Big Data Processing & Trend Analysis

Big Smile

김가윤 이규은 정석준 조인식 추연호

01
프로젝트 개요

- 주제 및 컨셉

02
프로젝트 목표

- 팀 목표
- 마일스톤

03
프로젝트 설계

- 아키텍쳐 설계

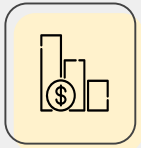
04
프로젝트 시연 영상

05
팀원 소개

- 개인 목표
- 기술 스택 및 R&R
- 달성한 점 & 아쉬운 점
- 소감

TweeTrend

코로나 관련 트윗 기반 트렌드 대시보드 서비스



주요 기능

- 트위터에서 언급되는 코로나 관련 트윗을 실시간 피드로 제공한다.
- 코로나 관련 연관어를 분석하여 기간별 트렌드를 대시보드 형태로 제공한다.

확장 가능성

- 특정 주제에 국한되지 않고, 유저가 검색한 키워드에 대한 트렌드 분석 페이지를 제공한다.
- 트위터 외에 다른 SNS의 데이터를 추가적으로 분석한다.

02 **프로젝트 목표** – 팀 목표

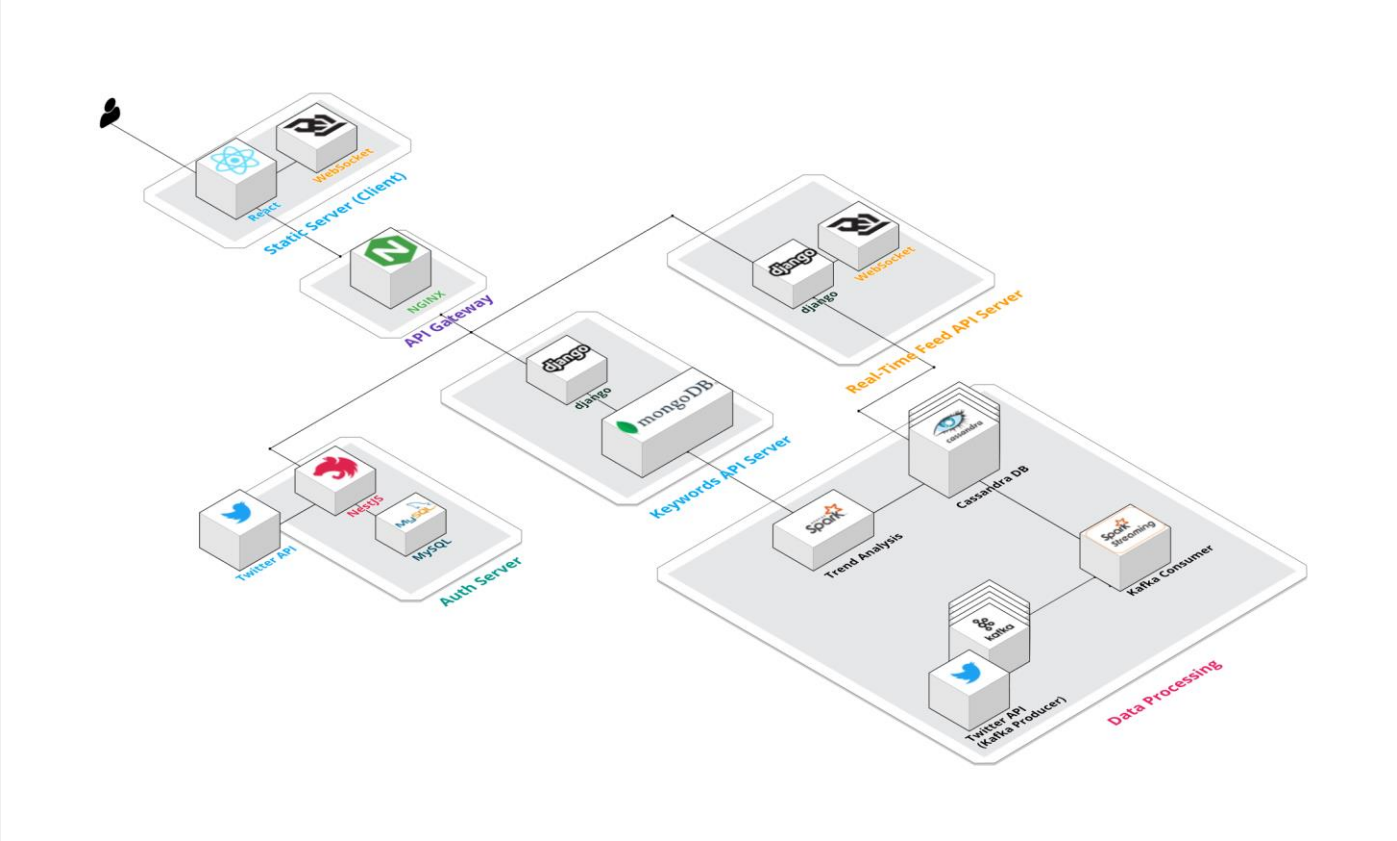
“ TweetDeck과 SomeTrend를 벤치마킹하여 각자 관심있는 기술을 습득하고, 기술을 통합하여 **TweeTrend**라는 하나의 서비스를 완성한다.

“ **TweeTrend** 서비스를 개발하면서 사용한 기술에 대해 서로에게 설명할 수 있을 정도로 관심 분야 기술에 대한 이해를 높이는 것.

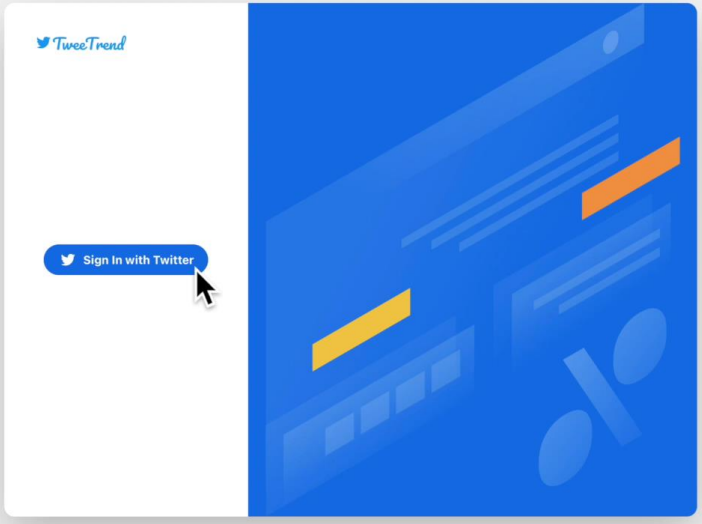
02 **프로젝트 목표** – 마일스톤

M1	**DevCamp 기간 내에 반드시 완료해야하는 목표**	한 키워드에 대해 데이터 처리부터 프론트엔드까지 완성된 하나의 흐름 만들기
M2	**기간 내 최선을 다한다면 가능할 수 있을 것 같은 목표**	부가적인 기능을 추가하고, 서버의 성능과 안정성을 높이는 것
M3	**기간 내에는 불가능하지만 궁극적으로 만들고자 하는 목표**	유저의 검색어 기반으로 확장하고, 다른 SNS 데이터까지도 활용하는 것

03 **프로젝트 설계** – 아키텍쳐 설계



04 프로젝트 시연 영상



05 **팀원 소개** – 개인 목표

조인식

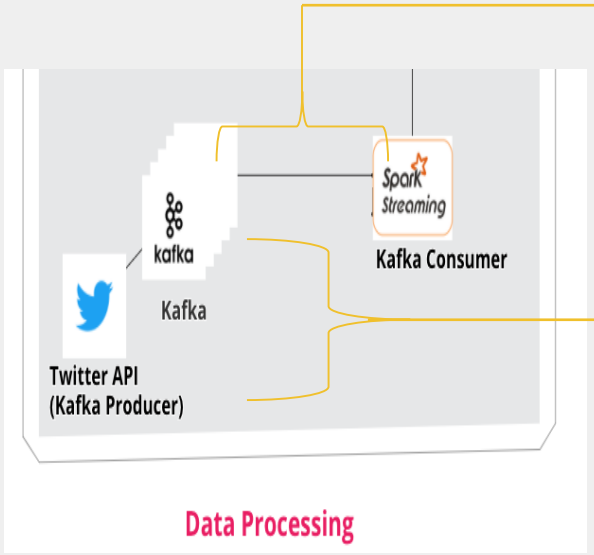
데이터처리 - Kafka

“실시간으로 생산되는 많은 양의 Twitter 데이터를
빠르고 유실없이 전달하고자 합니다.”

05 팀원 소개 – 기술 스택 및 R&R

조인식

데이터처리 - Kafka

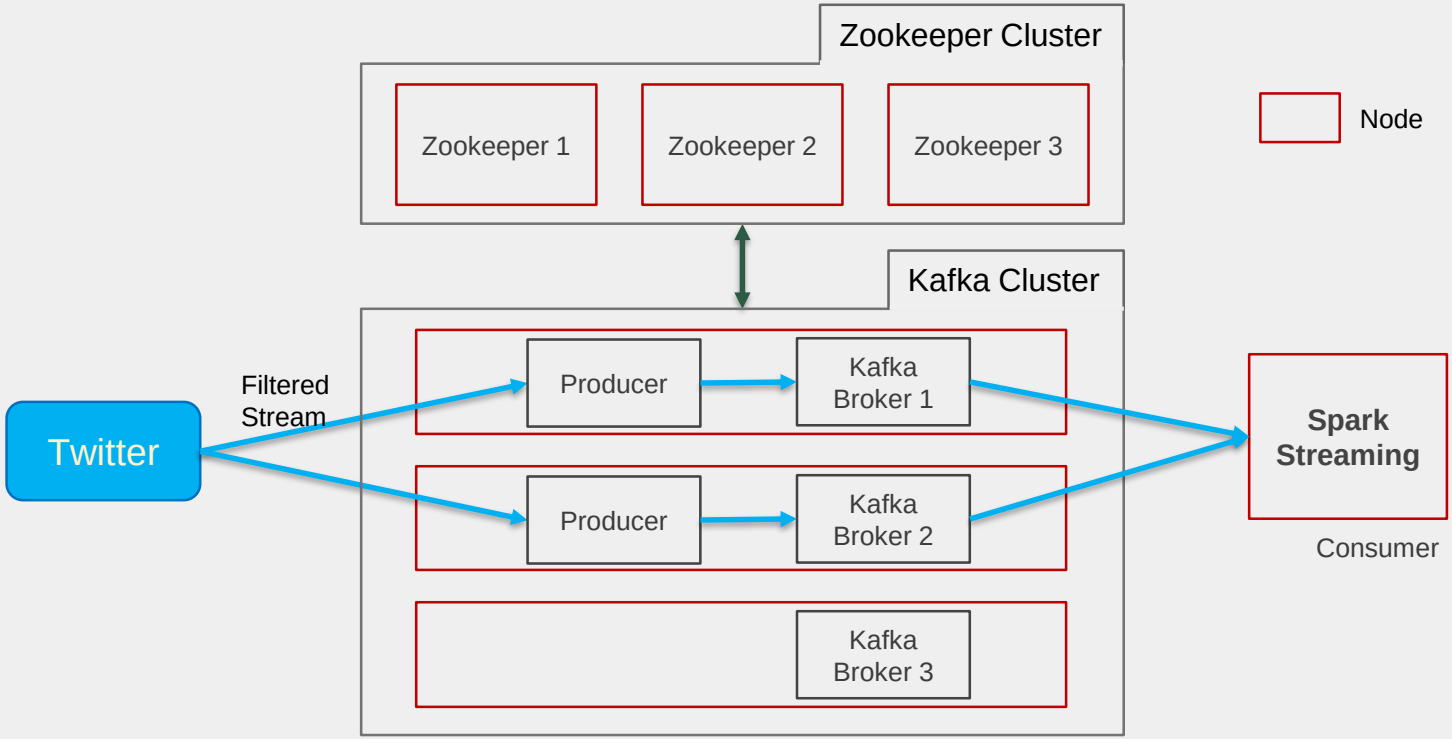


1. raw data 를 가공하기 위해 spark streaming 과의 연동
2. data consumer 로서 성능 최적화를 위한 파라미터 설정

1. Twitter API 로부터 게시되는 글을 실시간으로 수집
2. data producer 로서 성능 최적화를 위한 파라미터 설정
3. 안정적인 서버 운영을 위한 연결유지와 로깅시스템

05 **팀원 소개** – 개발 기능 및 역할

Kafka Architecture



05 **팀원 소개** – 개발 기능 및 역할

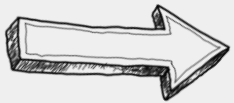
Analysis Data Source

```
[2021-02-01 18:37:03,285] [INFO] > [400] START...
[2021-02-01 18:37:03,286] [INFO] > {'remaining': None, 'limit': None, 'reset': None}
[2021-02-06 01:58:15,588] [ERROR] > BaseException: Error requesting bearer access token: HTTPSConnectionPool(hos
t='api.twitter.com', port=443): Max retries exceeded with url: /oauth2/token?grant_type=client_credentials (Caus
ed by NewConnectionError('<urllib3.connection.HTTPS
ew connection: [Errno 110] Connection timed out'))
[2021-02-06 13:59:42,066] [INFO] > { 'remaining' : None,  'limit' :
[2021-02-06 13:59:42,067] [ERROR] > RequestError: |This stream i
mit.
ed_tweets%2Csource&user.fields=entities%2Cid%
name%2Cid%2Ccountry%2Ccountry_code%2Cname%2Cp
onnection object at 0x7f2097d10250>, 'Connect
```

```
except TwitterRequestError as e:
    # ConnectionException will be caught here

    msg_list = []
    for msg in it
        msg_list.
    err_msg = "Re
    Log.e(err_msg

    if e.status_c
        self.prom
    elif e.status
        self.prom
    else:
        exit()

except TwitterCon
    Log.e(f"Conne
    self.prompt_r

except Exception as e:
    Log.e(f"Exception: {e}")
    self.prompt_reconnect_msg(2)
```

```
# DEBUG 레벨 이상의 로그를 `debug.log`에 출력하는 Handler
file_debug_handler = logging.FileHandler("debug.log")
file_debug_handler.setLevel(logging.DEBUG)
file_debug_handler.setFormatter(formatter)
logger.addHandler(file_debug_handler)

# ERROR 레벨 이상의 로그를 `error.log`에 출력하는 Handler
file_error_handler = logging.FileHandler("error.log")
file_error_handler.setLevel(logging.ERROR)
file_error_handler.setFormatter(formatter)
logger.addHandler(file_error_handler)
```

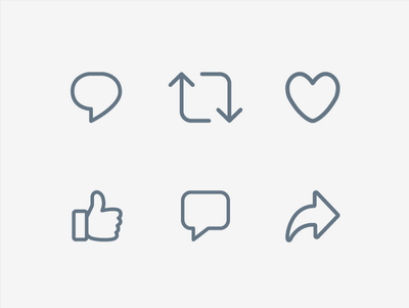
처리 방법도 다양한 많은 에러들



예외 처리와 로깅 시스템으로
데이터 수집 유지 및 원인분석

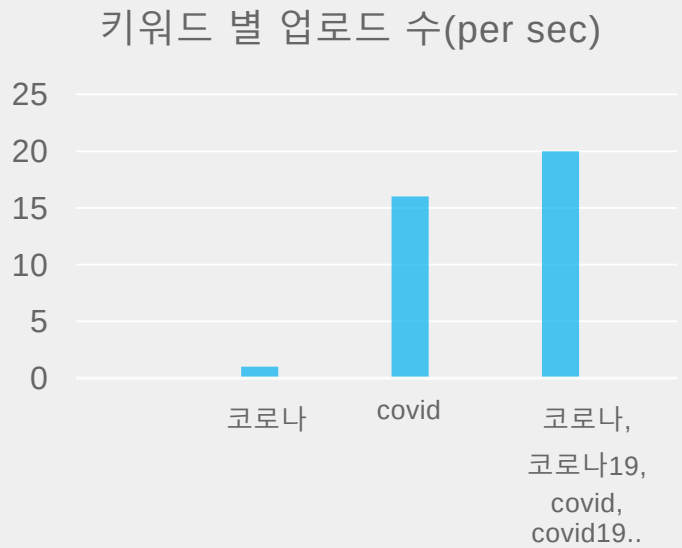
05 **팀원 소개** – 개발 기능 및 역할

Analysis Data Source



여러 종류의 Tweets

(1~6 KB/tweet)



100 ~ 200 KB / sec

05 **팀원 소개** – 개발 기능 및 역할

Parameter settting

Producer

- batch.size
- buffer.memory
- acks
- retries
- partitioner
- compression.type

Broker

- message.max.bytes
- num.network.threads
- num.partitions
- min.insync.replicas

Consumer

- fetch.min.bytes
- fetch.max.wait.ms
- auto.commit.enable
- heartbeat.interval.ms
- max.poll.interval.ms

05 **팀원 소개** – 개발 기능 및 역할

Parameter settting

Throughput

- Kafka 의 특성상 많은 데이터를 빠르게 쓰는 것은 문제가 없음

Latency

- 하나의 메시지를 가능한 빠르게 전달
- producer -> broker -> consumer

Durability

- 메시지의 유실을 최소화
- 이벤트 기반 micro-service 또는 데이터 수집 파이프라인

Availability

- Kafka 서버의 다운타임 최소화
- 장애발생 시 가장 빠르게 복구해야 하는 서비스

05 **팀원 소개** – 개발 기능 및 역할

Minimize Latency

1. broker 는 많이 partition 는 적게
   - 하나의 broker 에서 담당하는 partition 의 수를 줄인다.
   - 많은 수의 partition 은 데이터 복제를 위해 message 지연을 야기
2. linger.ms (default 0)
   - producer에서 broker로 데이터를 보내기 위해 기다리는 시간
3. acks (default 1)
   - producer가 broker에게 전달한 메시지의 저장 확인 방식
4. compression.type
   - 데이터를 전달할 때 사용할 압축 형식
5. fetch.min.bytes
   - broker에서 가져오는 최소 size

Guarantee Durability

1. Replication.factor
   - 데이터의 복제 수. 3 정도면 높은 수준의 durablility 지원
   - 많은 수의 partition 은 데이터 복제를 위해 message 지연을 야기
2. acks (= -1)
   - broker가 모든 replica에 복제한 후, producer에게 응답(ack)
3. log.flush.interval.ms / log.flush.interval.messages
   - 입력된 message 를 memory(page cache)에서 disk로 저장하는 수준
   - 값이 클수록 Disk I/O 가 적게 발생 -> 메모리 데이터 유실 가능
   - 값이 작으면 Disk I/O 가 많이 발생 -> 메모리 데이터 유실 거의 없음

05 **팀원 소개** – 개발 기능 및 역할

Parameter Test

Broker : 3, Partition : 9, Replicas : 3

```
inscho@inscho-D1 MINGW64 /c/kafka
$ bin/kafka-producer-perf-test.sh --topic test \
> --num-records 100000 \
> --record-size 3000 \
> --throughput 15000000 \
> --producer-props \
> acks=1 \
> bootstrap.servers=10.250.93.4:9093,10.250.93.16:9093,10.250.93.19:9093 \
> buffer.memory=33554432 \
> compression.type=gzip \
> batch.size=16384
100000 records sent, 24172.105390 records/sec (69.16 MB/sec) 144.69 ms avg latency, 558.00 ms max latency, 134 ms 50th, 292 ms 95th, 382 ms 99th, 399 ms 99.9th.
```

```
inscho@inscho-D1 MINGW64 /c/kafka
$ bin/kafka-producer-perf-test.sh --topic test \
> --num-records 100000 \
> --record-size 3000 \
> --throughput 15000000 \
> --producer-props \
> acks=-1 \
> bootstrap.servers=10.250.93.4:9093,10.250.93.16:9093,10.250.93.19:9093 \
> buffer.memory=33554432 \
> compression.type=gzip \
> batch.size=16384
100000 records sent, 25477.707006 records/sec (72.89 MB/sec) 184.84 ms avg latency, 548.00 ms max latency, 165 ms 50th, 388 ms 95th, 475 ms 99th, 500 ms 99.9th.
```

05 **팀원 소개** – 개발 기능 및 역할

Kafka Monitoring

Brokers

Id	Host	Port	JMX Port	Bytes In	Bytes Out	Size
1	10.250.93.4	PLAINTEXT:9092	9999	13k	51k	0 B
2	10.250.93.16	PLAINTEXT:9092	9999	13k	46k	0 B
3	10.250.93.19	PLAINTEXT:9092	9999	13k	45k	0 B

Combined Metrics

Rate	Mean	1 min	5 min	15 min
Messages in /sec	10.29	35.52	34.89	35.36
Bytes in /sec	11k	40k	39k	40k
Bytes out /sec	40k	143k	131k	132k
Bytes rejected /sec	0.00	0.00	0.00	0.00
Failed fetch request /sec	0.00	0.00	0.00	0.00
Failed produce request /sec	0.00	0.00	0.00	0.00

05 **팀원 소개** – 달성한 점

Accomplishment

- Kafka의 구조 및 설계 등을 이해하고, 여러 파라미터들에 대한 지식을 쌓음
- 쌓은 지식을 바탕으로 서비스에 맞게 파라미터를 조정하여 성능향상을 이룸
- 확장성을 고려하여 topic, partition 나아가 server를 늘릴 수 있도록 설계함
- 예외처리/로깅시스템을 활용하여 데이터 수집에 지장이 없도록 노력함
- 데이터 파이프라인에 다양한 요소들이 있음을 알고, 학습 의지가 생김

05 **팀원 소개** – 아쉬운 점

Regret

- 환경 구축에 들인 시간이 너무 많은 점.
- 개발보다는 문서 읽기, 구조 이해에 들인 시간도 많은 점.
- 그럼에도 개발 중엔 이렇게 하는 게 최선인지에 대한 의문.
- 데이터가 정말 걱정없을 정도로 많았으면 하는 점.
- Kafka 외 다른 부분 기술 경험에 대한 아쉬움.
- 팀원들에게 더 도움이 되지 못한 아쉬움.

05 **팀원 소개** – 소감

조인식

- 이전부터 경험해 보고 싶었던 데이터 엔지니어링 분야에 도전해볼 수 있었던 점이 가장 좋았다. 특히 여러 서버를 운영/관리 해보는 경험은 회사의 지원이 아니었다면 못해봤을 것이라고 생각한다.
- 팀 프로젝트만의 재미를 오랜만에 느껴봤다. 이렇게 잘 갖추어진 상태에서 좋은 팀원들과 값진 경험을 쌓고, 관계를 이어갈 수 있게 되어 기쁘다.
- 배운 점도 많고, 부족한 점을 인지하게 된 순간도 많았다. 항상 겸손하고 배움을 게을리하지 않는 개발자가 되어야겠다.
- 아낌없이 지원해 주신 직원분들, 옆에서 함께 고생해준 동료 인턴분들 모두에게 감사하다.

05 **팀원 소개** – 개인 목표

김가윤

데이터 처리 - Spark & Spark Streaming

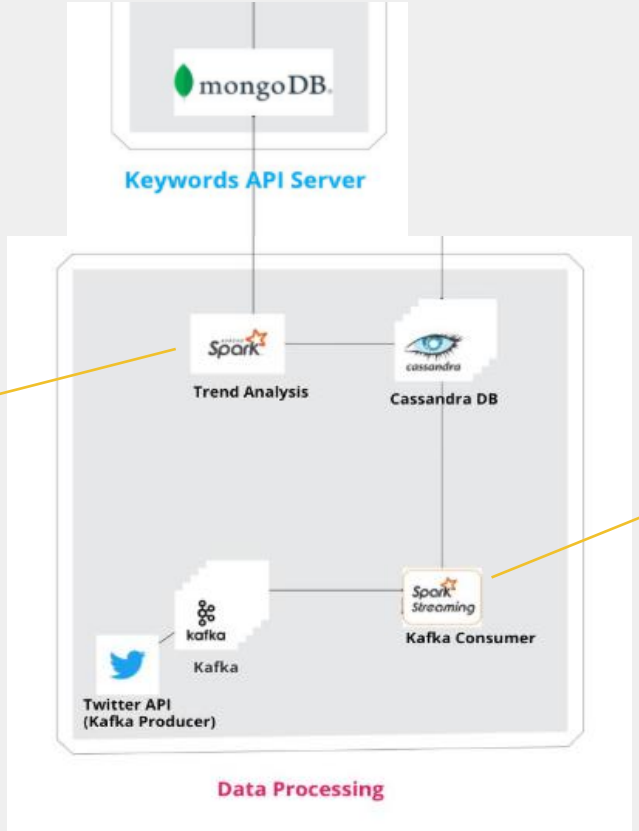
Kafka에서 받아온 실시간 데이터를 가공하고 분석하여 Cassandra DB와 Mongo DB에 각각 저장함으로써 데이터를 안정적으로 확보하는 것입니다 .

05 **팀원 소개** – 기술 스택 및 R&R

김가윤

데이터 처리 - Spark & Spark Streaming

1. Trend 대시보드를 위한 데이터 가공&분석
2. Mongo DB 저장 기능 구현
3. 데이터프레임의 빠른 가공 및 연산 가능



1. Kafka와 Cassandra DB 연결
2. 실시간 데이터처리 (수집&가공)
3. 메모리 저장 방식의 빠른 성능

김가윤

Spark Streaming: Kafka 컨슈머 기능

- **실시간 트윗 데이터를 가져오기 위한 Kafka 컨슈머 기능**
  - Kafka 브로커(서버)에 저장되어 있는 메시지(트윗 데이터)를 5초에 한번씩 가져오기
  - 실시간 피드에서 트윗 데이터를 보여줄 수 있도록 Cassandra DB 스키마에 맞춰 kafka에서 가져오는 동시에 데이터 가공 후 DB 저장하기
- **Kafka에서 가져오는 최초 메시지는 TransformedDStream 타입으로 가공 필요**
  - 트윗 데이터 json으로 변환 후 map 타입을 거쳐 foreachRDD로 개별 데이터에 대한 RDD 생성
  - RDD를 트윗 데이터 struct 타입에 맞춰 key값들을 필드로 구성하여 데이터프레임 생성
  - 불필요한 필드 삭제, 데이터 타입 변환 등 가공을 거쳐 Cassandra DB에 저장

김가윤

Spark: 코로나 Trend 분석

- **Cassandra DB에서 트윗 데이터 조회 후 데이터프레임 가공 및 합계 구하기**
  - Cassandra DB 파티션 기준인 1분 단위로 분석 (null값 처리, 리스트 값 꺼내오기 등)
  - 총 트윗 수, 리트윗 수, 사용자 수, 지역 수, 사용매체 수

- **코로나를 포함한 트윗 메시지에 언급된 연관어들을 활용한 실시간 트렌드 분석하기**
  - KoNLPy Okt 형태소 분석기를 사용하여 트윗 메시지 내 단어들의 누적 빈도수 구하기
  - 1분 동안 쌓인 단어들 중 가장 많은 빈도수 상위 5%로 자르고, 3개 미만인 단어들 제외하여 긍/부정 판별 후 Mongo DB 저장하기
  - 의미 없는 단어들을 제외하기 위해 stopwords에 추가하여 불용어 처리 및 코로나 주제에 맞게 단어별 긍정 또는 부정 스코어 점수 추가하기 (-2 부터 +2)

김가윤

Spark: 코로나 Trend 분석

```
MongoDB Schema
topic:  covid-19
collected_at:  1612612860
total_count:  18
user_count:  13
retweet_count:  17
region_count:  {}
related_words:  {'아산': (13, 0), '치료': (7, 1), '거리두기': (6, 2), '뉴스': (5, 0),
 '신천지': (5, 0), '현장': (5, 0), '대통령': (5, 0), '현황': (5, 0), '출장': (4, 0),
'공': (4, 0), '확진자': (4, -2), '사회': (3, 0), '차': (3, -1)}
reputation:  {'positive': 2, 'negative': 2, 'neutral': 9}
source:  {'Twitter Web App': 9, 'Twitter for iPhone': 5, 'Twitter for Android': 6}
```


```
"확": {
    "counts": 5884,
    "scores": 0
},
"진자": {
    "counts": 5813,
    "scores": 0
},
```

05 **팀원 소개** – 달성한 점

김가윤

1. Spark Streaming을 Kafka 컨슈머로 활용하여 Kafka 브로커에서 메시지(트윗 데이터) 가져오기
2. 트윗 데이터를 데이터프레임으로 변환하여 가공 후 Cassadra DB 스키마에 맞춰 저장하기
3. Cassandra DB에서 데이터 조회 후 데이터프레임으로 변환하여 가공 및 트렌드 분석 진행
4. 분석한 데이터를 Mongo DB에 실시간으로 저장 (1분 단위)
5. Spark Streaming과 Spark 분석 서버가 끊기지 않도록 오류 예외처리 완료

05 **팀원 소개** – 아쉬운 점

김가윤

1. Spark가 지원하는 다양한 기능들(Dataframe, RDD)을 좀 더 깊게 연구하고 사용해보지 못한 점
2. Trend 분석에서 Spark ML을 사용하여 머신러닝 분석에도 도전해보고 싶었지만 시간의 한계로 시도해보지 못한 점
3. 처음 목표했던 서비스 확장성을 고려하여 코로나 외 다양한 주제로 더 방대한 데이터 처리에 도전해보지 못한 점

05 **팀원 소개** – 소감

김가윤

대용량 데이터처리 기술에 처음 도전해봤는데 여러가지 플랫폼(Kafka, Spark, Cassandra DB)을 직접 활용해보며 데이터파이프라인 전반에 대한 이해도를 높일 수 있었습니다.

특히 Spark의 장점인 대용량 데이터 가공 기능을 익히고 구현해 볼 수 있어서 뿌듯했습니다.

열심히 배웠고, 삽질했고, 오류를 해결했고, 결과물을 직접 확인할 수 있어서 너무 재밌고 유익한 시간이었습니다. 감사합니다.

05 **팀원 소개** – 개인 목표

이규은

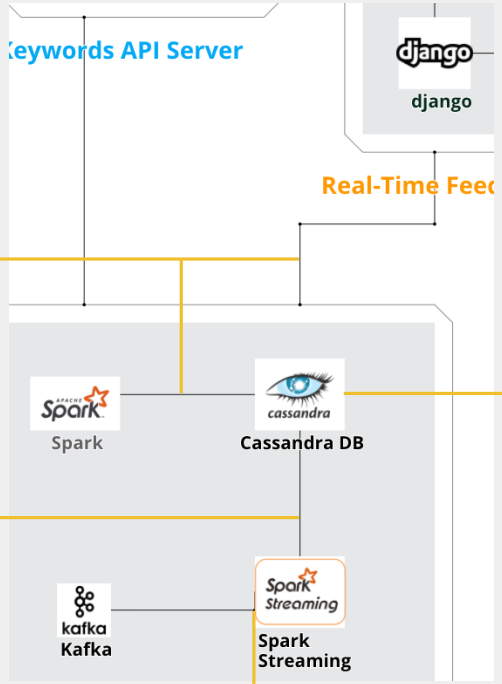
데이터 처리 - Cassandra DB

“ 빠른 데이터 CRUD를 가능하게 하여 실시간성을 확보하고,
장애 발생 시에도 안정적으로 데이터를 공급할 수 있도록 한다. ”

05 **팀원 소개** – 기술 스택 및 R&R

이규은

데이터 처리 - Cassandra DB



데이터 전달

필터링된 데이터를
DB 스키마에 맞게 조정

1. 데이터 모델링
2. 복제된 데이터 유효성 검사
3. 퍼포먼스 향상을 위한 DB 최적화

데이터 형태 변환

05 **팀원 소개** – 기술 스택 및 R&R

이규은

데이터 처리 - Cassandra DB

모든 노드가 동등한 Masterless 구조이다.

로드밸런싱을 통해 장애를 방지하고 성능을 향상시킨다.

Twitter, Facebook, Instagram 등 대용량 데이터를 실시간으로 빠르게 처리해야 하는 서비스에서 많이 이용된다.

복잡한 조건 검색이 불가능하다.

05 **팀원 소개** – 개발 기능 및 역할

이규은

- 대용량 데이터에 대해 빠른 CRUD가 가능한 데이터 모델링
- Spark streaming에서 다루는 데이터의 형태를 구성
- 복제된 데이터의 유효성 관리
- 카산드라 DB 최적화

05 **팀원 소개** – 달성한 점

이규은

대용량 데이터에 대해 빠른 CRUD가 가능한 데이터 모델링

- 해당 트윗의 주제와 생성된 시간을 기준으로 파티션을 나누어 각 파티션이 적절한 사이즈로 고르게 분배될 수 있도록 설계
- 시간 당 20만 건의 데이터까지도 문제 없이 저장할 수 있도록 파티션의 기준 설정

05 **팀원 소개** – 달성한 점

이규은

대용량 데이터에 대해 빠른 CRUD가 가능한 데이터 모델링

- query-driven 데이터 모델링 방식으로 설계
- 데이터 조회 시 데이터 단위의 기준과 가져갈 필드를 고려해 쿼리를 먼저 작성 후 그에 맞게 데이터를 모델링
- 실시간 피드 서버, Spark 모두 데이터가 수집된 주제와 생성된 시간 기준으로 조회하기 때문에 해당 두 필드를 기반으로 파티션 생성

05 **팀원 소개** – 달성한 점

이규은

Spark streaming에서 다루는 데이터의 형태를 구성

1. Twitter API의 구조를 파악해 kafka에서 오는 데이터를 Spark Streaming에서 다루기 쉬운 형태로 찍어내는 틀인 struct 설계

2. Spark Streaming에서 필터링된 데이터를 DB 스키마에 맞게 조정하는 기능 구현

TransformedDstream —1→ Dataframe → Dictionary —2→ Dictionary

05 **팀원 소개** – 달성한 점

이규은

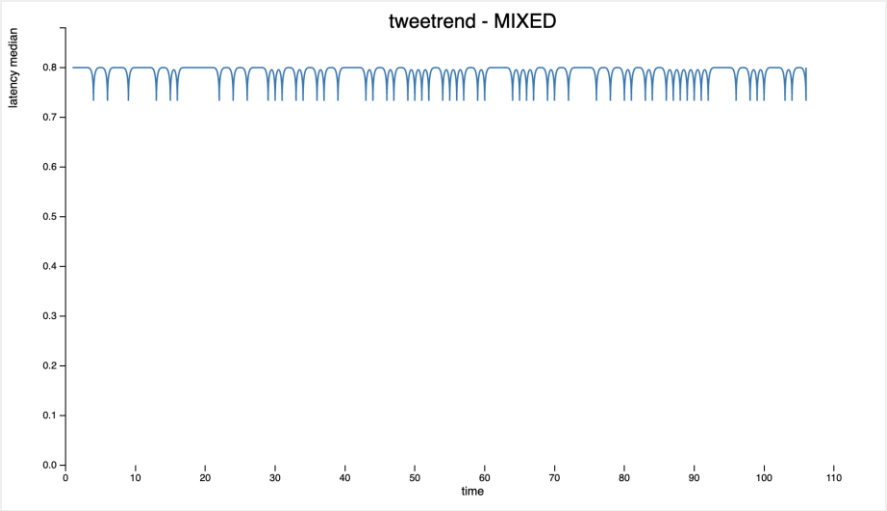
카산드라 DB 최적화 (파라미터 튜닝)

- write 성능 향상: concurrent writes: 32 -> 64
- read latency spike 방지: trickle_fsync: false -> true, trickle_fsync_interval_in_kb: 10240
- failure에 대한 민감도 증가: phi_convict_threshold: 8 -> 10
- 메모리 최적화: table compaction: TimeWindowCompactionStrategy, size:1, unit:DAYS

05 **팀원 소개** – 달성한 점

이규은

카산드라 DB 최적화 (로드 테스팅)



read/write 총 백만 건 발생

-> latency 중앙값: 0.8ms, 평균: 0.8ms

각 operation이 평균적으로 0.8ms에 처리

05 **팀원 소개** – 달성한 점

이규은

카산드라 DB 최적화 (장애 발생 시뮬레이션)

- 무한 루프로 초당 몇 백 건의 read를 오랜 시간 발생시켜 네트워크 과부화 테스트
- 실시간 데이터 저장에 딜레이는 발생했지만 데이터 유실은 발생하지 않았다.

05 **팀원 소개** – 아쉬운 점

이규은

- 트위터의 특성 상 트윗을 한 번 올리면 수정이 불가능 -> 데이터가 처음 저장된 이후 업데이트되지 않아 복제된 각 데이터들이 최신의 상태를 유지하고 있는지 검증할 수 없었다.

05 **팀원 소개** – 소감

이규은

과거에 경험했던 데이터 관련 프로젝트들에서는 이미 파일로 정리된 데이터를 받아 그것을 토대로 분석만을 했을 뿐, 데이터 파이프라인에 대해서는 전혀 고려해보지 않았습니다. 이번 프로젝트를 통해 카프카, 스파크, 카산드라 등 생소했던 기술들을 직접 경험해본 것이 굉장히 좋았습니다.

또한, 좋은 결과물을 위해 모든 팀원이 서로의 진행 상황이나 문제 발생 여부을 수시로 공유하고 협업하는 것이 굉장히 중요하다는 점을 깊게 깨달았습니다.

05 **팀원 소개** – 개인 목표

정석준

백엔드 - API 서버 & MSA 구축

실시간 서비스에 있어, 각 기능을 철저하게 독립적으로 분리하여
안정적인 서비스 환경을 제공할 수 있는 서버를 구축하고자 합니다.

05 **팀원 소개** – 기술 스택 및 R&R

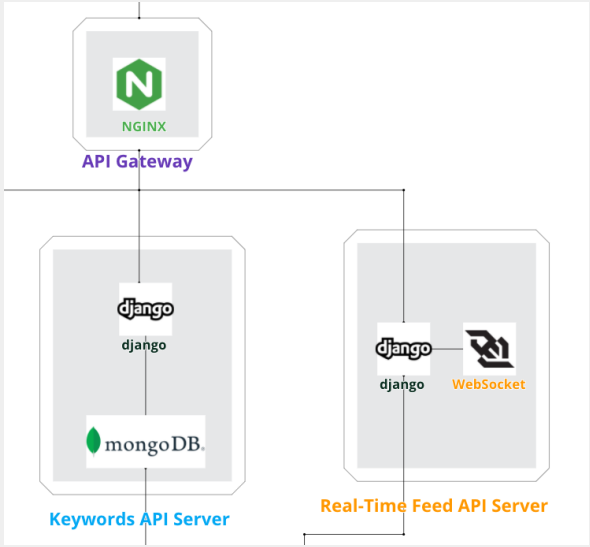
정석준

백엔드 - API 서버 & MSA 구축

API 게이트웨이를 통한 MSA 구조 구축

웹서버와 웹 인터페이스를 통한 연결

기간별 조회에 대한 통계 데이터 제공



실시간 최신 트윗 전송

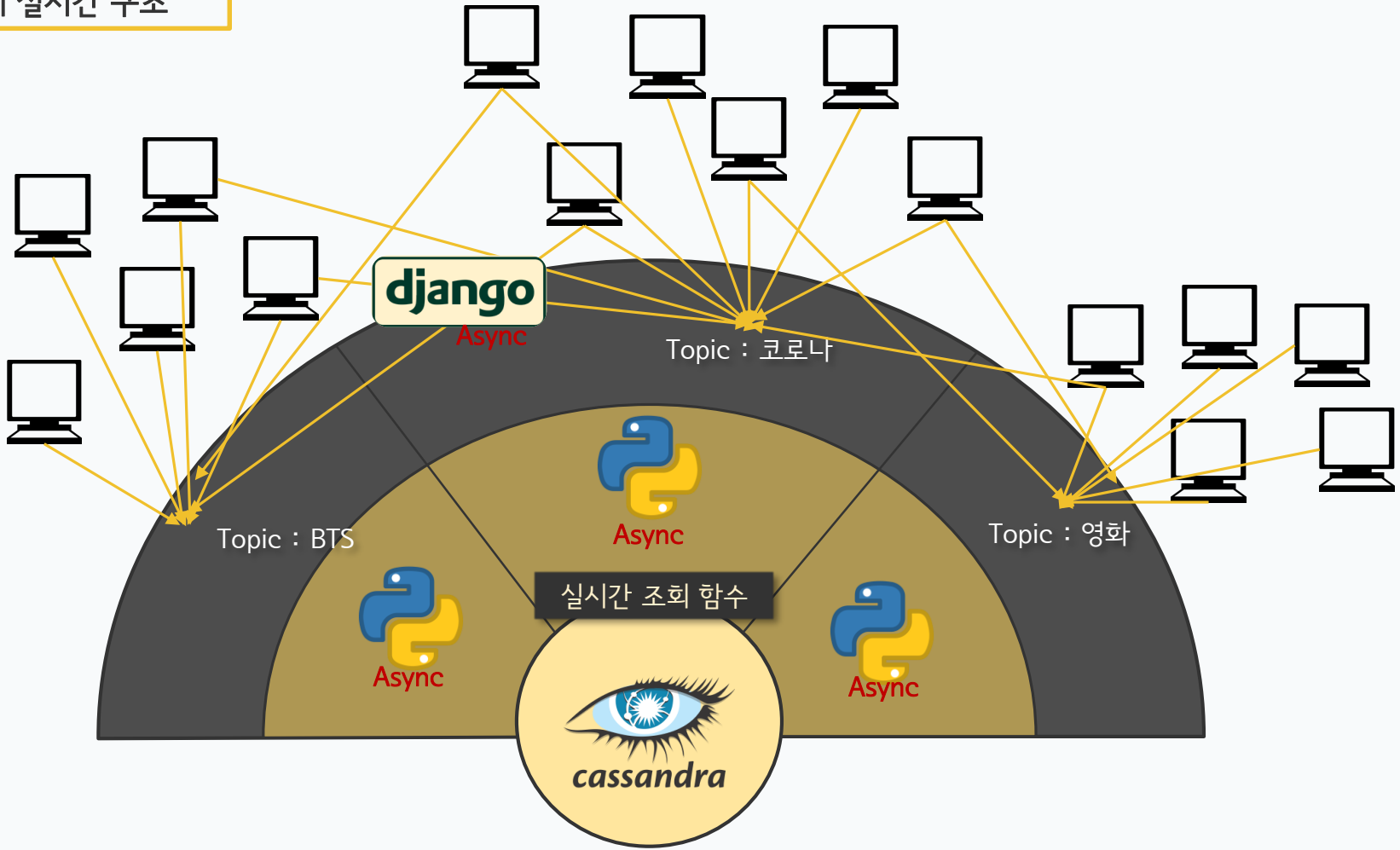
05 **팀원 소개** – 개발 기능 및 역할

1. 트렌드 API 서버 개발
- MongoDB 장점을 이용한 통한 NoSQL 형식 구축
- 데이터 제공 범위에 대한 분리를 위한 비동기 배치 프로그램 구축
- 기간별 조회 데이터 REST API 구현

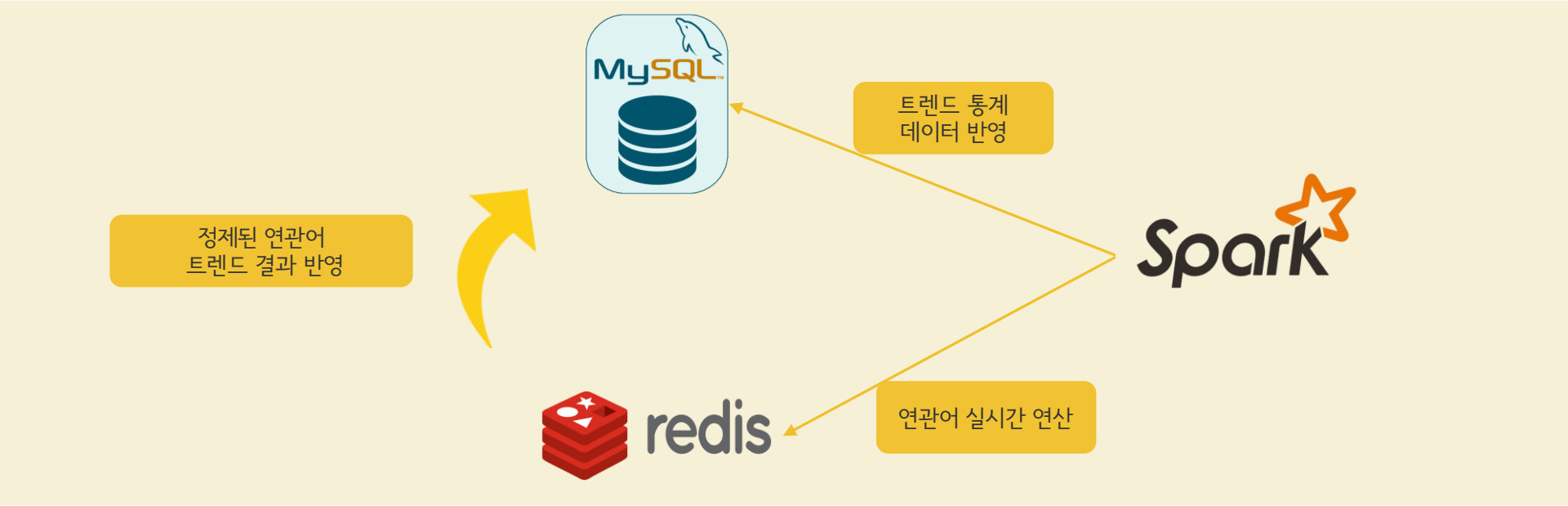
2. 실시간 피드 API 서버 개발
- Django Channels를 사용한 웹소켓 기능 구현
- 파이썬의 asyncio 코루틴을 통한 비동기 기능 구현
- 각 조회에 대해 pub/sub구조 형식 구현 및 비동기 데이터 제공 구현
- 요청과 제공에 대한 기능 분리

3. MSA 구조 구축
- Nginx 웹서버, API 게이트웨이 구축
- uwsgi 인터페이스를 통한 트렌드 서버 연결
- daphne 인터페이스를 통한 asgi 비동기 실시간 서버 연결

비동기 실시간 구조

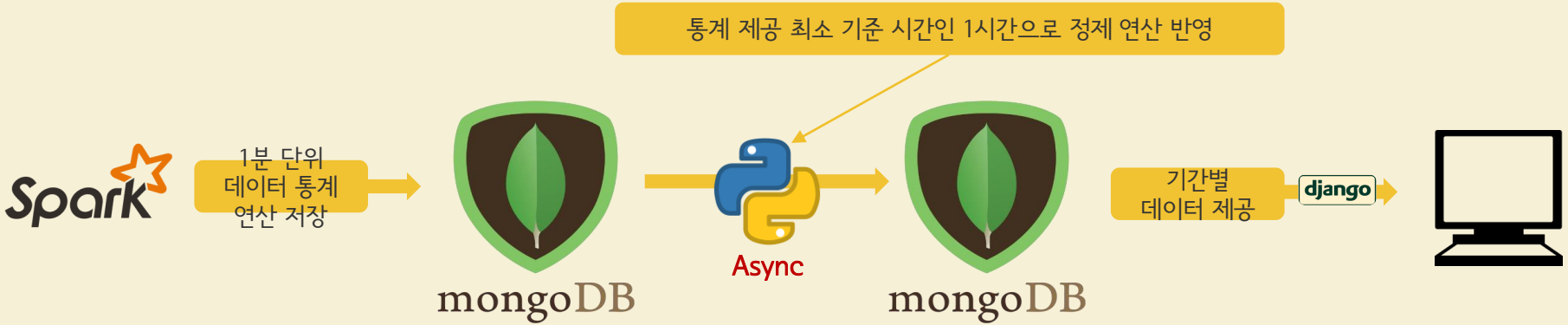

django
Async
Topic : 코로나
Topic : BTS
Async
Topic : 영화
실시간 조회 함수
Async
Async
cassandra

트렌드 데이터 초기 구상안



트렌드 데이터 관리 독립적 구상안

트렌드 데이터 관리 개선 구상안



05 **팀원 소개** – 성공한 점

- 파이썬의 Asyncio, 쓰레딩에 대한 이해를 통해 비동기 실시간 프로그램 구현
- HTTP, 웹소켓 전송에 대한 이해를 통해 서비스별 기능 제공 구현
- 각 내부 함수 기능 분리를 통하여 안정성 증진과 리팩토링 효율화
- 서버 분리 및 API GATEWAY, 웹서버 인터페이스 구축 연결을 통한 전체적인 인프라 구축

05 **팀원 소개** – 부족한 점

- 데이터 특성별 분리를 통해 캐싱을 사용한 효율적인 관리를 하지 못한 점
- 불가피하게 수집되지만 필요없는 데이터를 효과적으로 관리하지 못한 점
- 로드 테스트 및 로그 모니터링 등을 확인하며 전체적인 비교를 해보지 못하고 단순 시간 계산을 통한 일회성 부하 테스트
- 각 DB별 특징의 차이를 실제로 적용하면서 느껴보지 못하고 학습한 대로 사용하며 기능 구현에 그침
- 게이트웨이 구축에 대해 실질적 이점을 활용하기 위한 기능을 시도해보지 못함
- MSA 구조에 무색하게 다른 서비스의 문제와 독립적으로 안정적인 서버 유지를 하지 못하고 문제가 생길 때 마다 직접 반영

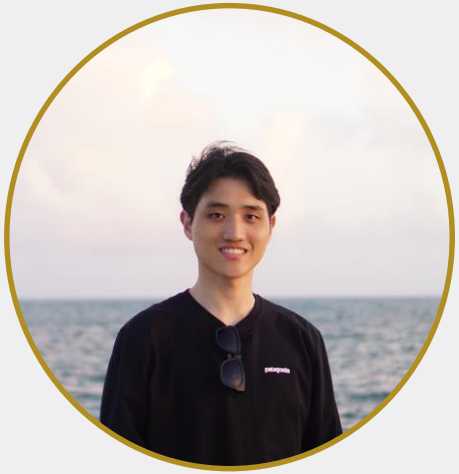
05 **팀원 소개** – 소감

- 하나의 웹 서비스를 완성하기 위한 프로세스를 이해하고 그때그때 필요한 기능을 찾아 구현할 수 있다는 자신감
- 지금 생각하는 이 방식이 괜찮은 것일까에 대한 고민의 연속
  (일정한 기준이 없어서 전부 직접 부딪혀 보며 구현해야 했다)
- 적절한 프레임워크 선택의 필요성
  (장고를 사용했지만 장고의 장점은 전혀 사용하지 못하고 그저 서버를 띄우는 역할만..)
- 손쉬운 CRUD 기능 제공 위주의 서비스에서 장고가 매우 강력하지만 실시간 정보 제공을 위한 서비스에 있어서는 노드JS나 Flask를 사용하는 것이 더욱 효율적일 것

05 **팀원 소개** – 개인 목표

추연호

프론트엔드 개발 및 인증 서버 개발

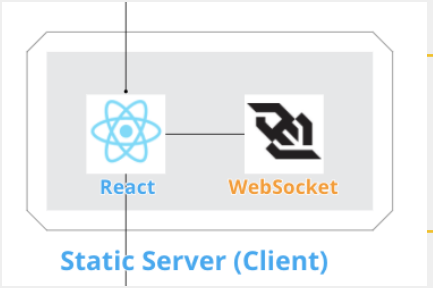


“ 트렌드 대시보드 웹 앱을 개발하며 디자인시스템을 함께 만들어 팀 내 소통 가능하고, 쉽게 확장할 수 있도록 개발하기 ”

05 **팀원 소개** – 기술 스택 및 R&R

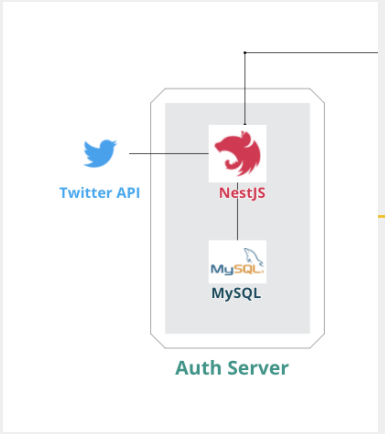
추연호

프론트엔드 개발 및 인증 서버 개발



트렌드 대시보드 형태의 웹 애플리케이션 구현 및 **통계 데이터 시각화**

서버와 웹 소켓 통신으로 **실시간 피드** 구현



트위터 OAuth 소셜 로그인

JWT 토큰 기반 인증 서버

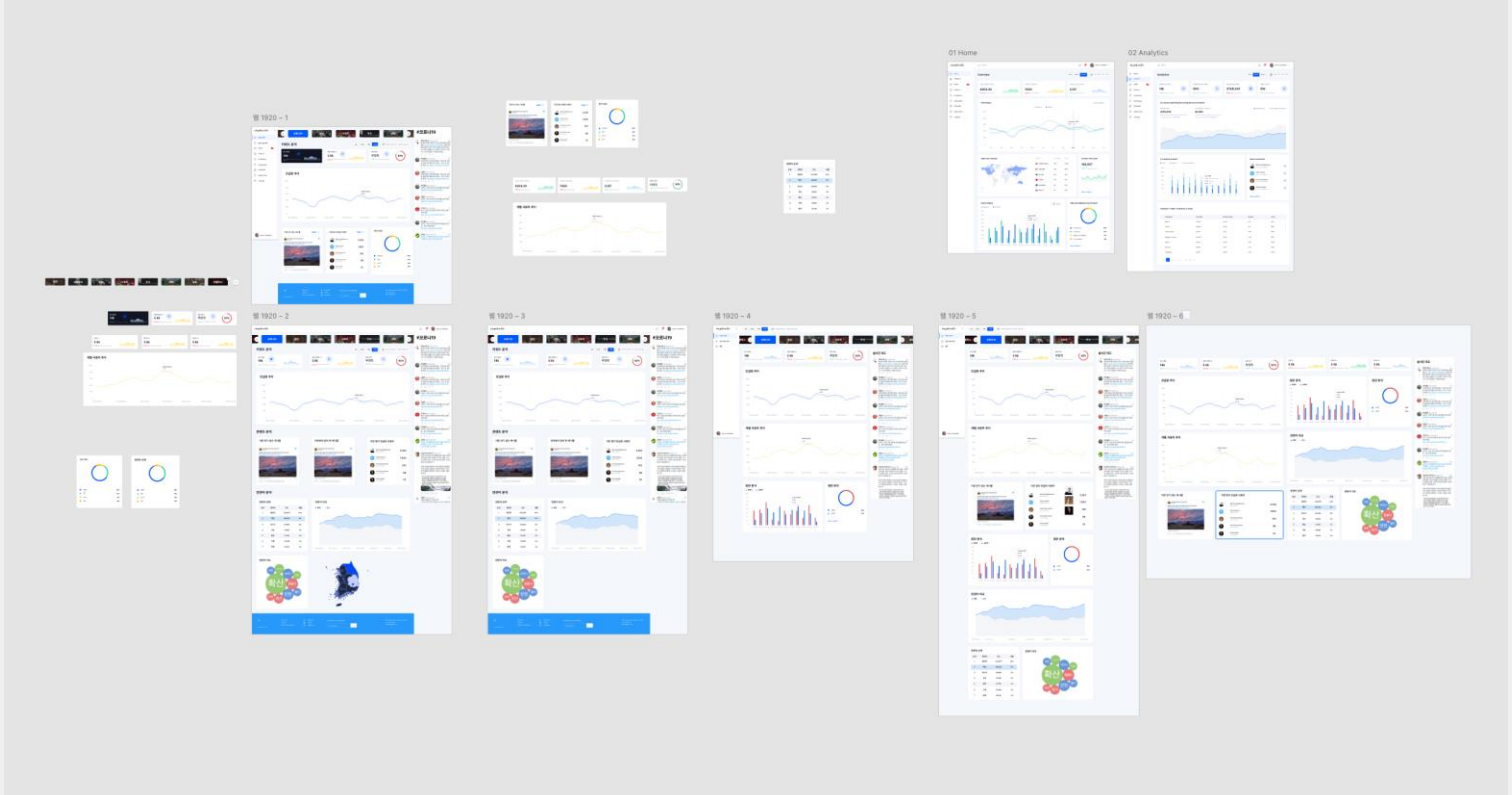
05 **팀원 소개** – 개발 기능 및 역할

추연호

- 대시보드 디자인 프로토타입 제작
- React & TypeScript 로 프론트엔드 개발
- 트위터 실시간 스트림 API를 사용한 Node.js 기반 개발 용 목 서버 개발
- NestJS & TypeScript 로 트위터 소셜 로그인을 사용한 인증 서버 개발
- Storybook을 사용한 디자인 시스템 개발

추연호

컴포넌트 주도 개발 - 1. Adobe XD 프로토타입



추연호

컴포넌트 주도 개발 - 2. Figma를 활용한 디자인 라이브러리 제작

Sizes

12 Almost before we knew it, we had left the ground.

14 Almost before we knew it, we had left the ground.

16 Almost before we knew it, we had left the ground.

20 Almost before we knew it, we had left the ground.

24 Almost before we knew it, we had left the ground.

28 Almost before we knew it, we had left the ground.

32 Almost before we knew it, we had left the ground.

40 Almost before we knew it, we had left the ground.

48 Almost before we knew it, we had left the ground.

Weights

Almost before we knew it, we had left the ground.

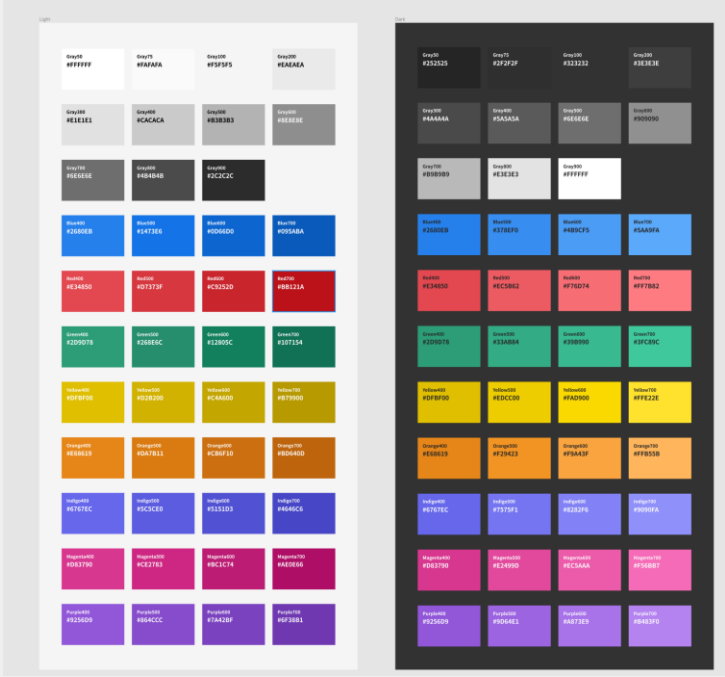
Almost before we knew it, we had left the ground.

Almost before we knew it, we had left the ground.

Almost before we knew it, we had left the ground.

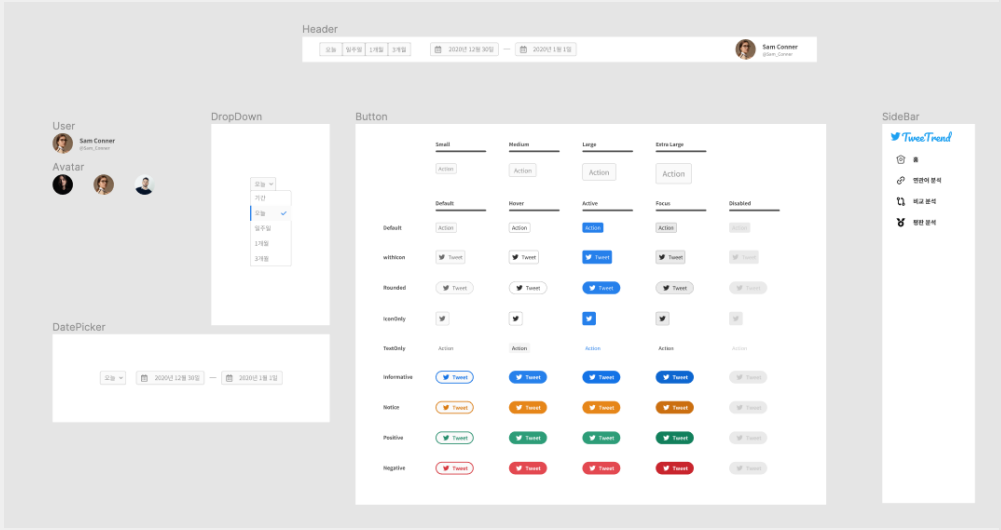
Almost before we knew it, we had left the ground.

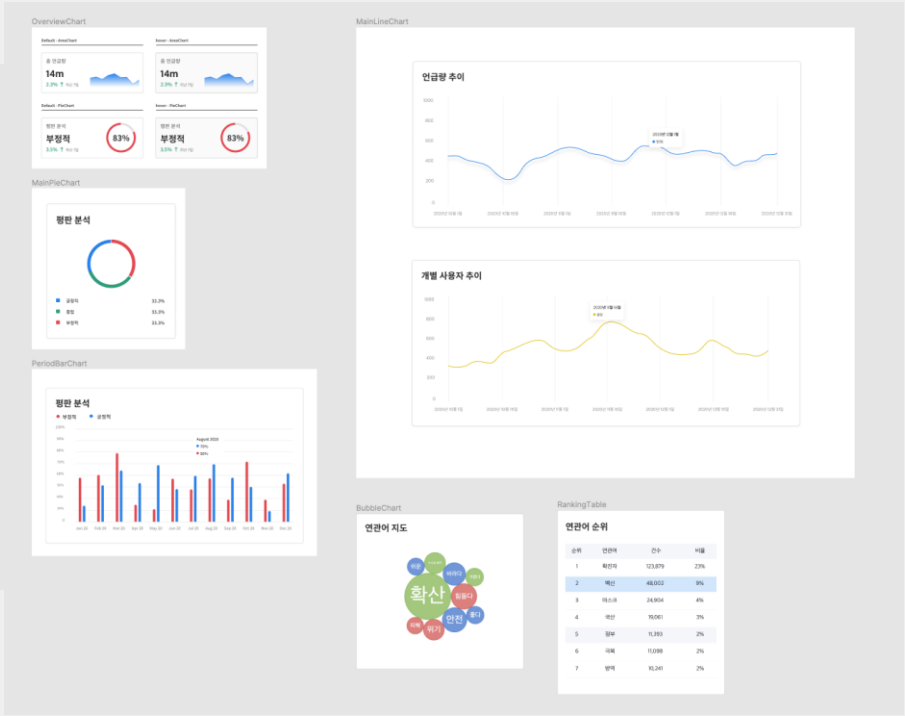
Almost before we knew it, we had left the ground.



추연호

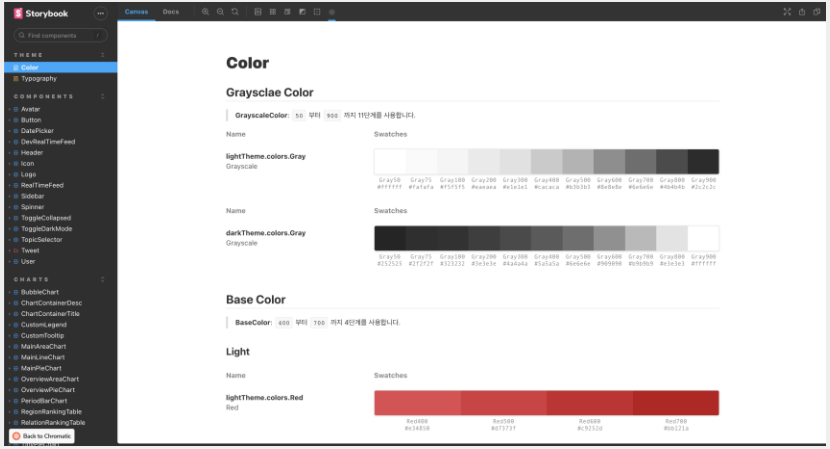
컴포넌트 주도 개발 - 2. Figma를 활용한 디자인 라이브러리 제작



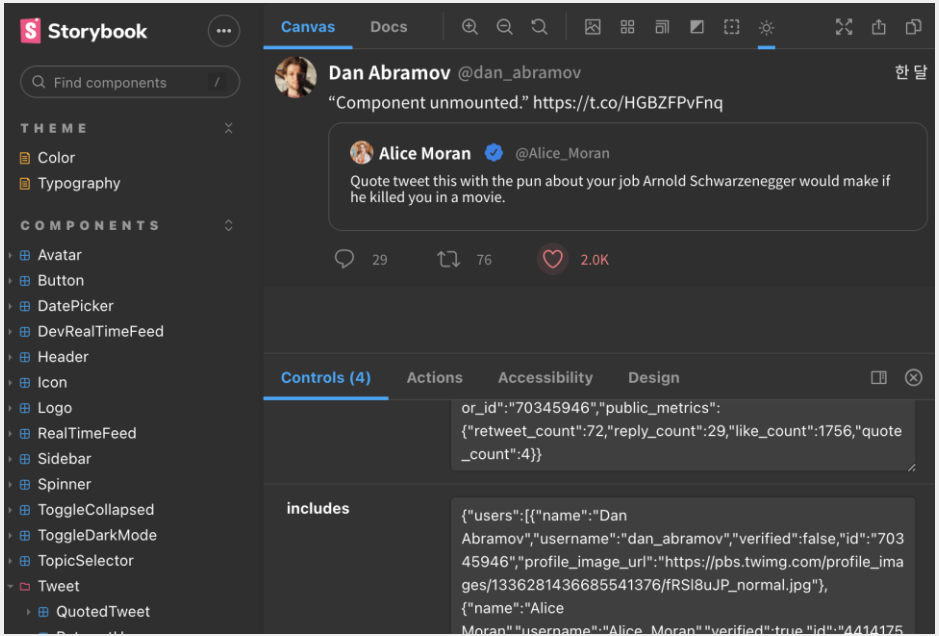


추연호

컴포넌트 주도 개발 - 3. React & TypeScript + Storybook



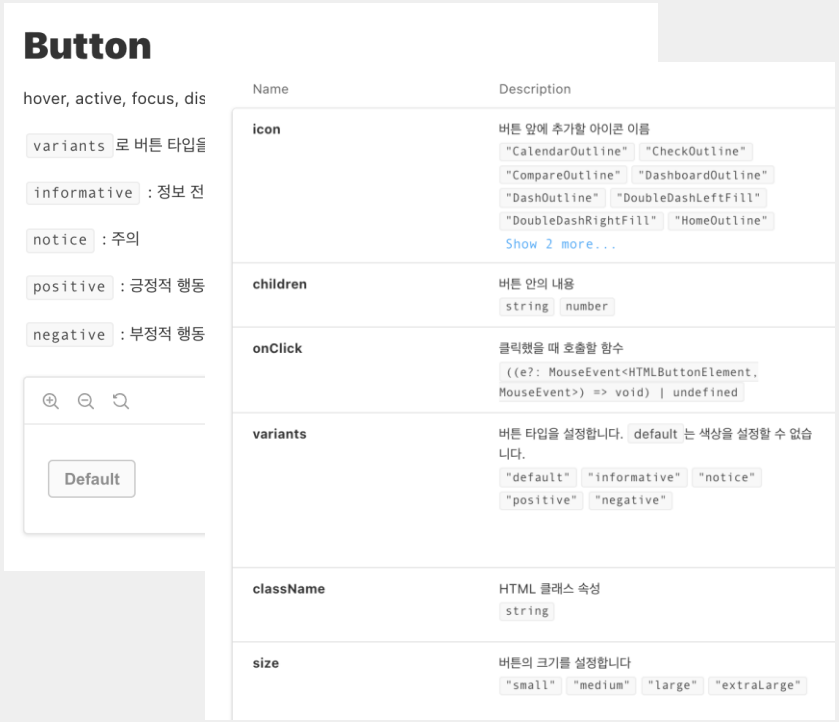
컬러 및 폰트 등 테마 문서화

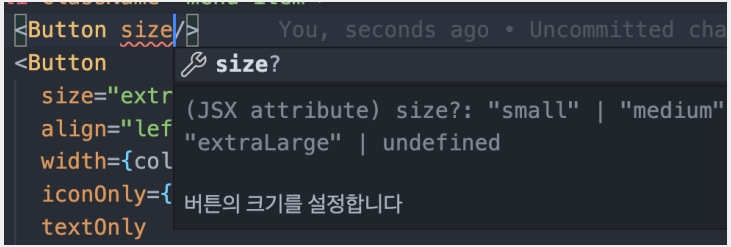


컴포넌트 별 개발 환경

추연호

컴포넌트 주도 개발 - 3. React & TypeScript + Storybook

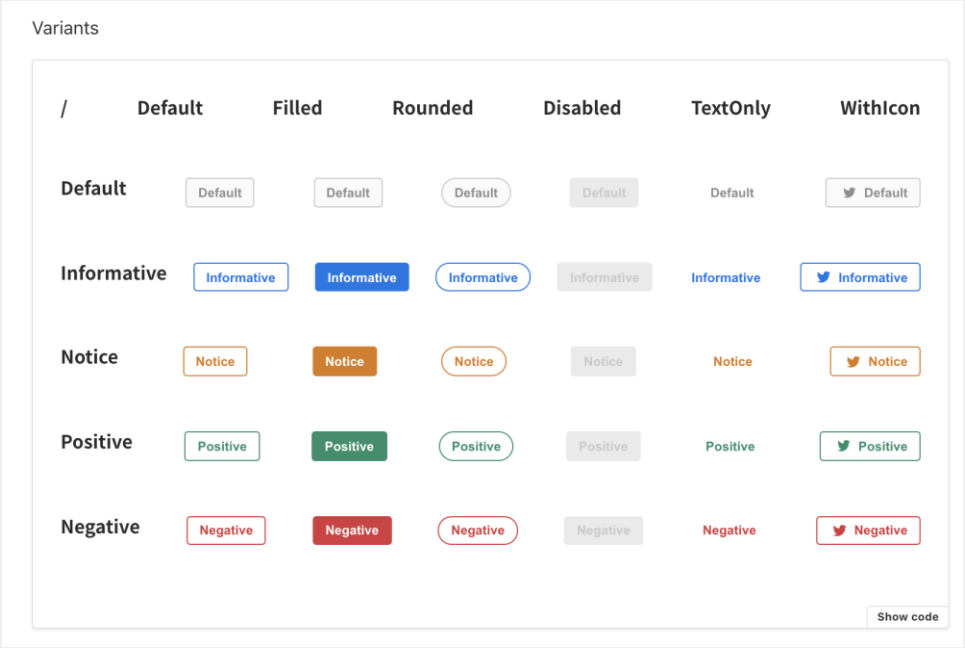




따로 문서화 도구를 사용하지 않고,
코드의 주석을 통해 컴포넌트의 사용법을
파악할 수 있고, 에디터의 도움을 받을 수 있음.

추연호

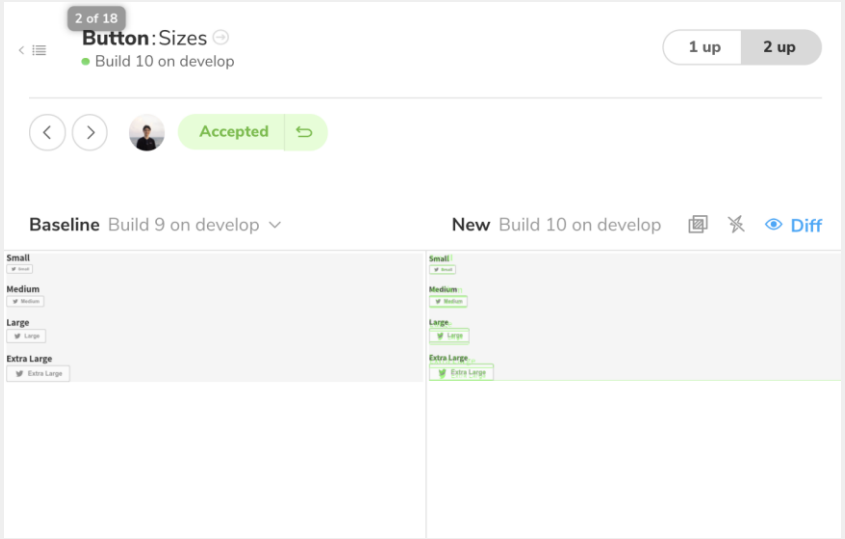
컴포넌트 주도 개발 - 3. React & TypeScript + Storybook



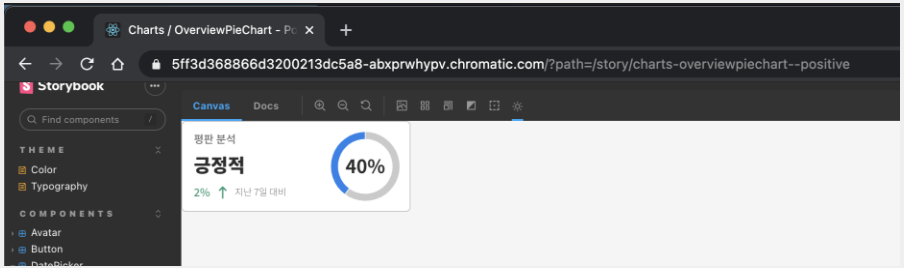
컴포넌트의 다양한 상태를 디자인 레벨에서 코드 레벨로 가져와 사용할 수 있음

추연호

컴포넌트 주도 개발 - 4. Chromatic으로 비주얼 테스팅 및 배포 환경 구축



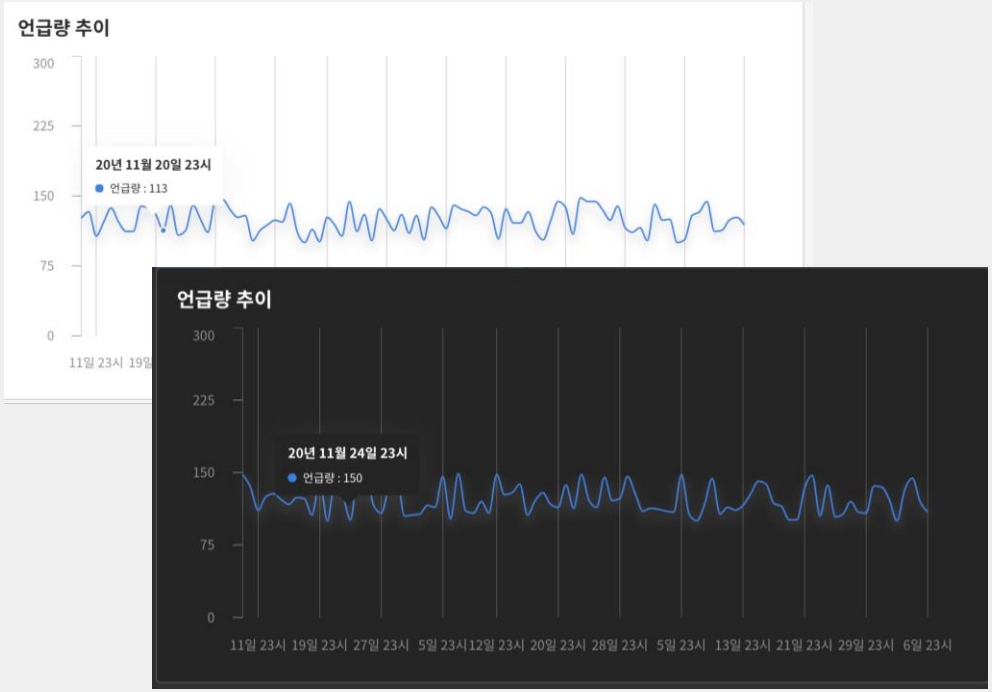
비주얼 테스팅 환경



Public으로 배포되는 디자인 시스템

추연호

컴포넌트 주도 개발 - 5. Storybook + Recharts 라이브러리



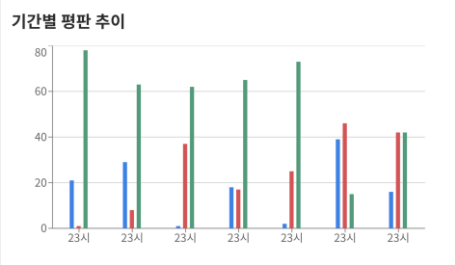
초기 디자인 프로토 타입과 일치한 결과물

+

외부 라이브러리를 사용하지만 서비스와 일관된 테마를 적용할 수 있도록 커스텀 (다크 모드 적용)

추연호

컴포넌트 주도 개발 - 5. Storybook + Recharts 라이브러리



dataKeys*

["positive","negative","neutral"]

data*

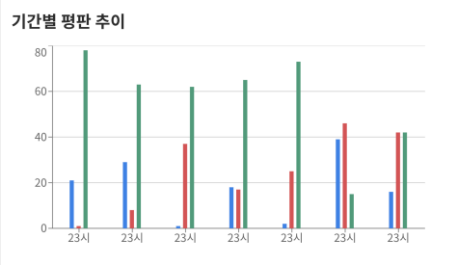
[{"date":1612104608484,"positive":21,"neutral":78,"negative":1},{"date":1612191008484,"pos
{"date":1612277408484,"positive":1,"neutral":62,"negative":37},{"date":1612363808484,"pos
{"date":1612450208484,"positive":2,"neutral":73,"negative":25},{"date":1612536608484,"po
{"date":1612623008484,"positive":16,"neutral":42,"negative":42}]

```
PASS src/utils/__tests__/mock.spec.ts
  기준 날짜에 대한 이전 날짜 객체 반환
    ✓ 오늘 기준 3일 전 (1 ms)
  기준 날짜의 시간에 대한 이전 시간 날짜 객체 반환
    ✓ 현재 기준 3시간 전 (2 ms)
  genRandomInt()
    ✓ min 이상 max 이하의 값을 반환한다. (1 ms)
  목 데이터 생성 테스트
    ✓ 5개의 가짜 데이터를 생성한다.
    ✓ retweet 키를 가진다. (1 ms)
    ✓ quoted 키를 가진다.
    ✓ 샘플 값은 0부터 100이하의 값을 가진다. (1 ms)
```

데이터 키, 시간, 데이터 범위에 따라 목 데이터를 생성하는 유틸 함수 제작

추연호

컴포넌트 주도 개발 - 5. Storybook + Recharts 라이브러리



dataKeys*

["positive","negative","neutral"]

data*

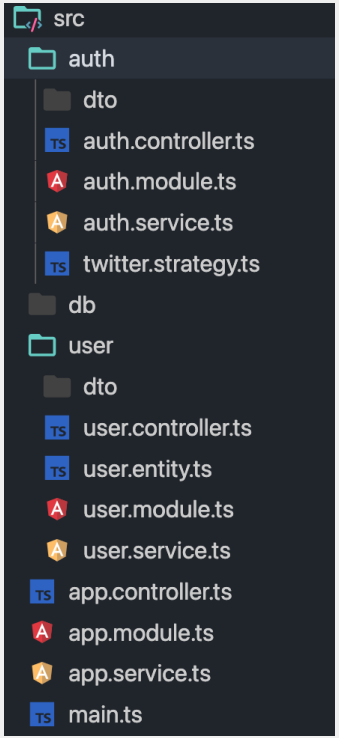
[{"date":1612104608484,"positive":21,"neutral":78,"negative":1},{"date":1612191008484,"pos
{"date":1612277408484,"positive":1,"neutral":62,"negative":37},{"date":1612363808484,"pos
{"date":1612450208484,"positive":2,"neutral":73,"negative":25},{"date":1612536608484,"po
{"date":1612623008484,"positive":16,"neutral":42,"negative":42}]

```
PASS src/utils/__tests__/mock.spec.ts
  기준 날짜에 대한 이전 날짜 객체 반환
    ✓ 오늘 기준 3일 전 (1 ms)
  기준 날짜의 시간에 대한 이전 시간 날짜 객체 반환
    ✓ 현재 기준 3시간 전 (2 ms)
  genRandomInt()
    ✓ min 이상 max 이하의 값을 반환한다. (1 ms)
  목 데이터 생성 테스트
    ✓ 5개의 가짜 데이터를 생성한다.
    ✓ retweet 키를 가진다. (1 ms)
    ✓ quoted 키를 가진다.
    ✓ 샘플 값은 0부터 100이하의 값을 가진다. (1 ms)
```

데이터 키, 시간, 데이터 범위에 따라 목 데이터를 생성하는 유틸 함수 제작

추연호

인증 서버 개발 - NestJS + TypeORM + JWT + 트위터 소셜 로그인




```typescript
@Entity()
export class User extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  provider_id: string;

  @Column({
    nullable: true,
  })
  @IsEmail()
  email: string;

  @Column()
  @IsString()
  name: string;

  @Column()
  @IsString()
  username: string;
```

Express가 해결해주지 않는 구조적인 고민을
NestJS가 쉽게 해결해 줌

+

강력한 CLI로 편리한 개발환경 제공

+

TypeScript, TypeORM와 함께 사용하면서
타입의 안정성과 데이터 validation도
적은 코드로 가능했음

05 **팀원 소개** – 달성한 점

추연호

- 모든 컴포넌트를 디자인부터 개발까지 일관된 워크플로우로 개발할 수 있었음.
- 일관된 워크플로우를 만들면서 업무 효율성을 높일 수 있었고, 일관된 컴포넌트를 쉽게 리팩토링할 수 있었음.
- 프론트엔드 개발에서 TypeScript의 장점을 느낄 수 있었음.
- NestJS와 TypeORM 등 새롭게 노드 환경에서 백엔드 프레임워크를 경험할 수 있었음.

05 **팀원 소개** – 아쉬운 점

추연호

- 팀의 주제인 데이터 처리 + 분석을 한꺼번에 다루는 서비스 주제를 기획하는데 어려움이 있었음.
- 기획을 하면서 개발을 하다보니, 전체 레이아웃보다 컴포넌트를 먼저 만드는 개발 방법이 주는 단점도 있었음
- 컴포넌트 개발 시에는 상태를 외부에서 주입 받는 것으로 생각하고 개발하다가, 마감 기간이 촉박해오자 페이지 단위에서 모든 컴포넌트의 상태를 한번에 관리하게 되었음.
- 팀 내에서 전체 페이지를 명확히 기획하지 못해서 개발 중간중간마다 수정하거나 팀끼리 혼란이 있는 경우가 있었음.
- 인증 서버에 사용한 NestJS의 구조에 대한 부족한 이해.

05 **팀원 소개** – 소감

추연호

- 많은 양의 데이터를 렌더링하면서 성능적인 최적화도 경험해보고 싶었지만, 데이터가 준비되는데 시간이 부족했음.
- 팀 내에서 데이터 처리 부분에 관여하지 않고 혼자 개발하면서, 스스로 개발 일정을 관리하는 방법을 배울 수 있었음.
- 목표했던 새롭게 접한 기술에 대한 블로깅을 개발 기간도 맞추면서 하기가 쉽지 않다는 것을 다시 한번 느낌.
- 팀원과 메인 기술 스택이 맞지 않다보니 서로 기술 공유할 수 있는 부분이 없었던 것에 대한 아쉬움

06 마무리