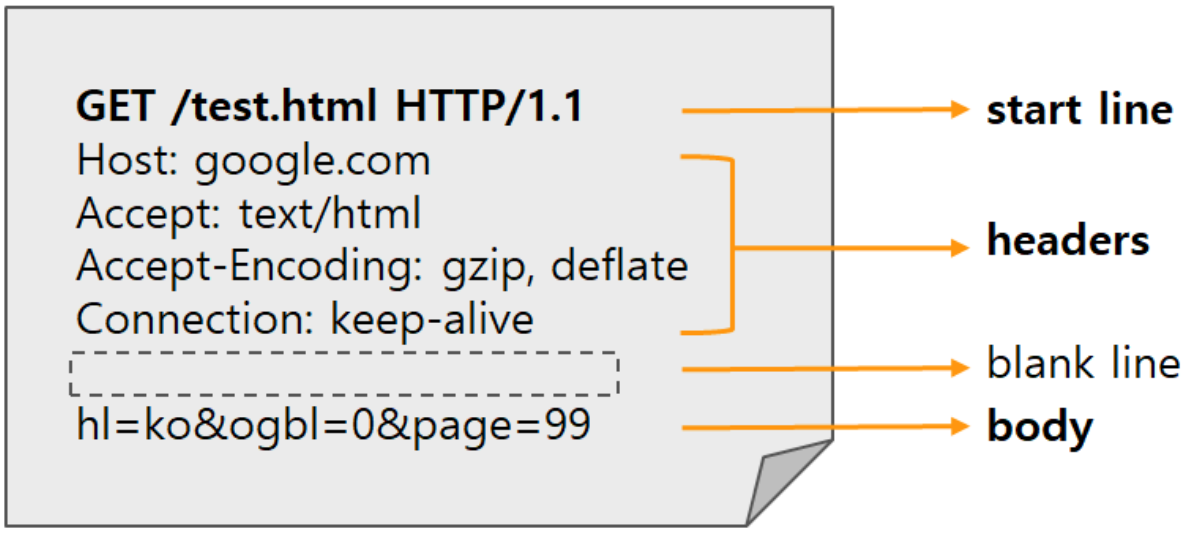
[과제 8] HTTP 요청/응답 메시지_문제

HTTP 요청 메시지의 구조는 웹의 기본이 되는 매우 중요한 개념

클라이언트(예: 브라우저)가 서버에게 요청할 때 보내는 메시지는 일정한 **포맷**을 따르며, 아래와 같은 **4가지 주요 구성 요소**로 이루어져 있음.

📦 HTTP 요청 메시지 구조

1. 요청라인 (Request Line)
2. 헤더 (Headers)
3. 빈 줄 (CRLF, 개행)
4. 메시지 바디 (Body, 선택적)



HTTP Request Message

✅ 1. 요청라인 (Request Line)

<메서드> <요청 URI> <HTTP 버전>

예시:

```
GET /index.html HTTP/1.1
```

항목	의미
`GET`, `POST`, `PUT`, `DELETE`	HTTP 메서드 (동작 종류)
`/index.html`	요청 대상 자원의 경로 (URI)
`HTTP/1.1`	사용하는 HTTP 프로토콜 버전

✅ 2. 헤더 (Headers)

요청에 대한 추가 정보를 담는 부분입니다.

형식: `헤더이름: 값`

예시:

```
Host: www.example.com
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
```

대표 헤더	설명
`Host`	요청 대상 서버 (도메인)
`User-Agent`	클라이언트 정보 (브라우저, OS 등)
`Content-Type`	바디에 담긴 데이터 형식
`Content-Length`	바디의 길이 (바이트 수)

✅ 3. 빈 줄 (CRLF)

- 헤더와 바디 사이를 구분하는 **빈 줄 하나** (`\r\n`)
- 이 줄이 없으면 서버는 "헤더가 끝났는지"를 알 수 없습니다

✅ 4. 메시지 바디 (Body)

- 주로 `POST`, `PUT` 요청에서 사용
- 클라이언트가 서버에 전달하려는 **데이터**가 담겨 있습니다

예시:

```
username=kimsc&password=1234
```

- 이 데이터는 `Content-Type` **헤더**에 따라 형식이 달라짐

Content-Type	데이터 형식
`application/x-www-form-urlencoded`	key=value&key2=value2 (폼)
`application/json`	{"key": "value"}
`multipart/form-data`	파일 업로드용

📄 전체 예시 (POST 요청)

```
POST /login HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 29

username=kimsc&password=1234
```

🧠 요약 도식

```
▶ Request Line
  POST /login HTTP/1.1

▶ Headers
  Host: www.example.com
  User-Agent: ...
  Content-Type: ...
  Content-Length: ...

▶ 빈 줄
```

```
▶ Body (선택)
  username=kimsc&password=1234
```

✅ 보충 설명

- `GET` 요청은 바디가 거의 없고, 데이터는 URL 쿼리 스트링에 포함됨
- `POST` 요청은 바디에 데이터를 담고 보냄

📦 HTTP 요청 메시지 구조 요약 (그래프 설명)

구성 요소	설명
요청라인 (Request Line)	`POST /login HTTP/1.1` 와 같이 요청의 시작 줄
헤더 (Headers)	`Host` , `User-Agent` , `Content-Type` 등
빈 줄 (CRLF)	헤더와 바디를 구분하는 공백 줄
본문 (Body, 선택적)	`username=kimsc&password=1234` 등 요청 데이터 포함 영역 (POST 등에서만 사용됨)

HTTP 응답 메시지

HTTP 응답 메시지는 클라이언트(예: 브라우저)가 서버에 요청을 보낸 후, 서버가 돌려주는 **응답의 형식**입니다.

아래에 구조, 구성 요소, 예시, 시각적 설명까지 **전체적으로 정리**해드릴게요.

📦 HTTP 응답 메시지 구조

```
1. 상태라인 (Status Line)
2. 헤더 (Headers)
3. 빈 줄 (CRLF)
4. 메시지 바디 (Body, 선택적)
```

HTTP Response Message

✅ 1. 상태라인 (Status Line)

```
HTTP/1.1 200 OK
```

구성 요소	설명
`HTTP/1.1`	HTTP 프로토콜 버전
`200`	상태 코드 (성공, 실패 등 숫자로 표현)
`OK`	상태 메시지 (상태 코드에 대한 설명)

📌 주요 상태 코드 예시

코드	의미
`200 OK`	요청 성공
`301 Moved Permanently`	리다이렉션
`400 Bad Request`	클라이언트 오류
`401 Unauthorized`	인증 실패
`403 Forbidden`	권한 없음
`404 Not Found`	리소스 없음
`500 Internal Server Error`	서버 내부 오류

✅ 2. 헤더 (Headers)

```
Content-Type: text/html; charset=UTF-8
Content-Length: 1234
Set-Cookie: sessionid=abc123
```

- 응답에 대한 부가 정보
- 클라이언트가 **응답을 어떻게 해석**할지 알려줌

대표 헤더	설명
`Content-Type`	응답 본문의 형식 (`text/html`, `application/json` 등)
`Content-Length`	본문의 길이 (바이트 수)
`Set-Cookie`	클라이언트에 쿠키 저장 요청
`Cache-Control`	캐시 관련 정책

✅ 3. 빈 줄 (`\r\n`)

- 헤더와 본문을 구분하는 **구분선**
- 반드시 **한 줄 빈 줄** 있어야 서버가 헤더의 끝을 인식

✅ 4. 메시지 바디 (Body, 선택적)

```
<html>
  <body>
    <h1>환영합니다!</h1>
  </body>
</html>
```

- 실제 콘텐츠가 담기는 곳 (HTML, JSON, 이미지, 파일 등)
- `Content-Type`에 따라 클라이언트가 이를 해석

📄 전체 HTTP 응답 예시

```
HTTP/1.1 200 OK
Date: Wed, 17 Apr 2025 12:00:00 GMT
Content-Type: text/html; charset=UTF-8
```

```
Content-Length: 56

<html><body><h1>Hello, World!</h1></body></html>
```

🧠 요약 도식

구성 요소	설명
상태라인	HTTP 버전 + 상태 코드 + 상태 메시지
헤더	본문 정보, 쿠키 등 메타데이터
빈 줄	헤더와 본문 구분
바디	실제 응답 내용 (웹페이지, JSON 등)

Django HTTP 응답

Django에서 HTTP 응답을 생성하는 방법은 여러 가지가 있지만, 가장 기본적인 것부터 JSON, 파일 다운로드까지 단계별로 정리해서 보여드릴게요.

🎯 1. 기본 HTML 응답

```
from django.http import HttpResponse

def hello_view(request):
    return HttpResponse("<h1>Hello, Django!</h1>")
```

- 기본적인 문자열(html 포함)을 반환
- `Content-Type: text/html` 자동 설정

🎯 2. 템플릿 렌더링 응답

```
from django.shortcuts import render

def greeting_view(request):
    context = {'name': '홍길동'}
    return render(request, 'myapp/greeting.html', context)
```

- `greeting.html` 템플릿을 렌더링해서 반환
- 내부적으로 `HttpResponse`로 반환됨

예시 템플릿 (`greeting.html`):

```
<h2>안녕하세요, {{ name }}님!</h2>
```

🎯 3. JSON 응답

```
from django.http import JsonResponse

def api_data(request):
    data = {'status': 'ok', 'message': '요청 성공'}
    return JsonResponse(data)
```

- `Content-Type: application/json` 자동 설정
- `dict` 또는 `list` 형태만 반환 가능

🎯 4. 파일 다운로드 응답

```
from django.http import FileResponse

def download_file(request):
    filepath = 'static/files/example.pdf'
    return FileResponse(open(filepath, 'rb'), as_attachment=True, filename
='example.pdf')
```

- 브라우저에서 **파일 다운로드** 창이 열림
- `as_attachment=True`가 핵심

🎯 5. 상태 코드 변경 응답

```
from django.http import HttpResponseNotFound

def not_found_view(request):
```

```
    return HttpResponseNotFound("<h1>404 - 페이지를 찾을 수 없습니다</h1>")
```

또는 수동으로:

```
return HttpResponse("권한 없음", status=403)
```

🧠 요약

응답 종류	클래스	Content-Type
일반 응답	`HttpResponse`	`text/html` 등
템플릿	`render()`	`text/html`
JSON	`JsonResponse`	`application/json`
파일 다운로드	`FileResponse`	바이너리 (`application/octet-stream` 등)
에러 응답	`HttpResponseNotFound`, `HttpResponseForbidden` 등	다양함

✅ 응답 헤더 설정 추가

```
response = HttpResponse("Hello")
response['X-Custom-Header'] = 'MyHeaderValue'
return response
```

DRF API 만들기

장고(Django) 프로젝트를 처음부터 설치하고 Django REST Framework(DRF)로 API를 만드는 예제

🚀 전체 단계

1. 가상환경 생성 (선택)
2. Django 프로젝트 설치 및 시작
3. 앱 생성 및 등록

4. Django REST Framework 설치
5. API 뷰 생성 (`GET` 응답)
6. URL 연결 및 테스트

✅ 1단계: 가상환경 만들기 (선택이지만 권장)

```
python -m venv venv
source venv/bin/activate     # macOS/Linux
venv\Scripts\activate        # Windows
```

✅ 2단계: Django 설치 및 프로젝트 생성

📦 Django 설치

```
pip install django
```

🏗️ 프로젝트 생성

```
django-admin startproject myproject
cd myproject
```

🎉 기본 구조

```
myproject/
├── manage.py
└── myproject/
    ├── __init__.py
    ├── settings.py ← 환경설정
    ├── urls.py     ← 전체 URL 관리
    └── wsgi.py
```

✅ 3단계: 앱 생성 및 등록

```
python manage.py startapp myapp
```

➕ `settings.py`에 앱 등록

```
INSTALLED_APPS = [
    ...
    'myapp',
]
```

✅ 4단계: Django REST Framework 설치 및 설정

```
pip install djangorestframework
```

➕ `settings.py`에 등록

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

✅ 5단계: API 뷰 생성

📄 `myapp/views.py`

```
from rest_framework.decorators import api_view
from rest_framework.response import Response

@api_view(['GET'])
def hello_api(request):
    return Response({"message": "안녕하세요, DRF API입니다!"})
```

✅ 6단계: URL 연결

📄 `myapp/urls.py` 생성

```
from django.urls import path
from .views import hello_api

urlpatterns = [
    path('hello/', hello_api),
]
```

📄 `myproject/urls.py` 수정

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('myapp.urls')),  # myapp의 API 라우팅 연결
]
```

✅ 7단계: 서버 실행 & 테스트

```
python manage.py runserver
```

- 브라우저에서:

  http://127.0.0.1:8000/api/hello/

- 응답:

```
{
  "message": "안녕하세요, DRF API입니다!"
}
```

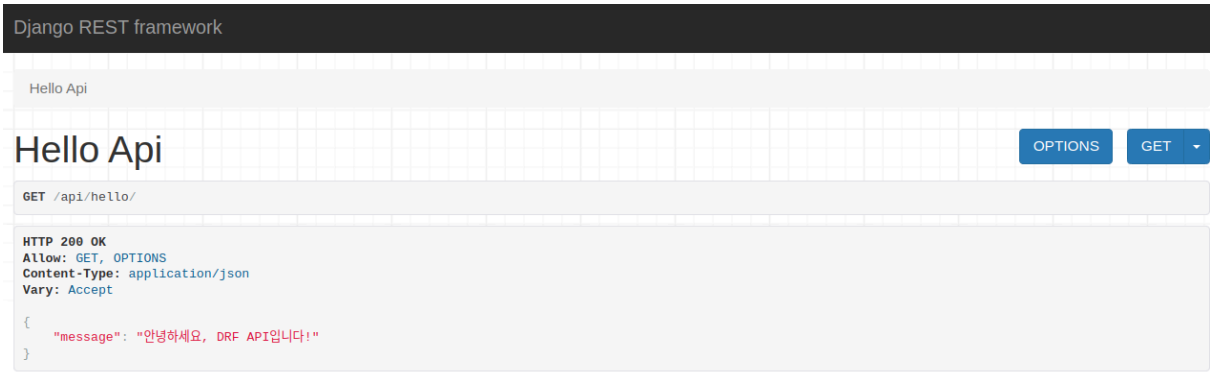
🧠 정리 요약

단계	내용

1	가상환경 만들기
2	Django 설치 및 프로젝트 생성
3	앱 생성 + settings.py 등록
4	DRF 설치 및 등록
5	`@api_view()` 로 뷰 생성
6	`urls.py` 로 경로 연결
7	서버 실행 후 브라우저에서 확인

결과화면

http://127.0.0.1:8040/api/hello/



Django REST Framework에서 POST 요청을 처리하는 예제

클라이언트가 보낸 데이터를 서버가 받아서 저장하거나 확인하는 과정을 다루는 예제

🎯 목표

- 클라이언트가 **JSON 형식으로 데이터 전송**
- 서버는 **POST 요청을 받아서 데이터 추출**
- Response로 **성공/실패 메시지 반환**

✅ 기본 구조

예: 사용자가 이름(name)과 메시지(message)를 전송하면,

```
{
  "name": "홍길동",
  "message": "안녕하세요!"
}
```

→ 서버는 이를 받아 JSON 응답으로 처리

📄 1. `views.py` – POST 요청 처리 : 총 4점

```
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status

@api_view(['POST'])
def receive_post(request):
    name = request.data.get('name')
    message = request.data.get('message')

    if not name or not message:
        return Response(
            {"error": "name과 message는 필수입니다."},
            status=status._____________  # ← 여기에 동작코드를 작성하세요(2점)
        )

    return Response(
        {"result": f"{name}님이 보낸 메시지: {message}"},
        status=status._____________ # ← 여기에 동작코드를 작성하세요(2점)
    )
```

🔍 설명

코드	의미
`request.data.get()`	POST로 전송된 데이터 추출
`Response(...)`	응답 반환
`status.HTTP_400_BAD_REQUEST`	필드가 없을 때 에러 코드
`status.HTTP_201_CREATED`	정상 저장/생성 응답 코드

📄 2. `urls.py` 연결 : 총 2점

```
from django.urls import path
from .views import receive_post

urlpatterns = [
    path('post/', _____________), # ← 여기에 동작코드를 작성하세요(2점)
]
```

✅ 3. Postman

Postman 설정

- URL: `http://localhost:8000/api/post/`
- Method: `POST`
- Body → `raw` 선택 → `JSON` 형식

```
{
  "name": "홍길동",
  "message": "DRF는 쉽다!"
}
```

✅ 4. 서버 응답

성공 시:

```
{
  "result": "홍길동님이 보낸 메시지: DRF는 쉽다!"
}
```

실패 시:

```
{
  "error": "name과 message는 필수입니다."
```

```
}
```

🧠 요약

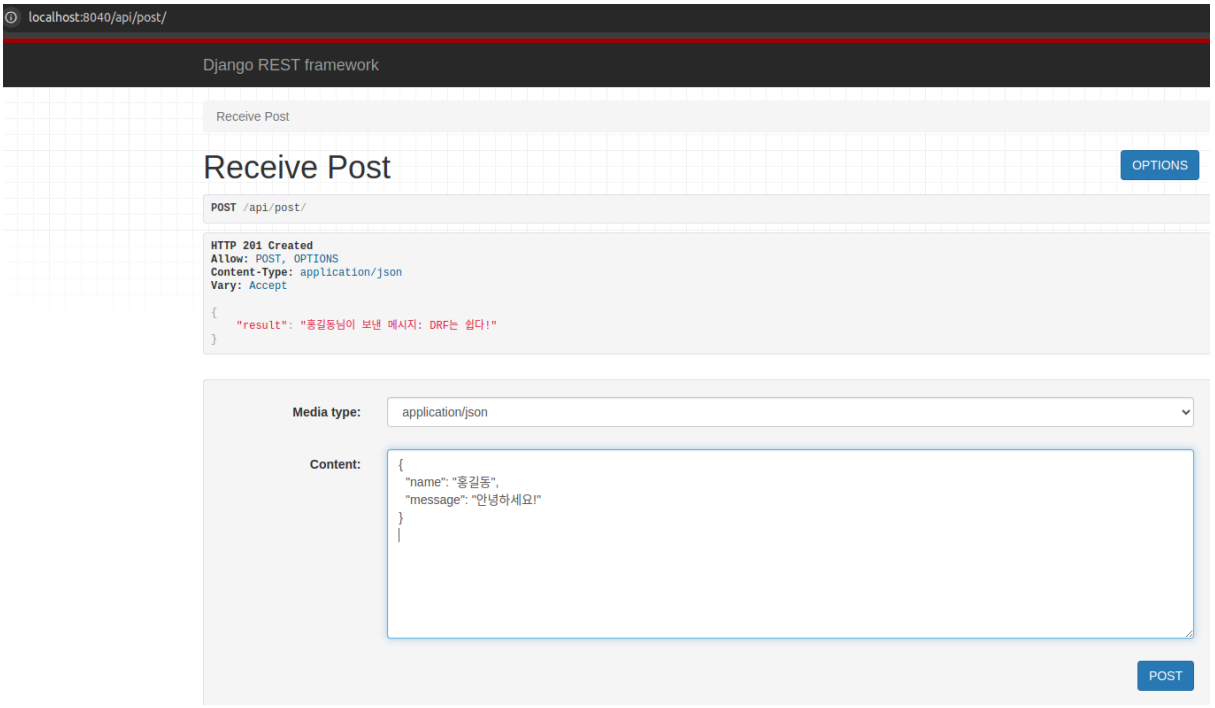
항목	설명
`@api_view(['POST'])`	POST 요청만 허용
`request.data`	클라이언트에서 전송한 JSON 데이터
`Response()`	JSON 응답 생성
`status`	응답 상태 코드 명시

< 과제 8 내용>

- 제출 내용 및 총점 : 총 10점
  - 1. `views.py` – POST 요청 처리 (총 4점)
  - 2. `urls.py` 연결 (총 2점)
  - 정상 응답 웹사이트 결과 화면 캡쳐 (1점)
  - 비정상 응답 웹사이트 결과 화면 캡쳐 (1점)
  - 구현 코드 깃허브 링크 (1점)
  - 위 내용을 정리한 문서 (노션 등) 제출 (1점)
- **제출일시: 2025년 5월 14일 (수) 13:30**
  - 가상강의실에 제출
  - 늦게 제출은 인정하지 않습니다

정상 응답 화면 캡쳐 (1점)

- HTTP 201 Created



비정상 응답 화면 캡쳐 (1점)

- HTTP 400 Bad Request