

조정민 Software개발자

명지대 정보통신공학과 / 26.08 졸업예정

Email. c4851007@gmail.com
GitHub. <https://github.com/JOJoungMin>
Moblie. 010-2932-5286

React, TypScript/ Python, Flask 기반의 신입 Software 개발자로,
사내 제품의 테스트 커버리지를 높이기 위한 API 서버의 TC 제작등의 개발을 해왔습니다.
검증되지 않은 코드는 무용지물 이라는 생각과 확장성 있는 구조 기획이 레거시 코드 생산을 막는다는 생각으로
DX를 지키는 개발을 하고있습니다.

Internship

Secui ui/UX그룹 25.11.03 - 26.02.27

장비 NGF의 테스트 커버리지를 높이기 위해 API 서버의 TestCase 개발을 진행했습니다.

| BackEnd REST API TC 개발

회사 제품인 NGF의 Test Coverage를 높이고, API의 누락/버그를 수색 검증했습니다.
이미 존재하는 1300개의 TC와 충돌하지 않고, 빌드 과정에서 성능에 문제가 없도록 신경썼습니다.

향후 개발을 이어나가는데에 지장이 없도록 Code Conventions을 지켰으며, 매우 낮은 확률의 에러를 잡아내기 위해 최적화를 공리했습니다.

Python, Flask, pytest, Jenkins

- 랜덤 데이터 기반 API 시나리오 테스트
 - 페이지 기반으로 POST 데이터를 랜덤으로 생성하기에 극단값으로 인한 오류 가능성 존재
 - Random 함수를 사용하기에 매우 낮은 확률이지만 수많은 반복에 의해 Edge Case 발생 가능성 존재
 - 성공 case의 데이터 객체 id를 해당 페이지 setup_class에 저장하며 에러 가능성을 제거
- 알고리즘 기반 API 시나리오 테스트
 - POST로 보내져야 하는 값이 너무 큰 경우 랜덤을 통해 생성할 수 없음 ex) 202601011200202602071222 → 26년1월11일12시0분 - 26년2월7일12시22분
 - 반복을 최소화하며 datetime 객체를 활용, 전체 모든 값이 예외없이 등장할 수 있도록 하는데에 구현을 집중

프로젝트

| Lyric Type 2인팀 / 25.12.01 - 진행중

웹 타이핑 프로젝트 역할/ 백엔드, 배포

단발성으로 끝나는 프로젝트로 남지 않기위해 구조와 확장성/ 검증 개발 원칙을 지키며 REST API 서버 제작 역량을 기르고자 노력했습니다.
프론트 개발자와 자주 볼수 없고 연락을 유지할 수 없기에 명세서를 작성하며 개발했습니다.
8시간 단위 자동 빌드 → 테스트 / 부하 검증 → 서버 배포 → 관리자 페이지 업데이트 순으로 GitHub Action CI/CD를 구축했습니다.
프론트 개발자가 최적화를 하기 편하도록 UI에 맞는 API를 신경썼습니다.

Python, Flask, pytest, locust, GitHub Action, EC2, RDS, S3

- 모듈형 API 서버 아키텍처 설계 및 구축
 - Blueprint를 활용 api 경로를 초기화해 프로트측에서 usePrms로 활용
 - 풀더 구조에 따른 API 명세서 제작을 위해 Swagger를 활용
- CI/CD 파이프라인 및 배포 프로세스 자동화
 - pytest/locust를 활용 4시간 단위 자동 빌드 및 테스트 결과 upload
- Redis / 비동기 처리
 - RPS 20 / 초당 평균 응답률 2.6회 환경에서 응답 시간 500ms 돌파 후 Redis 도입
 - 응답시간 500ms 에서 평균 200ms로 감소
- 통합 테스트 및 서버 성능 관리 모니터링 페이지 개발
 - RPS 에 따른 응답 시간 평균값 / unit test 결과 / 프론트 LCP등의 성능 값을 관리중



Experience

크래프톤에서 진행한 부트캠프에서 주 100시간 단위의 커리큘럼을 이수했습니다.

| BootCamp

- 크래프톤 정글 8기 2025.03 - 2025- 08

| 알고리즘, 어셈블리, C, 운영체제 프로젝트등의 코어 기술 습득

- 초소형 OS 구현 프로젝트
- 정렬/DFS/BFS/DP/이진트리/RB트리 등의 알고리즘 공부
- 매주 진행되는 전인원 코드리뷰
- 주 100시간 단위의 학습시간

| 6인 웹 개발 프로젝트 발표영상

6주간의 개발 기간동안 많은 소통을 하며 개발

- 팀장 / 데이터 전처리 역할을 수행
- Whshper를 활용해 영상 데이터를 텍스트로 변환
- 협업을 위한 FSD 구조 선택

Other Experience

바둑 프로기사 입단을 위해 교내 프로기사 준비반 활동을 했습니다.

| 프로기사 입단 준비

- 바둑고등학교 상비군 2016.03 - 2017.12

| 암기, 문제풀이, 복기등의 강도높은 바둑 교육 이수

- 매주 진행되는 승강 리그전
- 주 6회 야간 학습
- 기보 암기 검사 / 사활문제 풀이
- 상비군 전체를 대상으로 한 패배 대국 리뷰
- 매주 주말 진행되는 지역 바둑 연구생 리그전
- 2014 대전 시장배 중등부 우승

조정민 / NOTA

c4851007@gmail.com

010-2932-5286

소개

네트워크 보안회사 Secui에서 커리어를 시작했으며 백엔드 REST API TC를 짜며 API 안정성과 TDD 개발 철학에 대해 익혔습니다.

처음 회사에 들어갔을 당시에는 테스트 코드에 대해서 아무것도 몰랐으나 “해야하는 일이라면 어떻게든 구현한다” 라는 마음가짐으로 직무를 시작했습니다. 이어서 단순히 구현을 넘어 이후에 이 코드를 넘겨받을 개발자를 생각해 convention을 공리하고 확장성을 고민하며 성장했습니다. 특히 인턴십 과정 기준에 존재하는 1,300개가 넘는 REST API 테스트 코드의 컨벤션과 데이터의 연동성을 파악해 업무중 개발한 TC가 다른 프로그램의 영향을 최소화 되도록 연구하며 300개가 넘는 TC를 개발하여 테스트 커버리지를 끌어올린 경험이 있습니다.

이러한 경험을 통해, "안정적인 백엔드가 서비스의 신뢰도를 결정한다" 는 신념을 갖게 되었습니다. 이러한 경험은 단순히 기능을 만드는 개발자를 넘어, 시스템의 빈틈을 찾아내고 견고하게 다지는 엔지니어로 저를 성장시켰습니다.

현재는 부트캠프 동료와 함께 “타이핑 웹 프로젝트”를 사이드 프로젝트로 개발하고 있습니다. 확장성을 고려해 테스트 코드부터 작성하는 TDD 철학을 도입해 개발하고 있으며 pytest와 부하 테스트를 통해 서버가 언제 한계 지점에 이르는 지 측정하며 좋아보여서 넣는 성능 개선이 아닌 실제로 필요한 위치에 필요한 기술을 도입하며 개발을 진행하고 있습니다..

강점

- 인턴십 경험을 통해 REST API에 대한 개발과 지체없는 협업을 진행할 수 있습니다.
- CI/CD와 더불어 테스트 자동화를 구현 / 운영 해왔기에 곧장 실무에 들어설 수 있습니다.
- 부트캠프 크래프톤 정글을 이수하며 쌓은 CS지식과 6인 팀프로젝트의 팀장 경험이 있습니다. 이를 통해 회사에 지시에서 원하는 바가 무엇인지 빠르게 식별 할 수 있는 능력을 갖추었습니다.

NOTA 지원 동기

- 근본적인 문제 해결
노타가 지향하는 "제조 현장의 영상 분석 솔루션"은 기술이 실제 세상에 기여하는 가장 직접적인 방식이라고 생각합니다. 공정 단계에서 생기는 문제와 개선 점을 영상이라는 하나의 데이터 만으로 솔루션을 만들어 낸다면 반드시 폭발적인 생산력 증진을 이뤄낼 거라 생각합니다. 어려운 점은 영상 데이터에서 “어떻게 알맞은 솔루션을 판별하느냐?” 라고 생각합니다. 크래프톤 부트캠프에서 OS 프로젝트와 방대한 코드들 사이에서 최적화를 고려하며 개발 해본 저의 경험은 노타의 비전과 시너지를 낼 수 있다고 확신합니다.
- 안정성과 확장성을 동시에 고려하는 개발
노타는 AI로 다양한 산업 전반에 발을 넓히고 있습니다. 해당 과정에서 가장 중요한 것은 안정성, 확장성이라고 생각합니다. 특정 제품이 잘못된 데이터를 내놓는다면 이는 사람이 실수를 한 것보다 더 엄청난 상품성 하락이 생길 것이라 생각합니다. 제가 그간의 인턴십과 프로젝트에서 갈고닦은 테스트 자동화 및 모니터링 역량 노타의 MVP가 시장에서 신뢰를 얻는 데 반드시 기여할 것입니다.
- 업무 적합성
cursor나 Claude Code 등 최신 AI 도구를 활용해 개발 효율을 극대화하는 노타의 업무 방식에 깊이 공감합니다. 저는 이미 AI 도구를 활용해 복잡한 로직을 구현하고 검증하는 방식에 익숙하며, 이를 통해 노타의 개발 속도를 가속화할 준비가 되어 있습니다. 또한 CI/CD 파이프라인에서 부하 테스트 데이터를 수집하며, 초당 트랜잭션 처리량인 RPS(Requests Per Second) 3.0 이상으로 유지 및 latency를 평균값과 더불어 상위 95퍼 즉 극단값을 제외한 수치값으로 최대 값 또한 300ms를 넘지 않도록 유지한 저의 개발 경험은 노타의 해당 직무에 적합하다고 생각합니다.

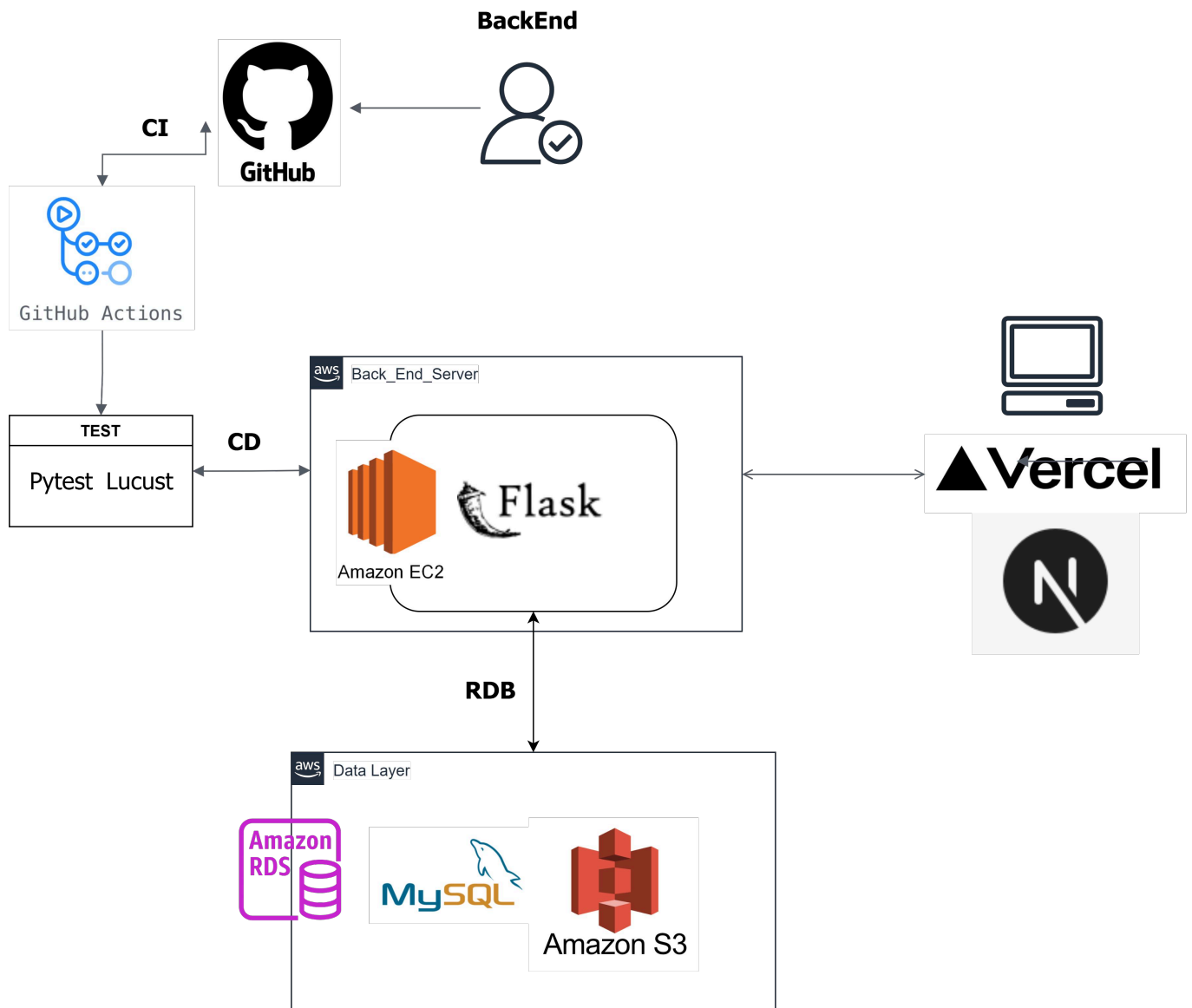
포트폴리오

Typring-somting (2인 프로젝트)

한컴 타자를 참고한, 가사, 명언, 대사등을 따라쳐보는 웹 프로젝트입니다.

<div>서비스</div> <div><ul style="list-style-type: none">서비스 바로 가기 (typing_something)소스코드 확인하기 (Github)API 명세서 확인하기 (Swagger)</div>	<div>작업기간</div> <div><ul style="list-style-type: none">26.01 (1.0 완료 / 운영중)25.12 - 26.01</div>
<div>구성원</div> <div><ul style="list-style-type: none">BE & Infra (조정민)FE & (권민성)</div>	<div>Back End 사용 기술</div> <div><ul style="list-style-type: none">FlaskMySQL / SQLiteGoogle OAuthFlasggerAWS ec2/RDS/S3Pytest / LucustGithub_Action</div>
<div>역할</div> <div><div>Admin</div><div><ul style="list-style-type: none">관리자 페이지 제작 (Typescript)서버 성능 측정</div><div>BeckEnd</div><div><ul style="list-style-type: none">Flask 앱 팩토리 패턴 구현Blueprint 기반 모듈화 라우팅 구조 설계데이터 베이스 모델 설계데이터 베이스 마이그레이션 시스템 활용Next_auth 활용 검증 로그인 API 제작REST API 제작텍스트 CRUD API 제작Swagger API 문서 자동 생성단위 테스트 작성 (pytest)부하 테스트 작성 (Lucust)API 성능 모니터링 (p95, p99, Latency 추적)에러 핸들링 및 응답 표준화</div><div>Infra</div><div><ul style="list-style-type: none">AWS ec2 / RDS / S3 연동Github Action CI/CD 구현테스트 → 빌드 → 부하체크 → Admin 페이지 파이프 라인 구현</div></div>	<div>프로젝트 회고</div> <div><ul style="list-style-type: none">마주치지 않고 메신저와 명세서를 통한 협업 개발 경험했음명세서를 확실히 작성해야 생산성에 처짐이 없기에 REST FUL API에 대한 심도 높은 경험을 해봄배포전 최소 구현(MVP) 개발에 대한 고민확장성을 위한 구조 개발에 대한 고민테스트 → 부하→ 배포 과정에서 이전 버전의 코드에 부하테스트를 하는 논리적 오류를 식별 / 스테이징 서버에 필요성을 인식TC를 개발하고 기능을 만드는 TDD개발의 장점을 식별 협업의 난이도가 높은 상황에서 최소한의 에러를 제압하는 경험부하 테스트를 진행할 때 실제 서버 성능이 흔들리는 환경에 대한 대응방한 미비프론트 성능 데이터와 병합 진행중Notion등을 활용한 문서 공유 작업 필요성 식별</div>

아키텍처



아키텍처 구현

• 요구사항:

검증된 코드만 EC2에 배포되어야 함
이미지는 용량이 크므로 S3를 활용

• 작업:

Github Action을 활용한 CI/CD 구현
TEST 코드를 거치면서 성능 데이터 양산
프론트 사이트에 요청으로만 데이터의 요청과 PUSH가 날아가도록 설정

• 문제:

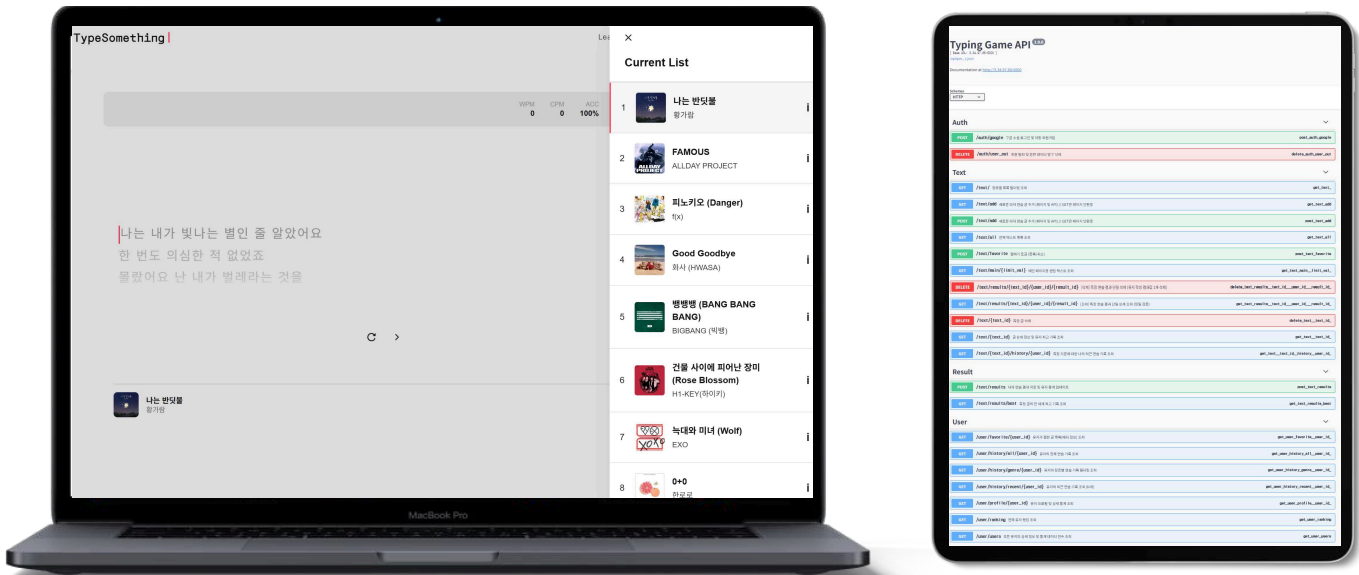
Pytest와 부하테스트가 TC를 처리하던 도중 실제 DB에 간섭하는 현상 발생
단위 테스트가 실패한 심각한 오류 케이스에서도 빌드가 계속되는 오류

• 문제 해결:

Pytest는 SQLITE를 활용 Github Action 내부 메모리를 활용하여 실제 db 간섭을 제거. 부하테스트를 테스트 유저를 생산 활용 후 데이터를 즉시 제거하는 로직으로 오류를 회피함
유닛 테스트가 실패시 빌드를 취소하는 revert 명령어를 실행하도록 변환

REST API 구현

프론트 페이지와 구조를 고려한 API



리소스 기반 URL 설계

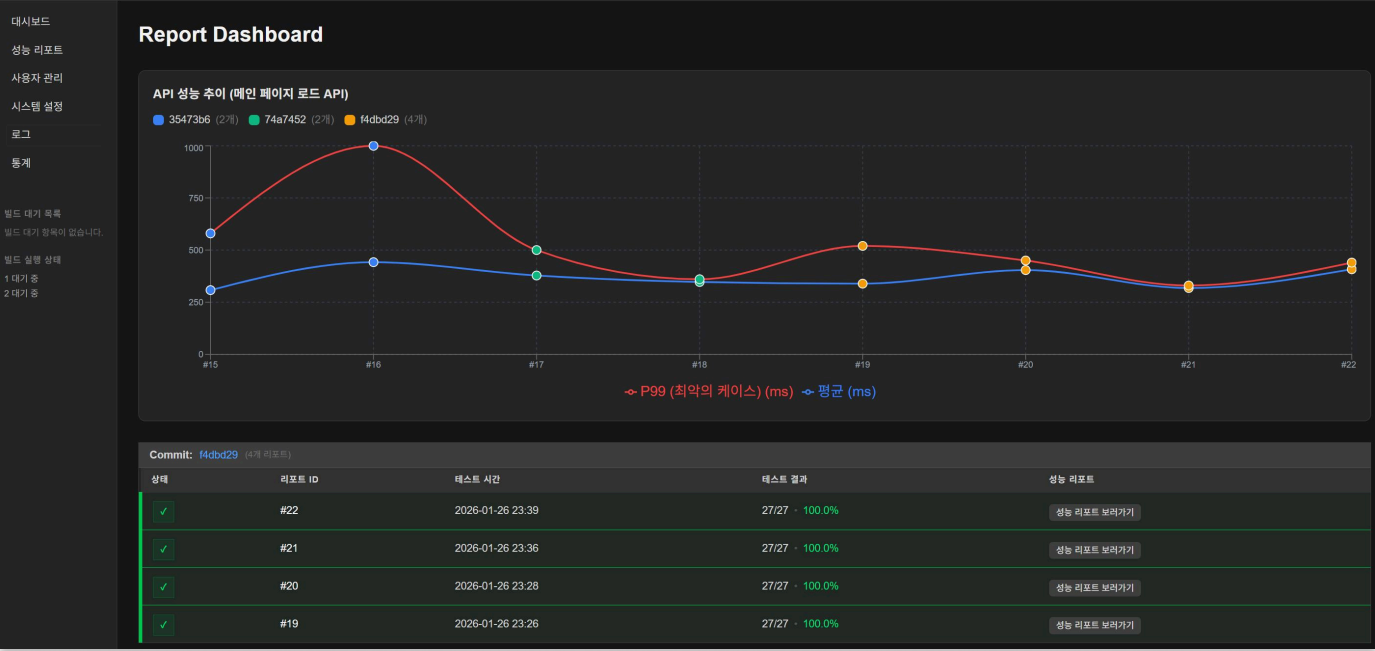
- **요구사항:**
직관적이고 예측 가능한 API 구조 제공
URL만 봐도 어떤 리소스를 다루는지 명확히 파악 가능하게 구현
- **작업:**
리소스 중심의 URL 패턴 설계
HTTP 메서드로 동작 구분 (GET: 조회, POST: 생성, DELETE: 삭제)
- **문제:**
메인 페이지에서 활용되는 `text/main/{limit_val}` 함수는 요구한 int 값 만큼 글을 보내주는 과정에서 로직상 **랜덤하게 같은 글이 날아갈 확률이 있음**
- **문제 해결 → 빠른 해결:**
백엔드 로직상 랜덤을 결정짓는 **cid값을 현재 시간값을 활용하여 반환** [문제는 여전히 낮은 확률의 중복이 발생한다]
- **문제해결 → 정확한 해결:**
프론트에서 요청을 보낼 당시 쿼리 스트링으로 **현재까지 본 글을 id를 보내고, 백엔드 로직상 그 id값들을 제외한다.**

Swagger를 통한 API 문서화

- **요구사항:**
프론트엔드 개발자와의 협업 효율 향상
API 사용법을 명확히 전달
- **작업:**
Flasgger를 사용하여 Swagger 문서 자동 생성
각 엔드포인트에 YAML 파일로 상태로 빌드
Swagger UI를 통한 실시간 API 테스트 환경 구축
- **기대효과:**
코드에서 자동으로 문서가 생성되어 **문서와 코드의 불일치 방지**
Swagger UI를 통해 **프론트엔드 개발자가 직접 API 테스트 가능**
- **결과:**
문서 유지보수 부담 감소
프론트엔드 개발자와의 협업 효율 향상

관리자 - 대시보드 / 배포

코드 버전에 따른 API 성능 비교 그래프 Report 대시보드



CI/CD 파이프 라인 구축

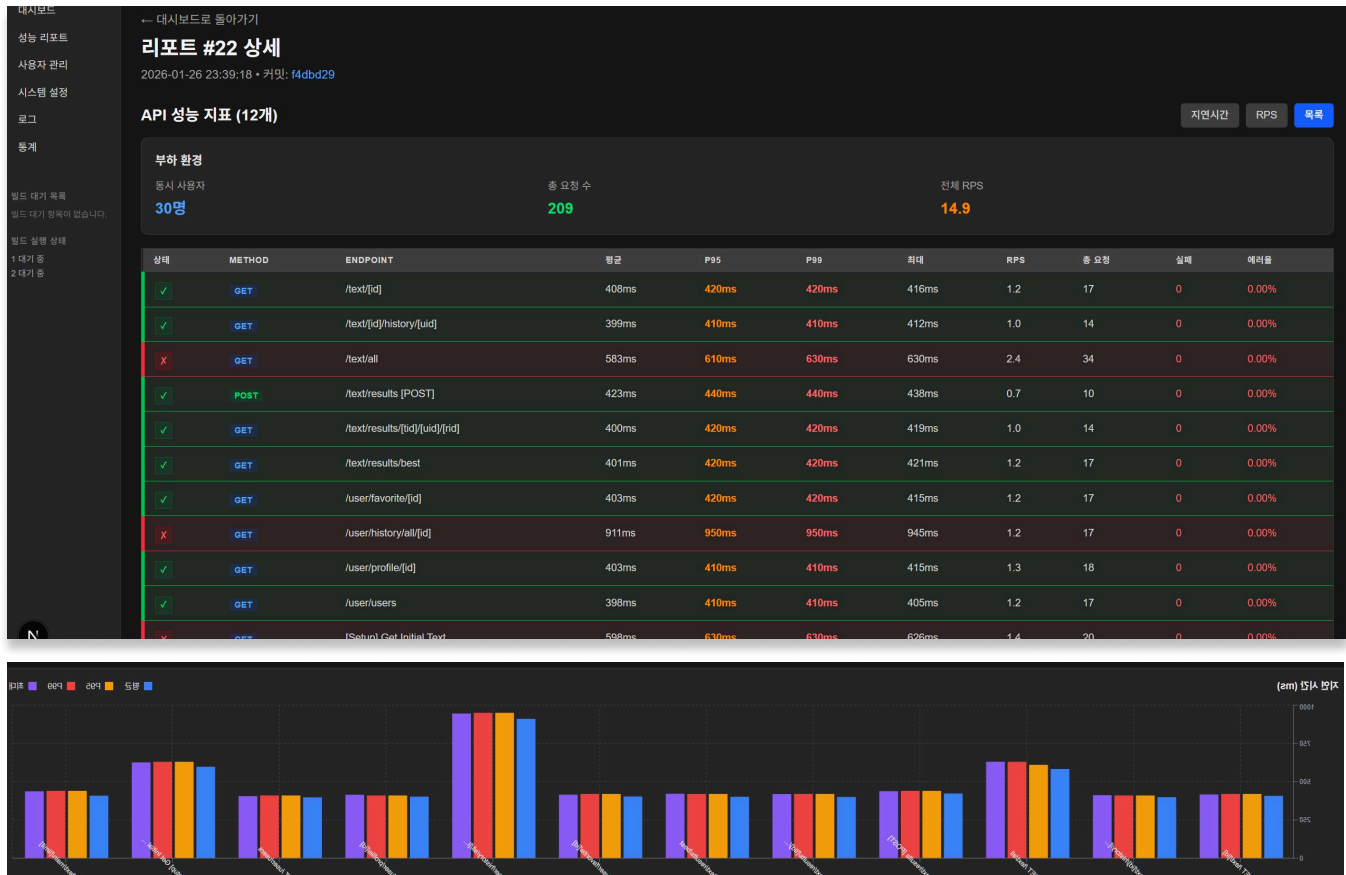
- 요구사항:**
코드 배포 시 자동으로 단위 테스트 및 부하 테스트 실행
배포 후 성능 지표 자동 수집 및 저장
- 작업:**
GitHub Actions 워크플로우 구성
Pytest **단위 테스트** 실행 (JSON 리포트 생성)
EC2 서버 자동 배포 (SSH 원격 실행)
Locust **부하 테스트** 실행 (동시 사용자 30명, 20초)
테스트 결과 수집 및 백엔드 API 전송
- 문제:**
GitHub Actions 서버 **CPU 리소스 부족**으로 부하 테스트 데이터가 텅 비어서 수집됨
Locust는 실행되지만 Request Count가 0으로 기록됨
- 문제 해결:**
GitHub Actions 내부에서 Locust를 실행하지 않고, **SSH로 EC2에 접속하여 EC2 내부에서 Locust 명령어 실행**
테스트 완료 후 생성된 `perf_stats.csv` 파일을 `scp` 명령어로 GitHub Actions로 가져와서 API 전송 로직 실행

성능 데이터 수집 및 처리

- 요구사항:**
API별 상세 성능 지표 수집 (평균/P95/P99 지연시간, RPS, 에러율)
커밋별 성능 추이 추적을 위한 Git 정보 연동
- 작업:**
Locust CSV 결과 파싱 및 JSON 변환
Pytest JSON 리포트 파싱
- 문제:**
POST 부하 테스트로 인해 실제 데이터가 **RDS에 저장됨** (결과 저장 API 테스트 시)
- 문제 해결:**
부하테스트를 위한 테스트 유저를 생산
부하테스트 종료 후 **현재 시간부터 30분 이전 시각을 기준으로** 테스트 유저로부터 파생된 모든 데이터를 삭제

관리자 - 성능지표 상세페이지

API별 상세 성능 체크



API 별 Leytency 분석

- 요구사항:
 - RPS/성능수치/pytest 결과값을 페이지에서 제공해야함
 - 배포 후 성능 지표 자동 수집 및 저장
- 작업:
 - 상태 관리 및 조건부 렌더링 패턴
 - 데이터 변환 및 집계 로직
 - 컴포넌트 분리 및 관심사 분리
- 문제:
 - 각 그래프와 테이블이 필요로 하는 데이터 구조가 다름에도 불구하고, 모든 데이터 변환을 매 렌더링마다 실행
 - 불필요한 컴포넌트 마운트로 인한 성능 저하
- 문제 해결:
 - Array.reduce()를 사용하여 불변성을 유지하면서 데이터 집계
 - 데이터 변환 로직을 컴포넌트 본문에 배치하여 렌더링 시점에만 실행