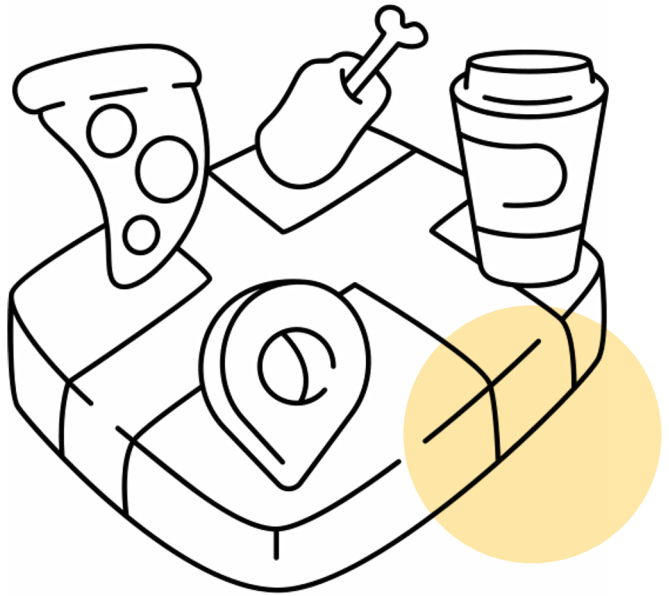
[멀티잇] 백엔드 개발자 취업캠프 9회차

최종 포트폴리오 - 2조 (맛.java)

개인 포트폴리오 _ 이후성

MAT. ZIP



“진짜 믿고 먹을 수 있는 맛집” 을 공유하는 플랫폼을 만들기 위해 뭉친 2조 맛.java 팀 입니다.

조장
함영휘

조원
김규환

조원
추재영

조원
전혜진

조원
이후성

조원
정서현

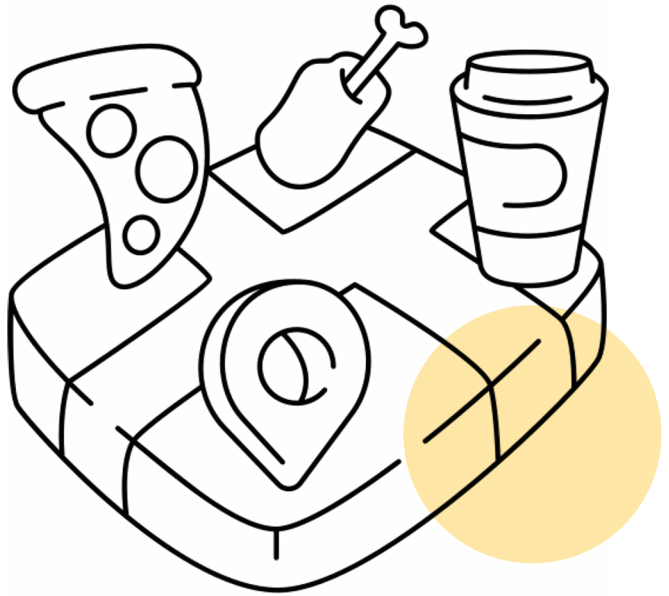
조원
최윤서

05. 개인 기능 구현 - 이후성

사장 커뮤니티

Team Mat. java

MAT. ZIP



개인별 요구사항 정의서

프로젝트명	맛.zip		
작성자	이후성	작성일	2023.05.03
버전	1.0.0	최종수정일	2023.06.06

구분	요구ID	내용	비고
User Requirement -Service	LWS-001	구독결제	구독한 사장회원이 아니면 페이지열람 X
	LWS-002	사장회원등록	구독결제완료시 회원등록완료
	LWS-003	결제내역 저장	결제완료 시 결제내역 DB에 저장
	LWS-004	자유게시판 기능	글 작성,수정,삭제,검색
	LWS-005	댓글 기능	댓글 달면 페이지 이동 하지 않고 바로 보여주기
	LWS-006	조회수 및 좋아요 기능	상세페이지 접속시마다 조회수 카운트, 좋아요버튼
	LWS-007	매출차트	매출장부 들어가면 최근 3달의 매출현황
	LWS-008	신규고객과 재방문고객 비교차트	재방문율, 결제금액,주문 수등을 비교한 차트
	LWS-009	리뷰 감정분석 차트	우리가게 긍정,부정리뷰 비율을 그래프화
Functional Requirement -Technology	LWS-010	구독결제	일반회원과 사장회원을 세션을 나눠서 B2C분리, 토스 결제 API 사용
	LWS-011	사장회원등록	구독결제완료시 회원등록
	LWS-012	결제내역 저장	결제완료 시 토스api로부터 응답받은 정보를 ajax 이용해서 결제내역DB에 insert
	LWS-013	자유게시판 기능	DB의 데이터를 가져올때 역순정렬 배치 , 제목과 내용으로 검색, 수정 삭제시 이전에 작성했던 목록 불러오기
	LWS-014	댓글 기능	ajax이용하여 댓글 CRUD처리
	LWS-015	조회수 ,좋아요 기능	페이지 접속당 조회수 +1 카운트, 좋아요버튼 누를때 DB 에 글번호와 유저 ID 저장해서 +1,-1구분
	LWS-016	매출차트	payment테이블의 amount, regdate데이터를 가져와서 구글차트api이용
	LWS-017	신규고객과 재방문 고객 비교차트	order테이블의 기록된 id가 2번이상이면 재방문으로 체크 1번인 데이터와 비교해서 구글차트api이용
	LWS-018	리뷰 감정분석 차트	리뷰DB에서 store_id와 review_content 값을 가져와서 sentimentAI api 이용해 분석해서 비율 통계
Non Functional Requirement -Protection &Change	LWS-019	결제시스템	토스 페이먼츠 id등록 테스트키 받기
	LWS-020	리뷰 감정분석	네이버 감정분석 API 신청
	LWS-021	구글차트	구글 차트 API 종류 선별

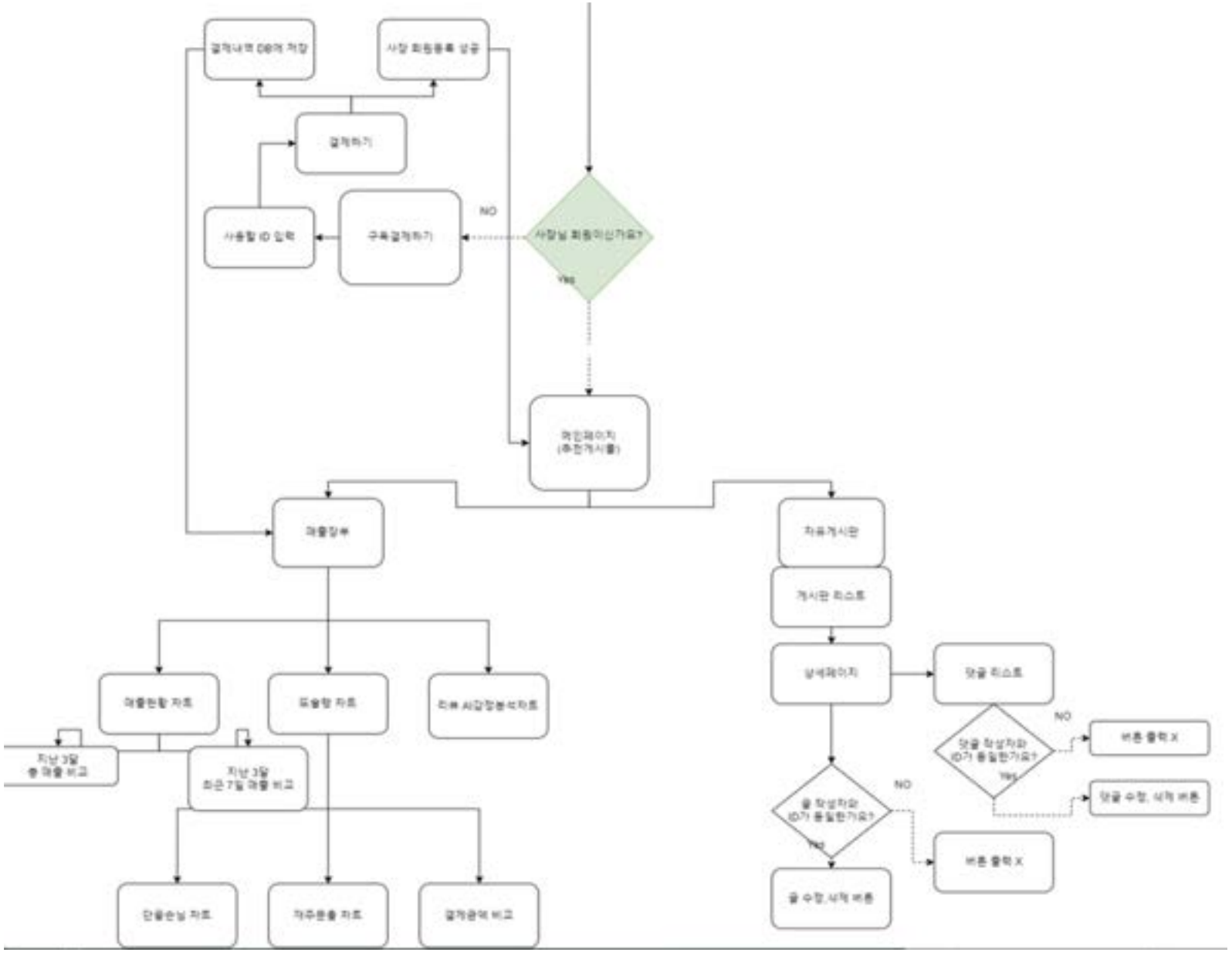
Table_LWS_01						
테이블ID		boss_member		작성자	이후성	
테이블설명		회원정보, 회원가입, 로그인 관리 테이블.				
번호	칼럼ID	형식	NULL	KEY	내용	비고
1	user_id	VARCHAR	Not Null	PK	회원 아이디	
2	password	VARCHAR	Not Null		회원 패스워드	
3	store_id	VARCHAR	Not Null		상점ID	
4	accountdate	VARCHAR	Not Null		등록일자	
5	nickname	VARCHAR	Not Null		닉네임	

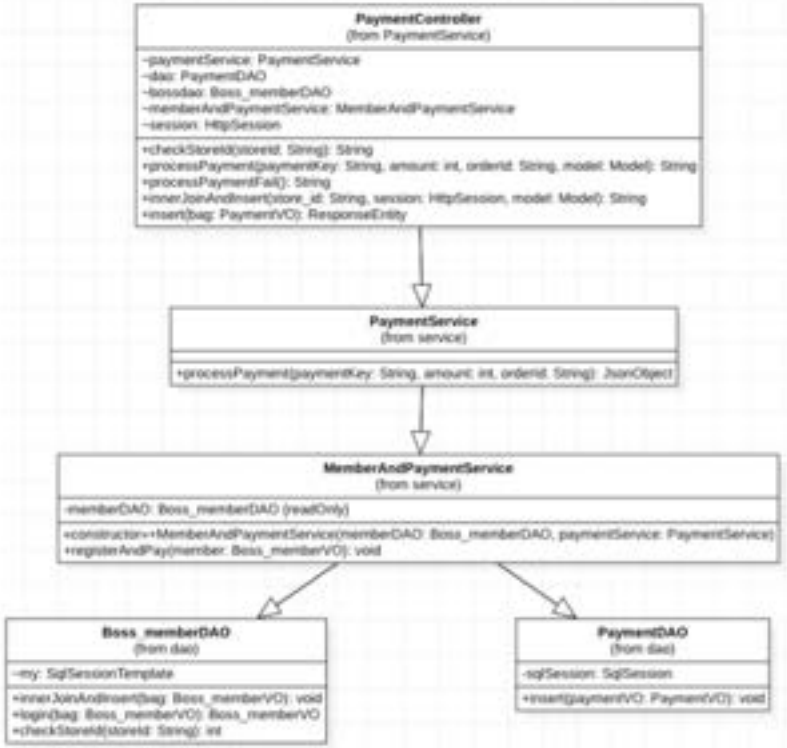
Table_LWS_02						
테이블ID		board		작성자	이후성	
테이블설명		게시판				
번호	칼럼ID	형식	NULL	KEY	내용	비고
1	board_id	INT	Not Null	PK	게시판번호	auto increment
2	writer	VARCHAR	Not Null		글 작성자	
3	title	VARCHAR	Not Null		글 제목	
4	content	VARCHAR	Not Null		글 내용	
5	regdate	DATETIME	Not Null		등록일자	디폴트값
6	update	DATETIME			수정일자	디폴트값
7	viewscount	INT			조회수	디폴트0
8	likecount	INT			좋아요수	디폴트0

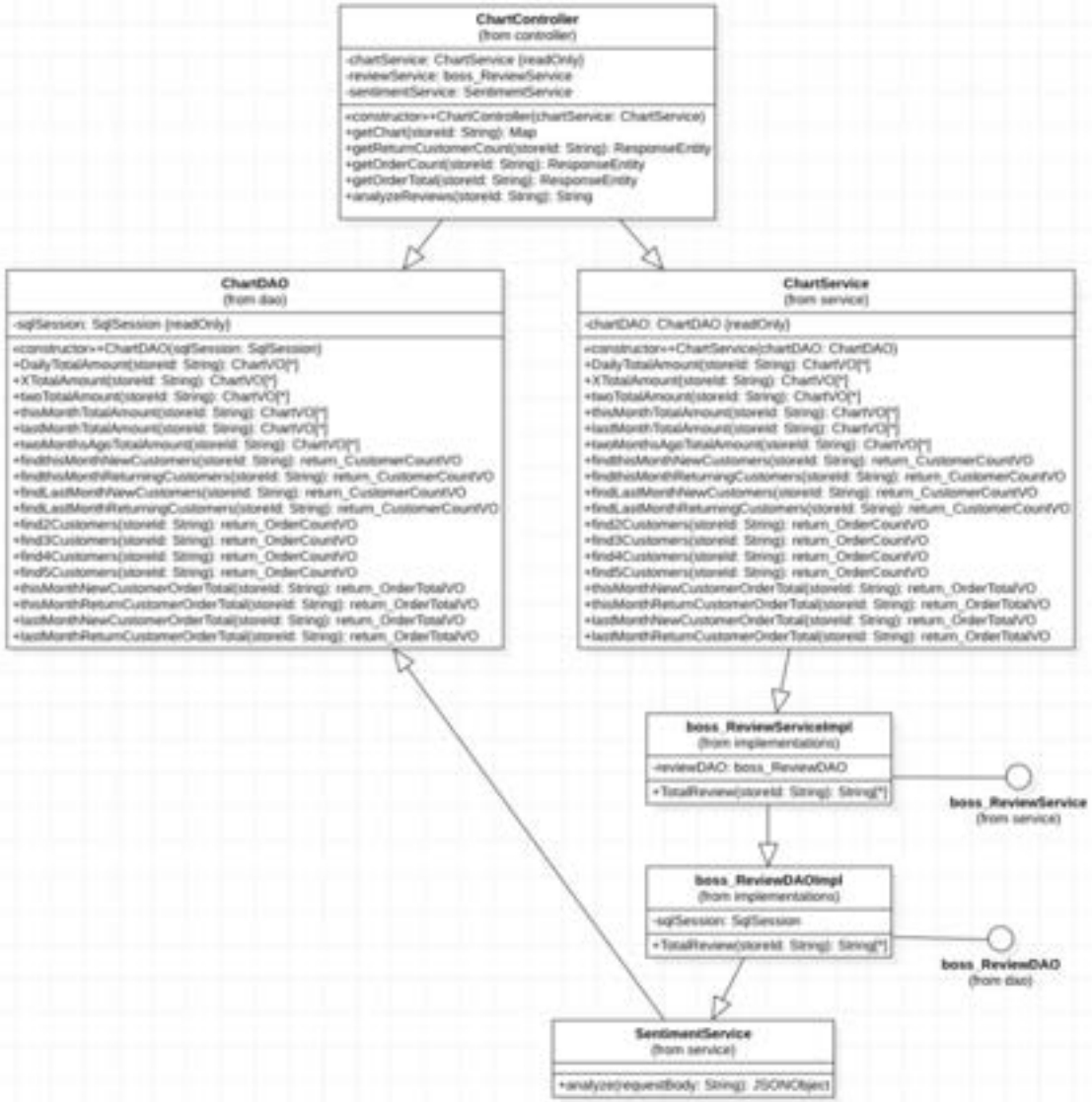
Table_LWS_03						
테이블ID		com		작성자	이후성	
테이블설명		댓글				
번호	칼럼ID	형식	NULL	KEY	내용	비고
1	board_id	INT	Not Null	FK	게시판번호	
2	reply_id	INT	Not Null	PK	댓글 아이디	auto increment
3	writer	VARCHAR	Not Null		댓글작성자	
4	content	VARCHAR	Not Null		댓글내용	
5	regdate	DATETIME	Not Null		등록일자	디폴트값

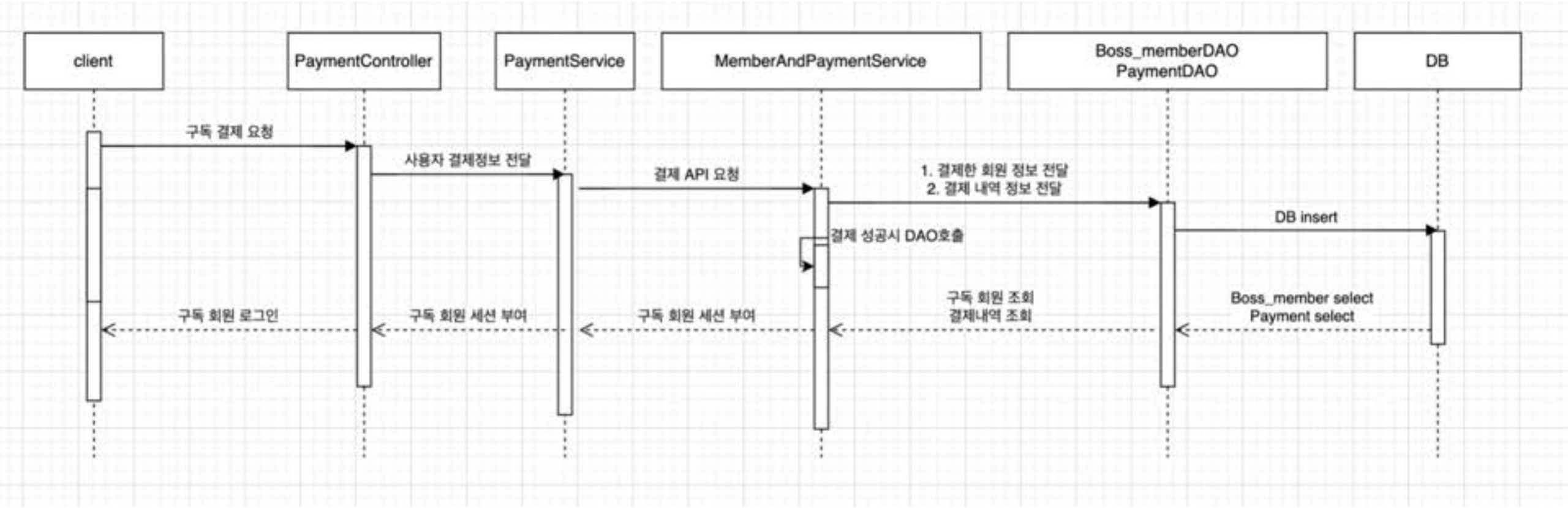
Table_LWS_04						
테이블ID		payment		작성자	이후성	
테이블설명		음식점 결제 테이블				
번호	칼럼ID	형식	NULL	KEY	내용	비고
1	id	INT	Not Null	PK	결제순번 번호	auto increment
2	store_id	VARCHAR	Not Null	FK	상점ID	
3	order_id	VARCHAR	Not Null	FK	주문자ID	
4	order_name	VARCHAR	Not Null		음식명	
5	amount	int	Not Null		가격	
6	time	DATETIME	Not Null		결제일자	

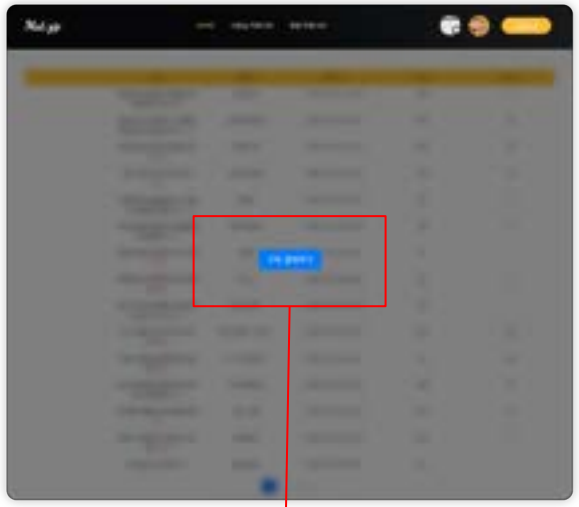
Table_LWS_05						
테이블ID		likes		작성자	이후성	
테이블설명		좋아요 수 체크				
번호	칼럼ID	형식	NULL	KEY	내용	비고
1	board_id	INT	Not Null	PK	게시판번호	
2	user_id	VARCHAR	Not Null	PK	상점ID	

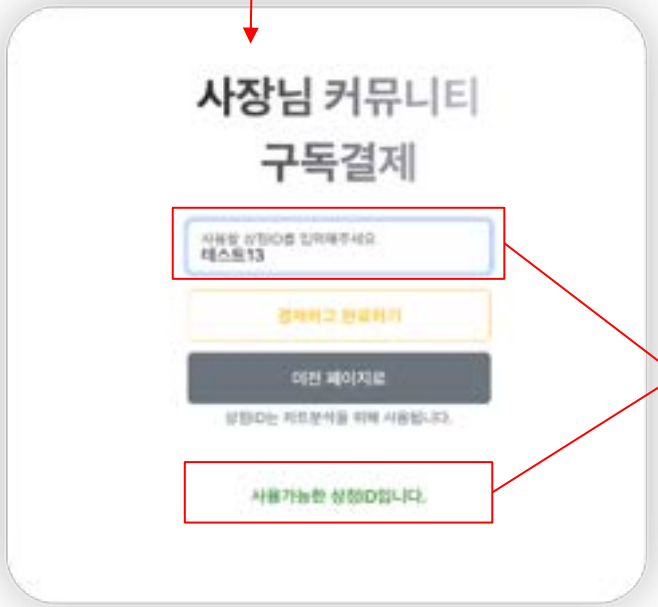

NO
Yes
NO
NO


PaymentController
(from PaymentService)
-paymentService: PaymentService
-dao: PaymentDAO
-bossdao: Boss_memberDAO
-memberAndPaymentService: MemberAndPaymentService
-session: HttpSession
+checkStoreId(storeId: String): String
+processPayment(paymentKey: String, amount: int, orderId: String, model: Model): String
+processPaymentFail(): String
+innerJoinAndInsert(store_id: String, session: HttpSession, model: Model): String
+insert(bag: PaymentVO): ResponseEntity
PaymentService
(from service)
+processPayment(paymentKey: String, amount: int, orderId: String): JsonObject
MemberAndPaymentService
(from service)
-memberDAO: Boss_memberDAO (readOnly)
+constructor<<MemberAndPaymentService(memberDAO: Boss_memberDAO, paymentService: PaymentService)
+registerAndPay(member: Boss_memberVO): void
Boss_memberDAO
(from dao)
-my: SqlSessionTemplate
+innerJoinAndInsert(bag: Boss_memberVO): void
+login(bag: Boss_memberVO): Boss_memberVO
+checkStoreId(storeId: String): int
PaymentDAO
(from dao)
-sqlSession: SqlSession
+insert(paymentVO: PaymentVO): void


ChartController
(from controller)
-chartService: ChartService (readOnly)
-reviewService: boss_ReviewService
-sentimentService: SentimentService
+constructor<<ChartController(chartService: ChartService)
+getChart(storeId: String): Map
+getReturnCustomerCount(storeId: String): ResponseEntity
+getOrderCount(storeId: String): ResponseEntity
+getOrderTotal(storeId: String): ResponseEntity
+analyzeReviews(storeId: String): String
ChartDAO
(from dao)
-sqlSession: SqlSession (readOnly)
+constructor<<ChartDAO(sqlSession: SqlSession)
+DailyTotalAmount(storeId: String): ChartVO[*]
+XTotalAmount(storeId: String): ChartVO[*]
+twoTotalAmount(storeId: String): ChartVO[*]
+thisMonthTotalAmount(storeId: String): ChartVO[*]
+lastMonthTotalAmount(storeId: String): ChartVO[*]
+twoMonthsAgoTotalAmount(storeId: String): ChartVO[*]
+findthisMonthNewCustomers(storeId: String): return_CustomerCountVO
+findthisMonthReturningCustomers(storeId: String): return_CustomerCountVO
+findLastMonthNewCustomers(storeId: String): return_CustomerCountVO
+findLastMonthReturningCustomers(storeId: String): return_CustomerCountVO
+find2Customers(storeId: String): return_OrderCountVO
+find3Customers(storeId: String): return_OrderCountVO
+find4Customers(storeId: String): return_OrderCountVO
+find5Customers(storeId: String): return_OrderCountVO
+thisMonthNewCustomerOrderTotal(storeId: String): return_OrderTotalVO
+thisMonthReturnCustomerOrderTotal(storeId: String): return_OrderTotalVO
+lastMonthNewCustomerOrderTotal(storeId: String): return_OrderTotalVO
+lastMonthReturnCustomerOrderTotal(storeId: String): return_OrderTotalVO
ChartService
(from service)
-chartDAO: ChartDAO (readOnly)
+constructor<<ChartService(chartDAO: ChartDAO)
+DailyTotalAmount(storeId: String): ChartVO[*]
+XTotalAmount(storeId: String): ChartVO[*]
+twoTotalAmount(storeId: String): ChartVO[*]
+thisMonthTotalAmount(storeId: String): ChartVO[*]
+lastMonthTotalAmount(storeId: String): ChartVO[*]
+twoMonthsAgoTotalAmount(storeId: String): ChartVO[*]
+findthisMonthNewCustomers(storeId: String): return_CustomerCountVO
+findthisMonthReturningCustomers(storeId: String): return_CustomerCountVO
+findLastMonthNewCustomers(storeId: String): return_CustomerCountVO
+findLastMonthReturningCustomers(storeId: String): return_CustomerCountVO
+find2Customers(storeId: String): return_OrderCountVO
+find3Customers(storeId: String): return_OrderCountVO
+find4Customers(storeId: String): return_OrderCountVO
+find5Customers(storeId: String): return_OrderCountVO
+thisMonthNewCustomerOrderTotal(storeId: String): return_OrderTotalVO
+thisMonthReturnCustomerOrderTotal(storeId: String): return_OrderTotalVO
+lastMonthNewCustomerOrderTotal(storeId: String): return_OrderTotalVO
+lastMonthReturnCustomerOrderTotal(storeId: String): return_OrderTotalVO
boss_ReviewServiceImpl
(from implementations)
-reviewDAO: boss_ReviewDAO
+TotalReview(storeId: String): String[*]
boss_ReviewService
(from service)
boss_ReviewDAOImpl
(from implementations)
-sqlSession: SqlSession
+TotalReview(storeId: String): String[*]
boss_ReviewDAO
(from dao)
SentimentService
(from service)
+analyze(requestBody: String): JSONObject


client
PaymentController
PaymentService
MemberAndPaymentService
Boss_memberDAO
PaymentDAO
DB
구독 결제 요청
사용자 결제정보 전달
결제 API 요청
1. 결제한 회원 정보 전달
2. 결제 내역 정보 전달
DB insert
결제 성공시 DAO호출
구독 회원 로그인
구독 회원 세션 부여
구독 회원 세션 부여
구독 회원 조회
결제내역 조회
Boss_member select
Payment select





ID 중복체크

```
//결제전 중복체크
@PostMapping("/checkStoreId")
@ResponseBody
public String checkStoreId(@RequestParam String storeId) {
    int count = bossdao.checkStoreId(storeId);
    if (count > 0) {
        return "1"; // "중복"
    } else {
        return "0"; // "사용 가능"
    }
}
```

Controller

```
public int checkStoreId(String storeId) {
    return my.selectOne("boss.checkStoreId", storeId);
}
```

DAO

```
function checkDuplication() {
  var storeId = $("input[name='store_id']").val();
  $("#DupliCheckmessage").text(""); // 함수가 실행될때마다 메시지 초기화
  $("#DupliCheckmessageS").text("");
  $.ajax({
    type: "POST",
    url: "checkStoreId",
    data: {storeId: storeId},
    success: function(response) {
      if (response == 1) {
        $("#DupliCheckmessage").text("중복된 상점ID입니다.");
        isDuplicated = true;
      } else {
        $("#DupliCheckmessageS").text("사용가능한 상점ID입니다.");
        isDuplicated = false;
      }
    }
  });
}
```

중복체크 메시지 표시

입력시마다
중복체크실행

```
$(document).ready(function() {
  $("input[name='store_id']").keyup(function() {
    // 사용자가 키를 입력할때 마다 중복 체크를 수행합니다.(keyup이벤트)
    checkDuplication();
  });

  // 클릭 이벤트 핸들러를 추가 -> 결제하고완료하기 버튼
  $("#payment").click(function() {
     requestPayment();
  });
});
```

API 요청


```
var clientKey = 'test_ck_...';
var tossPayments = TossPayments('test_ck_...');
var isDuplicated = false; // ID 중복 여부를 저장하는 변수

function generateOrderId() {
  // 영문 대소문자, 숫자, 특수문자 (-, _)로 이루어진 문자열
  const characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_";

  let orderId = "";

  // 길이가 10인 주문 ID 생성
  for (let i = 0; i < 10; i++) {
    // characters 문자열에서 무작위로 문자 선택
    const randomIndex = Math.floor(Math.random() * characters.length);
    // 선택된 문자를 주문 ID에 추가
    orderId += characters.charAt(randomIndex);
  }

  return orderId;
}
```

API 요청 ID 랜덤생성

```
if (session.getAttribute("user_id") != null) {
    var userId = '<%= session.getAttribute("user_id") %>';
    var orderId = generateOrderId();
    var storeId = $("input[name='store_id']").val();
    // storeId 값을 세션에 저장
    sessionStorage.setItem("storeId", storeId);
    console.log("생성된 orderId:", orderId);
    tossPayments.requestPayment('카드', {
        amount: 9900,
        orderId: orderId,
        orderName: '사장님커뮤니티 구독결제',
        customerName: userId,
        successUrl: 'http://localhost:8887/zip/boss/tossSuccess',
        failUrl: 'http://localhost:8887/zip/boss/tossFail'
    });
} else {
    // 세션에 user_id가 없는 경우에 대한 처리
    console.log("세션에 user_id가 없습니다.");
    // 적절한 대체 처리 방식을 추가하세요.
}
```

결제 API요청

결제
응답받기

```
<% if (session.getAttribute("user_id") != null) { %>
    var userId = '<%= session.getAttribute("user_id") %>';
    var orderId = generateOrderId();
    var storeId = $("input[name='store_id']").val();
    sessionStorage.setItem("storeId", storeId);  // storeId 값을 세션에 저장
    console.log("생성된 orderId:", orderId);
    tossPayments.requestPayment('카드', {
        amount: 2900,
        orderId: orderId,
        orderName:'사장님커뮤니티 구독결제',
        customerName: userId,
        successUrl: 'http://localhost:8887/zip/boss/tossSuccess',
        failUrl: 'http://localhost:8887/zip/boss/tossFail'
    });
<% } else { %>
    // 세션에 user_id가 없는 경우에 대한 처리
    console.log("세션에 user_id가 없습니다.");
    // 적절한 대체 처리 방식을 추가하세요.
<% } %>
}
```

```
@Service
public class PaymentService {

    public JsonObject processPayment(String paymentKey, int amount, String orderId) {
        String secretKey = [redacted]
        String encodedSecretKey = Base64.getEncoder().encodeToString((secretKey + ":").getBytes());

        try (CloseableHttpClient client = HttpClients.createDefault()) {
            HttpPost httpPost = new HttpPost("https://api.tosspayments.com/v1/payments/confirm");
            httpPost.setHeader("Authorization", "Basic " + encodedSecretKey);
            httpPost.setHeader("Content-Type", "application/json");
            String json = "{\"paymentKey\":\"" + paymentKey + "\",\"amount\":" + amount + ",\"orderId\":\"" + orderId + "\"}";
            StringEntity entity = new StringEntity(json);
            httpPost.setEntity(entity);
            HttpResponse response = client.execute(httpPost);

            // 결제 처리 성공 시 JsonObject를 반환
            if (response.getStatusLine().getStatusCode() == 200) {
                HttpEntity responseEntity = response.getEntity();
                String responseBody = EntityUtils.toString(responseEntity);
                return JsonParser.parseString(responseBody).getAsJsonObject();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // 결제 처리 실패 시 null을 반환
        return null;
    }
```

성공데이터 Json파싱

```
//결제하기 성공 and 실패
@GetMapping("/tossSuccess")
public String processPayment(@RequestParam("paymentKey") String paymentKey, @RequestParam("amount") int amount,
        @RequestParam("orderId") String orderId, Model model) {
    // 결제를 처리하고 그 결과를 받습니다.
    JsonObject paymentResponse = paymentService.processPayment(paymentKey, amount, orderId);

    // 결제가 성공했다면, success 페이지로 리다이렉트합니다.
    if (paymentResponse != null) {
        // 결제 응답 정보를 model에 추가합니다.
        model.addAttribute("paymentResponse", paymentResponse);
        return "/boss/tossSuccess";
    }
    // 결제가 실패했다면, fail 페이지로 리다이렉트합니다.
    else {
        return "/boss/tossFail";
    }
}

@GetMapping("/tossFail")
public String processPaymentFail() {
    // 결제 실패 처리 로직
    return "/boss/tossFail";
}
```

결제응답정보
성공페이지로 리턴

결제응답정보 DB Insert

회원등록

```
<!-- 결제성공내역 결제내역테이블로 전달 시작 -->
<script>
    $(document).ready(function() {
        var paymentVO = {
            "orderId": $("#orderId").text(),
            "amount": $("#totalAmount").text(),
            "orderName": $("#orderName").text()
        };

        $.ajax({
            type: "POST",
            url: "payment", // 
            contentType: "application/json",
            data: JSON.stringify(paymentVO),
            success: function() {
                $("#alertS").show();
            },
            error: function() {
                $("#alertF").show();
            }
        });
    });
    /* 결제성공내역 결제내역테이블로 전달 종료 */

    /* 사장회원등록 시작 */
    $(document).ready(function() {
        // 결제 성공 후 AJAX 요청
        if (sessionStorage.getItem("storeId")) {
            $.ajax({
                url: "innerJoinAndInsert",
                method: "POST",
                data: { store_id: sessionStorage.getItem("storeId") } // 세션에서 storeId 값을 읽어서 AJAX 요청에 포함
            }).done(function() {
                console.log("innerJoinAndInsert 성공");
                sessionStorage.removeItem("storeId"); // 요청이 완료된 후 storeId 값을 세션에서 제거
            }).fail(function() {
                console.log("innerJoinAndInsert 실패");
            });
        }
    });
```

결제응답정보 DB insert

```
//결제내역 DB등록
@PostMapping("/payment")
public ResponseEntity<String> insert(@RequestBody PaymentVO bag) {
    System.out.println("insert 요청됨.");
    System.out.println(bag);

    String orderName = bag.getOrderName();
    System.out.println("Order Name: " + orderName);
    String[] parts = orderName.split(" ", 2);
    System.out.println("Parts[0]: " + parts[0]);
    System.out.println("Parts[1]: " + parts[1]);
    bag.setStoreId(parts[0]);  // storeId에 parts[0] 설정
    bag.setOrderName(parts[1]);  // orderName에 parts[1] 설정

    String userId = (String) session.getAttribute("user_id"); // 세
    bag.setOrderId(userId);
    dao.insert(bag);
    return ResponseEntity.ok().body("결제 정보가 성공적으로 전송되
}
```

회원 등록

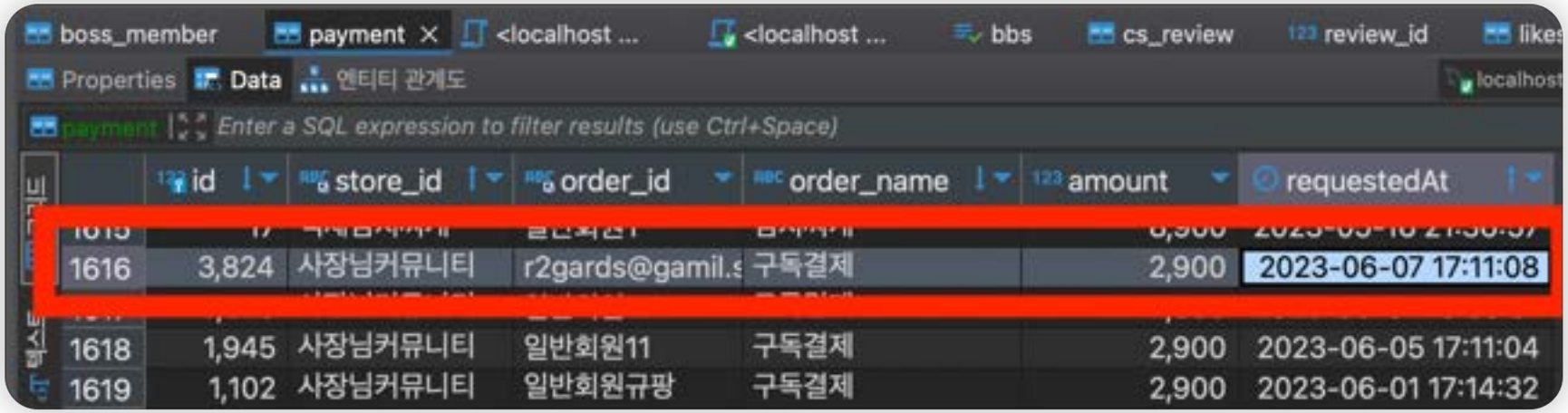
```
//사장등록
@PostMapping("/innerJoinAndInsert")
public String innerJoinAndInsert(@RequestParam String store_id, HttpSession session, Model model) {
    String user_id = (String) session.getAttribute("user_id");
    System.out.println(user_id);
    Boss_memberVO member = new Boss_memberVO();
    member.setUser_id(user_id);
    member.setStore_id(store_id);

    int count = bossdao.checkStoreId(store_id);
    if (count > 0) {
        model.addAttribute("error", "중복된 상점 ID입니다.");
        return "1";
    }

    memberAndPaymentService.registerAndPay(member);

    //성공시 사장세션부여
    String boss_id = member.getUser_id();
    session.setAttribute("boss_id", boss_id);

    return "/boss/tossSuccess";
}
```

결제실패시 트랜잭션처리

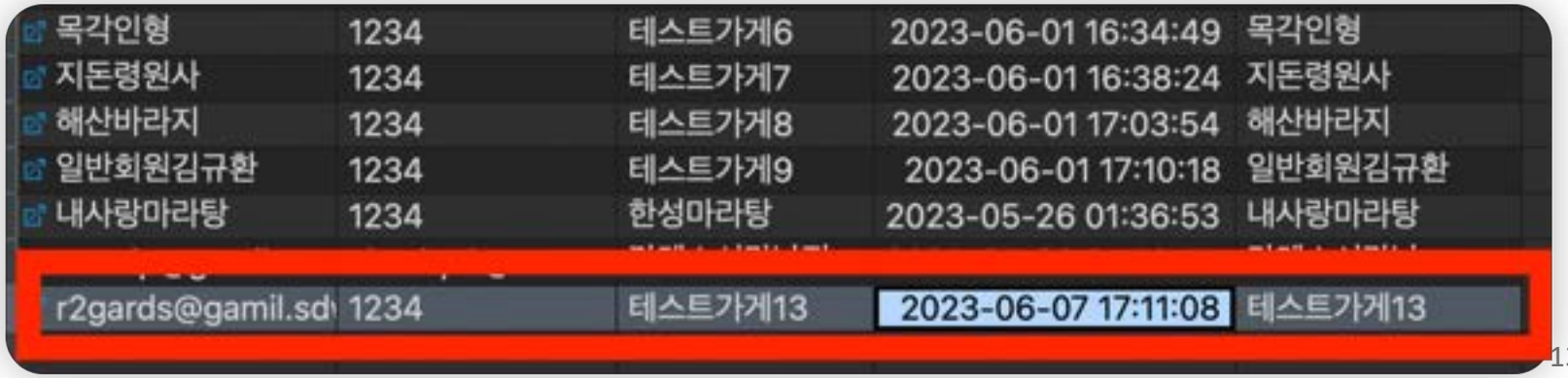
```
@Transactional
public void regis[illegible]mberVO member) {
    // 회원 정보 저장
    memberDAO.innerJoinAndInsert(member);
    System.out.println(member);
}
```

토스서버

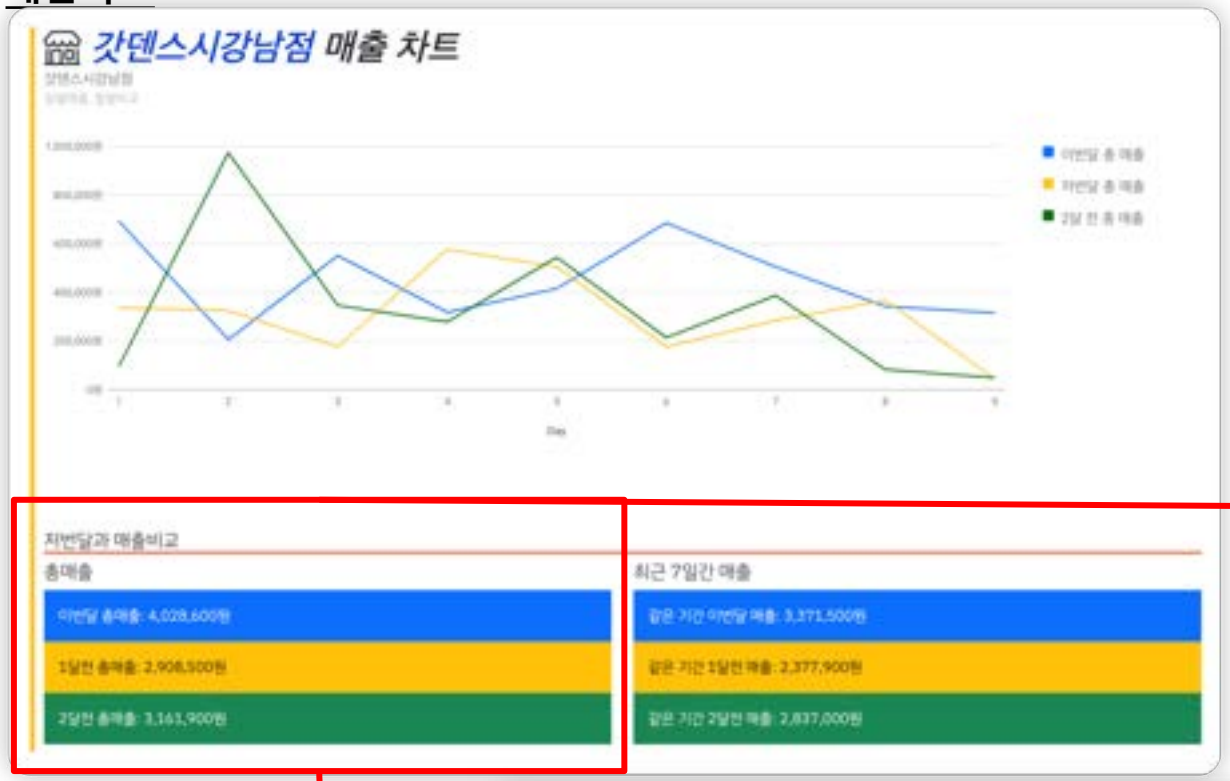
결제DB



회원등록



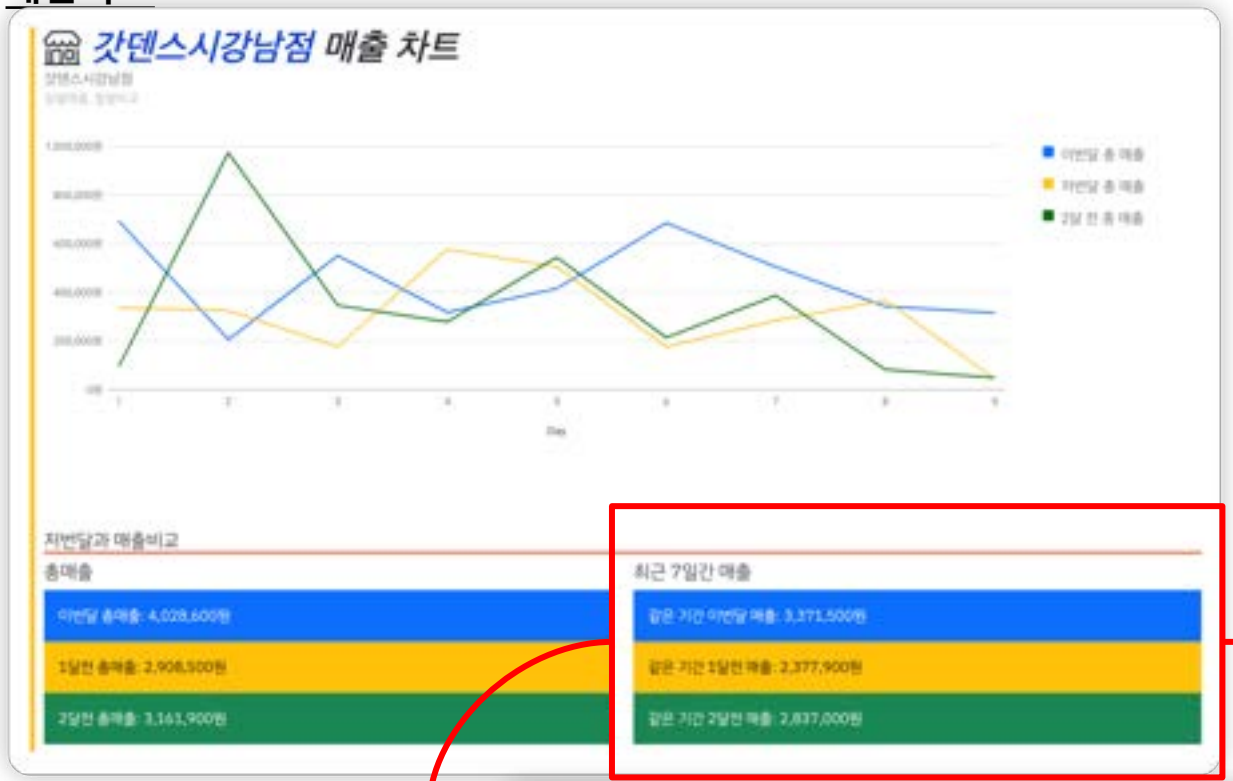
매출장부 1 -
매출차트




```
<!--이번달매출: 현재날짜 기준 일치하는 달에 총 매출데이터 -->
<select id="thisMonthTotalAmount" resultType="ChartVO">
    SELECT
    DATE(requestedAt) AS date,
    SUM(amount) AS total_amount
    FROM
    payment
    WHERE
    MONTH(requestedAt) = MONTH(CURDATE())
    AND YEAR(requestedAt) = YEAR(CURDATE())
    AND store_id = #{storeId}
    GROUP BY DATE(requestedAt)
    ORDER BY DATE(requestedAt);
</select>
<!-- 저번달 매출: 현재날짜 기준 이전달에 총 매출데이터 -->
<select id="lastMonthTotalAmount" resultType="ChartVO">
    SELECT
    DATE(requestedAt) AS date,
    SUM(amount) AS total_amount
    FROM
    payment
    WHERE
    MONTH(requestedAt) = MONTH(DATE_SUB(CURDATE(), INTERVAL 1 MONTH))
    AND YEAR(requestedAt) = YEAR(DATE_SUB(CURDATE(), INTERVAL 1 MONTH))
    AND store_id = #{storeId}
    GROUP BY DATE(requestedAt)
    ORDER BY DATE(requestedAt);
</select>
```

```
<!-- 2달 전 매출: 현재 날짜를 기준으로 2달 전에 해당하는 총 매출 데이터 -->
<select id="twoMonthsAgoTotalAmount" resultType="ChartVO">
    SELECT
    DATE(requestedAt) AS date,
    SUM(amount) AS total_amount
    FROM
    payment
    WHERE
    MONTH(requestedAt) = MONTH(DATE_SUB(CURDATE(), INTERVAL 2 MONTH))
    AND YEAR(requestedAt) = YEAR(DATE_SUB(CURDATE(), INTERVAL 2 MONTH))
    AND store_id = #{storeId}
    GROUP BY DATE(requestedAt)
    ORDER BY DATE(requestedAt);
</select>
```

매출장부 1 -
매출차트




```xml
<!-- 최근 7일간이 해당하면 매출데이터중에 store_id에 해당하는 값 -->
<select id="DailyTotalAmount" resultType="ChartVO">
    SELECT
    DATE(requestedAt) AS date,
    SUM(amount) AS total_amount
    FROM
    payment
    WHERE
    DATE(requestedAt) BETWEEN CURDATE() - INTERVAL 6 DAY AND CURDATE()
    AND
    store_id = #{storeId}
    GROUP BY DATE(requestedAt)
    ORDER BY DATE(requestedAt);
</select>

<!-- 이전달 7일의 데이터: 현재날짜기준 1달전의 같은 기간 데이터 -->
<select id="xTotalAmount" resultType="ChartVO">
    SELECT
    DATE(requestedAt) AS date,
    SUM(amount) AS total_amount
    FROM
    payment
    WHERE
    DATE(requestedAt)
    BETWEEN DATE_SUB(DATE_SUB(CURDATE(), INTERVAL 1 MONTH), INTERVAL 6 DAY)
    AND DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
    AND store_id = #{storeId}
    GROUP BY DATE(requestedAt)
    ORDER BY DATE(requestedAt);
```

```xml
<!-- 2달전 7일의 데이터: 현재날짜기준 1달전의 같은 기간 데이터 -->
<select id="twoTotalAmount" resultType="ChartVO">
    SELECT
    DATE(requestedAt) AS date,
    SUM(amount) AS total_amount
    FROM
    payment
    WHERE
    DATE(requestedAt)
    BETWEEN DATE_SUB(DATE_SUB(CURDATE(), INTERVAL 2 MONTH), INTERVAL 6 DAY)
    AND DATE_SUB(CURDATE(), INTERVAL 2 MONTH)
    AND store_id = #{storeId}
    GROUP BY DATE(requestedAt)
    ORDER BY DATE(requestedAt);
</select>
```

매출장부 1 -
매출차트

```
<!-- 매출 차트 -->
<script type="text/javascript">
$(document).ready(function(){
google.charts.load('current', {'packages':['line']});
google.charts.setOnLoadCallback(drawChart);

function drawChart() {
  var storeId = '<%= session.getAttribute("store_id") %>'; // 세션에서 storeId 가져오기
  var encodedStoreId = encodeURIComponent(storeId);

  $.ajax({
    url: 'chart/' + encodedStoreId,
    type: 'GET',
    dataType: 'json',
    success: function(response) {
      var data = new google.visualization.DataTable();
      data.addColumn('number', 'Day');
      data.addColumn('number', '이번달 총 매출');
      data.addColumn('number', '저번달 총 매출');
      data.addColumn('number', '2달 전 총 매출');
```

세션의 store_id를
encoding해서 ajax요청보냄

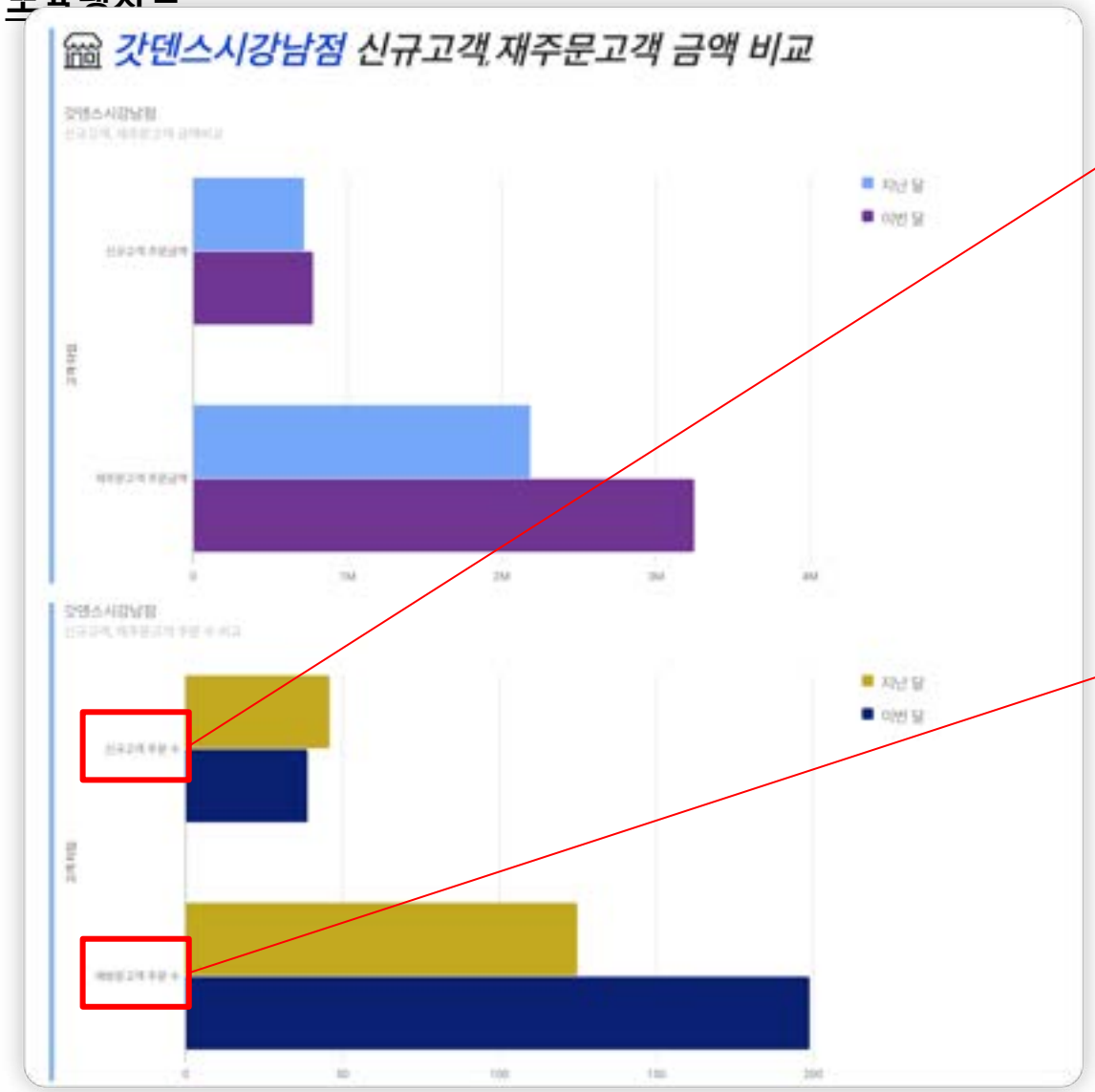
```
// HTML 태그에 매출액 설정
document.getElementById('thisMonthTotal').textContent = '이번달 총매출: ' + thisMonthTotalAmount.toLocaleString('ko-KR') + '원';
document.getElementById('lastMonthTotal').textContent = '1달전 총매출: ' + lastMonthTotalAmount.toLocaleString('ko-KR') + '원';
document.getElementById('twoMonthTotal').textContent = '2달전 총매출: ' + twoMonthTotalAmount.toLocaleString('ko-KR') + '원';
document.getElementById('thisMonthDays').textContent = '같은 기간 이번달 매출: ' + thisMonthDays.toLocaleString('ko-KR') + '원';
document.getElementById('lastMonthDays').textContent = '같은 기간 1달전 매출: ' + lastMonthDays.toLocaleString('ko-KR') + '원';
document.getElementById('twoMonthDays').textContent = '같은 기간 2달전 매출: ' + twoMonthDays.toLocaleString('ko-KR') + '원';
```

태그출력

```
// 데이터 행 추가
for (var i = 0; i < thisMonthData.length; i++) {
  var day = i + 1;
  var thisMonthAmount = thisMonthData[i].total_amount;
  var lastMonthAmount = lastMonthData[i].total_amount;
  var twoMonthsAgoAmount = twoMonthsAgoData[i].total_amount;
  data.addRow([day, thisMonthAmount, lastMonthAmount, twoMonthsAgoAmount]);
}

var options = {
      chart: {
        title: '<%= session.getAttribute("store_id") %>',
        subtitle: '일별매출, 월별비교'
      },
      width: 1200,
      height: 400,
      hAxis: {
        //날짜 포맷설정 format:'D' 원하는 문자열 삽입

      },
      vAxis: {
        format: '#,###원'  // Y축 단위변경
```

차트출력

매출장부 2 -
또슐랭차트




```
<!-- 재방문 차트 신규고객과 재방문 고객의 숫자를 찾는 쿼리 -->

<!--이번달 신규고객 수 -->
<select id="findThisMonthNewCustomers" resultType="return_CustomerCountVO">
  SELECT COUNT(*) as new_customers
  FROM (
    SELECT order_id
    FROM payment
    WHERE store_id = #{storeId}
    GROUP BY order_id
    HAVING MIN(requestedAt) BETWEEN DATE_FORMAT(NOW() , '%Y-%m-01') AND LAST_DAY(NOW())
  ) as subquery
</select>
```

now()를 연도,월,01일로 변환, 즉 이번달의 첫날

괄호안을 하위쿼리 처리하고 레코드 카운트

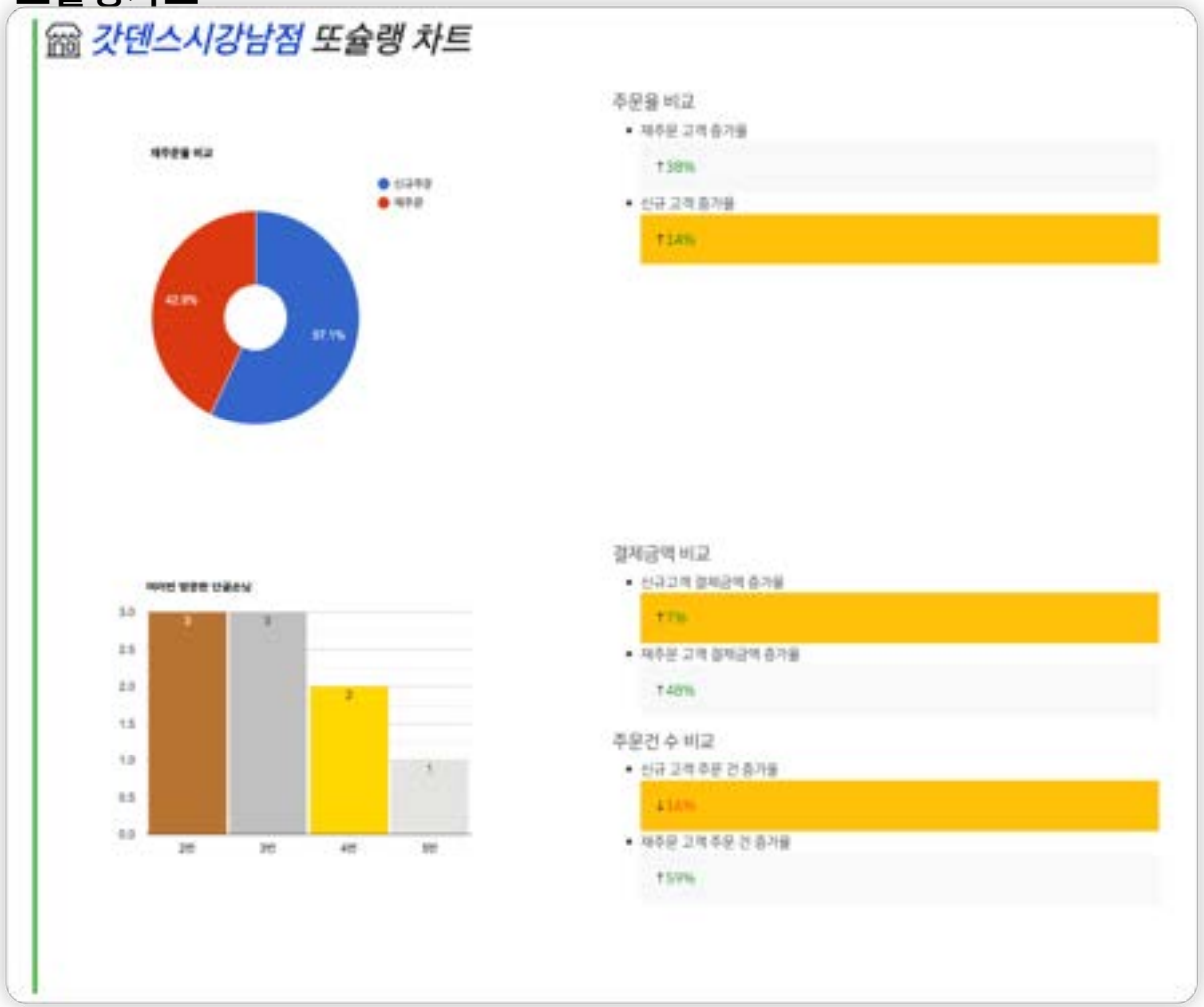
이번달의 마지막날

```
<!-- 이번달 재방문고객 수 -->
<select id="findThisMonthReturningCustomers" resultType="return_CustomerCountVO">
  SELECT COUNT(*) as returning_customers
  FROM (
    SELECT order_id
    FROM payment
    WHERE store_id = #{storeId}
    AND requestedAt BETWEEN DATE_FORMAT(NOW() , '%Y-%m-01') AND LAST_DAY(NOW())
    GROUP BY order_id
    HAVING COUNT(*) >= 2
  ) as subquery
</select>
```

order_id레코드의 카운트 2번이상인경우

1달전 데이터는 INTERVAL 1 MONTH 를 쿼리에 추가하여 구함

매출장부 2 -
또슐랭차트




```
<!-- 이번달 신규 고객 주문 금액 계산 -->
<select id="thisMonthNewCustomerOrderTotal" resultType="return_OrderTotalVO">
  SELECT COUNT(order_id) as newOrdersThisMonth, SUM(amount) as newOrderTotalThisMonth
  FROM payment
  WHERE store_id = #{storeId}
  AND MONTH(requestedAt) = MONTH(CURRENT_DATE())
  AND YEAR(requestedAt) = YEAR(CURRENT_DATE())
  AND order_id NOT IN (
    SELECT order_id
    FROM payment
    WHERE store_id = #{storeId}
    AND ((YEAR(requestedAt) &lt; YEAR(CURRENT_DATE())) OR (YEAR(requestedAt) = YEAR(CURRENT_DATE())
    AND MONTH(requestedAt) &lt; MONTH(CURRENT_DATE())))
  )
</select>
```

이번달에 요청된 주문중에서

과거에 주문한 이력이 없는 주문

올해 이전에 요청된 주문 이거나

올해이지만 이번달 이전에 요청된 주문

부등호 < 표시, xml 에러문제로 < 로 변환했음

재방문 고객 주문금액 계산은 order_id NOT IN -> order_id IN

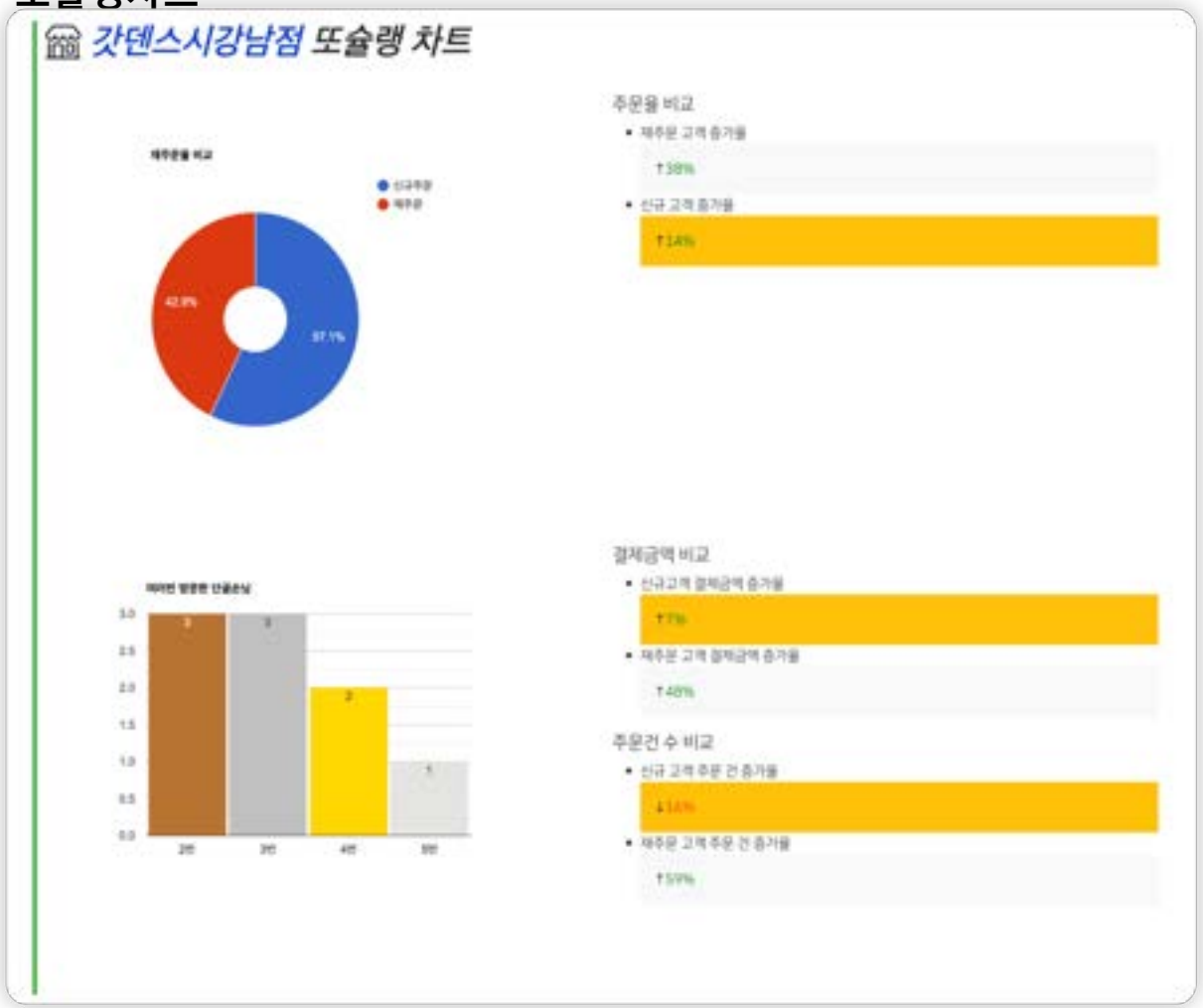
```
<!-- 여러번 주문한(2-5) 고객님들 수 찾기 -->
<!-- 2회 주문한 고객 찾기 -->
<select id="find2Customers" resultType="return_OrderCountVO">
    SELECT COUNT(order_id) as twoOrderCustomers
    FROM (
        SELECT order_id
        FROM payment
        WHERE store_id = #{storeId}
        GROUP BY order_id
        HAVING COUNT(order_id) = 2
    ) as subquery
</select>
```

order_id가 2인 레코드 모음

괄호안을 하위쿼리처리

숫자 카운트를 바꿔서
3회,4회,5회 도 구함

매출장부 2 -
또슐랭차트




```
$(document).ready(function(){
// 신규 고객 증가율과 재방문 고객 증가율 계산 함수
function calculateGrowth(thisMonth, lastMonth) {
    var growth;
    if (lastMonth === 0) {
        growth = thisMonth > 0 ? 100 : 0;
    } else {
        growth = (thisMonth - lastMonth) / lastMonth * 100;
    }
    return Math.floor(growth); // 소수점 이하를 잘라냄.
}
```

지난 월의 데이터가 0 이면 100퍼처리

(현재 월 - 지난 월) / 지난 월 * 100

```
function formatGrowth(growth, elementId) {
    var element = document.getElementById(elementId);
    if(growth >= 0) {
        element.innerHTML = '↑<span style="color:green; font-weight:bold;">'
        + growth + '%</span>';
    } else {
        element.innerHTML = '↓<span style="color:red; font-weight:bold;">'
        + Math.abs(growth) + '%</span>';
    }
}

// Google 차트 로딩
```

증가율이 양수면 ↑ 표시

증가율이 음수면 ↓ 표시

증가율의 절대값 구하기

매출장부 3 -
리뷰감정분석차트


```java
// 리뷰 감정분석 차트
@GetMapping(value = "/analyze/{storeId}", produces = "application/json; charset=UTF-8")
@ResponseBody
public String analyzeReviews(@PathVariable String storeId) {
    //세션에서 받아온 storeId 디코딩
    try {
        storeId = URLDecoder.decode(storeId, StandardCharsets.UTF_8.name());
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    System.out.println(storeId);
    // 리뷰 데이터 받아오기
    List<String> reviewContents = reviewService.TotalReview(storeId);

    // json형태로 바꿔서 api service요청보내기
    JSONArray results = new JSONArray();

    for (String reviewContent : reviewContents) {
        JSONObject requestBody = new JSONObject();
        requestBody.put("content", reviewContent);
        JSONObject analysisResult = sentimentService.analyze(requestBody.toString());
        results.put(new JSONObject(analysisResult.toString()));
    }

    return results.toString();
}
```

store_id에 해당하는
리뷰데이터 받아오기

각 리뷰에 대한 감정분석 결과를 JsonArray로 만들어 문자열형태로 리턴

```java
@Service
public class boss_ReviewServiceImpl implements boss_ReviewService {
    @Autowired
    private boss_ReviewDAO reviewDAO;

    @Override
    public List<String> TotalReview(String storeId) {
        return reviewDAO.TotalReview(storeId);
    }
```

매출장부 3 -
리뷰감정분석차트

```
// 리뷰 감정분석 차트
@GetMapping(value = "/analyze/{storeId}", produces = "application/json; charset=UTF-8")
@ResponseBody
public String analyzeReviews(@PathVariable String storeId) {
    //세션에서 받아온 storeId 디코딩
    try {
        storeId = URLDecoder.decode(storeId, StandardCharsets.UTF_8.name());
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    System.out.println(storeId);
    // 리뷰 데이터 받아오기
    List<String> reviewContents = reviewService.TotalReview(storeId);

    // json형태로 바꿔서 api service요청보내기
    JSONArray results = new JSONArray();

    for (String reviewContent : reviewContents) {
        JSONObject requestBody = new JSONObject();
        requestBody.put("content", reviewContent);
        JSONObject analysisResult = sentimentService.analyze(requestBody.toString());
        results.put(new JSONObject(analysisResult.toString()));
    }
                                각 리뷰에 대한 감정분석 결과를 JsonArray로 만들어 문자열형태로 리턴
    return results.toString();
}
```

store_id에 해당하는
리뷰데이터 받아오기

감정분석 api호출

```
@Service
public class boss_ReviewServiceImpl implements boss_ReviewService {
    @Autowired
    private boss_ReviewDAO reviewDAO;

    @Override
    public List<String> TotalReview(String storeId) {
        return reviewDAO.TotalReview(storeId);
    }
}
```

```
@Service
public class SentimentService {
    public JSONObject analyze(String requestBody) {
        RestTemplate restTemplate = new RestTemplate();   REST API호출
        HttpHeaders headers = new HttpHeaders();
                                                          HTTP 헤더설정-API키정보 문자열 인코딩정보
        headers.set("X-NCP-APIGW-API-KEY-ID", "3uzmdwnb3e");
        headers.set("X-NCP-APIGW-API-KEY", "[redacted]");
        headers.setContentType(new MediaType("application", "json", StandardCharsets.UTF_8)); //인코딩 형태는 고정.

        HttpEntity<String> entity = new HttpEntity<>(requestBody, headers);

        String url = "https://naveropenapi.apigw.ntruss.com/sentiment-analysis/v1/analyze";
        ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.POST, entity, String.class);
        String responseBody = response.getBody();
                                                          RestTemplate의 exchange메소드 사용-> post요청

        // 클라이언트에 응답 보내기전 UTF-8로 리턴해주기
        try {
            responseBody = new String(responseBody.getBytes("ISO-8859-1"), "UTF-8");
        } catch (Exception e) {
            e.printStackTrace();
        }

        return new JSONObject(responseBody);   json형태로 변환
    }
}
```


21

매출장부 3 -
리뷰감정분석차트

```
// 통계 변수 초기화
var totalCount = response.length;
var positiveCount = 0;
var negativeCount = 0;
var neutralCount = 0;

    // 통계 카운트 계산
    if (review.document.sentiment === "positive") {
        positiveCount++;
    } else if (review.document.sentiment === "negative") {
        negativeCount++;
    } else {
        neutralCount++;
    }
}

var positivePercentage = (positiveCount / totalCount) * 100;
var negativePercentage = (negativeCount / totalCount) * 100;
var neutralPercentage = (neutralCount / totalCount) * 100;
```

```
// 각각의 리스트를 정렬하기
positiveReviews.sort(function(a, b) {
    return b.document.confidence.positive - a.document.confidence.positive;
});
negativeReviews.sort(function(a, b) {
    return b.document.confidence.negative - a.document.confidence.negative;
});

// 최대 5개의 긍정 리뷰와 부정 리뷰 선택하기
var maxPositiveReviews = positiveReviews.slice(0, 5);
var maxNegativeReviews = negativeReviews.slice(0, 5);

// 최대 긍정 점수가 높은 리뷰 표시
var positiveReviewElement = $("#positiveReviews");
positiveReviewElement.empty(); // 로딩될때마다 이전데이터 삭제되게 함
for (var i = 0; i < maxPositiveReviews.length; i++) {
    var review = maxPositiveReviews[i];
    var reviewElement = $("<p>").text("리뷰 내용: " + decodeURIComponent(review.sentences[0].content)
            + ", 감정 분석 결과: " + review.document.sentiment);
    positiveReviewElement.append(reviewElement);
}

// 최대 부정 점수가 높은 리뷰 표시
var negativeReviewElement = $("#negativeReviews");
negativeReviewElement.empty(); // 로딩될때마다 이전데이터 삭제되게 함
for (var i = 0; i < maxNegativeReviews.length; i++) {
    var review = maxNegativeReviews[i];
    var reviewElement = $("<p>").text("리뷰 내용: " + decodeURIComponent(review.sentences[0].content)
            + ", 감정 분석 결과: " + review.document.sentiment);
    negativeReviewElement.append(reviewElement);
}
```

API 응답으로 돌아오는 confidence 필드안에 감정분석 결과값이 있음

긍정점수와,부정점수의 신뢰도 점수를 내림차순으로 정렬

[멀티잇 백엔드 개발자 취업캠프 9회차] 최종 포트폴리오 - 2조 (맛.java)

E.O.D

Team Mat. java

MAT. ZIP