

PORTFOLIO

Backend Developer 김범석

이름

김 범 석

이메일

kbs4520@naver.com

블로그

<https://tommykim.tistory.com/>

Github

<https://github.com/BeomSeogKim>

PORTFOLIO

CONTENTS

01.

우리 이거 삭제하지말자
(커플 다이어리 서비스)

[팀프로젝트] 데브코스 마지막 협업 프로젝트로 성능 개선 경험을 위해 진행

02.

CREAM
(KREAM 클론 코딩)

[팀프로젝트] 입찰 거래 도메인 이해 및 협업 능력 향상을 위해 진행

03.

Pet Meeting
(펫 기반 매칭 서비스)

[팀프로젝트] 테스트 코드 숙련도 향상 및 Jenkins 사용 경험을 위해 진행

04.

삼삼오오
(소셜 미팅 서비스)

[팀프로젝트] 항해99 협업 프로젝트로 Frontend 개발자와 협업 경험을 위해 진행

Project

우리 이거 삭제하지 말자

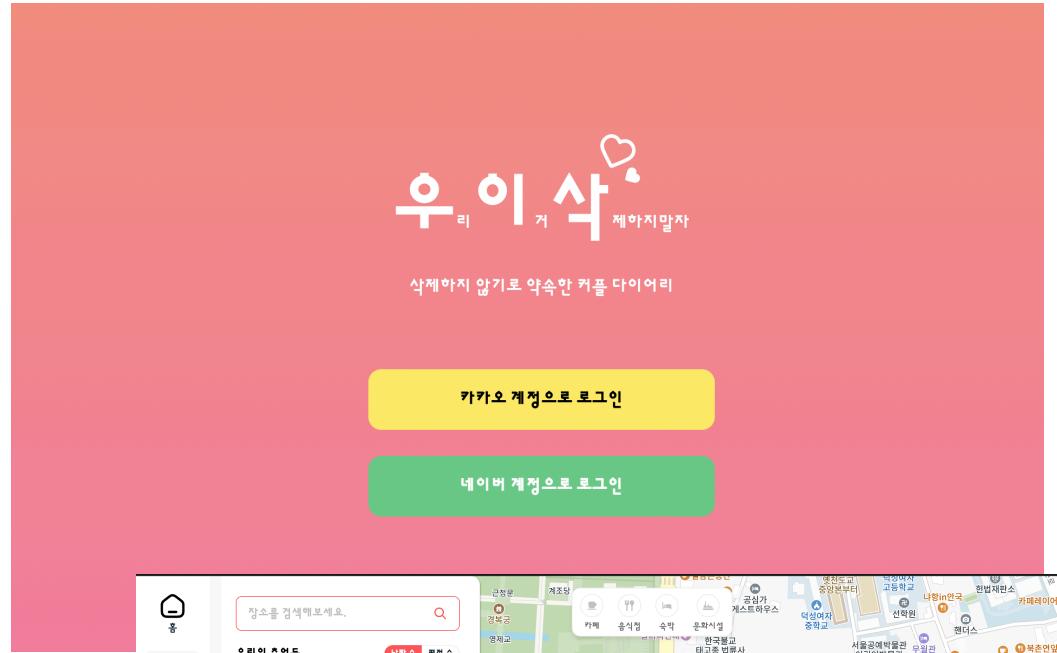
About project

데브코스 웹 교육과정 마지막으로 팀 프로젝트로 진행했습니다.

프로젝트의 도메인 선정에 있어 주된 관점은 다음과 같습니다.

- 테마의 차별성
- 실제로 구현 가능한 프로젝트
- 많은 사용자가 필요하지 않은 프로젝트

애플리케이션의 성능 개선에 초점을 두어 프로젝트를 진행했습니다.



Introduce project

작업 기간	2023. 10 ~ 2023. 12 (2개월)
인력 구성(기여도)	BE 3명 / FE 3명 (Backend 리더, Product Owner 담당)
프로젝트 목적	데브코스 마지막 협업 프로젝트로 성능 개선 경험을 위해 진행함
프로젝트 내용	연인간 데이트 했던 장소에 대해 느꼈던 감정들을 기록하고 다이어리 기반으로 장소에 대한 기억들을 추억 할 수 있는 서비스
주요 업무 및 상세 역할	<ul style="list-style-type: none"> 1) 애플리케이션 튜닝 2) SQL 튜닝 3) 동시성 이슈를 고려한 사랑의 온도 기능 구현 4) 테스트 환경 통합 5) ArchUnit 테스트 도입
사용언어 및 개발 환경	Java 17, Spring Boot 3.1, EC2, S3, RDS(MySQL)
참고 자료	Github 링크 Service 링크

Main work 1. 애플리케이션 튜닝

- 작업

비슷한 서비스의 트래픽 : 월간 4,000만 트래픽

목표 TPS = $40,000,000 / 30 / 24 / 60 / 60$ (한달 30일 가정)

평소 시간대 목표 TPS : 17

붐비는 시간대 목표 TPS : 51

- 작업 내용

- Hikari CP Size 조절

- SQL 튜닝

- HTTP 2.0 적용

- ETag 적용

- Caffeine Cache 적용

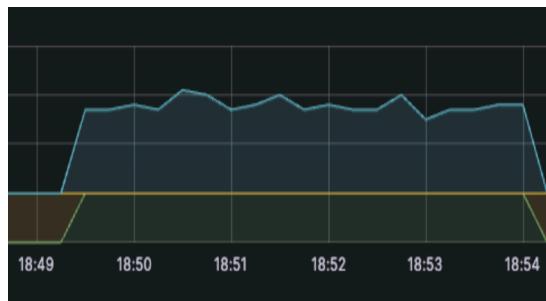
- 효과

- TPS 12배 증가 ($12.8 \rightarrow 368.8$)

- 에러율 감소 ($4.4\% \rightarrow 0.0\%$)

Hikari CP Size 조절

Vuser : 27의 경우 Connection 부족 현상 발견 -> Connection Pool 설정 변경



Hikari CP

```
spring:  
datasource:  
hikari:  
maximum-pool-size: 50  
minimum-idle: 20
```

Hikari CP 설정

HTTP 2.0 적용

Nginx 설정을 통한 HTTP 2.0 적용을 통해 보다 나은 클라이언트 경험 제공

```
server {  
    listen 443 ssl http2;  
    listen [::]:443 ssl http2;
```

Nginx 설정

Main work 1. 애플리케이션 튜닝

ETag 적용

자원이 변경되지 않았을 경우 네트워크 대역폭 절약 및 서버의 부하 감소를 위해 ETag 적용

Cache-Control : no-cache 설정을 통해 최신 상태의 자원 사용 강제함

```
● ○ ●
@Bean
public ShallowEtagHeaderFilter shallowEtagHeaderFilter() {
    return new ShallowEtagHeaderFilter();
}
```

ShallowEtagHeaderFilter Bean 등록

```
● ○ ●
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new HandlerInterceptor() {
        @Override
        public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws
Exception {
            if (request.getMethod().equals("GET")) {
                response.setHeader("Cache-Control", "no-cache");
            }
            return HandlerInterceptor.super.preHandle(request, response, handler);
        }
    });
}
```

GET 요청일 경우 Cache-Control 설정 헤더 추가

Caffeine Cache 적용

자주 조회되는 자원의 경우 메모리에서 조회하여 빠르게 응답할 수 있도록 Caffeine Cache 적용

```
● ○ ●
@Configuration
@Profile("!test")
@EnableCaching
@Slf4j
public class CacheConfig {
    @Bean
    public CacheManager cacheManager() {
        CaffeineCacheManager cacheManager = new CaffeineCacheManager();
        cacheManager.setCaffeine(
            Caffeine.newBuilder()
                .expireAfterAccess(10, TimeUnit.MINUTES)
                .initialCapacity(200)
                .softValues()
                .maximumSize(1000)
                .recordStats()
                .removalListener((key ,value, cause) -> log.debug("key: {}, value: {}가 제거 되었습니다. cause: {}", key,
value, cause));
        );
        return cacheManager;
    }
}
```

Caffeine Cache 설정

Main work 2. SQL 튜닝

- 작업

초기 : Primary Key에만 Index 적용

서비스 출시 후 : 실제로 서비스에서 자주 사용하는 쿼리를 분석해 Covering Index 적용

- 작업 내용

- 다이어리 조회 관련 SQL 튜닝 작업 진행.

다이어리 목록 조회

인덱스 생성 diary(dating_day DESC, couple_id)

```
SELECT
  diary d
FROM diary d
  LEFT JOIN location l ON l.id = d.location_id
WHERE d.couple_id = 1
ORDER BY d.dating_day DESC
LIMIT 20 OFFSET 0;
```

튜닝 전

id	table	rows	filtered	Extra
1	d	12110	100	Using filesort
1	l	1	100	null

튜닝 후

id	table	rows	filtered	Extra
1	d	157	12.68	Using where
1	l	1	100	null

다이어리 마커 조회

인덱스 생성 diary(location_id, couple_id)

```
SELECT
  diary d
  JOIN location l ON l.id =
d.location_id
WHERE l.kakao_map_id = 10978
AND d.couple_id = 1;
```

튜닝 전

id	table	rows	filtered	Extra
1	d	12102	100	Using where
1	l	1	5	Using where

튜닝 후

id	table	rows	filtered	Extra
1	d	40	100	null
1	l	1	100	null

다이어리 지도 조회

SQL 변경 (BETWEEN {min_val} AND {max_val})

인덱스 생성 location(latitude, longitude)

```
SELECT
  diary d
  JOIN location l ON l.id =
WHERE d.couple_id = 1
  AND 90.0 <= l.latitude <= 91.0
  AND 90.0 <= l.longitude <= 91.0;
```

튜닝 전

id	table	rows	filtered	Extra
1	d	12102	100	Using where
1	l	1	100	Using where

튜닝 후

id	table	rows	filtered	Extra
1	d	41	11.1	Using index
1	l	1	100	null

Main work**3. 동시성 이슈를 고려한 사랑의 온도 기능 구현**

- 작업

회원들의 서비스 참여도를 높이기 위해 사랑의 온도 기능 도입

온도의 증가 조건

- 다이어리 작성
- 오늘의 질문 답변

동시성 고려 상황

- 활발한 서비스 이용으로 인해 동시에 사랑의 온도 증가로직 실행

Optimistic Lock 적용

- 증가 로직의 경우 빈번하게 충돌이 일어나지 않을 것이라 판단
- Pessimistic Lock 대신 Optimistic Lock 적용

이벤트 기반 및 비동기적인 로직 실행

- 충돌이 발생해 재시도 하는 경우 시간 소요
- 비동기적으로 로직을 실행하여 시스템 처리량 개선

```

@Transactional
public Long createDiary(parameters) {
    // do something..

    // Event 발생
    Events.raise(new DiaryCreatedEvent(coupleId,
        String.valueOf(cacheId), CacheConstants.DIARY_LIST,
        CacheConstants.DIARY_MARKER,
        CacheConstants.DIARY_GRID));
    return savedDiary.getId();
}

@Async
@TransactionalEventListener(
    classes = DiaryCreatedEvent.class,
    phase = TransactionPhase.AFTER_COMMIT
)
public void handle(DiaryCreatedEvent event) {
    coupleService.increaseTemperature(event.getCoupleId());
}

@Retryable(retryFor = ObjectOptimisticLockingFailureException.class,
    maxAttempts = 3, backoff = @Backoff(delay = 1000))
@Transactional
public void increaseTemperature(Long coupleId) {
    Couple couple = coupleRepository.findByIdWithOptimisticLock(coupleId)
        .orElseThrow(() -> new EntityNotFoundException("존재하지 않는 커플 id 입니다."));
    couple.increaseTemperature();
}

```

다이어리 생성로직

다이어리 생성 시 DiaryCreatedEvent 발행

EventListener

DiaryCreatedEvent 발생시 실행되는 로직
Event를 호출한 곳에서 정상적으로 COMMIT 되었을 경우에 온도 증가 로직 실행

사랑의 온도 증가 로직

@Retryable을 이용한 재시도 로직 구현
동시성 문제 발생 시 최대 3번 재시도 실행

Main work 4. 테스트 환경 통합

- 작업

- 각각 Test 클래스에 @SpringBootTest 사용시 새로운 Spring Container 생성
- Spring Container의 경우 생성 비용이 비쌈
- 추상클래스를 사용한 테스트 환경 통합을 통해 하나의 Spring Container만 생성될 수 있도록 설정

- 효과

기존 테스트 시간 대비 10초 감소 (60초 -> 50초)

테스트 통합 전용 추상 클래스 생성

```
● ● ●  
@ActiveProfiles("test")  
@SpringBootTest  
public abstract class IntegrationTestSupport  
{  
}
```

테스트 환경 통합

```
● ● ●  
class DiaryQueryServiceTest extends IntegrationTestSupport  
{  
    // Test Logic  
}
```

테스트 환경 통합

```
● ● ●  
class MemberServiceTest extends IntegrationTestSupport  
{  
    // Test Logic  
}
```

Main work 5. ArchUnit 테스트 도입

- 작업

팀원간 사전에 정의한 규칙을 지키기 위해 ArchUnit 테스트 도입

- 작업 내용

- 패키지간 순환 참조 방지
- 클래스간 의존성 테스트
- Service 클래스에서 @Transactional 설정
- Controller, Service 클래스의 필드 변수 검사

```
class ArchTests {  
  
    private static JavaClasses importedClasses;  
  
    @BeforeAll  
    public static void setUp() {  
        importedClasses = new ClassFileImporter()  
            .importPackages("com.lovely4k.backend");  
    }  
  
    @DisplayName("패키지 간 의존 관계가 순환되면 안된다.")  
    @Test  
    void noCyclesInPackages() {  
        SlicesRuleDefinition.slices()  
            .matching("com.lovely4k.backend.(*)..")  
            .should().beFreeOfCycles()  
            .check(importedClasses);  
    }  
  
    // ...  
}
```

Main work 6. 그 외 고민 사항

• 각 계층의 테스트 방식에 대한 고민

- Service, Repository : 두 계층의 경우 상호작용이 중요하다고 판단 -> 통합테스트 진행 (@Spring Boot 수행)
- Controller : 해당 계층의 경우 사용자의 요청에 대한 검증이 중요하다고 판단 -> Service 계층 Mock 처리 (@WebMVCTest 수행)

• 비동기 메소드 테스트에 대한 고민

- 비동기 메소드 테스트 시 실행 결과를 기다리지 않고 넘어가는 경우가 발생 (깨지는 테스트 발생)
- @TestConfiguration을 사용해 동기 작업으로 변경, TestTransaction을 사용해 트랜잭션 커밋 및 종료 실행

• Cache 관련 삭제 로직에 대한 고민

- 다이어리 작성, 수정 및 삭제 시 커플에 해당되는 다이어리 캐시 삭제 로직 필요.
- 작성, 수정 및 삭제 시 이벤트를 발행하고, 이벤트 리스너에서 coupleId로 시작하는 캐시를 제거 하는 방식으로 문제 해결

• 무중단 배포와 관련한 고민

- 현재 활성화된 버전을 확인 후에 활성화되지 않은 port로 application을 시작해야함
- Spring actuator에서 제공하는 profile 속성을 통해 현재 활성화된 버전 확인을 할 수 있도록 처리

Project 02.

CREAM

About project

실제 서비스 중인 도메인에 대한 이해와 팀원 간 협업 능력 향상을 위해
프로젝트를 진행했습니다.

프로젝트의 주된 관점은 다음과 같습니다.

- 입찰 거래 도메인에 대한 이해
- 협업 능력 향상

Introduce project

작업 기간	2023. 08 ~ 2023. 09 (1개월)
인력 구성(기여도)	BE 3명 (Team Leader 담당)
프로젝트 목적	입찰 거래 도메인 이해 및 협업 능력 향상을 위해 진행
프로젝트 내용	KREAM 클론코딩
주요 업무 및 상세 역할	<ol style="list-style-type: none">1) git-flow 정책 도입2) Github Projects 도입3) 입찰 시스템 구축4) Monolithic -> Multi Module 적용
사용언어 및 개발 환경	Java 17, Spring Boot 3.1, EC2, S3, RDS(MySQL)
참고 자료	Github 링크 회고록

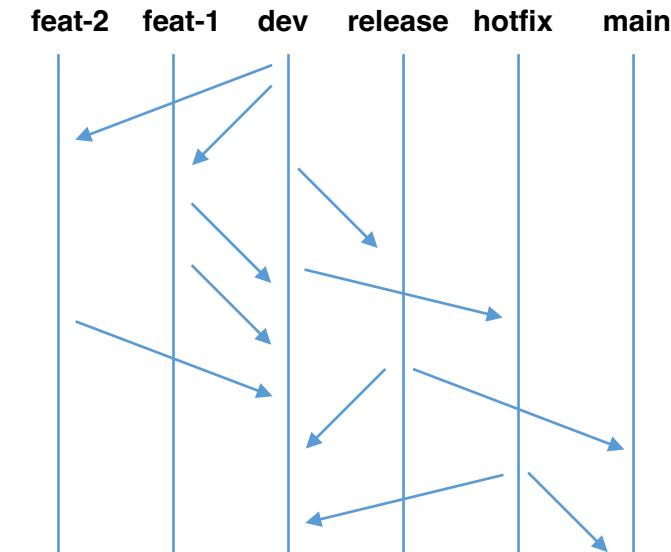
Main work 1. git - flow 정책 도입 [[wiki 바로가기](#)]

- 작업

각자 맡은 기능에 대해 최대한 충돌이 일어나지 않고, 빠른 배포주기를 갖기 위해 git - flow 정책을 도입

각 브랜치의 전략은 다음과 같습니다.

- main : 실제 배포가 가능한 코드만 존재하는 branch
- release : 출시를 위해 준비하는 branch
- hotfix : 출시버전에서 발생한 버그를 수정하는 branch
- dev : 다음 출시 버전을 개발하는 branch
- feature : 기능을 개발하는 branch



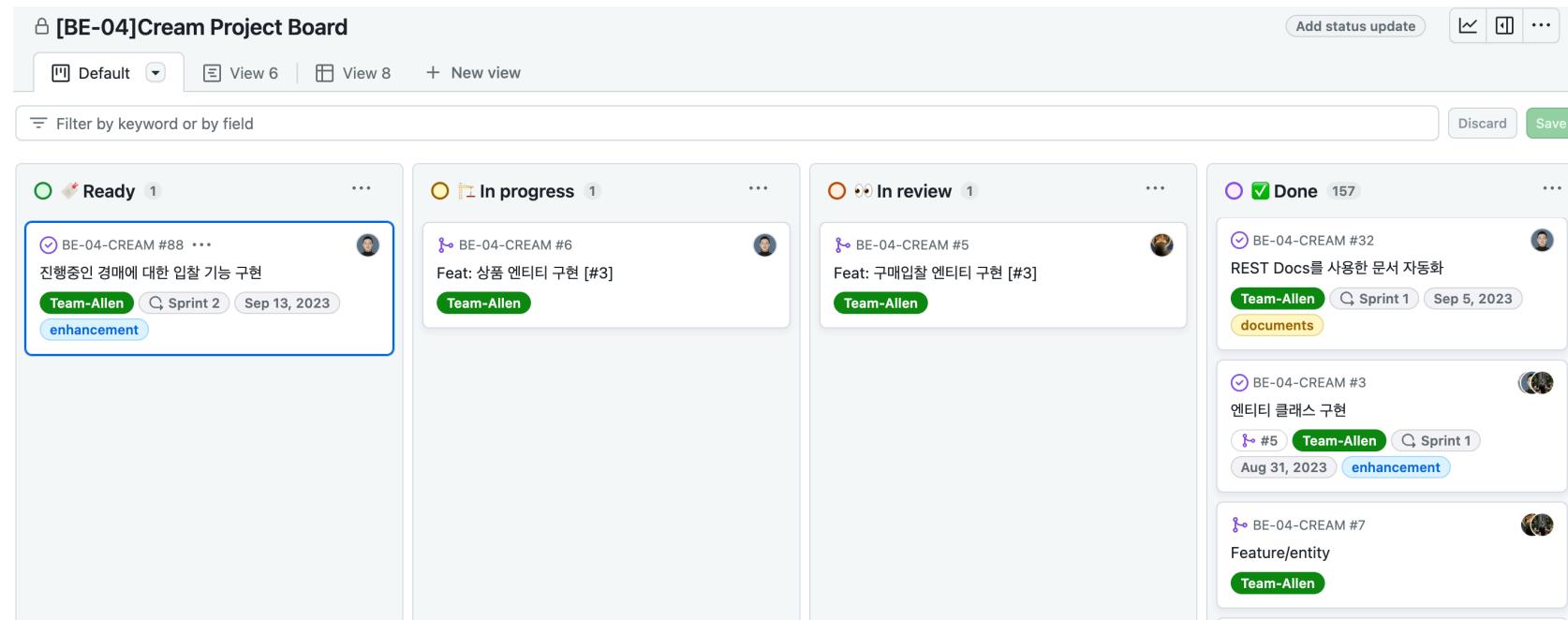
Main work 2. Github Projects 도입

- 작업

Project 관리를 위해 Github Projects를 도입

Jira와 Github Projects를 비교했을 때 다음과 같은 장점이 있다고 판단하여 Github Projects를 사용

- 낮은 Learning Curve
- Github과 긴밀한 통합
- 간편한 사용성



Project 02.

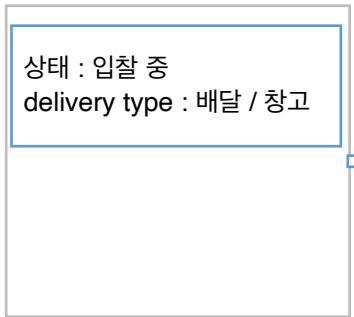
CREAM

Main work 입찰 시스템 구축 [[Wiki 바로가기](#)]

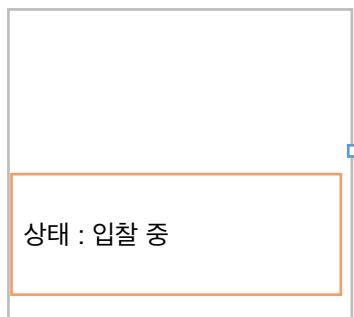
- 작업

입찰 시스템의 전반적인 흐름을 구축

[구매입찰]



[판매입찰]



: 구매 입찰

: 판매 입찰

빨간 글씨 : 제한 조건

상태 : 배송 완료 / 창고
delivery type : 배달 / 창고

상태 : 거래 완료

[구매자] 완료

상품이 배송 되었거나 창고에 저장 중 이어야 함

판매자는 거래 대금과 포인트를 받음

구매자는 포인트를 받음

상태 : 거래 중
delivery type : 배달 / 창고

상태 :
검수 합격 / 검수 불합격

[판매자] 물품 검수

판매 상품 검수 불합격 시
구매입찰 취소 처리

상태 : 배송 중 / 창고
delivery type : 배달 / 창고

상태 : 검수 합격

[구매자] 입금

판매 상품이 검수에 합격해야 함
구매자의 잔고가 충분해야 함
delivery type에 따라서 배송 혹은 창고저장으로 변경

[판매자] 입찰 상품 거래

상태 : 거래 중
delivery type : 배달 / 창고

상태 : 거래 중

[구매자] 입찰 상품 거래

구매 입찰보다 낮은 가격 등록 불가

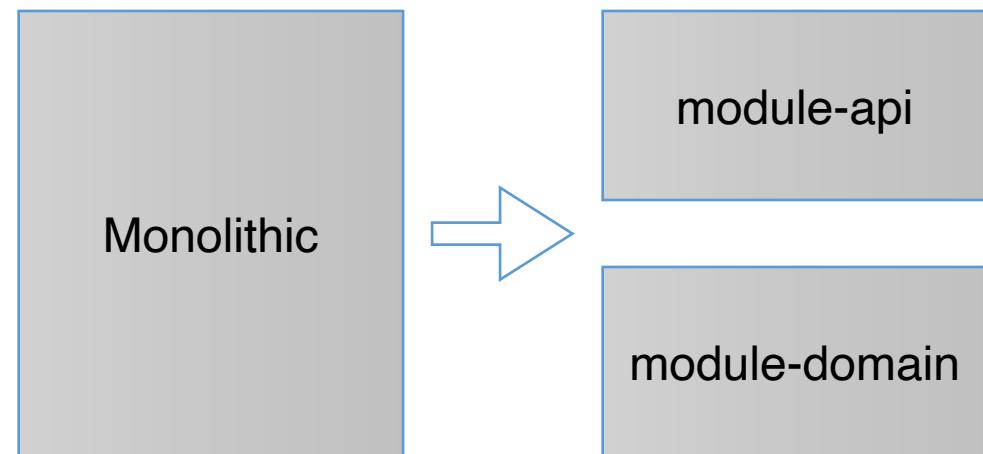
Main work 4. Monolithic -> Multi Module 적용

- 작업

개발 초기에는 Monolithic 방식으로 개발하였으나

다음과 같은 이점들로 인하여 Multi Module 방식으로 변경

- 관심사의 분리
- 유지보수성 및 확장성 용이



Project 03.

Pet Meeting

About project

테스트 코드에 대한 경험 및 숙련도 향상 및 Jenkins 사용 경험을 위해
프로젝트를 진행했습니다.

프로젝트의 주된 관점은 다음과 같습니다.

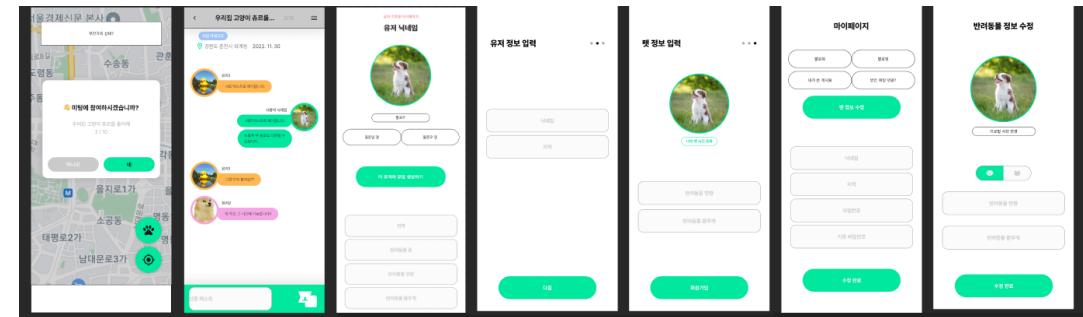
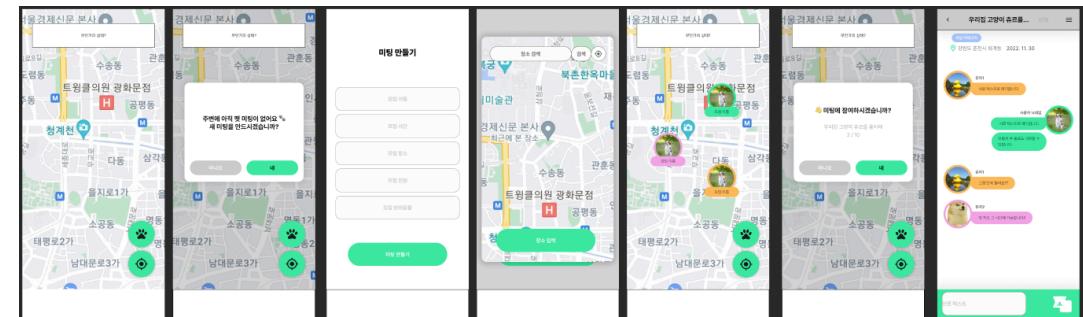
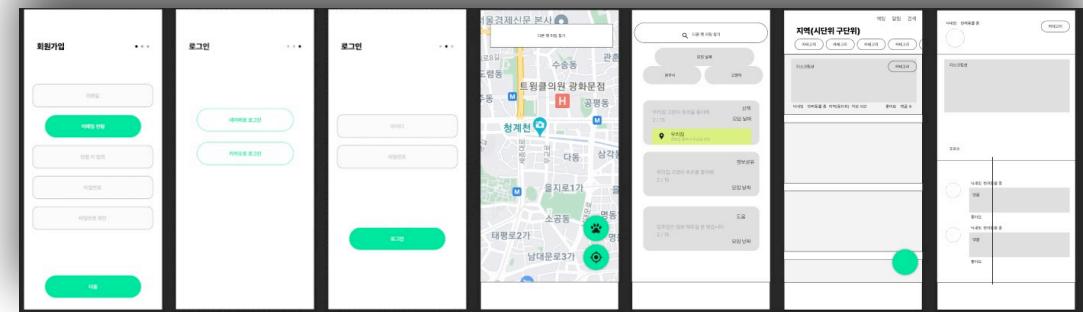
- 테스트 코드 경험 및 숙련도 향상
- Jenkins 사용

Project 03.

Pet Meeting

Introduce project

작업 기간	2022. 11 ~ 2023. 02 (3개월)
인력 구성(기여도)	BE 2명 / FE 3명
프로젝트 목적	테스트 코드 숙련도 향상 및 Jenkins 사용 경험을 위해 진행함
프로젝트 내용	반려 동물을 키우는 주인들을 위한 매칭 서비스
주요 업무 및 상세 역할	<ul style="list-style-type: none"> 1) 회원가입 시 SMTP를 사용한 검증 기능 구현 2) Restful API를 준수하기 위한 HATEOAS 제공 3) REST Docs를 사용한 테스트코드 기반 API 문서화 구현 4) Jenkins를 사용한 CI / CD 구축 5) Stomp를 사용한 채팅 기능 구현
사용언어 및 개발 환경	Java 11, Spring Boot 2.7, EC2, S3, RDS(MySQL), Redis
참고 자료	Github 링크



Main work 1. 회원가입 시 SMTP를 사용한 검증 기능 구현

- 작업

문제 상황 발생

- 이메일을 통한 회원가입 기능 제공
- 실제로 존재하지 않는 이메일로 가입하는 유저 발생

문제 해결

- 회원가입시 작성한 이메일에 확인 코드를 제공
- 코드가 일치해야 가입을 할 수 있도록 규제 강화

회원 가입 전 인증 코드 확인 관련 API

```
@PostMapping("/emailConfirm")
public ResponseEntity emailConfirm(@RequestParam String email) throws Exception
{
    String confirm = emailService.sendSimpleMessage(email);

    //Http Response 관련 처리 작업
    return new ResponseEntity<>();
}
```

인증 코드 생성 후 이메일 발송

```
@Override
public String sendSimpleMessage(String to) throws Exception
{
    MimeMessage message = createMessage(to);      // 메세지 생성
    try{
        emailSender.send(message);
    }catch(MailException es){
        // 예외 처리
    }
    return validationNumber; // 인증 코드
}
```

인증 완료 후 회원가입 진행

```
@PostMapping(value = "/signup")
public ResponseEntity signup() throws IOException
{
}
```

Main work 2. Restful API를 준수하기 위한 HATEOAS 제공

- 작업

REST 아키텍처 스타일 중 **Uniform Interface**을 지키는 것에 초점을 둠

Uniform Interface

- self-descriptive message
- HATEOAS(hypermedia as the engine of application state)

이점

- 서버의 메시지가 변경되어도 클라이언트는 메시지를 보고 이해할 수 있음
- 링크의 정보를 동적으로 변경할 수 있음

```
이점
@RequiredArgsConstructor
@RestController
@RequestMapping(value = "/api/post", produces = HAL_JSON_VALUE+";charset=UTF-8")
public class PostController {
    private final PostService postService;

    @PostMapping
    public ResponseEntity<Object> createPost(/* Parameters */) throws IOException {
        PostResponseDto postResponseDto = postService.createPost(postRequestDto, image,
        userDetails.getMember());
        ResponseResource responseResource = new ResponseResource(postResponseDto);
        responseResource.add(linkTo(PostController.class).withSelfRel());
        responseResource.add(linkTo(PostController.class).slash(postResponseDto.getId()).withRel("post-get"));
        responseResource.add(linkTo(PostController.class).slash(postResponseDto.getId()).withRel("post-edit"));
        responseResource.add(linkTo(PostController.class).slash(postResponseDto.getId()).withRel("post-
delete"));
        Response response = new Response(StatusEnum.CREATED, "게시글 작성 성공", responseResource);

        return new ResponseEntity<>(response, HttpStatus.CREATED);
    }
}
```

Main work 3. REST Docs를 사용한 테스트코드 기반 API 문서화 구현

- 작업

기존의 경우 Postman을 통해 API 문서화를 진행

Postman API 문서의 경우 애플리케이션의 API 변경 발생시 수동으로 업데이트 해주어야 하는 문제 발생

테스트코드 기반 문서의 경우 실시간으로 API 변경 사항이 반영됨 => 신뢰성 높은 API 제공

REST Docs를 사용해 테스트코드 기반 API 문서 제공

The screenshot shows the RestDocs API documentation interface. On the left is a sidebar with a tree view of API endpoints and their methods. The main content area has a title 'REST API 가이드' (REST API Guide) and a section titled '개요' (Overview) which contains a table of HTTP verbs and their descriptions.

설명	용례
GET	을 접할 때 사용
POST	새로 만들 때 사용
PUT	기존에 구성한 때 사용
PATCH	기존의 일부를 구성할 때 사용
DELETE	외부 구조를 유지하기 위해 사용

RestDocs API 문서

```

    @Test
    @DisplayName("메세지가 있을 때 메세지 조회")
    void getMessageList() throws Exception {
        // ... 테스트 준비 관련 코드
        this.mockMvc.perform(get("/api/message/" + chatRoom.getRoomId())
            .header("Authorization", getAccessToken())
            .contentType(APPLICATION_JSON)
            .accept(HAL_JSON))
            .andExpect(status().isOk())
            .andDo(document("get_MessageList",
                requestHeaders(
                    headerWithName(HttpHeaders.ACCEPT).description("accept header"),
                    headerWithName(HttpHeaders.AUTHORIZATION).description("access token"),
                    headerWithName(HttpHeaders.CONTENT_TYPE).description("content type")),
                responseHeaders(
                    headerWithName(HttpHeaders.CONTENT_TYPE).description("content type")),
                responseFields(
                    fieldWithPath("status").description("status of action"),
                    fieldWithPath("message").description("message of action"),
                    fieldWithPath("data.object[0].id").description("id of message"),
                    fieldWithPath("data.object[0].type").description("type of message"),
                    fieldWithPath("data.object[0].roomId").description("id of chatRoom"),
                    fieldWithPath("data.object[0].sender").description("message sender"),
                    fieldWithPath("data.object[0].senderImage").description("image of message sender"),
                    fieldWithPath("data.object[0].message").description("content of message"),
                    fieldWithPath("data.links[0].rel").description("relation"),
                    fieldWithPath("data.links[0].href").description("url of action"))
                )));
    }
}

```

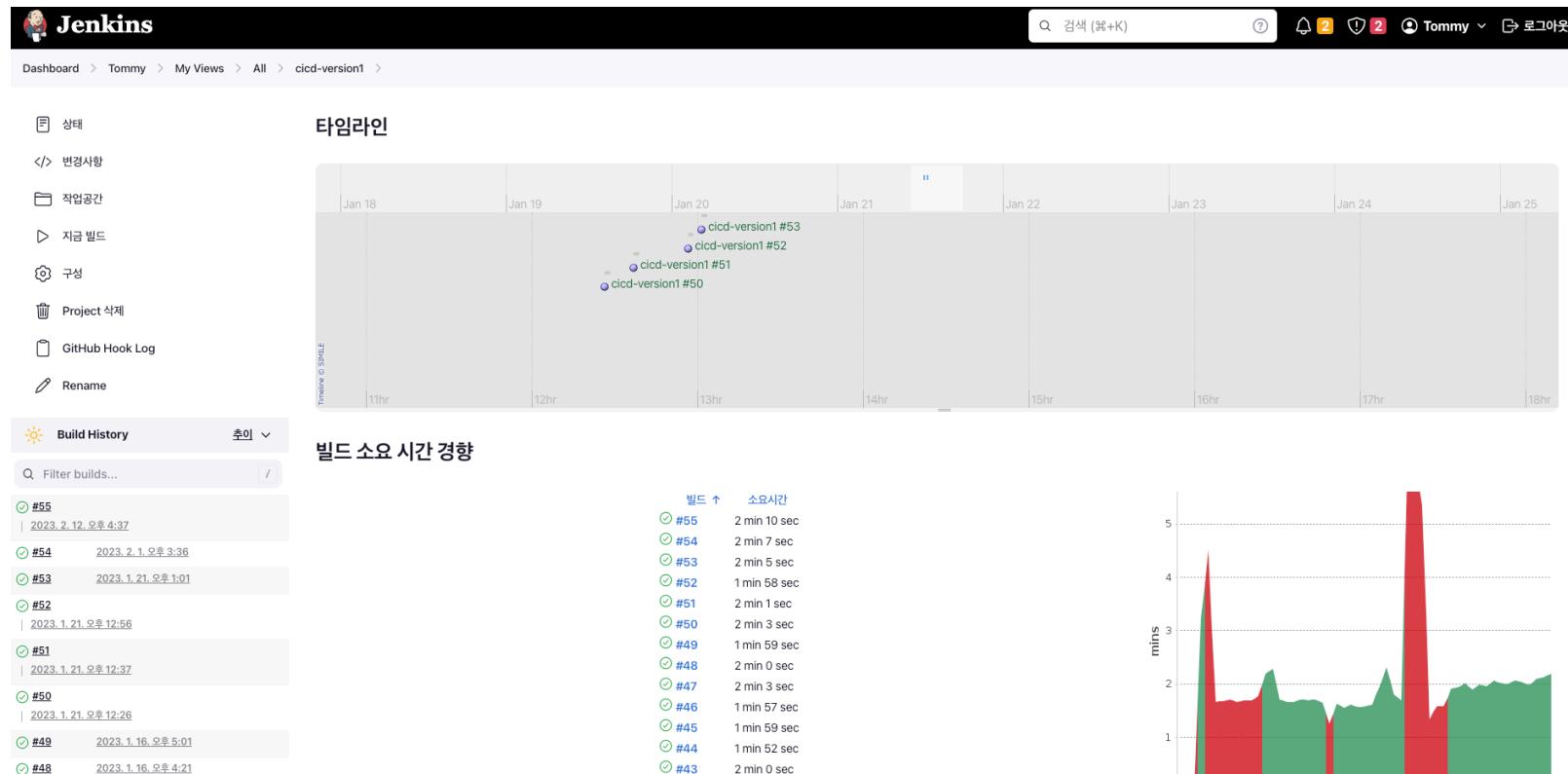
API 관련 테스트 코드 작성

Main work 4. Jenkins를 사용한 CI / CD 구축

- 작업

이전 프로젝트에서 CD 관련하여 Github Actions를 적용해 본 경험이 있음

CD와 관련해 Jenkins를 경험해 보고자 프로젝트에 적용



Main work 5. Stomp를 사용한 채팅 기능 구현

- 작업

모임에 참여하는 회원들간 실시간 소통을 위해 WebSocket 통신 기반 채팅 기능 구현

추후 Scale-out을 고려해 Message Broker로 Redis를 사용

WebSocket 및 STOMP 설정

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSockConfig implements WebSocketMessageBrokerConfigurer
{
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/sub");
        config.setApplicationDestinationPrefixes("/pub");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry)
    {
        registry.addEndpoint("/ws-stomp").setAllowedOriginPatterns("*")
            .withSockJS();
    }
}
```

Redis 관련 설정

```
@Configuration
public class RedisConfig {
    // Listener 설정
    @Bean
    public RedisMessageListenerContainer redisMessageListener(RedisConnectionFactory
                                                               connectionFactory,
                                                               MessageListenerAdapter listenerAdapter,
                                                               ChannelTopic channelTopic) {
        log.info("RedisConfig RedisMessageListenerContainer");
        RedisMessageListenerContainer container = new RedisMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.addMessageListener(listenerAdapter, channelTopic);
        return container;
    }

    // subscriber 설정 관련
    @Bean
    public MessageListenerAdapter listenerAdapter(RedisSubscriber redisSubscriber) {
        return new MessageListenerAdapter(redisSubscriber, "sendMessage");
    }
}
```

Project 04.

삼삼오오

About project

항해99에서 진행한 협업 프로젝트로 Frontend 개발자와 협업을 위해
프로젝트를 진행했습니다.

프로젝트의 주된 관점은 다음과 같습니다.

- Frontend와의 협업
- 프로젝트의 완성

Introduce project

작업 기간	2022. 09 ~ 2022. 10 (1개월)
인력 구성(기여도)	BE 2명 / FE 3명 (Backend 리더, Team 리더 담당)
프로젝트 목적	항해99 협업 프로젝트로 Frontend 개발자와 협업 경험을 위해 진행
프로젝트 내용	같은 관심사를 가진 사람들을 모으고 소통할 수 있도록 돋는 소모임 커뮤니티
주요 업무 및 상세 역할	<ol style="list-style-type: none">악성 유저 사용 제재 도입SseEmitter를 사용한 실시간 알림 서비스 기능 구현회원 탈퇴 기능 도입장애 해결
사용언어 및 개발 환경	Java11, Spring Boot 2.7, EC2, S3, RDS(MySQL)
참고 자료	Github 링크

Main work 1. 악성유저 사용 제재 도입

- 작업

문제 상황 발생

- 모집 게시글이나 댓글에 광고성 글을 작성하거나, 남을 비방하는 글을 적는 악성 회원 발생

문제 해결

- 불쾌감을 주는 글들의 경우 서비스의 질을 떨어트리기 때문에 제재가 필요
- 악성 게시글 및 댓글에 대해 총 10회 신고처리가 될 경우 서비스 이용 제재하는 방안 구축

신고 처리 관련 로직

```

@Transactional
public ResponseDto<?> executeReport(/* parameters */) {
    if (report.getPostId() != null) {
        Post post = postRepository.findById(report.getPostId()).orElse(null);
        assert post != null;
        post.executeRegulation();
        // do something ..
    } else {
        Comment comment =
commentRepository.findById(report.getCommentId()).orElse(null);
        assert comment != null;
        comment.executeRegulation();
        // do something ..
    }
    // do something ..
}

```

게시글 제재 로직

```

public class Post extends Timestamped{
    // ...
    public void executeRegulation() {
        this.regulation = REGULATED;
        this.member.executeRegulation();
    }
    // ...
}

```

회원 제재 로직

```

public class Member extends Timestamped{
    // ...
    @Transactional
    public void executeRegulation() {
        this.numOfRegulation++;
        if (this.numOfRegulation >= 10) {
            this.regulation = REGULATED;
            this.userRole = ROLE_GUEST;
        }
    }
    // ...
}

```

Main work 2. SseEmitter를 사용한 실시간 알림 서비스 기능 구현

- 작업

다음의 경우에는 실시간으로 알림을 주면 사용자에게 좋은 경험을 줄 수 있다고 판단.

- 새로운 채팅
- 모임 신청에 대한 결과
- 모임에 대해 지원 신청
- 게시글에 대한 댓글

이벤트가 발생할 때마다 실시간으로 알림을 줄 수 있는 방법인 Server-Sent-Event 방식 채택

알림 전송 로직

```

@Async
public void send(Member receiver, NotificationType notificationType, String notificationContent, String url)
throws Exception {

    Notification notification = notificationRepository.save(createNotification(receiver, notificationType,
notificationContent, url));

    String receiverId = String.valueOf(receiver.getId());
    String eventId = receiverId + "_" + System.currentTimeMillis();
    Map<String, SseEmitter> emitters = emitterRepository.findAllEmitterStartWithUserId(receiverId);
    if (notificationType.equals(CHAT)) {
        emitters.forEach(
            (key, emitter) -> {
                emitterRepository.saveEventCache(key, notification);
                sendNotification(emitter, eventId, key, NotificationChatDto.create(notification));
            }
        );
    } else {
        emitters.forEach(
            (key, emitter) -> {
                emitterRepository.saveEventCache(key, notification);
                sendNotification(emitter, eventId, key, NotificationDto.create(notification));
            }
        );
    }
}

```

Main work 3. 회원 탈퇴 기능 도입

- 작업

회원이 탈퇴를 할 경우 서버에서 처리하는 방식에 대해 다음과 같이 고려

- 데이터베이스에서 회원의 row를 삭제
- 회원의 개인정보와 관련한 속성만 의미 없는 값으로 대체

문제 해결 방안

- 데이터베이스에서 row 삭제시 관련한 row들도 모두 삭제해야 함
- 회원이 열었던 모임을 강제로 없애야하는 문제 발생
- 이에 따라, 의미 없는 값으로 대체하는 것으로 결정

추가적인 고민 사항

- 회원의 정보를 특정 기간 동안 들고 있는 것이 맞는 것인가?
 - 재가입을 고려해 특정 기간동안 회원의 정보를 별다른 테이블에 보관하는 것으로 결정

```

● ● ● 회원 탈퇴 관련 로직
@Transactional
public ResponseDto<?> signOut(HttpServletRequest request) {
    SignOutMember signOutMember = makeSignOutMember(member);
    signOutRepository.save(signOutMember); // 회원 정보 별도 보관

    member.signOut();
    return ResponseDto.success("회원 탈퇴가 성공적으로 수행되었습니다.");
}

● ● ● 회원 탈퇴 관련 로직
public void signOut() {
    this.password = UUID.randomUUID().toString();
    this.nickname = "탈퇴한 회원입니다.";
    this.minAge = 0;
    this.imgUrl = "http://// 서비스에서 기본적으로 제공하는 이미지";
    this.userRole = ROLE_GUEST;
}

● ● ● 특정 기간 지나면 삭제
@scheduled(cron = "0 0 1 * * *")
@Transactional
public void deleteInformation() {
    LocalDateTime now = LocalDateTime.now();
    LocalDateTime criticalLimit = now.minusYears(5);

    List<SignOutMember> signOutMemberList = signOutRepository.findAll();
    for (SignOutMember signOutMember : signOutMemberList) {
        if (criticalLimit.isAfter(signOutMember.getCreatedAt())) {
            signOutRepository.delete(signOutMember);
        }
    }
}

```

Main work 4. 장애 해결

Hikari Connection Pool 응답 지연 이슈

문제 상황

- 실시간 알림 서비스를 구현 후 서버에서 전반적인 API 모두 응답되지 않는 현상 발생
- WAS의 로그를 확인해본 결과 Connection의 request time out error 발생

문제 원인

- spring.jpa.open-in-view=true 설정
- open-in-view 설정이 true일 경우 사용자의 요청이 종료될 때 까지 Connection을 유지
- 실시간 알림의 경우 요청이 종료되지 않아 Connection이 반환되지 않는 현상 발생

문제 해결

- spring.jpa.open-in-view=false 설정
- Service Layer에서 Connection을 반환할 수 있도록 설정하여 문제 해결

Proxy 객체 관련 이슈 (LazyInitializationException)

문제 상황

- spring.jpa.open-in-view=false 설정 후 문제 발생
- MyPage에서 Proxy 객체의 값을 조회해오는 과정에서 **could not initialize proxy** 발생

문제 원인

- MyPage에서 @Transactional을 별도로 붙여주지 않음
- Proxy 객체를 조회해오는 과정에서 영속성 컨텍스트가 종료되어 문제 발생

문제 해결

- MyPage에서 @Transactional을 붙여 문제 해결

읽어주셔서 감사합니다.
