

Personal

이재석 / Jaesuk Lee
1995.11.27 (29세 / 만 27세)
jaesuk95@outlook.com
010-9657-4511
(21023) 인천 계양구 서부간선로

Personality

MBTI : ISTJ

#현실주의자
#논리적
#계획적
#워커홀릭

Experience

- 2022.04 – 2023.02

(주)모두디자이너
담당업무 : 백엔드 / DevOps
직급 : 개발팀 / 사원 (팀원)
- 2021.04 – 2021.06

(주)해강기술
담당업무 : R&D 빅데이터 해양자료
분석, 3D 수치모델링
직급 : 해양사업팀 / 사원 (팀원)
- 2020.03 – 2021.01

(주)오토로닉스
담당업무 : 해양시스템 운영관리
직급 : 해양사업팀 / 사원 (팀원)
- 2016.12 – 2017.02

(주)빛과울
담당업무 : 건축도면 설계
직급 : 인턴

Graduation

- 2021.07

코리아 IT 아카데미
안드로이드 및 웹 개발 수료
- 2017.11

Australian Maritime College
호주 해양대학교 (조선공학과) 졸업
- 2013.11

Massey High School
뉴질랜드 고등학교 졸업

(주)모두디자이너 소개

간단한 소개:

국내에서 유일한 커스텀 펜던트를 제공하는 플랫폼이며, 사용자들이 원하는 펜던트 앞/뒤 측면을 디자인해 온라인으로 판매하는 서비스이다.

추가로 펜던트에 유니크한 QR코드를 생성할 수 있으며 (무료) 미야방지 서비스를 운영하고 있다. 휴대폰 카메라로 QR코드를 찍으면 펜던트 착용하고 있는 아이 위치를 부모 또는 친척에게 채팅방으로 초대한다.

<https://modoodesigner.net/>



(주)모두디자이너 백엔드

나의 담당:

- Restful API 담당
 - 유저 커스텀 펜던트 디자인
 - QR 코드 서비스
 - 펜던트, 목걸이 줄, 길이, 잠금 장식 선택
 - 주문/결제
 - 네이버 페이
 - 네이버 쇼핑 등록
 - OAuth (네이버, 카카오, 구글 로그인)
 - 디자인 조회/캐싱
 - 회원가입/인증
 - 휴대전화 인증
 - 어드민 API (상품 등록, 주문수집 등)
 - 테스트 코드

- 엔티티 담당
 - 연관 관계 (예. @OneToMany, @ManyToOne)
- 메시지 서비스 담당
 - 메시지/이메일 템플릿 CRUD (등록/조회/삭제/업데이트)
 - 휴대전화 문자 메시지
 - 카카오톡 메시지
 - 사용자 회원가입/주문/주문취소 등 이메일
 - 슬랙 연동 (신규주문, 배포 등)

<https://{데브 서버 URI}>

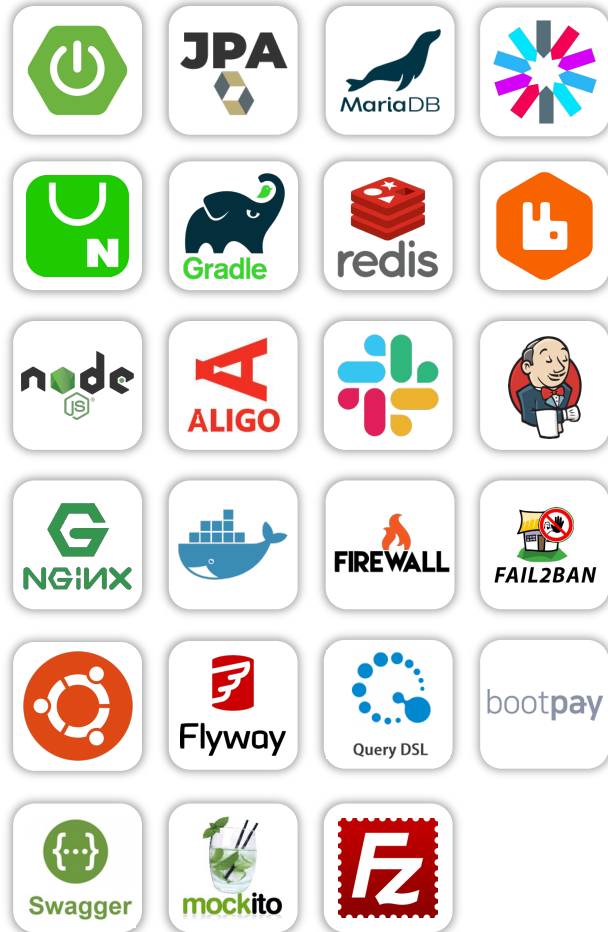
(주)모두디자이너 데브옵스

나의 담당:

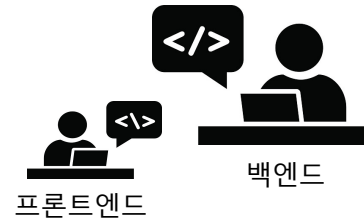
- 내부 개발서버 담당
 - 우분투 20 버전
 - 테스트 환경
 - 실제 운영 서버처럼 조건 맞추기
- 실제 운영 서버 담당
 - 베어메탈 centos 7
 - 방화벽
 - Ftp (UI 디자이너들을 위한 설계)
 - ssh/ftp 접근 추가적인 보안 설정 (예. fail2Ban)
 - Nginx
 - CI/CD (Jenkins 자동화 배포)
 - 도커
 - 데이터베이스, 백업
 - 레디스
 - 레빗MQ

<https://modoodesigner.net/shop/main>

실무 사용했던 기술들



운영 시스템



- 계획
- 사용자들 불편한점 파악/해결
- 내부 직원들(제작자, 디자이너)에게 필요한 데이터 공급
- Rest API 생성
- CI/CD 생성, 관리
- 프론트엔드 요구사항
- 테스트 코드 작성
- 로컬 테스트

Git. Webhook
Branch : 'dev'



데브 서버 (테스트 서버)

Git. Webhook
Branch : 'master'



운영 서버 (Cafe24 베어메탈)

의도

준비한 프로젝트는 실무 경험 바탕으로 현재 보유하고 있는 **기술들을 증명하기 위해 설계된 프로젝트**입니다.

시스템은 실무 경험과 비슷하게 설계되어 있습니다. 단 비즈니스 로직에 시간을 과하게 투자하지 않았고, 또한 구체적인 내용은 건너뛴 부분이 많습니다. 예를 들어, 포인트 적립, 디자이너에게 리워드, 후기, 게시판 등.

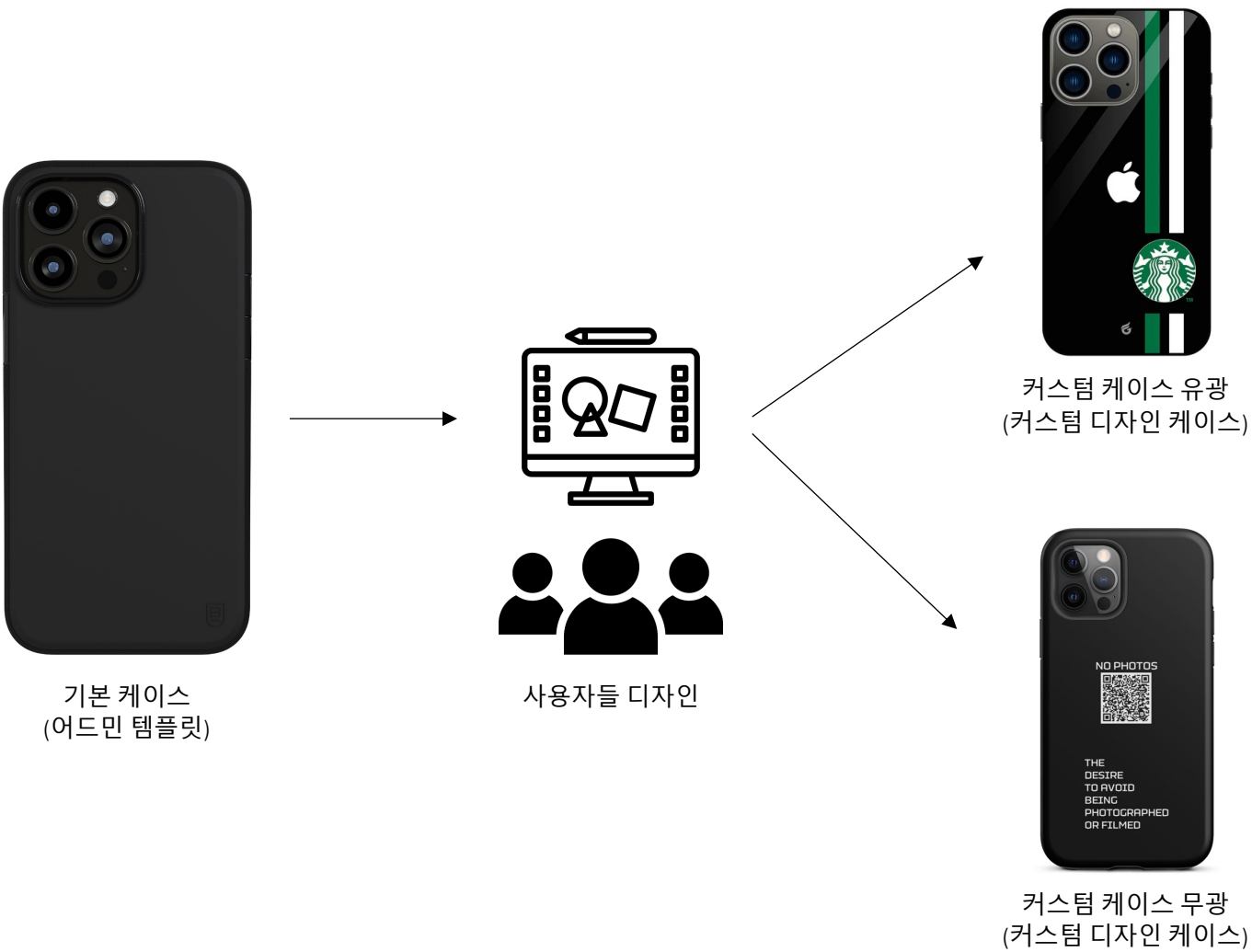
프로젝트 소개

커스텀 휴대폰 케이스를 디자인할 수 있는 플랫폼을 계획했으며 RESTFUL API 를 준비했습니다. 사용자들은 휴대폰 케이스를 자유롭게 디자인을 할 수 있고 그 **디자인을 구매**할 수 있도록 백엔드에서는 연동되어 있습니다. 구매를 하지 않더라도 사용자들은 디자인을 저장, 불러오기 기능이 준비되어 있습니다.

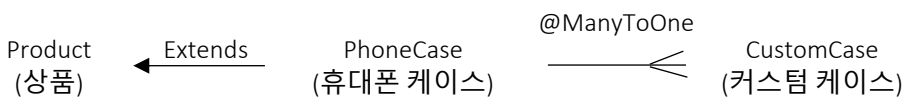
결제 시스템은 ‘부트페이’ 서비스로 구현되어 있으며, 결제되는 순간 사용자들에게 이메일을 발송합니다.

Springboot 8080 포트에서 동작하는 서버가 이 프로젝트의 핵심입니다. 대부분 **비즈니스 로직**에 총 책임 지고 있고, 또한 사용자들의 **데이터를 관리**하고 있습니다.

NodeJS 6000 포트에서 동작하는 서버의 주 역할은 **메시지를 담당**합니다. 예를 들어, 사용자가 회원가입, 주문결제, 결제취소, 휴대폰 문자, 카카오킬 메시지를 담당하고 있습니다.



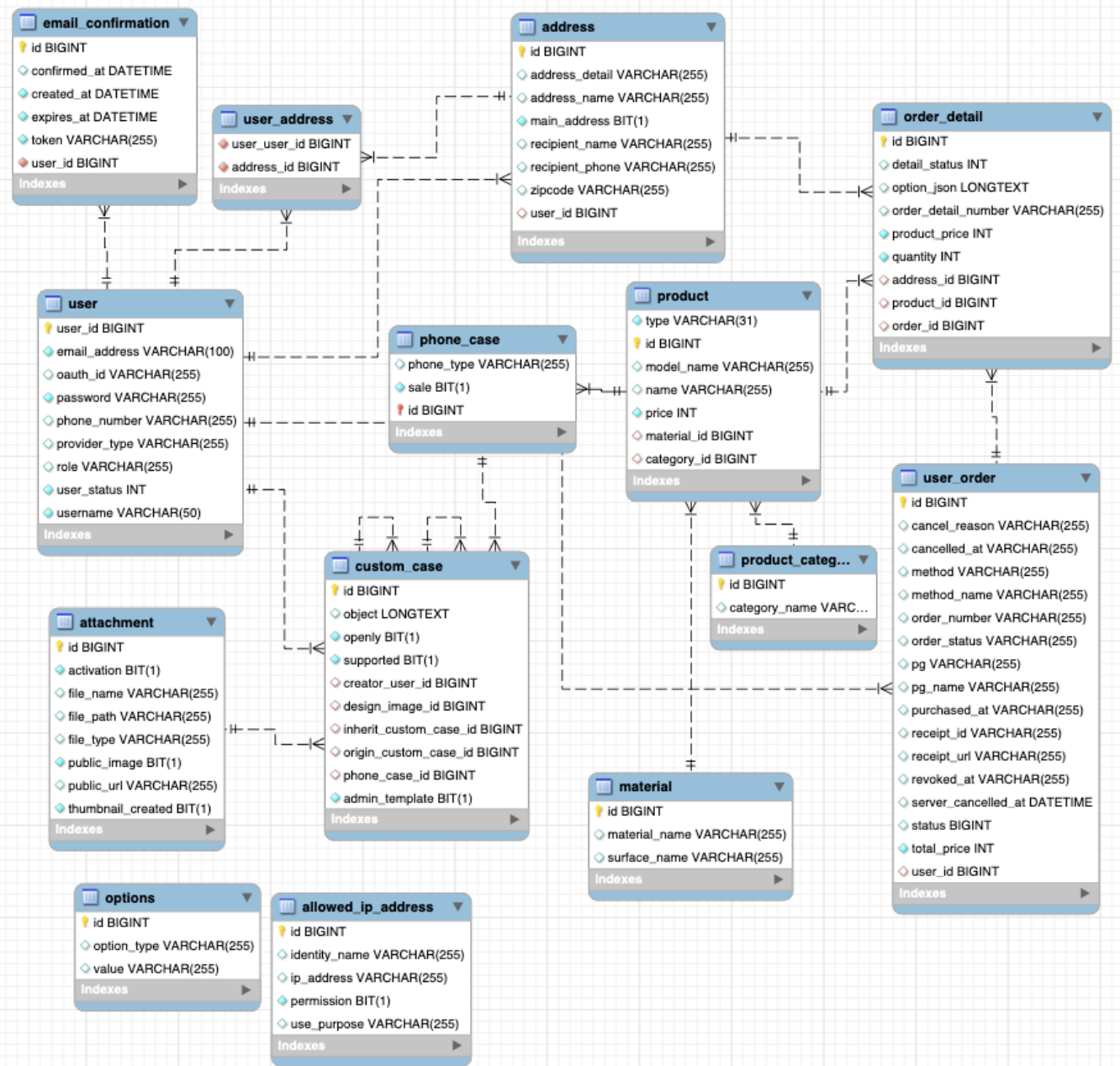
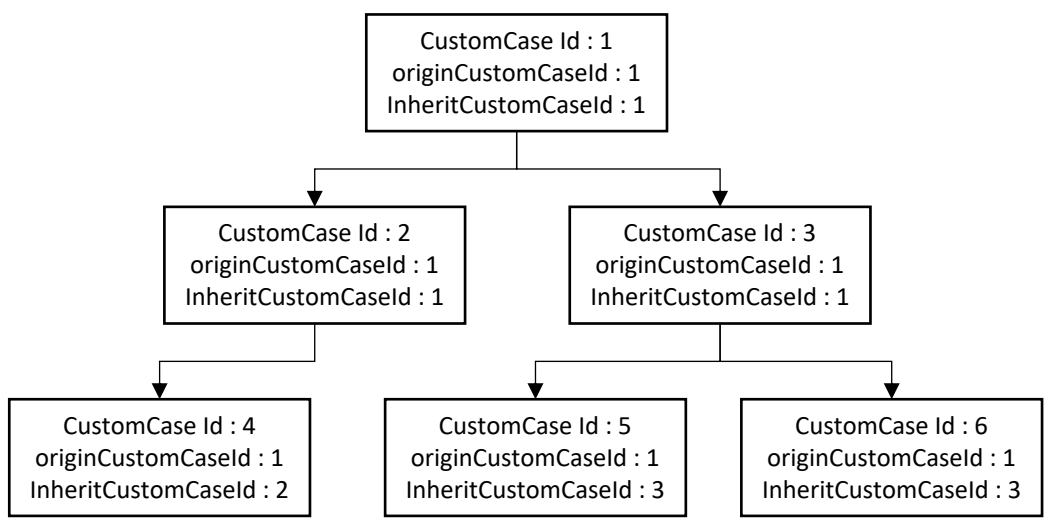
엔티티 특징



PhoneCase 는 Product 으로 부터 상속을 받고 있다.
상속 이유: 앞으로 특수 상품이 등록되면 데이터 관리가 편리하기 때문이다 (예. phoneCase 말고, phoneBag, phoneSticker)

CustomCase 는 PhoneCase 를 @ManyToOne 다대일로 관계 맺어 있으며 하나의 PhoneCase 데이터 위에서 여러 개의 커스텀 디자인이 가능하다.

CustomCase 엔티티 안에는 'origin_custom_case_id' 가 존재한다. 이것은 최초 상속자를 찾을 수 있는 방법이다.
또한, 'inherit_custom_case_id' 는 그 커스텀 케이스 바로 위의 상속자를 의미한다.



엔티티 코드로 소개

```
@Entity
@Getter
@NoArgsConstructor
public class Material {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID")
    private Long id;
    private String materialName;
    private String surfaceName;
}
```

```
@Entity
@Getter
@NoArgsConstructor
@ToString
public class Attachment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    1 usage
    private String fileName;
    1 usage
    private String filePath;
    1 usage
    private boolean thumbnailCreated;
    1 usage
    private String publicUrl;
    1 usage
    private boolean publicImage;
    1 usage
    private String fileType;
    1 usage
    private boolean activation;
}
```

Product .class

```
@Getter
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "TYPE",
    discriminatorType = DiscriminatorType.STRING)
@NoArgsConstructor
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    1 usage
    private String name;
    1 usage
    private String modelName;
    1 usage
    private int price;

    1 usage
    @ManyToOne
    @JoinColumn(name = "MATERIAL_ID")
    private Material material;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "CATEGORY_ID")
    private ProductCategory productCategory;

    1 usage
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "IMAGE_ID")
    private Attachment attachment;

    1 usage
    1 Jaesuk
    public Product(String name, int price, Material material,
        this.name = name;
        this.price = price;
        this.material = material;
        this.attachment = attachment;
    }
}
```

PhoneCase.class

```
@Getter
@Entity
@NoArgsConstructor
@PrimaryKeyJoinColumn(name = "ID") // product
@DiscriminatorValue("PHONE_CASE")
public class PhoneCase extends Product {

    1 usage
    @Enumerated(EnumType.STRING)
    private PhoneType phoneType;
    2 usages
    private boolean sale;

    1 usage
    1 Jaesuk
    @Builder
    public PhoneCase(String name,
        int price,
        PhoneType phoneType,
        Material material,
        Attachment attachment) {
        super(name, price, material, attachment);
        this.phoneType = phoneType;
        this.sale = false;
    }
}
```

Gradle

implementation

```
'org.springframework.boot:spring-boot-starter-data-jpa'
```

```
compileOnly 'org.projectlombok:lombok'
```

```
testImplementation 'org.projectlombok:lombok'
```

```
annotationProcessor 'org.projectlombok:lombok'
```

```
testAnnotationProcessor
```

```
'org.projectlombok:lombok'
```

CustomCase.class

```
@Entity
@Getter
@NoArgsConstructor
@AllArgsConstructor
public class CustomCase {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    2 usages
    @OneToOne(fetch = FetchType.LAZY)
    private Attachment designImage;

    2 usages
    2 usages
    @Lob
    private String object;
    2 usages
    private boolean supported;
    2 usages
    private boolean openly;

    2 usages
    2 usages
    @ManyToOne(fetch = FetchType.LAZY)
    private PhoneCase phoneCase;

    2 usages
    @ManyToOne(fetch = FetchType.LAZY)
    private User creator;

    2 usages
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "ORIGIN_CUSTOM_CASE_ID")
    private CustomCase originDesign;

    2 usages
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "INHERIT_CUSTOM_CASE_ID")
    private CustomCase inheritDesign;

    1 usage
    private boolean adminTemplate;
}
```


JPA

기본적으로 해당 프로젝트는 ORM 방식으로 CRUD 역할을 수행했으며,

또한 'CrudRepository' 에서 기본으로 제공하는 방식들을 잘 활용했다.

JPA 기능을 활용한 경우는 Pagination (페이징 처리) 할 때 또는 JPQL 을 편리하게 사용하기 위한 목적이었다.

UserOrderRepository.interface

```
public interface UserOrderRepository extends JpaRepository<UserOrder, Long> {  
    // 굳이 JPQL 을 여기서 사용할 필요는 없지만, JPQL 활용법을 표현하고 싶었다  
    2 usages   Jaesuk  
    @Query("select o from UserOrder o " +  
            "where o.orderNumber = :orderNumber")  
    UserOrder findByOrderNumberJPQL(@Param("orderNumber") String orderNumber);  
  
    1 usage   Jaesuk  
    @Query("select o from UserOrder o " +  
            "where o.orderNumber = :orderNumber and " +  
            "o.user.id = :userId")  
    UserOrder findMyOrderJPQL(@Param("orderNumber") String orderNumber,  
                               @Param("userId") Long userId);  
}
```

QueryDSL

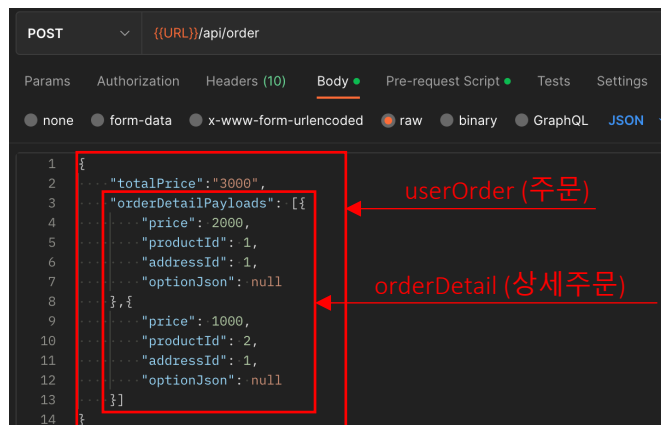
개인적으로 QueryDSL 이 가장 익숙하고 편리하다고 생각한다.

DB 레파지토리로 찾은 결과를 DTO 형식으로 한 번에 반환할 수 있고,

또한 JPQL 에서 복잡한 @OneToMany 를 'Transform(groupBy)' 방식을 QueryDSL 에서 적용하여 데이터를 쉽게 가져올 수 있는 장점이 있습니다. 추가적으로, where() 조건에서 확실히 차이가 많이 납니다.

오른쪽에 있는 'UserOrderQueryDslRepository.class' 는 하나의 사용자의 모든 주문을 수집하는 쿼리입니다.

중간에 Transform 에서 'address' 와 joining 되어 있기 때문에 쿼리는 최종 2번 날리며 모든 데이터를 가져옵니다. (userOrder - 주문, orderDetail - 상세주문)



UserOrderQueryDslRepository.class

```
public List<UserOrderData> getUserOrders(Long user_id) {  
    List<UserOrderData> userOrderDataList = select(  
        Projections.bean(UserOrderData.class,  
            QUserOrder.userOrder.id,  
            QUserOrder.userOrder.orderNumber,  
            QUserOrder.userOrder.totalPrice))  
        .from(QUserOrder.userOrder)  
        .where(QUserOrder.userOrder.user.id.eq(user_id))  
        .fetch();  
  
    List<String> orderNumberList = userOrderDataList.stream() Stream<UserOrderData>  
        .map(UserOrderData::getOrderNumber) Stream<String>  
        .collect(Collectors.toList());  
  
    Map<String, List<OrderDetailData>> transform = getQueryFactory() JPAQueryFactory  
        .from(QOrderDetail.orderDetail) JPAQuery<capture of ?>  
        .join(QOrderDetail.orderDetail.address, QAddress.address)  
        .orderBy(QOrderDetail.orderDetail.id.desc())  
        .where(QOrderDetail.orderDetail.userOrder.orderNumber.in(orderNumberList))  
        .transform(GroupBy.groupBy(QOrderDetail.orderDetail.userOrder.orderNumber)  
            .as(GroupBy.list(Projections.bean(OrderDetailData.class,  
                QOrderDetail.orderDetail.id,  
                QOrderDetail.orderDetail.productPrice,  
                Projections.bean(AddressData.class,  
                    QOrderDetail.orderDetail.address.addressDetail,  
                    QOrderDetail.orderDetail.address.addressName,  
                    QOrderDetail.orderDetail.address.recipientName,  
                    QOrderDetail.orderDetail.address.recipientPhone,  
                    QOrderDetail.orderDetail.address.zipcode  
                ).as(alias: "addressData")))))  
        );  
  
    for (UserOrderData userOrderData : userOrderDataList) {  
        List<OrderDetailData> orderDetailData = transform.get(userOrderData.getOrderNumber());  
        userOrderData.setOrderDetailData(orderDetailData);  
    }  
    return userOrderDataList;  
}
```

API 통신

```
Jaesuk *
@PostMapping("/api/order")
public ResponseEntity<ApiResult> userOrder(
    @RequestBody UserOrderRegisterPayload payload,
    HttpServletRequest request) {
    try {
        UserOrderRegisterCommand command = payload.toCommand();
        addTriggeredBy(command, request);
        String orderNumber = userOrderService.registerOrder(command);
        return Result.ok(orderNumber);
    } catch (Exception e) {
        return Result.failure(e.getMessage());
    }
}

Jaesuk
@PostMapping("/api/order/payment")
public ResponseEntity<ApiResult> payment(
    @RequestBody PaymentPayload payload, HttpServletRequest request) {
    try {
        PaymentCommand paymentCommand = payload.toCommand();
        addTriggeredBy(paymentCommand, request);
        userOrderService.validatePayment(paymentCommand);
        return Result.ok();
    } catch (Exception e) {
        return Result.failure(e.getMessage());
    }
}

Jaesuk
@PostMapping("/api/order/payment/cancel")
public ResponseEntity<ApiResult> cancelPayment (
    @RequestBody PaymentPayload payload, HttpServletRequest request) {
    try {
        PaymentCommand paymentCommand = payload.toCommand();
        addTriggeredBy(paymentCommand, request);
        PaymentResultData result = userOrderService.cancelPayment(paymentCommand);
        return Result.ok(ApiResult.data(result));
    }
}
```

```
@Getter
@Builder
public class UserOrderRegisterCommand extends UserCommand{
    private int totalPrice;
    private List<UserOrderRegisterPayload.OrderDetailPayload> detailCommands;
}
```

UserOrderRegisterCommand 는 UserCommand 을 상속을 받는다.
UserCommand 의 목적은 사용자의 ip 주소와, 쿠키, 유저 아이디를 가져오기 위한 목적이다.

UserOrderRegisterCommand 에서는 payload 값을 받아오며 추가로 내부에서 사용자 관련 객체를 생성한다.

```
public class UserCommand implements TriggeredBy{

    2 usages
    private UserId userId;
    2 usages
    private IpAddress ipAddress;
    2 usages
    private String clientId;
    2 usages
    private boolean admin;

    1 usage Jaesuk
    public void triggeredBy(UserId userId, IpAddress ipAddress, String
        this.userId = userId;
        this.ipAddress = ipAddress;
        this.clientId = clientId;
    }
}
```

```
static void addTriggeredBy(UserCommand command, HttpServletRequest request) {
    Assert.notNull(request.getUserPrincipal(), message: "UserPrincipal 이 요청에 있어야 합니다.");
    SimpleUser simpleUser = getSimpleUser(request);
    command.setAdmin(simpleUser.isAdmin());
    Cookie clientId = WebUtils.getCookie(request, name: "_uuid"); // 프론트에서 쿠키를 생성한다. 그것이 바로 d
    command.triggeredBy(simpleUser.getUserId(), RequestUtils.getIpAddress(request),clientId.getValue());
}
```

addTriggeredBy(command,request) 의 역할을 정리하면

1. 심플 유저로 통해 사용자의 권한을 가져오며
2. 현재 심플 유저로 생성한 유저가 관리자인지, 일반 회원인지 검증한다.
3. 내부 안에서 (프론트엔드) 생성한 쿠키를 확인한다. 여기 쿠키 값이 바로 jwt 의 'access_token' 이다 담겨있을 것이다.
4. 위에서 생성한 UserOrderRegisterCommand 의 ip 주소와, 쿠키, 아이디를 결과값을 다 찾았으며 이제 결과를 넣어주면 된다.

부트페이 검증

```

usage  Jaesuk
public void validatePayment(UserOrder userOrder, PaymentCommand command) throws JsonProcessingException {
    BootpayApiResultData resultData = bootpayService.getVerificationData(command.getReceiptNumber())
    // needs to validate between the server's api and user's payload
    if (!resultData.getOrder_id().equals(userOrder.getOrderNumber())) {
        throw new IllegalArgumentException("주문번호 오류");
    }
    if (resultData.getPrice() != userOrder.getTotalPrice()) {
        throw new IllegalArgumentException("결제금액 오류");
    }
    if (!resultData.getReceipt_id().equals(command.getReceiptNumber())) {
        throw new IllegalArgumentException("영수증 오류");
    }

    userOrder.setBootPayResults(
        resultData.getReceipt_url(),
        (long) resultData.getStatus(),
        resultData.getPg(),
        resultData.getPg_name(),
        resultData.getMethod(),
        resultData.getMethod_name(),
        resultData.getReceipt_id(),
        resultData.getPurchased_at(),
        UserOrderStatus.결제완료
    );
}

```

```

@Override
public void validatePayment(PaymentCommand command) throws JsonProcessingException {
    UserOrder userOrder = userOrderRepository.findByOrderNumberJPQL(command.getOrderNumber());
    bootpayPaymentProcess.validatePayment(userOrder, command);
    userOrder.updateDetailStatus();

    // 이메일
    rabbitMQManagement.sendUserOrderMessage(userOrder);
}

```

부트페이 검증 테스트 코드

```

@Test
@WithUser(value = "admin")
void bootpay_api_test() throws Exception {
    String orderNumber = "2304011";
    String receiptId = "6265f5cce38c300045508c75";

    PaymentPayload payload = new PaymentPayload();
    payload.setOrderNumber(orderNumber);
    payload.setReceiptNumber(receiptId);

    BootpayApiResultData bootpayApiResultData = new BootpayApiResultData(
        receipt_url: "https://app.bootpay.co.kr/bill/Q240ZF1ZcFVW",
        orderNumber,
        price: 5000,
        status: 0,
        pg: "inicis",
        pg_name: "이니시스",
        method: "card_rebill_rest",
        method_name: "ISP / 앱카드결제",
        receiptId,
        purchased_at: "AAAA",
        cancelled_at: ""
    );

    when(bootpayServiceMock.getVerificationData(any()))
        .thenReturn(bootpayApiResultData);

    mvc.perform(post( urlTemplate: "/api/order/payment" )
        .contentType(MediaType.APPLICATION_JSON)
        .cookie(new Cookie( name: "_uuid", value: "abc" ))
        .content(JsonUtils.toJson(payload)))
        .andExpect(status().isOk()).andDo(print()).andReturn();

    UserOrder userOrder = userOrderRepository.findByOrderNumberJPQL(orderNumber);
    UserOrderStatus orderStatus = userOrder.getOrderStatus();
    Assertions.assertThat(orderStatus).isEqualTo(UserOrderStatus.결제완료);
}

```

부트페이 검증

프론트엔드에서 주문번호와 영수증 번호를 스프링 서버로 입력해주면 부트페이 서버로부터 결제정보를 가져올 수 있다.

여기서부터 부트페이 서버로부터 받은 데이터를 검증하며 실제로 사용자가 결제를 정상적으로 마쳤는지 확인한다. 이후, 데이터베이스에 주문상태를 업데이트 한다.

검증 메소드를 정상적으로 마치면 스프링에서는 RabbitMQ 로 메시지를 전송한다. 사용자에게 주문결제 확인서를 이메일로 전송하고 또한, 관리자에게 신규주문이 등록되었다고 알림을 처리하기 위한 목적이다.

부트페이 검증 테스트 코드

오른쪽 그림에 있는 코드는 ' mocking ' 데이터이며 .getVerificationData() 메소드를 만나는 순간 사전에 등록한 데이터 'bootpayApiResultData' 로 대체하며 통과한다.

SMS 서버

주역할은 메시지 관련된 작업을 총책임 지고있다. RabbitMQ Listener 으로 메시지를 받으며 비즈니스 로직에 따라 문자, 이메일을 사용자에게 전송한다.

 jaesuk95/portfolio-sms-nodejs:\$1



이메일 서비스

주문해 주셔서 감사합니다.

주문하신 내역을 확인해주세요.

구매정보	수량	상품금액	진행상태
 모먼트 상품원 1.5 목걸이 14	1	5505	결제완료


배송지 정보

수령인 재식
휴대폰 010-9657-4511
우편번호06766
주소 서울 서초구 우면산로 359 123

결제 정보

주문금액	5505
배송비	0
쿠폰 할인	49545
최종 결제 금액	5505
결제 수단	카카오페이

구매내역 보러가기

 알리고 서비스 (문자, 카카오페이)

모두디자인

010-2809-3485

알림톡 차단

알림톡 차단

모두디자인

알림톡 도착

[문자충전 알림] 모두디자인 알림 서비스입니다.

- QR 코드: nvKgtBQ

- 문자 개수: 2

문자 개수를 모두 사용하셨을 경우 QR 메시지 서비스는 더 이상 전송되지 않습니다. 아래 링크에서 문자 요금을 충전하실 수 있습니다.

또한, 연락처 등록 페이지에서 대표 이메일을 입력하여 언제나 무료로 '우리아이 찾기' 서비스를 받아보세요.

'문자충전 알림' 메시지는 3회까지만 전송됩니다.

문자 요금 충전하러 가기

032-710-2587

[Web발신] [우리아이 찾기] 보호자 재식님, 모두디자인 우리아이 찾기 서비스입니다. 미아방지 목걸이를 통해 임시보호자로부터 연결 요청이 들어왔습니다.

[연락하기] <https://myi.ai/rtc/connection/nvKgtBQ?name=재식>

주의사항

연락을 하면 서로의 위치가 공유되며 상대방의 연락처를 몰라도 영상통화, 채팅이 가능합니다.

2월 28일 화요일


[Web발신] [우리아이 찾기] 보호자 나님, 모두디자인 우리아이 찾기 서비스입니다. 미아방지 목걸이를 통해 임시보호자로부터 연결 요청이 들어왔습니다.

[연락하기] <https://myi.ai/rtc/connection/nvKgtBQ?name=나>

주의사항

연락을 하면 서로의 위치가 공유되며 상대방의 연락처를 몰라도 영상통화, 채팅이 가능합니다.

오전 10:52

 슬랙 알림 서비스

jaesuk95 10:46 PM joined #test.

Portfolio APP 11:08 PM

Portfolio Test 주문 번호 : A123

Portfolio Test 주문 번호 : A123

TESTING : B123

Hello, team!

Let's get this off the ground

Message #test

+ 📎 🗣️ 😊 @ Aa

코드로 소개

서버가 정상적으로 RabbitMQ 와 연동 되었다면, RabbitMQ Listener 는 Queue 를 점검하여 맞는 형식에 따라 메시지를 받아온다.

```
/**
 * RabbitMQ Listener
 */
const consume = ({connection, channel}) => {
  return new Promise( {executor: (resolve, reject) => {
    // // 원하는 Queue 의 이름을 적어준다.
    queue.userRegisterEmail(channel);
    queue.userOrder(channel);
    queue.aligoText(channel); // 알리고 계정 없음
    queue.aligoKakao(channel); // 알리고 계정 없음

    // Queue 가 닫혔거나. 예러가 발생하면 reject
    connection.on('close', (err) => {
      return reject(err);
    })

    connection.on('error', (err) => {
      return reject(err);
    })
  })
}
```

위에 있는 코드는 RabbitMQ listener 코드이며, 'channel' 데이터 안에는 Queue 이름과 데이터가 포함되어 있습니다.

```
queue.userRegisterEmail(channel);
queue.userOrder(channel);
queue.aligoText(channel);
queue.aligoKakao(channel);
```

현재 이번 프로젝트에서는 4가지 queue 가 존재하고 있습니다. 각각 조건이 다르며 비즈니스 로직도 포함한 다른 의도로 설계되었습니다.

```
function userOrder(channel) {
  channel.consume('order', async (msg) => {
    const msgBody = msg.content.toString();
    try {
      // 1. 받은 메시지를 파싱하고.
      const data = JSON.parse(msgBody);
      // await emailService.sendRegisterEmail(data)
      // 2. 잘 받았으니 ACK를 보내자.
      await channel.ack(msg);

      slackService.userOrderAlert(data)
      await emailService.sendUserOrderEmail(data);
    } catch (e) {
      let parse = {}
      parse.body = msgBody.toString()
      parse.tag = "userOrder"
      parse.errorMessage = e.message

      await channel.sendToQueue(failedQueue, Buffer.from(JSON.s
      await channel.nack(msg, true, false)
    }
  })
}
```

알맞는 Queue 가 있으면 내부안에 있는 비즈니스 로직을 수행합니다.

이번 포트폴리오 예시로 주문 관련 데이터가 어떻게 동작하는지 설명하도록 하겠습니다.

slackService.userOrderAlert(data) 의 역할은 관리자에게 신규주문이 등록되었다고 슬랙 메시지를 전송합니다.

그 이후, 오른쪽 그림과 같이 이메일 서비스로 들어가서 데이터를 EJS 형식으로 생성하고, html 템플릿에 render 를 해주면 원하는 데이터를 사용자에게 이메일에 포함해 발송합니다.

```
async function sendUserOrderEmail(data) {
  const userOrder = data.variables.userOrder;

  // userOrder -> orderDetail (@OneToMany)
  let ejsArrayData = [];
  for (const orderDetail of data.variables.userOrder.orderDetails) {
    ejsArrayData.push({
      productName: orderDetail.productName,
      orderStatus: userOrder.status, // detailOrderStatus === '주문접수'
      quantity: orderDetail.quantity,
      productPrice: orderDetail.productPrice,
      // 주소
      postal: orderDetail.postal,
      addressDetail: orderDetail.addressDetail,
      name: orderDetail.username,
      phone: orderDetail.phone
    });
  }

  let ejsData = {
    email: data.receiver,
    totalPrice: userOrder.totalPrice,
    paymentMethod: userOrder.method,

    orderDetails: ejsArrayData
  }

  let template = findEmailTemplate(data);

  transporter.sendMail({
    data: {
      from: data.sender,
      to: data.receiver,
      subject: data.variables.userOrder.title,
      html: ejs.render(await template, {data: ejsData}),
      // text: ejs.render(await template, {data: ejsData}),
      ses: {
        Tags: [
          {
            Name: "tag_name",
            Value: "tag_value",
          },
        ],
      },
    },
  })
}
```

LOCALHOST



1. Push to Github Repository



2. Github Webhook (빌드 유발)

JENKINS http://192.168.64.2:8080



PIPELINE (스프링 기준)

```
Pipeline {
  stages {
    stage('clone git')
    stage('GIT MSG')
    stage('Image Version')
    stage('Slack notify start & success/failure')
    stage('Build Gradle')
    stage('Build Image')
    stage('Docker Login')
    stage('Dockerhub Push')
    stage('remote')
  }
}
```

Detailed Explanation (스프링 기준)



- 1. SLACK 어플리케이션으로 개발자에게 알림
- 2. Gradle 을 빌드한다, 이유: Dependency Management - 의존성 관리를 하며 테스트 코드도 또한 실행한다.
- 3. .jar 파일 생성, 이유: 압축 된 zip 파일이며 자바로 만든 클래스들을 한데 묶어 놓는다.
- 4. 도커 이미지 빌드 -> 도커허브에 저장

젠킨스 파이프라인 코드는
여기서 확인하시면 됩니다

[https://github.com/jaesuk95/
portfolio-jenkins-pipeline](https://github.com/jaesuk95/portfolio-jenkins-pipeline)

5. spring-docker-run.sh
shell 스크립트를 찾아
user 권한으로 실행한다

Jenkins Pipeline sshCommand
'bash */spring-docker-run.sh \${version}'

4. Jenkins Pipeline Remote -
젠킨스 파이프라인으로 우분투
서버에 ssh cli 를 입력한다

UBUNTU 22.04 LTS



spring-docker-run.sh Shell script 파일

```
// 앞에 부분 생략
docker pull jaesuk95/portfolio:$1

docker run --name=spring -v
/home/user/app/spring:/home/central/app/spring -p
8081:8081 jaesuk95/portfolio:$1
```

DOCKER



NETWORK 172.17.0.1



jaesuk95/portfolio:\$1



jaesuk95/portfolio-sms-
nodejs:\$1

환경 변수

스프링부트 어플리케이션을 가상서버에 동작할 경우, 환경 변수를 사전에 설정해야 한다.

스프링에서는 @Profiles 어노테이션을 간단한 방법으로 제안하지만, 개인적으로 이 방법은 github repository 등록을 해야만 하고 보안에 취약하다는 단점이 있다.

Github Repository 가 비공개여도 불안함이 존재하며 결국 환경 변수를 가상 서버에 저장했다. 스프링일 경우


“/home/central/app/spring/application.properties”

위치에 저장되어 있고 가상서버에 실행할 때 Dockerfile 에 경로를 잘 명시해주면 된다, nodeJS 경우

“/home/central/app/sms/.env”

위치에 있다.

Shell 스크립트

.SH

```
root@central:/home/central/scripts# cat spring-docker-run.sh
echo $1
docker stop spring || true
docker rm spring || true

docker login -u jaesuk95 -p password

docker pull jaesuk95/portfolio:$1
docker run -d --name=spring -v /home/central/app/spring:/home/central/app/spring -v /home/central/file:/home/central/file -p 8081:8081 jaesuk95/portfolio:$1
root@central:/home/central/scripts#
```

.SH

```
root@central:/home/central/scripts# cat sms-docker-run.sh
echo $1
npm install
Nom run build
docker stop sms || true
docker rm sms || true

docker login -u jaesuk95 -p password

docker pull jaesuk95/portfolio-sms-nodejs:$1
docker run -d --name=sms --env-file /home/central/app/sms/.env -p 6000:6000 jaesuk95/portfolio-sms-nodejs:$1
root@central:/home/central/scripts#
```

```
root@central:/home/central/scripts# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
60ad3f0de5f8	jaesuk95/portfolio-sms-nodejs:1.0.1	"docker-entrypoint.s..."	30 seconds ago	Up 29 seconds	0.0.0.0:6000->6000/tcp, :::6000->6000/tcp	sms
50bf394562ad	jaesuk95/portfolio:0.0.28	"java -Dspring.conf..."	About a minute ago	Up About a minute	0.0.0.0:8081->8081/tcp, :::8081->8081/tcp	spring

```
root@central:/home/central/scripts#
```


스태틱 이미지

스프링 서버에서 이미지를 등록할 수 있게 API가 준비되어 있고 (POST method “/api/image”) Linux OS 에서 이미지를 저장하는 위치는 /home/central/file/{오늘날짜}/image.png 입니다.

가상 서버는 Linux OS 이며 오른쪽 첫 번째 이미지에서 directory path 로 이미지의 저장 위치를 확인하실 수 있습니다.

그 아래 있는 이미지는 nginx 의 config 설정입니다. 이미지를 Public URL 으로 가져오는 방식은 root /home/central/ 경로를 지정했기 때문에 {ip 주소}/file/* 으로 시작하는 주소는 다 가져올 수 있게 설정되어 있습니다.

예시.
http://192.168.64.2/file/230403/e4ef8d22-1e3e-4326-bf8f-209ba5a11243-dogecoin.png



```
root@central:/home/central/file/230403# pwd
/home/central/file/230403
root@central:/home/central/file/230403# ls -l
total 192
-rw-r--r-- 1 root root 48404 Apr  4 04:20 540afc32-72d1-48d8-982e-e1348dc08072-dogecoin.png
-rw-r--r-- 1 root root 48404 Apr  4 04:03 804017e6-1bdf-44c8-b14a-ac4c2166f7a2-dogecoin.png
-rw-r--r-- 1 root root 48404 Apr  4 03:42 d6eb20fb-9644-4f5a-bcd1-44d535496ef2-dogecoin.png
-rw-r--r-- 1 root root 48404 Apr  4 04:36 e4ef8d22-1e3e-4326-bf8f-209ba5a11243-dogecoin.png
root@central:/home/central/file/230403#
```

```
server {

    listen 80;
    listen [::]:80;
    listen 192.168.64.2;

    location /api {
        proxy_pass          http://172.17.0.1:8081;
        proxy_redirect      off;
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /api/sms {
        proxy_pass          http://172.17.0.1:6000;
        proxy_redirect      off;
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /file {
        if ($request_method = 'GET') {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET, OPTIONS';
            add_header 'Access-Control-Allow-Headers' 'DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type';
        }
        root /home/central/;
    }
}
```