수 있다. main.py에서 발생한 요청은 REST API를 통해 받아지는데, url.py에서 url 패턴을 매칭 후 views.py에서 적절한 gRPC 클라이언트 함수를 호출한다. 이후 gRPC 클라이언트는 gRPC 서버와 통신을 하며 요청에 맞는 로직을 실행하며, 이 과정에서 실제 데이터베이스와 연동하여 데이터 관리와 처리가 이루어지도록 한다. 원래 gRPC server만 데이터베이스에 직접 접근하는 것이 가장 이상적이고 이렇게 설계했지만, 계획했던 API 중 1가지만 gRPC server가 관리하고 나머지는 gRPC server에 구현하지 못하여 REST API에서 처리하도록 했다. 이런 이유로 구조도에도 views.py가 데이터베이스와 상호작용하는 모습을 추가했다. 이제 각 서비스들의 상호작용과 관련된 핵심 코드만 간단하게 살펴볼 것이다.

todo > views.py > CompletedListView

```python
from django.shortcuts import render
from rest_framework import generics, status
from rest_framework.response import Response
from .models import Task
from .serializers import TaskSerializer
import json, sys

from django.views.decorators.csrf import csrf_exempt
from django.http import JsonResponse

sys.path.insert(1, 'C:/Users/32192530/Desktop/todolist/grpc_files') #상위 폴더의 grpc 파일들을 가져오기 위함
from grpc_files import grpc_client

@csrf_exempt
def TaskCreateView(request): #새로운 할일을 추가 (grpc 연동)
    if request.method == "POST":
        # 여기서 grpc_client의 메서드를 호출
        data = json.loads(request.body)
        grpc_client.create_task(data['content'], data['due_date'], data['prior_rank'])
        return JsonResponse({"message": "Add new data successfully"})
```

그림5. todo/views.py (line 1~20)

views.py에는 각 API의 핵심 로직들을 수행하는데 정리하면 다음과 같다.

class(func) 이름	구현	HTTP method
	설명	
TaskCreateView	gRPC 연동	POST
	새로운 to-do를 추가	
TaskListView	REST	GET
	to-do list 전체 조회	
CompletedListView	REST	GET
	완료된 to-do들만 조회	
IncompletedListView	REST	GET
	미완료된 to-do들만 조회	
HighPriorListView	REST	GET
	우선순위가 가장 높은 to-do들만 조회	
TaskView	REST	GET
	특정 to-do 조회	
TaskCompleteView	REST	PATCH
	특정 to-do를 완료로 처리	
TaskUpdateView	REST	PUT
	특정 to-do의 상세 내용 수정	
TaskDeleteView	REST	DELETE
	특정 to-do 삭제	
DeleteCompletedView	REST	DELETE
	완료된 to-do들 삭제	
DeleteAllView	REST	DELETE
	to-do list 전체 삭제	

views.py의 핵심은 TaskCreateView의 gRPC 연동 부분과 나머지 로직 내에 포함된 REST framework 모듈 활용이다. gRPC는 Django 내의 todo앱에 존재하지 않기 때문에, 앱 밖의 gRPC에 접근하려면 따로 path를 설정해서 연결해야 한다. 따라서 sys모듈을 이용하여 gRPC의 경로를 추가해주고, 여기서 grpc_client를 import한 후 TaskCreateView에서 grpc_client의 create_task라는 함수를 호출하는 모습을 확인할 수 있다.

```python
class TaskListView(generics.ListAPIView): #할일 목록을 모두 조회
    queryset = Task.objects.all()
    serializer_class = TaskSerializer

class CompletedListView(generics.ListAPIView): #Completed가 True인 할일 목록 조회
    serializer_class = TaskSerializer

    def get_queryset(self):
        return Task.objects.filter(completed=True)

class IncompletedListView(generics.ListAPIView): #Completed가 False인 할일 목록 조회
    serializer_class = TaskSerializer
```

그림6. todo/views.py (line 32~43)

```python
    path('tasks/create/', views.TaskCreateView, name='task-create'),
    path('tasks/list/', views.TaskListView.as_view(), name='task-list'),
    path('tasks/completed/', views.CompletedListView.as_view(), name='completed-list'),
```

그림7. todo/urls.py

나머지 기능들 구현에는 Serializers와 generics를 활용했다. 2가지 모두 REST framework에서 제공하는 메소드들이다. Serializaers는 DB 모델을 JSON 형태로 직렬화와 역직렬화하는 작업을 수행하는데, 이는 클라이언트와 서버 간의 데이터 교환을 용이하게 한다. 또한 REST framework의 generics 모듈을 사용했는데, 이들은 일반적인 API 작업을 보다 쉽고 빠르게 구현할 수 있도록 설계되었다. 각 generic view는 특정한 종류의 API 요청을 처리하기 위해 사전에 정의된 메소드를 포함하고 있는데, 사용한 메소드는 아래와 같다. 이때 CreateAPIView는 새로운 객체 추가 역할을 gRPC로 넘겼기 때문에 테스트용으로만 사용하고 주석처리 하였다.

이름	HTTP method	설명
ListAPIView	GET	데이터베이스 전체 record 반환
RetrieveAPIView	GET	url에서 지정한 식별자에 해당하는 단일 객체 반환
CreateAPIView	POST	새로운 record를 생성하고 반환
UpdateAPIView	PATCH	리소스의 특정 부분만 업데이트
	PUT	리소스의 전체 업데이트
DestroyAPIView	DELETE	특정 record를 삭제

그림7의 urls.py에서도 확인 가능한 메소드들의 공통함수 as_view는 이러한 class 기반 view를 Django의 url패턴과 연결할 수 있도록 url 패턴에 대한 entry point를 제공한다. 즉, 해당 class의 인스턴스를 생성하고 들어오는 HTTP 요청에 따라 적절한 메소드를 호출한다. 이러한 REST Framework는 대부분의 기본적인 CRUD(Create, Read, Update, Delete)작업을 처리하는 데 적합하며, REST API를 보다 빠르고 효율적으로 구축할 수 있도록 도와준다.

grpc_files > grpc_client.py > ...

```python
import grpc
import task_pb2
import task_pb2_grpc

def create_task(content, due_date, prior_rank):
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = task_pb2_grpc.TaskServiceStub(channel)
        task = task_pb2.Task(content=content, due_date=due_date, prior_rank=prior_rank)
        response = stub.CreateTask(task)
        return response
```

그림8. grpc_client.py

grpc_files > grpc_server.py > ...

```python
sys.path.append('C:/Users/32192530/Desktop/todolist')
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'config.settings')
django.setup()

# gRPC 생성된 모듈과 Django 모델 import

import task_pb2
import task_pb2_grpc
from todo.models import Task

# gRPC 서비스 구현
class TaskService(task_pb2_grpc.TaskServiceServicer):
    def CreateTask(self, request, context):
        task = Task(content=request.content, due_date=str(request.due_date), prior_rank=request.prior_rank)
        task.save()
        return task_pb2.Task(id=task.id, content=task.content, due_date=str(task.due_date), prior_rank=task.prior_rank)
```

그림9. grpc_server.py

두 코드는 gRPC 서비스를 위해 존재한다. gRPC 클라이언트는 view가 호출할 함수들이 정의되어있고, 각 함수에는 gRPC 서버와 통신할 수 있는 로직이 담겨있다. gRPC 클라이언트와 서버는 gRPC stub을 이용해 view의 요청을 완료하기 위한 정보들을 주고받는다. grpc_server.py는 todo앱에 존재하는 데이터베이스와 상호작용이 필수이기 때문에 sys로 todo의 path를 추가하고 todo.models 디렉터리에 접근 가능하도록 설정해줬다.

5. Test and Results

● Build Environment

Windows 10 Education x64

Visual Studio Code 1.76.2

GCC Compiler (MinGW.org GCC-6.3.0-1)

Python 3.10.5

libprotoc 25.1

MySQL 8.0.35 for Win64 on x86_64 (MySQL Community Server – GPL)

● Compile

우선 아래와 같이 MySQL과 Protoc에 대한 추가적인 환경변수 설정이 필요하다.

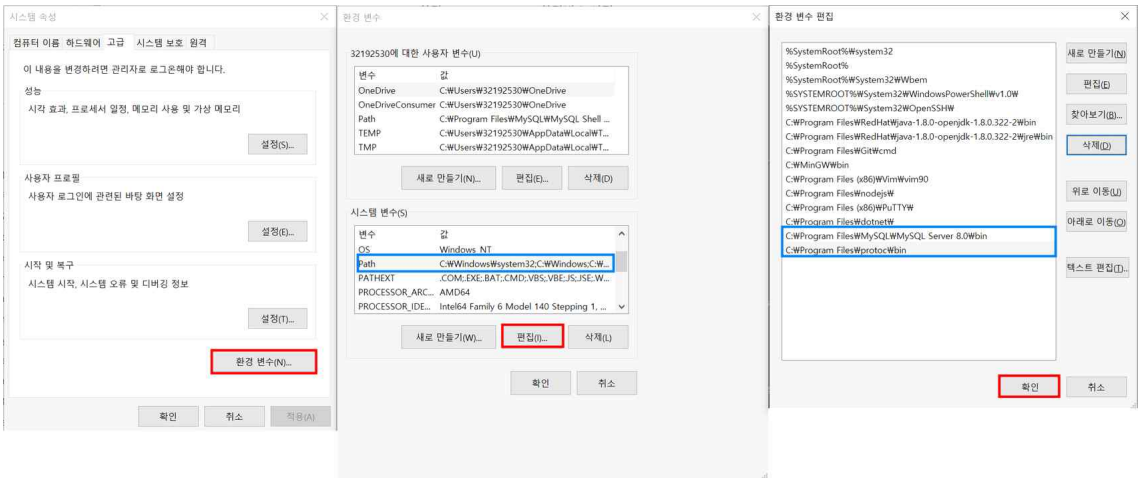


그림10. 환경변수 설정 (Windows 10 기준)

이후 32192530.zip을 해제하여 얻은 todolist 프로젝트 폴더를 저장한 뒤, 실제 저장 경로로 코드를 수정해줘야 한다. todo/views.py의 11번째 줄의 path에는 grpc_files 폴더에 대한 절대경로를, grpc_files/grpc_server.py의 7번째 줄의 path에는 todolist (프로젝트 root)에 대한 절대경로로 바꿔준다.

todo > views.py > ...

```
8   from django.views.decorators.csrf import csrf_exempt
9   from django.http import JsonResponse
10
11  sys.path.insert(1, 'C:/Users/32192530/Desktop/todolist/grpc_files')
12  from grpc_files import grpc_client
```

grpc_files > grpc_server.py > serve

```
1   import grpc
2   from concurrent import futures
3
4   import os, sys, django
5
6   # Django 설정
7   sys.path.append('C:/Users/32192530/Desktop/todolist')   # Django 프로젝트 경로 설정
8   os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'config.settings')
9   django.setup()
```

그림11. path 재설정

마지막으로 터미널 3개를 열고, 첫번째 터미널에는 *python manage.py runserver*로 Django 서버를, 두번째 터미널에는 *python grpc_files/grpc_server.py*로 gRPC 서버를 구동한다. 두 서버가 정상적으로 작동된다면 마지막 터미널에는 *python todo/main.py*로 프로그램을 실행한다. 이때 http://127.0.0.1:8000/admin으로 접속하여 실제 DB 상태를 확인할 수 있으며, admin의 ID는 32192530, PW는 1111이다.

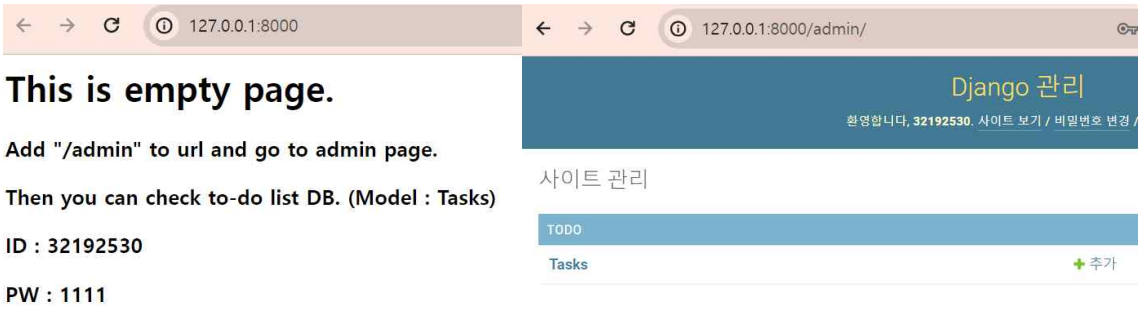


그림12. Django 테스트 페이지

● Working Proof

Working Proof에서는 MySQL로 데이터베이스를 적용한 과정과 gRPC를 위한 proto 파일을 작성한 것을 간단히 소개하겠다.

```
C:₩Users₩32192530>mysql -u root -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or ₩g.
Your MySQL connection id is 27
Server version: 8.0.35 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '₩h' for help. Type '₩c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
6 rows in set (0.00 sec)

mysql> create database todolist;
Query OK, 1 row affected (0.02 sec)
```

todo > models.py > ...

```python
from django.db import models

# Create your models here.
class Task(models.Model):
    content = models.CharField(max_length=255, null=False)
    due_date = models.DateField(null=True, blank=True)
    completed = models.BooleanField(default=False)
    PRIOR_RANK_CHOICES = [
        (1, 'High'),
        (2, 'Medium'),
        (3, 'Low'),
    ]
    prior_rank = models.IntegerField(
        choices=PRIOR_RANK_CHOICES,
        default=2,
    )

    def __str__(self):
        return self.content
```

```
PS C:\Users\32192530\Desktop\대학\3학년 2학기\고급프로그래밍실습\HW2\todolist> python manage.py makemigrations
Migrations for 'todo':
  todo\migrations\0001_initial.py
    - Create model tasks
PS C:\Users\32192530\Desktop\대학\3학년 2학기\고급프로그래밍실습\HW2\todolist> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, todo
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying todo.0001_initial... OK
```

config > settings.py

```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'todolist',
        'USER': 'root',
        'PASSWORD': 'abee3417',
        'HOST': 'localhost',
        'PORT': '3306',
```

그림13~16. MySQL & model Task 구축 과정

task.proto

```proto
syntax = "proto3";

import "google/protobuf/empty.proto";
import "google/protobuf/wrappers.proto";

package task;

// Task 메시지 정의
message Task {
  int32 id = 1; // 고유 식별자
  string content = 2; // 할일 내용
  string due_date = 3; // 마감일 (문자열 형식, 예: "2023-12-31")
  bool completed = 4; // 완료 여부
  int32 prior_rank = 5; // 우선 순위
}

// 할일 관련 서비스 정의
service TaskService {
  // 할일 추가
  rpc CreateTask(Task) returns (Task);
  // 할일 목록 조회
  rpc ListTasks(google.protobuf.Empty) returns (stream Task);
  // 특정 할일 조회
  rpc GetTask(google.protobuf.Int32Value) returns (Task);
  // 할일 업데이트
  rpc UpdateTask(Task) returns (Task);
  // 할일 삭제
  rpc DeleteTask(google.protobuf.Int32Value) returns (google.protobuf.Empty);
}
```

그림17. task.proto

● Test

테스트는 main.py로 원하는 옵션을 선택해서 진행할 수 있으며, 아래는 각 옵션별 테스트 결과이다.

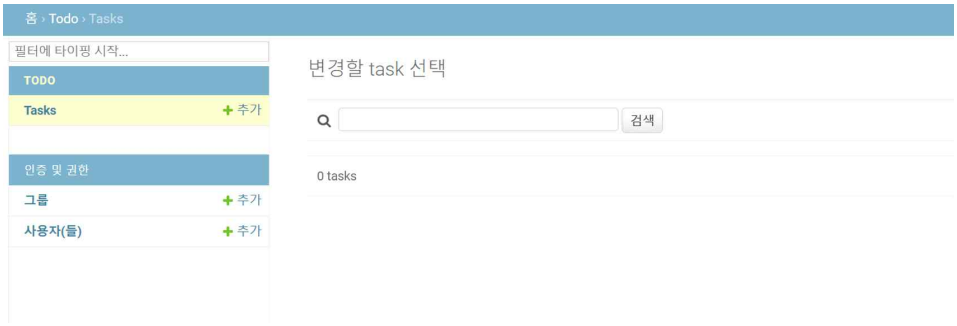


그림18. 테스트 전 비어있는 DB 상태

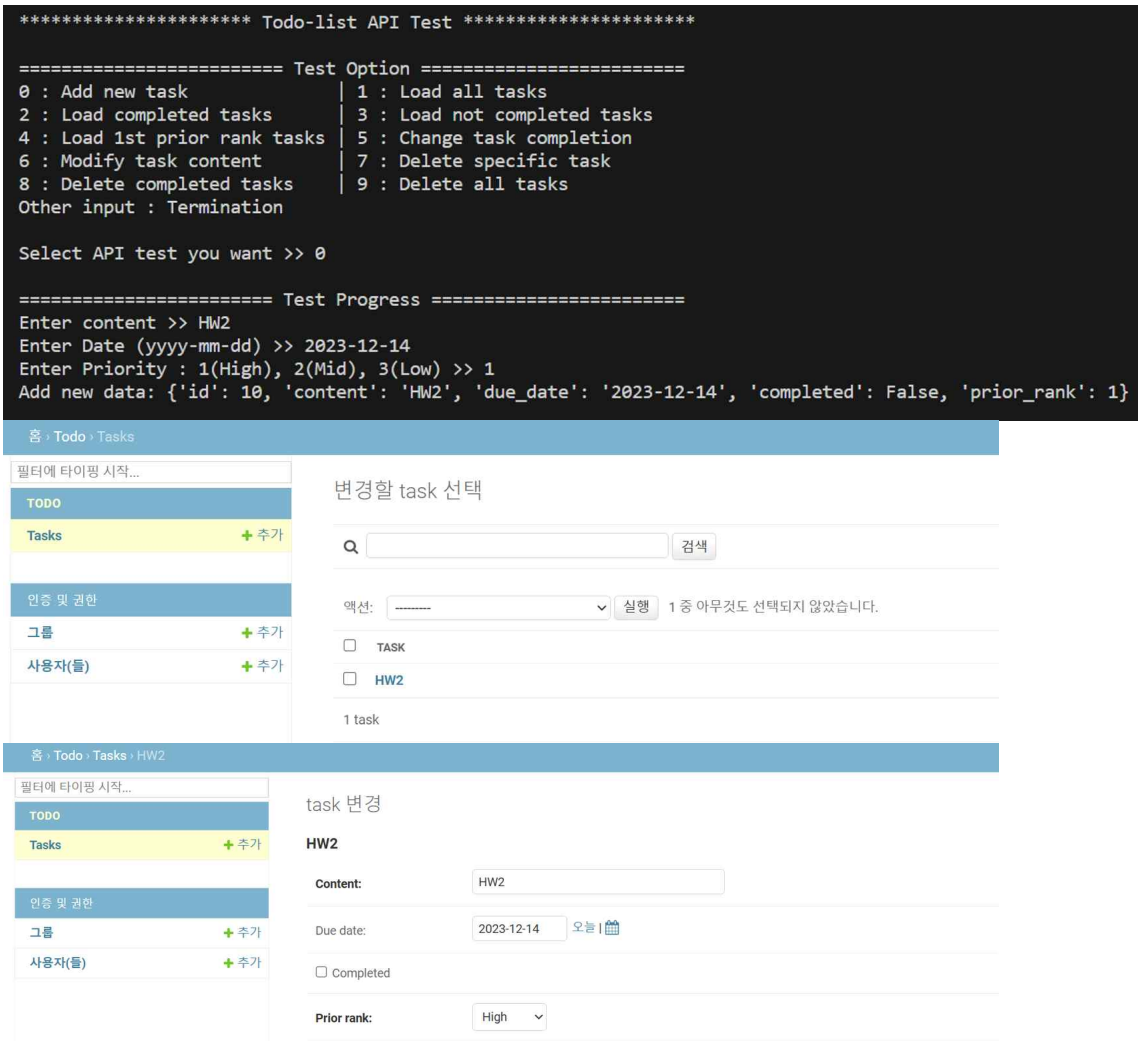
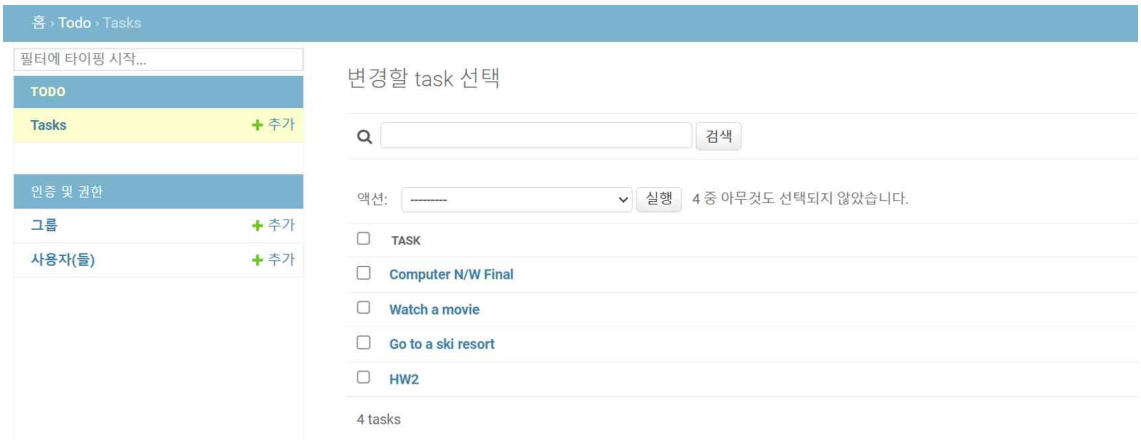


그림19. 새로운 to-do 추가 테스트 (REST ver.)

```
======================== Test Progress ========================
To-do list:

======================== Test Option ========================
0 : Add new task              | 1 : Load all tasks
2 : Load completed tasks      | 3 : Load not completed tasks
4 : Load 1st prior rank tasks | 5 : Change task completion
6 : Modify task content       | 7 : Delete specific task
8 : Delete completed tasks    | 9 : Delete all tasks
Other input : Termination

Select API test you want >> 0

======================== Test Progress ========================
Enter content >> Add new task using gRPC
Enter Date (yyyy-mm-dd) >> 2023-12-13
Enter Priority : 1(High), 2(Mid), 3(Low) >> 1
{"message": "Add new data successfully"}

======================== Test Option ========================
0 : Add new task              | 1 : Load all tasks
2 : Load completed tasks      | 3 : Load not completed tasks
4 : Load 1st prior rank tasks | 5 : Change task completion
6 : Modify task content       | 7 : Delete specific task
8 : Delete completed tasks    | 9 : Delete all tasks
Other input : Termination

Select API test you want >> 1

======================== Test Progress ========================
To-do list:
{'id': 29, 'content': 'Add new task using gRPC', 'due_date': '2023-12-13', 'completed': False, 'prior_rank': 1}
```

그림20. 새로운 to-do 추가 테스트 (gRPC 연동 ver.)




```
Select API test you want >> 1

======================== Test Progress ========================
To-do list:
{'id': 10, 'content': 'HW2', 'due_date': '2023-12-14', 'completed': False, 'prior_rank': 1}
{'id': 11, 'content': 'Go to a ski resort', 'due_date': '2023-12-20', 'completed': False, 'prior_rank': 2}
{'id': 12, 'content': 'Watch a movie', 'due_date': '2023-12-16', 'completed': False, 'prior_rank': 3}
{'id': 13, 'content': 'Computer N/W Final', 'due_date': '2023-12-11', 'completed': False, 'prior_rank': 1}
```

그림21. to-do list 전체 조회 테스트

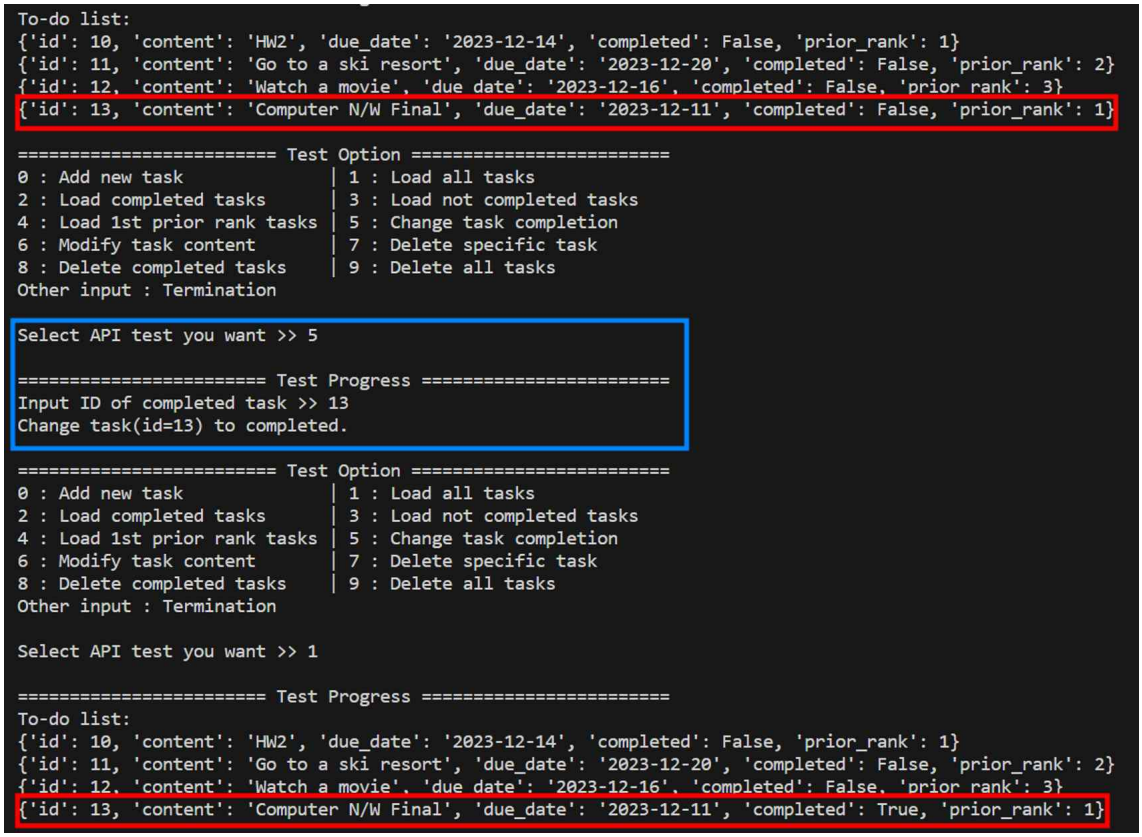


그림22. 특정 to-do를 완료로 변경하는 테스트

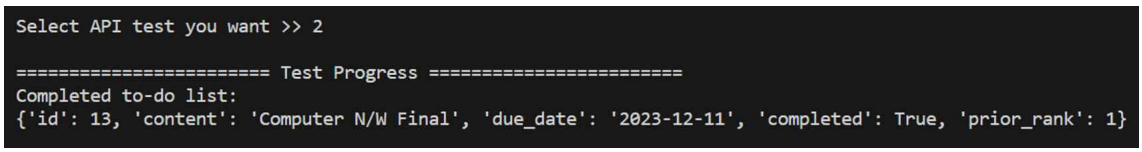


그림23. 완료된 to-do list만 조회하는 테스트

그림24. 미완료된 to-do list만 조회하는 테스트

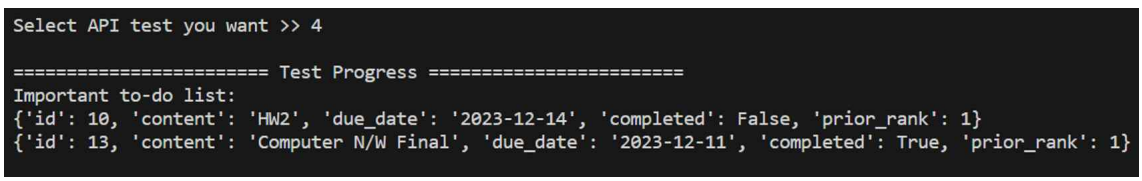


그림25. 우선순위가 가장 높은 to-do list만 조회하는 테스트

```
Select API test you want >> 6

======================== Test Progress ========================
Input ID you want to motify >> 11
Current todo: {'id': 11, 'content': 'Go to a ski resort', 'due_date': '2023-12-20', 'completed': False, 'prior_rank': 2}
Do you want to modify this? (y/n) >> y
Content >> Ski resort
Date (yyyy-mm-dd) >> 2023-12-23
Completion (y/n) >> n
Priority : 1(High), 2(Mid), 3(Low) >> 2
After todo: {'id': 11, 'content': 'Ski resort', 'due_date': '2023-12-23', 'completed': False, 'prior_rank': 2}
```

그림26. 특정 to-do의 상세내역을 수정하는 테스트

```
To-do list:
{'id': 10, 'content': 'HW2', 'due_date': '2023-12-14', 'completed': False, 'prior_rank': 1}
{'id': 11, 'content': 'Ski resort', 'due_date': '2023-12-23', 'completed': False, 'prior_rank': 2}
{'id': 12, 'content': 'Watch a movie', 'due_date': '2023-12-16', 'completed': False, 'prior_rank': 3}
{'id': 13, 'content': 'Computer N/W Final', 'due_date': '2023-12-11', 'completed': True, 'prior_rank': 1}

========================= Test Option =========================
0 : Add new task              | 1 : Load all tasks
2 : Load completed tasks      | 3 : Load not completed tasks
4 : Load 1st prior rank tasks | 5 : Change task completion
6 : Modify task content       | 7 : Delete specific task
8 : Delete completed tasks    | 9 : Delete all tasks
Other input : Termination

Select API test you want >> 8

======================== Test Progress ========================
Delete all completed tasks.

========================= Test Option =========================
0 : Add new task              | 1 : Load all tasks
2 : Load completed tasks      | 3 : Load not completed tasks
4 : Load 1st prior rank tasks | 5 : Change task completion
6 : Modify task content       | 7 : Delete specific task
8 : Delete completed tasks    | 9 : Delete all tasks
Other input : Termination

Select API test you want >> 1

======================== Test Progress ========================
To-do list:
{'id': 10, 'content': 'HW2', 'due_date': '2023-12-14', 'completed': False, 'prior_rank': 1}
{'id': 11, 'content': 'Ski resort', 'due_date': '2023-12-23', 'completed': False, 'prior_rank': 2}
{'id': 12, 'content': 'Watch a movie', 'due_date': '2023-12-16', 'completed': False, 'prior_rank': 3}
```

그림27. 완료된 to-do list 삭제하는 테스트

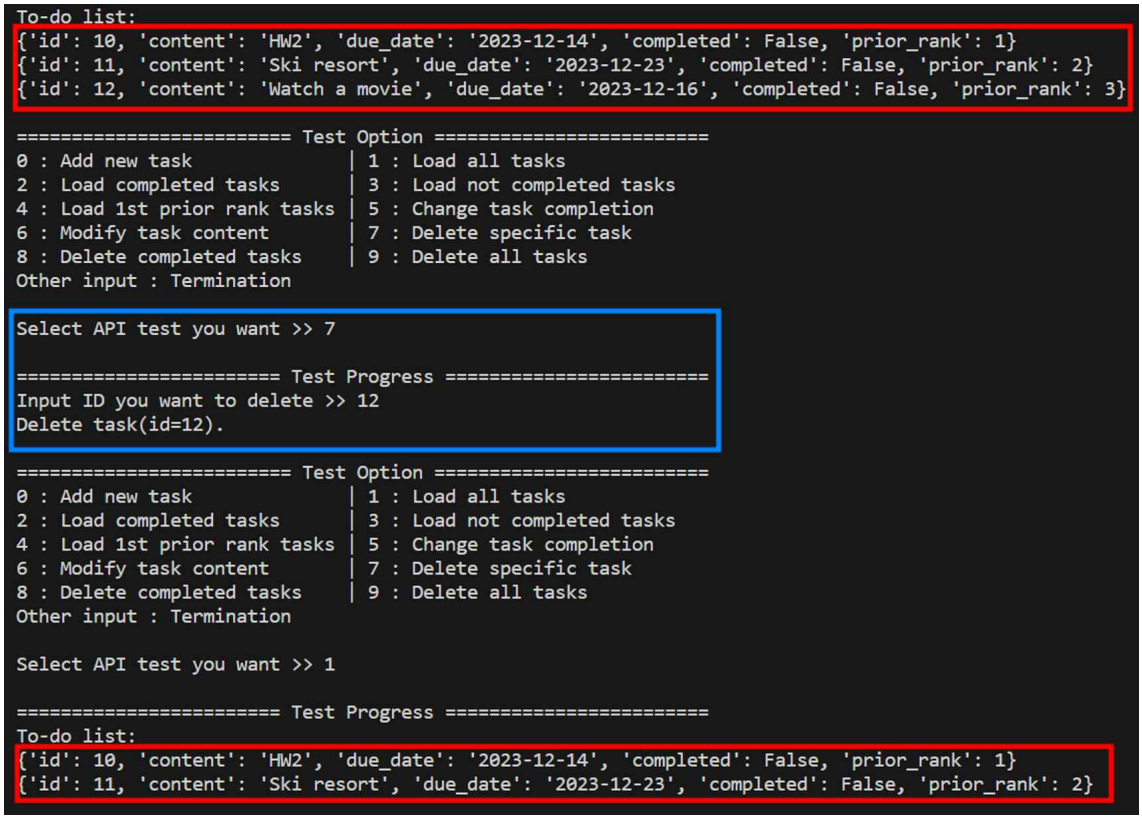


그림28. 특정 to-do를 삭제하는 테스트

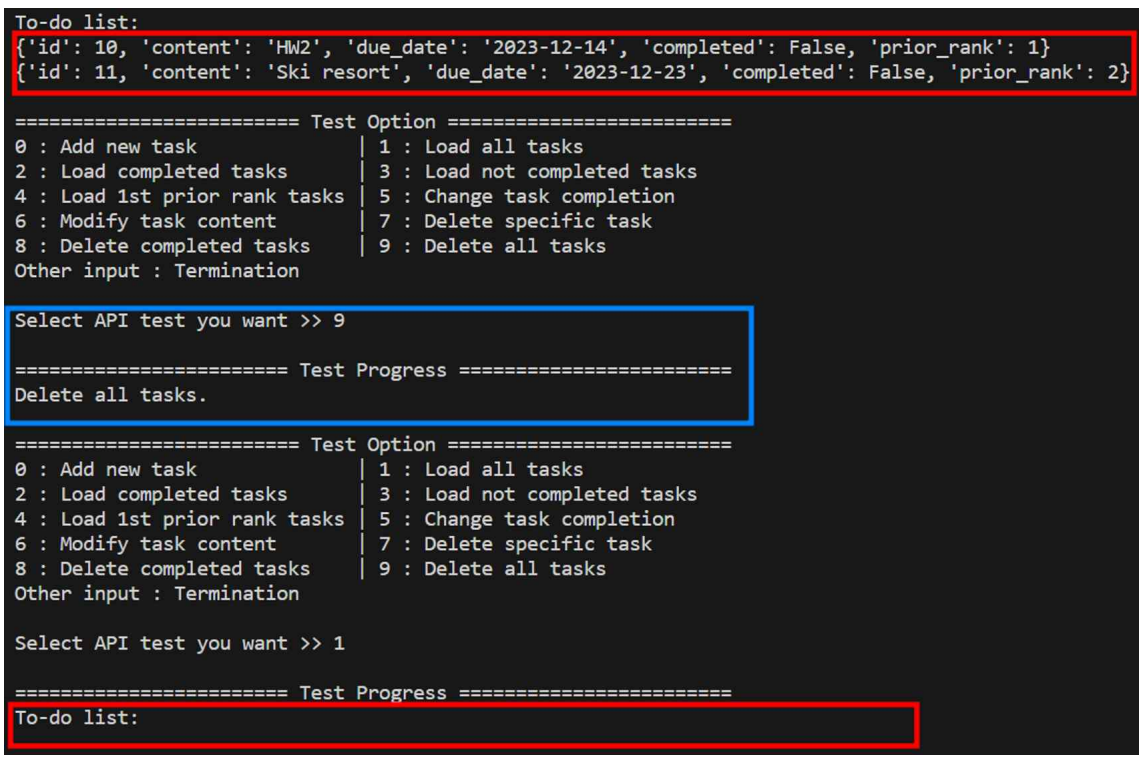


그림29. to-do list 전체를 삭제하는 테스트

```
Select API test you want >> 6

======================== Test Progress ========================
Input ID you want to motify >> 99
Error 404: Test failed

========================= Test Option =========================
0 : Add new task               | 1 : Load all tasks
2 : Load completed tasks       | 3 : Load not completed tasks
4 : Load 1st prior rank tasks | 5 : Change task completion
6 : Modify task content        | 7 : Delete specific task
8 : Delete completed tasks     | 9 : Delete all tasks
Other input : Termination

Select API test you want >> 7

======================== Test Progress ========================
Input ID you want to delete >> 100
Error 404: Test failed
```

그림30. url mapping 실패 예외처리 테스트

6. Conclusion

이번 프로젝트에서는 REST API, gRPC, MySQL을 활용하여 to-do list 프로그램을 구현해 보았는데, 현재 백엔드 개발에 필수적인 기술들을 실제로 적용해볼 수 있는 좋은 기회였다. 프로젝트의 핵심 부분 중 하나는 REST API와 gRPC를 구현하고 둘의 효율적인 연동이었는데, 이 부분을 공부하다 보니 서비스 하나하나와 이들의 상호작용이 프로그램의 전반적인 성능과 사용자 경험 측면에서도 큰 영향을 미친다는 것을 알았다. 그러나 이 연동이 완전히 완성되지 않았고, 원래 gRPC를 처음에 JAVA로 구현했지만 끝내 오류를 해결하지 못해 Python으로 똑같이 구현한 점이 정말 아쉬웠다. 지금까지 여러 프로젝트나 공모전에서 프론트엔드만 해온 입장으로써, 비록 간단한 백엔드 프로그램이고 완전하진 않았지만 처음으로 구현했고 테스트도 해보며 관련 주제들도 깊게 공부해봤다는 큰 의의가 있는 프로젝트였다. 결론적으로 백엔드 개발에 대한 중요한 통찰력을 얻을 수 있었고 이 과정에서 얻은 지식과 경험들은 향후 진행할 프로젝트를 넘어 더 나은 소프트웨어 개발자로 성장하는 데 도움이 될 거라 생각한다.