

CAMPANDTENT

풀스택

팀 프로젝트

리액트 & 코틀린 기반 웹

명승정 포트폴리오

**프로젝트 개요**

# 프로젝트 주제 및 동기

## 주제

- 리액트 & 코틀린 기반
- 캠핑 용품 판매 온라인몰 & 백오피스 서비스

## 동기

- 야외 활동 증가와 함께한 휴양 문화 부상 따른 캠핑 수요증가
- 자연 친화적 라이프스타일 증가
- 소셜 미디어 영향력과 캠핑 커뮤니티 확장

# 사용기 | 술

통신



데이터베이스



백엔드 프레임워크  
/간단한 API



사용언어



HTML



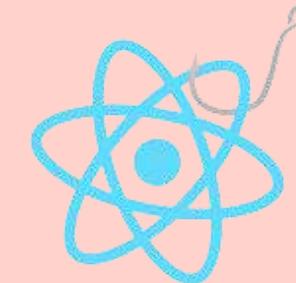
CSS



JS



프론트엔드 프레임워크  
/간단한 API



# 사례 분석

## [배달의 민족]

• 요식업계 사업자들이 주로 활용하는 서비스 중 하나인 배민의 가게 관리 시스템을 조사했습니다.

• 업장을 효과적으로 관리하기 위해 사용자의 편의성 향상, 정보 전달 강화, 시각적인 인상을 중점으로 벤치마킹했습니다

**김배민** 사장님,

입금 예정 금액은 **4,701,934원** 입니다.

배달의민족에서 광고 성과는 **53배**입니다.

입금 예정 금액 ②

[신규 3] [공지] 광고 가입 문의 증가에 따른 광고 등록 지원 안내 NEW

2020. 09. 01

주간매출 >

Day	Sales (만원)
월	70
화	70
수	105
목	35
금	35
토	70
일	105

주간주문수 >

Day	Order Volume
월	25
화	25
수	45
목	35
금	35
토	25
일	35

요청 처리 현황 >

최근 1개월

Status	Count
대기	0 건
진행	0 건
반려	0 건
취소	1 건
승인	1 건

우리 가게 리뷰 >

전체

Category	Count
미답변	616 건
지난 의심	10 건
0 건	0 건

새로운 시장이 기다리고 있습니다.  
확인해보세요.

우리가게 매출 언제 입금되는지  
무료로 알려드려요.

# 사례 분석

## [아임웹]

- 재고 관리 및 상품 등록과 같은 판매자가 필요로 하는 UI를 벤치 마킹했습니다.
- 온라인 물에서 상품 판매시 필요한 데이터를 수집하였습니다.

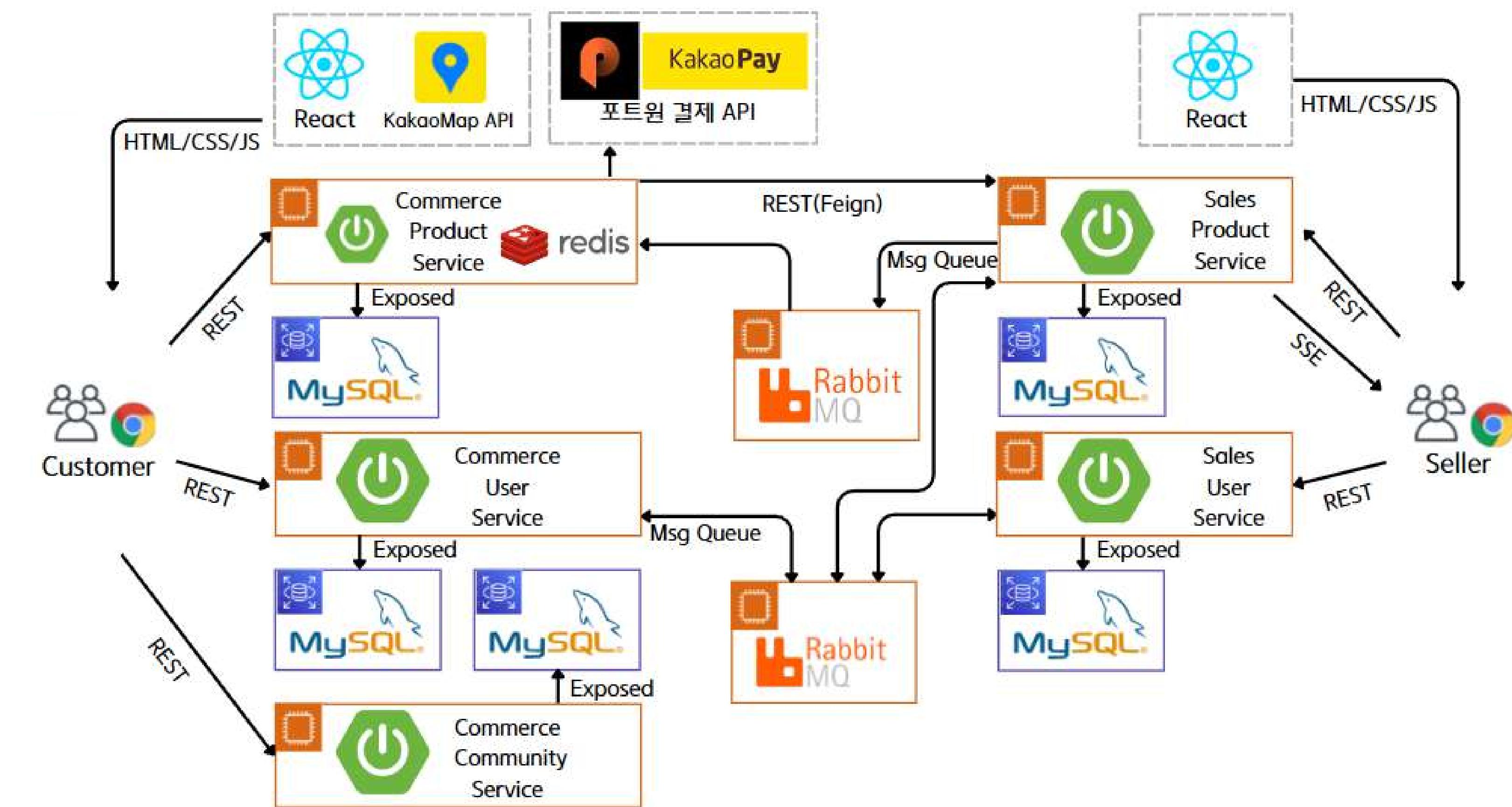
상품관리

상품 일괄 추가 상품 추가

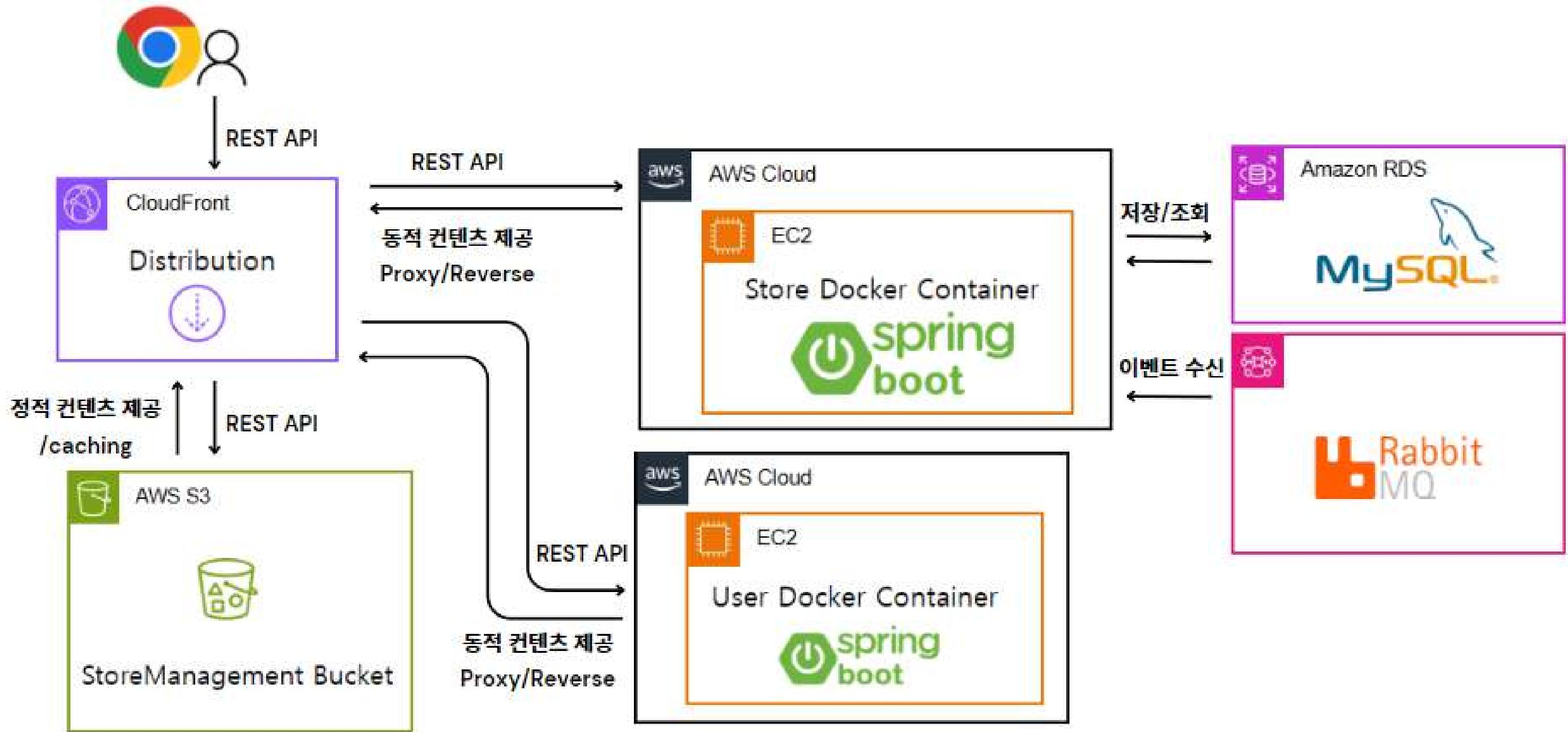
전체 19 판매중 19 품절 0 숨김 0
20개씩 보기 ▾

No	상품명	판매가	카테고리	기획전	상태	재고	등록일	수정일
106	나이키 에어포스1 블랙/화이트 로우 [직접배송] 블랙애로우 남성용 손목시계 이름 각인 서비스	10,000원 290,000 원 119,000 원	NEW ARRIVAL +1 ACCESSORY +1 NEW ARRIVAL +2	- - -	판매중 판매중 판매중	58 - 395	2022-03-31 2020-01-14 2021-05-13	2022-03-31 2022-03-17 2022-03-17
27	Unsplash 로고 플레인 티셔츠	39,000원	TOP +2	2021년 시즌오프	판매중	800	2019-07-15	2022-03-31
26	사이드 솔더 라인 레터링 티셔츠	39,000원	TOP +2	2021년 시즌오프	판매중	-	2019-07-15	2022-03-31
25	컬러 레터링 티셔츠	39,000원	TOP +2	2021년 시즌오프	판매중	-	2019-07-15	2022-03-31
24	웨이브 라인 패턴 티셔츠	39,000원	TOP +2	2021년 시즌오프	판매중	-	2019-07-15	2022-03-31
6	레이디나이트 워싱 스키니진	59,000원	PANTS +2	-	판매대기	120	2017-11-15	2022-03-24

# 팀 애플리케이션



# 개인 애플리케이션



# 팀 편성 및 역할



김성우

기술고문  
메인 페이지  
제품 관련 노출  
카카오 결제 API  
담당 페이지 UI 디자인



박강찬

구매 유저 관리  
제품 구매 / 관리  
유저 리뷰 / 문의  
제품 장바구니  
브랜드 좋아요  
담당 페이지 UI 디자인

IT ME!



명승정

상품 등록  
재고 관리  
주문 처리  
매출 관리  
판매 현황  
알림 기능  
공통 컴포넌트 디자인  
담당 페이지 UI 디자인



남소나

게시판  
카카오 맵  
담당 페이지 UI 디자인



양수열

고객 주문 통계  
판매자 프로필 등록  
리뷰 / 문의 관리  
일정관리  
담당 페이지 UI 디자인

**부서 및 설계**

# 요구사항 분석

## Must have

번호(중요도)	우선순위	요구사항명	요구사항 내용
1	Must have	제품 등록	판매자가 제품의 이름, 판매가, 상품 상태(숨김, 판매), 최대 주문수량, 할인 %, 상품 대분류 상품 설명, 대표 이미지, 서브 이미지를 등록 할 수 있다.
2	Must have	제품 목록	판매자가 등록한 제품의 목록을 보여준다.
3	Must have	제품 관리	판매자가 등록된 제품의 상태(숨김, 판매), 할인 %, 제품의 이름, 판매 가, 재고의 수량, 제품의 가격, 주문 최대 수량, 카테고리 등을 변경 할 수 있다.
4	Must have	주문 목록	구매자가 요청한 주문을 처리 완료, 처리 실패로 목록을 가져온다.
5	Must have	주문 처리	구매자가 요청한 제품이 재고가 있다면 자동으로 수락처리, 재고가 0이라면 자동으로 숨김 처리 되어서 주문을 하지 못한다.
6	Must have	판매 현황	선택한 연도에 맞춰 월별 카테고리 판매량 그래프를 보여준다.
7	Must have	매출 관리	선택한 연도들의 연간 매출을 그래프로 보여주며, 이번달 정산 금액과 저번 달의 정산 금액을 비교하여 나타낸다.
8	Must have	Top5 프로모션	전체 브랜드 중에서 카테고리 별로 판매량이 가장 높은 제품 5개씩 보여준다.

## Should have

번호(중요도)	우선순위	요구사항명	요구사항 내용
15	Should have	주문 알림	구매자가 주문한 제품 ID를 실시간 알림으로 보여준다.
16	Should have	재고 부족 알림	판매자가 보유하고 있는 제품 중 재고의 갯수가 3개 이하인 제품을 알려준다.
17	Should have	월간 주문 현황	월간 총 주문 현황(성공, 실패)을 간략히 보여준다.
18	Should have	제품 현황	판매자가 등록한 제품의 상태 별 현황을 간략히 보여준다.

## Won't have

번호(중요도)	우선순위	요구사항명	요구사항 내용
19	Won't have	채팅	구매자와 판매자 간의 실시간 문의 기능

# 인터페이스 명세서

## 서비스간 연동 정리

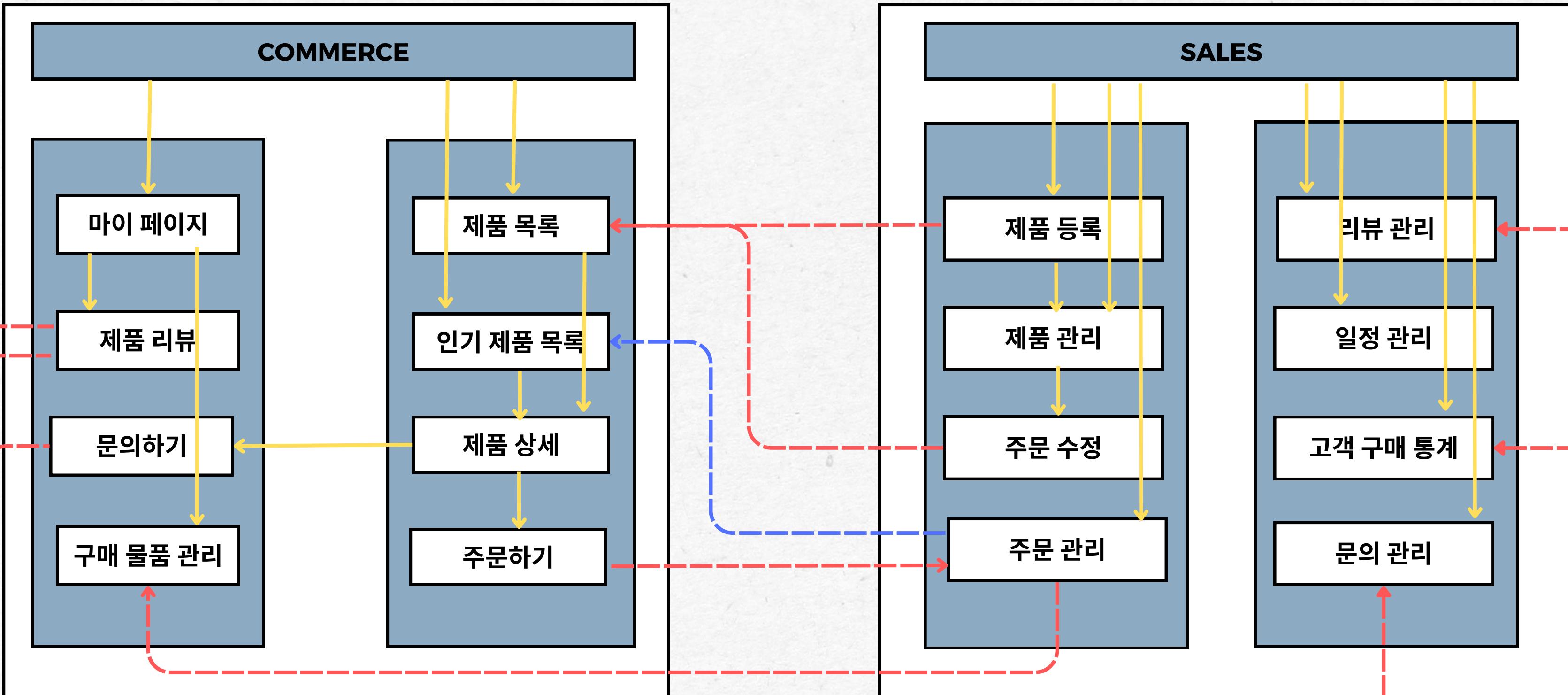
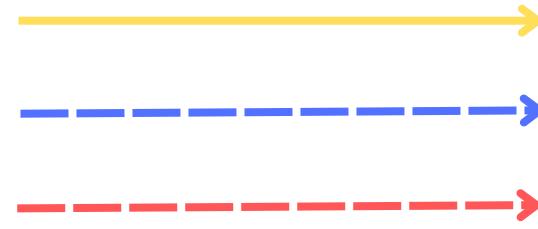
번호	이름	전송/요청 서비스	수신/응답 서비스	수신/응답 서비스	연동방법	연동주기	전송/요청데이터	수신/응답 데이터
1	Must have	제품 등록	sales	commerce	Message Queue	실시간 (요청)	제품 ID, 가격, 설명, 메인 이미지, 서브 이미지 등 전송	비동기
2	Must have	제품 수정	sales	commerce	Message Queue	실시간 (요청)	수정하는 제품의 ID, 수정된 데이터 전송	비동기
3	Must have	제품 이미지	commerce	sales	RestAPI	실시간 (요청)	제품 이미지 uuid 요청	이미지 uuid에 해당하는 이미지 응답
4	Must have	주문 처리	commerce	sales	Message Queue	실시간 (요청)	구매자가 주문 ID, 제품 ID, 수량 전송	주문처리 성공/실패 응답
5	Must have	프로모션 (할인)	sales	commerce	Message Queue	실시간 (요청)	할인하는 제품의 ID와 할인 % 전송	비동기
6	Must have	카테고리별 탑 5 제품	commerce	sales	RestAPI	스케줄 (1hour)	제품 ID 요청	카테고리별 탑 5 제품 ID 응답
7	Must have	판매자 정보	sales(product)	sales (user)	RestAPI	실시간 (요청)	브랜드 이름 요청	브랜드 이름 응답

# 프로세스 흐름도

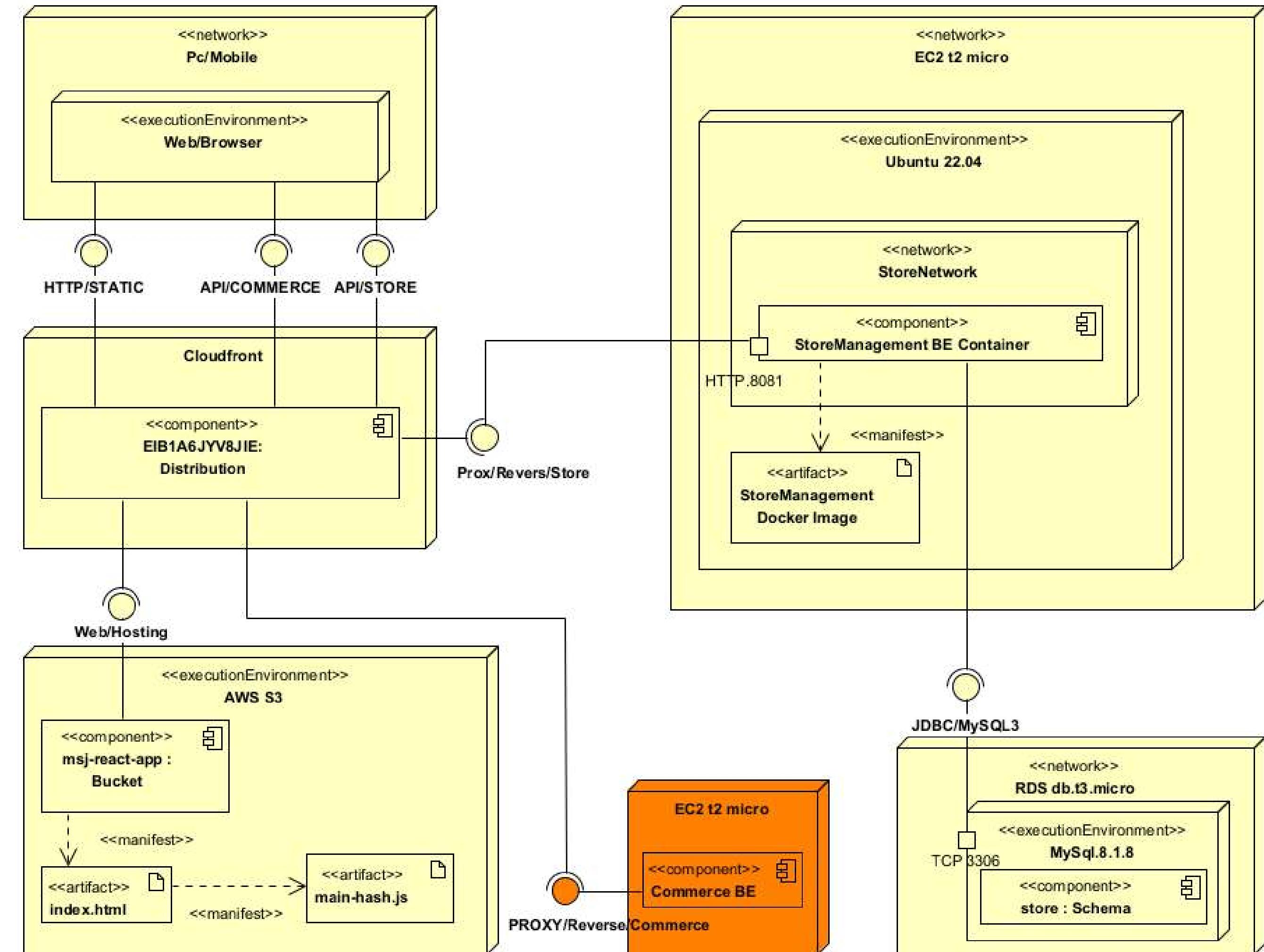
화면 이동

데이터 이동 (동기)

데이터 이동 (비동기)



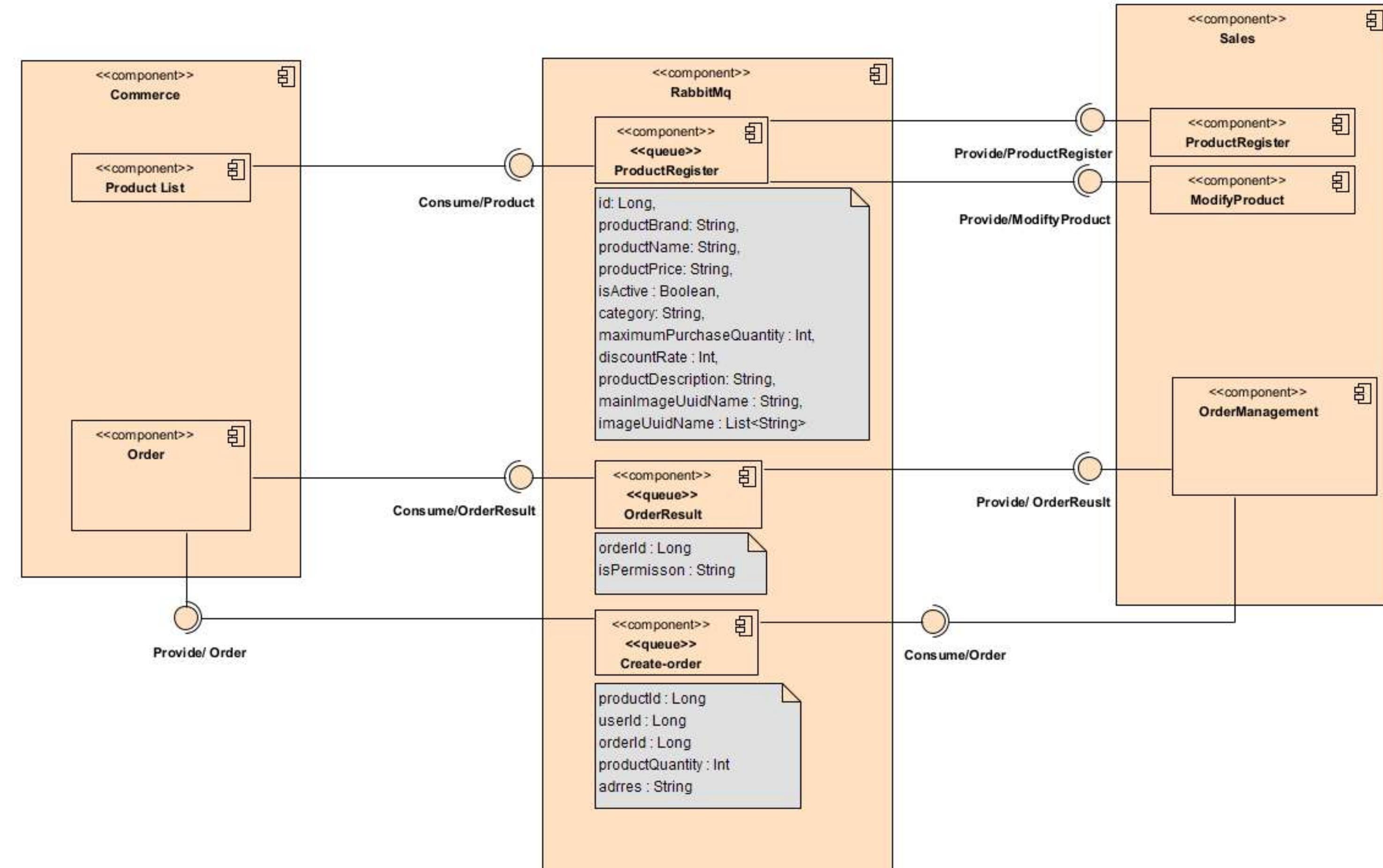
# 배포 다이어그램



# Component Diagram

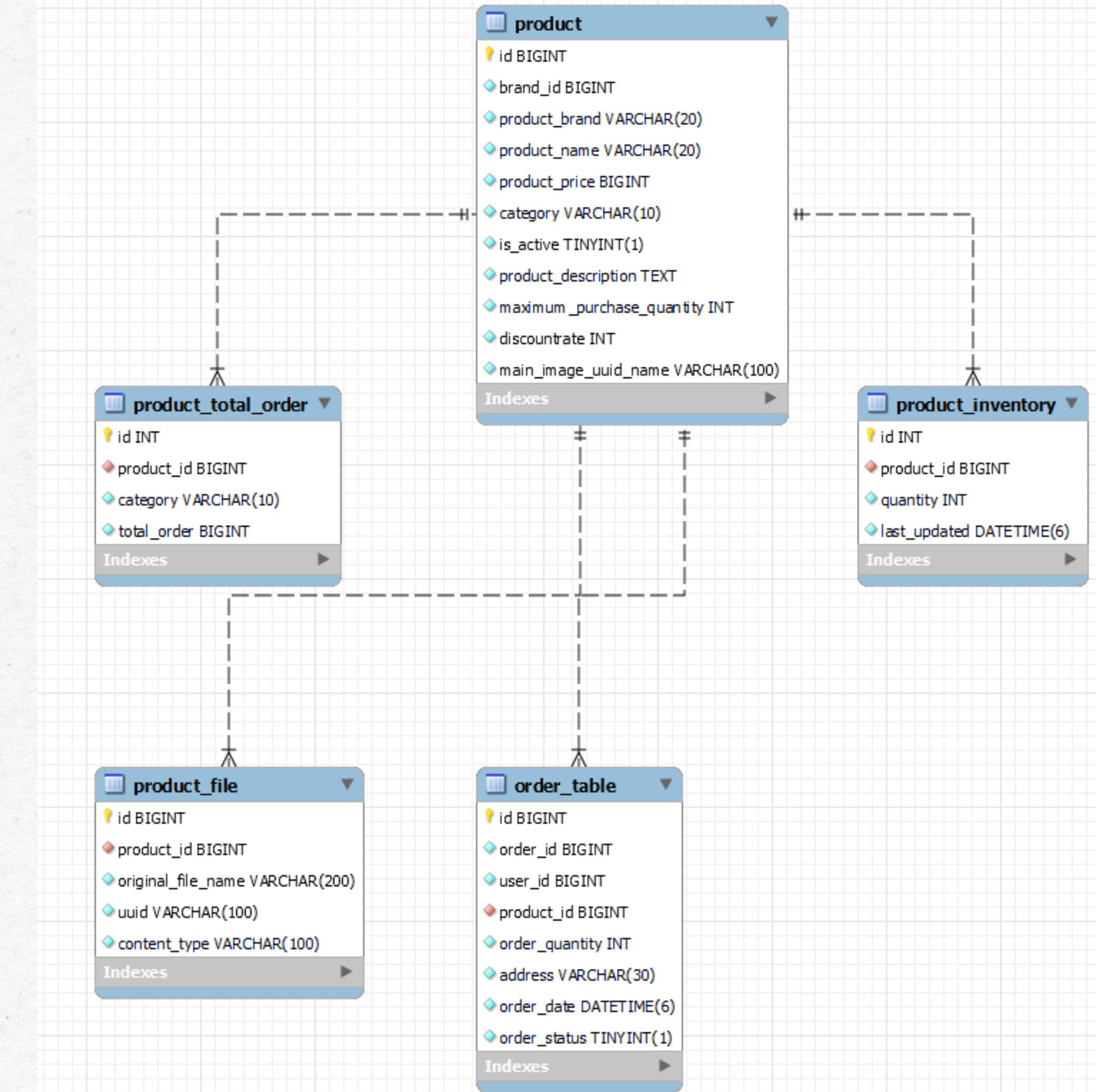
원활한 개발진행을 위해 컴포넌트 다이어그램을 설계 하였습니다.

소프트 웨어 공학 분석 설계 툴로 유명한 visual paradigm을 사용했습니다.



# ERD 다이어그램

ERD 다이어그램을 통해 테이블간의 관계를 나타냈습니다.



# UI 프로토타입

# KaKao Oven

상품 등록	
<b>가게 관리</b>	
상품 등록	상품명
재고 관리	판매가
주문 처리	
구매 이력 관리	상품 코드
매출 관리	
<b>고객관리</b>	
	상품 분류
	<div style="border: 1px solid #ccc; padding: 5px; width: fit-content;">         Dropdown         <div style="float: right;">▼</div> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;">         Dropdown 1          Dropdown 2       </div>
	상품 요약설명
	상품 상세 설명
	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <p>대표 이미지</p> </div> <div style="text-align: center;"> <p>목록 이미지</p> </div> <div style="text-align: center;"> <p>목록 이미지</p> </div> <div style="text-align: center;"> <p>목록 이미지</p> </div> </div>
	이미지

The chart displays the distribution of customer acquisition power across different product categories. The Y-axis represents the power level, ranging from 0 to 100. The X-axis lists categories: 텐트, 의자, 침낭, 램프, 모포.., and several unlabeled categories represented by ellipses (...). For each category, there are two bars: a blue bar representing one metric and a green bar representing another. In general, the blue bars are consistently higher than the green bars across all categories.

Category	Blue Bar (Metric A)	Green Bar (Metric B)
텐트	~85	~65
의자	~75	~55
침낭	~90	~60
램프	~70	~50
모포..	~80	~65
...	~85	~60
...	~75	~55
...	~90	~65
...	~70	~50
...	~80	~65

The screenshot shows a mobile application for managing a shop. The top navigation bar includes a logo, the title '내 가게 관리' (Manage My Shop), and a search bar. On the left sidebar, there are links for '상품 등록' (Product Registration), '재고 관리' (Inventory Management), '주문 처리' (Order Processing), '구매 이력 관리' (Purchase History Management), and '매출 관리' (Sales Management). The main content area features a chart titled '전체 매출' (Total Sales) with a dropdown menu for '주간 매출' (Weekly Sales). Below the chart, there are two sections: '내 가게 리뷰' (Review of My Shop) with buttons for '전체 리뷰 갯수' (Total Review Count) and '별점 평점' (Rating Points), and '요청 처리' (Request Processing) with buttons for '대기' (Pending), '요청' (Request), and '처리' (Handle).

# 프로젝트 구현

## UI 구현

## [메인 페이지]

## • 알림 useState부분

```
const [messages, setMessages] = useState<string>([]);

useEffect(() => {
  const eventSource = new EventSource(
    "http://192.168.100.151:8080/order/notifications"
  );
  eventSource.onmessage = (event) => {
    const eventData = event.data;
    setMessages([...prevMessages, eventData]);
  };
});
```

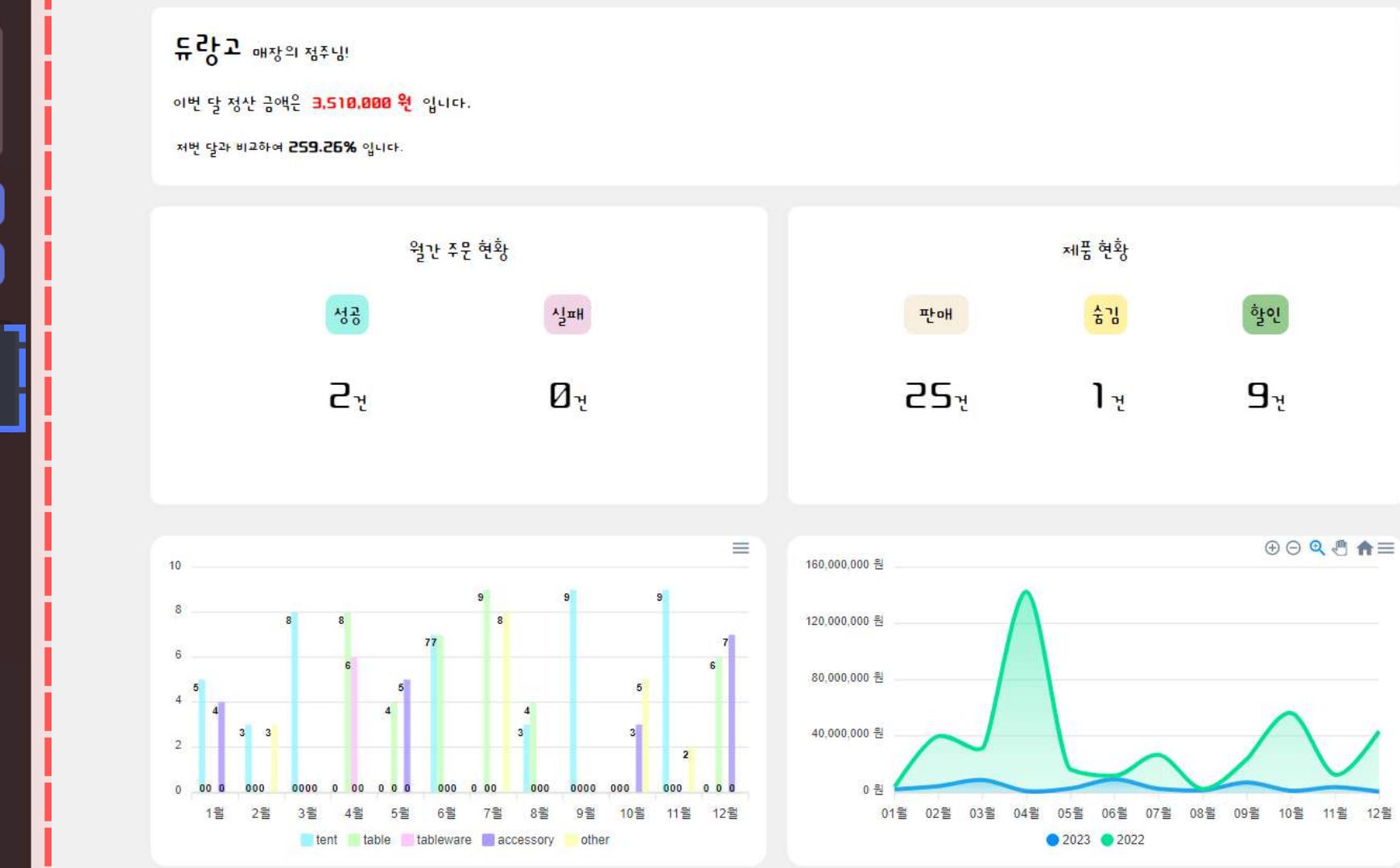
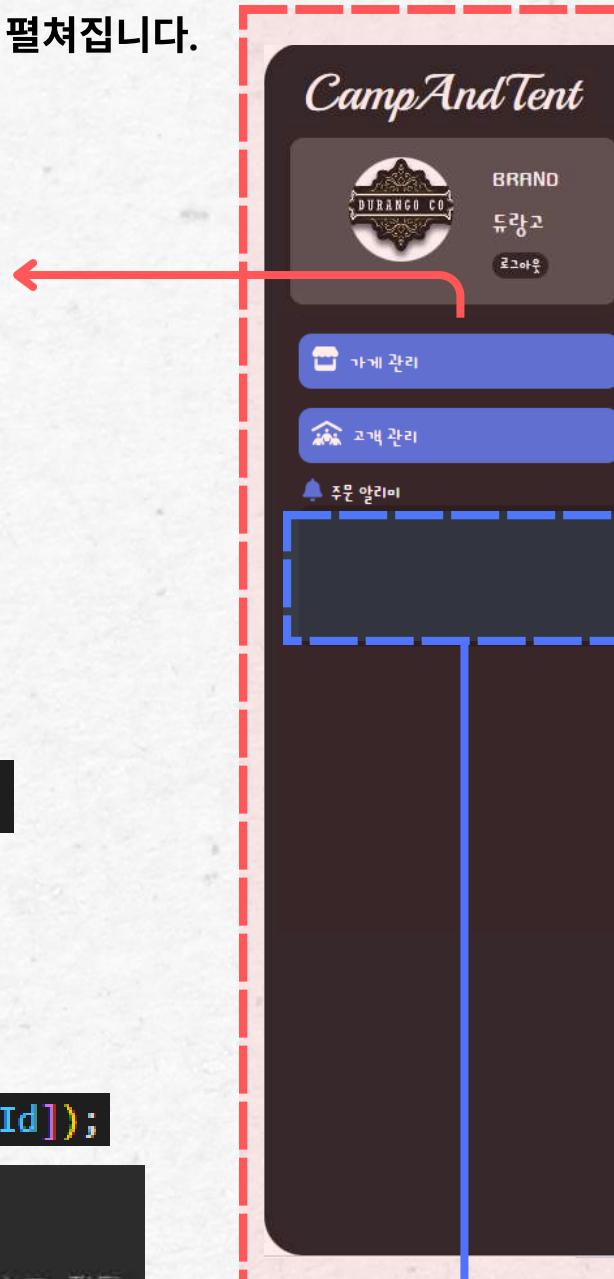
```
@Auth
@RabbitListener(queues = ["product-payment"])
product-payment에 있는 queue receiveOrder(message : String) 매개변수로 전달
fun receiveOrder(message : String){
  JSON 문자열(message)을 OrderRequest 클래스의 인스턴스로 변환
  val orderRequest : OrderRequest = mapper.readValue(message)
}
```

```
val deadEmitters : MutableList<SseEmitter> = ArrayList()
```

```
for (emitter in emitters) {
  try {
    emitter.send(message)
  } catch (e: IOException) {
    deadEmitters.add(emitter)
  }
}
```

주문 발생 시 실시간 주문 알림  
(Server-Sent-Event)

- 클릭 시 네비게이션 바가 펼쳐집니다.

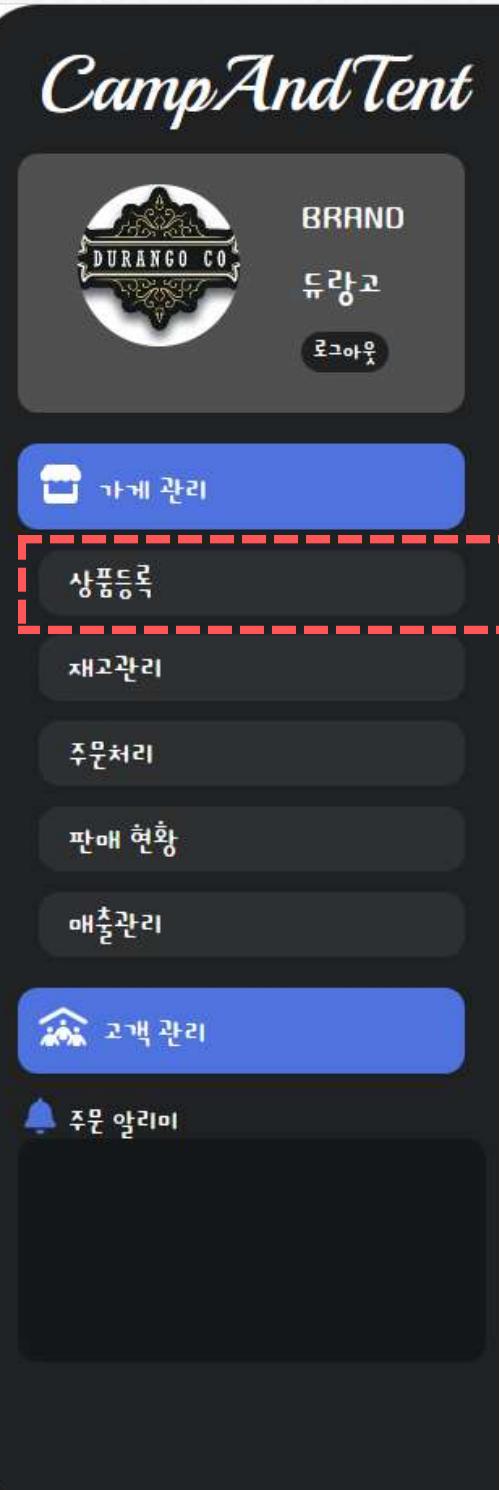


## 공통 컴포넌트

- SSE (Server-Sent-Event) 방식을 사용해 주문 알림 기능을 구현했습니다.
- 통신의 선제조건으로 Client와 Server가 지속적으로 연결되어 있어야합니다.
- 주문 이벤트가 발생하면 서버에서 클라이언트로 데이터를 푸시합니다.
- 알림의 state가 변경되면 해당 부분만 리랜더링합니다.

## UI 구현

## [상품 등록]



상품등록 클릭시 구매서버로 queue 전송

**상품 등록**

브랜드: DURANGO CO. 듀랑고

상품명: 듀랑고 에어텐트

판매가: 492000

상품 상태: 판매

주문 수량: 주문 최대 수량

할인: 0

상품 대분류: 옵션을 선택해주세요

상품 설명: 상품 상세설명을 적어주세요

대표 이미지: 파일선택 16786876327m2.jpg

상품 이미지: 파일선택 파일 2개

Click! **상품 등록**

## 상품정보를 입력하지 않았을 때 경고창

localhost:5000 내용:

필수 정보를 모두 입력해주세요.

확인

- 사용자가의 올바른 데이터 입력을 위해 유효성 검사를 추가했습니다.

```
const handleProductRegister = (e: React.FormEvent) => {
  e.preventDefault();

  if (
    !productNameRef.current.value ||
    !productPriceRef.current.value ||
    !maximumQuantityRef.current.value ||
    categoryRef.current.value === "null"
  ) {
    alert("필수 정보를 모두 입력해주세요.");
    return; // 폼 전송을 중단
  }
  if (!fileRef.current.files.length) {
    alert("이미지 파일을 선택해주세요.");
    return; // 폼 전송을 중단
  }
}
```

## 업로드 이미지 미리보기

`onChange={handleMainImgPreview}`

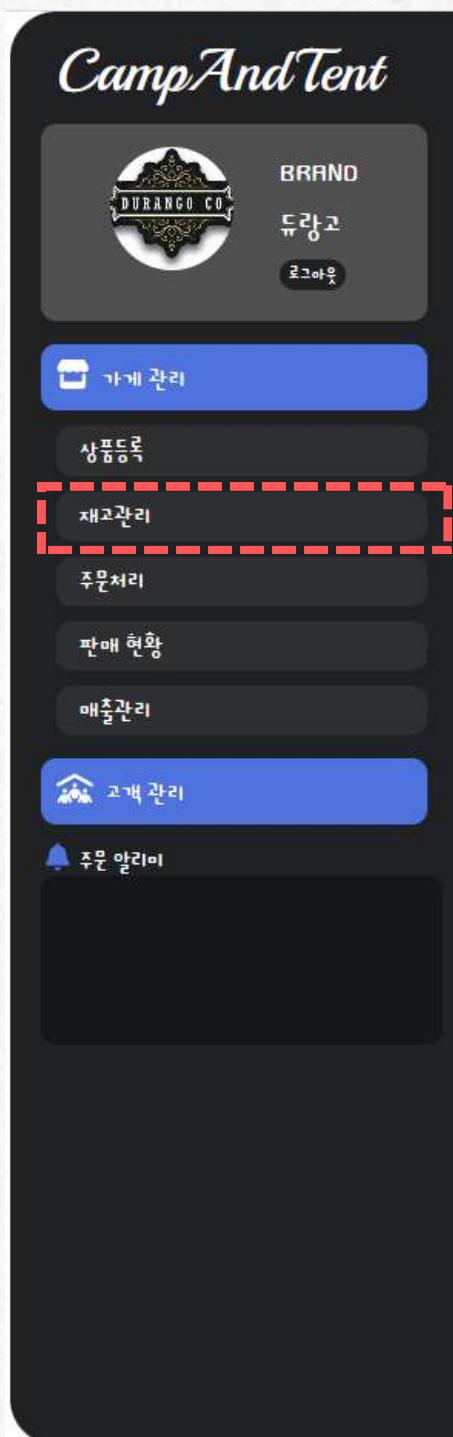
```
const handleMainImgPreview = (event) => {
  if (event.target.files.length > 1) {
    alert("대표 이미지는 1개만 선택할 수 있습니다.");
    event.target.value = ""; // 선택한 이미지를 지움
    return;
  }
  const files = event.target.files;
  const previews = [];

  for (let i = 0; i < files.length; i++) {
    const file = files[i];
    const reader = new FileReader();

    reader.onload = (e) => {
      previews.push(e.target.result);
      if (previews.length === files.length) {
        setMainImgPreview(previews); // 수정된 부분
      }
    };
    reader.readAsDataURL(file);
  }
};
```

## UI 구현

## [ 재고 관리 ]



**Click!**

No	이미지	이름	판매가	상태	카테고리	재고	최대 주문 수량	할인	등록일	수정
26		듀랑고 썬에어	360,000원	판매 중	tent	12	10	20%	2023. 11. 6.	<button>수정</button>
25		리버너 가스그릴	1360,000원	판매 중	other	63	10	20%	2023. 11. 6.	<button>수정</button>
24		듀랑고 케틀	190,000원	판매 중	other	6	20	12%	2023. 11. 6.	<button>수정</button>
23		듀랑고 구이바다	178,000원	판매 중	other	8	20	10%	2023. 11. 6.	<button>수정</button>
22		인터내셔널 V2	150,000원	판매 중	other	6	20	5%	2023. 11. 6.	<button>수정</button>
21		에어 쿠션매트	54,000원	판매 중	other	10	20	10%	2023. 11. 6.	<button>수정</button>
20		모기향 홀더	29,600원	판매 중	accessory	10	20	10%	2023. 11. 6.	<button>수정</button>

## 사용자를 위한 편리성과 명확성, 효율성을 높이기 위한 필터링

```

val stateProducts = when (state) {
    "true" -> p.select { (p.brand_id eq authProfile.id) and (p.isActive eq true) }
    "false" -> p.select { (p.brand_id eq authProfile.id) and (p.isActive eq false) }
    "할인" -> p.select { (p.brand_id eq authProfile.id) and (p.discountRate greater 0) }
    else -> p.select { p.brand_id eq authProfile.id }
}

val searchProduct = if (keyword.isNullOrEmpty()) {
    stateProducts
} else {
    stateProducts.andWhere { this.SqlExpressionBuilder
        if (keyword.matches(Regex(pattern: "\\d+"))) {
            p.id eq keyword.toLong() ^andWhere
        } else {
            p.productName like "%$keyword%" ^andWhere
        }
    }
}

```

## 사용자 편리를 위한 검색/페이지 기능

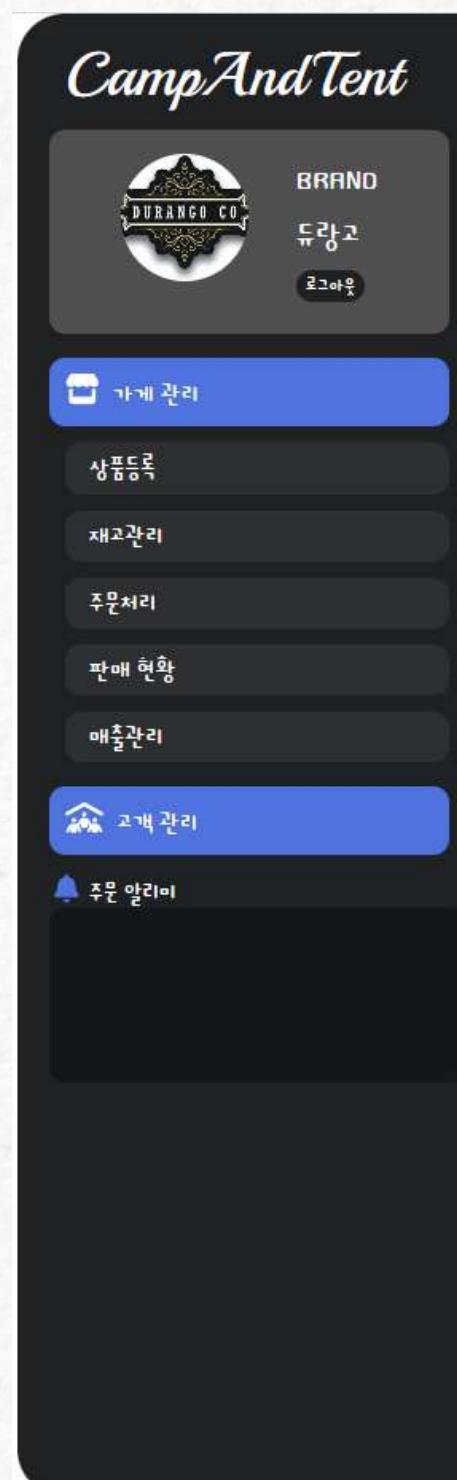
```

val searchProduct = if (keyword.isNullOrEmpty()) {
    stateProducts
} else {
    stateProducts.andWhere { this.SqlExpressionBuilder
        if (keyword.matches(Regex(pattern: "\\d+"))) {
            p.id eq keyword.toLong() ^andWhere
        } else {
            p.productName like "%$keyword%" ^andWhere
        }
    }
}

```

문자열 정규식 사용으로 숫자만 입력시 제품의 ID검색  
String입력시 제품의 이름으로 검색됩니다.

## UI 구현

[재고 관리]  
[테이블]

클릭시 재고가 3개 이하인 제품을 알려줍니다

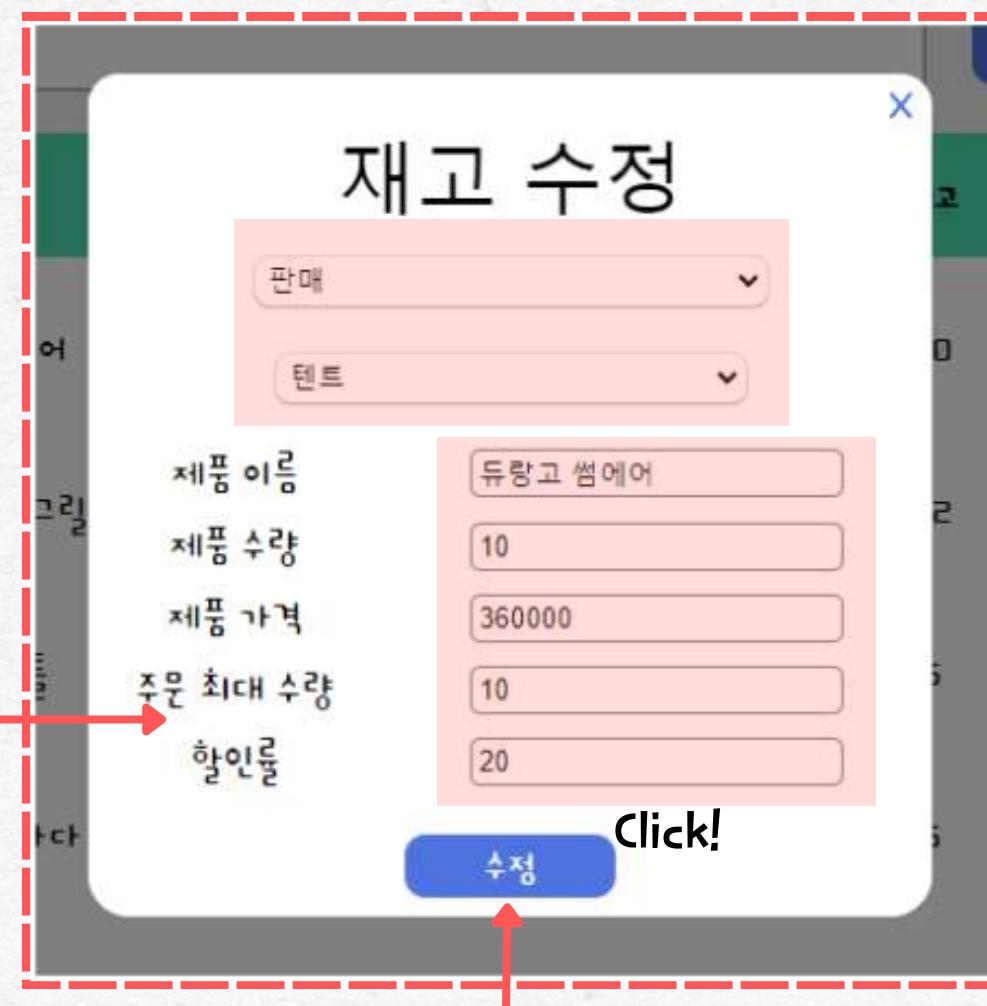
useRef코드와 화면에서 Ref value를 받아오는 부분입니다.

```
const activeRef = useRef<HTMLSelectElement>();
const productNameRef = useRef<HTMLInputElement>();
const quantityRef = useRef<HTMLInputElement>();
const productPriceRef = useRef<HTMLInputElement>();
const maximumPurchaseQuantityRef = useRef<HTMLInputElement>();
const categoryRef = useRef<HTMLSelectElement>();
const discountRateRef = useRef<HTMLInputElement>();
```

useRef코드와 화면에서 Ref value를 받아오는 부분입니다.

```
const activeRef = useRef<HTMLSelectElement>();
const productNameRef = useRef<HTMLInputElement>();
const quantityRef = useRef<HTMLInputElement>();
const productPriceRef = useRef<HTMLInputElement>();
const maximumPurchaseQuantityRef = useRef<HTMLInputElement>();
const categoryRef = useRef<HTMLSelectElement>();
const discountRateRef = useRef<HTMLInputElement>();
```

클릭시 제품 수정모달 창

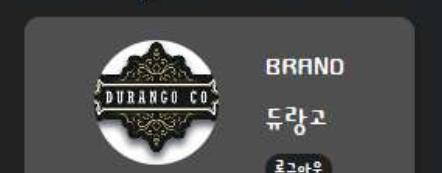


클릭시 수정된 제품의 정보가 queue로 구매 페이지로 전달

## UI 구현

## [주문 관리]

CampAndTent

BRAND  
드랑고  
로그아웃

주문 관리

상품등록

재고관리

주문처리

판매 현황

매출관리

고객 관리

주문 알림

주문 내역/관리

전체 처리 완료 처리 실패

주문Id로 검색해주세요

주문 요청 제품 재고가 있을시 자동으로 주문 수락 해당 제품 재고가 0이라면 자동으로 제품 숨김처리

```
// 성공 메세지 보내기
val successOrderRequest = OrderResultResponse(
    orderId = orderRequest.orderId,
    isPermission = "true"
)
println("successOrderRequest-----$successOrderRequest")
sendResultMessage(successOrderRequest)

else if (findProduct.empty() || currentQuantity == 0 || orderRequest.quantity == 0 || orderRequest.quantity > currentQuantity)
    Val falseOrderRequest = OrderResultResponse (
        orderId = orderRequest.orderId,
        isPermission = "false"
    )
    o.update({ o.orderId eq orderRequest.orderId }) { this.OrderTable it:UpdateStatement
        it[o.orderStatus] = false
    }
    println("falseOrderRequest-----$falseOrderRequest")
    sendResultMessage(falseOrderRequest)
```

## • 주문 정보, 제품정보, 제품 이미지 정보 합치는 응답 부분

```
val orderStateAndInfo = o.select { (o.productId eq productId) }.map { r ->
    OrderStateAndInfo(
        orderId = r[o.orderId],
        orderStatus = r[o.orderStatus],
        quantity = r[o.quantity],
        orderDate = r[o.orderDate].toString()
    )
}

// 검색
// 제품 + 제품 사진 합치기
val productInfoAndFile =
    (Product innerJoin ProductFiles).select { (Product.id eq r[o.productId]) }
        .map { it.ResultRow //로그인한 유저가 등록한 제품만 map
            ProductInfoAndFile(
                it[Product.id],
                it[Product.productName],
                it[ProductFiles.uuidFileName],
                it[ProductFiles.originalFileName],
                it[ProductFiles.contentType]
            )
        }

OrderDetailsResponse(
    orderId = r[o.id],
    quantity = r[o.quantity],
    orderDate = r[o.orderDate].toString(),
    orderState = r[o.orderStatus],
    productInfo = productInfoAndFile
) ^map
```

주문 ID	제품번호	제품 사진	제품 이름	처리 상태	주문 요청시간
100	12		와이드 캠핑반(mapping)	처리 완료	2020. 2. 12.
99	14		클래식 캠핑 나이프	처리 완료	2020. 1. 30.
98	19		다이캐스팅 해머	처리 완료	2020. 12. 18.
97	10		올리브 에고테이블	처리 완료	2020. 11. 9.
96	2		듀랑고 에어미니	처리 완료	2020. 10. 24.
95	8		와이드 그릴 테이블	처리 완료	2020. 9. 11.
94	4		캐빈 라이트	처리 완료	2020. 8. 5.

## UI 구현

## [판매 현황]

```

orders.foreach { order =>
    val productCategory = innerJoinOrderAndProduct
        .select { p.id eq order(o.productId) }
        .map { it[p.category] } // 순회중인 주문의 카테고리를 추출
        .firstOrNull()
    if (productCategory != null) {
        val orderDate = order(o.orderDate) //순회중인 주문의 orderDate
        val month = orderDate.monthValue // 순회중인 주문의 월
        val orderQuantity = order(o.quantity) // 순회중인 주문의 수량
    }
}

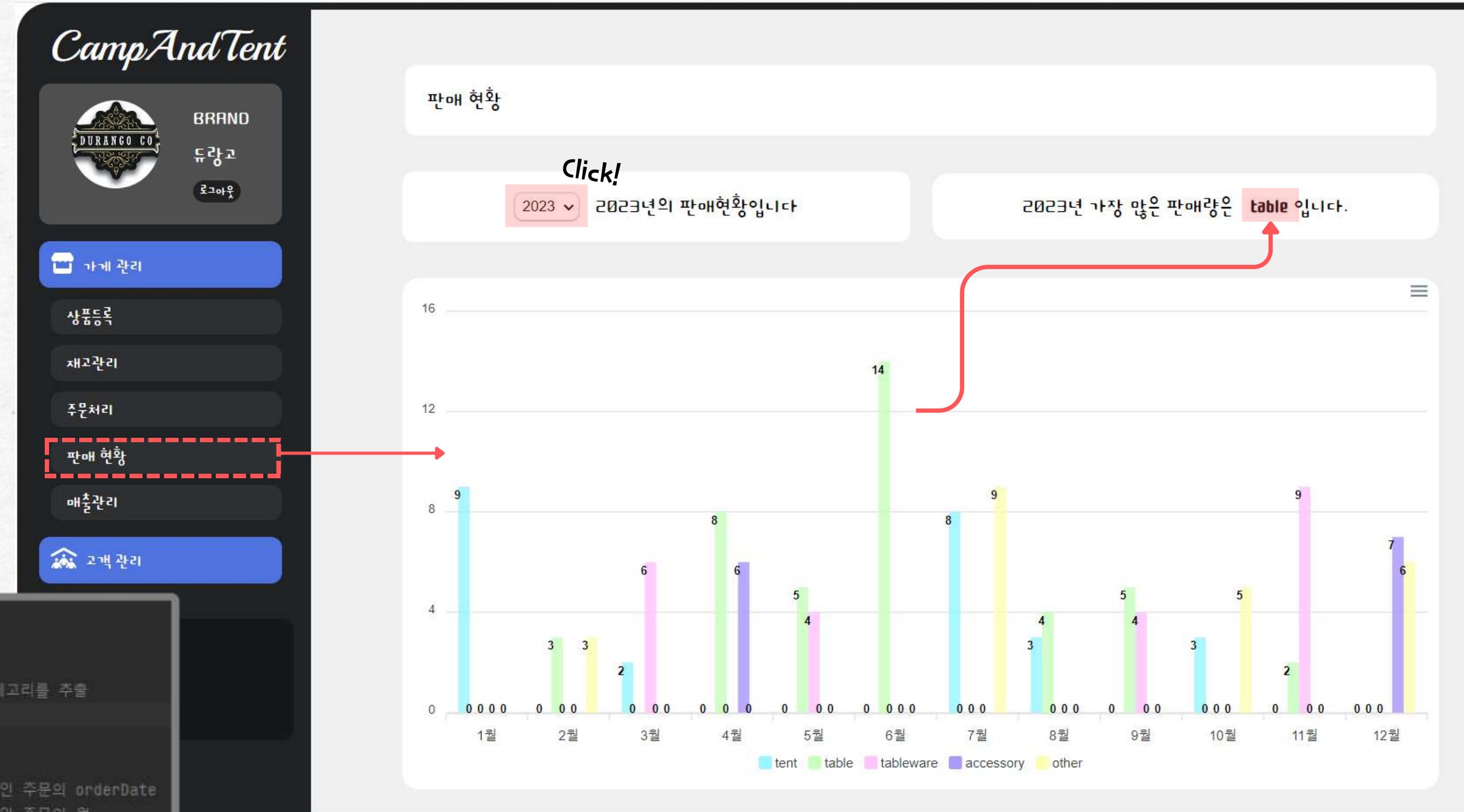
```

```
val categoryTotalMap = mutableMapOf<String, MutableList<Int>>()
```

```

categoryTotalMap //순회중인 카테고리 ex: 텐트면 텐트에 orderQuantity 추가
    .getOrPut(productCategory) { MutableList( size: 12) { 0 } }
// set 사용해서 내가 원하는 인덱스에 값 설정 ex : month 값이 11월이라면 -1 해서 10번째 인덱스에 값 설정
    .set(month - 1, categoryTotalMap[productCategory]!![month - 1] + orderQuantity)
}

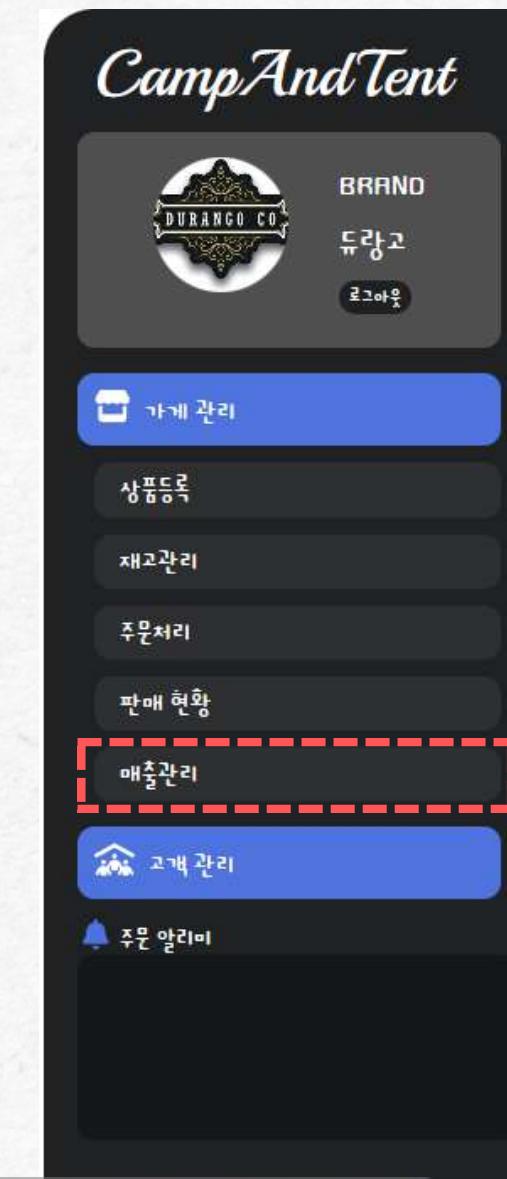
```



- Apex-Chart 라이브러리를 이용하여 데이터를 차트화 시켜 화면에 출력합니다.
- 선택하고자하는 연도별의 카테고리 판매현황을 볼 수 있습니다.
- 사용자 가시성을 위해 연도중 가장 많이 판매한 카테고리를 사용자에게 보여줍니다.
- 이 페이지의 개발목적은 월별 판매량 카테고리별 추이를 보고 판매자가 판매 계획을 수립 할 수 있도록 개발하였습니다.

## UI 구현

## [매출 관리]



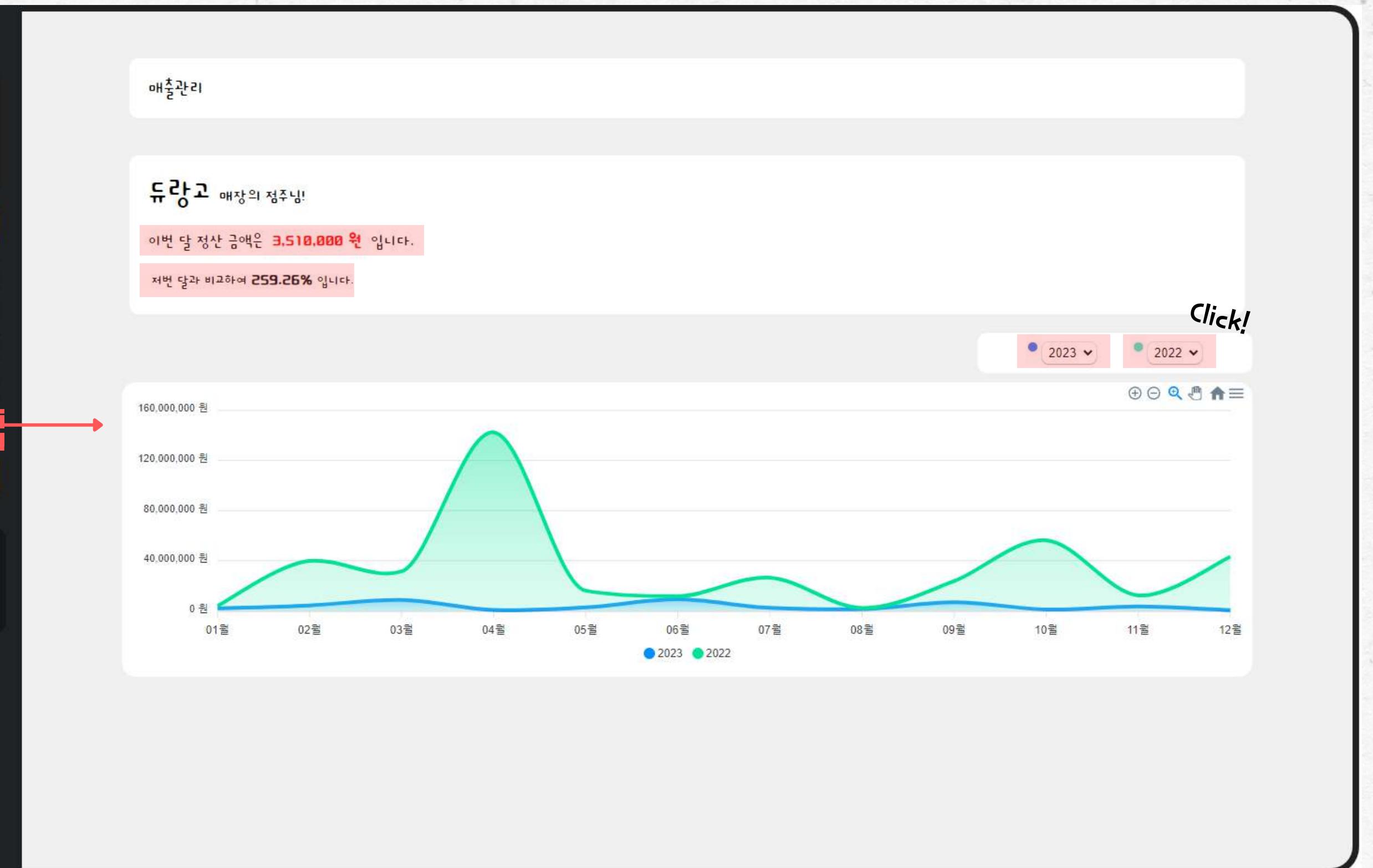
```

class SettlementMoney extends React.Component<{}, State> {
  constructor(props: {}) {
    super(props);

    this.state = {
      thisMonth: null,
      lastMonth: null,
      brandName: null,
    };
  }

  calculateChangePercentage(thisMonth: number, lastMonth: number) {
    if (lastMonth === 0) {
      return "저번 달의 정산 금액이 없습니다.";
    }
    const change = ((thisMonth - lastMonth) / lastMonth) * 100;
    return change.toFixed(2) + "%";
  }
}

```



- Apex-Chart 라이브러리를 이용하여 데이터를 그래프화 시켜 화면에 출력합니다.
- 저번 달과 이번 달의 정산 금액을 계산하여 사용자에게 매출의 상승률과 하락률을 사용자에게 알려줍니다.
- 사용자의 편의를 고려하여, 선택한 연도별 판매량을 비교할 수 있도록 두 개의 차트로 구성하였습니다.

# 프론트엔드

## [webPack]

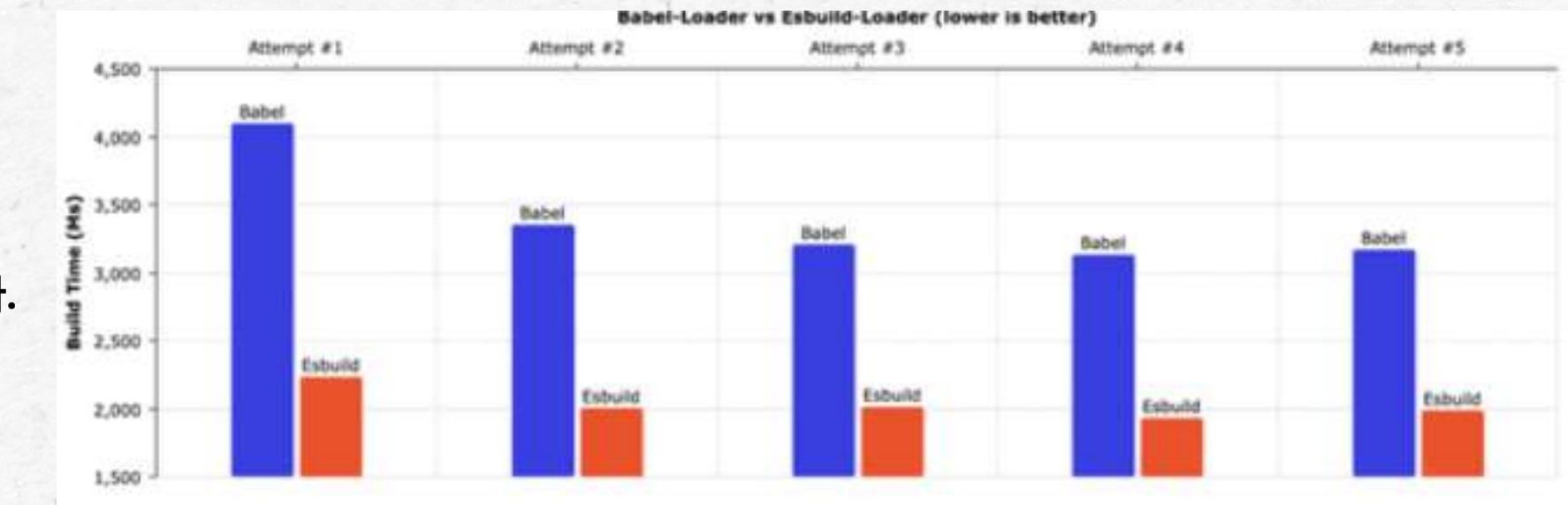


- webpack 모듈번들러를 사용하여 빌드하였습니다.
- 기존에 많이 사용하던 Babel-loader가 아닌 Ebuild-loader를 사용했습니다.

### webpack.config.js

```
module: {
  rules: [
    {
      test: /\.tsx$/,
      loader: "esbuild-loader",
      options: {
        target: "es2020",
      },
    },
    {
      test: /\.(gif|jpg|png|webp|svg|mp4)$/,
      type: "asset/resource",
    },
    {
      test: /\.css$/,
      use: ["style-loader", "css-loader"],
    },
  ],
},
```

- Babel-loader와 Ebuild-loader의 빌드 속도 차이



### 실제 테스트

```
webpack 5.88.2 compiled successfully in 8712 ms
Babel-loader
webpack 5.88.2 compiled successfully in 4932 ms
Ebuild-loader
```

- webpack의 사용이유  
모듈 시스템의 이해도 상승과 성능 튜닝 옵션 및 플러그인 생태계가 잘 갖추어져있어 성능 상승 및 학습 성과를 올릴 수 있을 거라 판단하여 사용했습니다.

- Ebuild-loader의 사용이유  
BabI-loader와는 다르게 Go 언어로 작성된 Ebuild-loader는 컴파일 단계 이전 미리 기계어로 변환되어 빠른 속도와 높은 효율성을 제공하기에 채택하여 사용했습니다.

# 메인페이지

## [서비스 구현부]



- 중첩라우팅 구현 부분

```
const App = () => {
  return (
    <ProfileProvider>
      <BrowserRouter>
        <ResetStyle />
        <Routes>
          <Route path="/" element={<LoginComponent />} />
          <Route path="/home" element={<Layout />}>
            {UserManagementRoutes}
            {StoreManagementRoutes}
          </Route>
        </Routes>
      </BrowserRouter>
    </ProfileProvider>
  );
};

export default App;
```

```
return (
  <LayoutContainer>
    <nav>...
    </nav>
    <main>
      <Outlet />
    </main>
  </LayoutContainer>
);

export default Layout;
```



<main>  
Outlet  
<main>  
영역

- Layout을 가장 상단에 위치시켜 공통 컴포넌트로 사용하였습니다.

- 중첩라우팅 기술을 사용하여 계층구조를 통해 모듈화와 가독성 유지보수성을 향상 시켰습니다.
- Layout컴포넌트 내부에서 Outlet을 사용하여 하위 라우트들을 렌더링 하였습니다.
- Outlet을 사용하여 계층적인 구조를 효과적으로 표현하였습니다.
- 유지보수성 향상과 클린한 코드를 위해 담당 역할의 루트를 분할하여 프로젝트를 진행하였습니다

# 상품 등록



# EXPOSED

DSL 사용



**BRAND**  
듀랑고  
로그아웃

-  [가게 관리](#)
-  [상품 등록](#)
-  [재고관리](#)
-  [주문 처리](#)
-  [판매 현황](#)
-  [매출 관리](#)
-  [고객 관리](#)
-  [주문 알리미](#)

## 상품 등록

브랜드

대표 이미지 파일 선택 16786873327m2.jpg

**상품명** 듀랑고 에어텐트

**판매가** 492000

**상품 상태:** 판매

**주문 수량** 주문 최대 수량

**할인** 0

**상품 대분류** 옵션을 선택해주세요 ▾

**상품 설명**

상품 상세설명을 적어 주세요

**상품 이미지** 파일 선택 파일 2개




**상품 등록**

```
spring.datasource.url=jdbc:mysql://localhost:3306/store?rewriteBatchedStatements=true
```

```
pf.batchInsert(combinedList) { this: BatchInsertStatement, it: Map<String, String?>
    this[pf.productId] = insertProduct[p.id]
    this[pf.originalFileName] = it["originalFileName"] as String
    this[pf.randomUUIDFileName] = it["randomFileName"] as String
    this[pf.contentType] = it["contentType"] as String
```

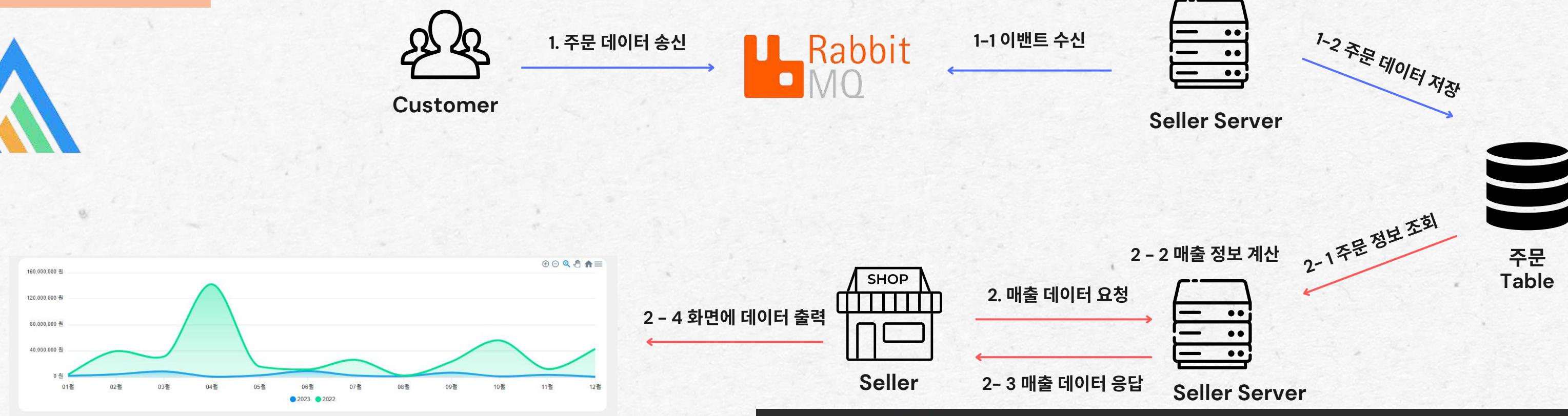
- Kotlin Exposed의 DSL을 사용하여 코드 가독성과 표현력 및 안정성을 향상 시켰습니다.
  - Type-Safe SQL로 구축한 서버를 통해 안전하고 효율적인 웹 어플리케이션을 제공합니다
  - `rewriteBatchedStatements=true` 설정을 통해 데이터베이스와의 효율적인 통신을 실현하며, 이로써 데이터베이스 성능을 획기적으로 향상시켰습니다

# 서비스 구현부

## [Apex-Charts]



- Apex-Charts라이브러리를 사용하여 차트로 출력하기까지의 흐름도입니다.



```

findUserProduct.forEach { productId ->
    val orderProductQuantity = productJoinOrder.select { o.productId eq productId }
        .andWhere { o.orderDate.year() eq year.toInt() }
        .map { it } //주문 정보 뽑기

    orderProductQuantity.forEach {order ->
        val productPrice = productJoinOrder.select { p.id eq order[o.productId] } //주문,제품id일치하는 애들만 select
            .map { it[p.productPrice].toInt() }
            .firstOrNull()

        if (productPrice != null) {
            val orderDate = order[o.orderDate]
            val year = orderDate.year
            val month = orderDate.monthValue
            val orderQuantity = order[o.quantity]

            monthSales.getOrPut(year) { MutableList( size: 12 ) { 0 } }[month - 1] += (productPrice * orderQuantity)
        }
    }
}

```

- 매출 계산의 로직부분입니다.

프로젝트 구현

# 서비스 통합 구현

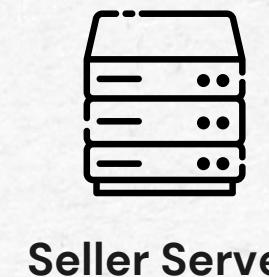
## [상품 등록]

판매 제품 메시지 큐

Seller → Customer



2. 제품 데이터 저장



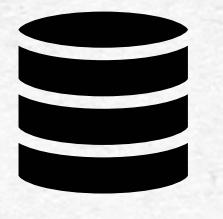
3. 제품 데이터 송신



4. 제품 데이터 수신



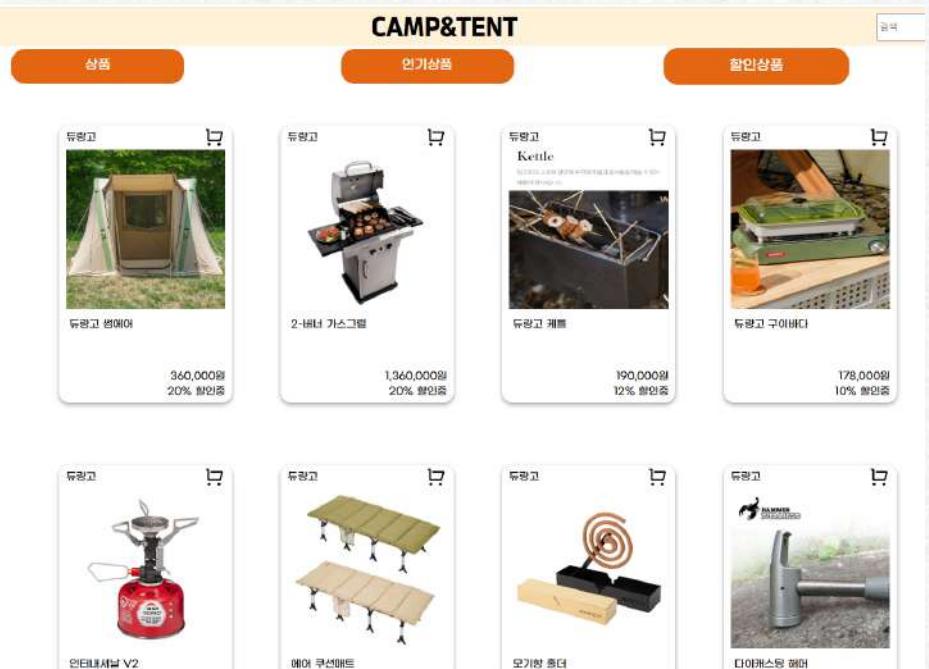
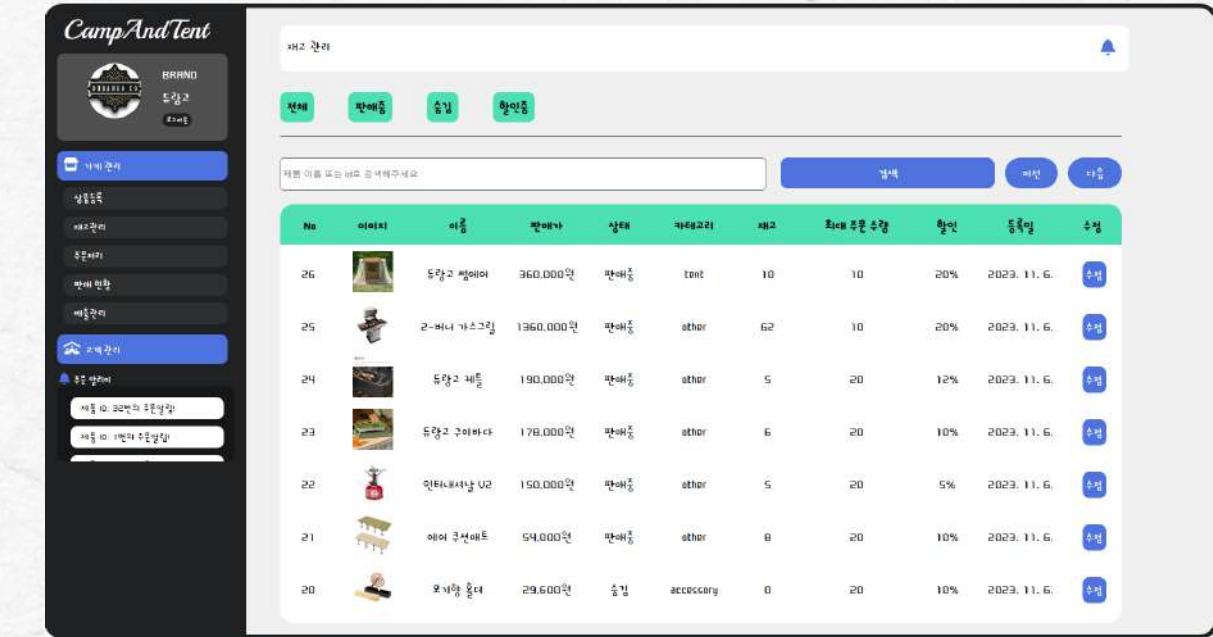
5. 제품 데이터 저장



2) 제품 데이터 조회



1. 제품 등록



Seller에서 등록한 제품을 Customer에서도 출력합니다.



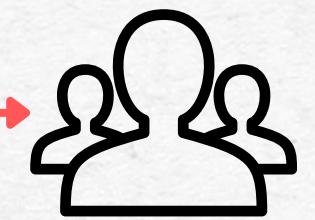
Customer

# 서비스 통합 구현부

## [상품 등록]



Overview					Messages			Message rates		
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	company-register	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	inquiry-response	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	product-inquiry	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	product-payment	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	product-payment-result	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	product-register	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	review-request	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	review-response	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s



```
val productMessageRequest = ProductMessageRequest(
    id = insertProduct[p.id],
    productBrand = productBrand,
    productName = productName,
    productPrice = productPrice.toString(),
    isActive = isActive,
    category = category,
    maximumPurchaseQuantity = maximumPurchaseQuantity,
    discountRate = discountRate,
    productDescription = productDescription,
    mainImageUuidName = mainImgUuid,
    imageUuidName = uuidList
```

```
productService.createProductMessage(productMessageRequest)
```

- 제품 등록하는 RabbitTemplate의 구현부입니다.

```
fun createProductMessage(productMessageRequest: ProductMessageRequest) {
    sendMessage(productMessageRequest)
}

fun sendMessage(productMessageRequest: ProductMessageRequest) {
    rabbitTemplate.convertAndSend("product-register", mapper.writeValueAsString(productMessageRequest))
}
```

- 표준 메시지 큐 프로토콜인 AMQP를 사용하는 메시지 큐 브로커 RabbitMQ를 활용하여 Seller,Customer로 각각 분리되어 있는 서비스 간에 데이터를 주고 받을 수 있도록 구현했습니다.
- 판매자는 제품 정보를 큐에 전송하면, 이를 수신하는 고객 측에서는 해당 정보를 데이터베이스에 저장하도록 구현했습니다.
- 이런 방식을 사용하여 서비스가 분리되어 있지만 같은 데이터를 보유하게 됩니다.

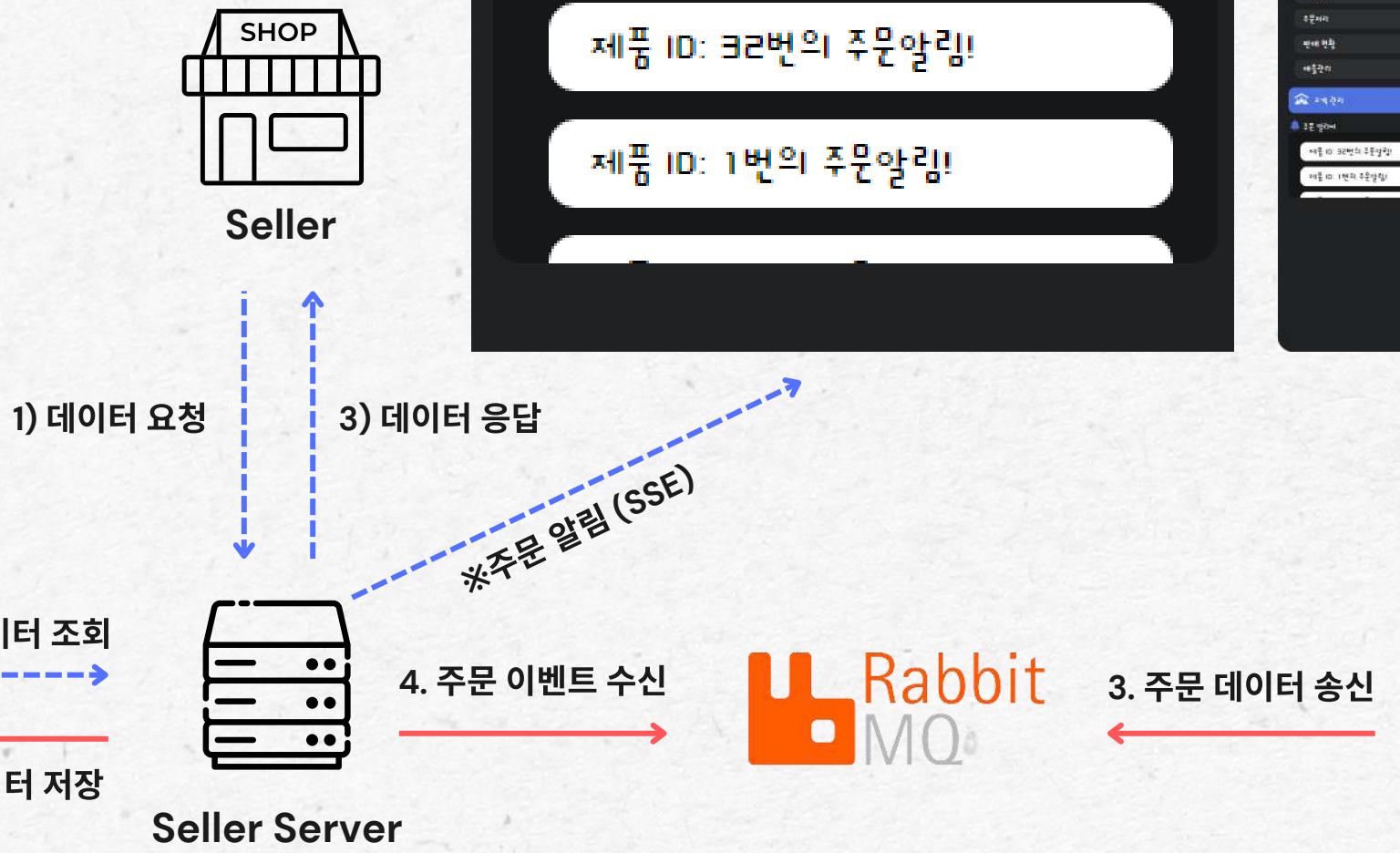
프로젝트 구현

## 서비스 통합 구현

### [주문 처리]

#### 주문 메세지 큐

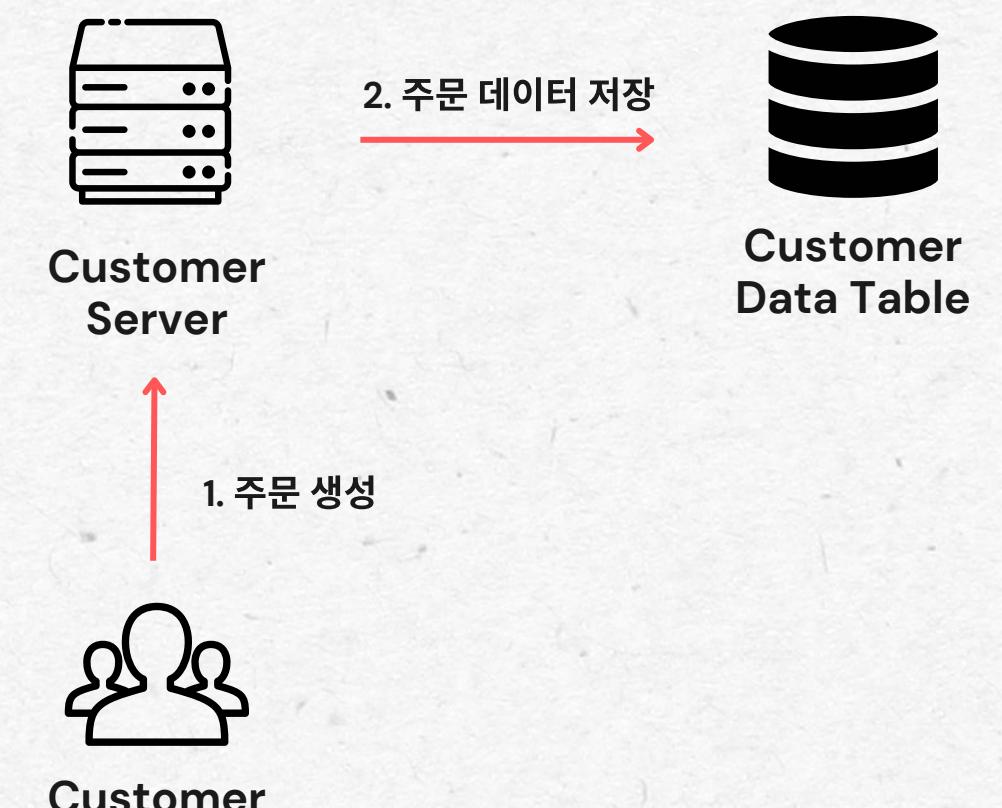
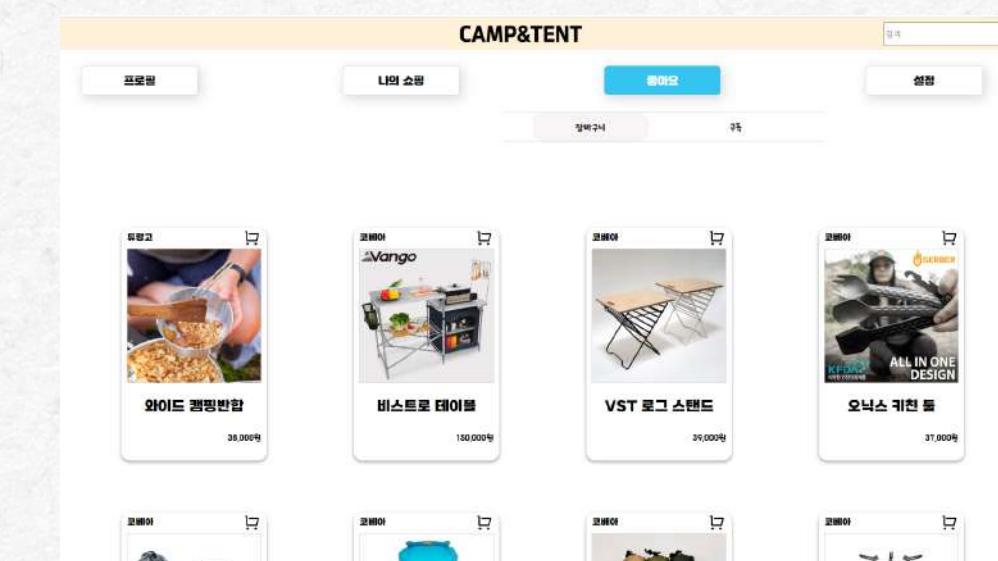
Customer → Seller



## RabbitMQ를 활용해 서비스 간 데이터를 주고받도록 구현 했습니다.

Customer가 주문한 주문 데이터를 Seller서버에서도 출력합니다.

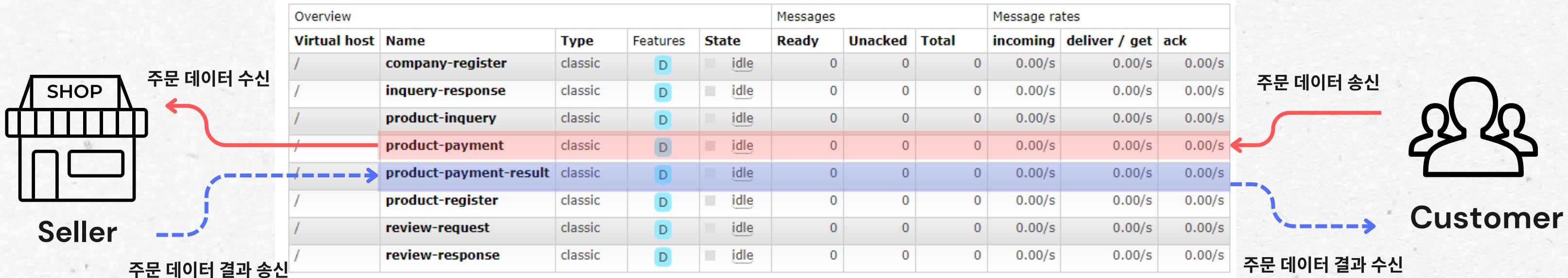
주문 ID	제품명	제품 사진	제품 가격	처리 상태		주문 요청시간
				판매처	처리 상태	
100	21		이어 캠핑테드	처리 완료	2023. 11. 17.	
99	18		마트리아 인서트	처리 완료	2023. 11. 17.	
98	17		도랑 2인용	처리 완료	2023. 11. 17.	
97	16		도랑 2인용 텐트	처리 완료	2023. 11. 17.	
96	15		도랑 2인용 낚시竿	처리 완료	2023. 11. 17.	
95	14		3인용 캠핑 나이트	처리 완료	2023. 11. 17.	
94	12		와이드 캠핑테드	처리 실패	2023. 11. 17.	



# 서비스 통합 구현부

## [주문 처리]

- Customer가 주문을 보내면 Seller는 연결된 product-payment Que저장소에 Listening을 하여 주문에 대한 처리를 할 수 있게 구현 하였습니다.
- 주문 처리에 대한 결과를 product-payment-result Que저장소로 응답을 전송합니다.



```

@Auth
@RabbitListener(queues = ["product-payment"])
product-payment에 있는 queue receiveOrder(message : String) 매개변수로 전달
fun receiveOrder(message : String){
    JSON 문자열(message)을 OrderRequest 클래스의 인스턴스로 변환
    val orderRequest : OrderRequest = mapper.readValue(message)

    orderId = orderRequest.orderId,
    isPermission = "true"
)
println("successOrderRequest-----$successOrderRequest")
sendResultMessage(successOrderRequest)
else if (findProduct.empty() || currentQuantity == 0 || orderRequest.quantity == 0 || orderRequest.quantity > currentQuantity)
    val falseOrderRequest = OrderResultResponse (
        orderId = orderRequest.orderId,
        isPermission = "false"
)
o.update{ o.orderId eq orderRequest.orderId } { this: OrderTable it: UpdateStatement
    it[o.orderStatus] = false
}
println("falseOrderRequest-----$falseOrderRequest")
sendResultMessage(falseOrderRequest)
    
```

- 주문처리하는 RabbitListener의 구현부입니다.

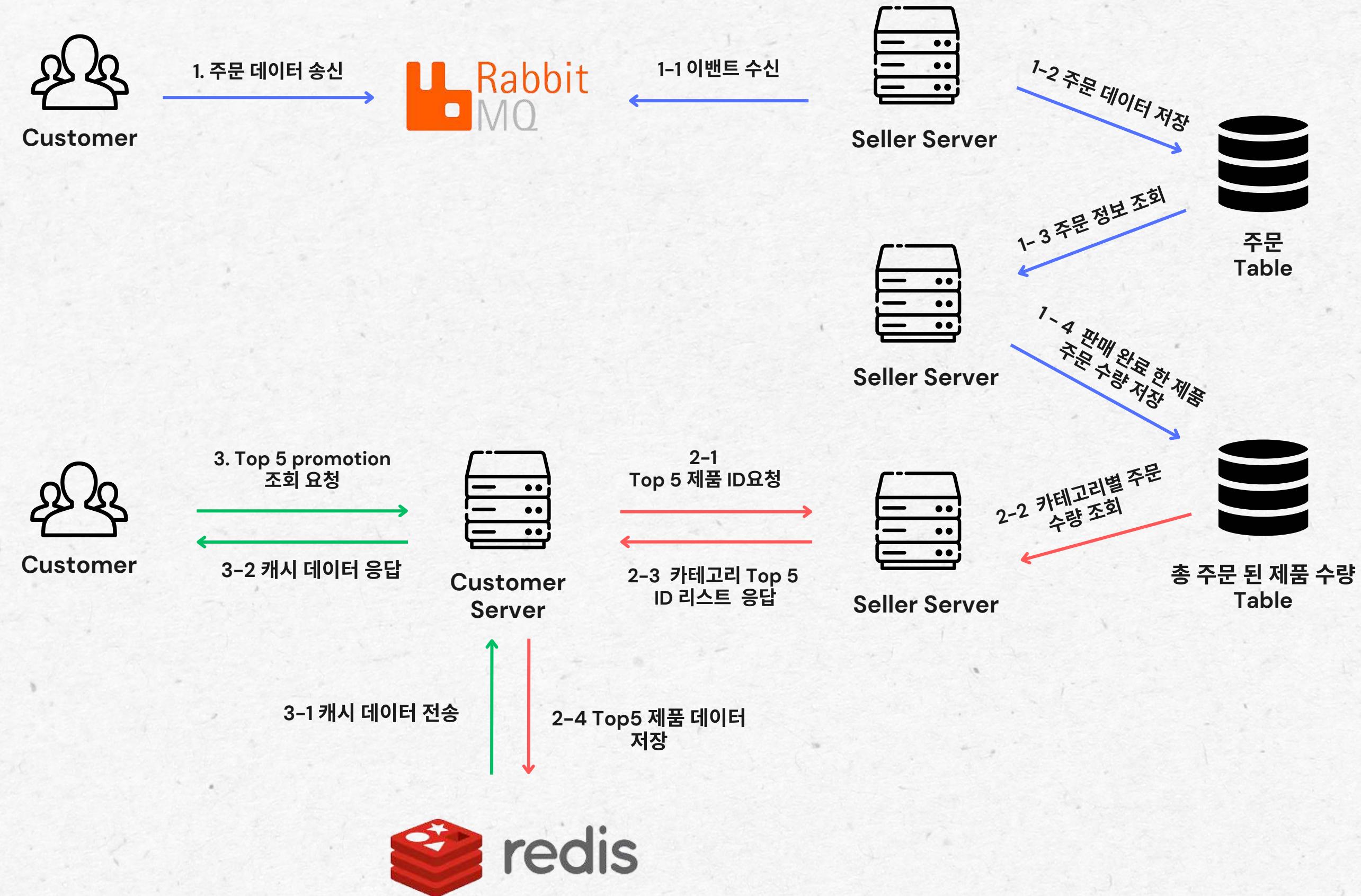
프로젝트 구현

## 서비스 통합 구현부

Top 5  
Promotion

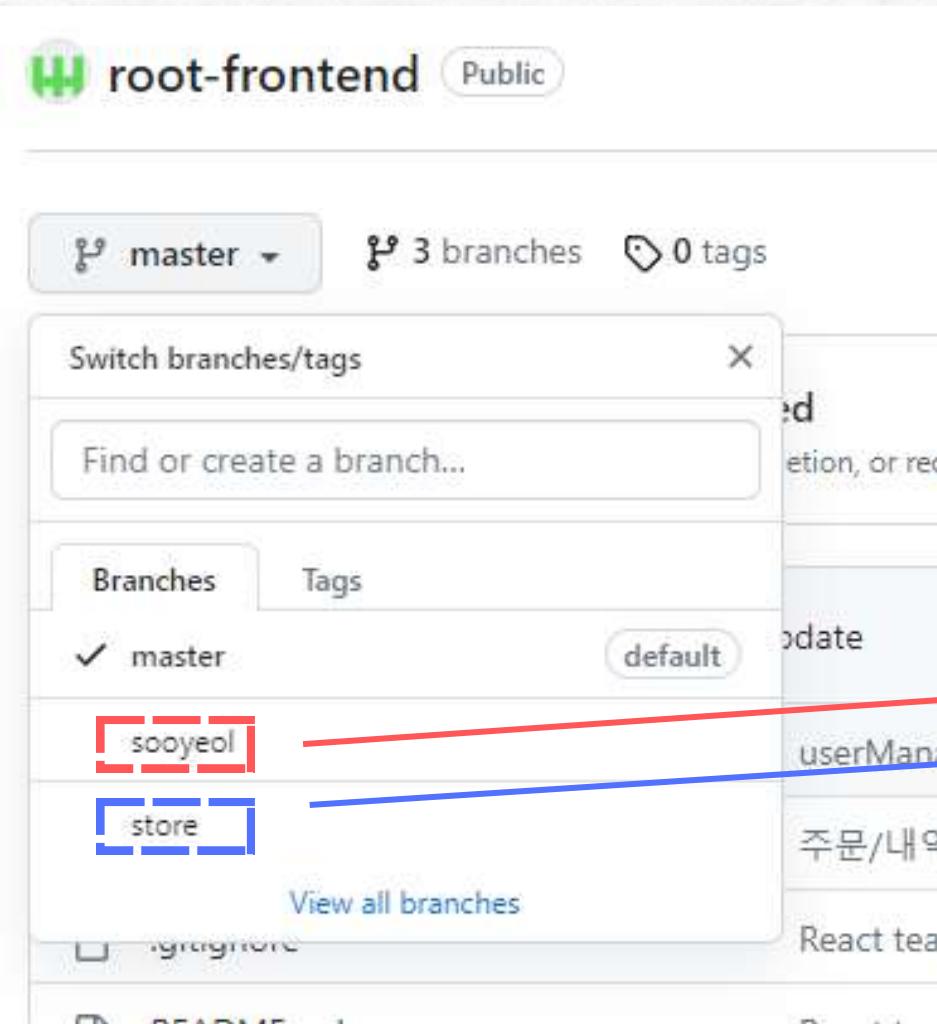


Redis DB를 사용해 자주 조회되는 데이터를 캐싱하여 빠른 속도로 조회가 되도록 구현했습니다.



# 프로젝트 협업

[git]



- 효율적이고 체계적인 작업을 위해 GIT branch를 활용한 개발 방법을 적용했습니다.
- master 브랜치는 항상 안정된 코드를 유지하여 프로젝트의 기능이나 버그수정이 완료된 경우에만 통합하였습니다.
- 각 기능은 독립된 브랜치에서 개발되어 프로젝트 전체에 영향을 주지 않도록 안정적으로 진행되었습니다.
- 각 브랜치는 특정 기능이나 작업 단위로 구성되어 유지보수를 용이하게 만들었습니다.

## • store브랜치로 switch

```
Switched to a new branch 'store'  
M package-lock.json  
M package.json  
D src/modules/StoreManagement/main/index.tsx  
D src/modules/StoreManagement/main/style.ts  
D src/modules/StoreManagement/registration/index.tsx  
D src/modules/StoreManagement/registration/style.ts  
M src/modules/UserManagement/LoginManagement/LoginComponent.tsx  
M tsconfig.json  
branch 'store' set up to track 'origin/store'.
```

## • 로컬에 최신 변경 사항 가져오기위한 master브랜치 fetch

```
PS C:\root-frontend> git fetch origin master  
From https://github.com/MyungSeoungJung/root-frontend  
 * branch master      -> FETCH_HEAD
```

## • master 브랜치 merge로 로컬에 변경 내역 업데이트

```
PS C:\root-frontend> git merge origin/master  
Updating b0ca0f5..4dac8f6  
Fast-forward  
src/modules/Layout.tsx | 13 +--  
.../InquiryManagement/InquiriesContainer.tsx | 130 ++++++-----  
.../InquiryManagement/answeredInquiries/index.tsx | 50 +++++++  
.../UserManagement/InquiryManagement/types.ts | 15 +++  
.../unansweredInquiries/index.tsx | 78 +++++++  
.../ReviewManagement/answeredReviews/index.tsx | 84 ++++++---  
.../UserManagement/ReviewManagement/api/api.ts | 21 ----  
.../ReviewManagement/reviewContainer.tsx | 77 ++++++---  
.../ReviewManagement/reviewItem/index.tsx | 26 +---  
.../UserManagement/ReviewManagement/types.ts | 1 -  
.../ReviewManagement/unansweredReviews/index.tsx | 108 ++++++---  
.../ReviewManagement/useFetchReviews.ts | 1 +  
src/modules/UserManagement/routes.ts | 8 +-  
13 files changed, 501 insertions(+), 107 deletions(-)  
create mode 100644 src/modules/UserManagement/InquiryManagement/InquiriesContainer.tsx  
create mode 100644 src/modules/UserManagement/InquiryManagement/answeredInquiries/index.tsx  
create mode 100644 src/modules/UserManagement/InquiryManagement/types.ts  
create mode 100644 src/modules/UserManagement/InquiryManagement/unansweredInquiries/index.tsx  
delete mode 100644 src/modules/UserManagement/ReviewManagement/api/api.ts
```

# 프로젝트 배포

## [서버]

작업 초반에는 직접 docker images build 및 rmi images, push,pull을 하여 전체적인 흐름을 이해하고 후반부에 Jenkins 자동 배포를 진행하였습니다.

### 1 jar 파일 build

```
PS C:\Users\tjoeun\storeManagement> ./gradlew jar
Starting a Gradle Daemon (subsequent builds will be faster)
<==> 20% EXECUTING [15s]
> :compileKotlin
> IDLE
[]

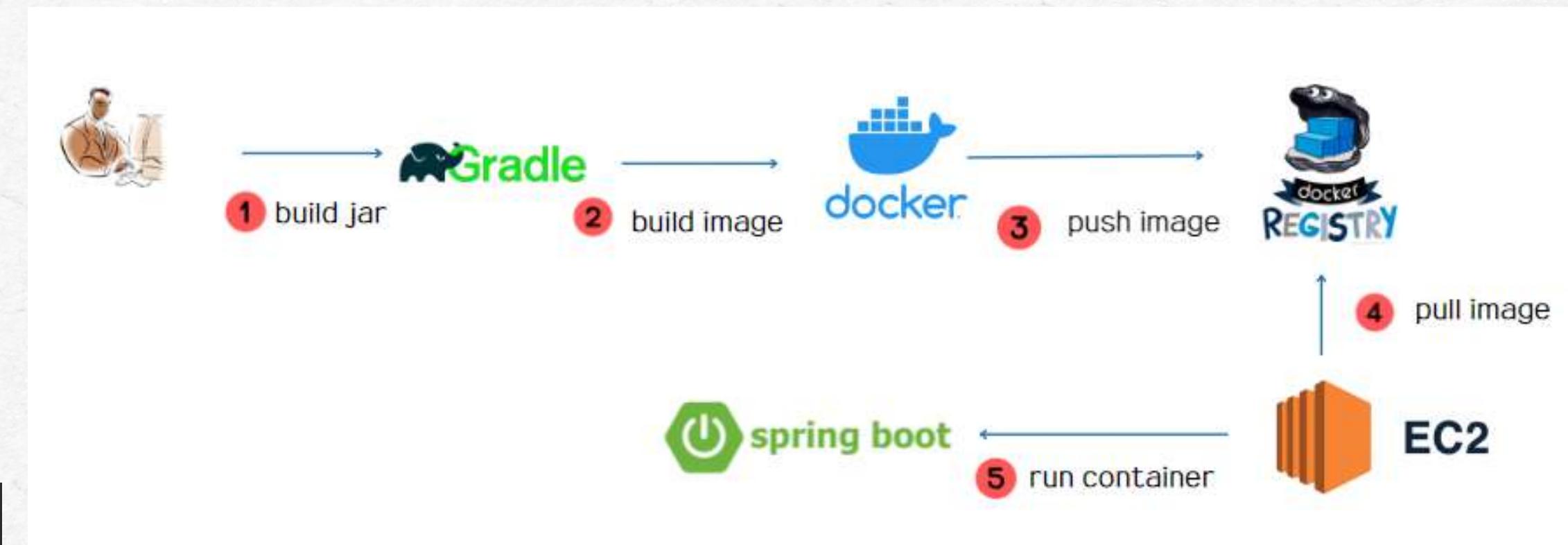
BUILD SUCCESSFUL in 41s
```

### 2 Docker repository에 push

```
PS C:\Users\tjoeun\storeManagement> docker push myungsj/storemanagement
Using default tag: latest
The push refers to repository [docker.io/myungsj/storemanagement]
80eaabb02289: Layer already exists
c28fdec40c4a: Layer already exists
8ddec224834c: Layer already exists
e9db4731c771: Layer already exists
3de9bbd55c1d: Layer already exists
8ceb9643fb36: Layer already exists
latest: digest: sha256:d94cda0374d261ec815f488bb979e5b08fa1c46b632dd6bb3e8b1ec9
[]
```

### 3 EC2 서버 Docker기존 이미지 삭제

```
ubuntu@ip-172-31-47-7:~/app/store$ sudo docker rmi -f f8205d85508f
Untagged: myungsj/storemanagement:latest
Untagged: myungsj/storemanagement@sha256:d94cda0374d261ec815f488bb979e5b632dd6bb3e8b1ec90f8b59
Deleted: sha256:f8205d85508ff53cb11f03a14ae43808ae2c65cd4cfaf7ffe47f7bed
Deleted: sha256:5d550f5e7d7dc5dc33cdd40a3f8068a30e6ed2ee880510d7dbc5104
Deleted: sha256:294d117a238d324a417a179d207af30a564fb27bc3eac03a4c2a5c6
Deleted: sha256:c2f7aced94ac810d00cdde4b94ea693e0ad339ef279b624e9434c2c
Deleted: sha256:f5f1090da1060835f40ae35740341a4374ef9877b4c1d512c49ebb8
Deleted: sha256:39a2259e820b896dff0aea9818be884077edc9895b361c8e670300c
Deleted: sha256:8ceb9643fb36a8ac65882c07e7b2ffff9fd117673d6784221a83d3ad
```



### 4 EC2 서버에서 새로운 컨테이너 pull

```
ubuntu@ip-172-31-47-7:~/app/store$ sudo docker pull myungsj/storemanagement
Using default tag: latest
latest: Pulling from myungsj/storemanagement
cbe3537751ce: Pull complete
6cd63fc495d1: Pull complete
9c42674dea4f: Pull complete
a992519bbaee: Pull complete
506583e9517b: Pull complete
44b4865b1c7a: Pull complete
Digest: sha256:d94cda0374d261ec815f488bb979e5b08fa1c46b632dd6bb3e8b1ec90f8b59
Status: Downloaded newer image for myungsj/storemanagement:latest
docker.io/myungsj/storemanagement:latest
```

### 5 EC2 서버에서 새로운 컨테이너 -d(daemon)으로 실행

```
ubuntu@ip-172-31-47-7:~/app/store$ sudo docker-compose up -d
Creating network "store_my-network" with driver "bridge"
Creating store_redis_1 ... done
Creating store_app_1 ... done
```

# 프로젝트 배포

[서버]

[Jenkins CI/CD]

Console Output

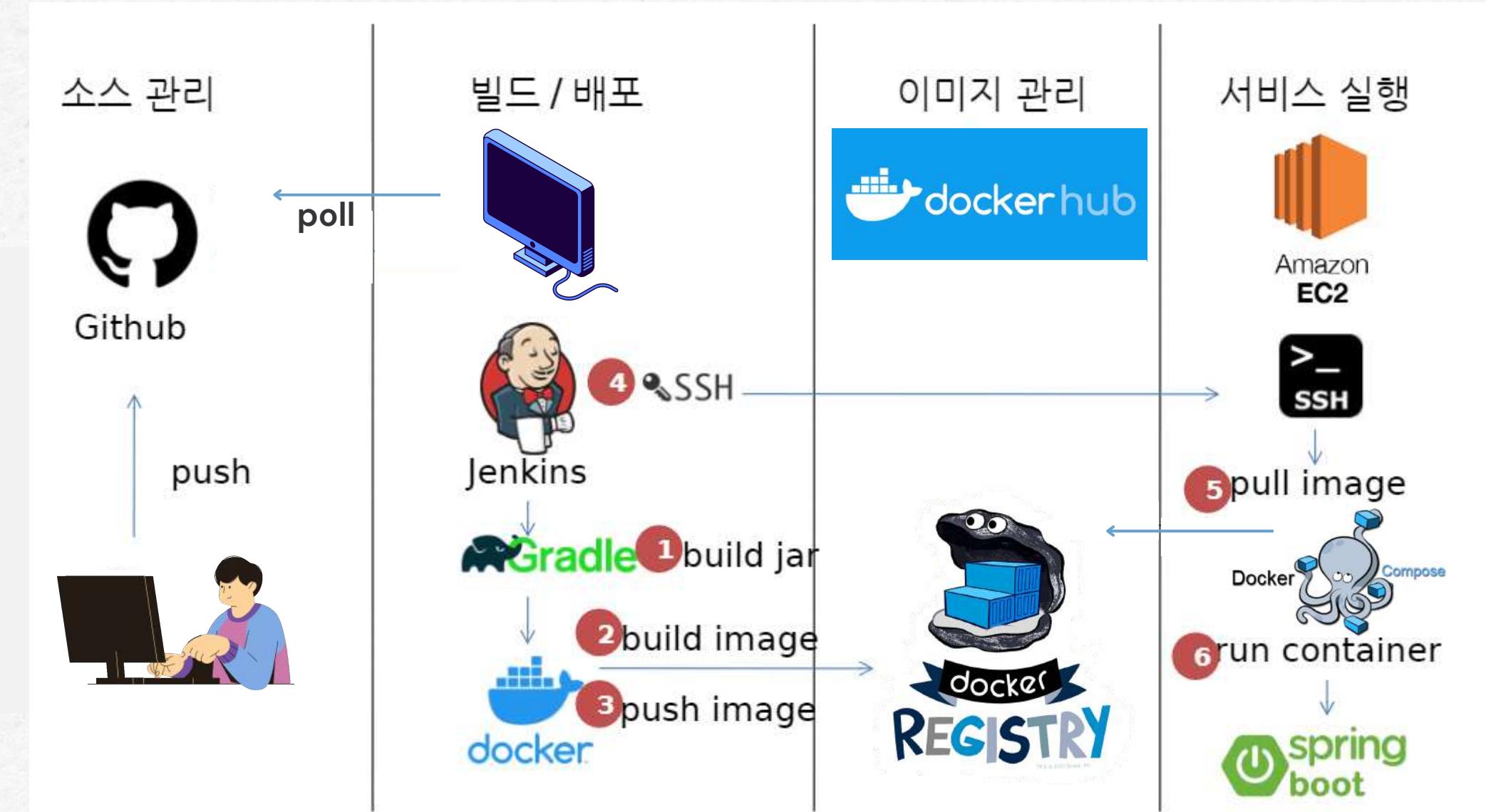
콘솔 출력

```
18:22:03 Status: Downloaded newer image for myngsj/storemanagement:latest
18:22:03 docker.io/myngsj/storemanagement:latest
18:22:04 Creating network "store_my-network" with driver "bridge"
18:22:04 Creating store_redis_1 ...
18:22:05 Creating store_redis_1 ... done
18:22:05 Creating store_app_1 ...
18:22:06 Creating store_app_1 ... done
18:22:06 SSH: EXEC: completed after 25,728 ms
18:22:06 SSH: Disconnecting configuration [myserver] ...
18:22:06 SSH: Transferred 1 file(s)
18:22:06 Finished: SUCCESS
```

Jenkins의 빌드 기록

	Dec 4	Dec 5	Dec 6
aws_myapp_msj	#3		#9
aws_myapp_msj	#2		#8
aws_myapp_msj	#1		#7
			#6
			#5
			#4

작업 초반에는 직접 docker images build 및 rmi images, push,pull을 하여 전체적인 흐름을 이해하고 후반부에 Jenkins 자동 배포를 진행하였습니다.



# 프로젝트 배포

[프론트]  
[Jenkins CI/CD]

## Jenkins의 빌드 기록

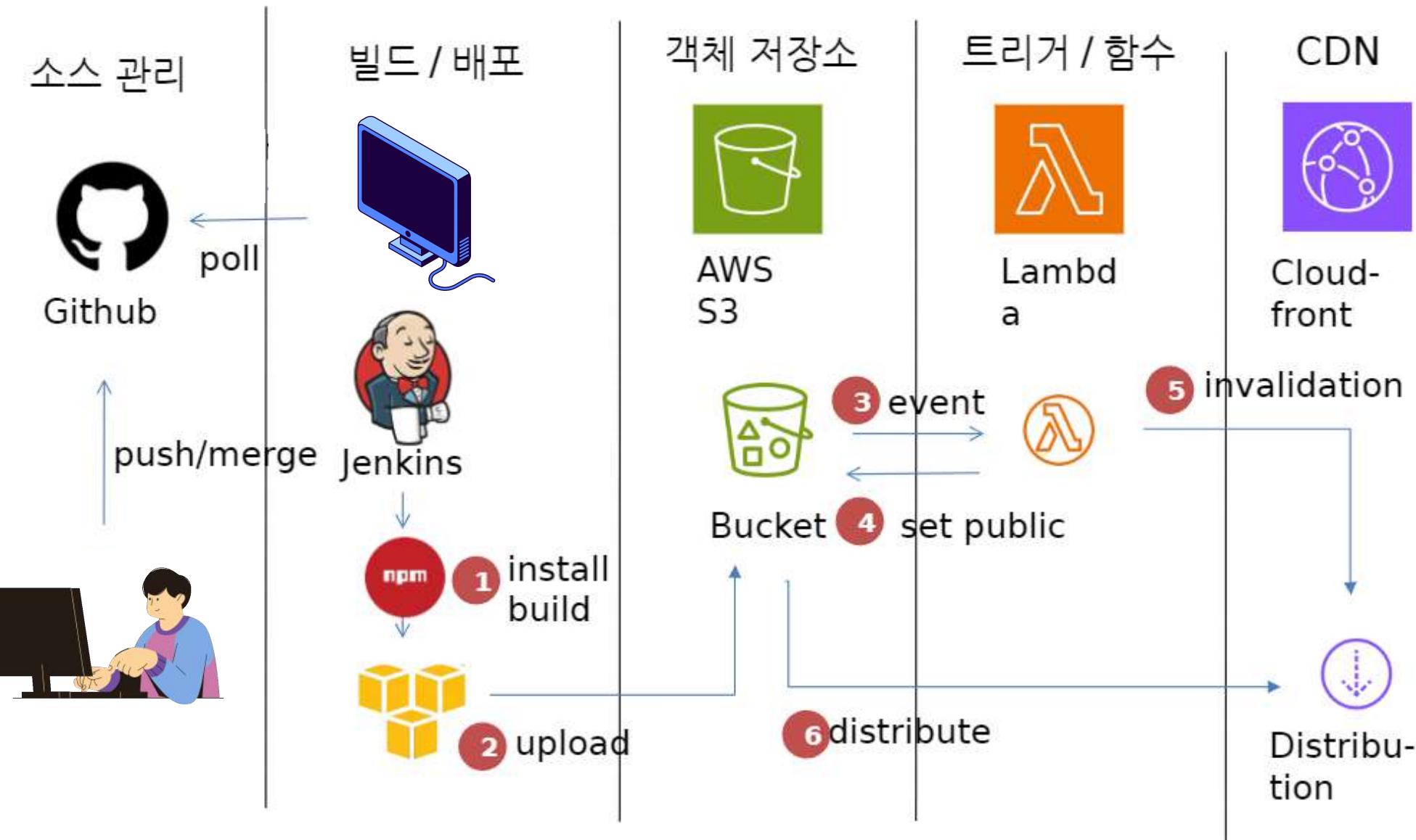
Dec 6	Dec 7
teamProject-react-front #1 teamProject-react-front #6 teamProject-react-front #5 teamProject-react-front #4 teamProject-react-front #3 teamProject-react-front #2	teamProject-react-front #7

Console Output

## 콘솔 출력

```
18:40:36 Publish artifacts to S3 Bucket Using S3 profile: msj-react-front
18:40:36 Publish artifacts to S3 Bucket bucket=msj-react-app, file=0e0a295f079
18:40:36 Publish artifacts to S3 Bucket bucket=msj-react-app, file=71988dd7eaa
18:40:36 Publish artifacts to S3 Bucket bucket=msj-react-app, file=8a34d27d168
18:40:36 Publish artifacts to S3 Bucket bucket=msj-react-app, file=e1d48bd56aa
18:40:36 Publish artifacts to S3 Bucket bucket=msj-react-app, file=main-8bbeaa
18:40:36 Publish artifacts to S3 Bucket bucket=msj-react-app, file=main-8bbeaa
false
18:40:36 Publish artifacts to S3 Bucket bucket=msj-react-app, file=report.html
18:40:37 Publish artifacts to S3 Bucket bucket=msj-react-app, file=index.html
18:40:37 Finished: SUCCESS
```

작업 초반에는 직접 docker images build 및 rmi images, push,pull을 하여 전체적인 흐름을 이해하고 후반부에 Jenkins 자동 배포를 진행하였습니다.



빌드된 dist 파일 버켓에 업로드

```
$ npm run build
> build
> webpack --mode production

Webpack Bundle Analyzer saved report to C:\Users\tjoeun\root-frontend\dist\report.html
assets by status 11.9 MiB [cached] 4 assets
assets by path . 1.09 MiB
  asset js/main-fc91dba52a9277b90997.js 1.09 MiB [emitted] [immutable] [minimized] [big]
  late asset
  asset index.html 423 bytes [emitted] [compared for emit]
```

이름	유형
asset/	폴더
index.html	html
js/	폴더
report.html	html

# Q&A

# THANK YOU VERY MUCH!

Phone : 010-5478-6535

E-mail : audtmdwjd@naver.com

Git frontend : <https://github.com/MyungSeoungJung/root-frontend>

Git backend : <https://github.com/MyungSeoungJung/storeManagement>

배포 URL : <https://d1mqxkgc07he9i.cloudfront.net/>