김건우 포트폴리오

김 건 우

Backend Developer

✉️ Contact

- email | takealook97@naver.com
- phone | +82 010-3545-7628

🔗 Channel

- github | https://github.com/takealook97
- blog | https://geonwoo.kim/blog

🎓 Graduated

- 동국대학교 국제통상학과 (2017.03 ~ 2023.02)
- 김포외국어고등학교 영어과 (2013.03 ~ 2016.02)

🪪 Certificate

- SQLD - 2024.04
- OPIC IH - 2023.10

🚀 Introduction

- 안녕하세요! 집요하게 파고드는 개발자 김건우 입니다.
- 혼자 빠르게 가는 것보다 함께 멀리 가는 것에 가치를 두고 있습니다.
- 일상의 불편함을 코드로 해결하는 사람이고 싶습니다.
- 최신기술에 관심이 많고, 프로젝트에 적용해보며 내 것으로 만드려고 노력합니다.
- 홈서버를 운영하며 나만의 서비스를 만들어가고 있습니다.
- 끊임없이 생각하고 탐구하는 과정을 즐깁니다.
- 원하는 결과가 나올 때까지 끝까지 매달리는 끈기와 열정을 가지고 있습니다.

📚 Education

삼성 청년 소프트웨어 아카데미 (SSAFY) 전공 과정

- 기간 | 2023.07 ~
- 교육 내용 | Java, 알고리즘, DB, MyBatis, 웹 프론트, 웹 백엔드, Vue.js, SpringBoot, 팀 프로젝트 등

코드스쿼드 백엔드 코스

- 기간 | 2023.01 ~ 2023.06
- 교육 내용 | CS, Java, Spring, DB, 팀 프로젝트 등

🛠️ Skills

Backend

- Java
- Spring Boot, Spring MVC
- Spring Data JPA

DevOps

- AWS EC2, S3, RDS
- Docker
- Nginx
- Proxmox
- github-action, gitlab-ci
- Jenkins

Database

- MySQL, MariaDB
- Redis

Collaboration

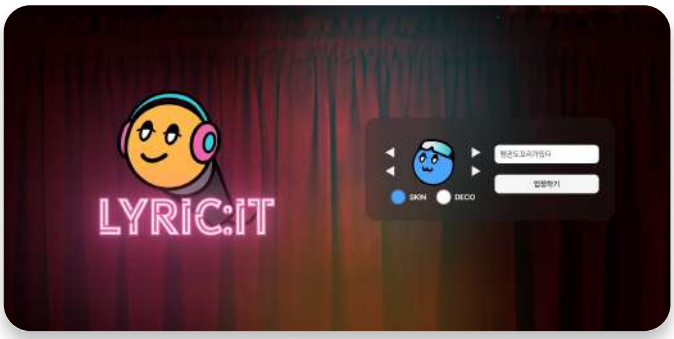
- Git
- Jira

🎯 Projects

리릭잇

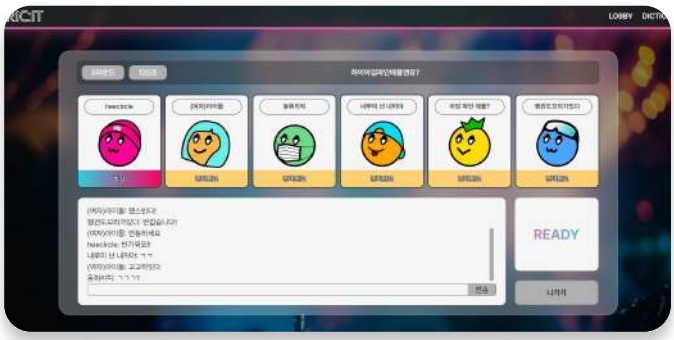
- 주제 | 노래 가사 기반 게임 서비스
- 기간 | 2024.02.19 ~ 2024.04.05
- 인원 | 총 6명 (백엔드 2인, 프론트엔드 2인, 데이터 2인)
- github | https://github.com/lyricit/lyricit-be
- notion | https://takealook97.notion.site/lyricit-b33c49f68e194e3692008dc9b8839c68

LYRIC:IT

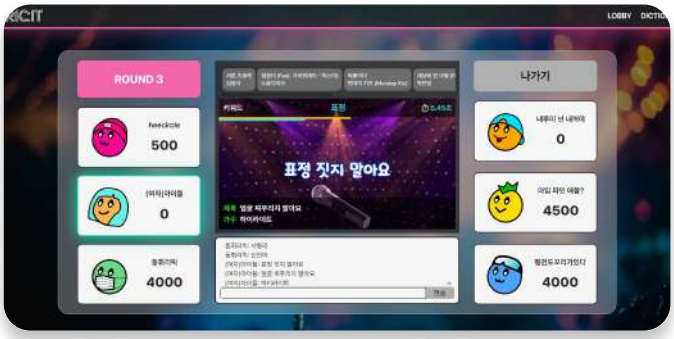


- **Language** | Java 17
- **Framework** | SpringBoot 3.2.1
  Spring Data JPA
  WebSocket (STOMP)
- **Database** | MySQL 8.0.35
  Redis 7.2
- **Server** | AWS EC2 t2.xlarge
  Nginx Proxy Manager
  Docker
- **CI / CD** | Jenkins
- **Tools** | gitlab, IntelliJ, Jira, figma

- WebSocket 통신 활용
- 실시간 방 정보 업데이트 및 채팅 기능



- 방 생성 및 게임 준비, 시작 기능
- 다양한 게임 진행 방식 설정



- 분산 처리된 가사 데이터(키워드) 제시
- 채팅을 통한 선착순 방식의 경쟁

- 가사 데이터 검색 기능
- 키워드를 포함한 다양한 가요 데이터 제시

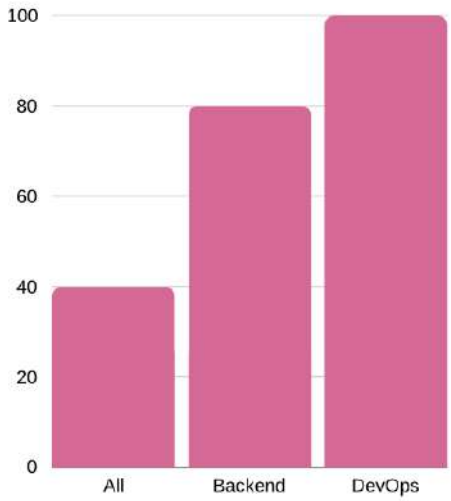
🎯 기획

- 노래 가사 기반 소통형 서비스
- 실시간 웹소켓 통신을 활용한 텍스트 기반의 인터렉티브 게임
- 개개인의 경험 및 채팅을 통한 힌트를 기반으로 정답 도출
- 크롤링 및 가사 데이터 분산 처리를 통한 역대 인기 가요 데이터 활용

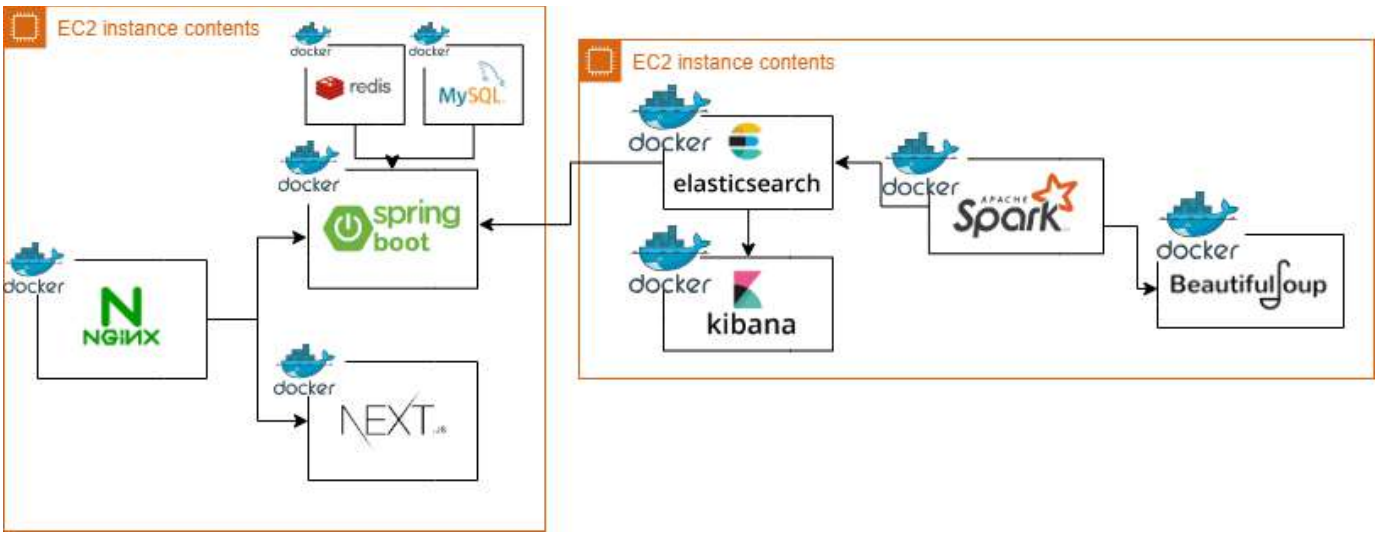
👨🏻‍💻 역할

- 백엔드 리더
- 서버 CI/CD 구축 및 인프라 총괄
- 웹소켓 통신 구축
- 회원 기능 API 구축
- 채팅 기능
- 방 생성 및 게임 참여 API 구축
- 게임 플로우 전체 리팩토링
- 게임 내 타이머 스케줄링

📊 기여도



🏗️ 배포 구조



🧐 어려웠던 점

1. pub / sub topic 설계

- 게임 내 존재하는 수많은 topic을 어떻게 관리하는게 유리할지 고민을 했습니다.
- 리소스를 효율적으로 활용하면서 동시에 채널 관리가 용이한 방식을 구상해야 했습니다.
- 개발 비용을 줄이기 위해서는 직관적인 방식을 선정해야 했습니다.
  - 클라이언트 컴포넌트 별로 topic 지정 vs 페이지 별 topic 지정 후 message type 설정

💡 이렇게 해결했습니다.

- 페이지 별로 topic을 싱글 채널로 관리하며 이에 대한 통일된 Response를 지정했습니다.
- 상황에 따른 다양한 message type을 활용하는 방식으로, 채널의 관리 복잡성을 줄였습니다.
- 이를 통해 리소스 부담을 줄일 수 있었고 동시에 기능 별 유연한 확장이 가능하도록 하였습니다.

2. 세션 Disconnect 문제

- 회원은 로그아웃의 개념이 아닌, 소켓 세션이 끊어지는 것을 기준으로 오프라인 처리를 했습니다.
- 이 때, Redis에서 관리하는 방과 게임에 대한 세부적인 업데이트 로직이 필요했습니다.
- 뿐만 아니라, 타임아웃 메커니즘으로 인해 연결이 끊어지는 문제도 해결해야 했습니다.

💡 이렇게 해결했습니다.

- 인터셉터를 통해 STOMP의 command를 확인하여 다양한 분기 처리를 하였습니다.
- 게임 중 단일 유저가 disconnect 되면, 해당 게임과 방의 데이터를 삭제 후 업데이트 하였습니다.
  - 채널 인터셉터에서 이벤트 리스너를 활용하여 순환참조를 예방할 수 있었습니다.
- 나아가 HeartBeat를 활용하여 세션 끊김 현상을 해결할 수 있었습니다.
  - ping, pong 프레임을 일정 시간마다 주고 받으며 연결성을 보장할 수 있었습니다.

🏆 성과

- 데이터 포지션과의 협업 경험
- 웹소켓 통신 기술 활용 경험
- Jenkins를 활용한 다양한 배포 시나리오 설정
- 서비스 플로우를 고려한 다양한 스케줄링 전략 도출
- 게임 프로젝트 경험 및 지속적인 QA를 통한 사용성 증대

명절 잔소리 영수증 (최우수 프로젝트)

- 주제 | 명절 기간 중 잔소리 및 대처 경험 공유 서비스
- 기간 | 2024.01.02 ~ 2024.02.16
- 인원 | 총 6명 (백엔드 3인, 프론트엔드 3인)
- github | https://github.com/jansorry/jansorry-be
- notion | https://takealook97.notion.site/jansorry-d926a1053a9e4e02ac849c5f1a3d263b

- **Language** | Java 17
- **Framework** | SpringBoot 3.2.1
  Spring Data JPA
  Spring Security
  Spring Batch
- **Database** | MySQL 8.0.35
  Redis 7.2
- **Server** | AWS EC2 t2.xlarge
  Nginx Proxy Manager
  Docker
- **CI / CD** | gitlab-runner
- **Tools** | gitlab, IntelliJ, Jira, figma

- 다양한 피드
- 좋아요, 팔로우 기능

- 마이페이지
- 게시글 확인

- 영수증 출력
- 영수증 공유

- 통계 데이터 도출
- 최종 데이터 시각화

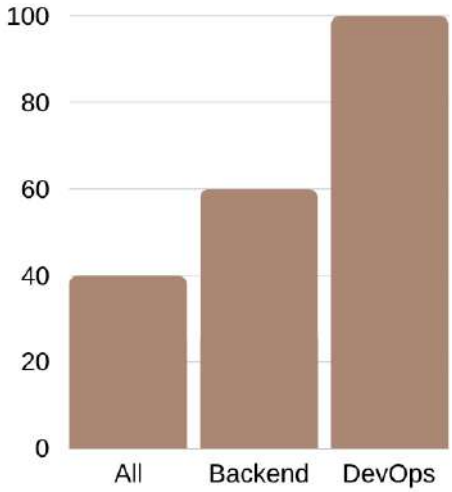
🎯 기획

- 명절 잔소리로 인한 젊은 세대의 스트레스
- 세대 간 단절된 구조 개선 시도
- SNS 기능을 통한 잔소리 및 대처 경험 공유 서비스 제공
- 영수증 발급 및 데이터 시각화를 통한 세대 갈등 완화

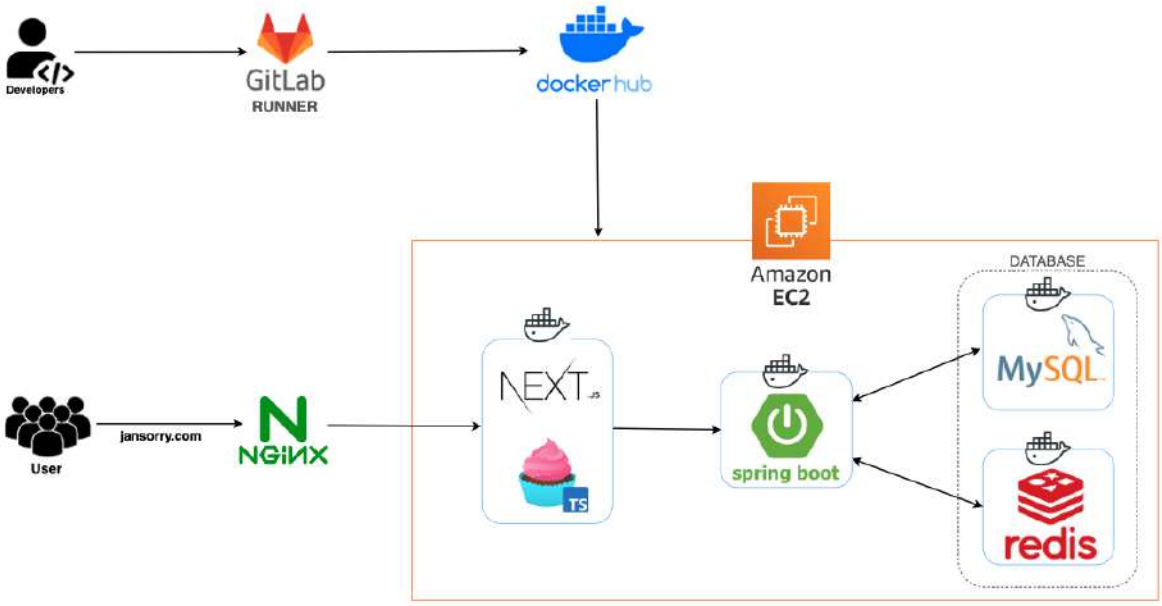
👨‍💻 역할

- 팀장
- 백엔드 리더
- 인프라 및 기술 총괄
- 서버 및 CI/CD 구축
- 메인 서비스 API 구축
  - 잔소리 관련 기능 전체
  - 좋아요 및 팔로우 기능
  - 영수증 생성 및 공유 기능
  - 배치 스케줄링 기능

📊 기여도



🏗️ 배포 구조



🧐 어려웠던 점

1. Redis 내에서의 팔로우 양방향 관계 지정

- 비관계형 데이터 베이스에서 팔로우 관계를 어떻게 처리할 것인지 긴 고민을 했습니다.
- 단방향 관계인 좋아요 기능과는 달리 팔로우의 N:M 관계를 어떻게 풀어나갈 지 초점을 맞췄습니다.
- 기회비용을 고려해야 했습니다.
  - 양방향 저장 시 데이터 적재량 증가 vs 단방향 데이터 저장 시 탐색 효율 저하

💡 이렇게 해결했습니다.

- 와일드카드를 최대한 지양하는 방향이 올바르다고 판단하여 양방향 저장을 선택했습니다.
- "following", "follower" 접두사를 붙이는 key 네이밍 전략을 활용하여 양방향으로 저장했습니다.
- 최종적으로, 유저의 팔로워 및 팔로잉 리스트 조회 시 탐색 시간을 크게 줄일 수 있었습니다.

2. Spring Batch를 통한 데이터 동기화 시 전체 탐색 문제

- 배치 작업에서 Redis의 데이터를 MySQL로 동기화하는 작업은 매일 새벽 3시에 진행됐습니다.
- 이 때, Redis에서 업데이트 된 데이터만을 배치 작업에 태우기 위한 다양한 시도를 했습니다.

💡 이렇게 해결했습니다.

- Redis의 ZSET 자료형을 활용하여, 스코어를 데이터 업데이트 시간으로 지정했습니다.
- 데이터가 업데이트 될 때마다, 해당 데이터의 key값을 update ZSET에도 반영했습니다.
- 이를 통해, 이전 배치 시간 이후 발생한 업데이트의 키 값만을 동기화 할 수 있었습니다.
- 배치 마무리 시점에는, ZSET을 초기화함으로써 새로운 업데이트 set만을 담을 수 있게 했습니다.
  - 나아가, 좋아요나 팔로우 취소로 인해 Value의 자료형이 Empty일 경우,
    이를 테이블에서 제거하여 추가적인 공간 확보 작업도 진행했습니다.

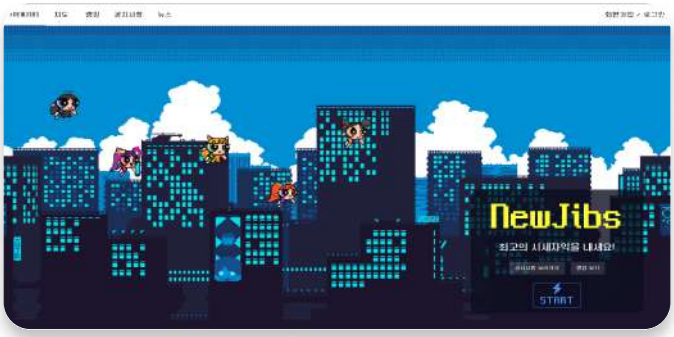
🏆 성과

- 우수 프로젝트 선정 (1위)
- 300명 이상의 스트레스 지수 설문 사전 데이터 확보
- 설 연휴 기간 동안 약 200명의 회원 확보
- 팔로잉 팔로우 리스트 조회 시 탐색 효율 증대
- 배치 작업 최적화 및 리소스 확보

뉴집스

- 주제 | 부동산 모의 투자 서비스
- 기간 | 2023.11.01 ~ 2023.11.24
- 인원 | 총 2명 (백엔드 1인, 프론트엔드 1인)
- github | https://github.com/NewJibs/newjibs-be
- notion | https://takealook97.notion.site/newjibs-5432f78d755c430c8d963b9056fba4f0

NEWJIBS



- Language | Java 8
- Framework | SpringBoot 2.7.16
  Spring Data JPA
  Spring Security
- Database | MariaDB 2.7.4
  Redis
  S3
- Server | AWS EC2 t2.micro
  Docker
- CI / CD | CloudType, github-action
- Tools | github, IntelliJ, figma

- 카카오 map api 활용
- 위,경도 데이터 클러스터 렌더링

- 2020 ~ 2022 부동산 매매가 기준 데이터 기반
- 100억원의 시드를 바탕으로 투자 진행

- 투자 결과 및 수익률 도출
- 랭킹 시스템

- 실시간 부동산 뉴스 크롤링 데이터 활용
- Jsoup을 활용한 크롤러 스케줄링

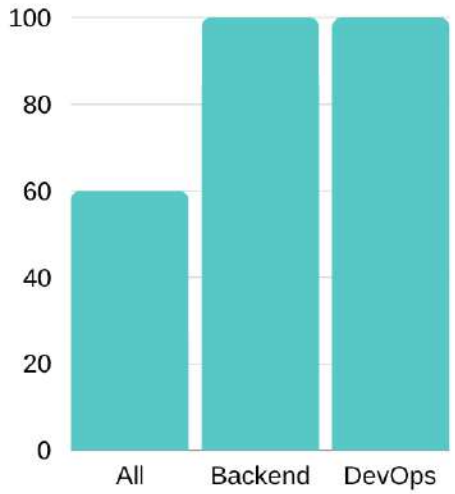
🎯 기획

- 코로나 기간 동안 가파른 부동산 가격 상승 배경
- 예산 100억을 기반으로 부동산에 투자하여 2년 뒤 수익률 확인
- 실제 시장 시뮬레이션을 통한 다양한 시장 시나리오 경험
- 부동산 투자 옵션 경험을 통한 포트폴리오 다각화

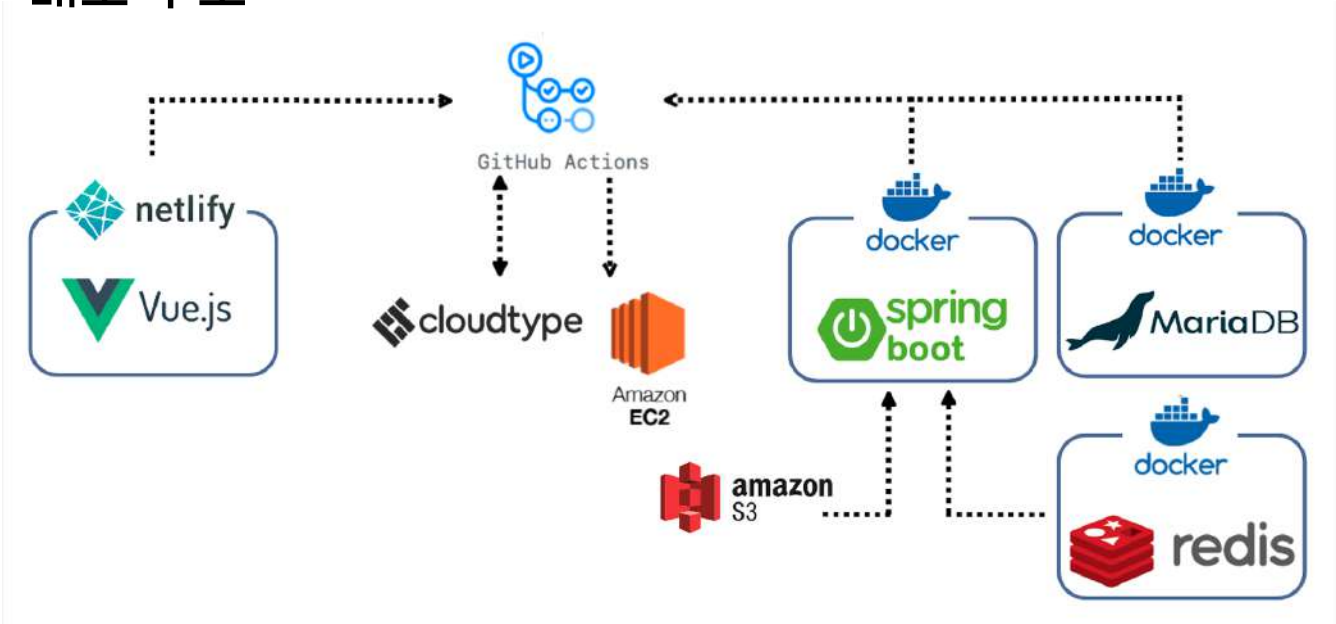
👨🏻‍💻 역할

- 팀장
- 백엔드 총괄 (1인)
- 서버 및 CI/CD 구축
- 회원 기능
- 랭킹 조회 기능
- 부동산 모의투자 기능
- 공지사항 기능
- 뉴스 기능
  - 크롤러 스케줄링

📊 기여도



🏗️ 배포 구조



🧐 어려웠던 점

1. 400만 개의 데이터 축소작업

- "부동산 모의 투자"라는 주제에는 400만개가 넘는 데이터가 전부 필요하지는 않았습니다.
  - 2016 이후 발생한 모든 매매 데이터가 있는 상황이었고 이는 렌더링 성능저하로 이어졌습니다.
- 이를 해결하기 위해 데이터를 임의의 기준에 맞추어 축소시켜야했습니다.
- 코로나19의 시작인 2020년을 기점으로 부동산 가격이 많이 오른 점을 감안했습니다.

💡 이렇게 해결했습니다.

- 2020년과 2022년에 동일한 층수와 평수를 가진 데이터를 1:1 매핑 시키는 작업을 진행했습니다.
- 동일 가구의 매매 데이터를 기반으로 하여 보다 신뢰도 높은 투자 결과를 파악하려고 했습니다.
- 최종적으로 400만 개 이상의 데이터를 6만 개 정도로 줄였고, 렌더링 성능 또한 크게 개선했습니다.

2. 뉴스 크롤링 시 발생하는 bot 인식 문제

- Jsoup을 활용하여 매 정각마다 최신 부동산 뉴스 데이터를 크롤링했습니다.
- 기계적 호출로 인해 bot으로 인식이 되었고, 매번 서버 ip가 차단당하는 문제가 있었습니다.
  - 셀레니움을 활용하기 보다는 크롤러 내부적으로 해결을 하고자 했습니다.

💡 이렇게 해결했습니다.

- 정각마다 데이터를 호출하는 문제 + 헤더값 때문에 bot으로 인식한다는 점을 파악했습니다.
- 헤더를 교체하고, 데이터 호출 시점에 임의의 랜덤 딜레이를 부여하였습니다.
- 이를 통해 안티 크롤링 매커니즘을 우회화여 스케줄링 패턴의 명확성을 낮출 수 있었습니다.

🏆 성과

- 데이터 축소를 통한 렌더링 성능 증대
- 백엔드 전체 담당 경험
  - 백엔드 개발 플로우 전체 학습
- 기술적 성장
  - JWT 활용 경험
  - Jsoup 크롤링 경험
  - 스케줄링 경험을 통한 task 설정 경험

트리플렛 (신한은행 해커톤)

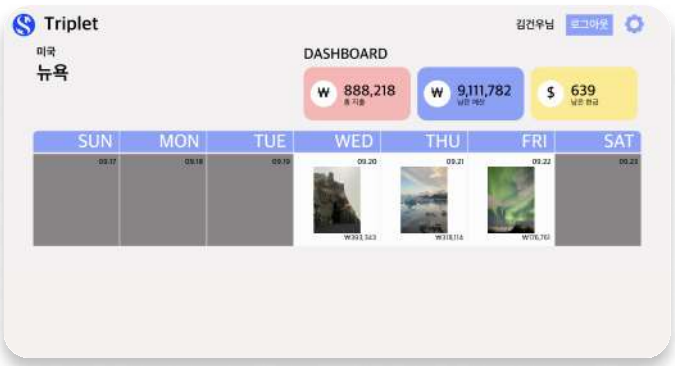
- 주제 | 여행 금융 다이어리 서비스
- 기간 | 2023.09.01 ~ 2023.09.17
- 인원 | 총 4명 (백엔드 3인, 프론트엔드 1인)
- github | https://github.com/Triplet-Shinhan/Triplet
- notion | https://takealook97.notion.site/triplet-a4a4b42ac9ff450ebaca5b3ea517002a

Triplet

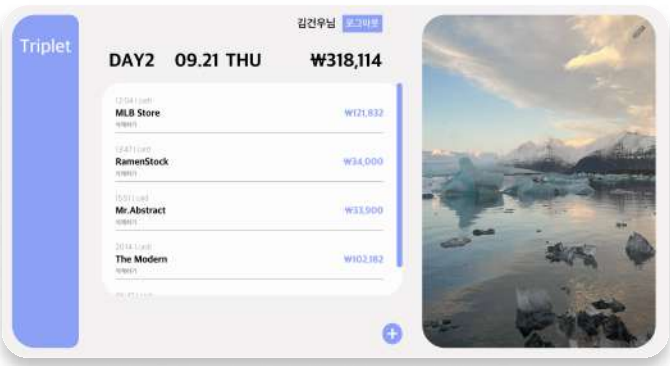
- Language | Java 11
- Framework | SpringBoot 2.7.10
  Spring Data JPA
- Database | MySQL 8.0.31
  S3
- Server | AWS EC2 t2.micro
  Docker
- CI / CD | github-action
- Tools | github, IntelliJ, figma

- 프로젝트(여행지) 관리
- 환전 신청 기능

- 프로젝트 생성 시 여행지 및 기간 설정
- 예산 및 보유 현금 설정



- 달력 형태의 금융 다이어리
- 일일 지출 금액 확인



- 지출 발생 시 실시간 환율 반영 후 자동 기록
- 이미지 업로드를 통한 다이어리 커스마이징

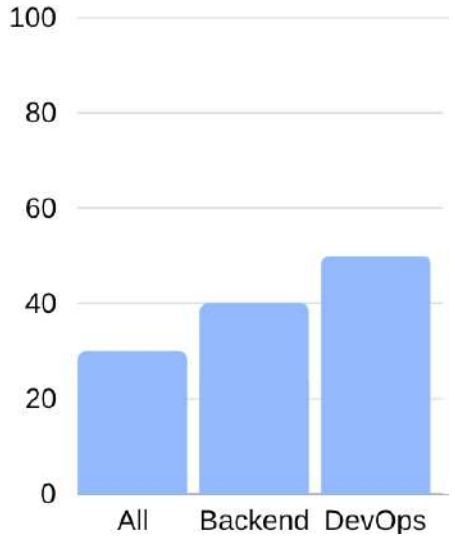
🎯 기획

- 일반적인 여행 지출 정리 방식의 문제점
  - 환전 현금 지출 및 카드를 통한 외화 지출
- 환율 API를 활용하여 수시로 변동하는 환율 고려
- 자동으로 지출을 원화로 계산하여 기록
  - 정확한 계산 및 자동 기록을 통한 편리함 도모

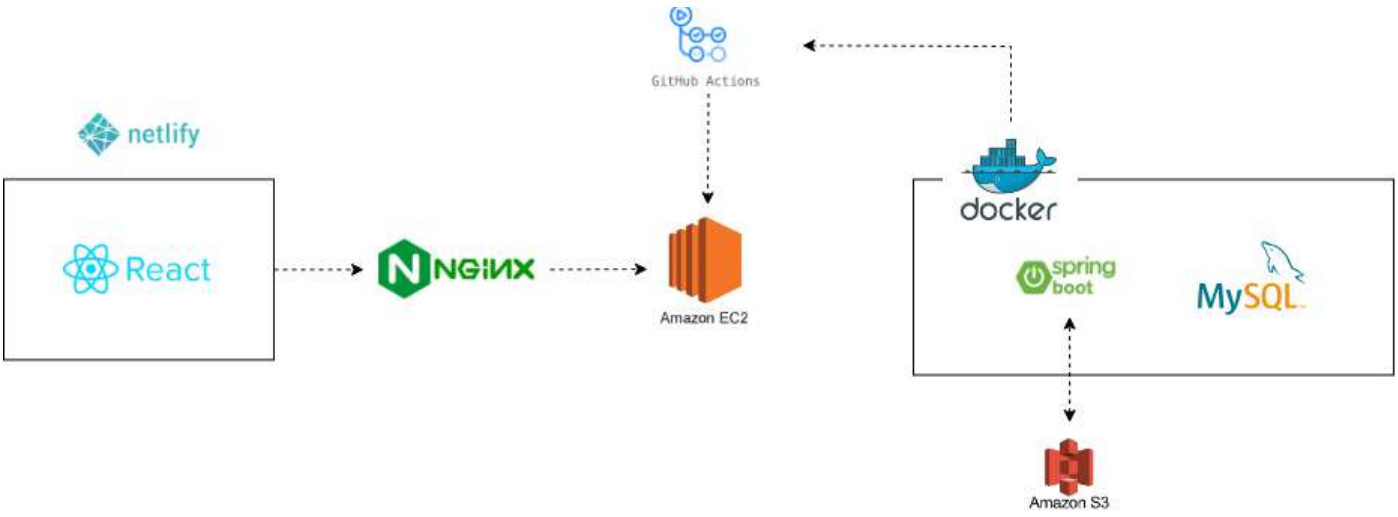
👨‍💻 역할

- 백엔드 메인 서비스 API 구축
  - 회원 기능
  - 여행 관리 기능
  - 데일리 관리 기능
  - 이미지 업로드 기능
- 서버 및 CI/CD 구축

📊 기여도



🏗️ 배포 구조



🧐 어려웠던 점

1. 현실과는 달랐던 API

- 실제 은행의 API에는 수 많은 보안, 기술적 이슈가 따랐기에 동작하는 API를 제공받지 못했습니다.
  - 그래서 해커톤은 더미 데이터로 이루어진 API를 제공 받은 상태로 진행됐습니다.
- 거래 로직부터 환율 정보 등을 직접 파악하여 프로젝트를 진행해야 했습니다.

💡 이렇게 해결했습니다.

- 지출부터 환율 정보 등등, 다양한 상황을 가정하여 추가적인 API 및 더미데이터를 만들었습니다.
- 이를 통해 유저 시나리오에 맞춘 다양한 상황을 연출할 수 있었습니다.

2. EC2 인스턴스 사양 문제

- AWS를 무과금으로 돌리기 위해 프리티어 계정의 인스턴스에 모든 컨테이너를 올려야 했습니다.
- 지속적으로 사용량이 치솟으면서 인스턴스가 종료되는 이슈가 있었습니다.
  - 이를 해결하기 위해 사양을 과금 모델로 올리는 방법도 생각했었습니다.

💡 이렇게 해결했습니다.

- 프리티어의 1기가 램은 너무나도 작았기에 메모리 스왑을 통해 해결해야했습니다.
- 첫 배포 작업이었기에 문제 원인을 파악하기까지 오래걸렸습니다.
- 최종적으로 한정된 자원을 최대한 효율적으로 쓰는 방법의 중요성을 깨달았습니다.

🏆 성과

- 해커톤 진출 (8:1 경쟁률)
- 다양한 기술 경험을 통한 개인적 성장
  - JPA를 활용한 첫 프로젝트
  - 첫 배포 경험
- 발표 경험을 통한 PT 역량 성장
- 협업 능력 증대