CAMPANDTENT

풀스택

팀 프로젝트

리액트 & 코틀린 기반 웹

명승정 포트폴리오

프로젝트 개요

프로젝트 개요

프로젝트 주제 및 동기

주제

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

동기

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

사용기술

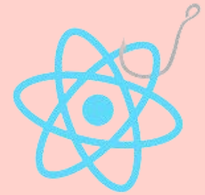
통신

데이터 베이스

백엔드 프레임워크 /라이브러리

사용언어

프론트엔드 프레임워크 /라이브러리



프로젝트 개요

사례 분석

- 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

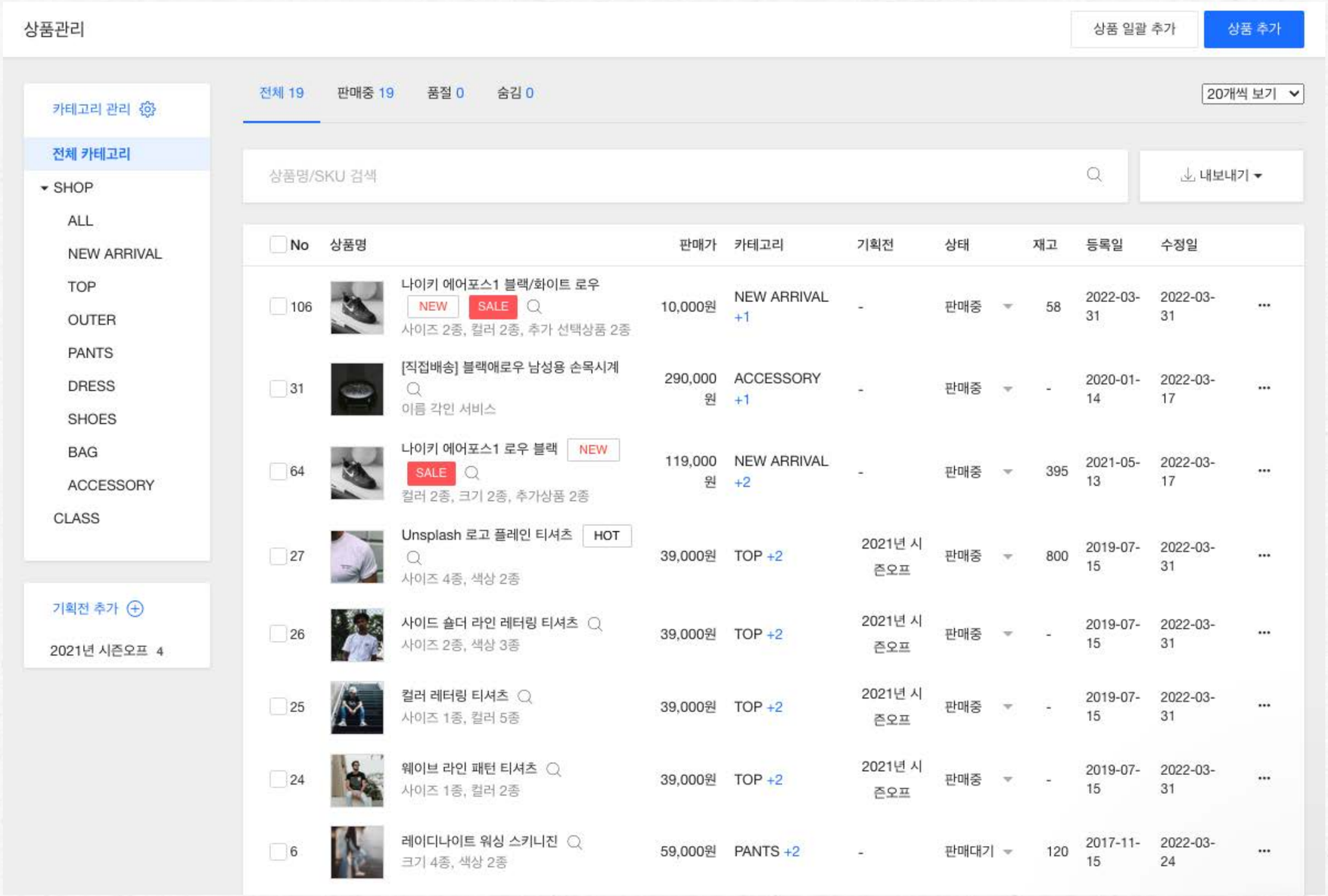
[배달의 민족]

프로젝트 개요

사례 분석

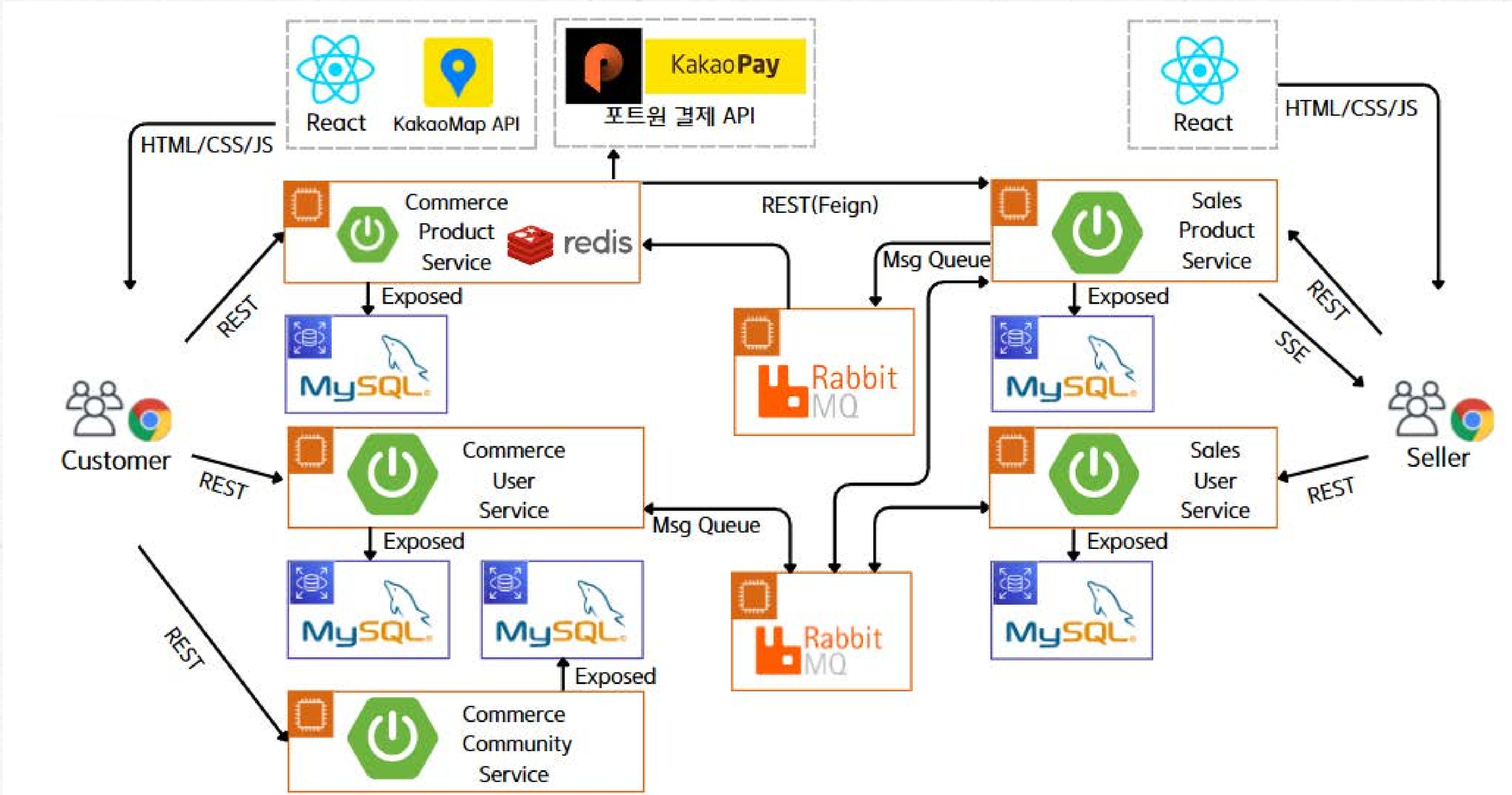
[아임웹]

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



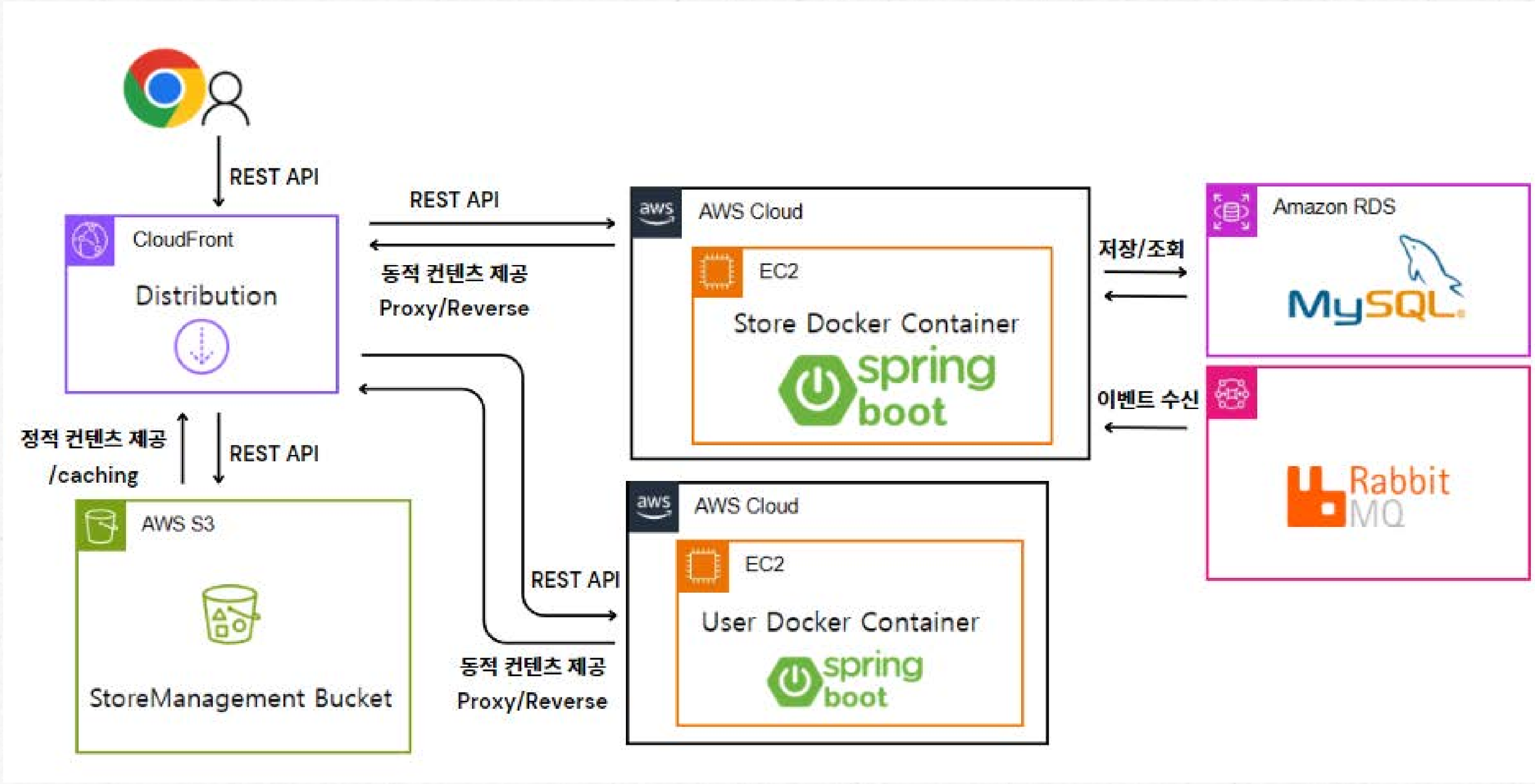
프로젝트 개요

팀 아키텍처



프로젝트 개요

개인 아키텍처



팀 편성 및 역할

김성우

기술고문
메인 페이지
제품 관련 노출
카카오 결제 API
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

박강찬

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

양수열

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

분석 및 설계

프로젝트 분석 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,판매가 ,재고의 수량, 제품의 가격, 주문 최대 수량,카테고리등을 변경 할 수있다.
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 나태낸다.
8	Must have	Top5 프로모션	전체 브랜드중에서 카테고리 별로 판매량이 가장 높은 제품 5개씩 보여준다.

Should have

번호(중요도)	우선순위	요구사항명	요구사항 내용
15	Should have	주문 알림	구매자가 주문한 제품ID를 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 판매자간의 실시간 문의 기능

프로젝트 분석 및 설계

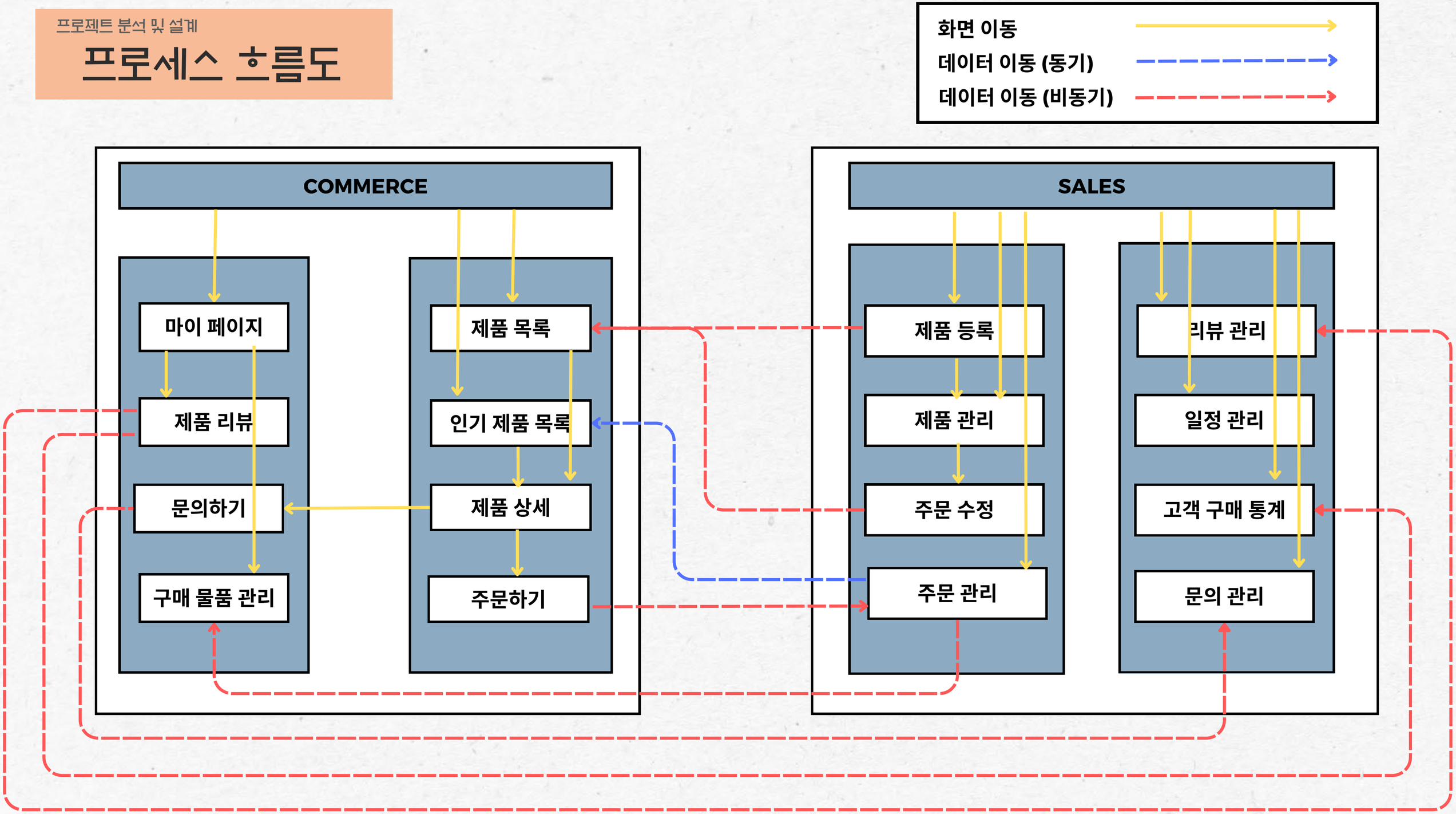
인터페이스 명세서

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,메인이미지,서브 이미지등 전송	비동기
2	Must have	제품 수정	sales	commerce	Message Queue	실시간 (요청)	수정 하는 제품의 ID, 수정된 데이터 전송	비동기
3	Must have	제품 이미지	commerce	sales	RestAPI	실시간 (요청)	제품 이미지 uuid요청	이미지 uuid에 해당하는 이미지 응답
4	Must have	주문 처리	commerce	sales	Message Queue	실시간 (요청)	구매자가 주문ID ,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	스케쥴 (1hour)	제품 ID 요청	카테고리별 탑 5 제품ID 응답
7	Must have	판매자 정보	sales(product)	sales (user)	RestAPI	실시간 (요청)	브랜드 이름 요청	브랜드 이름 응답

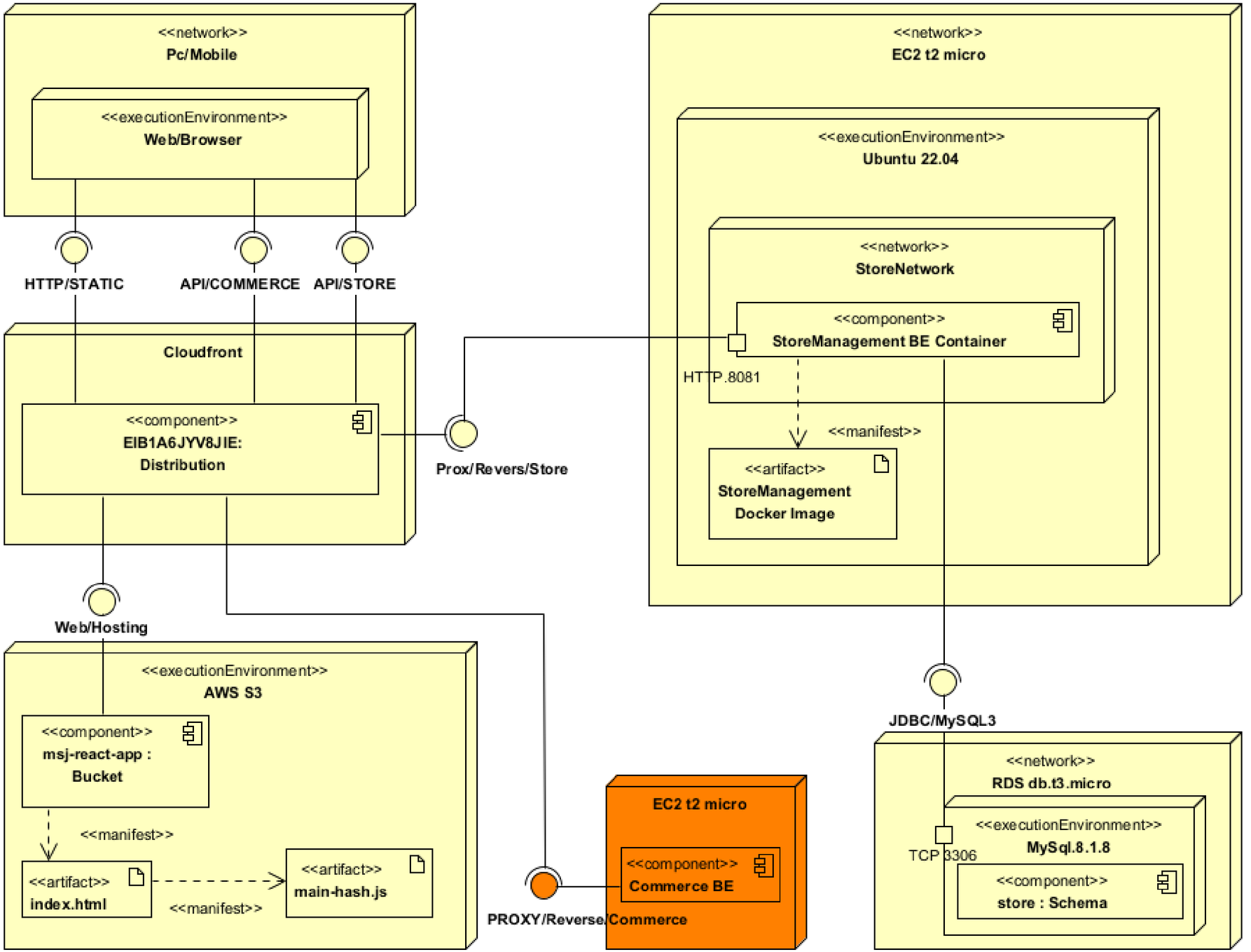
프로젝트 분석 및 설계

프로세스 흐름도



분석 및 설계

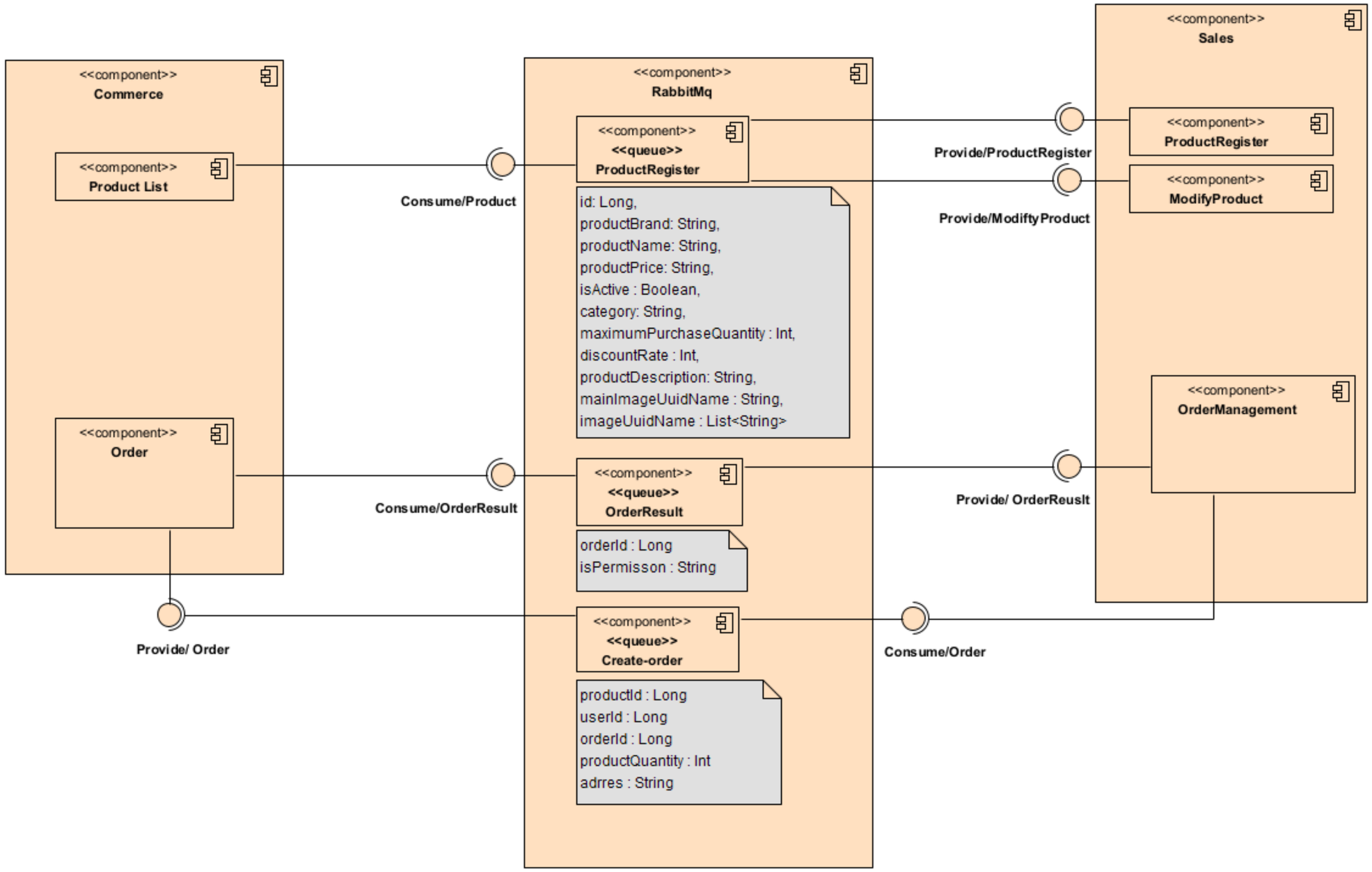
배포 다이어그램



분석 및 설계

Component Diagram

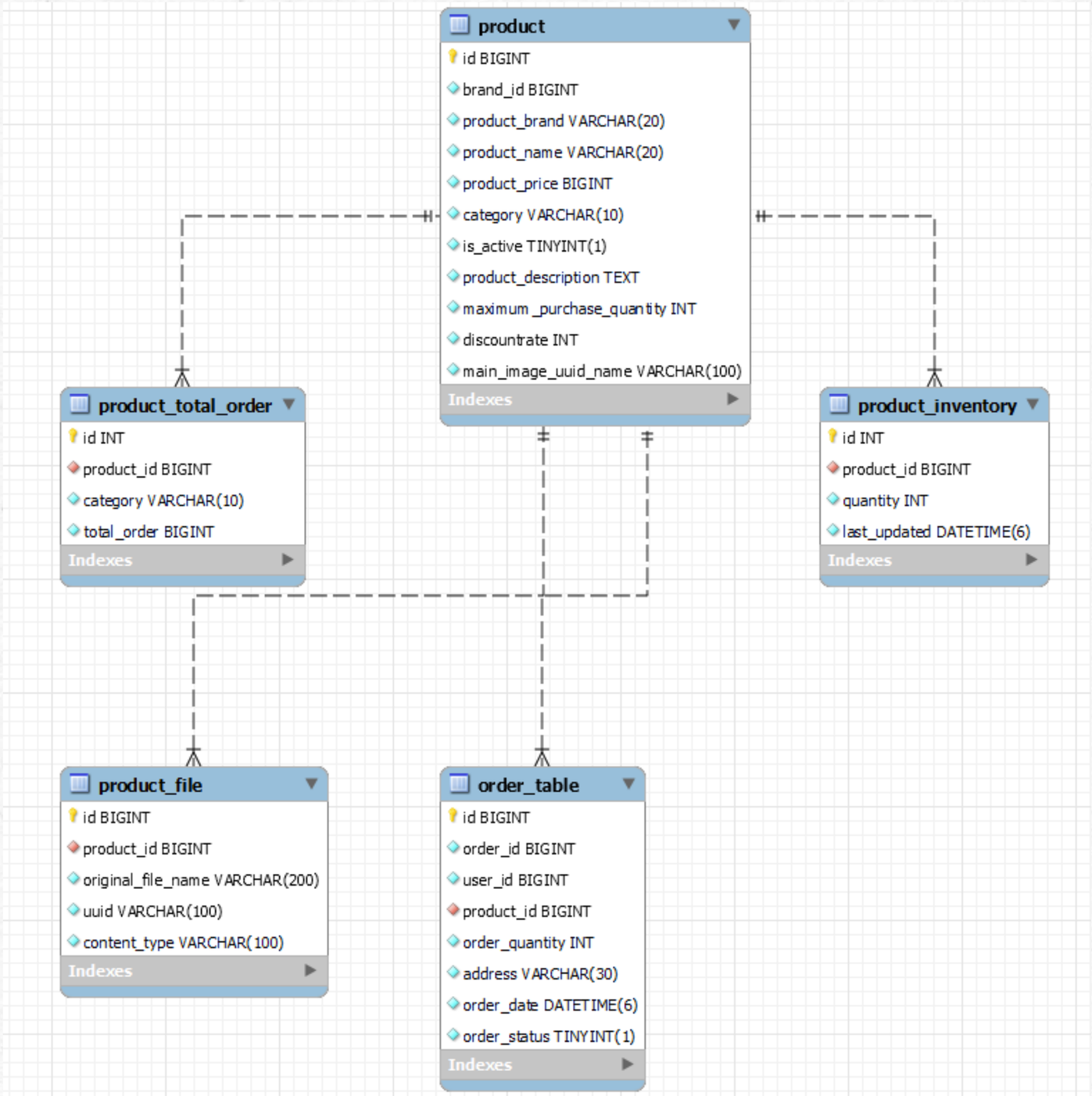
원활한 개발진행을 위해 컴포넌트 다이어그램을 설계 하였습니다.
소프트 웨어 공학 분석 설계 툴로 유명한 visual paradigm을 사용했습니다.



분석 및 설계

ERD 다이어그램

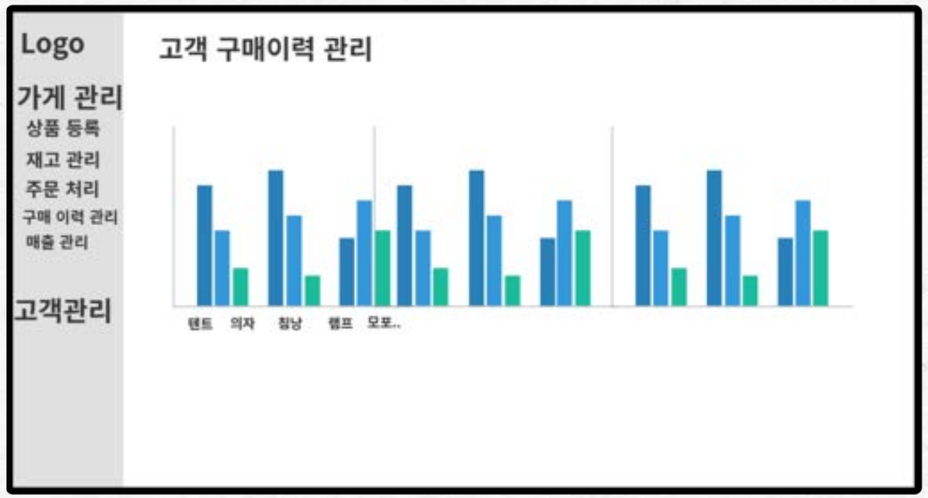
ERD 다이어그램을 통해 테이블간의 관계를 나타냈습니다.

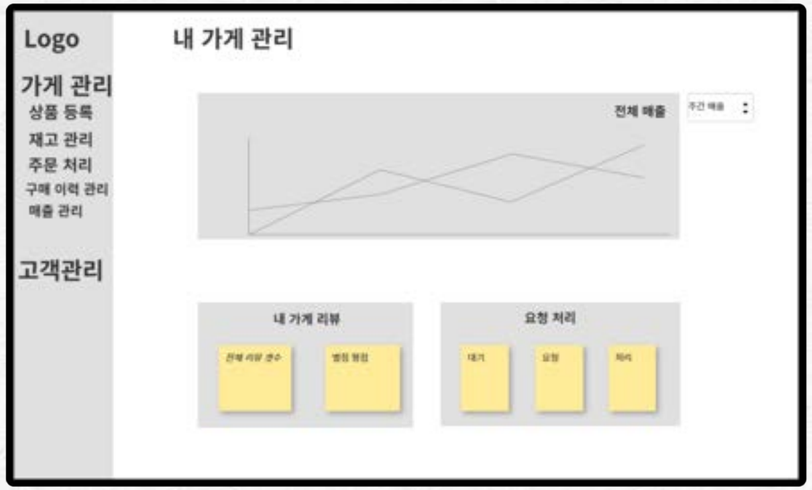


분석 및 설계

UI 프로토타입

KaKao Oven





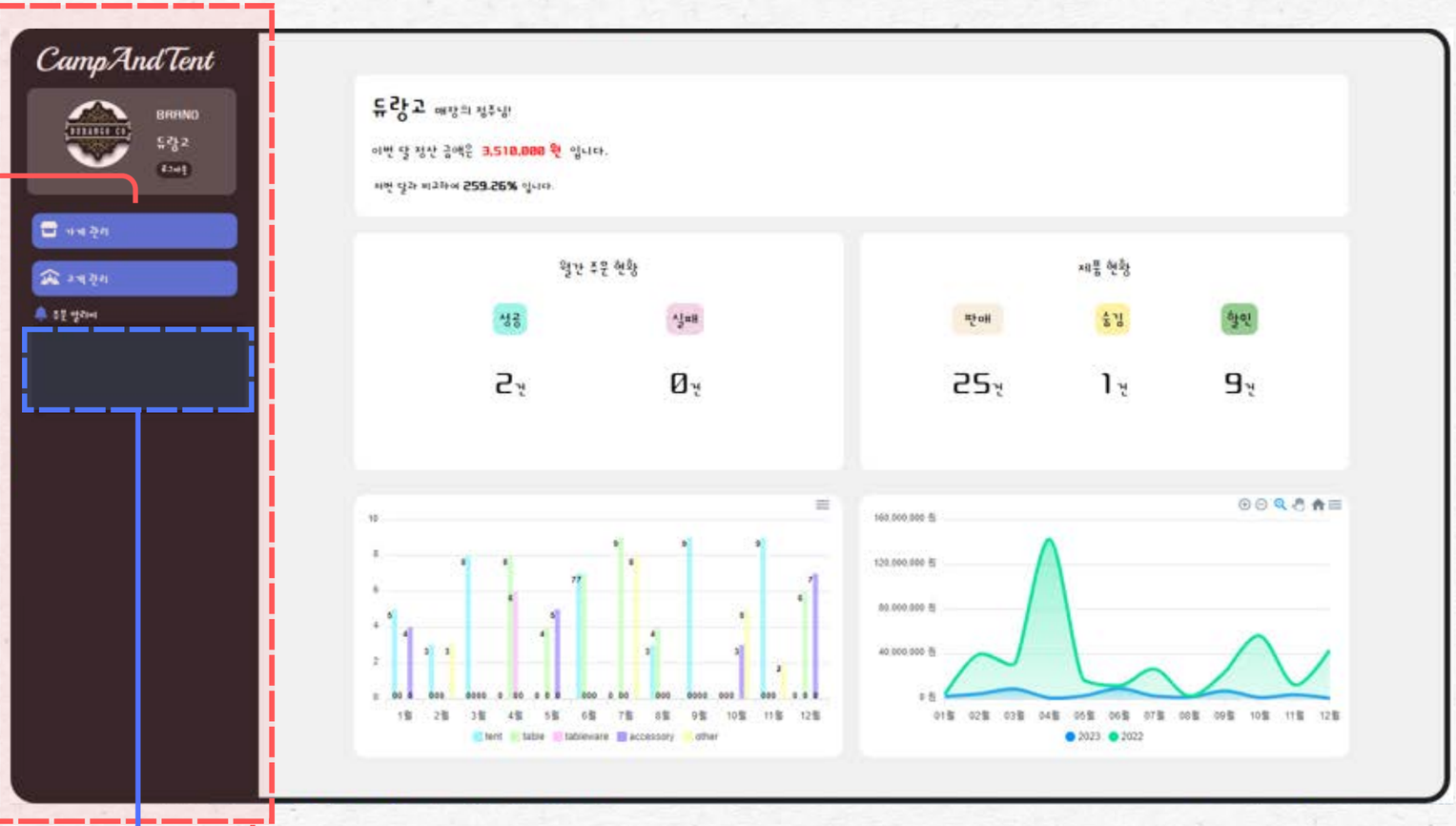
프로젝트 구현

프로젝트 구현

UI 구현

[메인 페이지]

- 클릭 시 네비게이션 바가 펼쳐집니다.



공통 컴포넌트

- 알림 useState부분

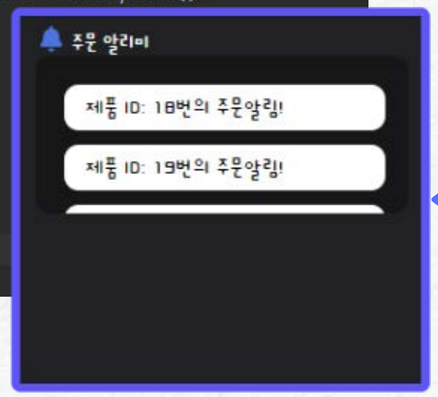
```
const [messages, setMessages] = useState<string[]>([]);

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

```
@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)
        }
    }
    emitters.removeAll(deadEmitters)
```



주문 발생 시 실시간 주문 알림
(Server-Sent-Event)

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

프로젝트 구현

메인페이지

[서비스 구현부]

REACT ROUTER v6

- 중첩라우팅 구현 부분

```jsx
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;
```

```jsx
return (
  <LayoutContainer>
    <nav>…
    </nav>
    <main>
      <Outlet />
    </main>
  </LayoutContainer>
);
};

export default Layout;
```

<main>
Outlet
<main>
영역

- 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

- Layout을 가장 상단에 위치시켜 공통 컴포넌트로 사용하였습니다.

프로젝트 구현

UI 구현

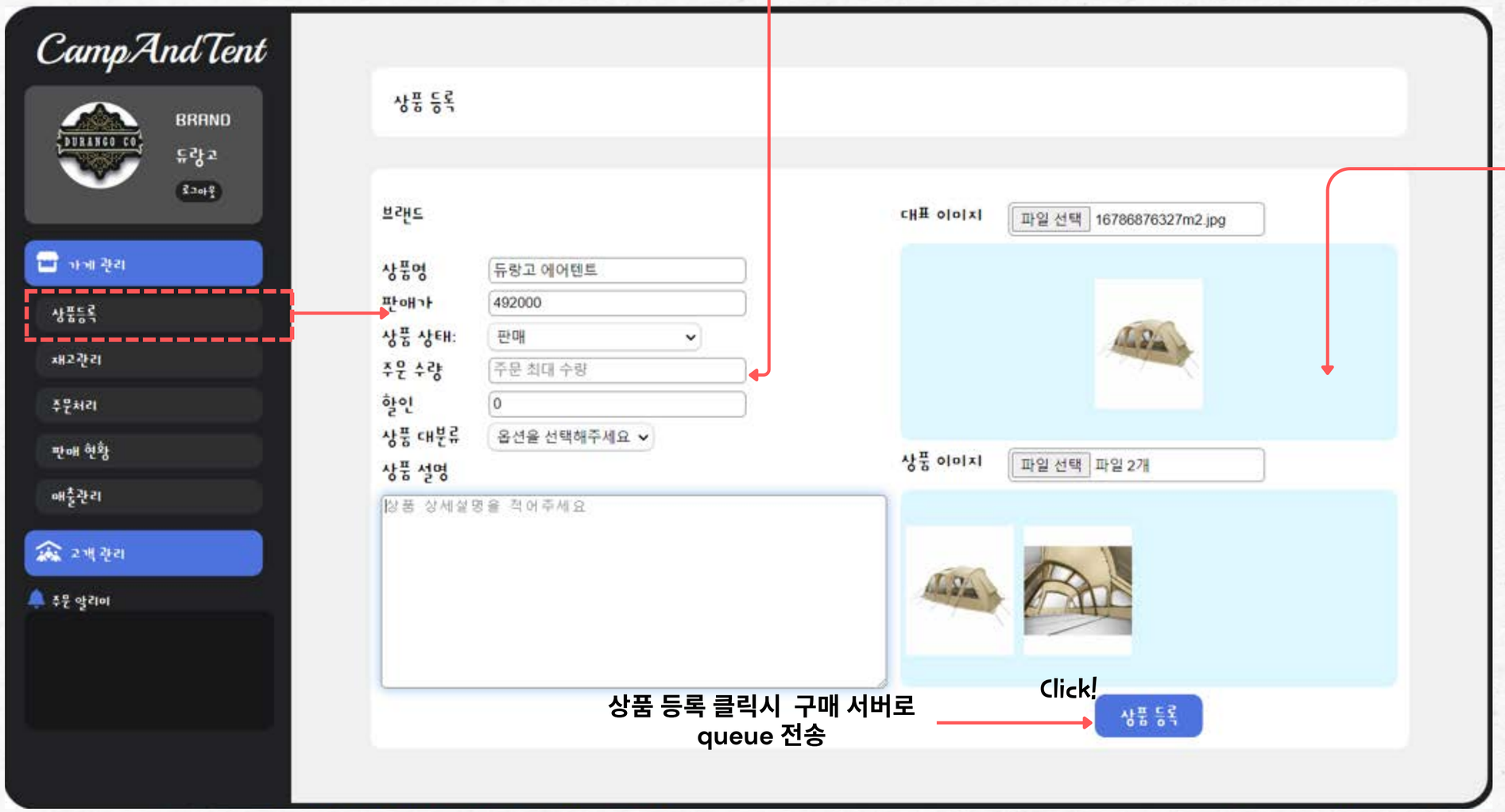
[상품 등록]

상품정보를 입력하지 않았을 때 경고창

- **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) {
        setMainImgPrewview(previews); // 수정된 부분
      }
    };

    reader.readAsDataURL(file);
  }
};
```

상품 등록 클릭시 구매 서버로 queue 전송

Click!

프로젝트 구현

상품 등록

[서비스 구현부]

EXPOSED

DSL사용

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

- 레코드 일괄 삽입

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


```
val insertProduct = p.insert { this: Product  it: InsertStatement<Number>
    it[this.productName] = productName
    it[this.brand_id] = authProfile.id  //로그인한 판매자id 삽입
    it[this.productBrand] = productBrand
    it[this.productPrice] = productPrice
    it[this.category] = category
    it[this.isActive] = isActive
    it[this.maximumPurchaseQuantity] = maximumPurchaseQuantity
    it[this.discountRate] = discountRate
    it[this.productDescription] = productDescription
    it[this.mainImageUuidName] = mainImgUuid
}
```

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

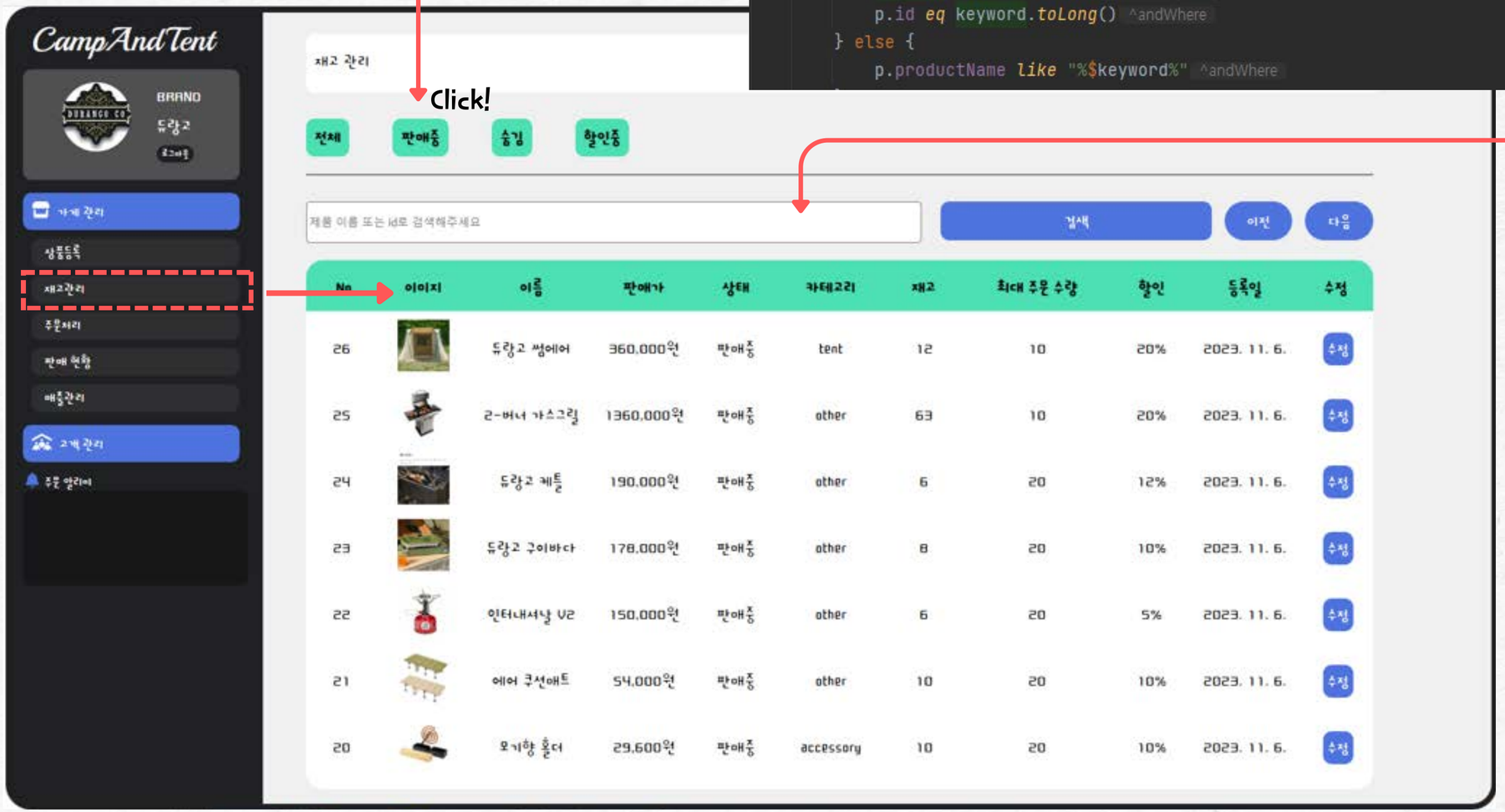
프로젝트 구현

UI 구현

[재고 관리]

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.isNullOrBlank()) {
    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/페이징 기능

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

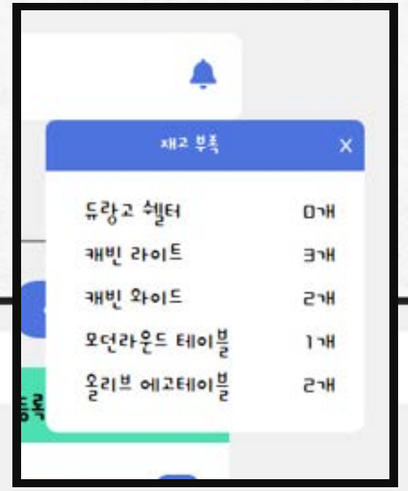
문자열 정규식 사용으로 숫자만 입력시 제품의 ID검색
String입력시 제품의 이름으로 검색됩니다.

프로젝트 구현

UI 구현

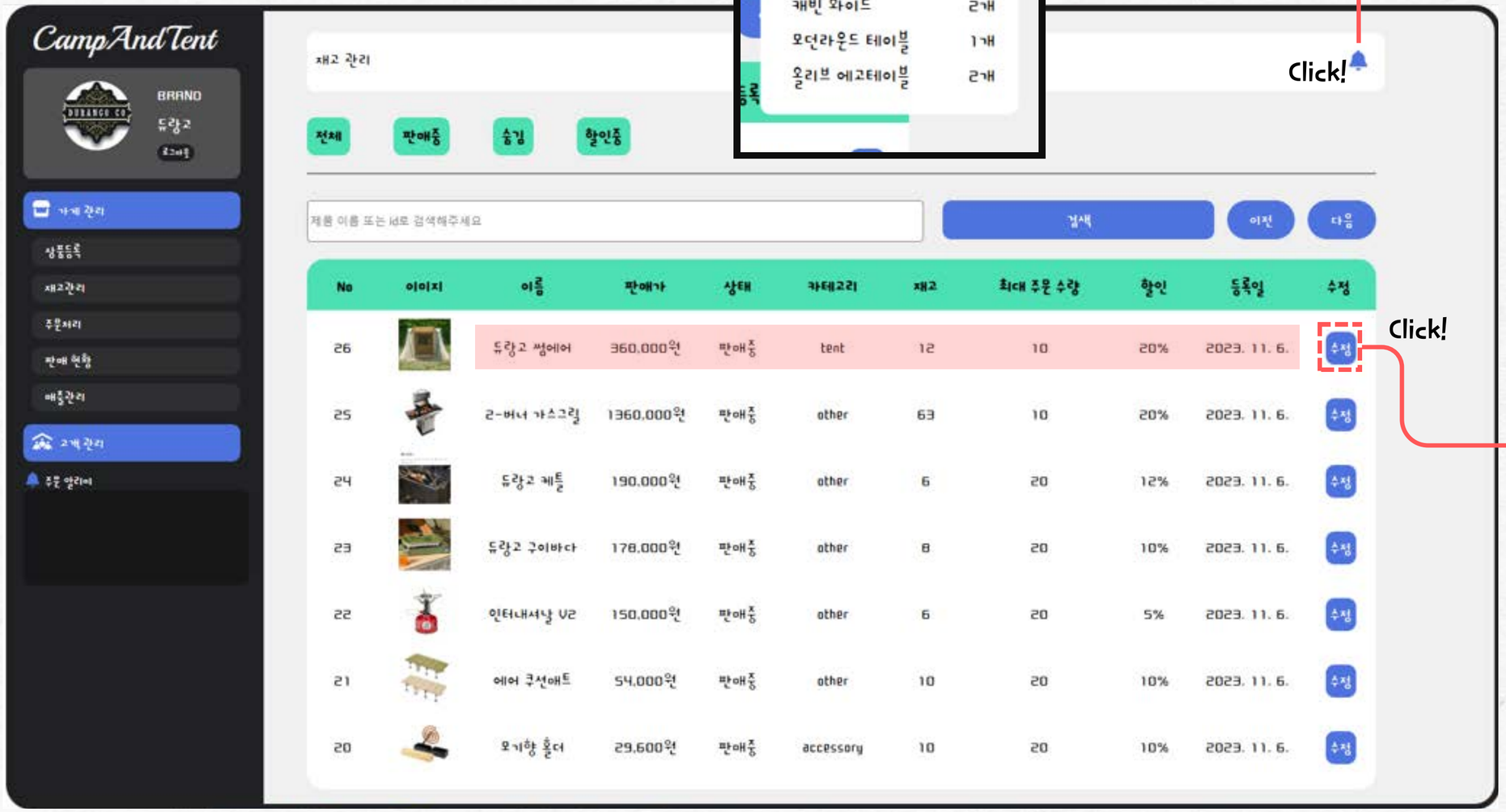
[재고 관리] 디테일

클릭시 재고가 3개 이하인 제품을 알려줍니다

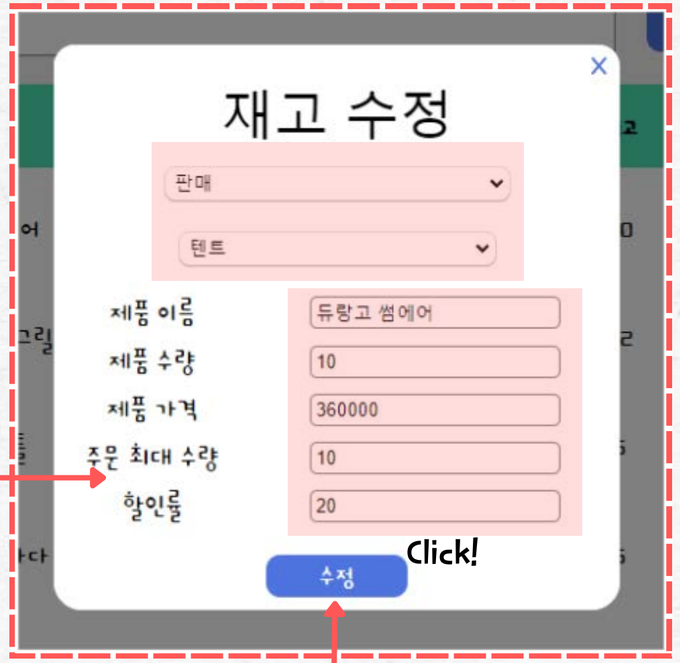


useRef코드와 화면에서 Ref value를 받아오는 부분입니다.

```
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>();
```



클릭시 제품 수정모달 창



클릭시 수정된 제품의 정보가
queue로 구매 페이지로 전달

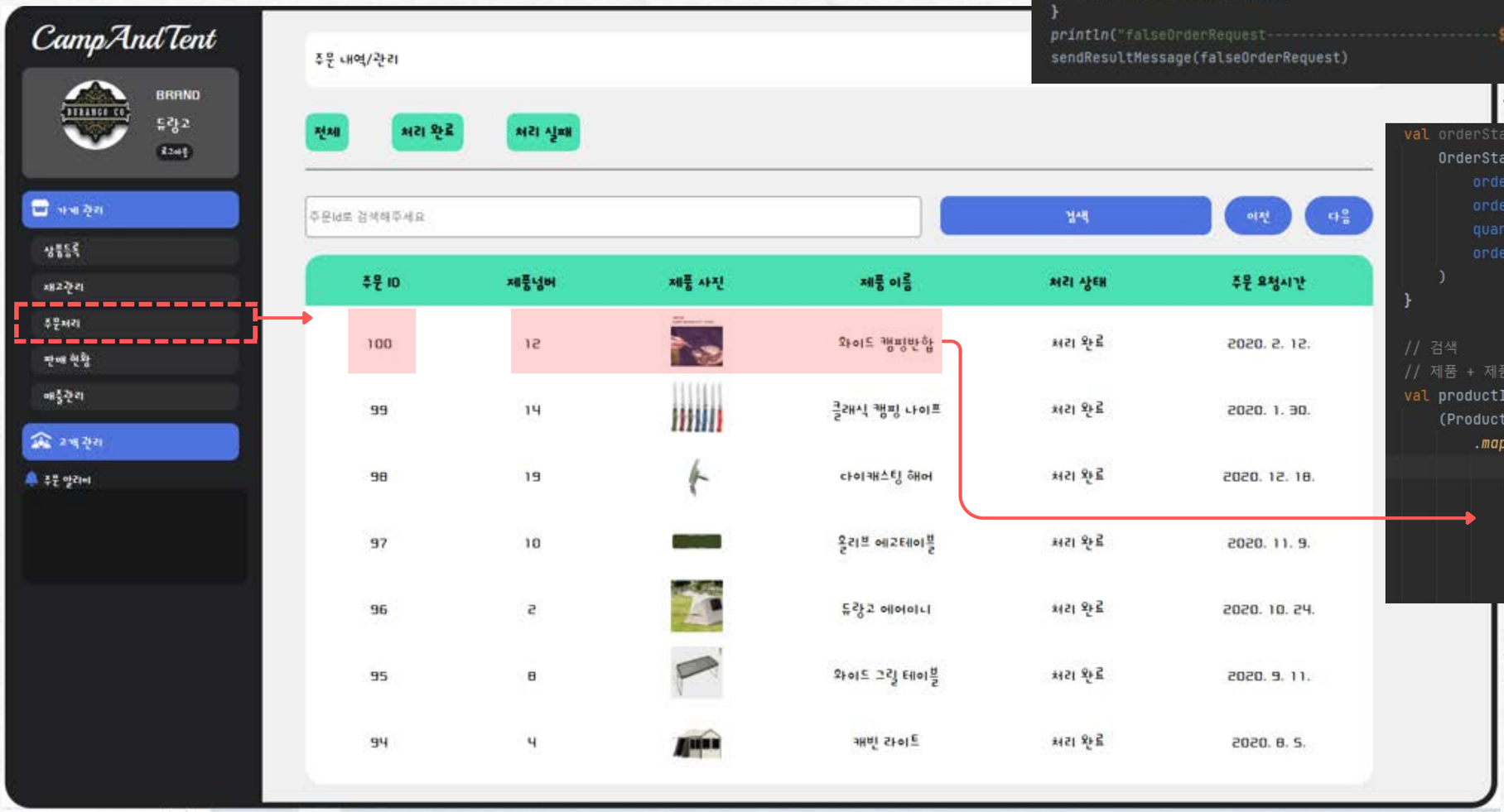
프로젝트 구현

UI 구현

[주문 관리]

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
```

프로젝트 구현

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.
- mutableMapOf를 활용하여 각 카테고리를 키로 하고 리스트의 크기를 12로 설정하여 연간 카테고리별 판매 현황 차트를 나타내었습니다.

프로젝트 구현

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) + "%";
}
```

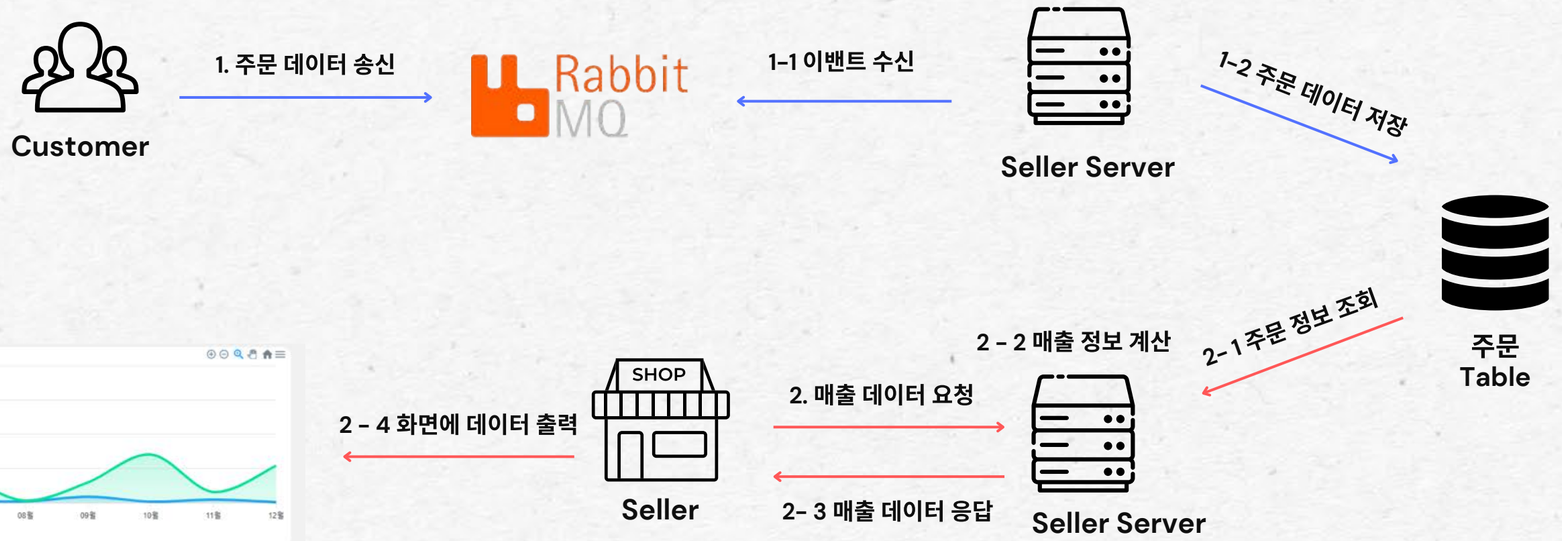
- Apex-Chart 라이브러리를 사용하는 컴포넌트는 React 클래스 컴포넌트로 제작하였습니다.
- 서버로부터 받은 저번 달과 이번 달의 정산 금액을 계산하여 상승률과 하락률을 사용자에게 알려줍니다.
- 사용자의 편의를 고려하여, 선택한 연도별 차트를 비교할 수 있도록 두 개의 차트로 구성하였습니다.

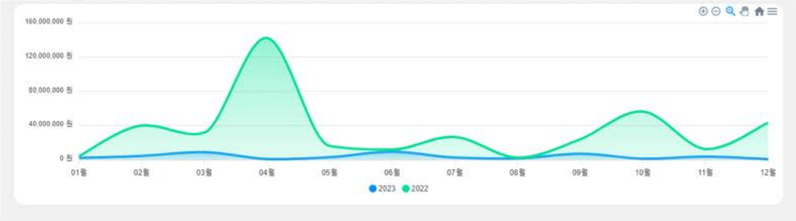
프로젝트 구현

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]

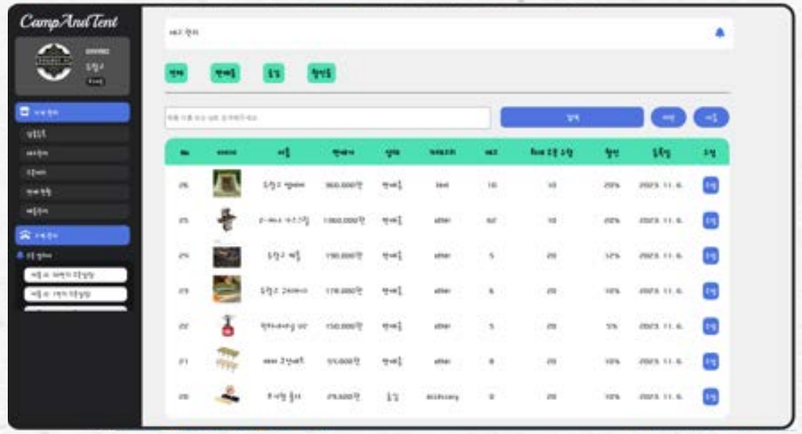
판매 제품 메시지 큐

Seller -> Customer

RabbitMQ

RabbitMQ를 활용해 서비스 간 데이터를 주고받도록 구현 했습니다.

Seller



1. 제품 등록

Seller
Data Table

2. 제품 데이터 저장

Seller Server

3. 제품 데이터 송신

RabbitMQ

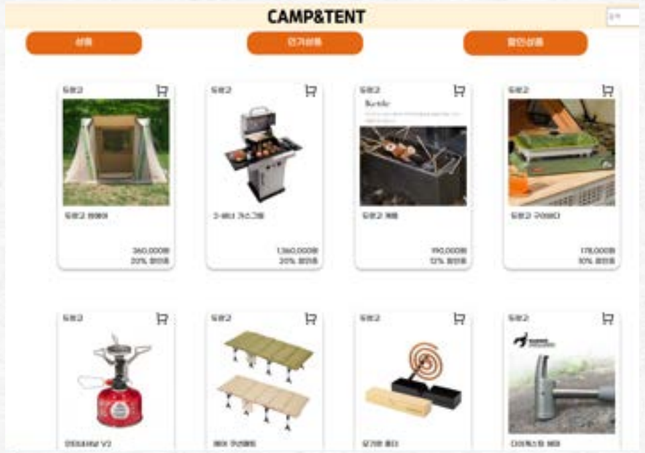
4. 제품 데이터 수신

Customer
Server

5. 제품 데이터 저장

2) 제품 데이터 조회

Customer
Data Table



1) 데이터 요청

3) 데이터 응답

Customer

Seller에서 등록한 제품을 Customer에서도 출력합니다.

프로젝트 구현

서비스 통합 구현부

[상품 등록]

Seller

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
/	inquery-response	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	product-inquery	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

Customer

```
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 구현부 입니다.

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

- 표준 메시지 큐 프로토콜인 AMQP를 사용하는 메시지 큐 브로커 RabbitMQ를 활용하여 Seller,Customer로 각 각 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

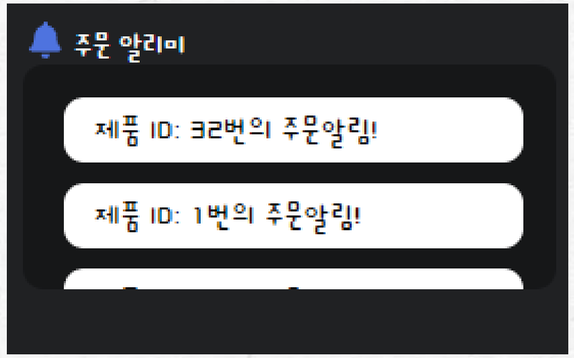
[주문 처리]

주문 메세지 큐

Customer -> Seller

RabbitMQ를 활용해 서비스 간 데이터를 주고받도록 구현 했습니다.

Customer가 주문한 주문 데이터를 Seller서버에서도 출력합니다.



Seller

1) 데이터 요청

3) 데이터 응답

※주문 알림 (SSE)

2) 제품 데이터 조회

5. 주문 데이터 저장

Seller Data Table

Seller Server

4. 주문 이벤트 수신

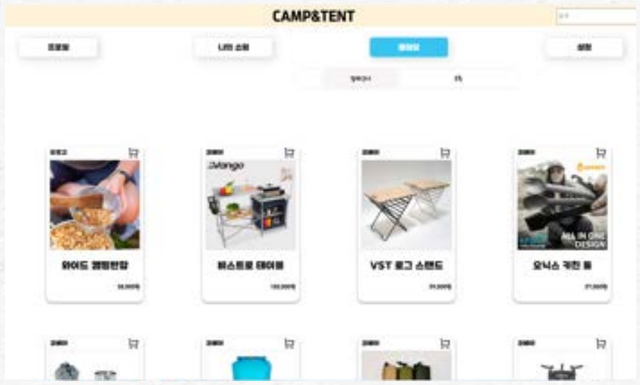
RabbitMQ

3. 주문 데이터 송신

Customer Server

2. 주문 데이터 저장

Customer Data Table



1. 주문 생성

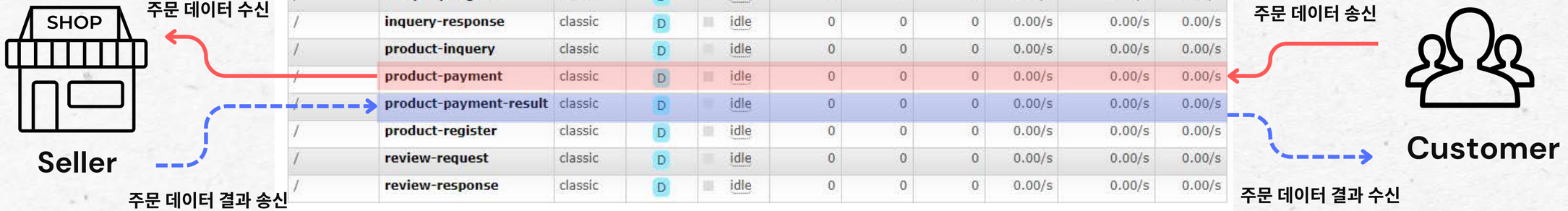
Customer

프로젝트 구현

서비스 통합 구현부

[주문 처리]

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



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
/	inquery-response	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/	product-inquery	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

```
@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)
```

- 주문처리하는 RabbitListener의 구현부 입니다.

```
        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)
```

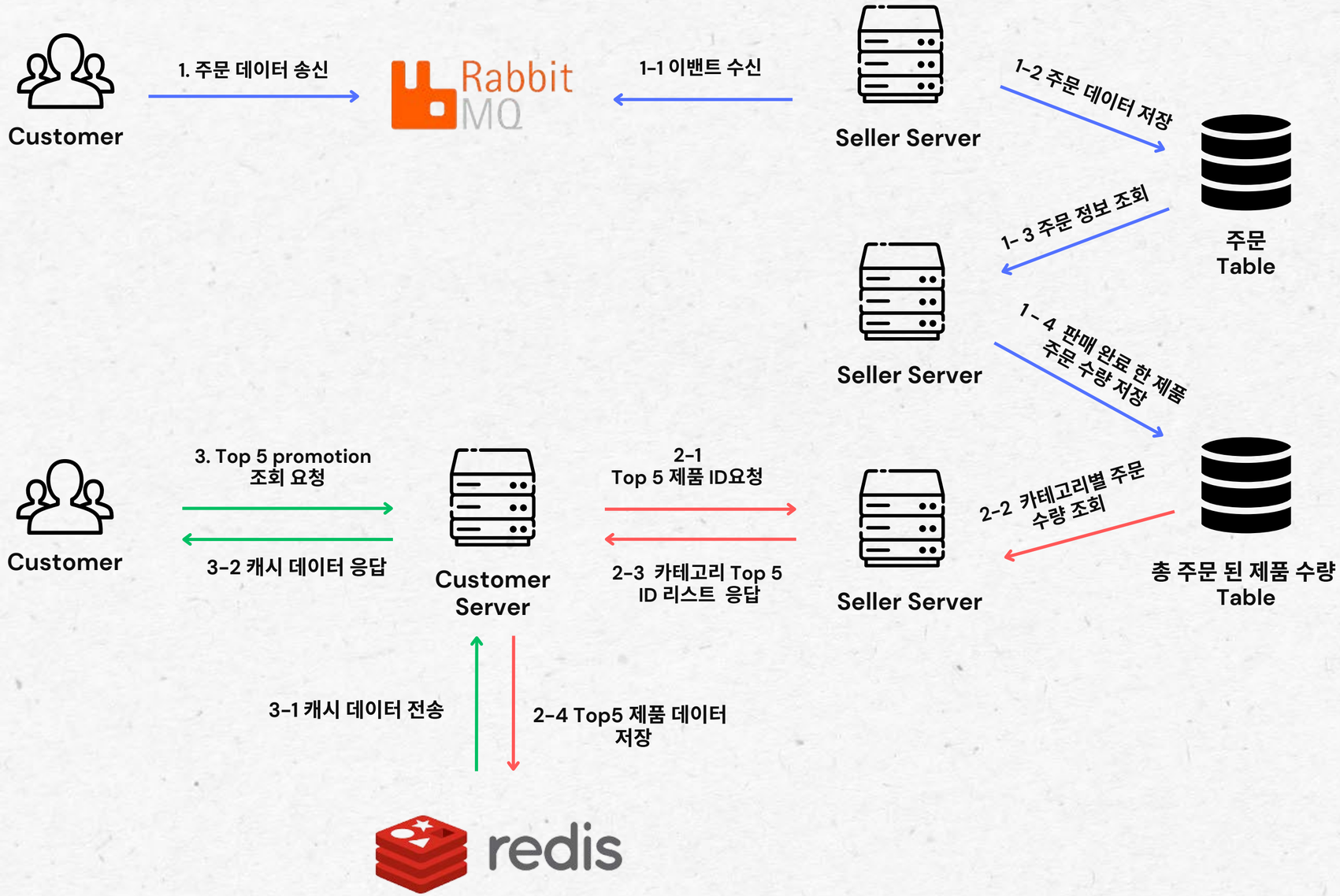
프로젝트 구현

서비스 통합 구현부

Top 5
Promotion

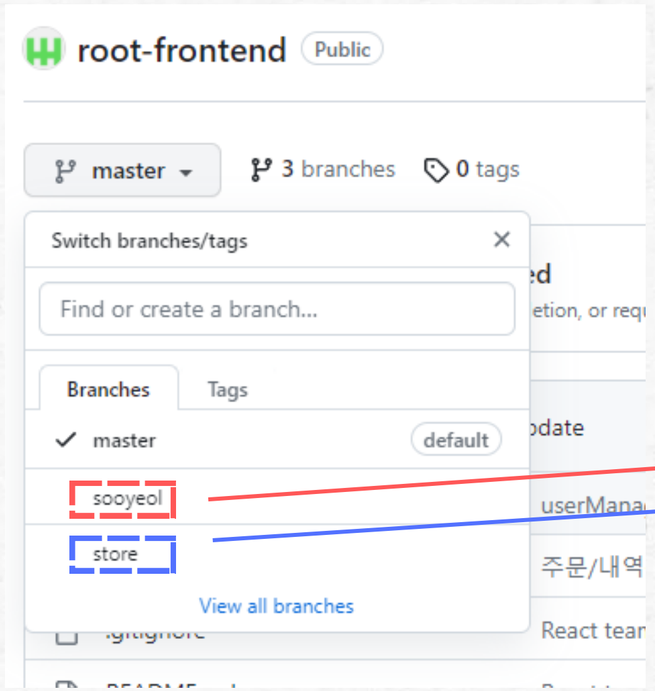
redis

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



프로젝트 협업

[git]




```
<ProfileProvider>
  <BrowserRouter>
    <ResetStyle />
    <Routes>
      <Route path="/" element={<LoginComponent />} />
      <Route path="/home" element={<Layout />}>
        {UserManagementRoutes}
        {StoreManagementRoutes}
      </Route>
    </Routes>
  </BrowserRouter>
</ProfileProvider>
```

- 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

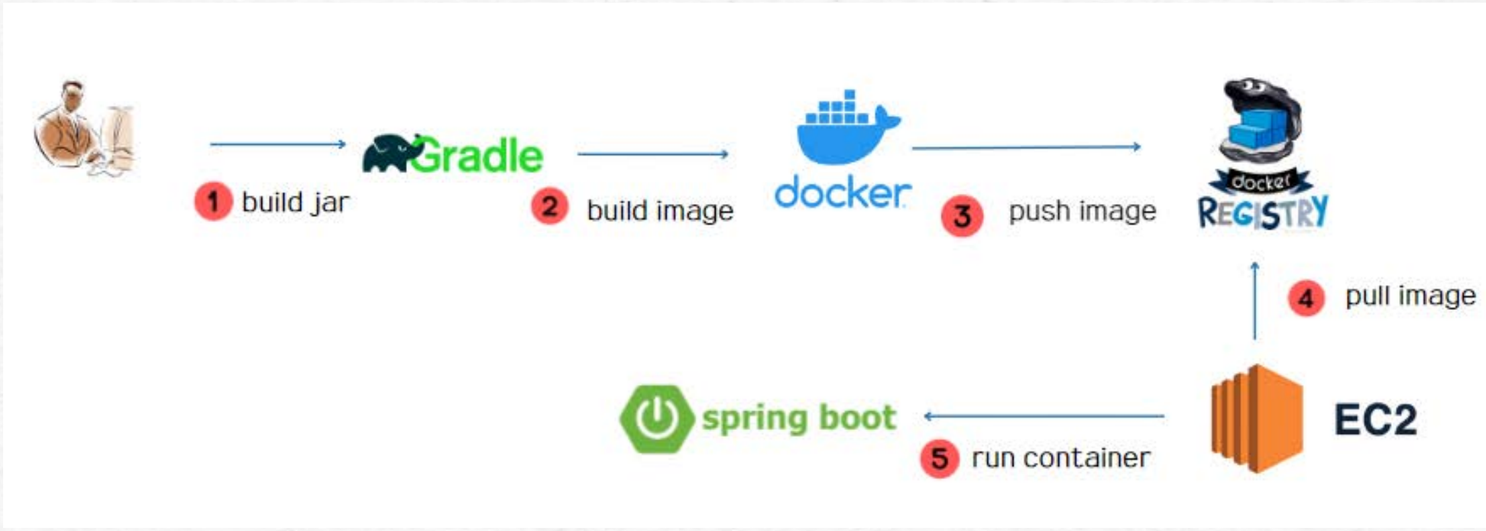
```
PS C:\root-frontend> git merge origin/master
Updating b0ca0f5..4dac8f6
Fast-forward
 src/modules/Layout.tsx                                  |  13 +--
 .../InqueryManagement/InqueriesContainer.tsx            | 130 ++++++++++++++++++++
 .../InqueryManagement/answeredInqueries/index.tsx       |  50 ++++++++
 .../UserManagement/InqueryManagement/types.ts           |  15 +++
 .../unansweredInqueries/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.tsx                   |   8 +-
 13 files changed, 508 insertions(+), 107 deletions(-)
 create mode 100644 src/modules/UserManagement/InqueryManagement/InqueriesContainer.tsx
 create mode 100644 src/modules/UserManagement/InqueryManagement/answeredInqueries/index.tsx
 create mode 100644 src/modules/UserManagement/InqueryManagement/types.ts
 create mode 100644 src/modules/UserManagement/InqueryManagement/unansweredInqueries/index.tsx
 delete mode 100644 src/modules/UserManagement/ReviewManagement/api/api.ts
```

- 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 주지않도록 안정적으로 진행되었습니다.
- 각 브랜치는 특정 기능이나 작업 단위로 구성되어 유지보수를 용이하게 만들었습니다.

프로젝트 배포

[서버]

작업 초반에는 직접 docker images build 및 rmi images, push,pull을 하여
전체적인 흐름을 이해하고 후반 부에 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:d94cda0374d261ec815f488bb979e5
b632dd6bb3e8b1ec90f8b59e1
Deleted: sha256:f8205d85508ff53cb11f03a14ae43808ae2c65cd4cfa7ffe47f7bed
Deleted: sha256:5d550f5e7d7dc5dc33cdd40a3f8068a30e6ed2ee880510d7dbc5104
Deleted: sha256:294d117a238d324a417a179d207af30a564fb27bc3eac03a4c2a5c6
Deleted: sha256:c2f7aced94ac810d00cdde4b94ea693e0ad339ef279b624e9434c2c
Deleted: sha256:f5f1090da1060835f40ae35740341a4374ef9877b4c1d512c49ebb8
Deleted: sha256:39a2259e820b896dff0aea9818be884077edc9895b361c8e670300c
Deleted: sha256:8ceb9643fb36a8ac65882c07e7b2fff9fd117673d678422la83d3ad
```

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]

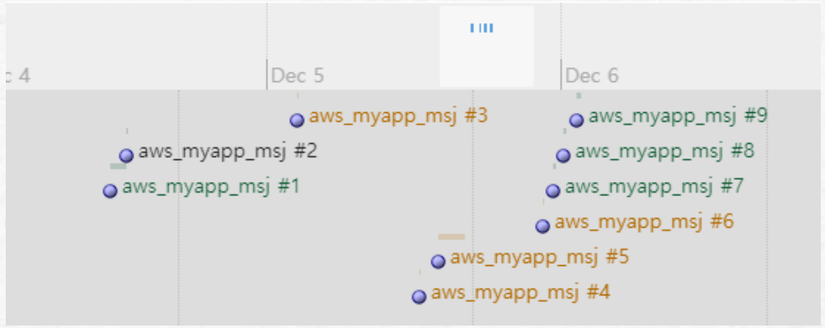
작업 초반에는 직접 docker images build 및 rmi images, push,pull을 하여 전체적인 흐름을 이해하고 후반 부에 Jenkins 자동 배포를 진행하였습니다.

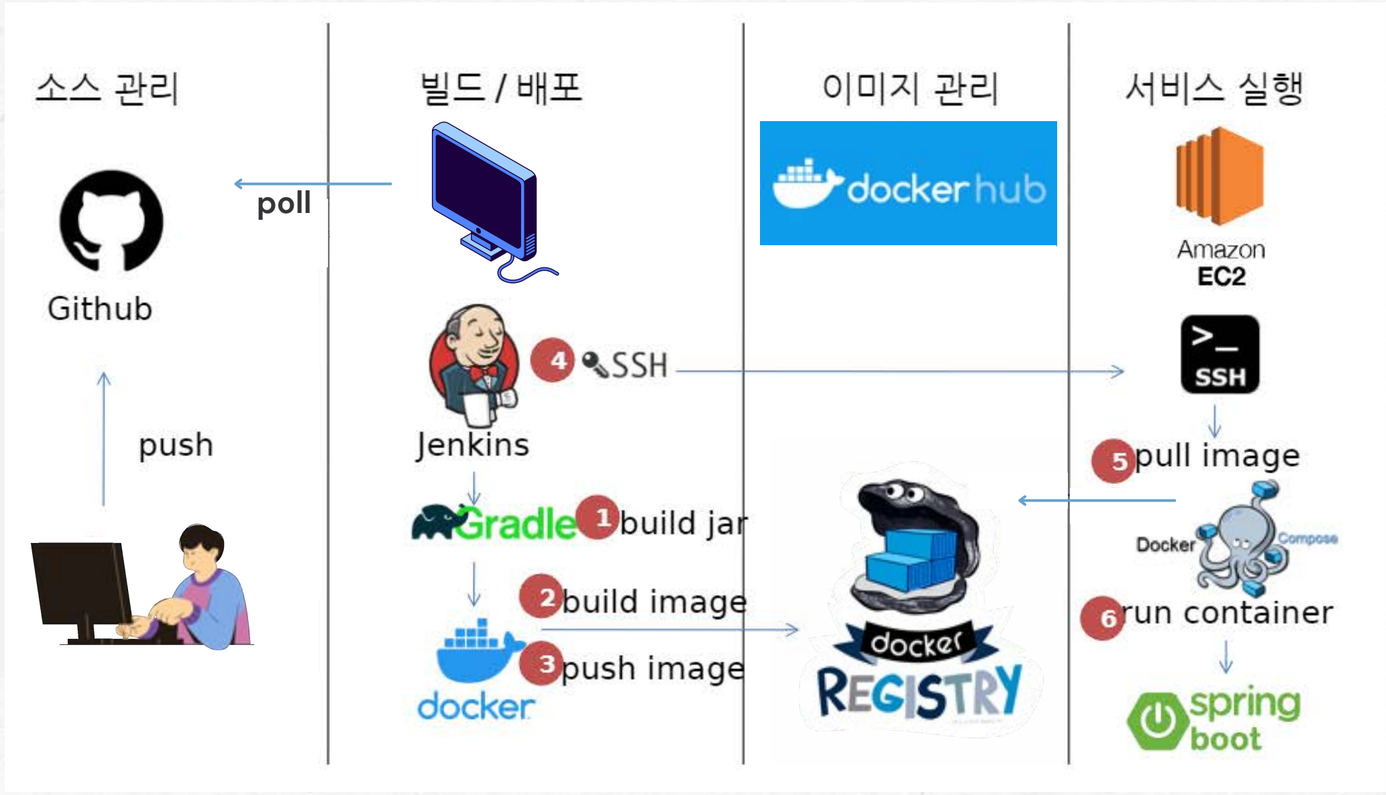
Console Output

콘솔 출력

```
18:22:03 Status: Downloaded newer image for myungsj/storemanagement:latest
18:22:03 docker.io/myungsj/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

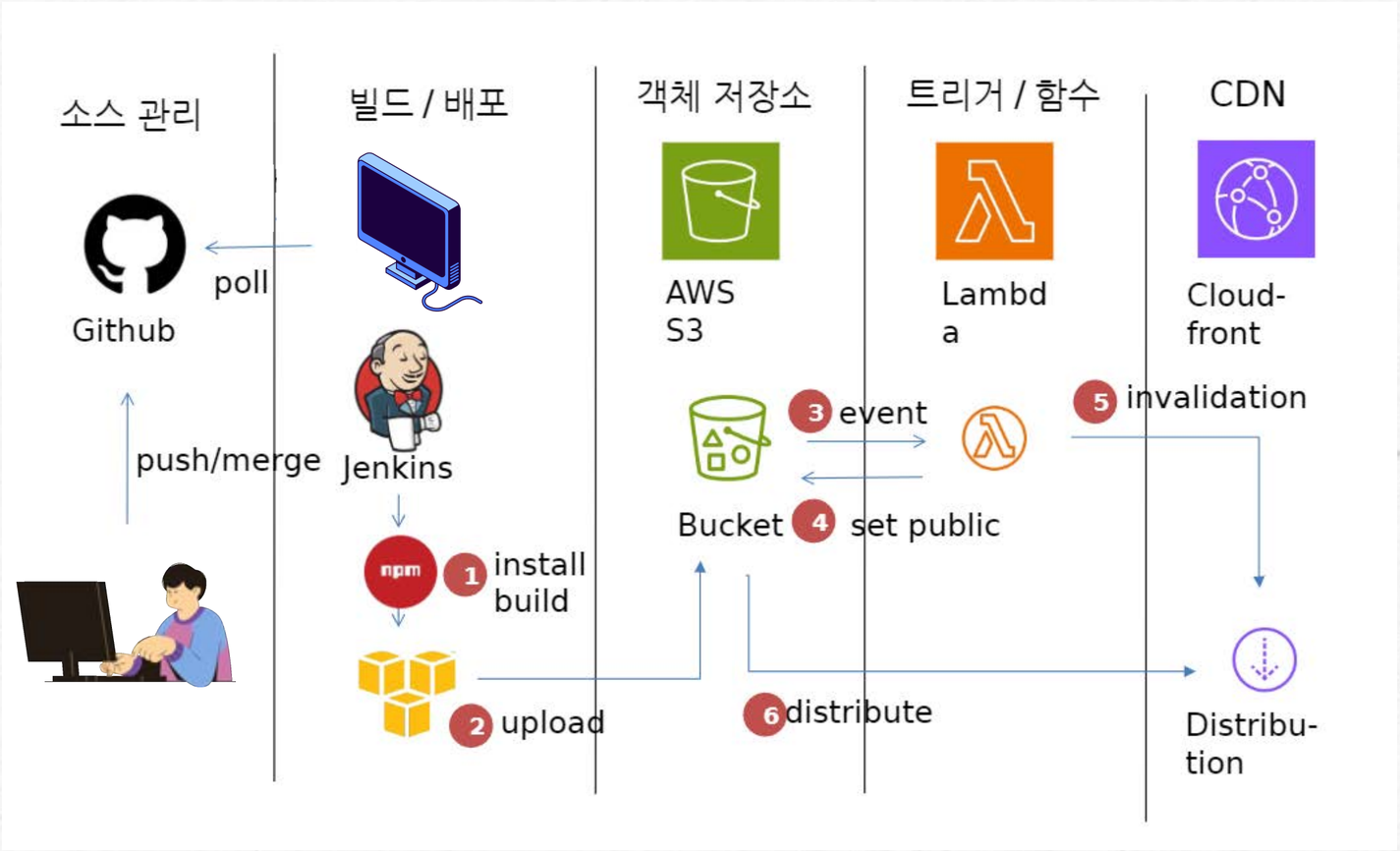




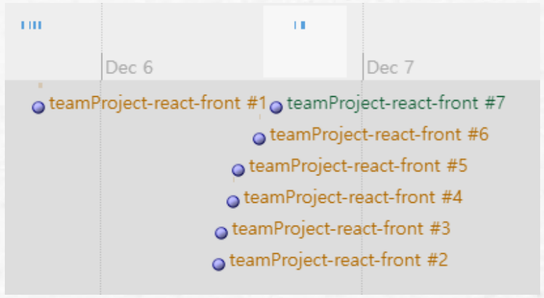
프로젝트 배포

[프론트]
[Jenkins CI/CD]

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



Jenkins의 빌드 기록



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
```

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]
lated asset
  asset index.html 423 bytes [emitted] [compared for emit]
```

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://d1mqxkgco7he9i.cloudfront.net/