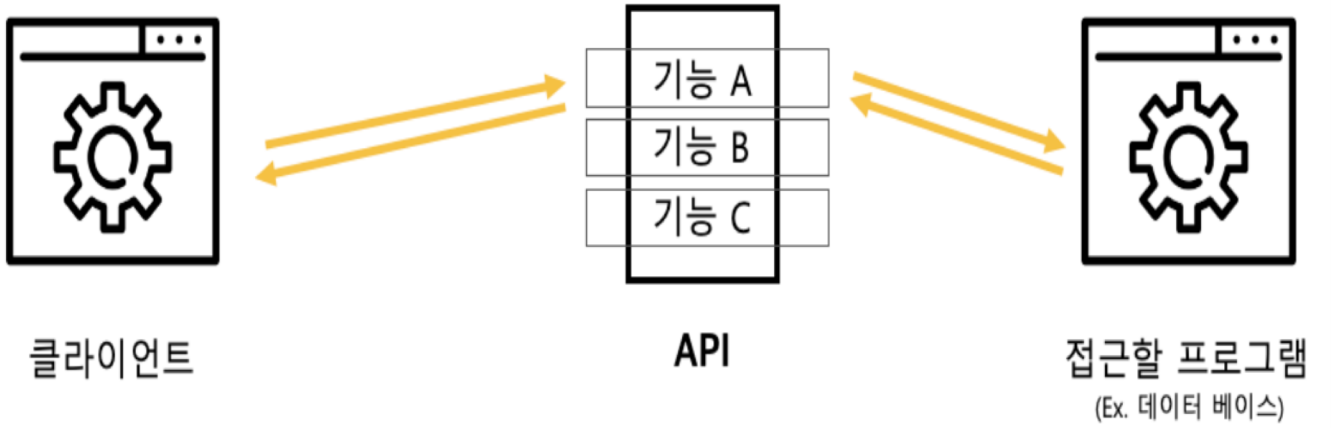
REST API

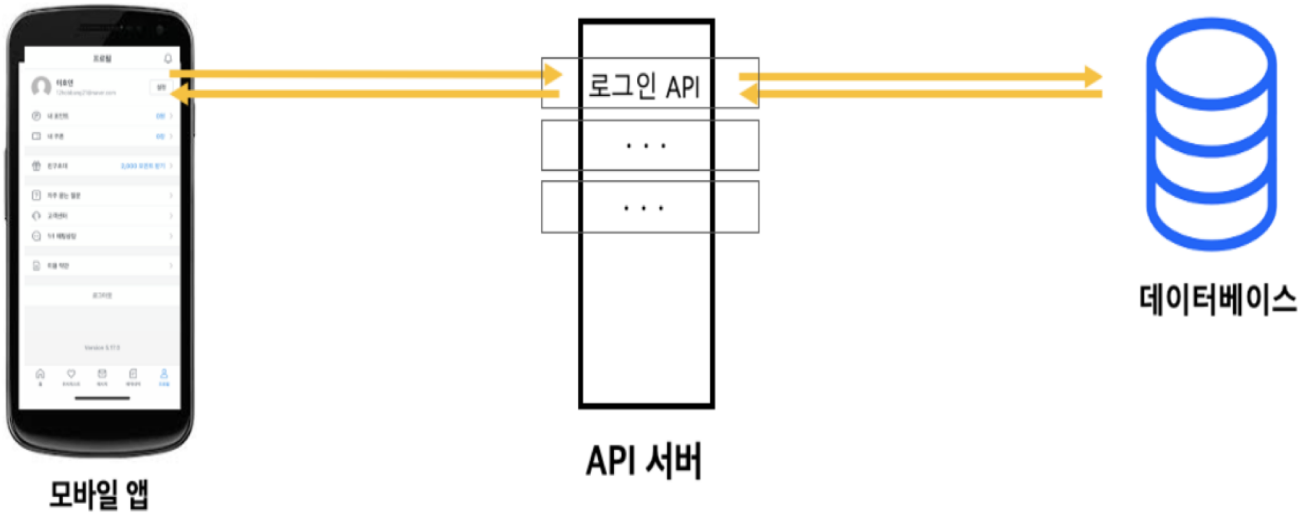
REST API

- API(Application Programming Interface)
  - ✓ 개념
    - 프로그램과 프로그램을 연결시켜주는 매개체
    - 프로그램끼리 통신을 하기 위해선 프로그램을 만드는 개발자가 해당 프로그램이 잘 통신할 수 있도록 규칙들을 잘 설계하는 게 중요
    - 접근할 프로그램의 규칙이 잘 짜이지 않고 복잡한 경우 나 프로그램 보안 상 외부에서 누구나 사용할 수 없고 제한된 기능들을 간접적으로 제공하고 싶을 때 이용하는 경우가 많음



REST API

- API(Application Programming Interface)
  - ✓ 개념
    - 대신할 프로그램의 기능들을 미리 정리해서 규칙을 잘 세워둔다면 클라이언트는 접근할 프로그램을 모르더라도 API를 이용해 손쉽게 통신을 할 수 있음

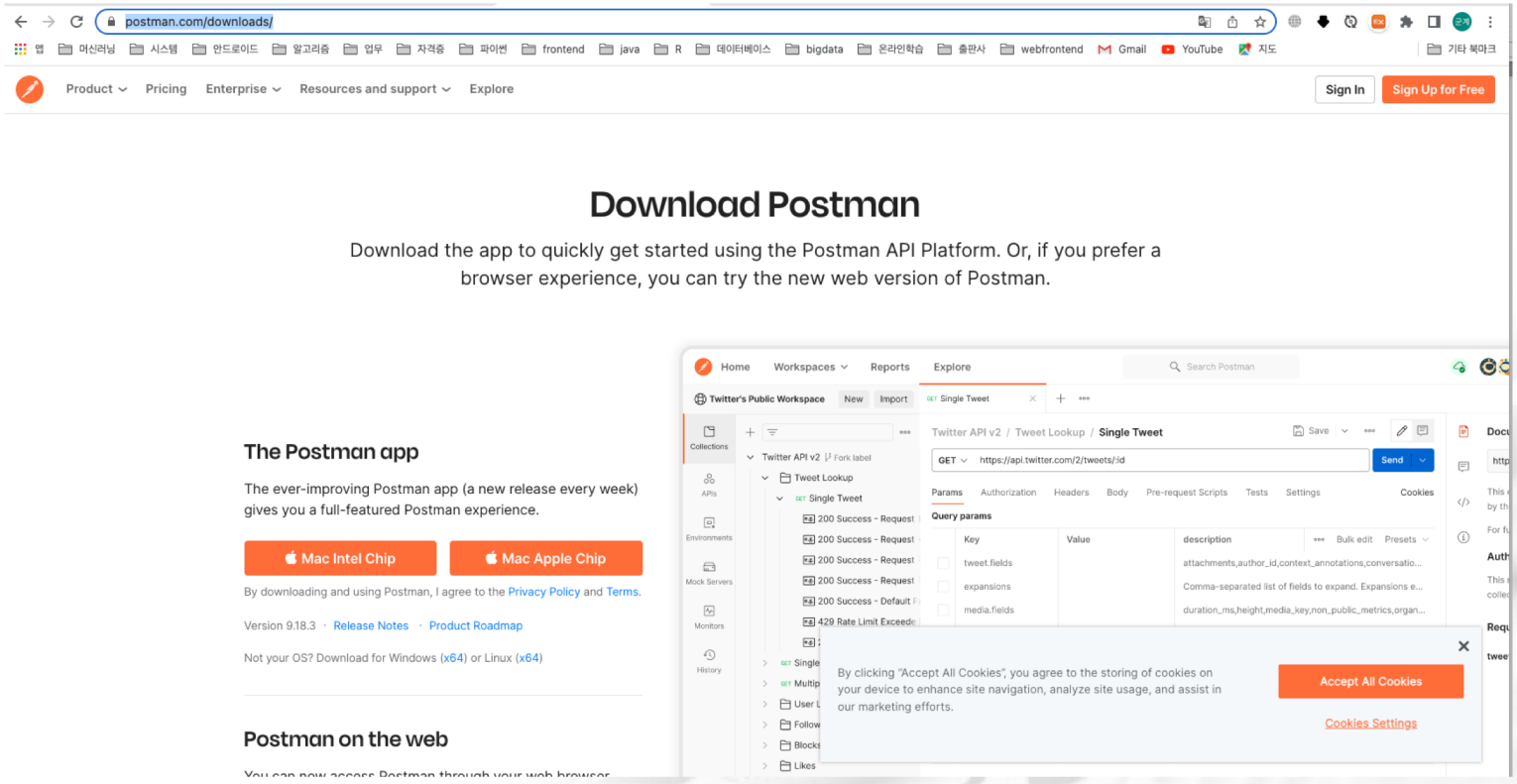


REST API

- CSR
  - ✓ API 서버는 화면 구성은 별도의 클라이언트 프로그램에서 처리하고 서버에서는 순수한 데이터만을 전송하는데 이러한 구성이 클라이언트 사이드 렌더링(Client Side Rendering - CSR)
  - ✓ CSR 방식은 클라이언트에서 데이터를 가공해서 화면에 보여주기 때문에 데이터를 어떻게 구성해서 주고받을 것인지가 중요한데 주로 JSON/XML 포맷으로 데이터를 구성하고 호출 방식은 REST 방식을 이용하는 경우가 많음
  - ✓ API 서버는 화면을 구성하지 않는다는 특징 외에도 무상태(stateless) 라는 특징도 있는데 REST나 HTTP의 특징이지만 개발에 약간의 차이가 있기 때문에 주의가 필요
  - ✓ API 서버는 데이터를 요청하고 응답받는 방식으로 구성

REST API

- REST API 테스트를 위한 애플리케이션
  - ✓ https://www.postman.com/downloads/ 에서 다운로드 받아서 설치



REST API

- REST
  - ✓ 특징
    - REST(REpresentational State Transfer)는 월드 와이드 웹과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식
    - 엄격한 의미로 REST는 네트워크 아키텍처 원리의 모음
    - 네트워크 아키텍처 원리란 자원을 정의하고 자원에 대한 주소를 지정하는 방법 전반을 일컫는데 웹 상의 자료를 HTTP 위에서 SOAP(Simple Object Access Protocol) 나 쿠키를 통한 세션 트랙킹 같은 별도의 전송 계층 없이 전송하기 위한 아주 간단한 인터페이스를 의미
    - REST 원리를 따르는 시스템은 종종 RESTful이란 용어로 지칭됨

REST API

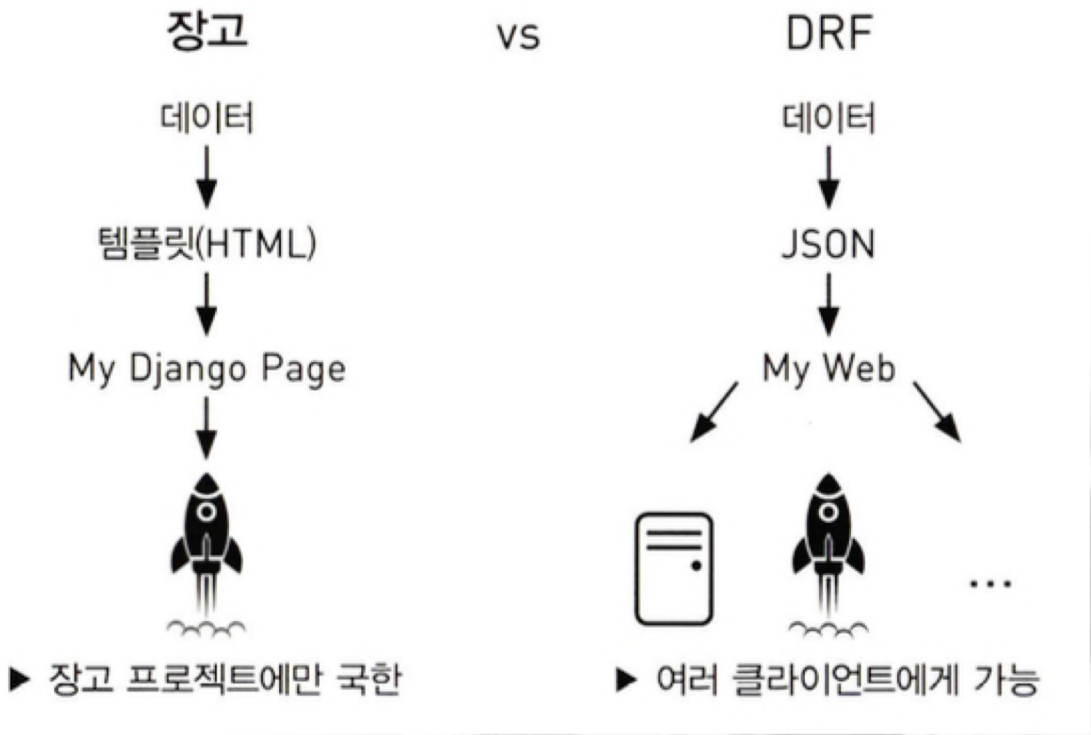
- REST API
  - ✓ 6가지 제약 조건
    - Client-Server
      - 클라이언트-서버라는 것은 리소스를 관리하는 서버가 존재하고 다수의 클라이언트가 리소스를 소비하려고 네트워크를 통해 서버에 접근하는 구조를 의미
      - 가장 친숙한 것이 바로 웹 애플리케이션
    - Stateless
      - 상태가 없다는 것은 클라이언트가 서버에 요청을 보낼 때 이전 요청의 영향을 받지 않음을 의미
    - Cacheable Data
      - 서버에서 리소스를 리턴할 때 캐시가 가능한지 아닌지 명시할 수 있어야 함
      - HTTP에서는 cache-control이라는 헤더에 리소스의 캐시 여부를 명시할 수 있음
    - Layered System
      - 클라이언트가 서버에 요청을 할 때 여러 개의 레이어로 된 서버를 거칠 수 있음
      - 서버가 인증 서버, 캐싱 서버, 로드 밸런서를 거쳐서 최종적으로 애플리케이션에 도착한다고 가정했을 때 이 사이의 레이어들은 요청과 응답에 어떤 영향을 미치지 않으며 클라이언트는 서버의 레이어 존재 유무를 알지 못함

REST API

- REST API
  - ✓ 6가지 제약 조건
    - A Uniform Interface
      - 리소스에 접근할 때 인터페이스가 일관적이어야 한다는 의미
      - Todo 아이템을 가져오려고 http://fsoftwareengineer.com/todo를 사용했는데 Todo 아이템을 업데이트하는 데 http://fsoftwareengineer2.com/todo 사용해야 한다면 이는 일관적인 인터페이스가 아님
      - http://fsoftwareengineer.com/todo는 JSON 형식의 리소스를 리턴했는데 http://fsoftwareengineer.com/account는HTML을 리턴하는 경우 리턴 타입에 일관성이 있다고 할 수 없음
      - 리소스에 접근하는 방식, 요청 형식, 그리고 응답 형식 즉 URI, 요청의 형태와 응답의 형태가 애플리케이션 전반에 걸쳐 일관적이어야 한다는 것이 일관적인 인터페이스 방침
      - 서버가 리턴하는 응답에는 해당 리소스를 수정할 수 있는 충분한 정보가 있어야 하는데 Todo 아이템을 받아왔는데 ID가 없다면 이후 클라이언트는 Todo 아이템을 업데이트하거나 삭제하지 못하게 되는데 이는 리소스를 수정하는 데 충분한 정보가 부족한 것
    - Code-On-Demand
      - 선택 사항으로 클라이언트는 서버에 코드를 요청할 수 있고 서버가 리턴한 코드를 실행할 수 있음

REST API

- Django REST Framework
  - ✓ REST API를 만들 수 있는 프레임워크
  - ✓ 설치: pip install djangorestframework



Django REST Framework

- 기본 설정
  - ✓ 가상 환경 생성
  - ✓ 필요한 패키지 설치
    - pip install django
    - pip install djangorestframework
    - pip install mysqlclient
  - ✓ 프로젝트 와 앱 생성
    - 프로젝트 생성: django-admin startproject apiserver .
    - 앱 생성: python manage.py startapp apiapp

Django REST Framework

- 기본 설정
  - ✓ settings.py 수정

```
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = "django-insecure-
    =5!6(l=%dj(=f5uf2l+w7f4&h5)%d_k9_x$2+i1gto0%@1&hyj"

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []
```

Django REST Framework

- 기본 설정
  - ✓ settings.py 수정

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "rest_framework",
    "apiapp",
]
```

Django REST Framework

- 기본 설정
  - ✓ settings.py 수정

```
MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
```

Django REST Framework

- 기본 설정
  - ✓ settings.py 수정

```
ROOT_URLCONF = "apiserver.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ],
        },
    },
]

WSGI_APPLICATION = "apiserver.wsgi.application"
```

Django REST Framework

- 기본 설정
  - ✓ settings.py 수정

```
# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'adam',
        'USER': 'root',
        'PASSWORD': 'wnddkd',
        'HOST': '127.0.0.1',
        'PORT': ''
    }
}
```

Django REST Framework

- 기본 설정
  - ✓ settings.py 수정

```
# Password validation
# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {"NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",},
    {"NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",},
    {"NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",},
]
```

Django REST Framework

- 기본 설정
  - ✓ settings.py 수정

```
# Internationalization
# https://docs.djangoproject.com/en/4.1/topics/i18n/

LANGUAGE_CODE = "en-us"
TIME_ZONE = "Asia/Seoul"
USE_I18N = True
USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.1/howto/static-files/

STATIC_URL = "static/"

# Default primary key field type
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"
```

Django REST Framework

- ❖ 기본 요청 처리
  - ✓ URL 처리 설정
    - ● 프로젝트의 urls.py 파일을 수정

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

#example 로 시작하는 url은 api.app의 urls.py 파일에서 처리하도록 설정
urlpatterns = [
    path("admin/", admin.site.urls),
    path("example/", include("apiapp.urls"))
]
```

Django REST Framework

- 기본 요청 처리
  - ✓ apiapp 앱에 urls.py 파일을 추가하고 작성

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

urlpatterns = [
    path("hello/", helloAPI),
]
```

Django REST Framework

- 기본 요청 처리
  - ✓ apiapp 앱에 views.py 파일에 helloAPI 함수 작성

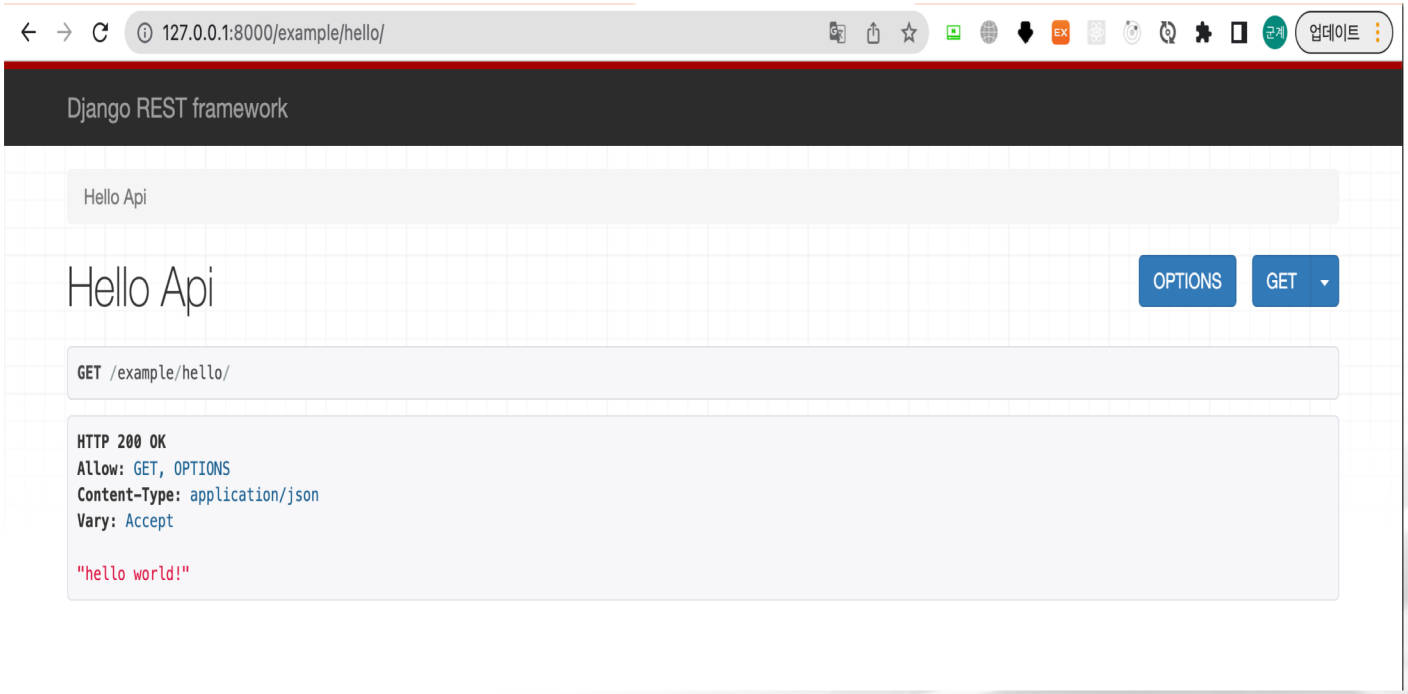
```
from rest_framework.response import Response
from rest_framework.decorators import api_view

@api_view(['GET'])
def helloAPI(request):
    return Response("hello world!")
```

Django REST Framework

- 기본 요청 처리
  - ✓ Response
    - 응답 결과를 위한 클래스
    - 객체 생성 시 첫번째 매개변수는 전달할 데이터이고 status는 상태 코드
    - 주요 상태 코드
      - 200: OK
      - 201: Created – 요청은 처리되어서 새로운 리소스를 생성
      - 202: Accepted – 요청은 접수했지만 처리가 완료되지 않음
      - 3xx: Redirection – 클라이언트는 요청을 마치기 위해서 추가적인 동작을 수행
      - 301: Moved Permanently – 지정한 리소스가 새로운 URI로 이동
      - 303: See Other – 다른 위치로 요청
      - 307: Temporary Redirect – 임시로 리다이렉션 요청이 필요
      - 400: 잘못된 요청
      - 401: 권한이 없음
      - 403: 권한 처리 이외의 사유로 리소스에 대한 엑세스가 금지
      - 404: 지정한 리소스를 찾을 수 없는 경우
      - 500: 내부 서버 오류

Django REST Framework

- 기본 요청 처리
  - ✓ 프로젝트 실행 흐 브라우저에서 http://127.0.0.1:8000/example/hello 로 접속



Django REST Framework

- ❖ 기본 요청 처리
  - ✓ Django 와 다른 점
    - 실행 화면에 보이는 부분이 HTTP의 헤더
    - HTTP는 웹의 Front End 와 Back End 간 요청/응답을 위한 프로토콜
    - HTTP가 요청/응답을 할 때 누구나 같은 방식으로 통신을 하기 때문에 통일된 규격이 필요하고 그 규격 내에 있는 주요 정보를 담아 놓는 것이 HTTP의 헤더인데 HTTP 뿐만 아니라 모든 프로토콜에 공통적으로 해당
    - HTTP의 경우 요청/응답 에 대한 결과 상태 및 데이터의 타입 등을 포함하는데 상태 코드가 HTTP 헤더에 포함되는 내용
    - DRF는 Response를 제공하는 API의 형태로 결과물을 전송
    - 템플릿의 형태가 아닌 JSON과 같은 형태의 응답을 제공하게 되는데 이 경우 템플릿을 대신할 무언가가 필요하게 되는데 DRF에서는 Serializer가 그 역할을 수행

Django REST Framework

- 기본 요청 처리
  - ✓ Serializer(직렬화)
    - Model을 JSON 데이터로 변환하는 작업
    - Model 데이터는 Python 객체의 형태라서 이 데이터를 클라이언트에게 직접 전송하게 되면 클라이언트가 해석을 할 수 없기 때문에 클라이언트에게 데이터를 전송할 때는 데이터를 표현하는 공통된 방식 중의 하나인 JSON 문자열로 변환해서 전달해야 하는데 이렇게 객체를 JSON 문자열로 변환하는 작업이 직렬화
    - 반대로 JSON 문자열을 Model의 형태로 변경하는 것이 역 직렬화인데 이 작업은 클라이언트가 문자열로 데이터를 전송하는 경우 서버가 Model의 형태로 변경해서 사용하는 작업

도서 정보 API

- 모델 생성
  - ✓ 도서 정보를 위한 모델을 생성 – apiapp/model.py

```
from django.db import models

class Book(models.Model):
    bid = models.IntegerField(primary_key=True)
    title = models.CharField(max_length=50)
    author = models.CharField(max_length=50)
    category = models.CharField(max_length=50)
    pages = models.IntegerField()
    price = models.IntegerField()
    published_date = models.DateField()
    description = models.TextField()
```

도서 정보 API

- ❖ 모델 생성
  - ✓ 도서 정보를 위한 모델을 생성하기 위한 명령 수행
    - python manage.py makemigrations apiapp
    - python manage.py migrate

도서 정보 API

❖ 모델 생성

✓ 확인

desc apiapp_book;

Field	Type	Null	Key	Default	Extra
bid	int(11)	NO	PRI	[NULL]	
title	varchar(50)	NO		[NULL]	
author	varchar(50)	NO		[NULL]	
category	varchar(50)	NO		[NULL]	
pages	int(11)	NO		[NULL]	
price	int(11)	NO		[NULL]	
published_date	date	NO		[NULL]	
description	longtext	NO		[NULL]	

도서 정보 API

- 직렬화
  - ✓ apiapp 애플리케이션 디렉토리에 serializers.py 파일을 생성하고 작성

```
from rest_framework import serializers
from .models import Book

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = ['bid', 'title', 'author', 'category', 'pages', 'price', 'published_date',
 'description',]
```

도서 정보 API

- API 처리
  - ✓ views.py 파일에 요청을 처리하는 함수를 작성

```
from rest_framework import status
from rest_framework.response import Response
from rest_framework.decorators import api_view
from rest_framework.generics import get_object_or_404
from .models import Book
from .serializers import BookSerializer

@api_view(['GET'])
def helloAPI(request):
    return Response("hello world!")
```

도서 정보 API

- API 처리
  - ✓ views.py 파일에 요청을 처리하는 함수를 작성

```
@api_view(['GET', 'POST'])
#GET 방식으로 요청할 때는 모든 데이터를 가져와서 리턴하고
#POST 방식으로 요청한 경우에는 데이터를 추가
def booksAPI(request):
    if request.method == 'GET':
        books = Book.objects.all()
        serializer = BookSerializer(books, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)
    elif request.method == 'POST':
        serializer = BookSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

도서 정보 API

- API 처리
  - ✓ views.py 파일에 요청을 처리하는 함수를 작성

```
#bid에 해당하는 하나의 데이터를 찾아서 리턴
@api_view(['GET'])
def bookAPI(request, bid):
    book = get_object_or_404(Book, bid=bid)
    serializer = BookSerializer(book)
    return Response(serializer.data, status=status.HTTP_200_OK)
```

도서 정보 API

- url 연결
  - ✓ apiapp 디렉토리의 urls.py 파일에 요청을 처리하는 함수 와 URL을 연결

```
from django.urls import path
from .views import helloAPI, bookAPI, booksAPI

urlpatterns = [
    path("hello/", helloAPI),
    path("fbv/books/", booksAPI),
    path("fbv/book/<int:bid>/", bookAPI),
]
```

도서 정보 API

❖ 실행

✓ 실행 한 후 브라우저에서 http://127.0.0.1:8000/example/fbv/books/ 를 요청해서 입력 후 POST 클릭

```
GET /example/fbv/books/
```

```
HTTP 200 OK
Allow: OPTIONS, POST, GET
Content-Type: application/json
Vary: Accept

[]
```

Media type: application/json

Content:

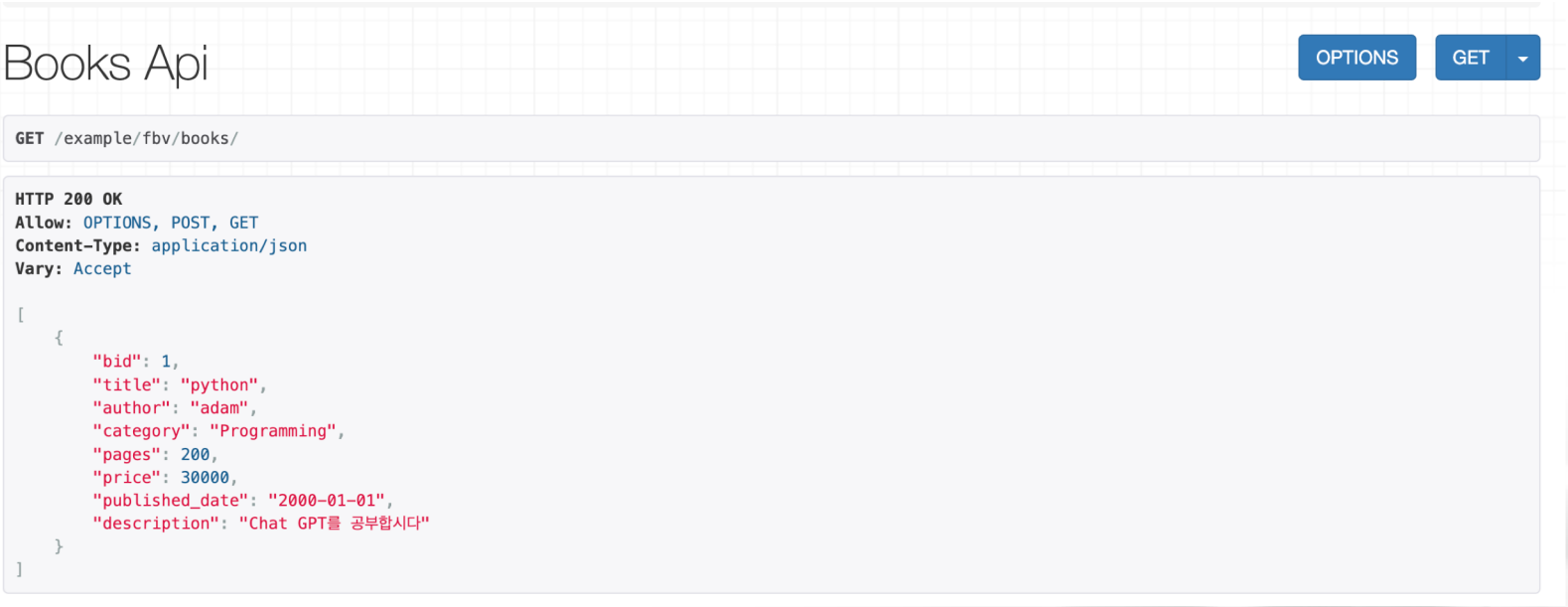
```
{
"bid":1,
"title": "처음 사용해보는 POST 방식의 API",
"author":"adam",
"category":"development",
"pages":300,
"price":30000,
"published_date:"1972-06-30",
"description": "게으리지 말자"
}
```

POST

도서 정보 API

❖ 실행

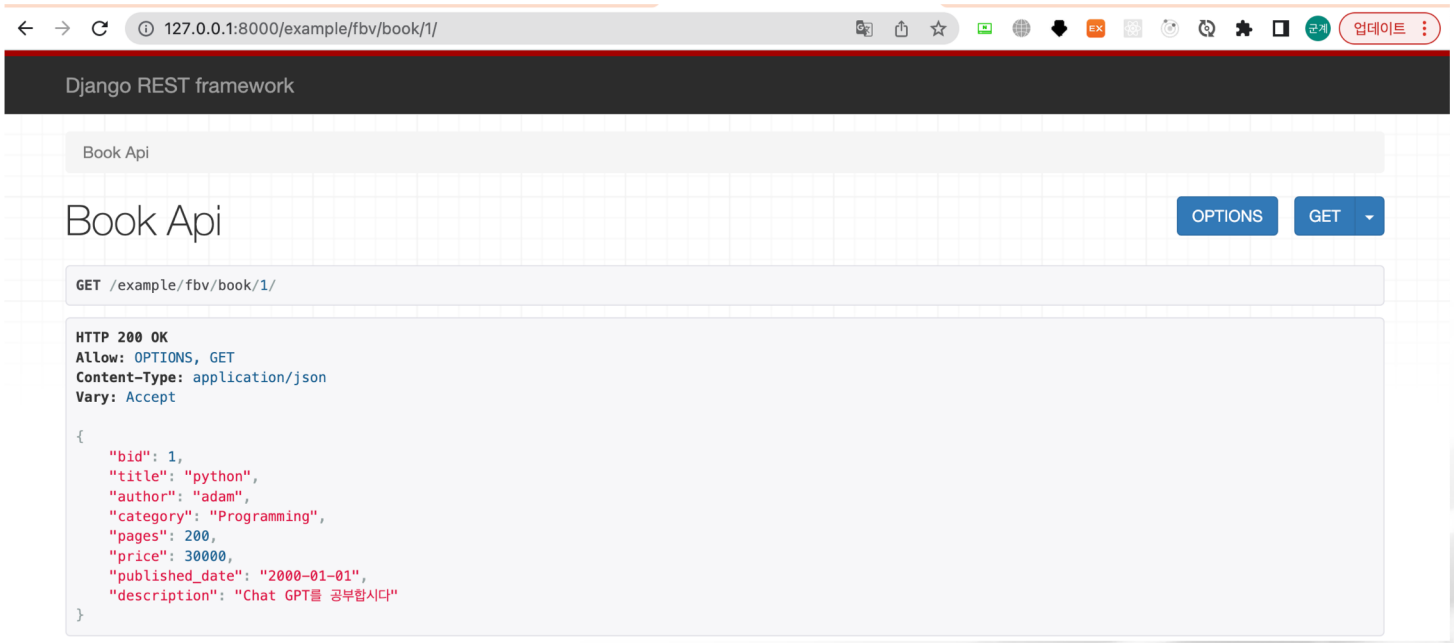
✓ 실행 한 후 브라우저에서 http://127.0.0.1:8000/example/fbv/books/ 를 요청



도서 정보 API

❖ 실행

✓ 실행 한 후 브라우저에서 http://127.0.0.1:8000/example/fbv/book/1/ 를 요청



도서 정보 API

- 사용자의 요청 처리 방식
  - ✓ 뷰(사용자의 요청)의 로직을 처리하는 방식에는 함수 와 클래스를 사용하는 방식이 있음
  - ✓ 함수로 처리하는 뷰를 Function Based View 라고 하고 클래스로 처리하는 뷰를 Class Based View 라고 부름
  - ✓ 함수형 뷰는 상단에 @api_view를 이용해서 API를 위한 View를 생성
  - ✓ 클래스 형 뷰는 method 처리 방식에 따라 메서드를 다르게 해서 처리하고 URL 과 연결할 때 as_view() 라는 함수를 이용해서 처리

도서 정보 API

- 클래스 형 뷰 사용
  - ✓ views.py 파일에 클래스 형 처리 내용 추가

```
from rest_framework import generics, mixins

class BooksAPIMixins(mixins.ListModelMixin,
     mixins.CreateModelMixin,generics.GenericAPIView):
    queryset = Book.objects.all()
    serializer_class = BookSerializer

    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)
    def post(self, request, *args, **kwargs):
        return self.create(request, *args, **kwargs)
```

도서 정보 API

- 클래스 형 뷰 사용
  - ✓ views.py 파일에 클래스 형 처리 내용 추가

```
class BookAPIMixins(mixins.RetrieveModelMixin, mixins.UpdateModelMixin,
    mixins.DestroyModelMixin, generics.GenericAPIView):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
    lookup_field = 'bid'

    def get(self, request, *args, **kwargs):
        return self.retrieve(request, *args, **kwargs)
    def put(self, request, *args, **kwargs):
        return self.update(request, *args, **kwargs)
    def delete(self, request, *args, **kwargs):
        return self.destroy(request, *args, **kwargs)
```

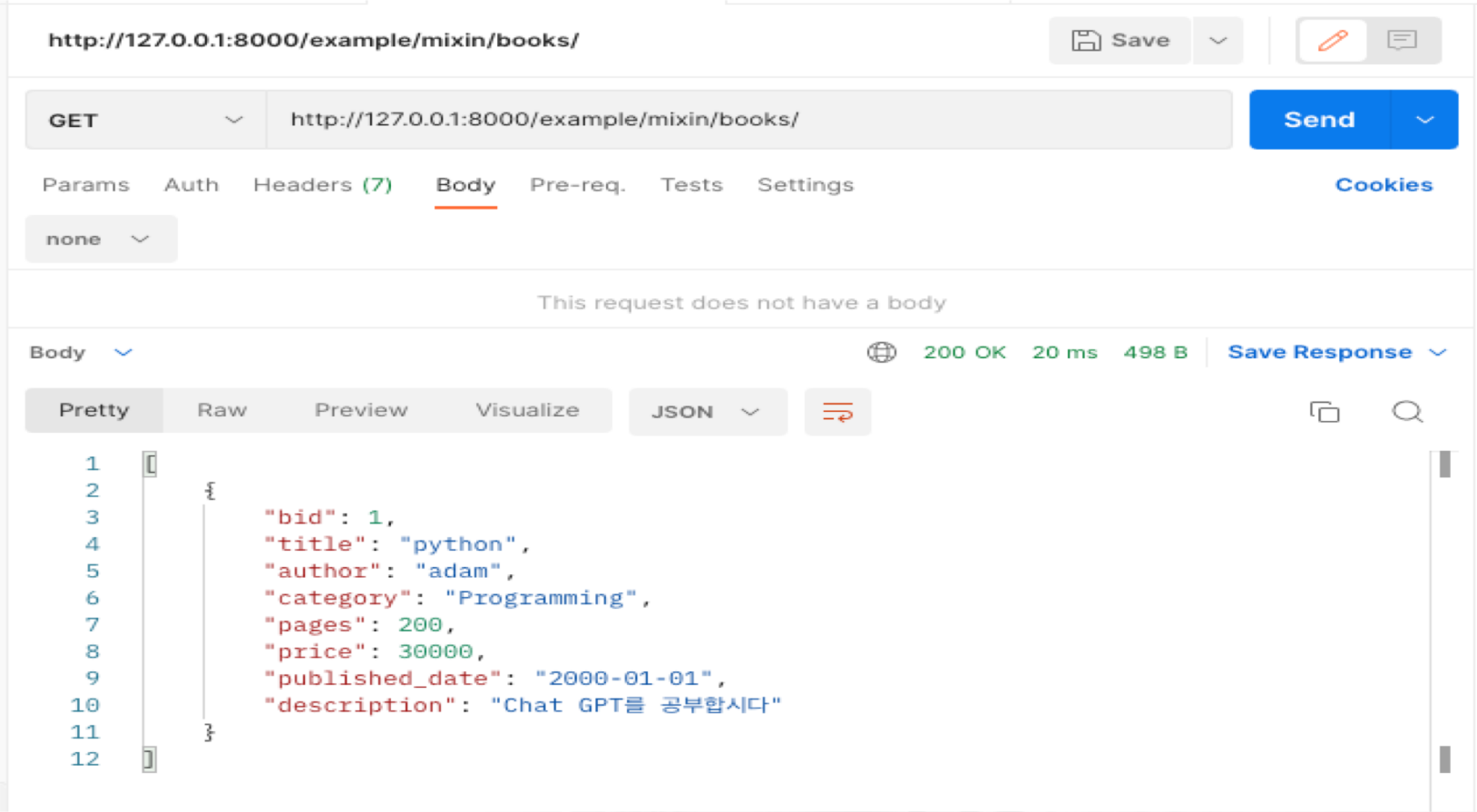
도서 정보 API

- 클래스 형 뷰 사용
  - ✓ apiapp 디렉토리의 urls.py 파일에 클래스 형 처리 내용을 URL 과 연결

```
from django.urls import path, include
from .views import helloAPI, bookAPI, booksAPI,  BooksAPIMixins, BookAPIMixins

urlpatterns = [
    path("hello/", helloAPI),
    path("fbv/books/", booksAPI),
    path("fbv/book/<int:bid>/", bookAPI),

    path("mixin/books/", BooksAPIMixins.as_view()),
    path("mixin/book/<int:bid>/", BookAPIMixins.as_view()),
]
```

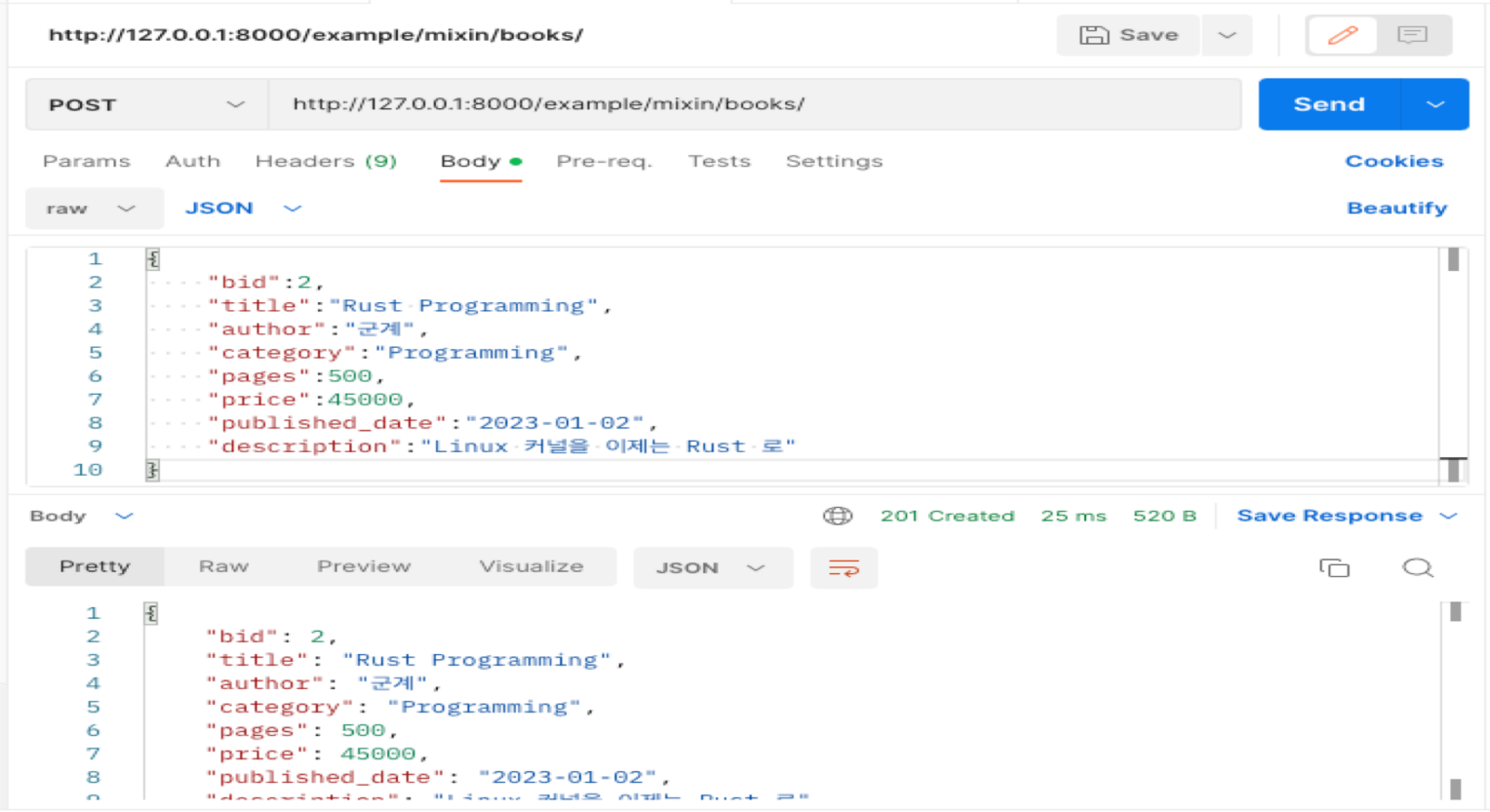
도서 정보 API

- ❖ 클래스 형 뷰 사용
  - ✓ 실행 후 확인



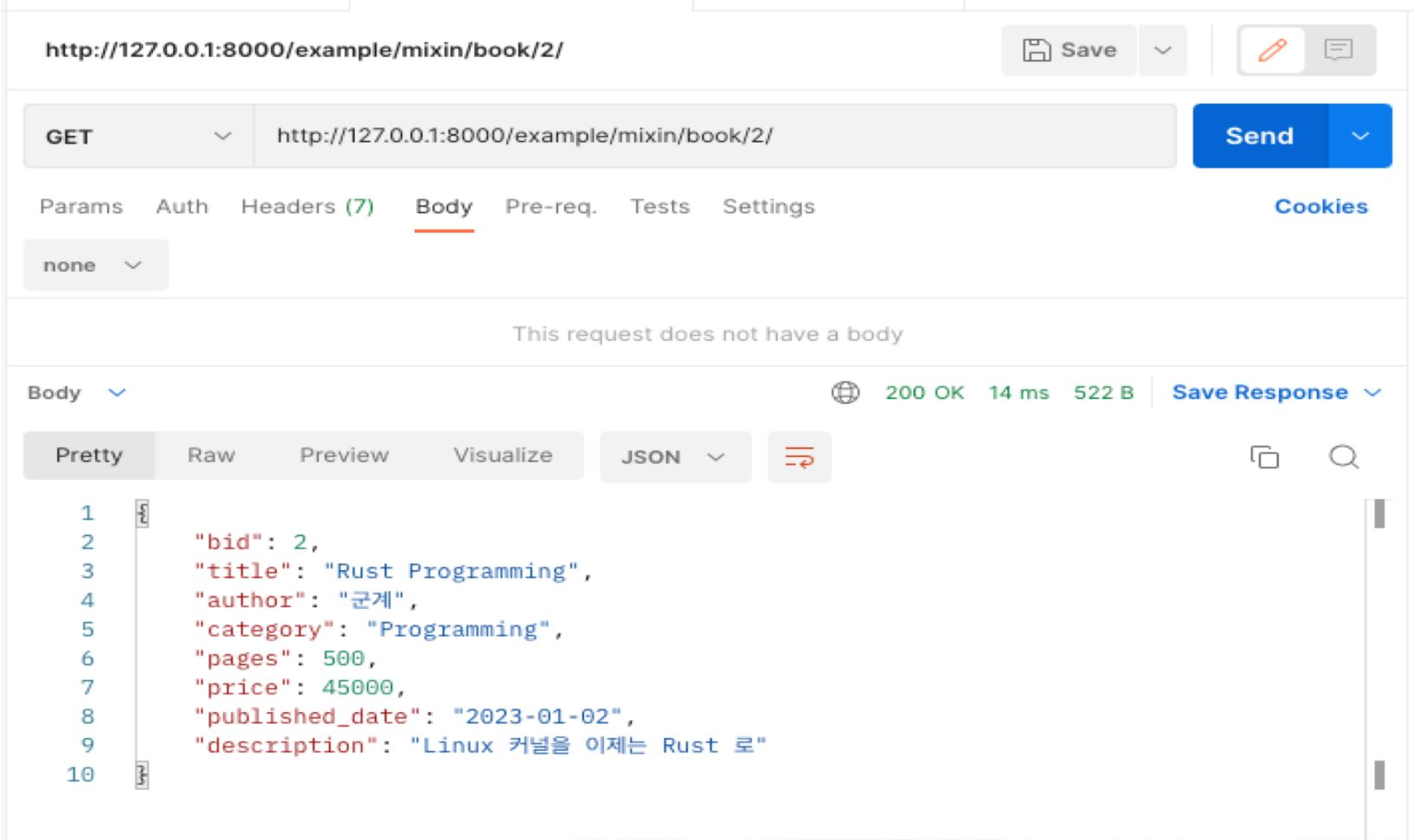
도서 정보 API

- ❖ 클래스 형 뷰 사용
  - ✓ 실행 후 확인



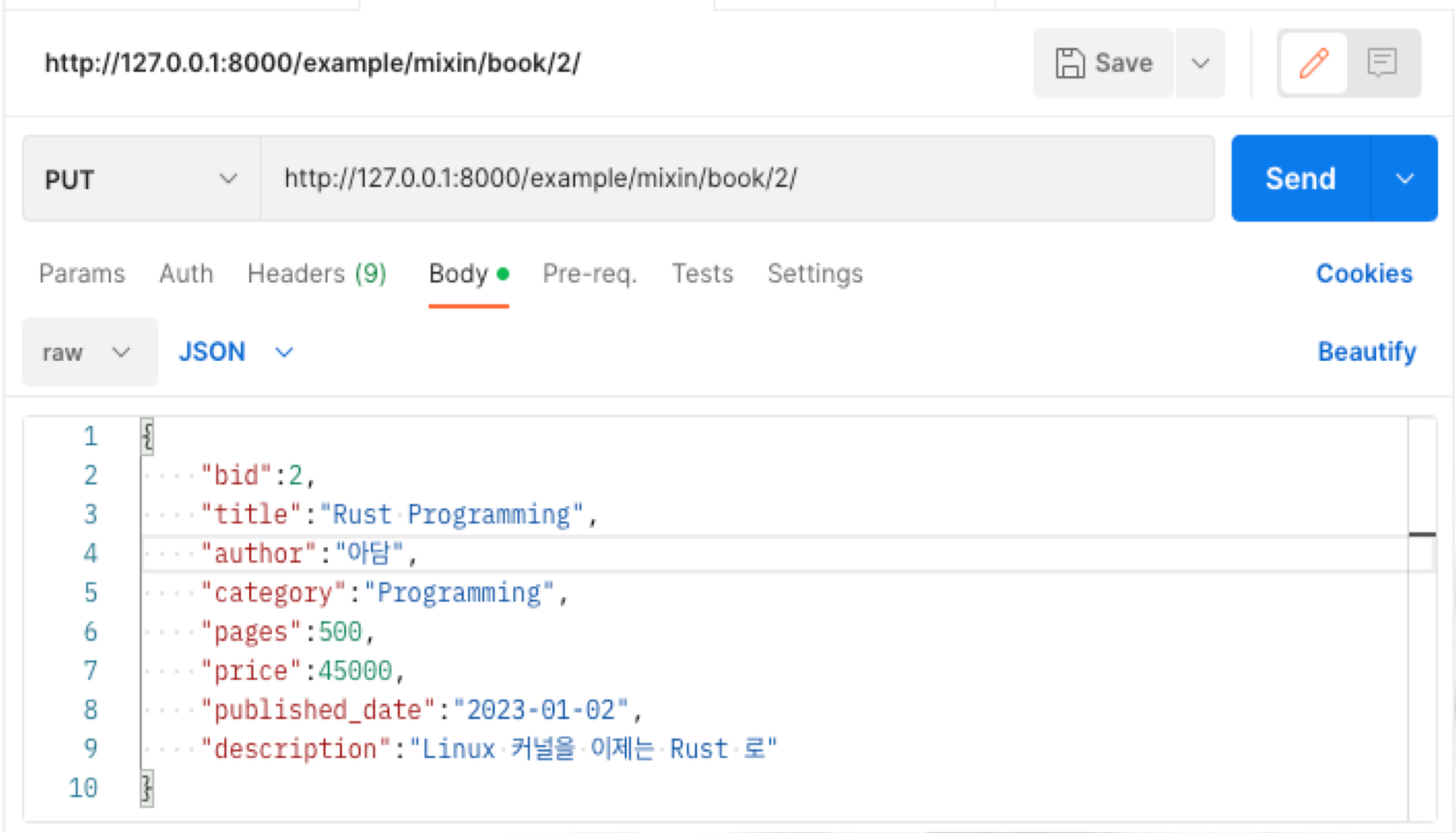
도서 정보 API

❖ 실행 후 확인



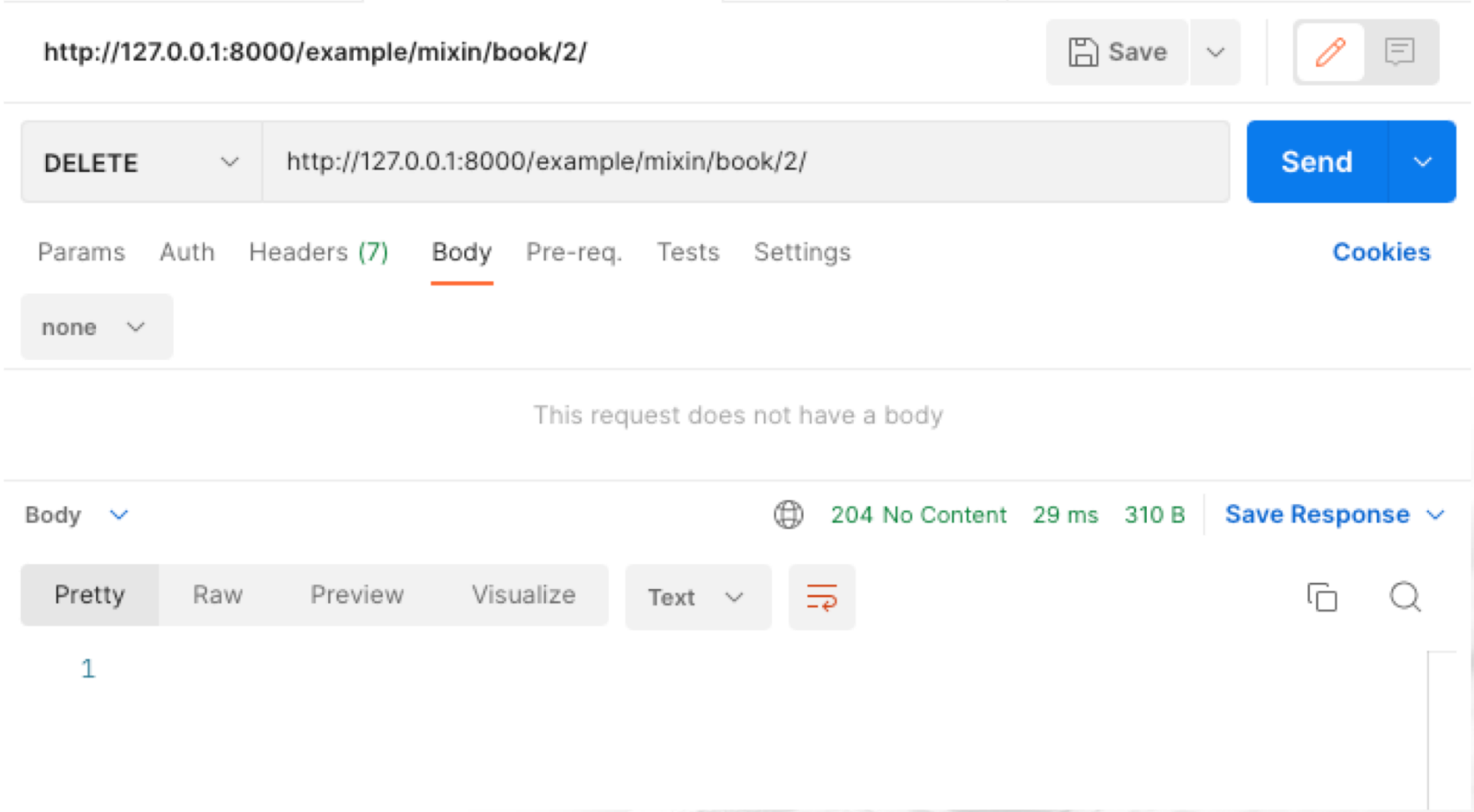
도서 정보 API

❖ 실행 후 확인



도서 정보 API

❖ 실행 후 확인



자바스크립트 비동기 요청

- ajax
  - ✓ Asynchronous Javascript + XML(eXtensible Markup Language)의 약자로 JavaScript와 XML을 이용한 비동기 통신 처리를 구현하는 기술
  - ✓ 웹 페이지가 사용자가 하고 있는 것을 방해하지 않으면서 페이지의 일부를 업데이트할 수 있도록 함
  - ✓ JavaScript에서는 ajax를 XMLHttpRequest 객체로 구현
    - XMLHttpRequest(XHR) 객체는 서버와 상호 작용하기 위하여 사용
    - XMLHttpRequest 는 AJAX 프로그래밍에 사용
    - XMLHttpRequest 는 이름으로만 봐서는 XML 만 받아올 수 있을 것 같아 보이지만 모든 종류의 데이터를 받아오는데 사용할 수 있으며 HTTP 이외의 프로토콜도 지원 (file 과 ftp 포함)
    - 통신을 통해 서버로부터 이벤트나 메시지 데이터를 받아야 한다면 EventSource 를 통한 server-sent events 사용을 고려하고 완전 양방향 통신을 해야 한다면 웹 소켓이 더 나은 선택일 수 있음

자바스크립트 비동기 요청

- ajax
  - ✓ 작업 순서
    - XMLHttpRequest 객체 생성
    - 처리 결과를 받을 이벤트 리스너(콜백 함수) 등록
    - 서버로 보낼 데이터 생성(key=value & key=value)
    - 클라이언트와 서버 간의 연결 요청 준비(open() 이용)
    - 데이터 전송(send() 이용)
    - 응답 처리(status 값을 조사해서 처리)
    - 데이터 처리(csv, xml, json 인지 여부에 따라 처리)

자바스크립트 비동기 요청

- ajax
  - ✓ XMLHttpRequest 속성
    - readyState: 객체의 상태를 나타내는 속성으로 숫자로 저장되는데 0이면 객체 생성 직후이고 1이면 open()을 호출한 상태이고 2이면 send()를 호출한 상태이며 3이면 서버에서 응답이 오기 시작한 상태이고 4이면 서버 응답이 완료된 상태
    - status: 서버로부터의 응답의 상태를 나타내는 숫자로 100번 대 숫자이면 처리 중이고 200 번 대 숫자이면 정상적으로 응답을 받은 것이고 300번 대 숫자이면 현재 redirect 중이며 400번 대 숫자이면 오류가 발생한 것인데 그 중에서도 404이면 페이지를 찾지 못한 것이며 500번 대 숫자이면 대는 서버 오류
    - statusText: 서버로부터의 응답의 상태를 나타내는 문자열로 정상적으로 응답을 받으면 OK가 되고 파일을 찾지 못하면 Not Found
    - responseURL: 응답의 연속된 URL을 리턴하는데 null 인 경우는 빈 문자열을 리턴
    - responseText: 서버로부터 받은 응답 내용 문자열
    - responseXML: 서버로부터 받은 XML 개체
    - timeout: 요청이 자동으로 종료될 때 까지 걸린 시간을 밀리 초 단위로 리턴

자바스크립트 비동기 요청

- ajax
  - ✓ XMLHttpRequest 함수
    - abort(): 요청 취소
    - getAllResponseHeaders(): 응답의 모든 헤더 정보를 문자열로 리턴
    - getResponseHeader(인자): 인자에 해당하는 헤더 정보만 문자열로 리턴
    - open(요청방식, 요청주소, 비동기 전송 여부): ajax 요청 초기화를 수행하는데 openRequest() 권장
    - send(내용): ajax 요청
    - setRequestHeader(인자, 값): 인자에 해당하는 정보를 값으로 설정
    - sendAsBinary: 바이너리 데이터를 전송하는 send 의 다른 방식

자바스크립트 비동기 요청

- ajax
  - ✓ XMLHttpRequest 이벤트
    - abort – onabort 이벤트 핸들러 속성 이용 가능
    - error – onerror 이벤트 핸들러 속성 이용 가능
    - load – 처리 과정이 성공적으로 완료되면 발생하는데 onload 이벤트 핸들러 속성 이용 가능
    - loadend – 요청이 성공이든 실패 든 발생
    - loadstart – 요청이 데이터를 받기 시작하면 발생
    - progress – 요청이 데이터를 받는 동안 발생
  - ✓ XMLHttpRequest 이벤트 핸들러
    - onreadystatechange: readyState 속성의 값이 변경될 때 마다 호출되는 이벤트 핸들러
    - ontimeout: 요청 시간 초과 시마다 호출되는 이벤트 핸들러

자바스크립트 비동기 요청

- ajax
  - ✓ GET 요청
    - ● 프로젝트의 urls.py 파일에 요청 처리 함수 추가

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

urlpatterns = [
    path("admin/", admin.site.urls),
    path("example/", include("apiapp.urls")),
    path("ajax/", views.ajax)
]
```

자바스크립트 비동기 요청

- ajax
  - ✓ GET 요청
    - 애플리케이션 디렉토리의 views.py 파일에 요청 처리 함수 추가

```
def ajax(request):
    return render(request, "ajax.html")
```

자바스크립트 비동기 요청

- ajax
  - ✓ GET 요청
    - 애플리케이션 디렉토리에 templates 디렉토리 추가

자바스크립트 비동기 요청

- ajax
  - ✓ GET 요청
    - templates 디렉토리에 ajax.html 파일을 추가하고 작성 – GET 요청

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ajax</title>
</head>
<body>
  <h1>ajax</h1>
</body>
<script>
  let request = new XMLHttpRequest();
    request.open('GET', '../example/fbv/books/');
  request.send('');
    request.addEventListener('load', function(){
     alert(request.responseText);
    });
</script>
</html>
```

자바스크립트 비동기 요청

- ajax
  - ✓ GET 요청

자바스크립트 비동기 요청

- ajax
  - ✓ JSON Parsing
    - JSON: JavaScript의 객체를 텍스트로 표현하는 방식
    - JSON.parse(JSON 문자열): JSON 문자열을 JavaScript 객체로 변환해서 리턴
    - JSON.stringify(자바스크립트 객체): 자바스크립트 객체를 JSON 문자열로 변환해서 리턴

자바스크립트 비동기 요청

- ajax
  - ✓ JSON Parsing
    - JSON: JavaScript의 객체를 텍스트로 표현하는 방식
    - JSON.parse(JSON 문자열): JSON 문자열을 JavaScript 객체로 변환해서 리턴
    - JSON.stringify(자바스크립트 객체): 자바스크립트 객체를 JSON 문자열로 변환해서 리턴

자바스크립트 비동기 요청

- ajax
  - ✓ JSON Parsing
    - ajax.html 수정

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ajax</title>
</head>
<body>
  <h1>ajax</h1>
</body>
```

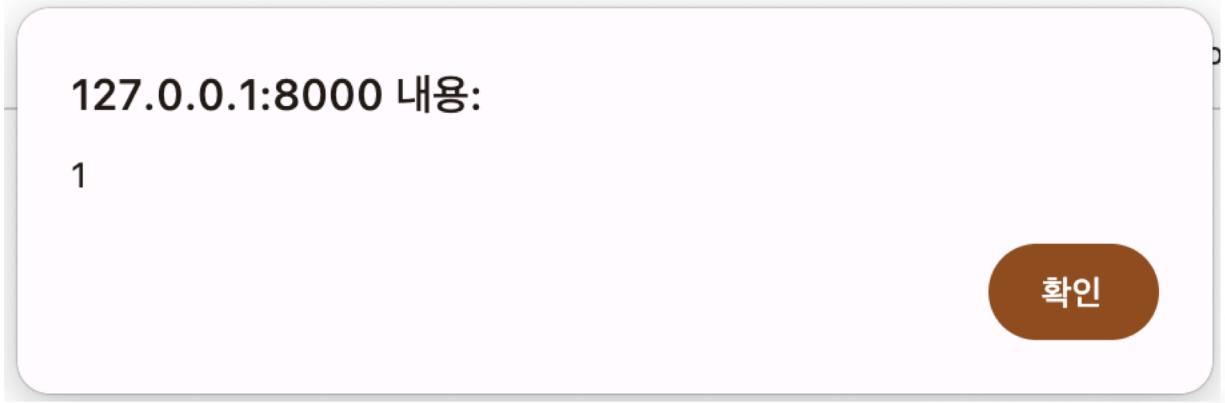
자바스크립트 비동기 요청

- ajax
  - ✓ JSON Parsing
    - ● ajax.html 수정

```
<script>
    let request = new XMLHttpRequest();

    request.open('GET', '../example/fbv/books/');
     request.send('');
    request.addEventListener('load', function(){
     let json = request.responseText;
        //JSON 파싱
        let data = JSON.parse(json)

        for(let idx in data){
            alert(data[idx].bid)
        }
    });

</script>
</html>
```

자바스크립트 비동기 요청

- ajax
  - ✓ JSON Parsing
    - ● 확인



자바스크립트 비동기 요청

- ajax
  - ✓ POST 방식
    - ajax.html 파일의 스크립트 수정

```
<script>
  let request = new XMLHttpRequest();

  //form 데이터를 생성한 후 필요한 파라미터 추가
  //form이 존재하는 경우는 form에 해당하는 DOM을 대입하면 form에 만들어진 데
    이터는 자동으로 추가됨
  let formdata = new FormData();
  formdata.append('bid', 8);
  formdata.append('title', '파이썬');
  formdata.append('author', '귀도반로썸');
  formdata.append('category', '프로그래밍 언어');
  formdata.append('pages', 123);
  formdata.append('price', 345);
  formdata.append('published_date', '2023-07-26');
  formdata.append('description', '설명');
```

자바스크립트 비동기 요청

- ajax
  - ✓ POST 방식
    - ajax.html 파일의 스크립트 수정

```
request.open('POST', "../example/fbv/books/", true);

 request.setRequestHeader("Content-type", "application/x-www-form-
  urlencoded");
let param = "";
for (let pair of formdata.entries()) {
    param += pair[0]+ '=' + pair[1] + "&";
}
request.send(param);

  request.addEventListener('load', function(){
  alert(request.responseText)
 });
</script>
</html>
```

자바스크립트 비동기 요청

- ajax
  - ✓ File Upload
    - ajax.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>post parameter 전송</title>
</head>
<body>
<form id="myform" method="POST" enctype="multipart/form-data">
  <fieldset>
    <legend>파일 업로드</legend>
    이름<input type='text' name='writer' /><br/>
    파일<input type="file" name="file" /><br/>
  </fieldset>
  <input type="button" value="전송" id="sendbtn" />
</form>
```

자바스크립트 비동기 요청

- ajax
  - ✓ File Upload
    - ajax.html

```
<script>
  document.getElementById('sendbtn').addEventListener('click', function(){
    let request = new XMLHttpRequest();
    //form에 enctype이 설정되어 있어서 별도로 설정하지 않아도 됨
    let formdata = new FormData(document.getElementById('myform'));
    for(attr in formdata){
        console.log(attr);
    }
    //form 에 없는 경우 직접 추가
    //formdata.append("image", DOM객체.files[0]);
    request.open('POST', "../postfile", true);
    request.send(formdata);
    request.addEventListener('load', function() {
        alert(request.responseText);
    });
  });
</script>
</body>
</html>
```

자바스크립트 비동기 요청

- ❖ Fetch API
  - ✓ 개요
    - HTTP 파이프라인을 구성하는 요청과 응답 등의 요소를 JavaScript에서 접근하고 조작할 수 있는 인터페이스를 제공
    - 전역 fetch( ) 함수로 네트워크의 리소스를 쉽게 비동기적으로 가져올 수 있음
    - 이전에는 이런 기능을 XMLHttpRequest를 사용해 할 수 있었는데 Fetch는 더 좋은 방법이면서 서비스 워커 등 다른 기술에서도 쉽게 사용할 수 있는 API이며 또한 CORS와 같이 HTTP와 관련된 다른 개념들을 한 곳에 모아서 정의할 수 있는 논리적인 장소도 제공

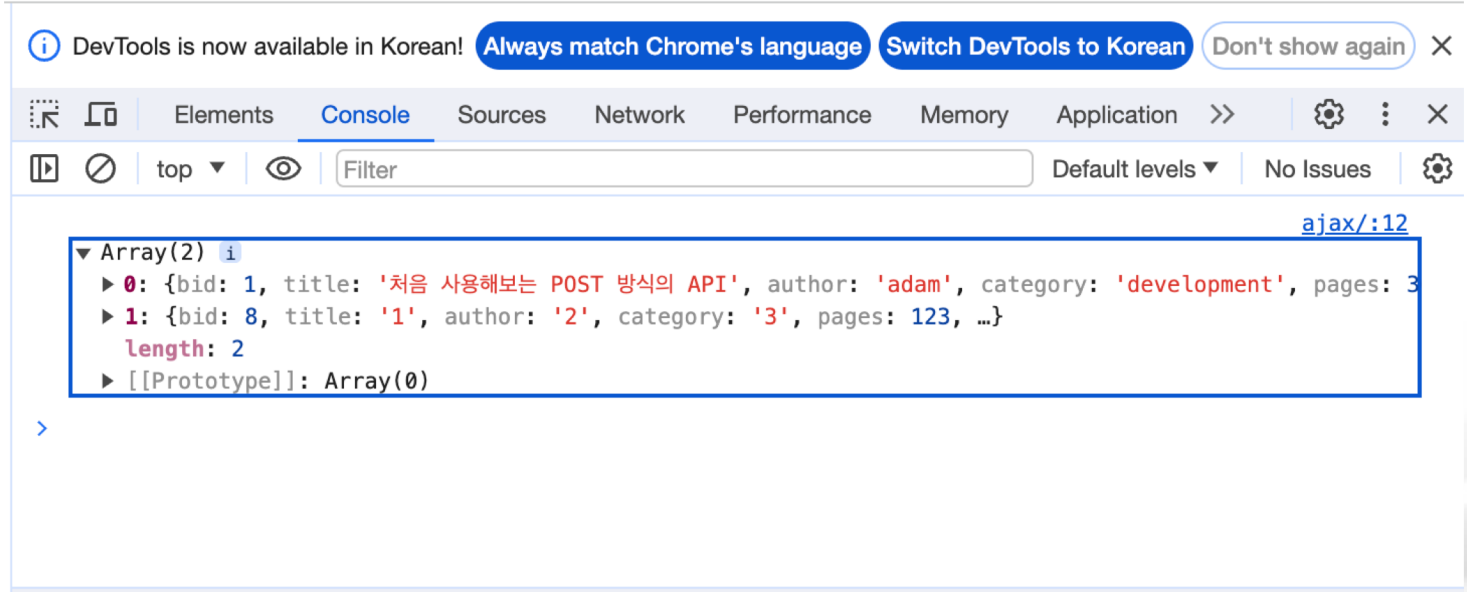
자바스크립트 비동기 요청

- Fetch API
  - ✓ GET 방식
    - ajax.html 파일 수정

```
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>fetch</title>
    <script>
        fetch('../example/fbv/books/')
            .then((response) => response.json())
            .then((data) => console.log(data));
    </script>
</head>

<body>
</body>

</html>
```

자바스크립트 비동기 요청

- ❖ Fetch API
  - ✓ GET 방식
    - ● 확인



자바스크립트 비동기 요청

- ❖ Fetch API
  - ✓ 옵션 설정
    - 단순한 형태의 fetch()는 가져오고자 하는 리소스의 경로를 나타내는 하나의 인수만 받으며 응답은 Response 객체로 표현되고 직접 JSON 응답 본문을 받을 수는 없음
    - Response 객체 역시 JSON 응답 본문을 그대로 포함하지는 않는데 Response는 HTTP 응답 전체를 나타내는 객체로 JSON 본문 콘텐츠를 추출하기 위해서는 json() 메서드를 호출해야 하며 json()은 응답 본문 텍스트를 JSON으로 파싱한 결과로 이행하는 또 다른 Promise를 반환
    - 옵션을 설정할 수 있음 - fetch(resource, options)
    - 옵션 - https://developer.mozilla.org/en-US/docs/Web/API/fetch

자바스크립트 비동기 요청

- Fetch API
  - ✓ 옵션 설정
    - ajax.html 파일 수정

```
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>fetch</title>
```

자바스크립트 비동기 요청

- ❖ Fetch API
  - ✓ 옵션 설정
    - ● ajax.html 파일 수정

```
<script>
    const data = {
        bid:9,
        title:'Java',
          author:'제임스 고슬링',
          category:'프로그래밍 언어',
          pages:300,
        price:35000,
        published_date:'2023-07-26',
        description:'객체 지향 언어'
    };
    fetch('../example/fbv/books/', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
    },
        body: JSON.stringify(data),
    })
```

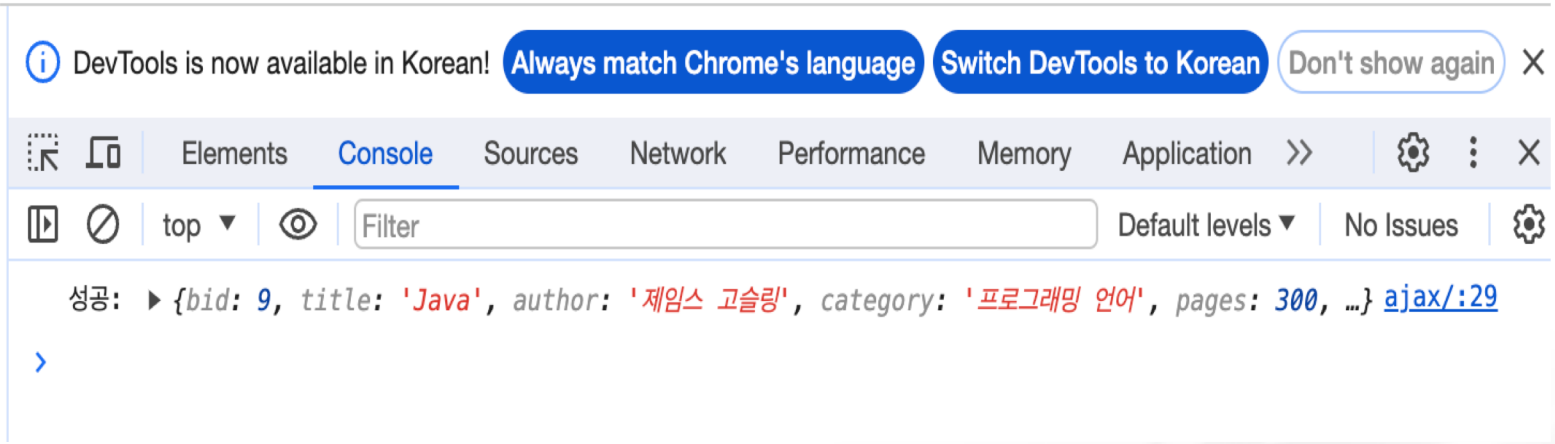
자바스크립트 비동기 요청

- ❖ Fetch API
  - ✓ 옵션 설정
    - ● ajax.html 파일 수정

```
            .then((response) => response.json())
            .then((data) => {
                console.log('성공:', data);
            })
            .catch((error) => {
                console.error('실패:', error);
            });
        </script>
    </head>

    <body>
    </body>

    </html>
```

자바스크립트 비동기 요청

- ❖ Fetch API
  - ✓ 옵션 설정
    - ● ajax.html 파일 수정



자바스크립트 비동기 요청

- Fetch API
  - ✓ 하나의 파일 업로드

```
const formData = new FormData();
const fileField = document.querySelector('input[type="file"]');

formData.append('username', 'abc123');
formData.append('avatar', fileField.files[0]);

fetch('https://example.com/profile/avatar', {
  method: 'PUT',
  body: formData,
})
.then((response) => response.json())
.then((result) => {
  console.log('성공:', result);
})
.catch((error) => {
  console.error('실패:', error);
});
```

자바스크립트 비동기 요청

- Fetch API
  - ✓ 여러 개의 파일 업로드

```
const formData = new FormData();
const photos = document.querySelector('input[type="file"][multiple]');

formData.append('title', '제주도 수학여행');
for (let i = 0; i < photos.files.length; i++) {
  formData.append(`photos_${i}`, photos.files[i]);
}

fetch('https://example.com/posts', {
  method: 'POST',
  body: formData,
})
.then((response) => response.json())
.then((result) => {
  console.log('성공:', result);
})
.catch((error) => {
  console.error('실패:', error);
});
```

자바스크립트 비동기 요청

- ❖ Fetch API
  - ✓ 응답 객체
    - ● 속성
      - ◆ Response.status - 상태 코드 값을 담은 정수 값으로 기본 값은 200
      - ◆ Response.statusText - 상태 코드 메시지를 담은 문자열 값으로 기본 값은 빈 문자열
      - ◆ Response.ok - 상태 코드가 200 이상 299 이하인지 간단하게 확인할 수 있는 불리언 값

자바스크립트 비동기 요청

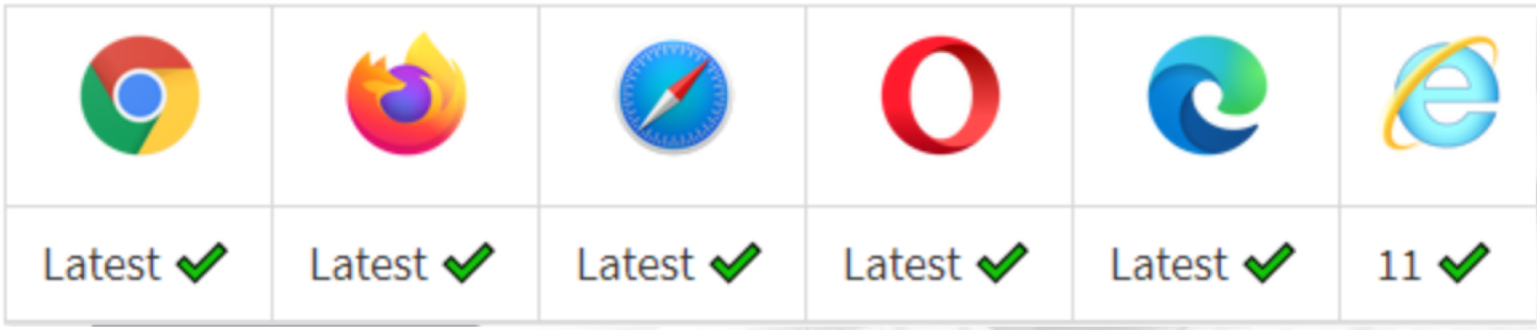
- Fetch API
  - ✓ 응답 객체
    - ● 본문 데이터
      - ◆ ArrayBuffer
      - ◆ ArrayBufferView(Uint8Array 등등)
      - ◆ Blob/File
      - ◆ 문자열
      - ◆ URLSearchParams
      - ◆ FormData

자바스크립트 비동기 요청

- Fetch API
  - ✓ 응답 객체
    - Request와 Response (en-US) 인터페이스는 본문을 추출할 수 있는 다음의 메서드들을 공유
      - Request.arrayBuffer() / Response.arrayBuffer()
      - Request.blob() / Response.blob()
      - Request.formData() / Response.formData()
      - Request.json() / Response.json()
      - Request.text() / Response.text()

자바스크립트 비동기 요청

- ❖ axios
  - ✓ axios
    - node.js 와 브라우저를 위한 Promise 기반 HTTP 클라이언트
    - 동일한 코드 베이스로 브라우저와 node.js에서 실행할 수 있음
    - 서버 사이드에서는 네이티브 node.js의 http 모듈을 사용하고 클라이언트(브라우저)에서는 XMLHttpRequests를 사용
    - https://axios-http.com/kr/docs/intro
    - 설치
      - ◆ node.js: yarn add axios
      - ◆ 클라이언트 사이트: <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>



자바스크립트 비동기 요청

- axios
  - ✓ 기본 사용

```
// ID로 사용자 요청
axios
  .get('/user?ID=12345')
  // 응답(성공)
  .then(function(response) {
    console.log(response)
  })
  // 응답(실패)
  .catch(function(error) {
    console.log(error)
  })
  // 응답(항상 실행)
  .then(function() {
    // ...
  })
```

자바스크립트 비동기 요청

- axios
  - ✓ async await를 이용한 사용

```
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345')
    console.log(response)
  } catch (error) {
    console.error(error)
  }
}
```

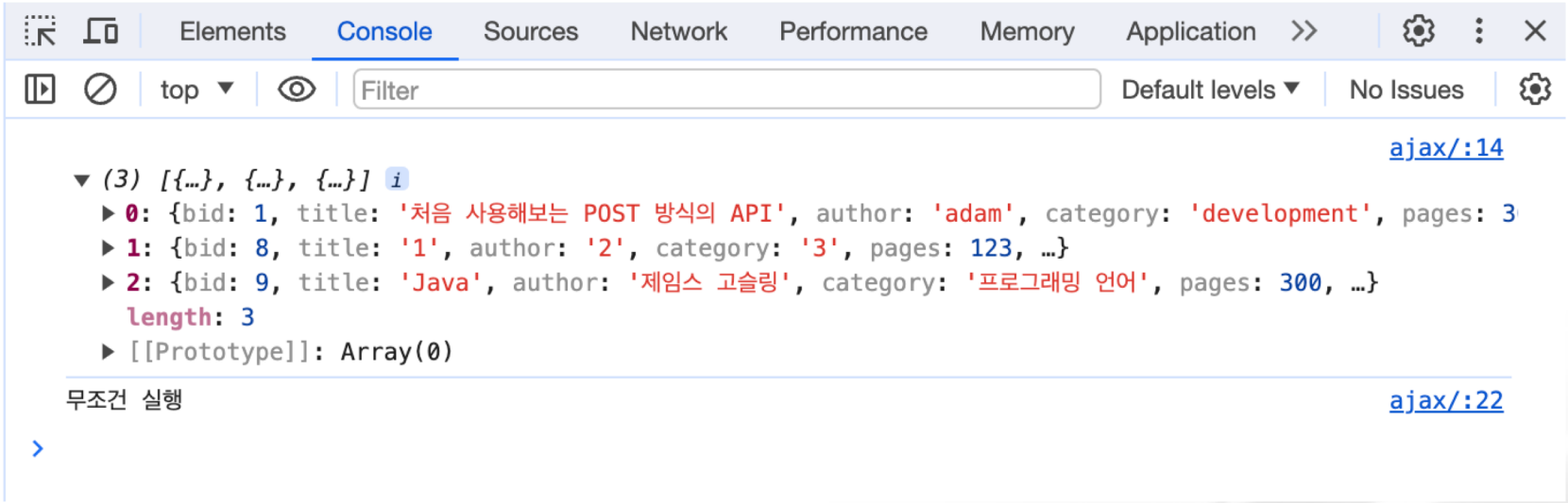
자바스크립트 비동기 요청

- axios
  - ✓ ajax.html 파일의 script 수정

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
  <script>
      axios.get('../example/fbv/books/')
          .then(function (response) {
              // 성공한 경우 실행
              console.log(response.data);
          })
          .catch(function (error) {
              // 에러인 경우 실행
              console.log(error);
          })
          .then(function () {
              // 항상 실행
              console.log("무조건 실행")
          });

  </script>
```

자바스크립트 비동기 요청

- axios
  - ✓ 확인



자바스크립트 비동기 요청

- SOP(Same Origin Policy - 동일 출처 정책)
  - ✓ 어떤 출처(origin)에서 불러온 문서나 스크립트가 다른 출처에서 가져온 리소스와 상호 작용하는 것을 제한하는 브라우저의 보안 방식
  - ✓ 다른 출처와 같은 출처를 구분하는 기준은 URI의 프로토콜 그리고 호스트 및 포트가 동일한지 여부
  - ✓ 모든 방식의 요청에 적용 되는 것은 아님
  - ✓ 적용되지 않는 경우
    - <img> 태그로 다른 도메인의 이미지 파일을 가져오는 경우
    - <link> 태그로 다른 도메인의 CSS를 가져오는 경우
    - <script> 태그로 다른 도메인의 javascript를 가져오는 경우
    - <video> <audio> <object> <embed> <applet> 태그
    - SOP는 script에서 XMLHttpRequest 나 Fetch API 를 사용해 다른 출처에 리소스를 요청할 때 적용

자바스크립트 비동기 요청

- 교차 출처 리소스 공유 (CORS)
  - ✓ 교차 출처 리소스 공유(Cross-Origin Resource Sharing, CORS)는 추가 HTTP 헤더를 사용하여 한 출처에서 실행 중인 웹 애플리케이션이 다른 출처의 자원에 접근할 수 있는 권한을 부여하도록 브라우저에 알려주는 체제
  - ✓ 웹 애플리케이션은 리소스가 자신의 출처(도메인, 프로토콜, 포트)와 다를 때 교차 출처 HTTP 요청을 실행함
  - ✓ 보안 상의 이유로 브라우저는 스크립트에서 시작한 교차 출처 HTTP 요청을 제한함
  - ✓ XMLHttpRequest 와 Fetch API는 동일 출처 정책을 따르기 때문에 이 API 를 사용하는 웹 애플리케이션은 자신의 출처와 동일한 리소스만 불러올 수 있으며 다른 출처의 리소스를 불러오려면 그 출처에서 올바른 CORS 헤더를 포함한 응답을 반환해야 함
  - ✓ 서버의 설정을 변경할 수 없는 경우에는 Proxy Server를 이용해서 서버의 데이터를 ajax 나 Fetch API를 이용해서 사용할 수 있음

자바스크립트 비동기 요청

- 교차 출처 리소스 공유 (CORS)
  - ✓ Django 에서 CORS 설정
    - 패키지 설치: pip install django-cors-headers
    - settings.py 파일에 설정 추가

```
INSTALLED_APPS =[
...

'corsheaders', # CORS 관련 추가

]
...
MIDDLEWARE = [
   'corsheaders.middleware.CorsMiddleware',      # CORS 관련 추가
...
]
...
```

자바스크립트 비동기 요청

- 교차 출처 리소스 공유 (CORS)
  - ✓ Django 에서 CORS 설정
    - settings.py 파일에 설정 추가

```
# CORS 관련 추가
CORS_ORIGIN_WHITELIST = ['http://127.0.0.1:3000'
                         ,'http://localhost:3000']
또는
CORS_ORIGIN_ALLOW_ALL=True # <- 모든 호스트 허용

CORS_ALLOW_CREDENTIALS = True
```

자바스크립트 비동기 요청

- 교차 출처 리소스 공유 (CORS)
  - ✓ Django 에서 CORS 설정
    - settings.py 파일에 설정 추가

```
CORS_ALLOW_METHODS = (  #<-실제 요청에 허용되는 HTTP 동사 리스트
    'DELETE',
    'GET',
    'OPTIONS',
    'PATCH',
    'POST',
    'PUT',
)
```

자바스크립트 비동기 요청

- 교차 출처 리소스 공유 (CORS)
  - ✓ Django 에서 CORS 설정
    - settings.py 파일에 설정 추가

```
CORS_ALLOW_HEADERS = ( <-실제 요청을 할 때 사용될 수 있는 non-standard HTTP
    헤더 목록// 현재 기본값
    'accept',
    'accept-encoding',
    'authorization',
    'content-type',
    'dnt',
    'origin',
    'user-agent',
    'x-csrftoken',
    'x-requested-with',
)
```