
SHOP & HEALTH COMMUNITY
E S
T D
2 0
2 3
#FIT

발표 목차

01

프로젝트 개요

- 팀원 소개 / 주제 선정 사유

- 발표자 : 오연수

01 팀원 소개

팀장 / ISFP

진동선

https://github.com/DongSeonJin

팀원 / INFP

권아영

https://github.com/KwonAYeong

팀원 / INFJ

오연수

https://github.com/lilylemonoh

팀원 / INFJ

한국인

https://github.com/HANKUGIN

02 주제 선정 사유

#FIT(샵피트)
= shop + fitness

최근 많은 사람들이 건강한 삶을 추구하기 위해 운동, 올바른 식단, 헬스 정보에 대한 관심을 가지고 있습니다.

이 정보를 한번에 제공하는 사이트의 필요성을 느끼고 헬스 관련 다양한 기능을 제공하는 웹 애플리케이션을 개발하였습니다.

 # 커뮤니티

회원들 간의 운동 경험 및 지식 공유 촉진
목표 달성에 대한 동기부여 제공

 # 쇼핑

다양한 식품 및 운동 장비 구매 가능
제품 소개와 리뷰 제공

 # 뉴스

최신 헬스 및 웰빙 정보 제공
피트니스 대회 관련 정보 제공

02

분석 / 설계

- 요구사항 정의서 / 화면 설계 / ERD / 협업 방식(지라, 깃플로우, 슬랙, 노션) / API명세서

발표자 : 오연수

01

요구사항 정의서

요구사항 정의서는 프로젝트의 개발에 필요한 기능명, 처리내용(상세설명), 입력자료, 출력자료 등을 구체적으로 명시한 문서입니다.

요구사항정의서

시스템명 : 샵피트 작성일 : 2023년 7월 21일 ~ 2023년 8월 3일 작성자 : 권아영 진동선 오연수 한국인

요구사항명	요구사항 ID	기능명	처리내용 (상세설명)	(사용자) 입력자료	출력자료
회원가입	R1000	비회원	비회원 사용자는 회원 가입 및 조회만 가능하다. 조회는 쇼핑몰, 커뮤니티, 뉴스, 전체 검색 기능이 가능하다.		
	R1001	회원가입 (일반)	이메일 인증, 비밀번호 확인, 이름, 닉네임 중복 체크(필수), 주소 기입, 프로필 사진 업로드의 과정을 거친 후 회원가입을 할 수 있다. 프로필 사진 업로드 제외한 모든 항목 필수 입력.	이메일*, 비밀번호*, 이름*, 닉네임*, 프로필사진(선택)	"회원가입이 완료되었습니다." 알림 후 메인페이지 이동
	R1002	간편 회원가입 (네이버, 카카오)	인증 절차 없이 닉네임만 입력 후 회원가입 가능.	닉네임	"회원가입이 완료되었습니다." 알림 후 메인페이지 이동
	R1003	~~사업자 회원가입 (일반)~~			
	R1004	~~관리자 회원가입~~			
로그인	R1005	로그인 (일반 회원)	입력한 이메일과 패스워드에 해당하는 회원정보가 있는 경우 로그인 처리.	이메일, 비밀번호	
	R1006	로그인 (네이버,카카오 회원)		닉네임	
	R1007	~~아메일 찾가~~	~~아름과 휴대전화 인증을 통해 아메일 찾가~~	~~아름, 휴대전화번호~~	~~아메일 주소~~
	R1008	비밀번호 찾기 (재설정)	이메일 인증 후에 새 비밀번호로 재설정.	이름, 이메일, 인증번호, 새 비밀번호	
로그아웃	R1009	로그아웃	로그인 상태의 계정을 로그아웃.		로그인 화면(메인 페이지) 출력
회원 탈퇴	R1010	회원탈퇴	상세정보 페이지에서 탈퇴 버튼 클릭 시 탈퇴 페이지로 이동 후 비밀번호 입력 후 회원 탈퇴 **(soft delete)**	비밀번호	
마이 페이지	R1011	내 정보 보기	본인의 정보를 확인 할 수 있다.	인바디 올리기(기능 구현 여부에 따라)	프로필 사진, 닉네임, 쿠폰, 포인트, 주문내역, 찜 목록, 장바구니, 인바디
회원 정보 수정	R1012	회원 정보 수정 (일반/네이버)	닉네임, 비밀번호, 프로필 이미지를 수정할 수 있다. (비밀번호 변경 시에는 현재 비밀번호를 확인받아야 한다.)	닉네임, 비밀번호, 프로필사진	
주문 내역	R1013	주문 내역	본인의 주문 내역을 조회할 수 있다.		회원 아이디(닉네임), 결제 항목 별 상품명, 결제 금액, 결제 날짜, 배송 상태
	R1014	~~결제 수단 관리~~	~~결제 수단을 등록할 수 있다~~	~~카드 별칭, 카드 번호,~~	
마이페이지 - 나의 활동	R1015	내가 쓴 글 목록 보기	작성한 모든 글 목록을 조회할 수 있다.		작성한 글 카테고리, 제목, 날짜, 좋아요, 조회수
	R1016	내가 쓴 글 내용 보기	작성한 글 목록에서 글 클릭 시 글의 내용을 볼 수 있다.		작성한 글 카테고리, 제목, 닉네임, 글 내용, 조회수, 좋아요, 댓글 목록
	R1017	내가 좋아요 누른 글 목록 보기	내가 좋아요 누른 모든 글의 목록을 조회할 수 있다.		좋아요 누른 글 카테고리, 제목, 글쓴이, 날짜, 좋아요, 조회수
	R1018	내가 좋아요 누른 글 내용 보기	좋아요 누른 글 목록에서 글 클릭 시 글의 내용을 조회할 수 있다.		좋아요 누른 글 카테고리, 제목, 글쓴이, 글 내용, 조회수, 좋아요, 댓글목록

요구사항정의서

시스템명 : 샵피트 작성일 : 2023년 7월 21일 ~ 2023년 8월 3일 작성자 : 권아영 진동선 오연수 한국인

요구사항명	요구사항 ID	기능명	처리내용 (상세설명)	(사용자) 입력자료	출력자료
	C1005	관리자 권한 글 삭제	관리자 권한을 가진 사람은 모든 글을 삭제할 수 있다.		
	C1006	조회수	게시글 누르면 조회수 +1. 한 사람이 여러번 눌러도 다 반영됨.		
	C1007	좋아요	좋아요 버튼 누르면 좋아요 +1. 다시 누르면 취소 한 게시글에 여러번 반영 X	좋아요(선택)	
커뮤니티 (댓글)	C1008	댓글 등록	회원은 커뮤니티 내의 글에 댓글을 작성할 수 있다.	댓글 내용	댓글 목록 새로고침
	C1009	댓글 수정	본인이 작성한 댓글을 수정할 수 있다.	댓글 내용	댓글 목록 새로고침 (수정된 날짜/시간)
	C1010	댓글 삭제	본인이 작성한 댓글을 삭제할 수 있다.		댓글 목록 새로고침
	C1011	관리자 권한 댓글 삭제	관리자 권한을 가진 사람은 모든 댓글을 삭제할 수 있다.		
쇼핑 (상품 등록)	~~S1000~~	~~메인 페이지~~	~~제품 전체 목록~~	~~제품 전체 목록~~	
	S1001	관리자 권한 상품 업로드	관리자 권한, 관리자는 상품 목록에서 상품 등록 버튼을 눌러서 상품을 등록할 수 있다.	상품명, 사진, 가격, 상세설명, 카테고리, 상품 수량	
	S1002	관리자 권한 상품 삭제	관리자 권한, 관리자는 상품 상세페이지에서 상품 삭제 버튼을 눌러서 상품을 삭제할 수 있다.		
	S1003	관리자 권한 상품 수정	관리자 권한, 관리자는 상품 수정페이지에서 상품 수정 버튼을 눌러서 상품을 수정할 수 있다.	상품명, 사진, 가격, 상세설명, 카테고리, 상품 수량	
쇼핑	S1004	상품 상세 페이지	상품 상세 정보 및 리뷰를 조회할 수 있다.		상품명, 사진, 가격, 상세설명, 카테고리, 리뷰 내용, 별점, 리뷰 작성자, 리뷰 작성시간
	S1005	상품 구매	상품 및 수량 선택	구매수량	
	S1006	상품 결제	배송정보 기입, 결제수단 선택, 포인트 및 쿠폰 사용 선택	이름, 주소, 전화번호, 쿠폰(선택), 포인트(선택), 결제수단	
	S1007	장바구니	장바구니 목록을 조회하고, 상품과 수량 선택 후 구매 페이지로 이동 가능하다. 장바구니에 여러 목록이 있을 때 선택해서 구매페이지로 넘어간 항목은 장바구니에서 삭제된다.	상품 선택, 수량	
	S1008	찜	상품 상세 페이지에서 찜 버튼을 누르면 찜을 할 수 있다. 찜 버튼을 다시 누르면 취소할 수 있다. 찜한 제품 리스트는 마이페이지에서 조회 가능	찜(선택)	
			주문 내역에서 배송 완료 된 제품		

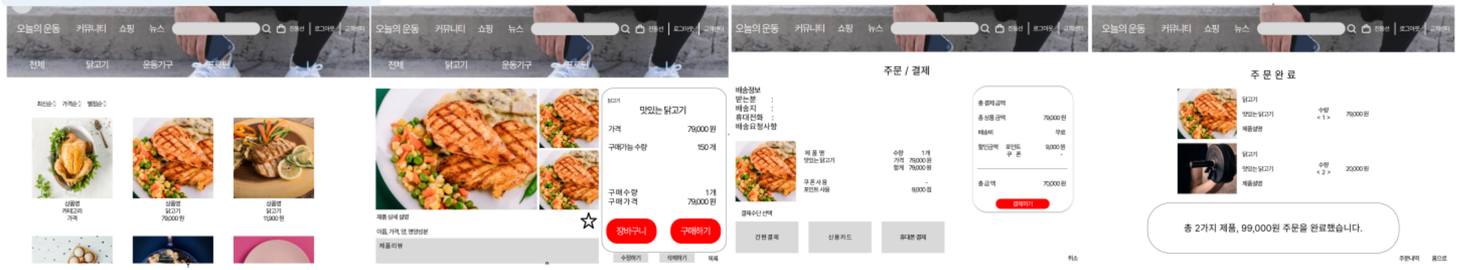
02

화면 설계

프로젝트를 본격적으로 구현하기 전, 필요한 기능을 피그마를 활용하여 임시 레이아웃으로 작성하였습니다.

메인 / 마이페이지

쇼핑



커뮤니티

03

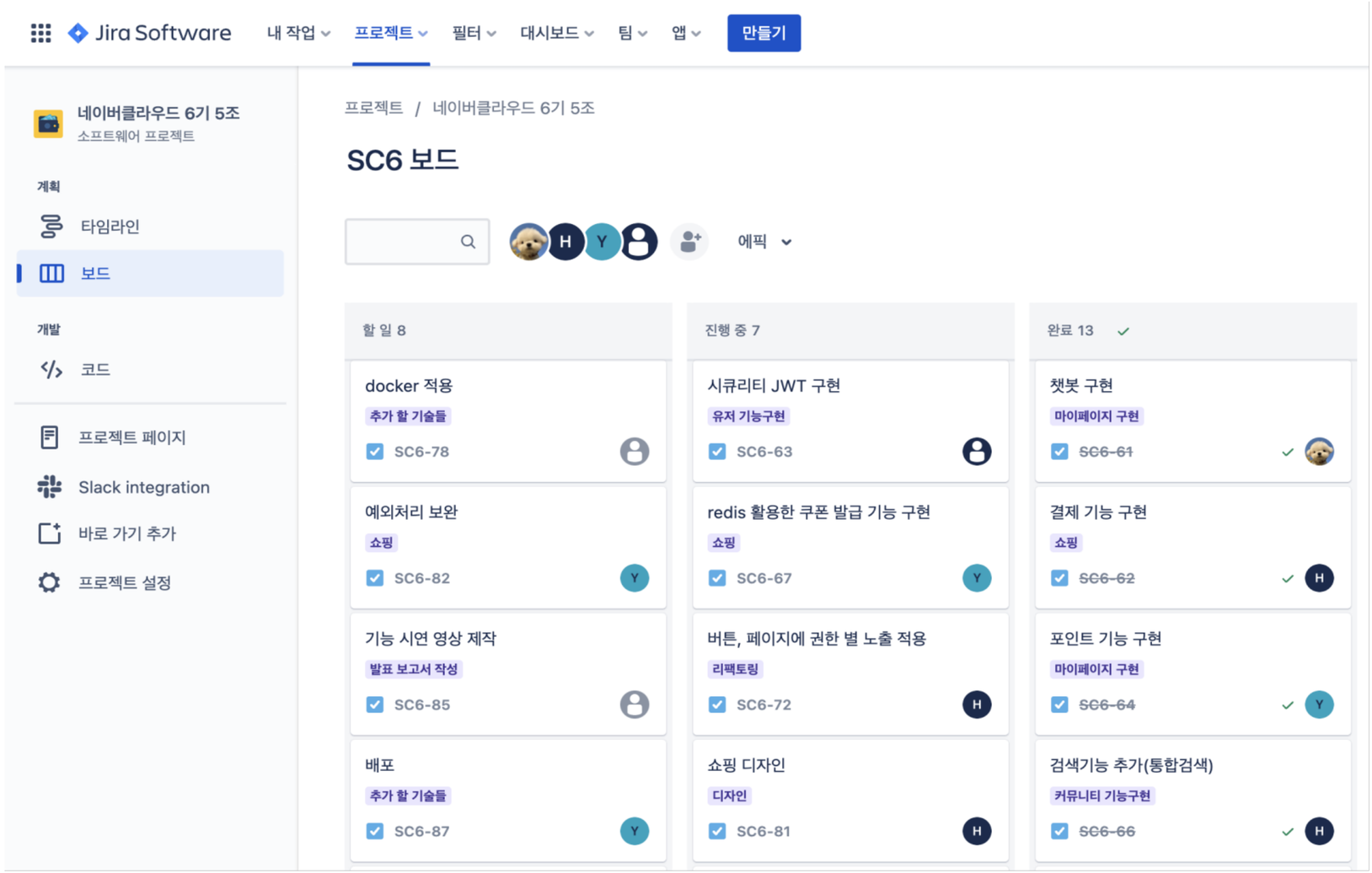
ERD

프로젝트의 본격적인 구현 전, ERD (엔티티-관계 다이어그램)를 통해 데이터베이스 테이블 구조를 설계하고, 각 테이블의 관계를 정의하였습니다.

04

협업 방식 (지라)

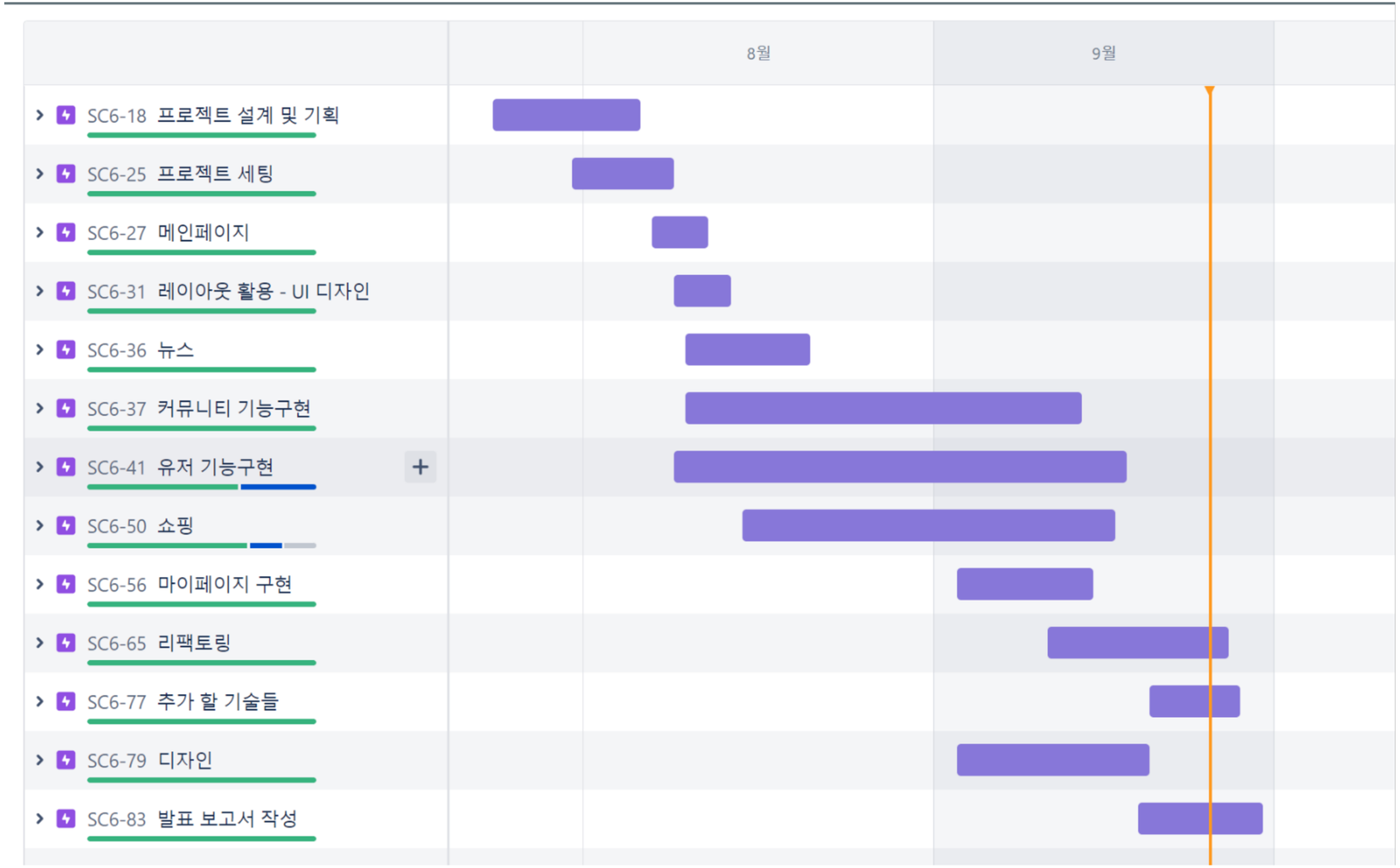
지라 보드에 해야 할 일들을 자세하게 작성합니다.



04

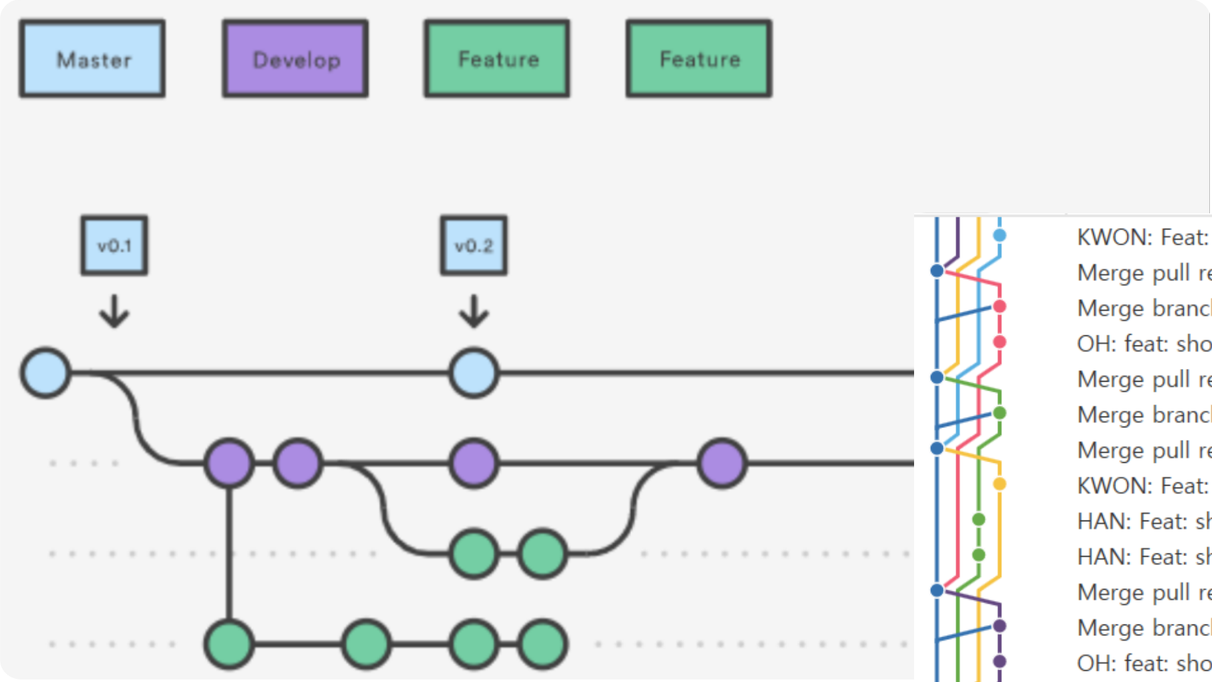
협업 방식 (지라)

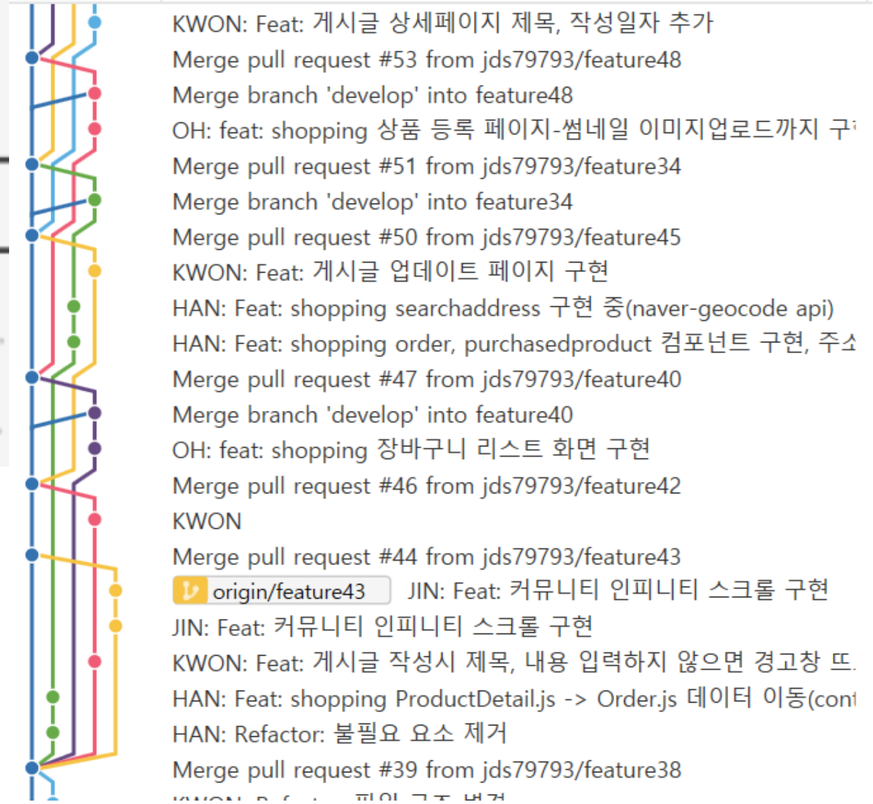
지라 타임라인으로 일정을 관리하고 공유합니다.



04

협업 방식 (깃 플로우)





지라에 등록한 이슈를 바탕으로
깃허브 이슈 탭에 등록합니다.
등록한 이슈 번호로 feature 브랜치를
생성해 작업하였습니다.

04

협업 방식
(깃 커밋 룰)

Git commit 규칙을 통해
일관된 커밋 메시지를 작성함으로써
코드 변경 이력의 가독성을 향상시켰습니다.

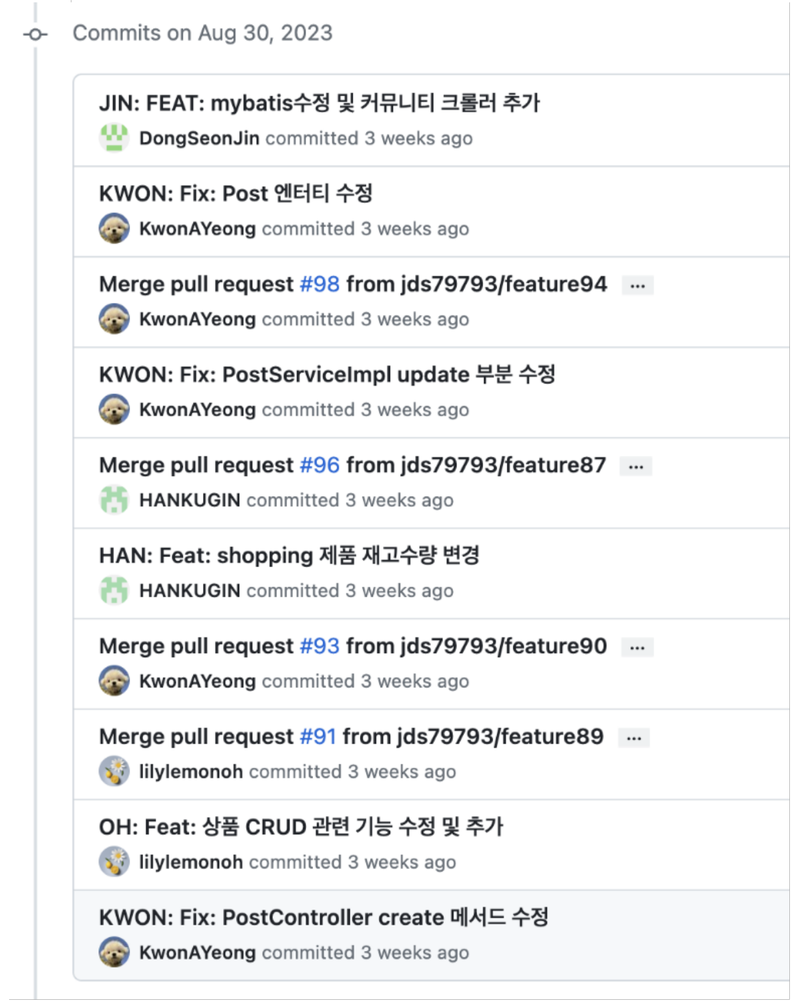
♠

git commit rule

- Commit Type

Feat:	새로운 기능 추가
Fix:	버그 수정 또는 typo
Refactor:	리팩토링
Design:	CSS 등 사용자 UI 디자인 변경
Comment:	필요한 주석 추가 및 변경
Style:	코드 포맷팅, 세미콜론 누락, 코드 변경이 없는 경우
Test:	테스트(테스트 코드 추가, 수정, 삭제, 비즈니스 로직에 변경이 없는 경우)
Chore:	위에 걸리지 않는 기타 변경사항(빌드 스크립트 수정, assets image, 패키지 매니저 등)
Init:	프로젝트 초기 생성
Rename:	파일 혹은 폴더명 수정하거나 옮기는 경우
Remove:	파일을 삭제하는 작업만 수행하는 경우

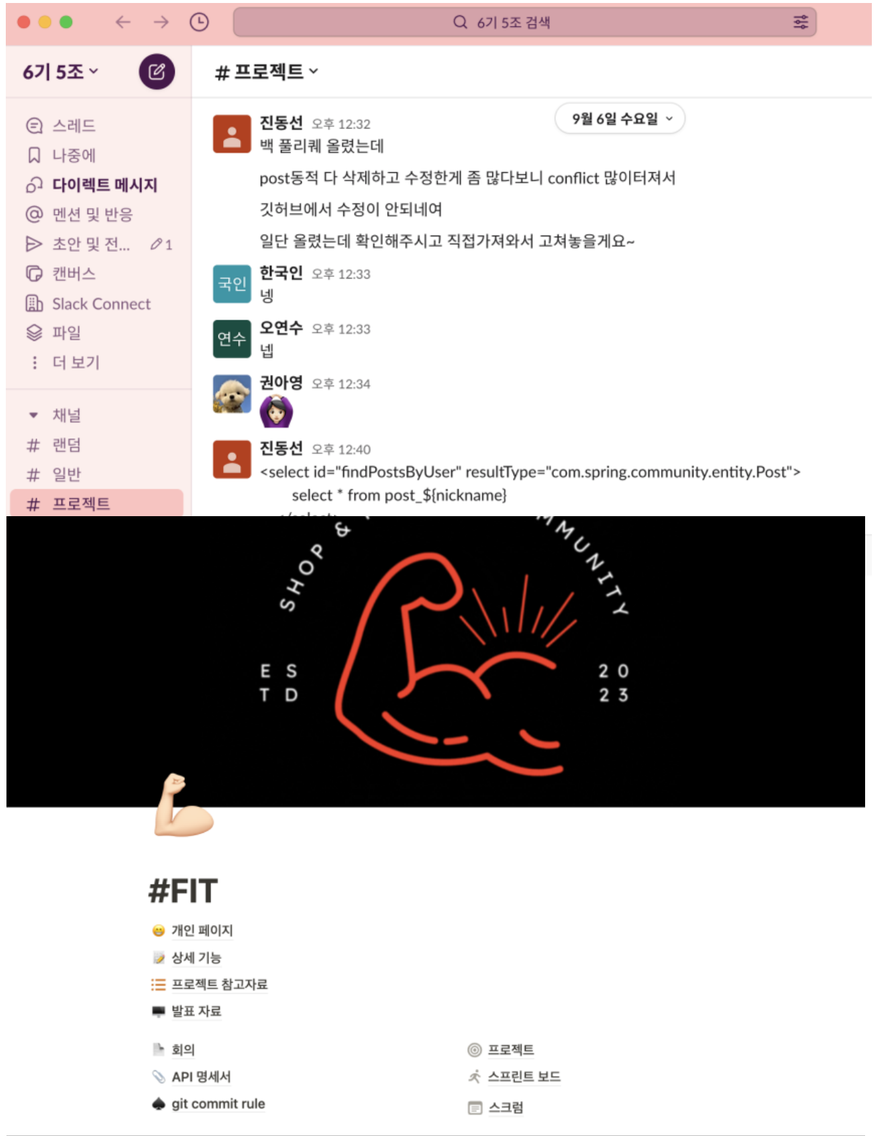
- "각자 이름 성 (대문자) : 작업 유형 (소문자) : 상세 작업 내용"
  - ex) OH : Feat : 뉴스 등록 기능 추가

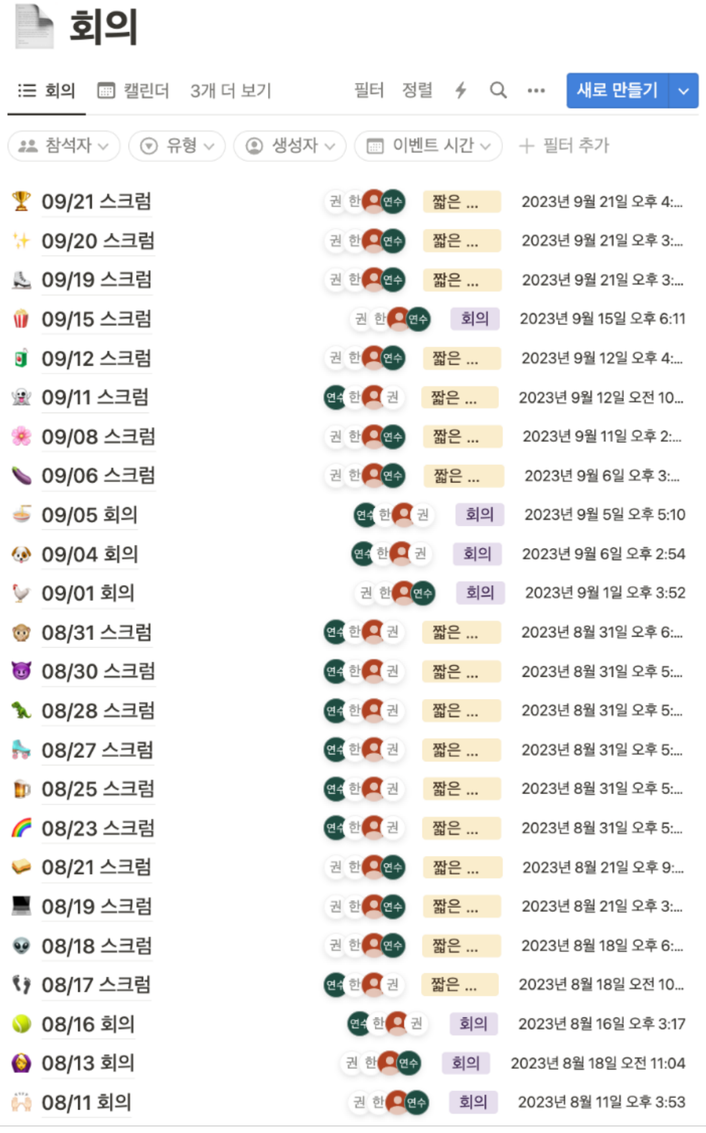


04

협업 방식 (슬랙, 노션)

슬랙을 통해 팀원간 커뮤니케이션,
노션을 통한 문서 공유 및 협업을
진행하였습니다.





05

API 명세서

프로젝트의 기능을 구현하기 위해 HTTP 메서드명, 엔드포인트 URL, 필요한 요청과 응답의 형식을 작성하였습니다.

API 명세서

권 댓글 추가

API + 필터 정렬 새로 만들기

메서드	요청주소	설명	상태코드	바디
	— HTTP 상태코드와 에러타입			
	커뮤니티			
GET	/post/list	모든 post를 조회합니다.	200 ok	List<POST> posts
GET	/post/list/{categoryId}	카테고리별 post를 조회합니다.	200 ok	list<PostListResponseDTO> posts
GET	/post/{postId}	postId를 받아 post를 조회합니다.	200 ok	Post post
POST	/post	post 정보를 받아 게시글을 저장합니다.	200 ok	"게시글이 저장되었습니다."
DELETE	/post/{postId}	postId를 받아 해당되는 post를 삭제합니다.	200 ok	
PATCH	/post/{postId}	post를 업데이트합니다.	200 ok	noContent
GET	/like/{postId}	좋아요 정보를 저장합니다	200ok	"좋아요 누르기 성공"
GET	/reply/{postId}/all	postId에 해당하는 모든 댓글 조회	200 ok	
GET	/reply/{replyId}	replyId에 해당하는 댓글 조회	200 ok	
POST	/reply	댓글 등록	200 ok	"댓글이 등록되었습니다."
DELETE	/reply/{replyId}	replyId에 해당하는 댓글 삭제	200 ok	"댓글이 삭제되었습니다."
PATCH	/reply/{replyId}	replyId에 해당하는 댓글 수정	200 ok	"댓글이 수정되었습니다."

메서드	요청주소	설명	상태코드	바디
	쇼핑-장바구니(Cart)			
POST	/cart	사용자의 장바구니에 아이템을 추가합니다.	201 Created	• userId: 사용자의 ID (파라미터) • productId: 상품의 ID (파라미터) • quantity: 추가할 수량 (파라미터)
DELETE	/cart/{cartId}	사용자의 장바구니에서 아이템을 제거합니다.	204 No Content	
GET	/cart/{userId}	사용자의 장바구니에 있는 아이템 목록을 가져옵니다.	200 OK	List<CartDTO>: 사용자의 장바구니 아이템 목록 (JSON 형식)
GET	/cart/checkCart	사용자의 장바구니에 이미 아이템이 있는지 확인합니다.	200 OK	Boolean : 있으면 true, 없으면 false
	쇼핑-주문(Order)			
POST	/orders	주문을 생성합니다.	201 Created	• userId : 사용자 ID(파라미터) • productId: 상품의 ID(파라미터) • totalPrice: 총 금액(파라미터) • address: 주소(파라미터) • phoneNumber: 핸드폰번호(파라미터)
GET	/orders/user/{use 열기	특정 사용자의 주문 목록을 가져옵니다.	200 OK	List<OrderDTO>: 사용자의 주문 목록 (JSON 형식)
GET	/orders/{orderId}	주문의 상세 정보를 가져옵니다.	200 OK	OrderDTO: 주문의 상세 정보 (JSON 형식)
PUT	/orders/{orderId}/status	주문의 상태를 업데이트합니다.	200 OK	• orderId: 업데이트할 주문의 ID (파라미터) • orderStatus: 업데이트할 주문의 상태 (파라미터)
	쇼핑-주문-상품(Order-Product)			
POST	/order-products	주문을 생성합니다.	201 Created	• orderId: 생성할 주문의 ID(파라미터) • productId: 생성할 주문 상품의 ID(파라미터) • quantity: 생성할 주문의 상품의 수량(파라미터)
GET	/order-products/order/{orderId}	특정 주문에 대한 주문 상품 목록을 가져옵니다.	200 OK	List<OrderProductDTO>: 주문의 주문 상품 목록 (JSON 형식)

03

시연 영상

시연 영상

발표자 : 한국인

04

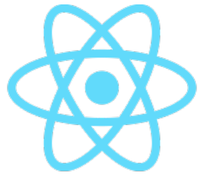
기술 소개 - 1

- 기술 스택 / CI/CD / TDD / API / 동시성 제어(비관적 락, 레디스)

발표자 : 오연수

01 기술 스택

프론트엔드



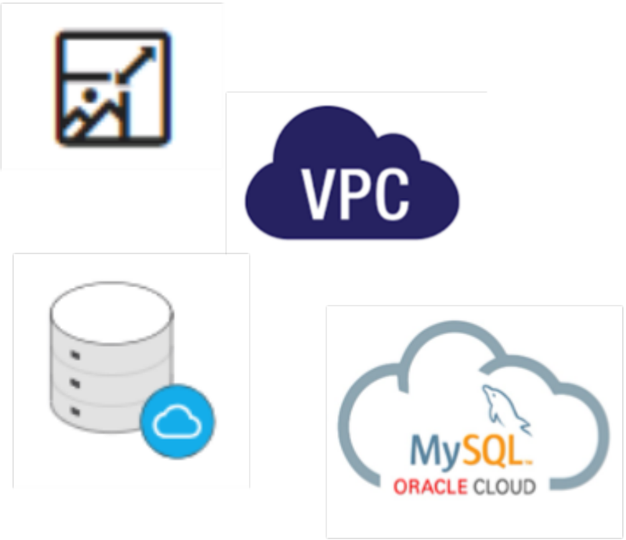
백엔드

01 기술 스택

TOOL

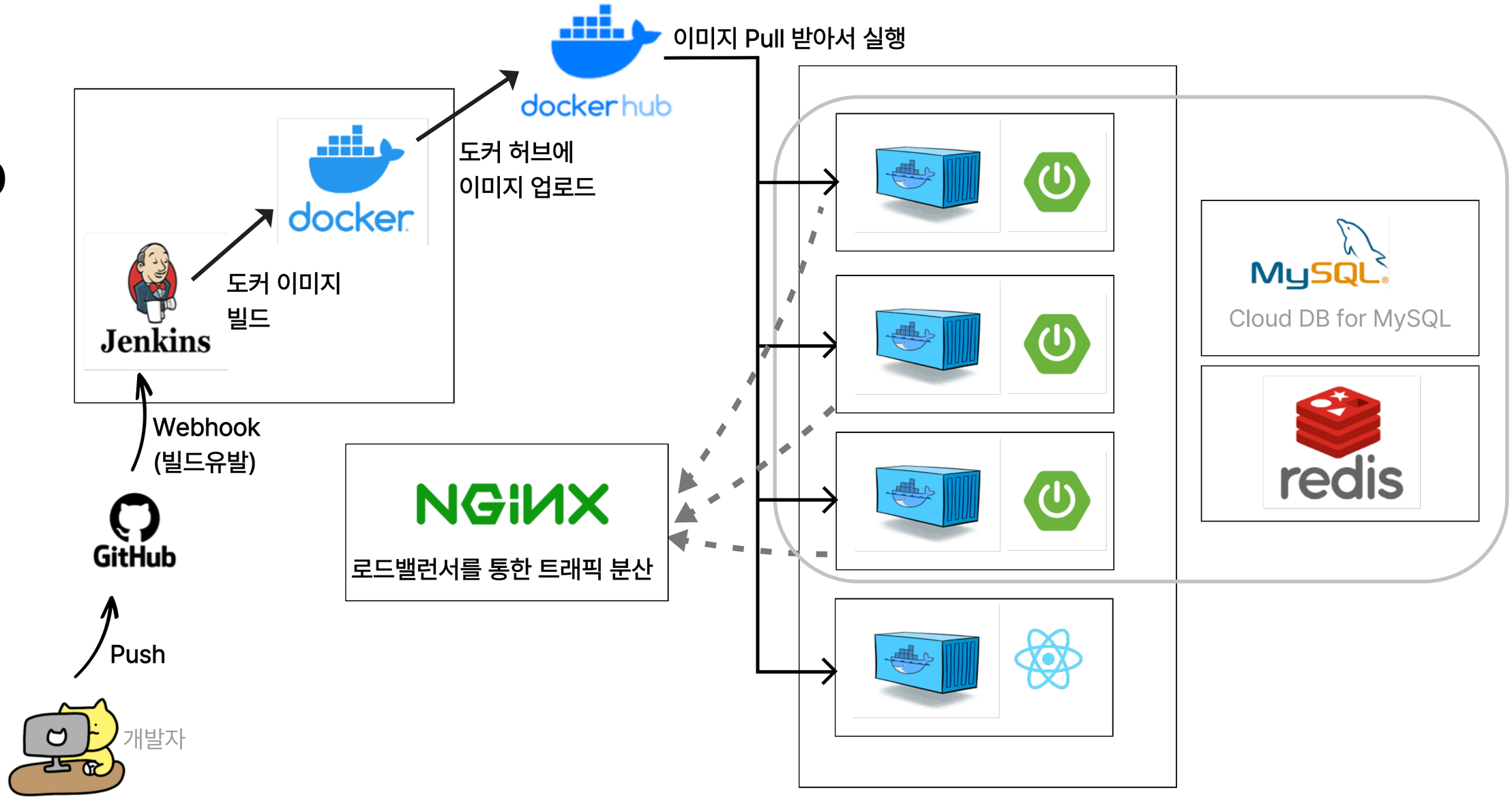
CI / CD

NAVER CLOUD



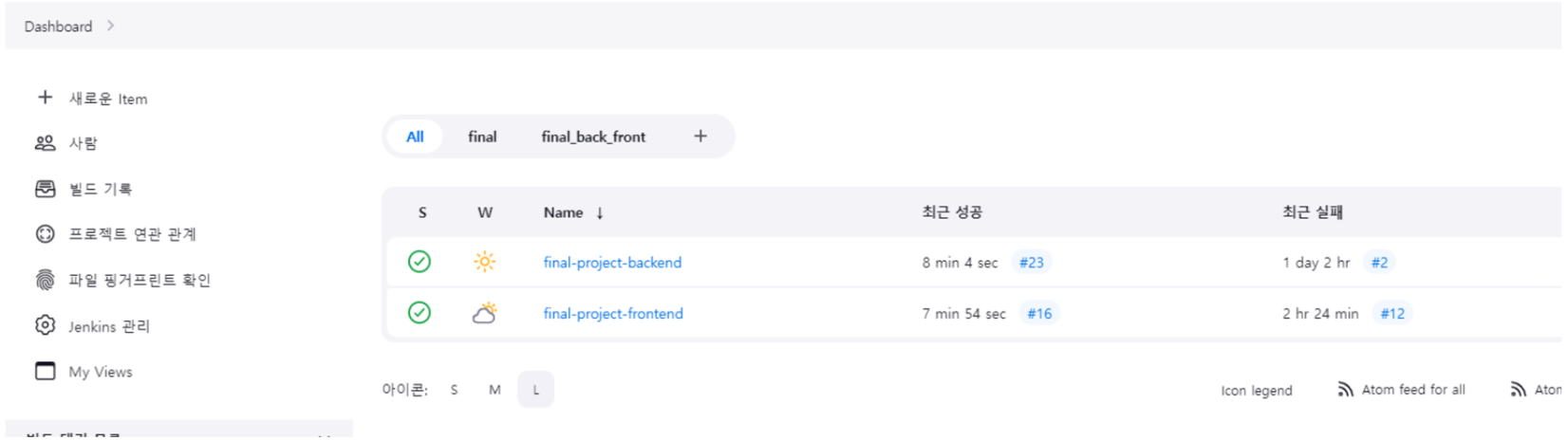
02

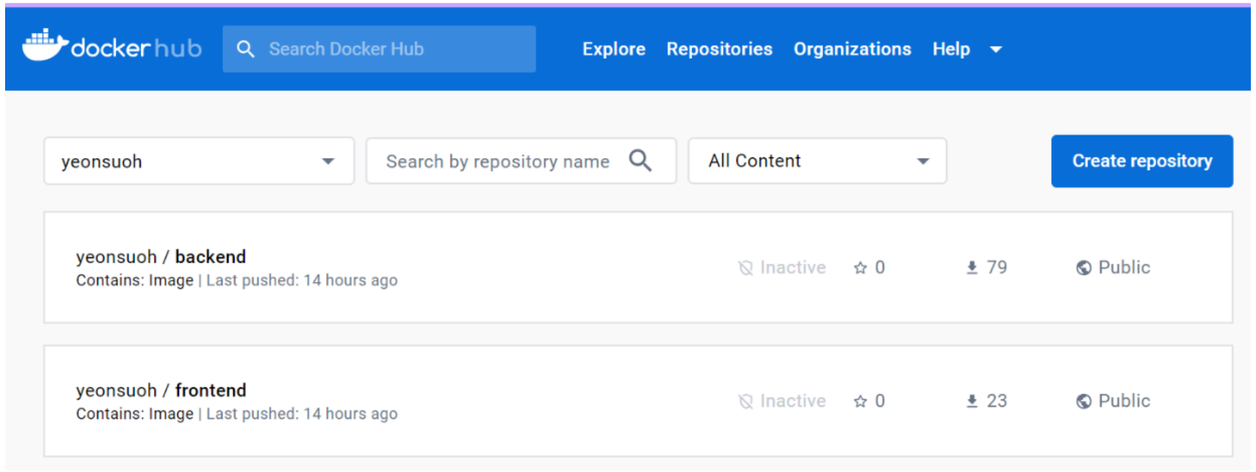
CI/CD


이미지 Pull 받아서 실행
dockerhub
docker
도커 허브에
이미지 업로드
Jenkins
도커 이미지
빌드
Webhook
(빌드유발)
GitHub
Push
개발자
NGINX
로드밸런서를 통한 트래픽 분산
MySQL
Cloud DB for MySQL
redis

02

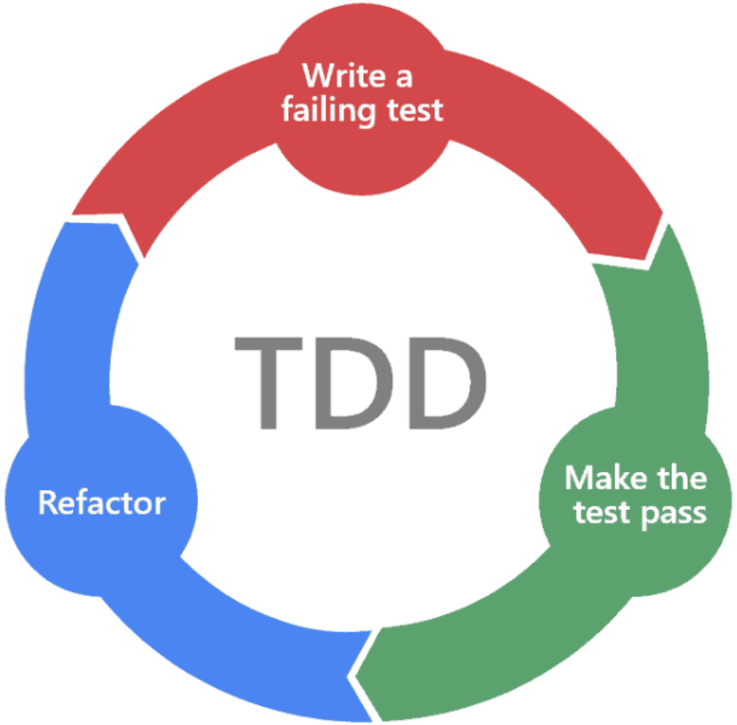
CI/CD





Nginx 로드밸런싱 환경 구축, Webhooks를 통한 자동 배포, 롤링 방식 무중단 배포를 구현하였습니다.

03 TDD (Test - Driven - Development)




```java
// 상품 1개 상세 조회 테스트
lilylemonoh
@Test
@Transactional
public void getProductDetailByIdTest() {
    // given: 테스트용 데이터베이스에 존재하는 상품의 ID fixture 세팅
    long productId = 1L;

    // when : 상품 id로 조회
    ProductDetailResponseDTO result = productService.getProductDetailById(productId);

    // then: 해당 상품이 예상한 값과 일치하는지 검증
    assertThat(result.getProductId()).isEqualTo(productId);
    assertThat(result.getProductName()).isEqualTo( expected: "찜닭"); // 상품명은 찜닭
    assertThat(result.getProductImageUrls().size()).isEqualTo( expected: 2); // 상품 이미지는 2장
}
```

04 API

naver cloud api

NAVER CLOUD PLATFORM

I'mport api

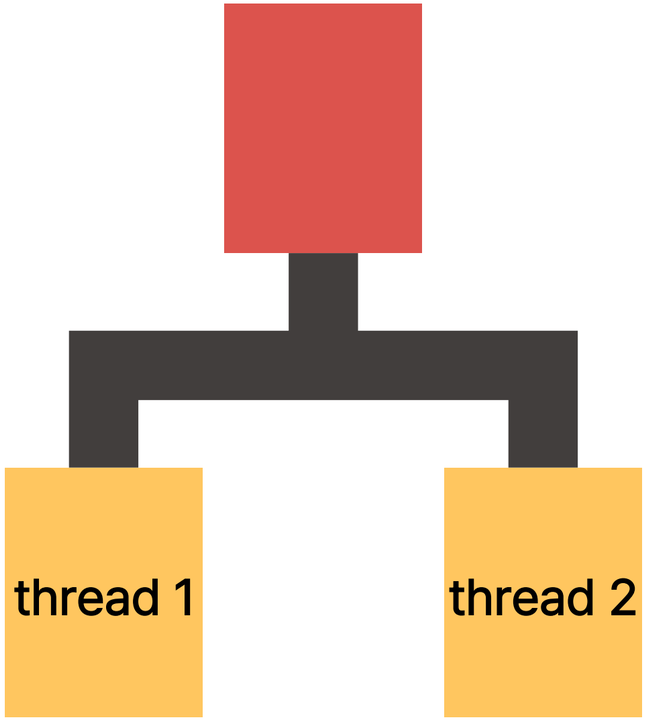
I'mport;

daum address api

Daum

05

동시성 문제 회피



예상되는 동시성 문제를 회피하고, 해결하기 위해 비관적 락과 레디스를 도입하였습니다.

레이스 컨디션 (Race Condition)

- 두 개 이상의 프로세스 혹은 스레드가 공유 자원을 서로 사용하려고 경합(Race)하는 현상
- 동기화 문제가 발생할 수 있다.

재고 관리 시스템

여러 사용자가 동시에 같은 상품을 구매하려 할 때, 실제 재고량과 데이터베이스의 재고량 사이에 불일치가 발생

선착순 이벤트 (쿠폰 발급)

제한된 개수의 쿠폰을 배포하는 경우, 동시에 많은 사용자가 쿠폰을 요청하면 제한 개수를 초과하여 쿠폰이 발급되는 문제 발생

05

동시성 문제 회피 : 비관적 락 (Pessimistic Lock)

예상되는 동시성 문제를 회피하고, 해결하기 위해 비관적 락과 레디스를 도입하였습니다.

```java
// 비관적 락 - 다른 트랜잭션과의 동시 접근을 허용하지 않고 데이터를 읽거나 수정하기 위해 사용되는 락(잠금) 적용하기
1 usage   ** lilylemonh
@Lock(LockModeType.PESSIMISTIC_WRITE)
@Query("select p from Product p where p.productId = :id")
Product findByIdPessimistic(@Param("id") Long id);
```

```java
@Override
@Transactional
public boolean updateProductStock(ProductStockUpdateRequestDTO requestDTO) {
    try {
        // 비관적 락(Pessimistic Lock) 적용하기
        Product product = productRepository.findByIdPessimistic(requestDTO.getProductId());

        // 상품의 재고(stock) 수량을 업데이트합니다.
        product.setStockQuantity(requestDTO.getStockQuantity());

        // 업데이트된 상품 정보를 저장합니다.
        productRepository.save(product);
```

비관적 락

- 항상 최악의 경우(즉, 충돌)를 가정하고 데이터를 보호하는 방식
- 재고 업데이트 시 비관적 락으로 다른 트랜잭션이 해당 데이터를 변경하지 못하도록 함

05

동시성 문제 회피 : 레디스(Redis)

고성능의 키-값 구조의 인메모리 비정형 데이터베이스

```java
@Repository
public class CouponCountRepository {

    private RedisTemplate<String, String> redisTemplate;

    3 usages  lilylemonoh *
    public Long increment() {
        // Redis 데이터베이스에서 "count"라는 키의 값을 증가시키는 메서드 - 카운터 구현
        Long result = redisTemplate.opsForValue().increment( key: "count",  delta: 1); //1을 증가시키도록 수정
        return result;

    }
```

예상되는 동시성 문제를 회피하고, 해결하기 위해 비관적 락과 레디스를 도입하였습니다.

레디스 템플릿

레디스에서 제공하는 increment 메서드 사용
→ 여러 사용자가 동시에 같은 쿠폰을 요청하더라도 각 요청이 순차적으로 처리되어, 'count' 값이 안전하게 1씩 증가한다는 것을 보장

04

기술 소개 - 2

enum을 활용한 전역 예외처리 / JWT / 좋아요 기능 동적테이블, 파티셔닝 적용 / Java의 Optional 클래스

발표자 : 권아영

01 enum을 활용한 전역 예외처리 (Global Exception Handler)

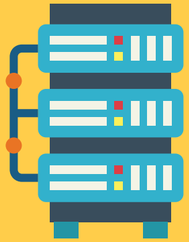
```
// enum을 활용했을 시의 장점 > 일관성 있는 응답 포맷(Response Format)(우리는 status, code, message로 통일)
Dongja +3
public enum ExceptionCode { // 예외 발생시, body에 실어 날려줄 상태, code, message 커스텀

    // 예외 이름들 상수로 구분해서 적어줄 것.

    3 usages
    POST_NOT_FOUND( status: 404, code: "POST_001", message: "해당되는 id 의 글을 찾을 수 없습니다."),
    2 usages
    REPLY_NOT_FOUND( status: 404, code: "REPLY_001", message: "해당되는 id의 댓글을 찾을 수 없습니다."),
    1 usage
    ALREADY_LIKED( status: 400, code: "LIKE_001", message: "이미 '좋아요'를 누른 상태입니다."),


    /* 예외들은 서로 , 콤마로 구분하고 있으므로 잘 확인하고 작성해주세요. */
// ================================== community ======================================
    // ==칸으로 구분하여 예외처리 추가 할 것.(이칸은 shopping)
    9 usages
    PRODUCT_ID_NOT_FOUND( status: 404, code: "SHOP_001", message: "해당되는 id의 상품을 찾을 수 없습니다."),
    1 usage
    CART_ID_NOT_FOUND( status: 404, code: "SHOP_002", message: "해당되는 id의 장바구니를 찾을 수 없습니다."),
    3 usages
    ORDER_ID_NOT_FOUND( status: 404, code: "SHOP_003", message: "해당되는 id의 주문을 찾을 수 없습니다."),
    1 usage
    IMAGE_ID_NOT_FOUND( status: 404, code: "SHOP_004", message: "해당되는 id의 이미지를 찾을 수 없습니다."),
    4 usages
    CATEGORY_ID_NOT_FOUND( status: 404, code: "SHOP_005", message: "해당되는 id의 카테고리를 찾을 수 없습니다."),
    1 usage
    WISHLIST_ID_NOT_FOUND( status: 404, code: "SHOP_006", message: "해당되는 id의 찜을 찾을 수 없습니다"),
    1 usage
```

```
@Slf4j // Lombok 라이브러리의 다양한 logging시스템을 쓸 수 있음(예 : log.debug(), log.info(), log.error())
@RestControllerAdvice // 비동기식 controller의 예외처리 관리시 붙이는 어노테이션
public class GlobalExceptionHandler {

    no usages
    private final HttpStatus HTTP_STATUS_OK = HttpStatus.OK;

    //비즈니스 로직의 예외처리(Unchecked Exception 발생시 처리)
    no usages  Dongja
    @ExceptionHandler(BusinessException.class) // 만들어준 커스텀익셉션 발생시 처리해주는 곳
    public ResponseEntity<ErrorResponse> handleCustomException(BusinessException ex) {
        log.error("Business Exception Error", ex);

        final ErrorResponse errorResponse = ErrorResponse.of(ex.getExceptionCode(), ex.getMessage());


        return new ResponseEntity<>(errorResponse, HttpStatus.valueOf(errorResponse.getStatus()));

    }

    //여기부턴 클라이언트 측의 잘못된 요청에 의한 에러를 처리해줌.
    no usages  Dongja
    @ExceptionHandler(NullPointerException.class) // nullPointerExceptiono발생시
    protected ResponseEntity<ErrorResponse> handleNullPointerException(NullPointerException e) {
        log.error("handleNullPointerException", e);
        final ErrorResponse response = ErrorResponse.of(ExceptionCode.NULL_POINT_ERROR, e.getMessage());
        return new ResponseEntity<>(response, HttpStatus.valueOf(response.getStatus()));
    }


    // @valid 유효성 검증에 실패했을 경우 발생하는 예외 처리
    no usages  Dongja
    @ExceptionHandler(MethodArgumentNotValidException.class)
    protected ResponseEntity<ErrorResponse> handleMethodArgumentNotValidException(MethodArgumentNotValidException ex) {
        log.error("handleMethodArgumentNotValidException", ex);
        BindingResult bindingResult = ex.getBindingResult();
        StringBuilder stringBuilder = new StringBuilder();
        for (FieldError fieldError : bindingResult.getFieldErrors()) {
            stringBuilder.append(fieldError.getField()).append(":");
            stringBuilder.append(fieldError.getDefaultMessage());
            stringBuilder.append(", ");
```

02 JWT

Client

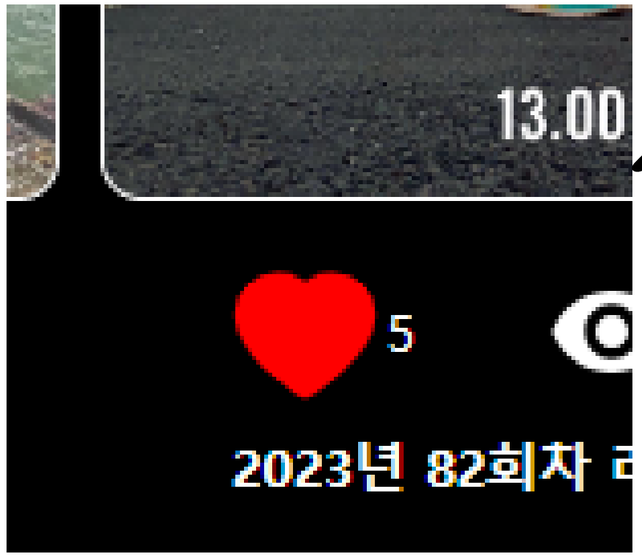
Server



Database

1. 로그인 요청 (id, pw)
2. 회원 정보 확인
3. Access Token, Refresh Token 발급
4. Refresh Token 저장
5. Access Token 검증
6. Access Token 검증 후 응답
7. Access Token 만료, Refresh Token 검증 요청
8. Refresh Token 검증
9. 검증 후 Access Token, Refresh Token 둘 다 재발급
10. 새 Refresh Token 저장

03 좋아요 기능 동적테이블, 파티셔닝 기법 적용




```
<insert id="createDynamicLike" parameterType="com.spring.community.DTO.
    CREATE TABLE IF NOT EXISTS likes_${postId} (
    like_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
    );
```

- likes_10
- likes_100
- likes_101
- likes_102
- likes_103
- likes_104
- likes_105
- likes_106
- likes_107
- likes_108
- likes_109
- likes_11
- likes_110
- likes_111
- likes_112
- likes_113
- likes_114

04 Java의 Optional 클래스

Before

```
//새로운 리프레시토큰 refreshToken 엔터티 객체에 담기
RefreshToken newRefreshToken = new RefreshToken(userInfo.getUserId(), refreshToken);
// DB에서 userId에 해당하는 refresh토큰 검색
RefreshToken existingToken = refreshTokenRepository.findByUserId(userInfo.getUserId());

if(existingToken != null){ // 기존에 발급받은 토큰이 있다면
    existingToken.update(newRefreshToken.getRefreshToken());
    refreshTokenRepository.save(existingToken); // 토큰 갱신
} else {
    System.out.println("리프레시토큰 저장:" + newRefreshToken.getRefreshToken());
    // 리프레시토큰 DB에 저장
    refreshTokenRepository.save(newRefreshToken);
}
```

After

```
//새로운 리프레시토큰 refreshToken 엔터티 객체에 담기
RefreshToken newRefreshToken = new RefreshToken(userInfo.getUserId(), refreshToken);
// DB에서 userId에 해당하는 refresh토큰 검색
Optional<RefreshToken> existingToken = Optional.ofNullable(refreshTokenRepository.findByUserId(userInfo.getUserId()));

// 기존 토근이 있든 없든 갱신하거나 새로 저장합니다.
refreshTokenRepository.save(existingToken.orElse(newRefreshToken).update(newRefreshToken.getRefreshToken()));
```

05

후기

- 후기

감사합니다