이력서:
OOO

인적사항	**성명**	
	생년월일	
	휴대전화	
	E-mail	
	Blog	https://deveric.tistory.com
	Github	https://github.com/yyy9942
자격증	2019.04	**정보처리기사**
병역	구분 (필)	0000.00 - 0000.00 **병장 만기전역**
학력사항	**2014.03 - 2020.02**	
	한남대학교 공과대학	
	멀티미디어공학과 학사 졸업예정	
활동 내역	기간	2018.09-2018.12
	장소	**한남대학교**
	활동 내용	**멀티미디어 학과 연구실 인턴**
	-	**NAS 서버 구축**
	-	OS: ubuntu16.
	-	**성과 세부내용:** 클라우드 파일 서버 구축 및 운영을 진행 - 우분투 설치 및 환경 설정 - 사용자별 권한 관리 - 클라우드 파일 저장소 관리
	기간	2018.09-2018.12
	장소	**한남대학교**
	활동 내용	**학과 코딩 튜터링**
	-	**C++, Java 기초 튜터링** **성과 세부내용:** 학과 내 후배들을 대상으로 C++, Java 기초부터 자료구조 구현까지 강의 진행. 언어의 특성 보다는 프로그래밍 기초 지식을 위주로 진행.

보유 기술 및 프로젝트		
	Java Spring	배달 매칭 API(제작 중), 커뮤니티 게시판
	Java FX	부동산 통합 중개 플랫폼
	Android	일정관리 앱, 메시징 앱
	DBMS	MariaDB, Oracle, Redis
	R	후쿠시마 원전 사고 이후 방사능 수치변화 분석
	Cloud	AWS, Cafe24, Naver Cloud Platform

포트폴리오

DelFood 음식 배달 O2O 서비스

사용 기술 : Spring-boot, MyBatis, MariaDB, Redis, Docker, Jenkins, Naver cloud platform

기간 : 2019.09 - 2020.02

Github : https://github.com/f-lab-edu/food-delivery

Wiki : https://github.com/f-lab-edu/food-delivery/wiki

Blog : https://deveric.tistory.com/category/Project/DelFood

Jenkins : http://106.10.51.119:8080/

프로젝트 개요

'배달의 민족'을 모티브로 만든 음식 배달 O2O Rest API입니다. 위치 기반 배달 서비스를 제공하며 실시간 라이더 매칭을 제공합니다. 대한민국 공공 도로명 주소 DB를 활용하여 라이더에게 배달하는 건물 좌표와 출입구 위치를 제공합니다.

고객, 사장님, 배달원 3가지의 타입으로 서비스를 이용할 수 있습니다. 고객은 자신의 위치를 기반으로 주위 배달 음식점들을 조회하여 주문할 수 있습니다. 사장님은 자신의 매장만 등록하면 고객, 배달원과 자동으로 매칭됩니다. 배달원은 자신의 실시간 위치를 등록하여 매장과 매칭되고, 고객에게 배달을 진행합니다.

Naver Cloud Platform을 사용하여 배포하고 있습니다. Delfood의 서버는 CI/CD를 담당하는 Jenkins 서버, RDBMS를 담당하는 MariaDB 서버, 캐시와 세션데이터를 저장하는 Redis 서버로 구성되어 있습니다. Public IP는 메인 서버에만 열어두어서 외부에서 직접 DB에 접근할 수 없도록 설계하였습니다.

어플리케이션 성능 테스트를 위해 실제 트래픽을 발생시켜 모니터링하고 있습니다. NGrinder를 클라우드 서버에 설치하여 모니터링하고 있으며 그 결과를 바탕으로 성능 튜닝을 준비하고 있습니다.

프로젝트 진행 과정

프론트 개발에 들어가는 시간을 아껴 서버 공부에 투자하기 위하여 kakao oven을 이용해 프로토타입을 제작하고 기능 요구 사항을 추출하여 설계를 진행하였습니다

프로젝트의 전반적인 내용은 Github의 Readme, WIKI에 문서화를 진행하였습니다. Readme에는 프로젝트 설계를 요약본을 넣었습니다. wiki에는 Business Rule과 발생한 문제를 기술적으로 풀어낸 글을 정리하고 있습니다.

Github을 통해 코드를 리뷰하며 상호간 코드의 허술함을 줄여 코드 품질을 보완하고 있습니다. 서로의 코드에 대해

이해하기 어려운 부분 있으면 코멘트하여 그 부분에 대한 상세한 설명 주석을 달고 있습니다. 이를 통해 서로의 개발 진행상황을 파악할 수 있었고 공부한 기능을 공유할 수 있었습니다.

서로가 작성한 코드를 이해하기 위해 다른 사람이 작업한 코드에 테스트코드를 작성하고 있습니다. 이를 통해 서로 다른 내용의 작업을 했더라도 어떤 작업을 했고 어떤 기능을 작성하였는지 알 수 있었습니다.

이슈단위로 브랜치를 관리하고 있습니다. 브랜치 관리 전략은 Git Flow를 적용하고 있습니다.
프로젝트에 대한 더 자세한 정보는 Github의 ReadMe를 참고해주시길 바랍니다.

로그인 시 Redis를 활용한 세션 데이터 관리

해당 프로젝트는 확장이 용이한 구조를 지향하고 있습니다. 그렇기 때문에 회원 로그인 아이디 정보를 세션에 저장하였을 때 Scale-out 시 세션이 분리되어 일관적인 세션 데이터관리가 어렵다는 문제를 어떻게 해결하는 것이 좋을까 고민하였습니다. 세션 정보를 데이터베이스에 연동하여 관리한다면 여러 서버에서 하나의 세션 저장소를 공유할 수 있을 것이고, 어떤 DBMS를 이용해야 세션 관리가 효율적일지 장단점을 비교해보았습니다.

속도 보다는 운영의 효율성을 고려하여 Memcached보다는 Redis를 사용하기로 결정하였습니다. Memcachaed는 안정적인 응답 속도와 빠른 저장속도를 가진다는 장점이 있지만 상대적으로 서버 운영에 필요한 기능이 부족했습니다. 그에 비해 Redis는 Spring에서 공식적으로 지원하고 있기 때문에 참고할 수 있는 자료가 풍부하고 다양한 Eviction을 지원하여 메모리 관리 및 선택에 정교한 접근법 사용이 가능하다는 장점이 있고, DB 이미지를 Disk에 저장하여 서버 손상시에도 데이터를 복구할 수 있다는 장점이 있었습니다.

이를 통해 Redis서버를 구축하여 클라이언트와의 세션을 관리하기로 결정하였습니다. 멀티 서버 환경에서도 모든 서버가 Redis를 통해 세션 데이터를 저장하고 참조할 수 있게 설정하였습니다. 이를 통해 서버간 세션의 공유를 가능하게 만들었고 사용자의 세션에 저장한 데이터를 관리할 수 있었습니다.

검색 시 DB 서버 부하를 고려한 캐시 전략

회원 가입을 진행하거나 회원이 주소를 변경하고자 할 때 주소를 검색하는 로직이 DB에 큰 부하를 주었습니다. 1000만건 이상의 공공 주소 데이터를 DB에서 검색하기 위하여 인덱스도 설정해 보았지만 인덱스 만으로는 주소 검색 시 마다 DB에 가해지는 부하를 감당하기 어려웠습니다.

주소 DB의 사용을 추적해본 결과 동일 주소의 검색이 빈번하다는 것을 확인하였습니다. 회원이 가지고 있는 주소 코드를 기반으로 사장님에게 상세 주소를 전달해야 하고 회원에게도 배달 받을 주소를 전달해야 하기 때문입니다.

그렇기 때문에 WAS와 DB서버의 부하를 줄이기 위하여 캐싱을 하기로 하였습니다. 주소 검색은 동일 검색조건 하에 대부분 동일한 결과값의 반환이 이루어 지고 DB서버에 부하가 커서 캐싱을 적용하였을 때 성능과 속도 개선의 여지가 크기 때문입니다. 또한 주소데이터는 데이터의 변경이 거의 발생하지 않기 때문에 캐시 데이터의 만료시간을 길게 주어서 캐시의

재사용성을 높였습니다.

한번 캐싱된 데이터를 다른 서버에서도 조회할 수 있도록 Redis를 사용해 글로벌 캐싱을 적용하였습니다. 또 주소 데이터는 변경이 거의 일어나지 않기 때문에 캐시 데이터의 expire time을 길게 주었습니다. 이를 통해 평균 응답시간이 300ms에서 10ms미만으로 감소하였습니다.

Redis의 동시성 이슈를 고려한 로직 처리

고객은 매장에서 검색한 메뉴를 장바구니에 저장할 수 있습니다. 장바구니 기능은 expire 설정이 필요하고, 상대적으로 데이터의 중요도가 낮기 때문에 DB 서버의 부하를 줄이기 위하여 Redis를 사용하였습니다
장바구니에 메뉴를 저장하는 로직은 멀티스레드 환경에서 동시성 문제가 생길 가능성이 있었습니다. 장바구니는 최대 10개의 메뉴가 저장될 수 있고, 비어 있는 상태에서 장바구니를 생성하기 위해서는 expire time을 설정해 줘야 하는 로직이 있었습니다.

예를 들어 Redis에 저장하는 장바구니 기능에서 물품을 추가할 때 다음과 같은 로직이 있습니다.

```
해당 회원 장바구니에 몇 개의 물품이 있는지 조회한다
-> 비어 있거나 최대 수량을 초과할 경우 적절한 로직을 호출한다
-> 장바구니에 물품을 추가한다.
```

이 때 장바구니를 조회한 후 유효성 검사를 하는 동안 다른 스레드가 장바구니에 접근하여 상태를 바꾸어 버린다면 동시성 이슈가 발생할 수 있을 것입니다. 그렇기 때문에 Redis를 호출하여 작업 시 해당 키를 watch하여 변경되는지 모니터링하며 로직을 진행하는 Redis Transaction을 적용하였습니다.

라이더의 매칭 도중 생길 수 있는 동시성 이슈 해결

매장에서 메뉴를 준비하며 배달원 매칭을 서버에 요청할 경우 매장 주위 일정 반경 이내에 있는 배달원들에게 푸시 알림 메시지를 전송합니다. 이 때 가장 먼저 매칭을 수락한 배달원에게 매장 배달 권한이 주어지는데 동시에 여러 배달원에게 요청이 올 경우에 동시성 이슈가 생길 수 있었습니다. 그 이유는 배달원의 현재 위치와 상태를 서버 로컬 메모리에 Map 형태로 관리하고 있었고, 라이더의 매칭 작업을 하는 도중 해당 정보가 변경된다면 여러 라이더가 동시에 매칭될 수 있었기 때문입니다.

이를 해결하기 위해 Map의 구현체로 동시성 이슈를 해결할 수 있는 SynchronizedMap과 ConcurrentHashMap을 적용하기로 하였습니다. 두 구현체를 비교하여 보니 SynchronizedMap은 전체적으로 lock을 거는 것에 비해 ConcurrentHashMap은 lock striping 기법을 적용하여 적용되는 lock을 분할하여 사용한다는 것을 알게 되었습니다.
성능적으로 ConcurrentHashMap이 월등하게 빨랐고 이를 적용하여 로직의 원자성을 지켜 동시성 이슈를 해결할 수 있었습니다.

로컬 Map에는 트랜잭션을 적용하기 어렵기 때문에 Redis를 사용하는 것도 좋은 방법이라고 생각하였습니다. 라이더 매칭과 관련된 기능을 추상화하여 구현체를 어떤 것을 사용하는지에 따라 로컬 메모리(Map), Redis중 하나를 선택하여

사용할 수 있도록 구현하였습니다. Redis를 사용할 경우 Redis 트랜잭션과 Key를 watch하는 방법을 통해 연산을 단일 연산으로 만들어 동시성 이슈를 해결하였습니다.

메뉴 주문 시 데이터 무결성을 위한 트랜잭션 적용

주문 로직은 여러 테이블에 insert와 update를 진행합니다. 저장과 수정을 진행하는 도중 오류 또는 예외가 발생한다면 데이터의 무결성이 깨질 것입니다. 그렇기 때문에 예외가 발생한다면 적용된 사항들을 저장하지 않고 롤백할 수 있도록 트랜잭션을 적용하였습니다.

주문이 실패하였다면 왜 실패했는지 기록하는 데이터가 필요했습니다. 해당 유저가 주문을 시도하였다는 정보를 따로 저장하기 위해 해당 로직을 별도의 트랜잭션으로 분리하였습니다. @Transactional의 propagation 속성을 NESTED로 설정함으로써 해당 로직은 상위 로직에 종속적이지 않는 별도의 트랜잭션으로 작동할 수 있도록 구현하였습니다.

AOP를 적용한 공통 관심사 분리

DelFood의 대부분의 서비스는 로그인을 해야 이용할 수 있습니다. 처음 프로젝트를 제작할 때 모든 서비스 마다 세션에서 로그인 정보가 있는지 확인하는 코드를 작성하였습니다. 하지만 이를 AOP를 적용하여 횡단관심사를 분리할 수 있었습니다. 이를 통해 로그인이 되었는지 확인하는 중복 코드들을 제거할 수 있었습니다.

공통 로직의 적용을 위해 어노테이션을 기반으로 AOP를 적용하였습니다. 이를 통해 하나의 클래스 안에서도 권한별로 호출 가능한 메소드를 더 세밀하게 조정할 수 있었습니다. 주소 검색시에도 비회원도 가능한 일반 검색, 회원이 가능한 내 주변 매장 주소 검색 등을 권한별로 나눌 수 있었습니다.

Reflection 적용한 사장님의 매장 권한 체크

사장님이 로그인하여 매장에 대한 조작을 진행할 때 해당 매장이 사장님의 소유인지 확인하는 로직을 먼저 작성해야 했습니다. Controller에서 매장에 대한 조작을 진행할 때 항상 매장 ID를 파라미터로 받아오는 것에서 아이디어를 얻어서 어노테이션을 적용한 후, 내부에 있는 파라미터에 매장 아이디를 저장하는 변수 명을 전달하면 자동으로 권한 체크를 진행하도록 하였습니다.

MethodSignature를 사용하여 Method와 Parameter를 추출하였고, 해당 파라미터의 이름을 조회하여 입력 받은 변수명과 일치하는지 검사하였습니다. 일치할 경우 적절한 서비스를 호출하여 권한의 검사를 진행합니다. 일치하는 변수명이 없을 경우 예외처리를 진행하도록 구현하였습니다.

Mockito와 Junit으로 고립된 단위테스트 작성

소스코드를 작성하거나 변경 시 오류가 나지 않는지 기능을 하나하나 URL에 데이터를 보내 직접 테스트를 했어야 했습니다. 이런 불편을 개선하기 위해 서비스 로직을 대상으로 Junit과 Mockito를 활용한 단위 테스트를 작성하였습니다.

의존성이 있는 객체를 대상으로 Mock DAO 객체를 만들어 주입하였고, DB에서 예상되는 결과값들을 반환하도록 설정하였습니다. 그 결과 Service 레이어에 있는 로직을 중심으로 고립된 단위 테스트를 제작할 수 있었습니다. 또한 Mockito Framework를 사용하여 Spring container가 올라가지 않고, DB와의 커넥션을 하지 않기 때문에 빠른 테스트가 가능하도록 하였습니다.

Jenkins를 활용한 테스트 자동화와 CI/CD 적용

소스코드를 작성하거나 수정할 때 직접 테스트를 진행하고 서로 그 결과를 공유해야 하는 번거로움이 있었습니다. 이러한 번거로움을 해소하기 위하여 Naver Cloud Platform 서비스를 이용하여 CI 서버를 구축하였습니다. 이 서버에서 자동으로 테스트를 진행할 수 있도록 Github에서 PR이 발생할 때 Jenkins 서버로 hook을 날려 빌드와 테스트를 진행하도록 적용하였습니다. 그 결과 협업할 때 서로의 소스코드 신뢰성을 보장받을 수 있었습니다.

배포 전 어플리케이션 기능의 무결성을 검사하기 위해 꼼꼼한 단위테스트를 작성했습니다. CI서버에서 배포 전 해당 단위테스트를 실행하고, 이상이 발생할 때 개발자에게 메일로 알림을 주도록 설정했습니다.

서버 배포의 편의성을 위해 빌드, 테스트 완료 시 도커 이미지를 제작하여 메인 서버에 배포하도록 설정하였습니다. CI 서버에서 이미지를 구축하면 원격 스크립트를 통해 메인 어플리케이션 서버에서 이미지를 받아 실행할 수 있도록 스크립트를 작성했습니다. 그 결과 소스코드 변경이 발생하면 자동으로 서버에서 빌드, 테스트, 배포까지 진행하도록 만들 수 있었습니다.

Java 스펙을 활용한 코드 품질 관리

Java에 어울리는 코드를 만들기 위하여 꾸준히 코드를 검수하고 있습니다. '이펙티브 자바', '자바 성능 튜닝 이야기' 등의 책을 참고하여 코드 품질 향상에 목표를 설정하였습니다. 이를 통해 프로젝트 코드를 아래와 같이 개선하였습니다.

프로젝트 초기에 null값 체크, 유효성 검사 등의 처리를 하기 위해 if-else문을 많이 만든 적이 있습니다. Controller에서 로직을 호출한 후 결과값을 받아 정상적인 결과값이 아니라면 직접 예외처리를 진행해야 했습니다. 이를 각 레이어의 책임을 분리하고, 이상이 생길 때 하위 로직으로 결과값을 반환하지 않고 예외를 던지도록 개선하였습니다. 이를 통해 무분별한 if-else문과 유효성 검사 코드를 제거할 수 있었습니다.

요청에 대한 반환 값을 Map을 통해 전달하는 것에서 명확한 객체를 통해 전달하는 것으로 변경하였습니다. Map은 어떤 데이터가 들어있는지, 데이터 타입이 무엇인지 명확하게 알 수 없다는 단점이 있기 때문입니다. 실제로 명확한 DTO, Response 객체를 사용하도록 변경하였을 때 개발의 편의성과 유지보수의 편리함이 있다는 것을 알 수 있었습니다.

스레드 안정성을 위해 필요한 경우가 아니라면 setter를 설정하지 않고 값을 복사하여 넘겨주어 불변 객체를 만들어 전달하였습니다. 이를 통해 의도치 않은 값의 변화를 억제하였습니다.

static final 상수를 사용하여 불필요한 객체 생성을 지양하였습니다. 정적인 반환 값이 있다면 이를 상수로 지정해 준 뒤

반환하도록 설정하였습니다. 이를 통해 객체 생성에 필요한 리소스를 줄일 수 있었습니다.

Java 스펙에 맞는 코드를 제작하기 위해 계속해서 코드를 리팩토링하고 있습니다. 작동하는 코드에 만족하지 않고 더 안정적이고 좋은 코드를 만들기 위해 노력하고 있습니다. Github에 commit하는 코드의 약 절반정도는 기존 코드를 개선하는 코드입니다.

니방내방 - 사용자 위치 기반 통합 주거 중개 플랫폼

사용 기술 : Java Fx, MariaDB, iBatis, PHP7, kakao Map, Geocoding

기간 : 2019.06.10 - 2019.06.30

Github : https://github.com/yyy9942/nibang-naebang

방 추천 알고리즘 개발 및 검증

사용자의 위치와 검색 기록, 조회 기록을 통해 추천 점수를 계산하여 맞춤 방 추천 기능을 구현하였습니다. 주소 - 좌표 변환 API를 활용하여 사용자 위치에서 가까운 방을 우선 조회하였습니다. 그 다음 사용자가 검색한 키워드가 포함되어 있거나 비슷한 조건을 가지고 있는 경우 방 추천 점수를 높여 다른 방보다 상단에 노출하였습니다.

매물 퀄리티 평가 알고리즘 개발 및 검증

사용자가 검색하거나 조회한 방에 기반한 추천 점수가 있다면 반대로 사용자에게 객관적으로 이 방이 얼마나 좋은 조건을 가지고 있는지 보여줄 수 있는 객관적인 점수가 필요할 것입니다. 공공데이터를 활용하여 방 주변 상권 데이터를 분석하였습니다. 그리고 주위 방들과 조건을 비교하여 방의 점수를 계산한 후 사용자에게 다양한 차트를 이용하여 참고할 수 있는 점수를 제공하였습니다. 이를 통해 실제로 퀄리티가 좋은 방이 어느 정도 정렬되어 보이게 되었습니다.

정보의 최신 성 고려

고객, 임대인이 주소를 입력할 때 일부 주소만으로 자동 완성 기능을 제공하고, 이를 좌표로 변환해 주는 서비스를 구현하였습니다. 구현 방법으로 두 가지 방법을 고려하였는데, DB에 모든 주소 정보를 저장하고 사용자가 입력하는 주소를 키워드로 검색하여 제공하는 방법과 API 서버를 이용하여 제공하는 방법을 고려하였습니다.

DB에 모든 주소를 저장해 두고 제공하는 방법은 주소 체계가 바뀌거나, 건물 상호가 바뀔 경우 정보의 최신성이 유지되지 않는다는 단점이 있었습니다. 정보의 최신성을 유지하는 외부 API를 사용하여 기능을 구현하는 것이 합리적이라고 판단하여 주소 제공 기능은 외부 API를 사용하여 구현하게 되었습니다.

Cool Calendar - 일정 관리 어플리케이션

사용 기술 : Android 6.0 Marshmallow, SQLite

기간 : 2019.05.13 - 2019.05.18

Github : https://github.com/yyy9942/coolendar

Google Play : https://play.google.com/store/apps/details?id=kr.dj.hnu.multi.ccal

프로젝트 개요

외부 통신 없이 내부 DB를 사용하는 일정 관리 애플리케이션입니다. 안드로이드 스튜디오에서 제공하는 캘린더 API를 사용하지 않고 직접 Recycler View를 사용하여 달력을 구현하였습니다. 일정을 월, 주, 일별로 조회할 수 있습니다. 일정 추가 시 DatePicker를 사용하여 편의성을 고려하였습니다. 제작된 애플리케이션은 Google Play에 배포하여 서비스 중입니다.

비동기를 활용한 속도 개선

일정을 조회하기 위해서 DB와 연동하는 도중 이 연결을 동기적으로 진행하면 화면이 늦게 출력되는 이슈를 발견하여 어떻게 해결할지 고민하게 되었습니다. 찾아낸 원인은 DB에서 데이터가 오기까지 기다리는 동기 통신이 너무 오래 걸린다는 것이었고, 이 통신을 AsyncTask를 활용하여 비동기 통신으로 전환하고 속도 문제를 해결하였습니다.

자기소개서

개발을 시작한 이유

어린 시절부터 컴퓨터를 이용해 무언가를 만드는 것을 좋아하였습니다. Flash 게임을 제작하여 주전자 닷컴에 올려서 다른 사람이 플레이 해주는 것이 재미있었습니다. 이후 고등학교에 진학하며 정말 제대로 된 프로그램을 직접 만들어 보고 싶다는 생각을 하게 되었습니다. 툴을 이용하여 제작하는 것이 아닌 직접 코드를 타이핑하여 무언가를 만들어 보고 싶다는 생각을 하게 되었고 코딩과 더불어 이미지 제작과 각종 콘텐츠를 생산하는 방법을 배울 수 있는 멀티미디어공학 전공을 선택하여 공부를 시작하게 되었습니다.

개발을 실생활에 적용하기를 좋아합니다

한참 일본 여행이 유행할 때 '일본 여행을 가도 방사능 피폭 걱정을 정말 안 해도 될까?' 라는 의문이 생겼습니다.
일본측에서는 방사능이 거의 해결되었다고 하고 후쿠시마에도 멀쩡하게 사람이 살고 있었지만 제가 알고 있는 방사능은 그렇게 쉽게 사라지는 성질이 아니었기 때문입니다. 관련 자료를 찾아보니 방사능 수치 변화 데이터 자체는 많았지만 그 데이터가 너무 많아서 한 눈에 파악하기에는 어려움이 있었습니다. 그렇기 때문에 데이터를 시각화 하여 데이터의 변화를 측정할 수 있는 R을 사용하였고 각 사이트에서 일본 주요 도시의 방사능 수치 변화 데이터를 크롤링하여 차트로 표시하였습니다. 이 자료를 토대로 방사능 데이터 수치 변화를 분석하였고 '후쿠시마 원전 사고 이후 방사능 변화 분석을 통한 일본 여행 위험성 측정' 이라는 보고서를 제작하여 공유하였습니다.

성능 튜닝을 좋아합니다

성능을 개선하는 것은 얼마나 개선이 되었는지 수치로 확인할 수 있다는 장점이 있습니다. 성능을 개선하고 안정성을 향상시켜 탄탄한 프로그램을 제작하는 것이 재미있고, 개선에 성공하였을 때 수치적으로 개선된 것을 확인할 수 있어서 성능 이슈를 해결하는 것을 좋아하게 되었습니다.

그렇기 때문에 하나의 기능을 개발하더라도 조금 더 빠르고, 안정적인 코드를 작성하고 있습니다. 예를 들어 DelFood 프로젝트를 진행할 때 공공데이터에서 주소 데이터를 받아와 DB서버에 저장하고, 이를 직접 관리한 적이 있습니다. 이 때 주소 검색을 하면 DB에서 검색까지 약 12초가 넘는 시간이 걸렸습니다.

이를 개선하기 위해 주소 검색 조건에 맞도록 복합 인덱스를 설정하였습니다. 인덱스를 생성하고, 지우며 속도 측정을 진행하였고 빠른 속도를 내는 인덱스를 설정하였을 때의 검색 속도는 약 0.3초였습니다.
눈에 띄는 성능 개선을 이루었지만 조금 더 빠른 방법은 없을까 생각하였습니다. 프로그램의 로그 데이터를 모니터링한 결과 한번 검색된 주소를 여러 번 조회하는 것을 확인했습니다. 이를 통해 검색된 주소를 캐싱하였고 검색 속도는 0.1초 미만으로 줄어들게 되었습니다.

위 주소 검색 이외에도 푸시 메시지를 전송할 때 비동기 처리를 통한 대량 메시지 발송 속도 개선 등 여러 방면으로 성능과 안정성을 높이기 위해 노력하고 있습니다.

좋은 코드를 만들기 위해 노력합니다

코드의 질적 향상을 위하여 프로젝트에 사용되는 기술을 깊게 공부해보는 습관을 가지려고 노력하고 있습니다. 이전에 주먹구구식으로 만들던 방식에서 벗어나 체계적인 설계를 지향하기 위해 책을 기반으로 공부하려고 노력하고 실제 기술을 사용할 때에는 관련 도큐먼트를 찾아보기도 합니다. 실재로 Redis를 도입할 때 Redis Doc를 읽고 다른 기술들과 비교하여 장단점을 비교하며 도입을 진행하였습니다.

코드의 재사용성을 높이는 것을 좋아합니다. 중복되는 로직을 통합하여 메서드로 만드는 리팩터링을 자주 진행합니다. Code convention을 지켜 다른 사람이 읽기 편한 코드를 작성하고 적절한 주석을 사용하여 팀원이 내가 작성한 코드를 쉽게 재사용할 수 있도록 만들고 있습니다. 이러한 방식을 통해 저는 좋은 코드를 제작하기 위해 노력하고 있습니다.

가지고 있는 지식 공유를 즐깁니다

지식 공유를 좋아하는데 그 중 코드를 리뷰하는 것을 가장 좋아합니다. 다른 사람의 개발 사고방식을 보는 것이 재미있었기 때문입니다. 개발자가 어떤 것을 하고싶어 하는지, 어떻게 해결하고자 하는지 보고 리뷰하여 소통하는 것이 재미있었습니다. Github에 올라온 PR을 리뷰하며 버그가 발생할 수 있는 부분에 대해 코멘트를 남기기도 하고 더 좋은 방향으로 코드를 변경시킬 수 있는 점에 대해서 코멘트를 남겨 토론을 진행하기도 합니다.

혼자 공부하기 보다 커뮤니티를 만들어 공부하고, 공부한 내용에 대해 토론하는 것을 좋아합니다. 실제로 해당 방식으로 공부했을 때 혼자 공부하는 것 보다 더 재미있고 효율이 잘 나와서 학과 후배들과 코딩 튜터링을 운영하기도 하고 자격증, 개발, 알고리즘 등 스터디 그룹을 운영하며 다양한 활동을 이어 나가고 있습니다.

더 많은 사람들과 지식을 공유하기 위해 프로젝트를 제작하며 겪는 다양한 기술적인 이슈와 해결 방법을 개인 기술 블로그에 포스팅하며 운영하고 있습니다. 작성한 글을 페이스북 '생활 코딩'에 공유하기도 하며 동시에 피드백을 받고 있습니다.

이렇게 고민한 이슈들을 하나 둘 공유하다 보니 개발자 블로그 글을 모아 놓는 Awesome devblog라는 사이트에서 제 글을 크롤링하기 시작하였습니다. 이를 통해 같은 고민을 하는 개발자들과 댓글로 질의를 주고받기도 하며 지식 공유의 범위를 늘리고 있습니다.