

# 08 PJT

# Django 에서 알고리즘 구현 및 성능측정

## 챕터의 포인트

- 목표
- 준비사항 및 제공사항
- 성능 테스트 개념 (with. Locust)
- 요구사항
- 제출

# 목표

## | 프로젝트 목표

- 데이터 분석용 파이썬 패키지(Numpy, Pandas)에 대한 이해
- 테스트 및 성능 테스트에 대한 이해
- Django 를 활용한 REST API 구현
- 부하 테스트에 대한 이해
- **Pandas 를 활용한 알고리즘 구현 및 알고리즘 성능 측정**

## 준비사항 및 참고사항

## | 개발도구

- Visual Studio Code
- Google Chrome
- Django 3.2 +
- Python Library - Numpy, Pandas, Locust

## | 데이터 형식 - CSV

- 몇 가지 필드를 쉼표(,)로 구분한 텍스트 데이터 및 텍스트 파일
- 일반적으로 표 형식의 데이터가 일반 텍스트 형태로 저장된다.
- 엑셀과 비교하였을 때, 일반 텍스트로 저장되므로 저장 및 전송하고 처리할 수 있는 프로그램이 다양하다는 큰 장점이 있다.



## | 데이터 형식 - CSV

- 표 형식의 데이터

이름	생년	월	일	성별	직업	사는 곳
홍길동	1992	7	17	남	강사	서울
희동이	1997	4	3	여	학생	대구
금땡구	1988	2	15	남	개발자	광주

- csv 로 표현하면 아래와 같다.

이름,생년,월,일,성별,직업,사는 곳  
홍길동,1992,7,17,남,강사,서울  
희동이,1997,4,3,여,학생,대구  
금땡구,1988,2,15,남,개발자,광주

## JSON vs CSV

- 둘 모두 서로 다른 시스템이나 디바이스 간에 데이터를 쉽게 교환할 수 있도록 도움

특징	JSON	CSV
장점	- 모양과 규칙 자체가 단순해서 타 언어에서도 구현하기가 쉽다.	- 용량이 가장 작다. - csv는 용량이 작기 때문에 변하지 않는 많은 양의 데이터를 제공할 때 주로 이용이 가능하다.
단점	- 콤마가 누락되거나 중괄호가 잘못 닫히는 등 문법 오류에 취약하다.	- 데이터의 의미를 표시할 수 없기 때문에 데이터가 많아지면 어떤 데이터가 항목을 나타내는 지 직관적인 이해가 어렵다.
주요 사용처	- 서버 통신 REST API를 사용할 때 가장 많이 사용	- 간단한 테이블 작성 또는 읽는 속도가 중요한 부분에서 사용

# Numpy & Pandas

## | Numpy

- 다차원 배열을 쉽게 처리하고 효율적으로 사용할 수 있도록 지원하는 파이썬 패키지
- 장점
  - Numpy 행렬 연산은 데이터가 많을수록 Python 반복문에 비해 훨씬 빠르다 !
  - 다차원 행렬 자료 구조를 제공하여 개발하기 편하다.
- 특징
  - CPython(공식 사이트의 Python) 에서만 사용 가능
  - 행렬 인덱싱(Array Indexing) 기능 제공

## | Pandas

- Numpy 의 한계
  - 유연성(데이터에 레이블을 붙이거나, 누락된 데이터로 작업)이 부족함
  - 그룹화, 피벗 등 구조화가 부족함
- Pandas 는 마치 프로그래밍 버전의 엑셀을 다루듯 고성능의 데이터 구조를 만들 수 있음
- Numpy 기반으로 만들어진 패키지로, Series(1차원 배열) 과 DataFrame(2차원 배열) 이라는 효율적인 자료구조 제공

## | 실습 파일

- 파일
  - `Numpy_Basic.ipynb` - Numpy 기초 코드
  - `Pandas_Baisc.ipynb` - Pandas 기초 코드
  - `Pandas_Advanced.ipynb` - Pandas 응용 코드
- Pandas 학습 - 테스트 데이터
  - `data/test_data.CSV` - 결측치 미포함 테스트 데이터 (50개)
  - `data/test_data_has_null.CSV` - 결측치 포함 테스트 데이터 (50개)

# 테스트

## | 어떤 문제점이 발생할까 ?



개발자

코드 작성



코드

바로 적용



서비스



# | 어떤 문제점이 발생할까 ?



개발자

코드 작성



코드

안되는 기능은 없나 ?

버그는 없나 ...?

바로 적용

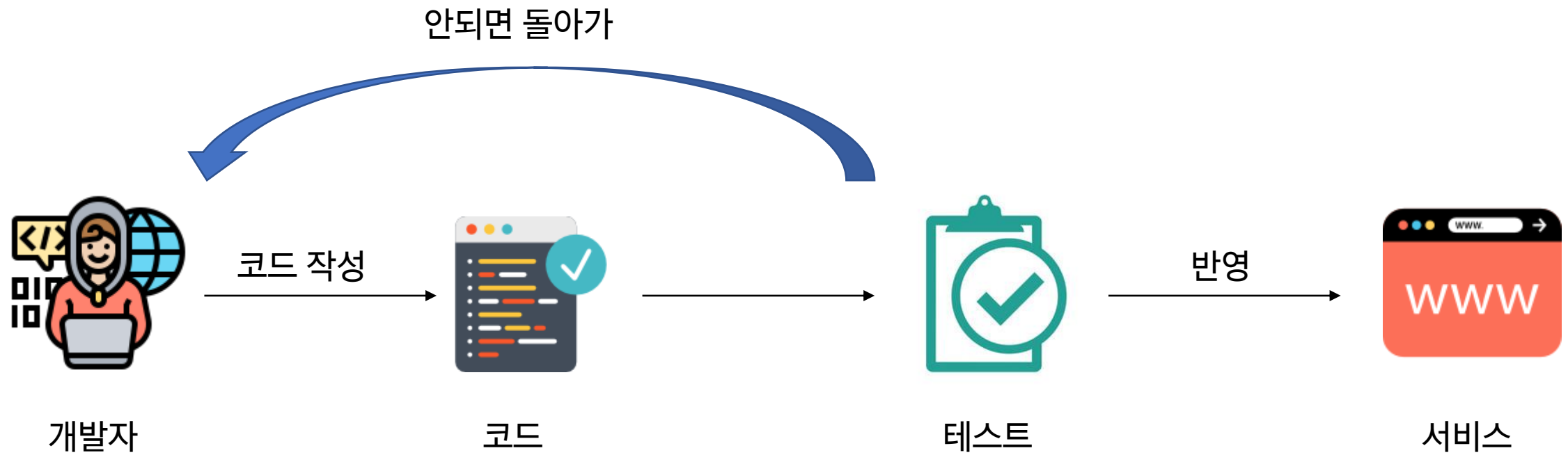


서비스

기능은 다 만들었나 ?

내 컴퓨터에서는 잘 작동하던데...

## | 테스트란 ?

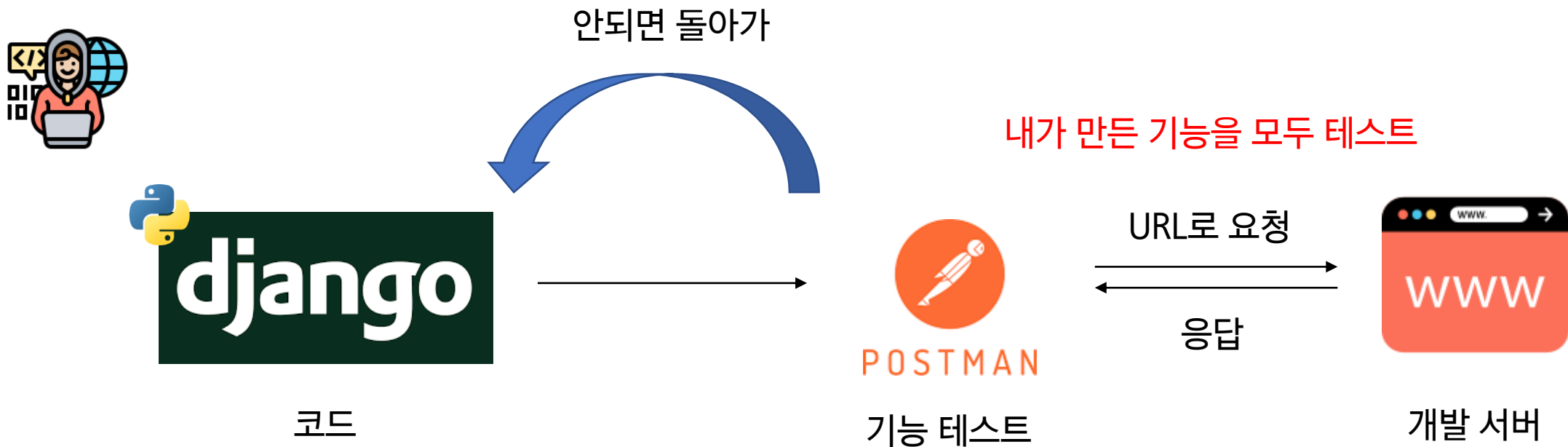


## | 테스트란 ?

- 테스트의 종류는 너무 많다..!! ( 100가지가 넘는다! )
- 상황에 맞게 필요한 테스트를 진행해야 한다.
  - 외우지 말고 검색을 통해 필요한 것만 찾아서 사용하자!

## | 예시 - 파이썬 반 API 테스트

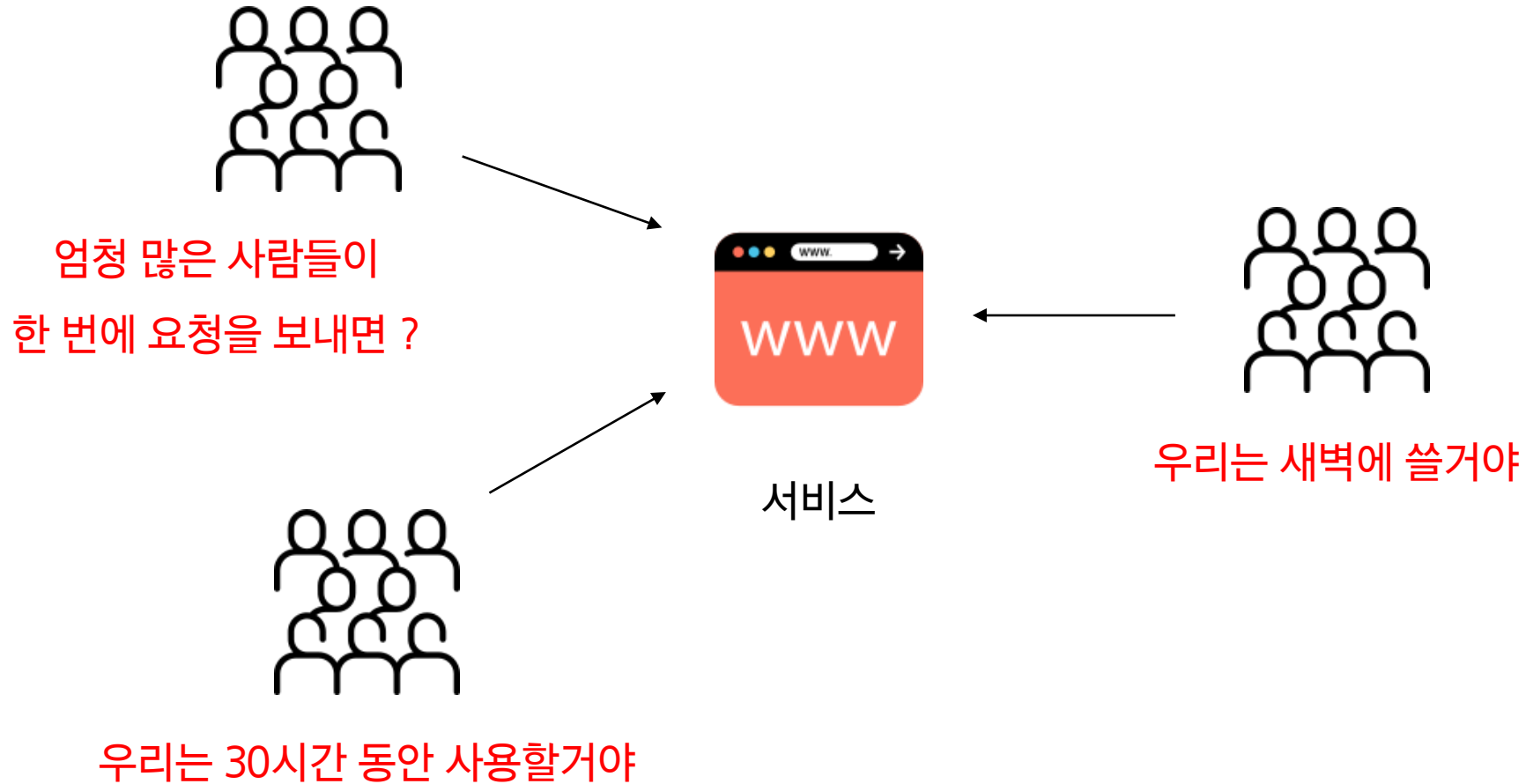
- 우리는 API 가 정상적인 응답을 주는 지 이미 테스트를 진행해 보았다.



## | 성능 테스트

- 핵심 적인 테스트 중 하나
- 특정 상황에서 시스템이 어느 정도 수준을 보이는가 혹은 어떻게 대처를 하는가를 테스트하는 과정
- 목적
  - 여러 테스트를 통해 **성능 저하가 발생하는 요인을 발견하고 제거**하기 위함
  - 시장에 출시되기 전에 발생할 수 있는 위험과 개선사항을 파악하기 위함
  - 안정적이고 신뢰할 수 있는 제품을 빠르게 만들기 위함

## | 성능 테스트 예시

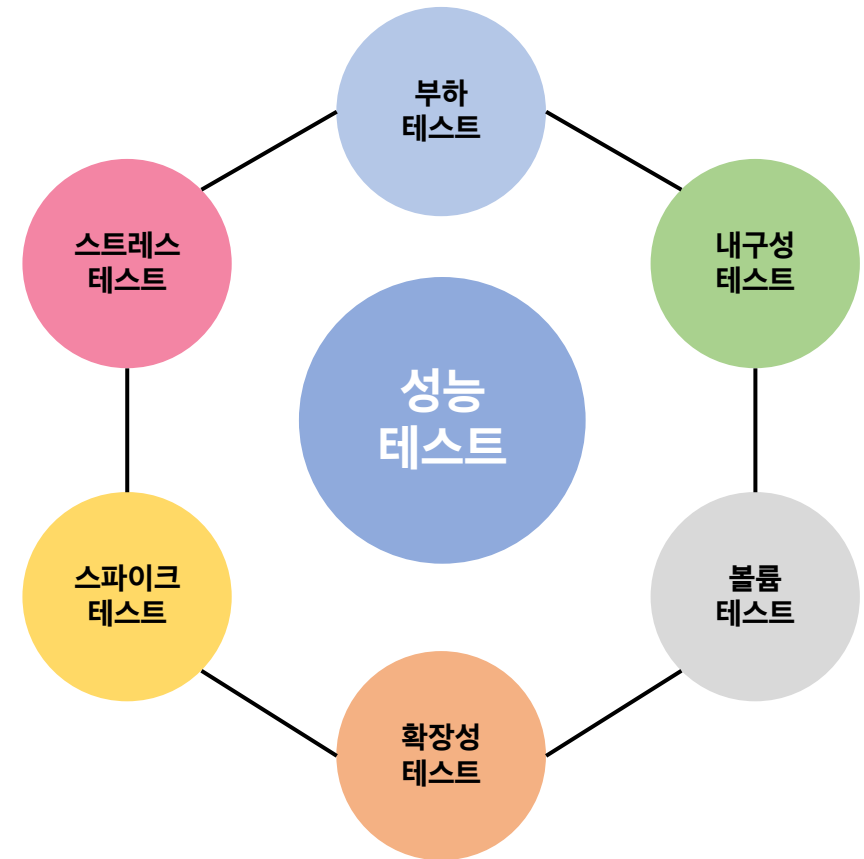


## | 성능 테스트

- 성능 테스트의 종류도 많다.
- 핵심인 **부하 테스트**와 **스트레스 테스트**만 알고 있자.
- 종류: 부하 테스트, 스트레스 테스트, 스파이크 테스트, 확장성 테스트, 볼륨 테스트 등등

## | 성능 테스트

- 부하 테스트(Load Testing)
  - 시스템에 임계점의 부하가 계속될 때 문제가 없는가 ?
  - 시스템의 신뢰도와 성능을 측정하기 위함
- 스트레스 테스트(Stress Testing)
  - 시스템에 과부하가 오면 어떻게 동작할까 ?
  - 장애 조치와 복구 절차가 효과적이고 효율적인가

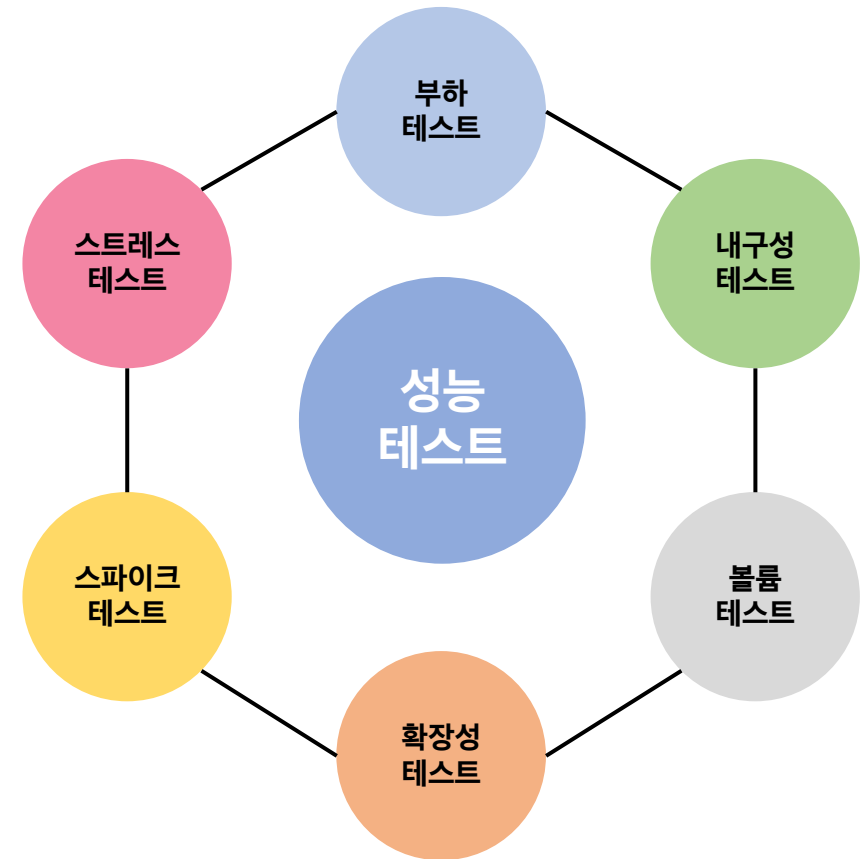


임계점: 응답시간이 급격히 느려지는 구간



## [참고] 성능 테스트

- 스파이크 테스트(Spike Testing)
  - 과부하를 짧은 시간 반복적으로 발생시켜 나타나는 문제점 확인
- 내구성 테스트(Endurance Testing)
  - 시스템을 장시간 사용 시 나타나는 문제점을 확인
- 확장성 테스트(Scalability Testing)
  - 용량 등 시스템 확장 시 문제점은 없는가 ?
  - 서버 수, DB 등 확장, 축소 시 문제점 확인
- 볼륨 테스트(Volume Testing)
  - 많은 양의 데이터 사용 시 문제가 없는가 ?
  - 원하는 데이터 양을 모두 다룰 수 있는 지 검사



# 부하 테스트 vs 스트레스 테스트

	부하 테스트(Load Testing)	스트레스 테스트(Stress Testing)
도메인	성능 테스트의 하위 집합	성능 테스트의 하위 집합
범위	볼륨 테스트 내구성 테스트	스파이크 테스트 내구성 테스트 중단점 테스트
테스트 목적	전체 시스템의 성능 확인	중단점에서의 동작, 복구 가능성 확인
테스트 방법	임계점까지의 가상 유저 수를 유지하며 모니터링	중단점 이상까지 가상 유저를 점진적으로 증가
테스트 대상	전체 시스템	식별된 트랜잭션에만 집중하여 테스트
테스트 완료 시기	예상 부하가 모두 적용된 경우	시스템 동작이 중단되었을 경우
결과	부하 분산 문제 최대 성능 시간 당 서버 처리량 및 응답 시간 최대 동시 사용자 수 등	안정성 복구 가능성

# API 성능 테스트 (Locust)

## | Locust



- **오픈 소스 부하 테스트 도구**
- 번역하면 메뚜기. 테스트 중 메뚜기 떼가 웹 사이트를 공격한다는 의미로 착안된 이름
- 내가 만든 서버에 수많은 사용자들이 동시에 들어올 때 어떤 일이 벌어 지는 지를 확인하는  
부하 테스트를 할 수 있는 도구
- Locust 를 선택한 이유
  - 파이썬 언어로 테스트 시나리오를 간편하게 작성할 수 있다.
  - 결과를 웹에서 확인할 수 있는 UI를 지원한다.

## Locust 사용법 - (1/10)

### 1. 테스트 스크립트 작성하기 ([공식문서](#) 참조)

#### 공식 문서 코드

```
1  import time
2  from locust import HttpUser, task, between
3
4  class QuickstartUser(HttpUser):
5      wait_time = between(1, 5)
6
7      @task
8      def hello_world(self):
9          self.client.get("/hello")
10         self.client.get("/world")
11
12     @task(3)
13     def view_items(self):
14         for item_id in range(10):
15             self.client.get(f"/item?id={item_id}", name="/item")
16             time.sleep(1)
17
18     def on_start(self):
19         self.client.post("/login", json={"username": "foo", "password": "bar"})
```

- `HttpUser`: HTTP 요청을 만드는 가상 유저
- `wait_time`: 작업 간 대기 시간
- `on_start()`: 가상 유저 생성 시 실행
- `@task`: 유저가 실행할 작업
- `@task(N)`: 가중치 ( 실행 확률 )
  - N만큼 높은 확률로 작업을 수행
- `self.client.get`: HTTP GET 요청 전송

## | Locust 사용법 - (2/10)

### 2. Django 서버 실행 - 제공된 Django API 서버를 실행합니다

```
$ cd performance_test  
$ python -m venv venv  
$ source venv/Scripts/activate  
  
(venv) $ pip install -r requirements.txt  
  
(venv) $ python manage.py makemigrations  
  
(venv) $ python manage.py migrate  
  
(venv) $ python manage.py runserver
```

## | Locust 사용법 - (2/10)

### 3. vscode 터미널 추가 & Locust 설치 및 실행

```
(venv) $ pip install locust
```

```
(venv) $ locust -f ./locust_test.py
```

## | Locust 사용법 - (3/10)

4. Locust 정상 실행 시 터미널에 아래와 같이 접속할 수 있는 URL 이 출력된다.

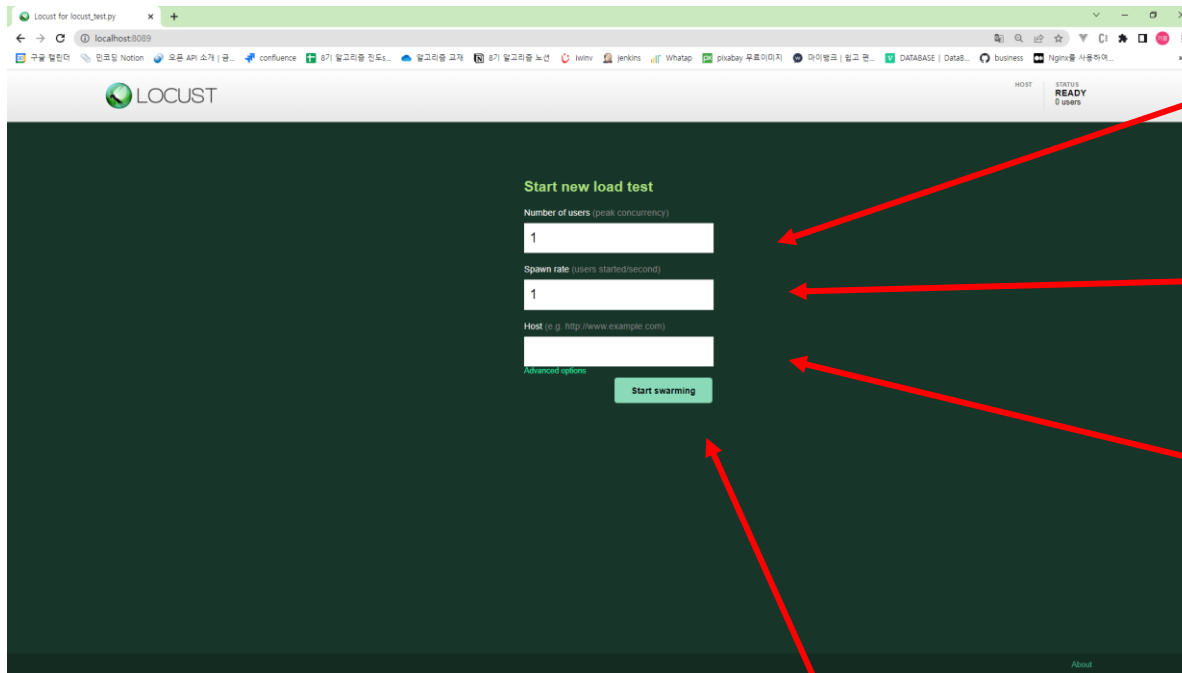
```
(venv)
GGR@DESKTOP-K5GQ6M2 MINGW64 ~/Desktop/ssafy-python-9th-pjt/pjt08 (master)
$ locust -f ./locust_test.py
[2023-02-08 16:05:10,980] DESKTOP-K5GQ6M2/INFO/locust.main: Starting web interface at http://0.0.0.0:8089 (accepting connections from all network interfaces)
[2023-02-08 16:05:10,996] DESKTOP-K5GQ6M2/INFO/locust.main: Starting Locust 2.14.2
[2023-02-08 16:09:07,111] DESKTOP-K5GQ6M2/INFO/locust.runners: Ramping to 500 users at a rate of 10.00 per second
test start
test start
test start
test start
test start
test start
test start
test start
test start
test start
test start
test start
```

- <http://localhost:8089> 로 접속하면 Web 화면을 볼 수 있다.
- [주의사항] 콘솔에서 출력되는 <http://0.0.0.0:8089> 로 접속하면 에러가 난다.



## Locust 사용법 - (4/10)

### 5. 웹 실행 화면 (<http://localhost:8089> 접속)



- Number of users

- 생성할 총 가상유저 수

- Spawn rate

- 동시에 접속하는 유저 수

- Host

- 서버 주소(Django 서버)

- ex) <http://localhost:8000>

클릭 시 가상 유저에 등록된 작업을 수행한다.

## Locust 사용법 - (5/10)

### 6. 웹 실행 화면 - Statistics 탭

접속 유저 수  
edit: 유저 수 설정 수정 가능

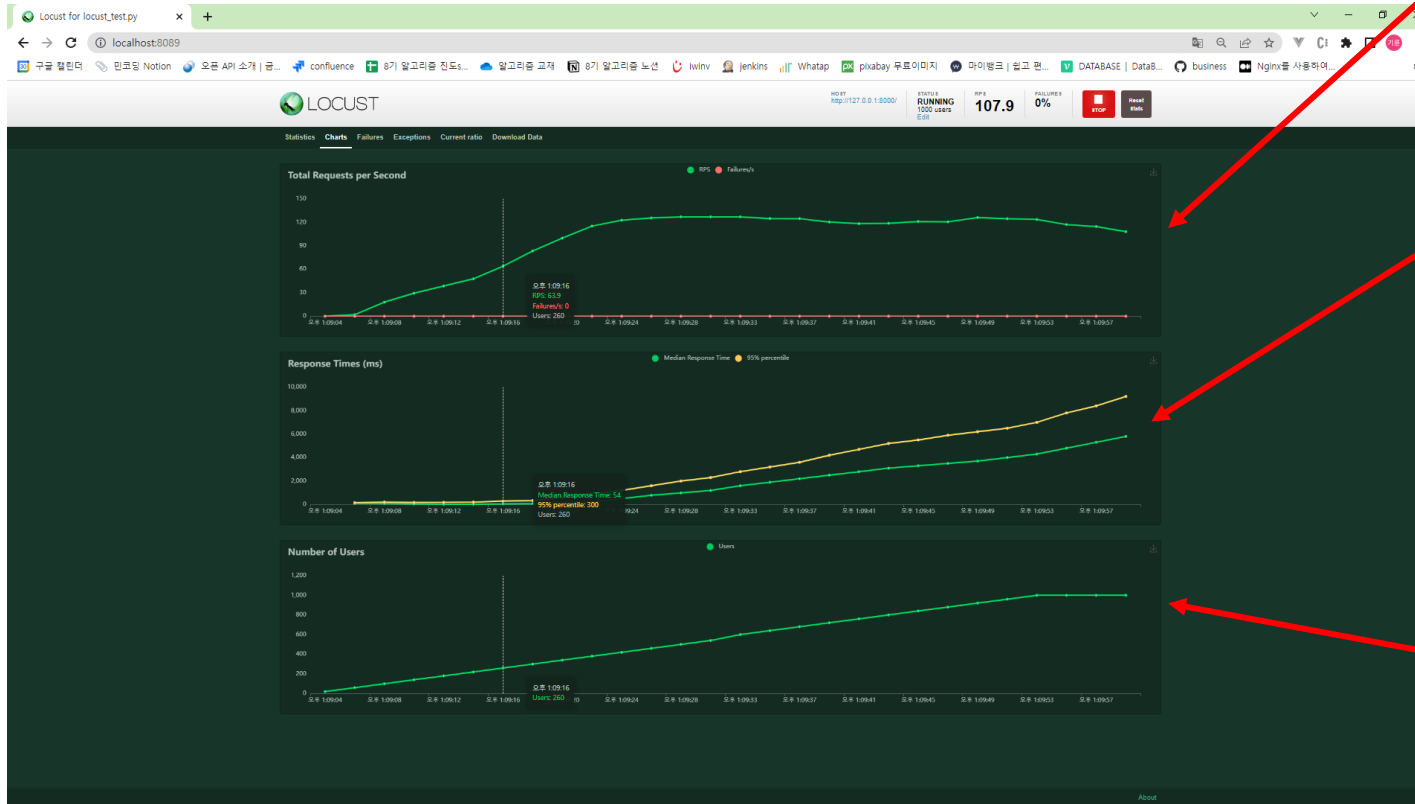
작업 중지



- 각 URL 에 대한 요청 수, 실패 수, 각 기준에 대한 응답 시간, 평균 응답 크기, RPS 등 다양한 통계 내용 확인 가능
- 전체 분석은 터미널에서 터미널 종료(Ctrl + C) 입력 또는 Download Data 탭의 Download Report 클릭 시 확인 가능

## Locust 사용법 - (6/10)

### 7. 웹 실행 화면 - Charts 탭



- 가로는 모두 시간을 의미한다.
- Total Requests per Second
  - 초록선: 초당 요청 수(RPS)
  - 빨간선: 초당 실패한 요청 수
- Response Times(ms)
  - 각 응답에 대한 평균 응답 시간
  - 노란선: (95% Percentile)
    - 95% 응답이 해당 시간 내에 처리되었다.
  - 초록선: (Median)
    - 응답 시간의 중앙값
- Number of Users
  - 동시에 요청을 보내는 유저 수

## Locust 사용법 - (7/10)

### 8. 웹 실행 화면 - Failures 탭

LOCUST

HOST <http://127.0.0.1:8080/> STATUS **RUNNING** 3000 users [Edit](#) RPS **82.5** FAILURES **26%** [STOP](#) [Reset Stats](#)

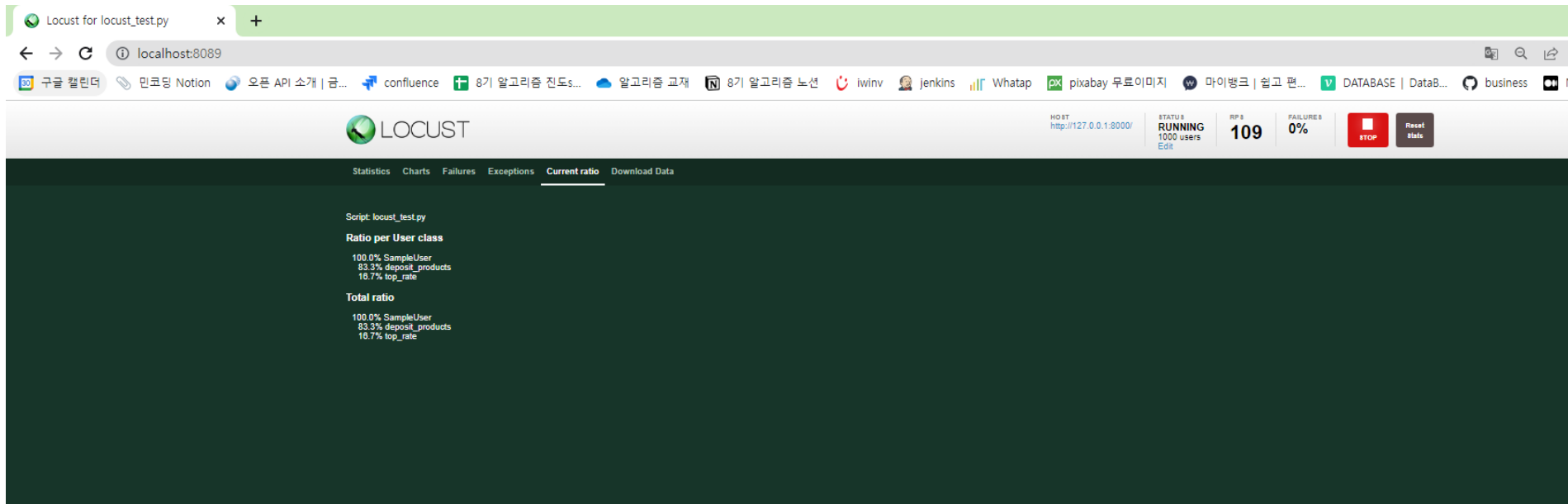
Statistics Charts **Failures** Exceptions Current ratio Download Data

# fails	Method	Name	Type
1114	GET	/finlife/deposit-products/	ConnectionRefusedError(10061, '[WinError 10061] 대상 컴퓨터에서 연결을 거부했으므로 연결하지 못했습니다.')
215	GET	/finlife/save-deposit-products/	ConnectionRefusedError(10061, '[WinError 10061] 대상 컴퓨터에서 연결을 거부했으므로 연결하지 못했습니다.')

- 실패한 요청에 대한 정보와 실패 원인이 출력된다.
  - ex) 대상 컴퓨터에서 연결을 거부했으므로 연결하지 못했습니다.

## Locust 사용법 - (8/10)

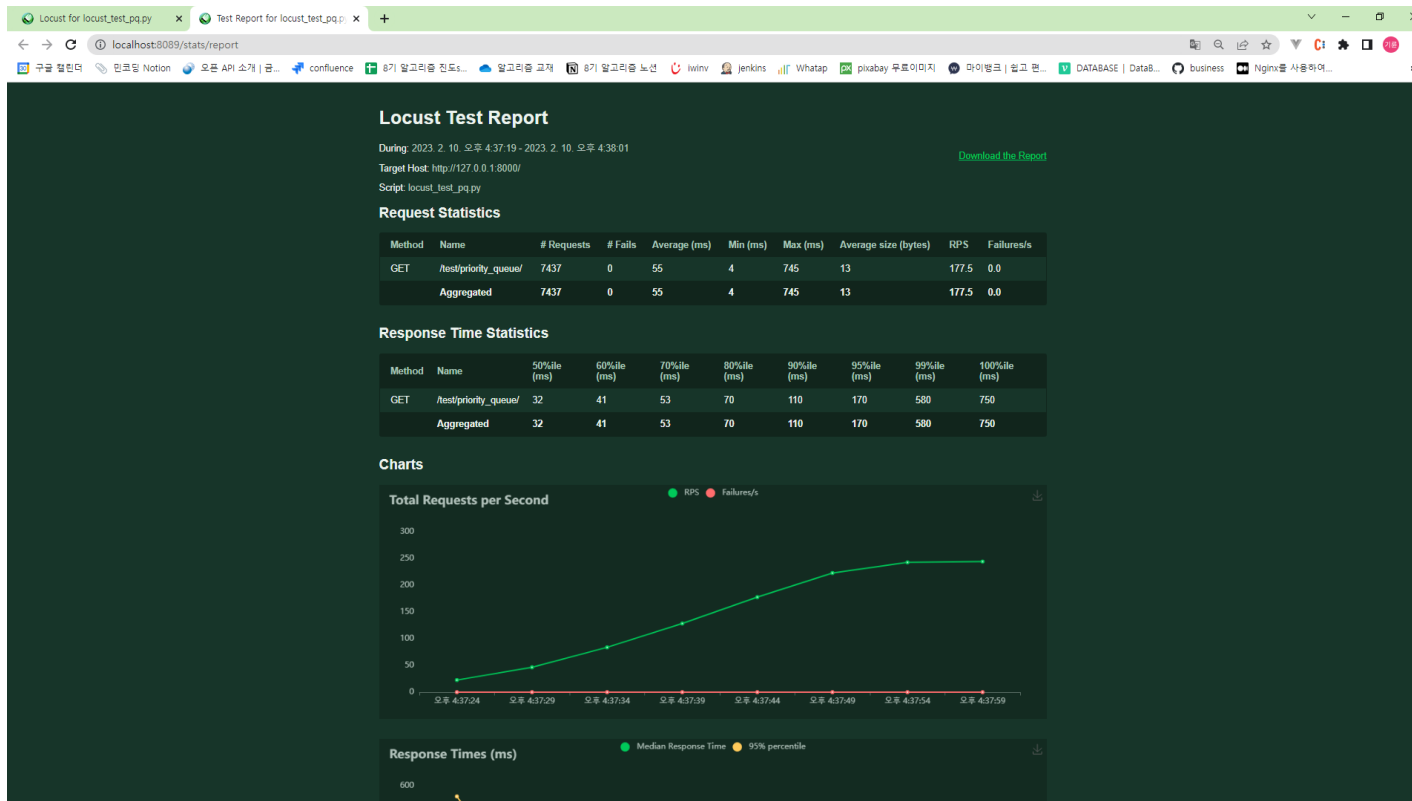
### 9. 웹 실행 화면 - Current ratio 탭



- 현재 작업이 수행된 비율을 출력한다.

## Locust 사용법 - (9/10)

### 10. 웹 실행 화면 - 결과 화면(Download Data -> Download Report)



## Locust 사용법 - (10/10)

### 11. 콘솔 종료 화면

```
Traceback (most recent call last):
  File "C:\Users\GGR\Desktop\ssafy-python-9th-pjt\pjt08\venv\lib\site-packages\gevent\ffi\loop.py", line 270, in python_check_callback
    def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument
KeyboardInterrupt
2023-02-09 11:53:20.53:207
[2023-02-09 11:53:20,975] DESKTOP-K5GQAM2/INFO/locust.main: Shutting down (exit code 1)
Type      Name                                     # reqs      # fails | Avg      Min      Max      Med | req/s    failures/s
-----
GET  /finlife/deposit-products/                1397      221 (15.82%) | 5051     189    13341    4400 | 51.79     8.19
GET  /finlife/save-deposit-products/             41       38 (92.68%) | 2700    2040    12215    2100 |  1.52     1.41
-----
Aggregated                                1438      259 (18.01%) | 4984     189    13341    4300 | 53.31     9.60

Response time percentiles (approximated)
Type      Name                                     50%      66%      75%      80%      90%      95%      98%      99%      99.9%  99.99%  100% # reqs
-----
GET  /finlife/deposit-products/                4400     6300     7500     8000     9300    10000    11000    11000    13000    13000    13000  1397
GET  /finlife/save-deposit-products/             2100     2100     2200     2200     2200     7200    12000    12000    12000    12000    12000    41
-----
Aggregated                                4300     6300     7400     8000     9300    10000    11000    11000    13000    13000    13000  1438

Error report
# occurrences  Error
-----
38      GET /finlife/save-deposit-products/: ConnectionRefusedError(10061, '[WinError 10061] 대상 컴퓨터에서 연결을 거부했으므로 연결하지 못했습니다.')
```

RPS 관련 통계

응답 시간 관련 통계

에러 관련 내용

- 콘솔에서 Locust 종료 시(Ctrl + C) 위와 같이 전체 요청에 대한 분석을 콘솔에서 확인할 수 있다

실습



## | 테스트 주의사항

- 원래는 서버에 배포된 API 또는 프로그램에 부하 테스트를 해야한다.
- 하지만, 현재는 PC 에서 작동 중인 서버로 요청을 보내는 것
  - PC 의 성능에 따라 결과가 매우 달라진다.
  - 현재 서버가 작동 중인 PC 에서 테스트를 진행하므로, 테스트 중 다른 조작을 하지 말 것!

## | 정렬 알고리즘 구현하기

- 대상
  1. 파이썬 내장 정렬함수 -  $O(N \log N)$
  2. 버블 정렬 -  $O(n^2)$
  3. 우선순위 큐 - 삽입:  $O(\log N)$ , 삭제:  $O(\log N)$
- 시나리오1. 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 의 배열을 만들어 가장 큰 값 찾기
- 시나리오2. (10배) 랜덤 배열 크기 10,000 / 랜덤 범위 50,000 의 배열을 만들어 가장 큰 값 찾기

## | 정렬 알고리즘 구현하기

- 가상 환경 설정

```
# performance_test 폴더에서 진행  
$ source venv/Scripts/activate
```

- 테스트용 Django 프로젝트 및 앱 생성

```
(venv) $ django-admin startproject mypjt .
```

```
(venv) $ python manage.py startapp test
```

## | 정렬 알고리즘 구현하기

- 각 정렬 알고리즘에 요청을 보낼 수 있도록 코드 작성

test/urls.py

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('normal_sort/', views.normal_sort),
6     path('priority_queue/', views.priority_queue),
7     path('bubble_sort/', views.bubble_sort),
8 ]
```

test/views.py

```
1 from django.http import JsonResponse
2 import random
3
4 def bubble_sort(request):
5     li = []
6     for i in range(1000):
7         li.append(random.choice(range(1, 5000)))
8     for i in range(len(li) - 1, 0, -1):
9         for j in range(i):
10             if li[j] < li[j + 1]:
11                 li[j], li[j + 1] = li[j + 1], li[j]
12     context = {
13         'top': li[0]
14     }
15     return JsonResponse(context)
16
17 # Create your views here
18 def normal_sort(request):
19     li = []
20     for i in range(1000):
21         li.append(random.choice(range(1, 5000)))
22     li.sort(reverse=True)
23     context = {
24         'top': li[0]
25     }
26     return JsonResponse(context)
27
28 from queue import PriorityQueue
29
30 def priority_queue(request):
31     pq = PriorityQueue()
32     for i in range(1000):
33         pq.put(-random.choice(range(1, 5000)))
34     context = {
35         'top': -pq.get()
36     }
37     return JsonResponse(context)
```

## | 정렬 알고리즘 구현하기

- 테스트 스크립트 작성하기

locust\_test.py

```
1 from locust import HttpUser, task, between
2
3 class SampleUser(HttpUser):
4     wait_time = between(1, 3)
5
6     def on_start(self):
7         print('test start')
8
9     @task
10    def normal_sort(self):
11        self.client.get("test/normal_sort/")
12
13    @task
14    def priority_queue(self):
15        self.client.get("test/priority_queue/")
16
17    @task
18    def bubble_sort(self):
19        self.client.get("test/bubble_sort/")
```

테스트 시나리오

1. 모든 Task 를 주석 처리 한다.
  - task 를 하나씩만 주석을 풀어 활성화 시킨다.

2. Locust 를 실행한다.

```
(venv) $ locust -f locust_test.py
```

3. 결과를 웹에서 확인한다

<http://localhost:8089> 접속

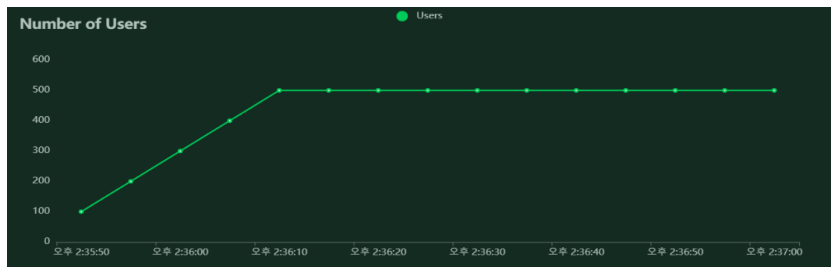
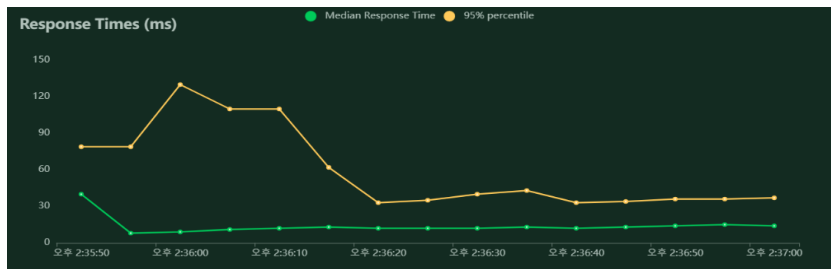
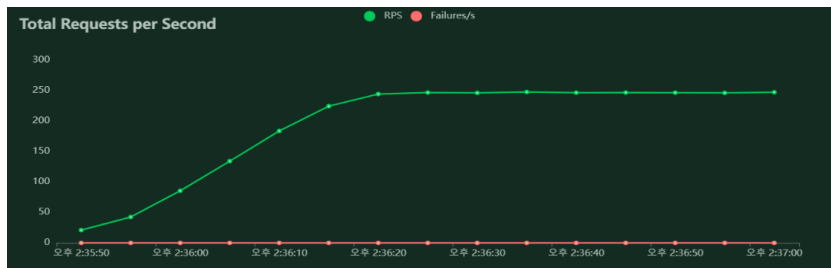
4. 테스트가 끝난 task 를 주석처리 후 다음에 테스트 할 task 주석을 풀어 활성화 시킨다.

5. 위 과정을 반복한다.

## 예시 - 시나리오 1

## 테스트 결과 - Python Built-in Sort

- 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 / 동시 사용자: 500 / 동시 접속자: 20



### Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/normal_sort/	16739	0	20	2	667	13	212.0	0.0
Aggregated		16739	0	20	2	667	13	212.0	0.0

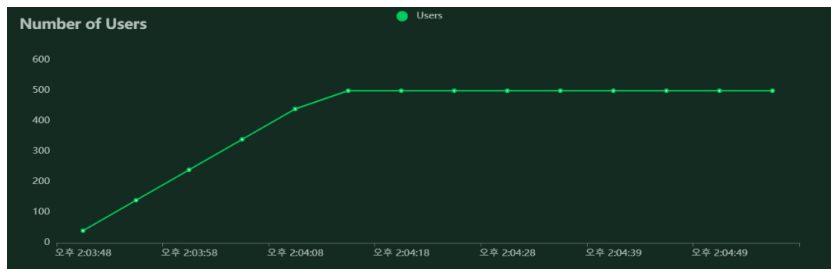
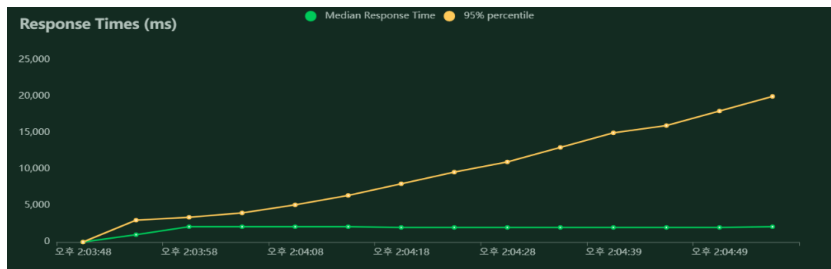
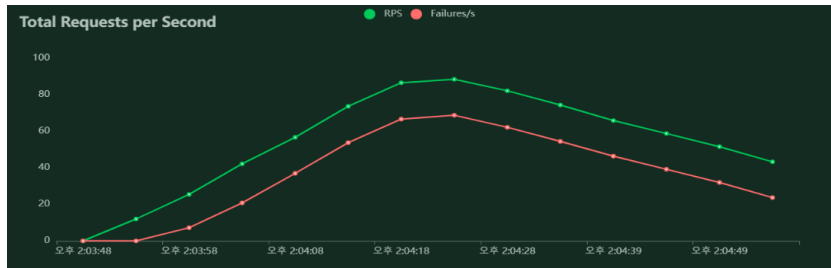
### Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/normal_sort/	13	16	19	23	34	50	120	670
Aggregated		13	16	19	23	34	50	120	670

- 평균 RPS: 212.0
- 응답 시간: 모든 응답이 0.6초 이내
- 시작 할 땐 병목이 잠깐 발생하지만 곧 해결된다
- 결론: 사용자 수가 늘어날 때를 제외하고 매우 안정적

## 테스트 결과 - 버블 정렬

- 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 / 동시 사용자: 500 / 동시 접속자: 20



Request Statistics									
Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/bubble_sort/	3988	2665	3917	122	20908	4	59.3	39.7
Aggregated		3988	2665	3917	122	20908	4	59.3	39.7

Response Time Statistics									
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/bubble_sort/	2000	2100	2100	3800	11000	16000	19000	21000
Aggregated		2000	2100	2100	3800	11000	16000	19000	21000

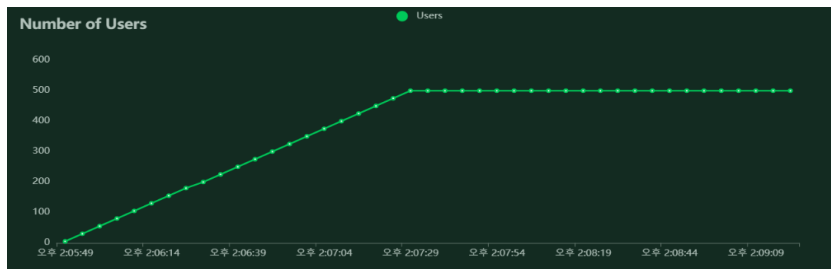
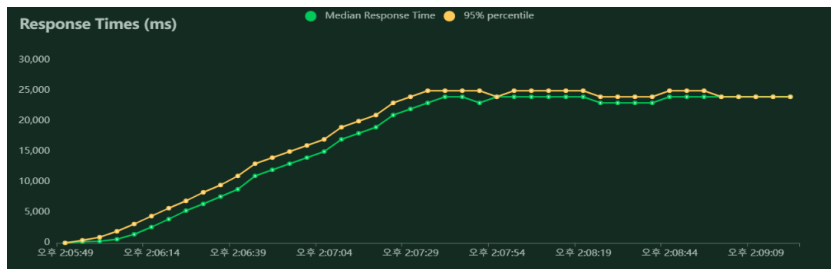
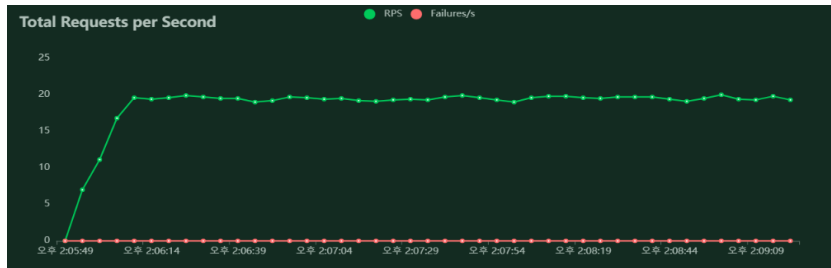
- 평균 RPS: 59.3
- 응답 시간: 중간: 0.2초 / 최대: 21초
- 응답 시간에서 중간 값과 95% percentile 의 차이가 커진다.
- 결론: 요청이 늘어나면서 병목 현상이 바로 발생한다.
  - 이로 인해 실패가 점점 늘어난다.

서버가 감당하기 힘든 알고리즘이다!



## 테스트 결과 - 버블 정렬

- 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 / 동시 사용자: 500 / 동시 접속자: 5 (서버가 감당할 수준)



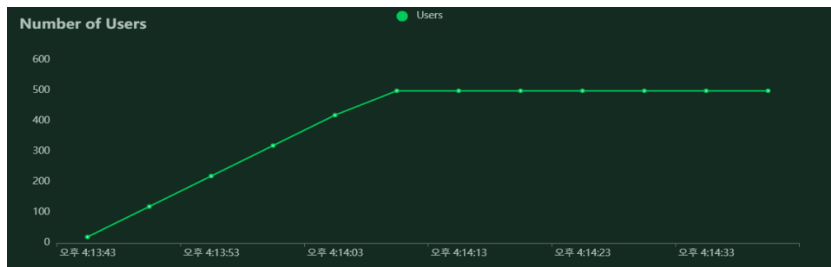
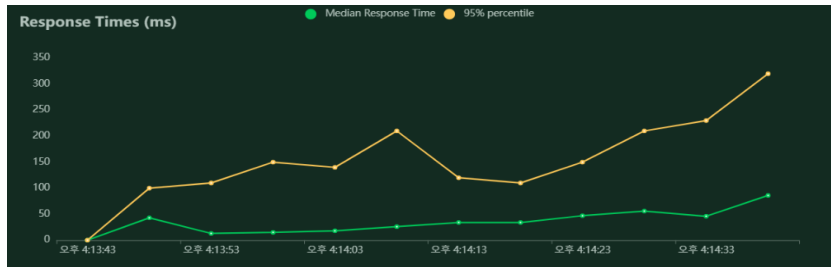
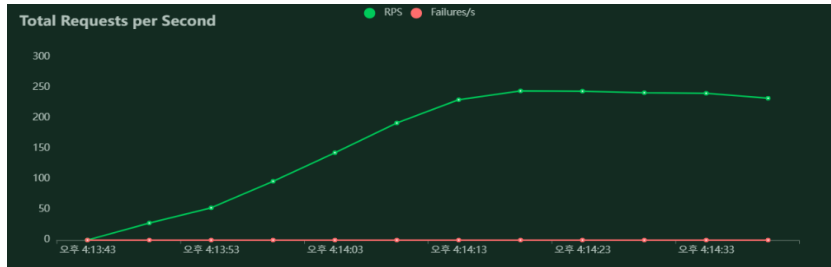
Request Statistics									
Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/bubble_sort/	4131	0	16644	50	24920	13	19.2	0.0
Aggregated		4131	0	16644	50	24920	13	19.2	0.0

Response Time Statistics									
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/bubble_sort/	23000	23000	23000	24000	24000	24000	25000	25000
Aggregated		23000	23000	23000	24000	24000	24000	25000	25000

- 평균 RPS: 19.2
- 응답 시간: 모든 요청이 2.5초 이내
- 최고/최악일 때 응답 차이가 거의 없다.
- 결론: 유저 수와 응답 시간이 비슷하게 올라간다는 점과 RPS 나 응답 시간이 더 늘어나지 않고 일정하다는 점에서 결과는 이상적이나, 응답 시간(2.5초)이 오래 걸린다!

## 테스트 결과 - 우선순위 큐

- 랜덤 배열 크기 1,000 / 랜덤 범위 5,000 / 동시 사용자: 500 / 동시 접속자: 5 (서버가 감당할 수준)



Request Statistics									
Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/priority_queue/	25581	0	57	3	1116	13	219.5	0.0
Aggregated		25581	0	57	3	1116	13	219.5	0.0

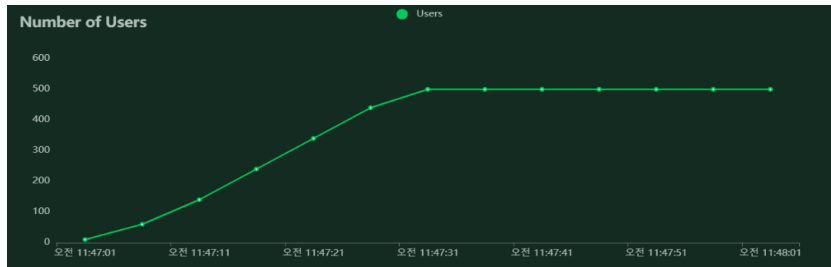
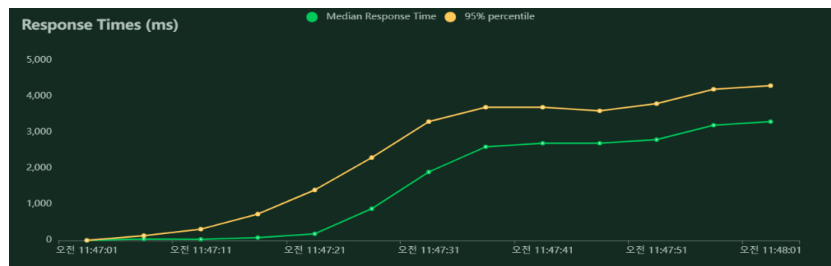
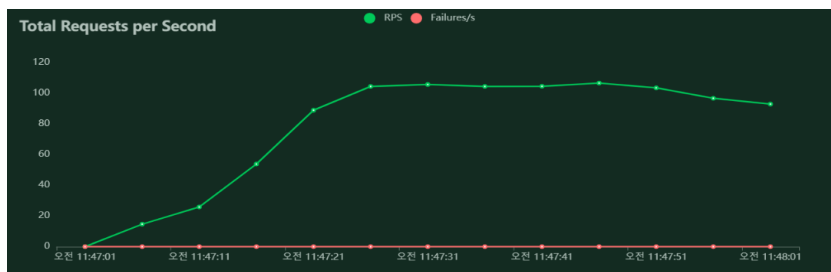
Response Time Statistics									
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/priority_queue/	36	46	58	76	110	160	510	1100
Aggregated		36	46	58	76	110	160	510	1100

- 평균 RPS: 219.5
- 응답 시간: 모든 응답이 1.1초 이내
- 응답 시간에서 중간값과 95% percentile 의 차이가 커진다.
- 결론: 알고리즘의 어딘가에서 병목 현상이 발생한다!  
언젠가 서버에 과부하가 온다.

## 예시 - 시나리오 2

## 테스트 결과 - Python Built-in Sort

- 랜덤 배열 크기 10,000 / 랜덤 범위 50,000 / 동시 사용자: 500 / 동시 접속자: 20



Request Statistics									
Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/normal_sort/	5795	0	2119	9	8486	14	90.7	0.0
Aggregated		5795	0	2119	9	8486	14	90.7	0.0

Response Time Statistics									
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/normal_sort/	2400	2700	3000	3300	3600	3800	4300	8500
Aggregated		2400	2700	3000	3300	3600	3800	4300	8500

- 평균 RPS: 90.7 ( 시나리오1: 212 )
- 응답 시간: 중간: 0.2초 / 최대: 8.5초
- RPS 가 확연히 감소하였다.
- 응답 시간에서 중간 값과 95% percentile 의 차이가 변동이 없다.
- 결론: 서버의 응답 시간이 점점 늘어난다.

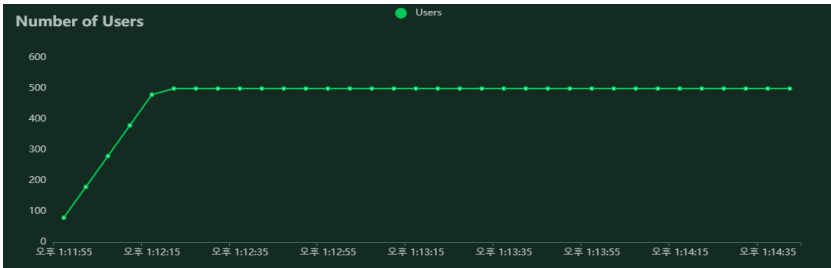
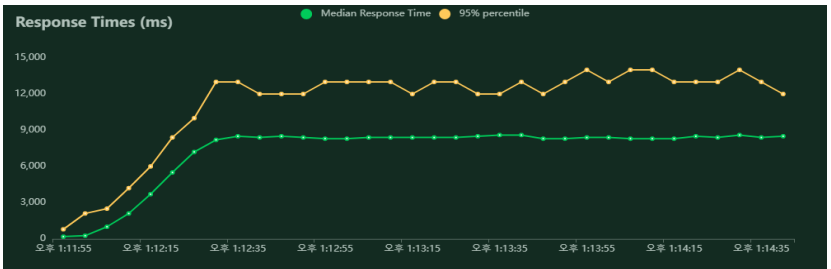
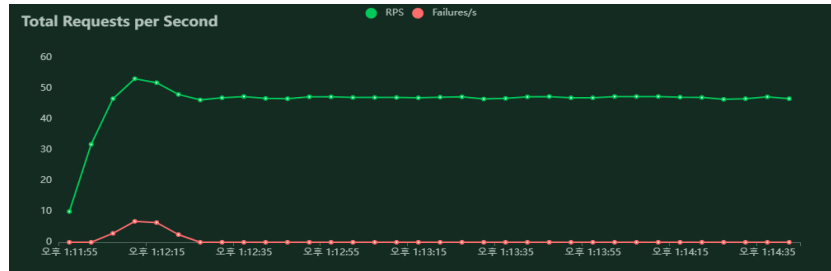
즉, 갈수록 서버가 느려지는 현상(RPS 감소)가 발생할 것이다.

## | 테스트 결과 - 버블 정렬

- 랜덤 배열 크기 10,000 / 랜덤 범위 50,000 / 동시 사용자: 500 / 동시 접속자: 20
- 동시 사용자와 동시 접속자 수에 관계 없이 버블 정렬은 결과를 나타내지 못하였다.
  - 서버 시간 초과
- 응답에 너무 많은 시간이 걸리는 알고리즘은 테스트가 불가능하다 !

## 테스트 결과 - 우선순위 큐

- 랜덤 배열 크기 10,000 / 랜덤 범위 50,000 / 동시 사용자: 500 / 동시 접속자: 20



Request Statistics									
Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/test/priority_queue/	8065	93	7639	21	21536	13	47.0	0.5
Aggregated		8065	93	7639	21	21536	13	47.0	0.5

Response Time Statistics									
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/test/priority_queue/	8100	8500	9000	9700	11000	12000	15000	22000
Aggregated		8100	8500	9000	9700	11000	12000	15000	22000

- 평균 RPS: 47.0 ( 시나리오1: 219.5 )
- 응답 시간: 중간: 8.1초 / 최대: 15초
- RPS 가 확연히 감소하였다.
- 응답 시간에서 중간 값과 95% percentile 의 차이가 변동이 있다.
- 결론: 전체 응답 시간은 그대로 인 것으로 보아

알고리즘의 특정 부분에서 병목 현상이 발생한다.

또한, 점점 느려지진 않지만, 응답 시간 자체가 너무 느리다!

## 테스트 결론

## | 결론

- 직접 구현한 우선순위 큐보다 파이썬의 내장 함수가 안정적이고 빠르다.
  - 병목 현상이 발생하지 않음
  - 응답 시간이 최고/최악 모두 빠르다
- 알고리즘에 따라 서버 성능이 크게 좌우될 수 있다.
- 테스트 결과가 보여주는 내용은 작성한 결론 외에도 수 많은 정보를 내포하고 있다!
  - 깊게 공부해야 한다.



## | 요약

- 테스트 및 성능 테스트의 개념을 알아보았다.
- Locust 를 활용한 부하 테스트 실습을 진행해 보았다.
- 스트레스 테스트 등의 다른 테스트는 명확한 목표를 정하고 진행해야 한다.
  - 개선사항의 방향을 찾기 위해 테스트를 진행한다.
  - 예시) 우리 서버는 반드시 0.8초 이내에 모든 응답을 주어야 한다.
- 여러 번 테스트를 해보아야 정확한 결과를 받을 수 있다.
  - 최대 부하 지점(임계점)의 부하를 지속하여 서버를 테스트해보자. (부하 테스트)
  - 과부하가 오는 시점(중단점)을 찾아 지속적 혹은 반복적으로 서버를 테스트해보자. (스트레스 테스트)

# 요구사항

## | 공통 요구사항

- Locust 테스트 시 활용한 Django 프로젝트(mypjt)를 활용합니다.
- 제공된 CSV 파일(test\_data.CSV) 를 활용합니다.
- .gitignore 파일을 추가하여 불필요한 파일 및 폴더는 제출하지 않도록 합니다.
- 명시된 요구사항 이외에는 자유롭게 작성해도 무관합니다.

## | 세부 요구사항

- A. CSV 데이터를 DataFrame 으로 변환 후 반환
- B. 결측치 처리 후 데이터 반환
- C. 알고리즘 구현하기(평균 나이와 가장 비슷한 10명)
- D. Locust 를 활용한 알고리즘 성능 측정

## | A. CSV 데이터를 DataFrame 으로 변환 후 반환

- 제공한 데이터(data/test\_data.CSV)를 Django 에서 읽어오도록 구현합니다.
  - 프로젝트 경로와 동일한 폴더에 data 폴더를 저장합니다.
- Numpy 혹은 Pandas 의 CSV 를 읽어오는 함수를 활용하여 완성합니다.
- DataFrame 생성 시 columns 옵션을 적절히 활용합니다.
- [참고] DataFrame 은 아래 방법을 사용하여 반환할 수 있습니다.

```
# records: 리스트 원소를 각각 하나의 레코드로 만들기 위해 주는 옵션
data = df.to_dict('records')

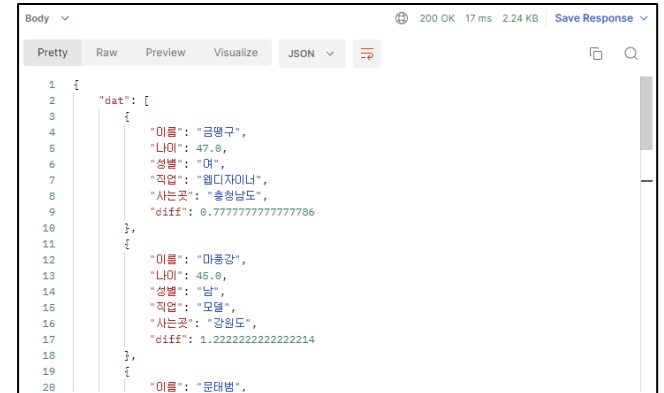
# JSON 형태로 응답합니다.
return JsonResponse({ 'dat': data })
```

## | B. 결측치 처리 후 데이터 반환

- Pandas 라이브러리의 특정 값 반환 함수를 활용합니다.
- 제공한 3.Pandas\_Advanced.md 파일을 참고합니다.
- 비어 있는 값을 “NULL” 문자열로 치환 후 DataFrame 을 반환합니다.

## C. 알고리즘 구현하기(평균 나이와 가장 비슷한 10명)

- DataFrame 의 “나이” 필드를 활용합니다.
- 평균 나이와 가장 비슷한 나이인 10개 행을 새로운 DataFrame 으로 만들어 반환합니다.
- [힌트] 힌트와 다르게 구현하셔도 좋습니다.
  1. 나이 필드의 평균값을 구합니다.
  2. 각 행과 평균값의 차이를 구한 후, 절대값을 취하여 새로운 필드로 만듭니다.
  3. 새로 생성한 필드를 기준으로 가장 작은 10개의 행을 선택합니다.
  4. 선택된 10개의 행을 JSON 형태로 응답합니다.



```
1 {
2   "dat": [
3     {
4       "이름": "김영구",
5       "나이": 47.0,
6       "성별": "여",
7       "직업": "웹디자인",
8       "사는곳": "송정남도",
9       "eiff": 0.7777777777777778
10    },
11    {
12      "이름": "마종강",
13      "나이": 45.0,
14      "성별": "남",
15      "직업": "모델",
16      "사는곳": "강원도",
17      "eiff": 1.2222222222222224
18    },
19    {
20      "이름": "문태범",
```

## | D. Loucst 를 활용한 알고리즘 성능 측정

- C 번에서 구현한 함수를 활용합니다.
- C 번의 함수를 테스트 할 수 있도록 Locust 스크립트 파일을 작성합니다.
- 총 접속자 수와 동시 접속자 수를 변경하며 여러 번 성능 테스트를 시도합니다.
  - README.md 파일에 “총 접속자, 동시 접속자, 평균 RPS, 응답 시간“ 을 기록합니다.
- 다른 사람들이 구현한 알고리즘과 나의 알고리즘의 성능을 비교해봅니다.
  - PC 성능에 따라 다른 결과가 나올 수 있으므로, 코드를 전달 받도록 합니다.
  - 내 PC 에서 다른 사람들의 알고리즘을 동작 시켜보고, 결과를 기록합니다.
- 나의 알고리즘과 성능 차이가 나는 이유를 분석하여 README.md 에 작성합니다.



# 제출

## | 제출 시 주의사항

- 제출기한은 금일 18시까지입니다. 제출기한을 지켜 주시기 바랍니다.
- 반드시 README.md 파일에 단계별로 구현 과정 중 학습한 내용, 어려웠던 부분, 새로 배운 것들 및 느낀 점 등을 상세히 기록하여 제출합니다.
  - 단순히 완성된 코드만을 나열하지 않습니다.
- 위에 명시된 요구사항은 최소 조건이며, 추가 개발을 자유롭게 진행할 수 있습니다.
- <https://lab.ssafy.com/> 에 프로젝트를 생성하고 제출합니다.
  - 프로젝트 이름은 '프로젝트 번호 + pjt' 로 지정합니다. (ex. 01\_pjt)
- 반드시 각 반 담당 교수님을 Maintainer 로 설정해야 합니다.