portfolio.

최세은 | seeunchoi99@gmail.com

gihub.com/dahyen0o

sechoi.tistory.com

introduce.

- 안정적인 서비스를 위해 항상 고민합니다. 테스트를 통한 안정성 향상과 변화에 용이한 코드를 작성하는 것에 관심이 많습니다.
- 자동화를 통해 다양한 불편함을 해소해왔습니다.
  - 자동 TIL 봇 (github.com/dahyen0o/auto-TIL)
  - 출석/퇴실 체크 알림 봇 (github.com/dahyen0o/ssafy-attendance-notification-bot)

education.

서강대학교 컴퓨터공학과
졸업 | 2019.03 - 2023.02

awards.

서강대학교 캡스톤디자인2 우수 프로젝트 경진대회 최우수상
2022.02 | 현대모비스
🔗 https://github.com/ASAP-STAYA

project.

ZI9ZA9(지구재구)

🔗 https://github.com/wootecam-gugucon/shopping-mall

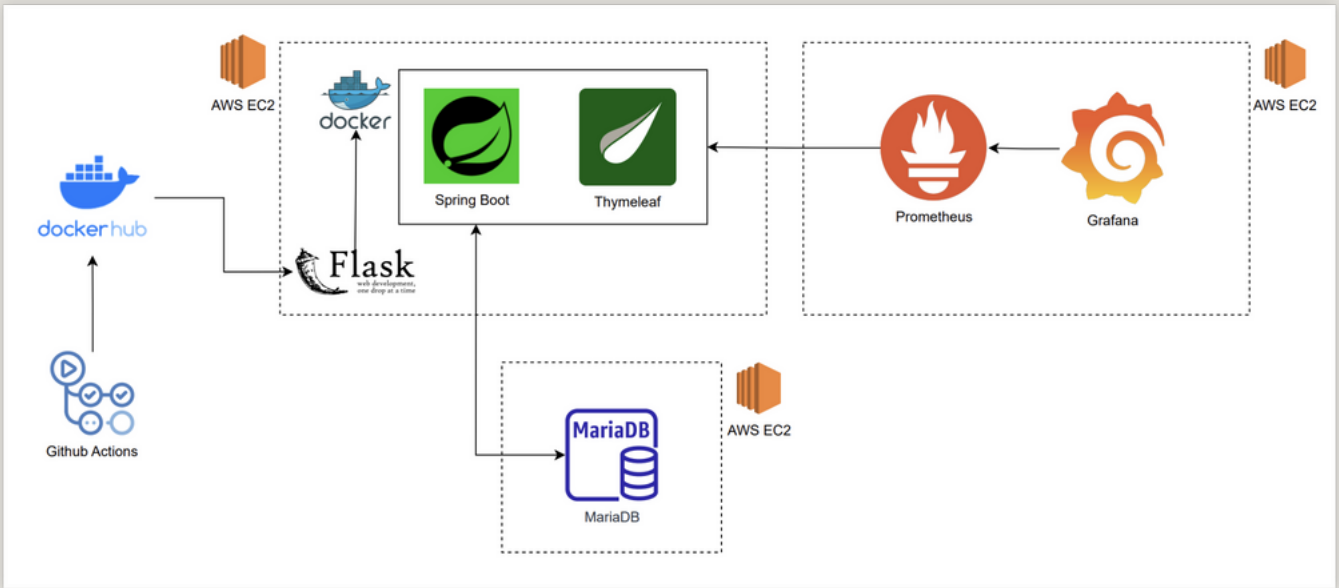
"온라인 의류 쇼핑몰"

- 우아한테크캠프 6기 팀 프로젝트
- 팀 구성: 백엔드 4명 (본인 포함)
- 개발 기간: 2023년 8월 (3주)

프로젝트 소개

- 주문 500만개, 주문 품목 1250만개, 별점 600만개 등 **대규모 데이터**가 존재
- **상품 검색** 시 다양한 조건을 적용
  - 구매자의 성별 및 나이대별 필터링
  - 주문 및 별점 순 정렬
- 상품 상세 조회 시 해당 **상품에 대한 통계 기능**을 제공
  - 함께 주문된 횟수가 많은 순으로 상품 추천
  - 성별 및 나이대별 평균 별점 제공
  - 접속한 회원과 같은 성별 및 나이대의 평균 별점 제공
- **두 가지 결제 방식**을 지원
  - 포인트 결제
  - 토스페이먼츠 결제 위젯 기반 간편 결제

시스템 아키텍처



상세 기여 및 성과

(1) 조회 쿼리 성능 향상 - Covering Index

| 상황

- 주문 데이터 500만 개, 주문 상품 데이터 1250만 개로 대규모 데이터가 존재
- 상품 검색 시, 쿼리를 사용해 **검색어를 이름에 포함하는 상품들을 주문 순으로 정렬**하는 기능 구현

```
select p.* from products p
    left join order_items oi on oi.product_id = p.id
where p.name like '%니트%'
group by p.id
order by sum(oi.quantity) desc
```

| 문제점

1250만 개의 주문 상품 데이터를 JOIN으로 조회하고, 검색어에 LIKE 연산을 사용하므로 슬로우 쿼리로 예상
실제 로컬 MariaDB 서버 기준 '니트'로 검색하면 **48s** 소요

| 해결 과정

```
+--+-----------+-----+----------+------+-------------+-------+-------+---------------------+-------+--------+-----------------------------+
|id|select_type|table|partitions|type  |possible_keys|key    |key_len|ref                  |rows   |filtered|Extra                        |
+--+-----------+-----+----------+------+-------------+-------+-------+---------------------+-------+--------+-----------------------------+
|1 |SIMPLE     |oi   |null      |ALL   |null         |null   |null   |null                 |2487090|100     |Using temporary; Using filesort|
|1 |SIMPLE     |p    |null      |eq_ref|PRIMARY      |PRIMARY|8      |shopping.oi.product_id|1      |11.11   |Using where                  |
+--+-----------+-----+----------+------+-------------+-------+-------+---------------------+-------+--------+-----------------------------+
```

- explain 명령어로 **실행 계획**을 출력 후 분석
  - order_items에 연산에 사용되는 인덱스가 없음
  - 따라서 GROUP BY 연산을 위해 가상 테이블을 생성(using temporary)하고 정렬(using filesort)
- order_items 테이블에 외래 키 대신 product_id에 인덱스 추가 → **1.7s**로 성능 향상
  - 외래 키 제약 조건을 추가하면 조회 시 관련 테이블에 대해 공유 읽기 전용 잠금(shared read-only lock)을 수행하기 때문에 부모-자식 테이블 간 잠금이 전파되어 성능이 저하됨
- order_items 테이블에 복합 인덱스(product_id, quantity)를 추가해 **Covering Index**를 사용 → **24ms**로 성능 향상

(2) 조회 쿼리 성능 향상 - 역정규화

| 상황

- 상품 검색 시 필터링 조건으로 주문자의 성별 및 나이대가 추가

```
select p.* from products p
    left join order_items oi on oi.product_id = p.id
    left join rates r on r.order_item_id = oi.id
    left join orders o on oi.order_id = o.id
    left join members m on m.id = o.member_id
where m.gender = 'FEMALE'
    and m.birth_date >= str_to_date('1998-01-01', '%Y-%m-%d')
    and m.birth_date <= str_to_date('2000-12-31', '%Y-%m-%d')
    and p.name like '%니트%'
group by p.id
order by avg(r.score) desc
```

| 문제점

- 1250만 개의 주문 상품 데이터, 600만 개의 별점 데이터, 500만 개의 주문 데이터를 모두 JOIN으로 가져옴
  - 실제 로컬 MariaDB 서버 기준 '니트'로 검색하면 **48s** 소요
- (1)처럼 Covering Index을 추가해 성능을 향상해도 **3.8s** 소요 → 슬로우 쿼리
- 별점이 높은 순으로 정렬한 상품을 조회하는 쿼리도 마찬가지로 Covering Index를 적용하고도 약 **3s** 소요

| 해결 과정

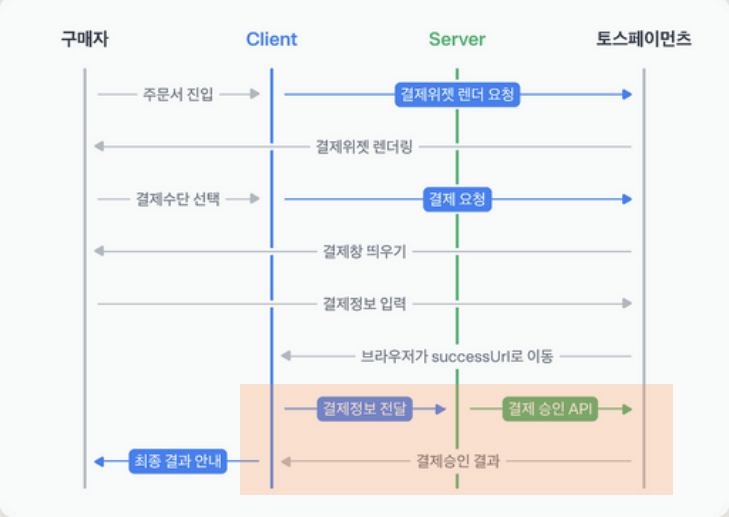
- 조회 성능을 향상하기 위해 **역정규화** 사용
  - 하나의 행에 주문자의 성별과 나이대마다 주문 수를 저장하는 주문 통계 테이블 생성
  - 하나의 행에 리뷰어의 성별과 나이대마다 별점 수를 저장하는 별점 통계 테이블 생성
  - 각각 주문 및 리뷰가 완료되었을 때 동기적으로 통계 테이블 업데이트
- 위 쿼리를 비롯한 **모든 조회 쿼리 실행 시간이 15ms 이하**로 크게 성능이 향상됨

(3) 결제 완료 시 재고 차감 최적화

🔗 관련 포스팅: https://sechoi.tistory.com/35

| 상황

- 토스페이먼츠 결제위젯을 통한 간편 결제 지원



| 문제점

- 결제가 승인될 때 재고가 부족해서 결제가 취소되는 경우가 없어야 하므로, 하나의 트랜잭션에서 상품 테이블(products)의 재고 컬럼에 **베타 락(X lock)**을 획득
- 로컬 환경(램 16기가, Apple M1)에서 가상 유저 99명이 동시에 접속할 때 **평균 TPS가 31.9** 로 병목이 발생

| 해결 과정

- 실제 영화 예매 서비스 참고
  - 좌석을 선택하고 **결제를 요청한 순간** 해당 좌석을 일정 시간동안 점유
  - 결제를 완료하지 않으면 일정 시간 뒤 좌석의 점유가 해제
- 결제 승인 API의 상품 테이블 베타 락 획득 로직을 결제 요청 API로 이동
  - TPS 100으로 병목을 제거
- 미완료 주문을 주기적으로 취소하는 **주문 취소 스케줄러** 생성
  - 30분 동안 아무런 업데이트가 일어나지 않는 주문 데이터 업데이트
  - 결제가 시작된 주문을 취소할 때 **새로운 물리적 트랜잭션을 생성**해 재고를 신속하게 복구

```
@Transactional
public void cancelIncompleteOrders() {
    final LocalDateTime scanStartTime = lastScanTime == null ? DEFAULT_SCAN_START_TIME : lastScanTime;
    final LocalDateTime scanEndTime = LocalDateTime.now().minus(CANCEL_INTERVAL);

    // 비관적 락으로 취소할 주문들 조회
    final List<Order> incompleteOrders = orderRepository.findAllIncomplete();

    boolean allSucceeded = true;
    for (Order incompleteOrder : incompleteOrders) {
        final boolean succeeded = cancelIncompleteOrders(incompleteOrder);
        if (!succeeded) {
            allSucceeded = false;
        }
    }

    // 모든 취소에 성공하면 lastScanTime 업데이트
    if (allSucceeded) {
        lastScanTime = scanEndTime;
    }
}
```

Floney(플로니)

🔗 https://github.com/Floney-2023
🍎 iOS 앱 다운로드: https://apps.apple.com/kr/app/%ED%94%8C%EB%A1%9C%EB%8B%88-floney/id6462989500

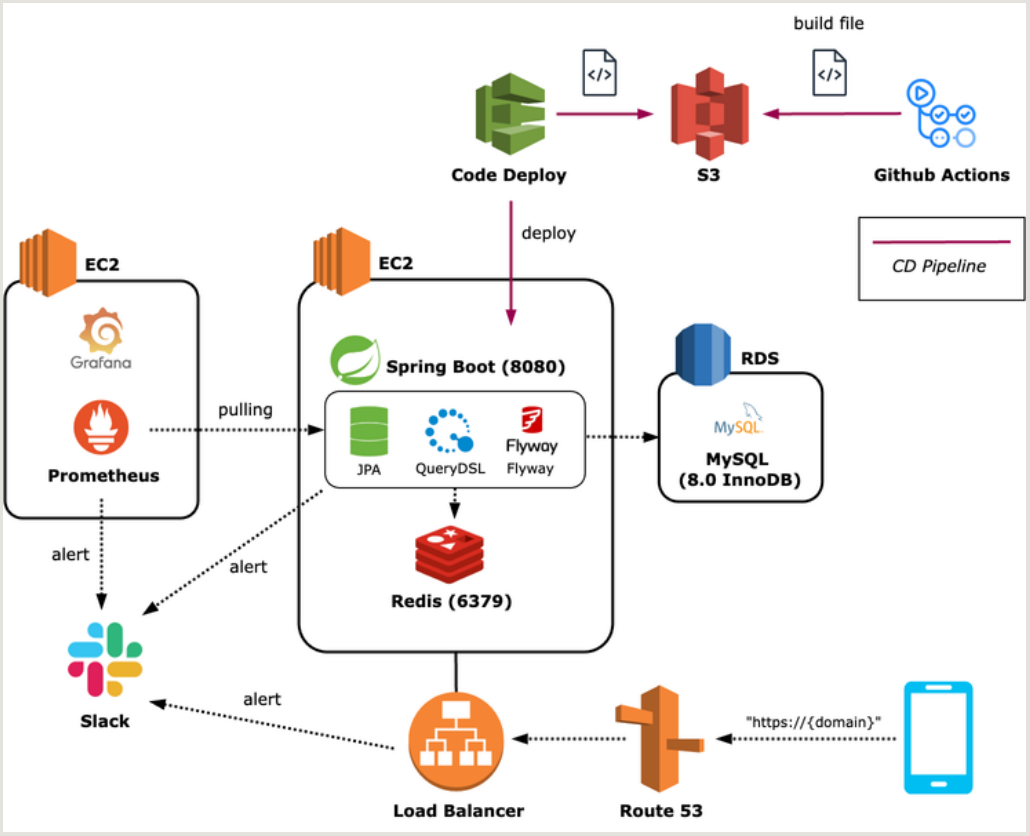
"함께 기록하는 공유가계부 앱"

- 팀 구성: 기획/디자인/운영 2명, iOS 1명, 안드로이드 1명, 백엔드 3명 (본인 포함)
- 개발 기간: 2023년 4월-진행 중

프로젝트 소개

- 캘린더·일별로 기록한 내역 조회
  - 공동 작성자의 기록과 일별 지출, 수입을 조회
  - 캘린더에서 월 별 총 지출, 총 수입을 확인
- 가계부 내역 추가
  - 지출·수입·이체 총 3가지의 내역 목록 중 하나를 선택해서 내역을 추가/수정/삭제
  - 매일/매주/매달 등을 기준으로 반복 내역 추가 가능
- 지출·수입·예산·자산에 대해 가계부 마다 분류 항목을 관리
- 지출·수입·예산·자산의 총 4가지 항목에 대한 분석 그래프 제공
- 가계부 내역들을 바탕으로 정산 가능

시스템 아키텍처



상세 기여 및 성과

(1) 데이터베이스 동시 INSERT 및 UPDATE 해결

🔗 관련 포스팅: https://sechoi.tistory.com/31

| 상황

- 자산 테이블에 {가계부, 년도-달}을 컬럼으로 데이터를 저장
- 가계부 내역 추가 시, 관련 자산 데이터 60개(5년)에 대해 {가계부, 년도-달}이 있으면 업데이트, 없으면 생성

```
// 가계부 내역의 money를 자산에 추가
public void createAsset() {

    for (int i = 0; i < FIVE_YEARS; i++) {
        Optional<Asset> asset = assetRepository.find(날짜, 가계부);

        if (asset.isEmpty()) {
            Asset newAsset = new Asset();
            assetRepository.save(newAsset);
        } else {
            asset.get().update();
        }
    }
}
```

| 문제점

- 트랜잭션 격리 수준이 Repeatable Read일 때 동시에 자산 데이터를 조회 후 업데이트하면 **갱신 손실** 발생
- 동시에 자산 데이터를 추가하면 {가계부, 년도-달}이 같은 중복 데이터 발생

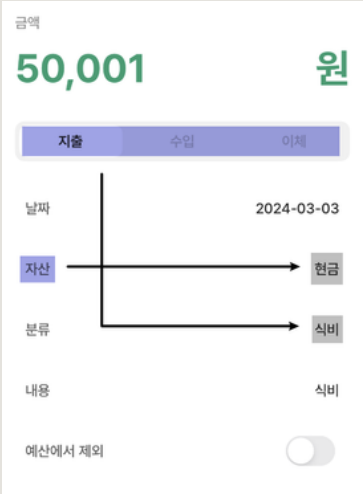
| 해결 과정

- {가계부, 년도-달}로 **유니크 복합 인덱스** 추가
- 조회 후 삽입 혹은 수정하던 로직을 MySQL 문법(INSERT ... ON DUPLICATE KEY UPDATE)을 사용해 하나의 쿼리(atomic query)로 동작하도록 변경

```
insert into asset (date, money, book_id) values (:date, :money, :book)
    on duplicate key update money = money + :money, updated_at = now()
```

(2) 카테고리 관련 리팩토링

🔗 관련 포스팅: https://sechoi.tistory.com/34



| 상황

- 가계부 내역 추가 시 3가지 관련 카테고리{내역 종류(지출/수입/이체), 하위 분류, 자산 분류}를 참조
- 하위 분류 및 자산 분류 항목은 삭제가 불가능한 기본 분류 항목과, 삭제가 가능한 커스텀 분류 항목으로 구분
- **JPA 상속 전략(SINGLE_TABLE)**을 사용해 카테고리 도메인을 구현
  - 어플리케이션에서 기본 카테고리 객체를 상속한 여러 종류의 카테고리 객체가 존재
  - 데이터베이스 상에서는 하나의 테이블에 dType으로 카테고리 종류를 구분

| 문제점

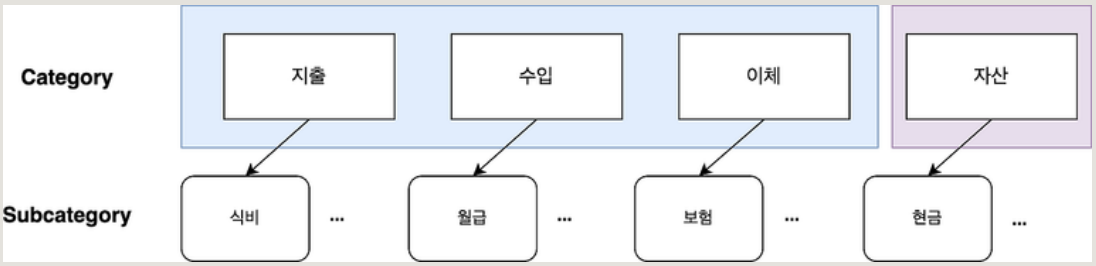
- QueryDSL의 QueryProjection을 사용할 때 컬렉션으로 여러 개의 가계부 내역-카테고리 연결 데이터를 한 번에 가져오지 못하는 현상 발생

- 카테고리는 {가계부, 상위 카테고리, 이름}을 기준으로 데이터가 유일해야 하지만 동시적으로 INSERT가 발생하면 해당 제약 조건이 위반됨
- 하위 분류 항목을 조회할 때 기본 분류 항목과 커스텀 분류 항목 각각에 대해 별개의 조회 쿼리를 실행하며 N*2 문제 발생

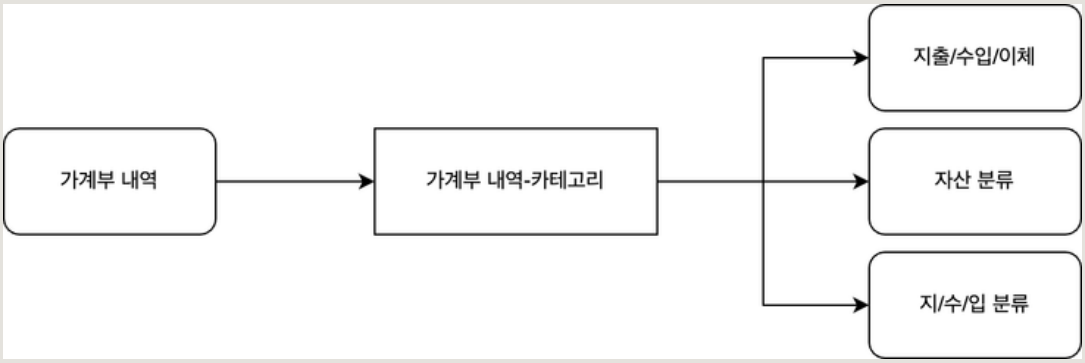
| 해결 과정

[재설계]

- JPA 상속 관계를 없애고 별개의 객체로 여러 종류의 카테고리 객체를 관리
- 기본 분류 항목과 커스텀 분류 항목을 같은 객체(Subcategory)로 저장
  - 하나만 생성해 여러 가계부가 공유하던 기본 분류 항목 데이터를 가계부마다 생성하도록 변경



- 가계부 내역-카테고리 연결 테이블 구조 변경
  - 가계부 내역이 3개가 아니라 1개의 연결 데이터를 참조
  - 연결 테이블은 3개의 카테고리를 컬럼으로 묶어서 참조



[구현]

- 테스트에 @Nested를 사용한 **DCI 패턴**을 도입
  - 체계적인 테스트 코드 작성으로 대규모 코드 변경에 유연하게 대처
- Flyway를 사용해 기존 데이터 마이그레이션 진행 (스크립트 링크)
  - 가계부 내역-카테고리 연결 테이블(book_line_category)는 참조하는 카테고리 데이터의 식별자가 변경되었으므로 그에 맞게 마이그레이션 진행
  - 어플리케이션 서버 시작 시 Flyway를 통해 자동으로 실행되고 결과를 확인하며 편리하고 빠르게 진행