REDUCE

REUSE RECYCLE

REFIT 의류 쇼핑몰

1조 REFIT

GitHub: https://github.com/inhan99/2025-hackathon-teamproject

고인한

김찬영

고윤호

송승찬

REDUCE

REUSE RECYCLE

목 차

REDUCE

REUSE RECYCLE

프로젝트 배경

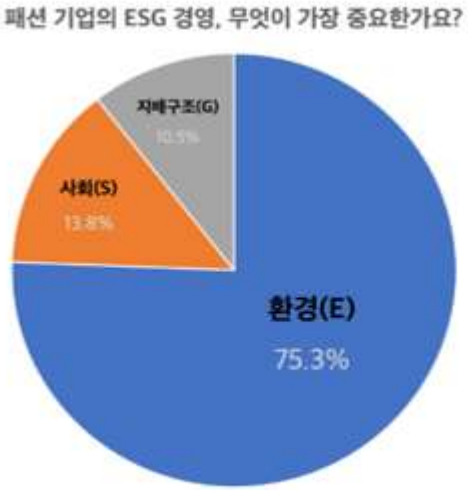
한국섬유패션정책연구원 조사에
의하면 '패션 기업의 ESG 경영 중 가장 중요한 것이 무엇이냐'
는 질문에 응답자의 75%가 환경(E)를 택

**누군가는 옷을 고르고,
누군가는 옷을 기다립니다.**

**같은 쇼핑몰에서,
모두가 따뜻한 선택을 할 수 있다면
어떨까요?**

Re:Fit은 기부와 나눔, 그리고 AI 기술이 결합된 새로운 형태의 패션 플랫폼입니다.

**기부로 선한 영향력을 만들고, AI로 자동화된 시스템으로 스마트하게 쇼핑하는 경험.
Re:Fit이 제안하는 새로운 패션 생태계입니다.**



국내 패션 기업 ESG 경영 기업 및 소비자 설문조사 (차트 재구성=김호정 기자)

출처: 2050탄소 중립위원회

쏟아지는 의류 폐기물, 이대로 괜찮은가

현재 우리나라에서 매년 버려지는 옷의 양은 심각한 수준입니다. 환경부에 의하면 2018년 6만 6000톤에 이르렀던 의류 폐기물은 코로나19 이후 2022년 11만 톤까지 급증했고, 올해 들어서는 지난 5월까지 10만 6000톤의 의류가 폐기되었습니다.

프로젝트 구현 기능 & 팀 역할

구현 기능	팀원	담당 업무		세부설명
		Front	Back	
로그인	고인한 (팀장)	API를 통한 소셜 로그인 구현, react-cookie를 활용한 JWT 저장, Redux를 활용한 전역 인증 상태 관리	Spring Security 기반 회원가입, 로그인, 회원 정보 수정 로직 처리, Restful 구조에 맞는 컨트롤러 구현	Kakao API를 통한 소셜 로그인을 구현 및 회원에 대한 권한을 세분화 하여 각 역할에 맞는 접근 권한 부여를 통해 보안성과 기능 명확성을 확보
AI챗봇	고윤호	사용자 입력과 챗봇 응답을 주고받는 대화형 UI를 구현	LLM 기반 응답 생성 및 DB 연동 처리 기능을 구현	**AI챗봇**: React 대화형 UI와 음성인식 기능을 구현하고, FastAPI와 OpenAI GPT를 활용한 LLM 기반 응답 생성 및 DB 연동으로 지능형 고객 상담 서비스 제공 **결제**: React 포트원 SDK 연동 결제 UI와 적립금 시스템을 구현하고, Spring Boot 기반 포트원 API 연동 및 결제 검증/취소 처리로 안전한 결제 시스템 제공
결제		사용자 결제 화면과 결제 과정을 포함한 UI를 구현	포트원 결제 시스템 연동 및 결제 요청 처리 기능을 구현	
나눔 상품	김찬영	나눔 등록 절차 및 검증 과정 페이징 처리 구현 및 나눔 상품 페이지 구현	각 기능에 해당하는 Restful구조에 맞는 컨트롤러 구현 및 구현 상태별 검수 과정 상태 구현	검수처리전 기부상품 상태를 관리자 권한을 통해 승인된상태로 DB에 업데이트 처리 나눔페이지 취약계층별 의류상품 분류 메인페이지 사용자 편의성을 위한 UI/UX개선
커뮤니티	송승찬	게시글, 댓글, 파일 처리, 이미지 업로드 등 커뮤니티에 필요한 페이징 처리 구현	각 기능에 해당하는 Restful구조에 맞는 컨트롤러 구현 및 CRUD로직 작성	커뮤니티 CRUD 구현, 이미지 다중 업로드 가능, JWT 토큰 기반 인증으로 로그인한 사용자만 게시글과 댓글 작성이 가능, 본인이 작성한 내용만 수정/삭제 가능 챗봇 장바구니 상품 담기 및 구매 가능, 사용자 체형에 맞는 상품 추천 및 Toast 처리 구현
AI챗봇				

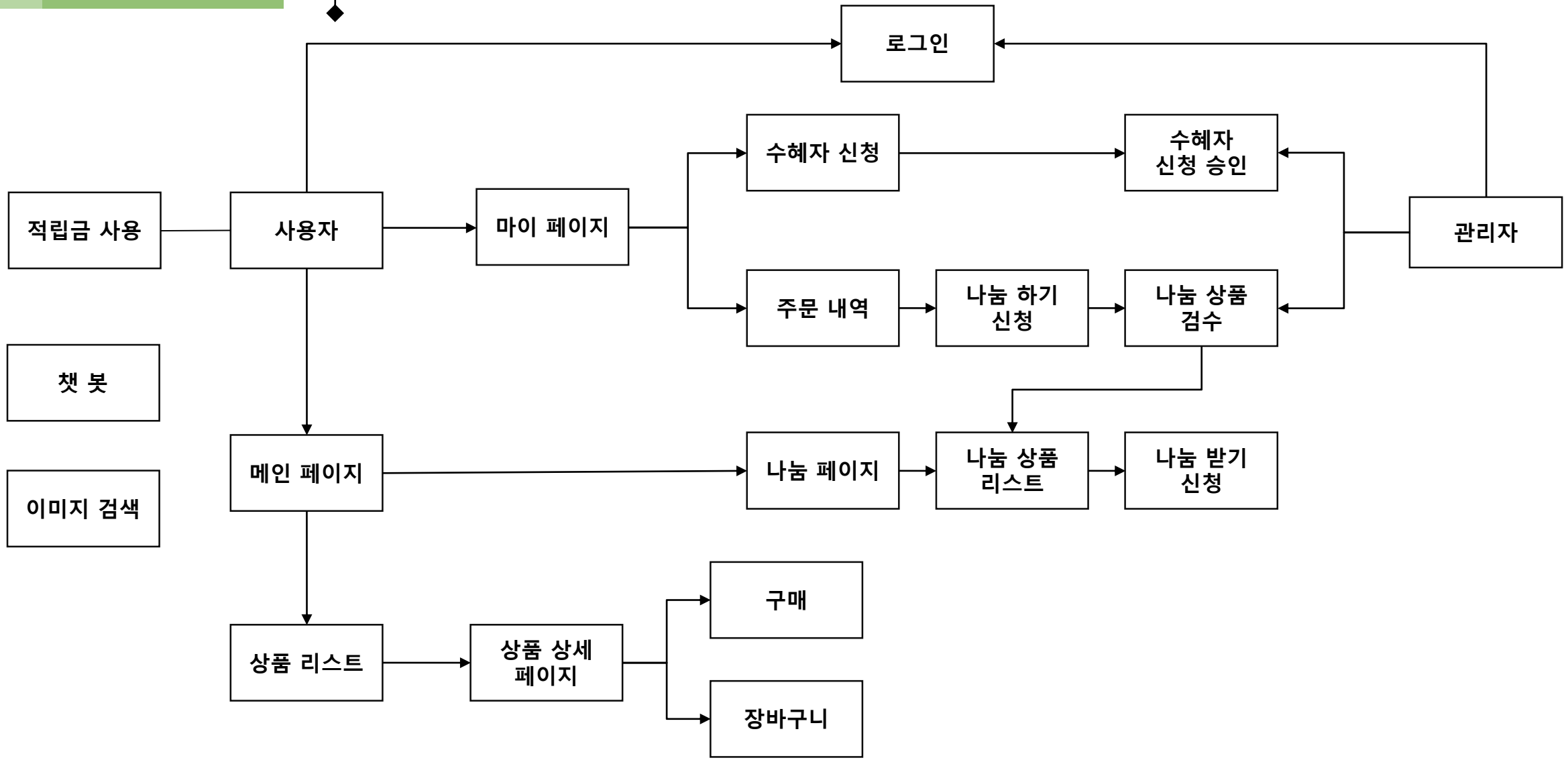
핵심기능 & 개발 도구

분야	개발 도구	상세 설명
Back-End	**JAVA 21** **Spring Boot 3** **Spring Security** **JWT** **Maria DB** **JPA**	**Spring Security** 및 **JWT**를 활용하여 **보완성 강화** JPA Repository를 이용한 쿼리메서드를 사용하여 로직을 간결하고 직관적으로 구현 **Restfull 방식**을 사용하여 **프론트엔드와 백엔드의 원활한 통신** MairaDB 를 사용하여 데이터베이스를 효율적으로 생성 및 관리
Front-End	**React18** **HTML5** **TailwindCSS** **Node.js** **JavaScript**	**React를 사용**하여 **SPA기반**으로 반응형 웹 페이지 구현 Router를 사용하여 URL파라미터의 효율적인 관리 **Redux를 통해 전역 상태 관리**를 구현하여 데이터를 효율적으로 처리 Custom Hook을 사용하여 코드의 재사용 및 편의성 증가
API	**PortOne** **Google Cloud Platform** **KakaoLogin** **KakaoMap**	카카오 API를 사용하여 소셜로그인 구현 카카오맵 API를 사용하여 주소지 검색기능 구현
Python	**Python FastAPI** **LangChain** **OpenAI GPT** **Google Cloud Vision API** **Google Cloud Speech-to-Text** **ApexCharts**	**FastAPI를 활용**하여 챗봇 서버구현 **OpenAI GPT를 활용한 챗봇 AI구현** PortOne API를 사용하여 결제시스템 구현 구글 클라우드API(이미지 검색, 음성인식) 구현

REDUCE

REUSE RECYCLE

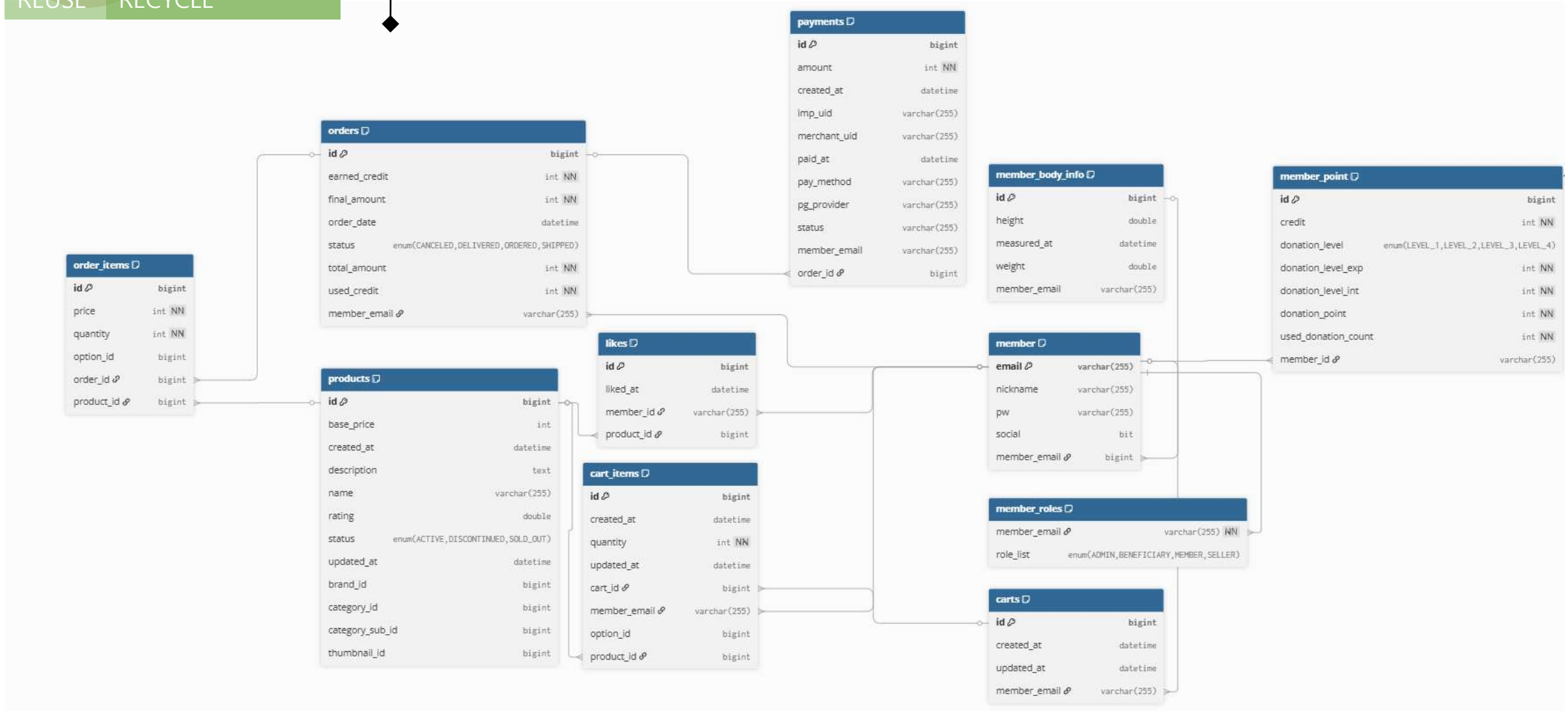
유스케이스 다이어그램



REDUCE

REUSE RECYCLE

ERD

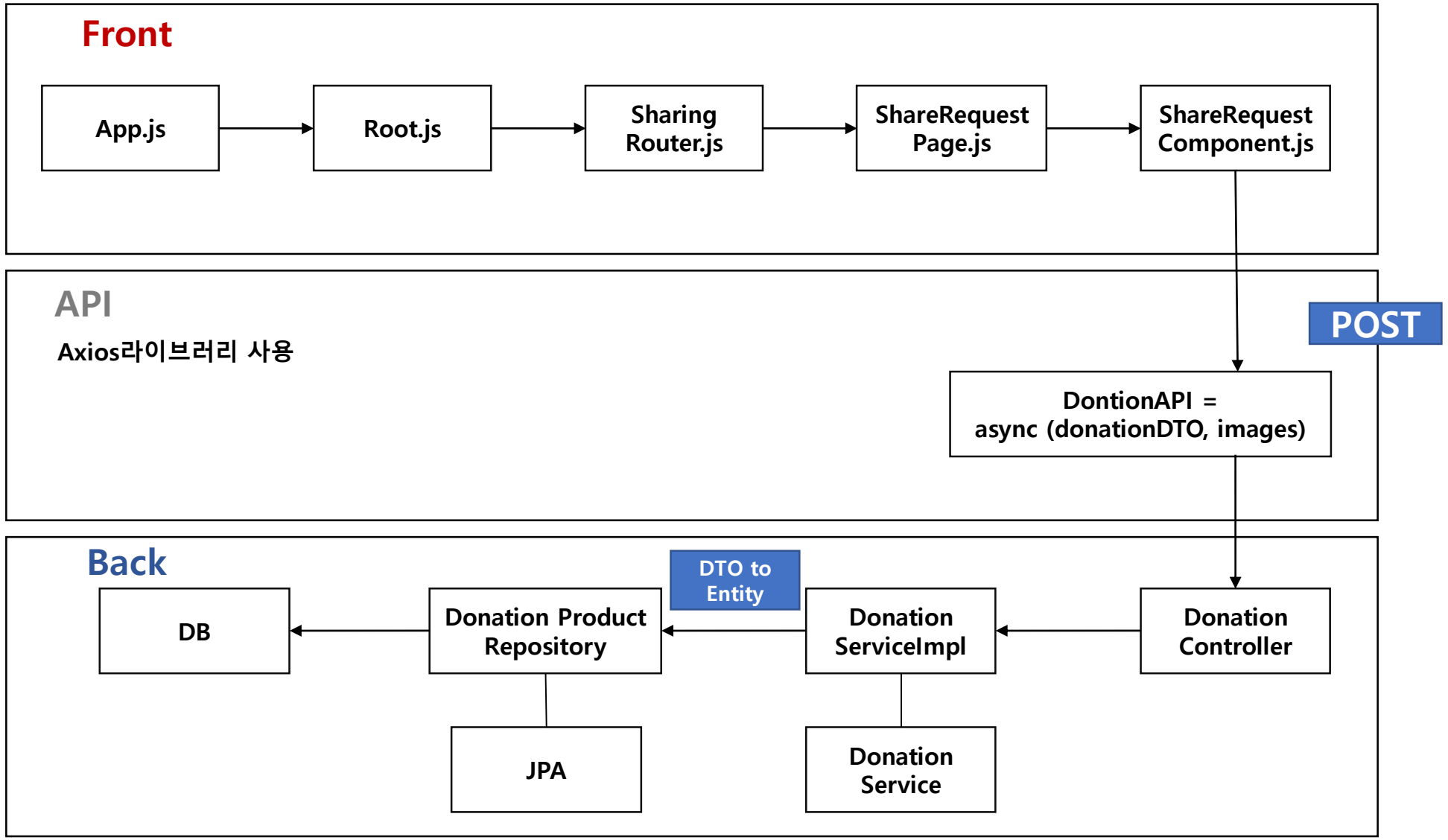


수행절차

구분	기간	활동	비고
기획	6월 30일(월) ~ 7월 4일(금)	• 아이디어 기획 • 프로젝트 핵심 객체 도출 • 개별 역할 분담	참고 사이트: 무신사, 쿠팡
설계	7월 5일(토) ~ 7월 8일(화)	• 핵심 페이지 목업 제작 • DB 도메인 설계 • 샘플 코드 작성	Figma를 이용한 페이지 목업 ERD를 이용한 핵심 객체 관계 시각화
구현	7월 9일(수) ~ 7월 20일(일)	• MVP 우선 개발 • 외부 API를 활용한 로그인, 결제수단, 위치기반 서비스 구현 • 챗봇 기반의 사용자 편의 기능 제공	Security와 상품 구매, 나눔 상품 등록 등과 같은 핵심 기능을 구현 후 외부 API를 활용해 KakaoLogin을 사용한 소셜 로그인, KakaoMap을 통한 주소지 설정, PayOne을 사용한 카카오페이, 토스페이 결제수단 구현 LangChain과 OpenAI API 기반의 음성 및 텍스트 인터페이스를 통해 사용자와의 자연스러운 상호작용 구현
최종 구현	7월 21일(월) ~7월 25일(금)	• 디버깅 및 UI/UX 개선 • 추가 기능 구현	사용자 신체정보를 통한 상품 추천 서비스 구현 편의성, 접근성, 반응성을 종합적으로 고려한 UI/UX 전면 개편

REDUCE
REUSE RECYCLE

나눔 상품 등록



REDUCE

REUSE RECYCLE

나눔 상품 등록

마이페이지 클릭 후
주문내역 클릭

나눔하기 버튼 클릭


```
// 5. 나눔 상품 등록 API
export const createDonation = async (donationDTO, images) => {
  try {
    const formData = new FormData();
    const memberCookie = getCookie("member");
    const accessToken = memberCookie?.accessToken;

    formData.append(
      "donation",
      new Blob([JSON.stringify(donationDTO)], { type: "application/json" })
    );

    if (images && images.length > 0) {
```

```
@RestController
@RequestMapping("/api/donation")
@RequiredArgsConstructor
@Log4j2
public class DonationController {

    private final DonationProductService donationProductService;
    private final JWTUtil jwtUtil;
```

```
// 4. 사용자 조회
Member donor = donationProductService.findMemberByEmail(userEmail);

// 5. 기부 저장 서비스 호출
donationProductService.saveDonation(donationDTO, donor, images);
```

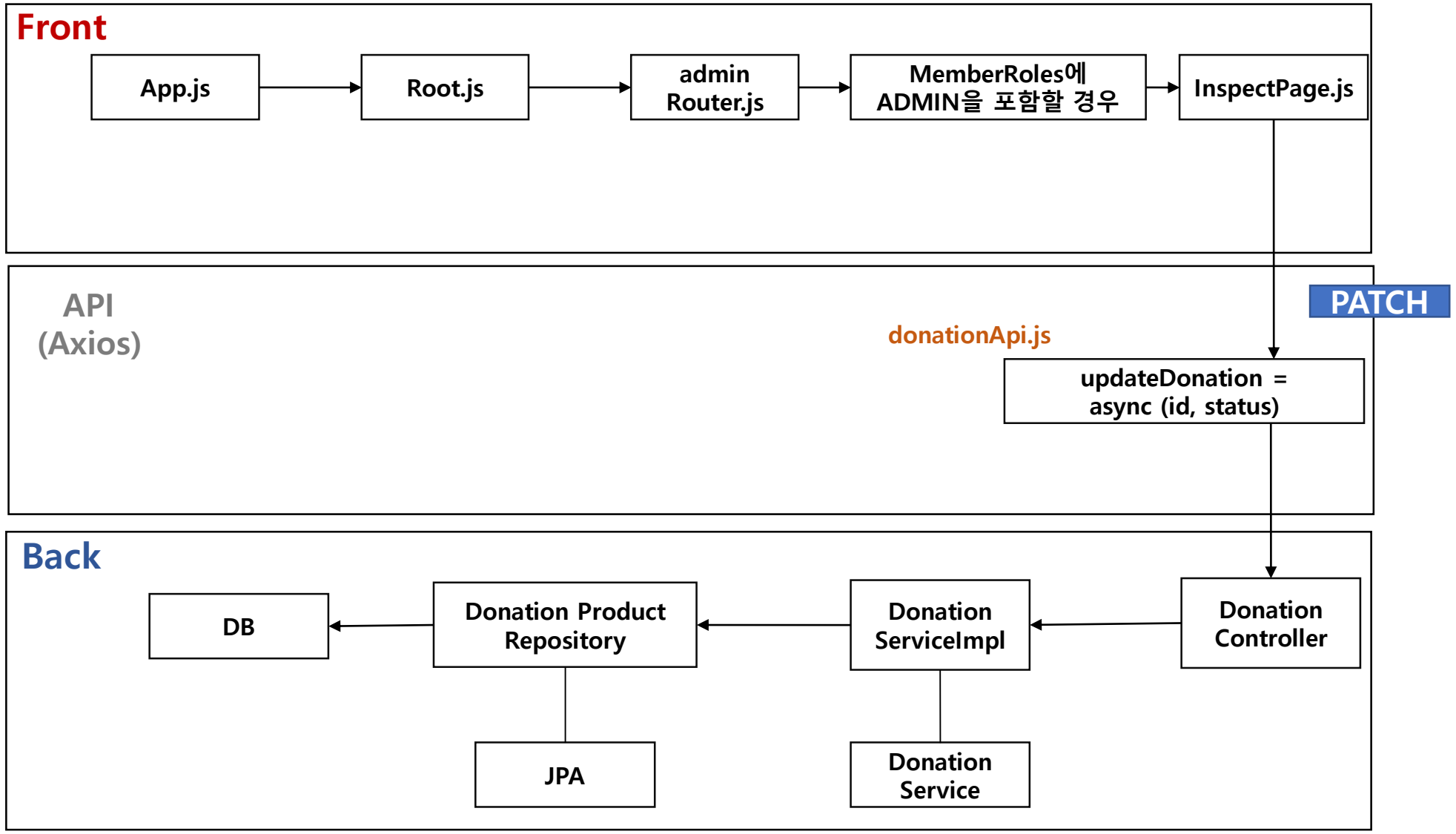
```
// 4. DonationProduct 저장
DonationProduct savedDonationProduct = donationProductRepository.save(donationProduct);
```

id	condition_note	donated_at	status	donor_id	product_i
8	오염 및 손상 없음, 구매 시기 5월 25일	2025-07-29 16:48:59.184451	INSPECTING	sh879797@naver.com	14

나눔 신청을 한 후 DB에 나눔 상품 데이터를 저장
상태는 "INSPENTING"상태로 저장되어있음.

REDUCE
REUSE RECYCLE

나눔 상품 검수



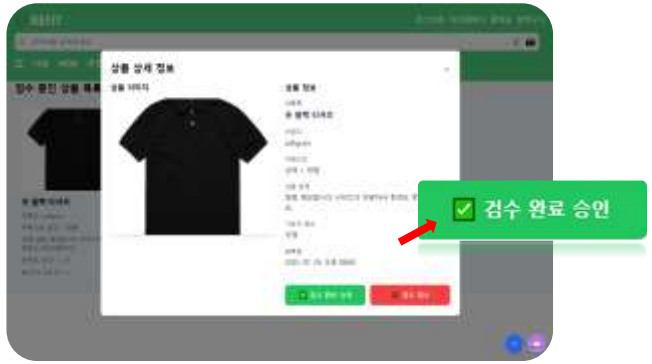
REDUCE

REUSE RECYCLE

나눔 상품 검수

시스템

나눔 상품 검수 내역



	id	condition_note	donated_at	status	donor_id
▶	1	엄청 깨끗합니다 사이즈가 안맞아서 한번도 못...	2025-07-29 20:08:13.954190	APPROVED	u1@aaa.com

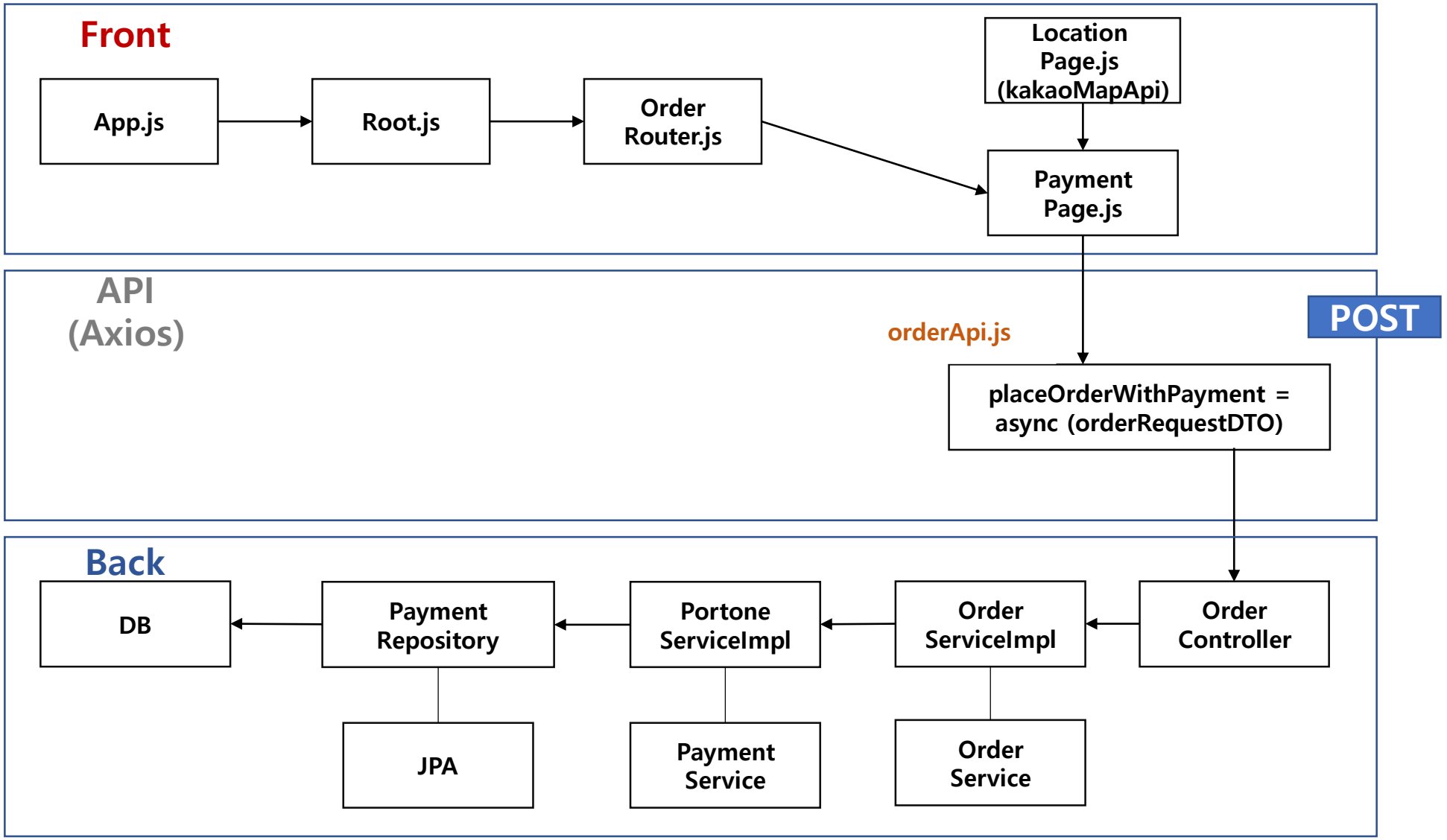

```
// 2. 검수 중인 기부 상품 목록 조회
export const getInspectingDonationProducts = async () => {
  try {
    const response = await axios.get(
      `${API_SERVER_HOST}/api/donation/products/inspecting`
    );
    return response.data; // DTO 배열 반환
  } catch (error) {
    console.error("검수 중인 기부 상품 목록 조회 실패:", error);
    throw error;
  }
};
```

```
@PatchMapping("/products/{id}/status")
public ResponseEntity<?> updateDonationProductStatus(
        @PathVariable Long id,
        @RequestParam("status") String statusStr,
        @RequestHeader("Authorization") String authorizationHeader) {
```

```
public DonationProduct updateDonationProductStatus(Long donationProductId, DonationStatus newStatus) {
    DonationProduct product = donationProductRepository.findById(donationProductId)
        .orElseThrow(() -> new IllegalArgumentException("기부상품을 찾을 수 없습니다. id=" + donationProductId));

    product.setStatus(newStatus);
    return donationProductRepository.save(product);
}
```

DB에 승인된 상태로 변경후
변경된 데이터를 프론트로 전달

의류 상품 결제



REDUCE

REUSE RECYCLE

의류 상품 결제

REDUCE

REUSE RECYCLE

의류 상품 결제

프론트엔드에서 백엔드로
결제 주문 API 호출

```
export const placeOrderWithPayment = async (orderRequestDTO) => {
    const memberCookie = getCookie("member");
    const headers = { "Content-Type": "application/json", Authorization: `Bearer ${memberCookie?.accessToken}` };

    const res = await axios.post(`${orderHost}/payment?email=${encodeURIComponent(memberCookie?.member?.email)}`,
    orderRequestDTO, { headers, withCredentials: true });
    return res.data;
};
```

포트원 API를 통한 실제 결제 검증 수행

```
@Override
public boolean verifyPayment(String impUid, String merchantUid, int amount) {
    // 포트원 결제 검증 (테스트 환경에서는 간단 검증)
    if (impUid == null || impUid.isEmpty() || merchantUid == null || merchantUid.isEmpty() || amount < 0) {
        log.error("결제 정보 검증 실패");
        return false;
    }
    log.info("결제 검증 성공: imp_uid={}, merchant_uid={}, amount={}", impUid, merchantUid, amount);
    return true;
}
```

백엔드에서 결제 검증, 주문 생성,
적립금 처리를 모두 담당하는 핵심 로직

```
@Override
@Transactional
public Long placeOrderWithPayment(String memberEmail, OrderRequestDTO orderRequestDTO) {
    // 결제 금액 및 적립금 계산
    int totalAmount = calculateTotalAmount(orderRequestDTO.getItems());
    int usedCredit = orderRequestDTO.getUsedCredit() != null ? orderRequestDTO.getUsedCredit() : 0;
    int finalAmount = totalAmount - usedCredit;
    int earnedCredit = "point".equals(orderRequestDTO.getPaymentMethod()) ? 0 : (int) Math.round(finalAmount * 0.08);

    // 적립금 차감 및 포트원 검증
    if (usedCredit > 0) deductMemberCredit(memberEmail, usedCredit);
    if (finalAmount > 0) portoneService.verifyPayment(orderRequestDTO.getImpUid(), orderRequestDTO.getMerchantUid(), finalAmount);

    // 주문 생성 및 결제 정보 저장
    Long orderId = placeOrder(memberEmail, orderRequestDTO);
    savePaymentInfo(orderId, memberEmail, orderRequestDTO, finalAmount);
    if (earnedCredit > 0) addMemberCredit(memberEmail, earnedCredit);

    return orderId;
}
```

결제 검증 후 백엔드 API 호출하고
주문 완료 페이지로 이동

```
const verifyPayment = async (paymentResponse, paymentMethod, customUsedCredit = null) => {
    // 적립금 계산 및 백엔드 API 호출
    const earnedCredit = paymentMethod === "point" ? 0 : Math.round(finalAmount * 0.08);
    const orderRequestDTO = { ...orderData.orderRequestDTO, impUid: paymentResponse.imp_uid,
        merchantUid: paymentResponse.merchant_uid, paymentMethod, usedCredit: customUsedCredit || usedCredit };

    const result = await placeOrderWithPayment(orderRequestDTO);
    navigate("/order/success", { state: { orderInfo: { totalAmount, usedCredit, finalAmount, earnedCredit } } });
};
```

DB에 저장되는 내용

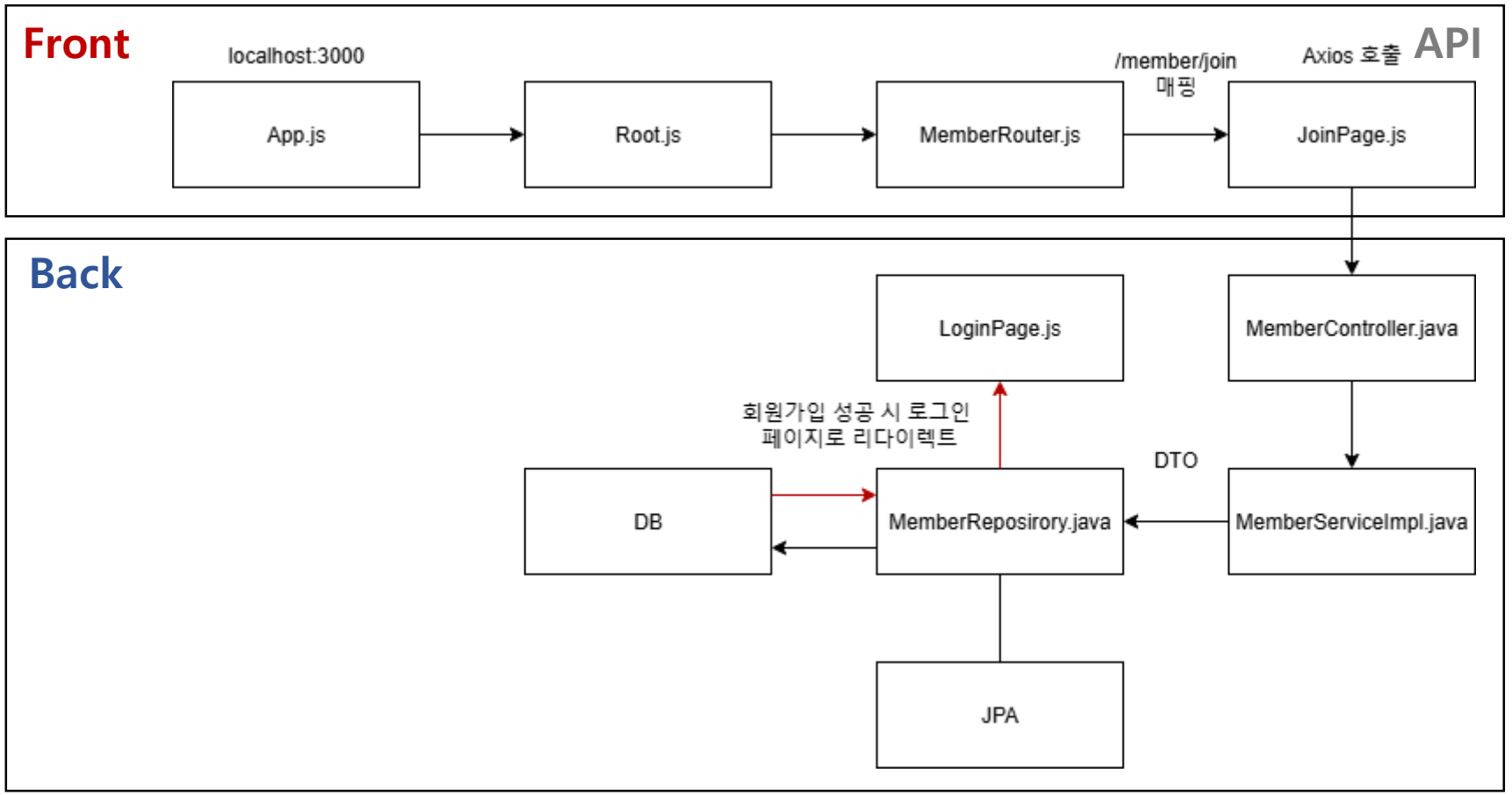
id	amount	created_at	imp_uid	merchant_uid	paid_at	pay_method	pg_provider	status	member_email	order_id
1	20000	2025-07-30 17:22:50.207599	imp_636751592897	order_1753863750636	2025-07-30 17:22:50.207599	kakaopay	kakaopay	paid	cccc5606@naver.com	2

로그인

로컬 회원 가입

소셜 로그인

일반 회원가입



일반 회원가입

사용자 입력 데이터를 백엔드로 전송

```
const handleJoin = async (e) => {
  const res = await axios.post("http://localhost:8080/api/member/join", form);
  navigate("/member/login");
};
```

이메일 중복 확인, 비밀번호 암호화,
회원 엔티티 생성 및 DB 저장

```
public void join(MemberJoinDTO joinDTO) {
    boolean exists = memberRepository.existsByEmail(joinDTO.getEmail());
    if (exists) throw new IllegalStateException(s:"이미 가입된 이메일입니다.");
    Member member = Member.builder().email(joinDTO.getEmail()).pw(passwordEncoder.encode(joinDTO.getPw()))
    .nickname(joinDTO.getNickname()).social(false).build();
    member.addRole(MemberRole.MEMBER);
    memberRepository.save(member);
}
```

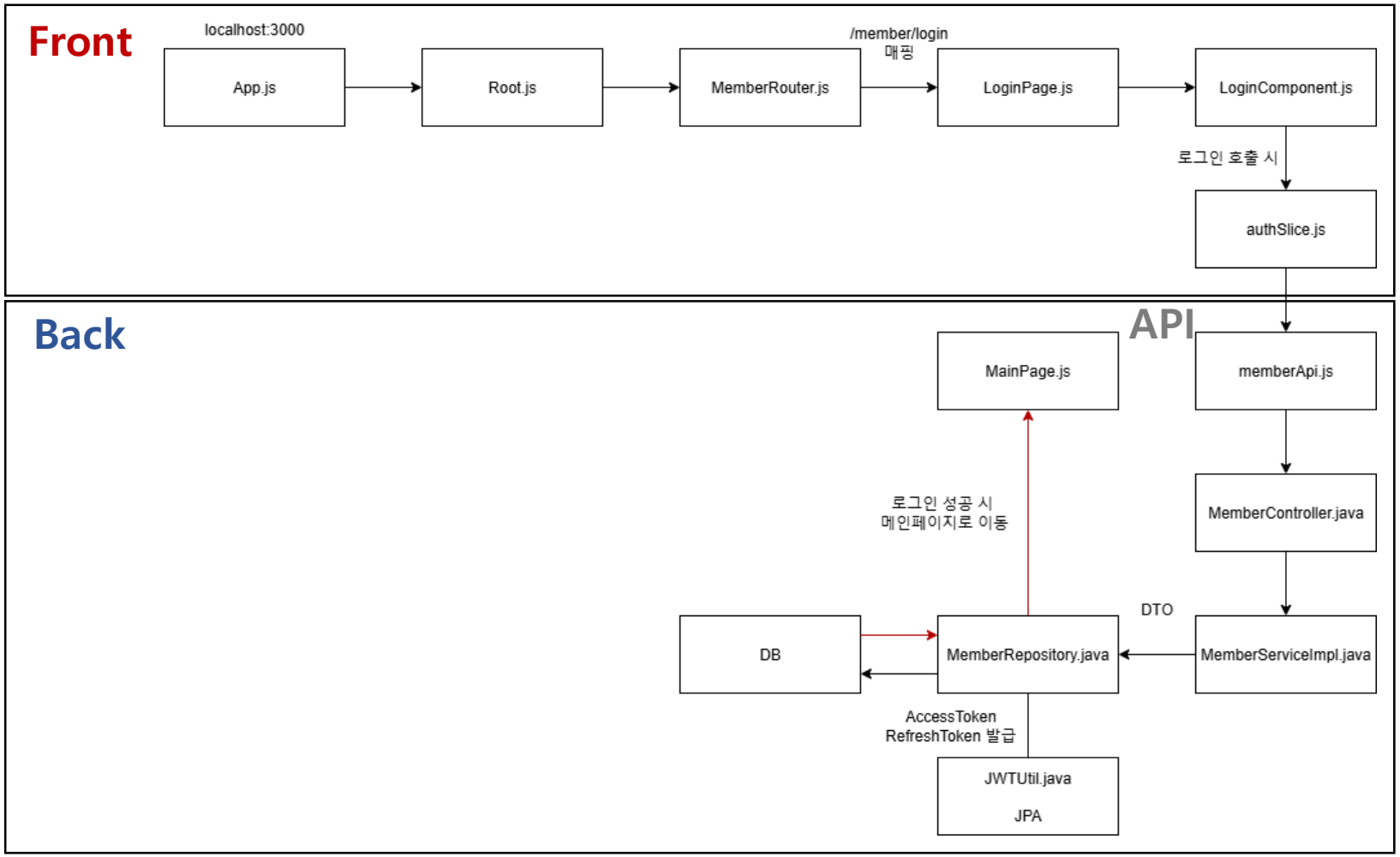
프론트엔드 요청을 받아 서비스 계층으로 전달

```
@PostMapping("/join")
public String join(@RequestBody MemberJoinDTO joinDTO) {
    memberService.join(joinDTO);
    return "회원가입 성공";
}
```

회원가입 성공 시 DB 내용

	email	nickname	pw	social
	test@test.com	test	$2a$10$crT510ef/Eo5QpBUGQo.9OXJ9c/mLAZZWJV1CdPjKd.TpGFE0oK0u	0

일반 로그인



일반 로그인

사용자 입력을 Redux 액션으로
디스패치 후 성공 시 메인 페이지로 이동

```
const handleSubmit = async (e) => {
  const resultAction = await dispatch(loginPostAsync({ email, pw }));
  if (loginPostAsync.fulfilled.match(resultAction)) {
    navigate("/main");
  }
};
```

이메일 존재 확인, 비밀번호 검증 후
회원 정보를 DTO로 변환하여 반환

```
public MemberDTO login(MemberLoginDTO loginDTO) {
    Optional<Member> result = memberRepository.findById(loginDTO.getEmail());
    if (!result.isPresent()) throw new RuntimeException("존재하지 않는 이메일입니다.");
    Member member = result.get();
    if (!passwordEncoder.matches(loginDTO.getPw(), member.getPw())) throw new RuntimeException("비밀번호가 일치하지 않습니다.");
    return entityToDTO(member, memberBodyInfoRepository);
}
```

백엔드 로그인 API를 호출하여
사용자 인증 및 토큰을 받아옴

```
export const loginPost = async ({ email, pw }) => {
  const res = await axios.post(`${API_SERVER_HOST}/api/member/login`, { email, pw });
  return res.data;
};
```

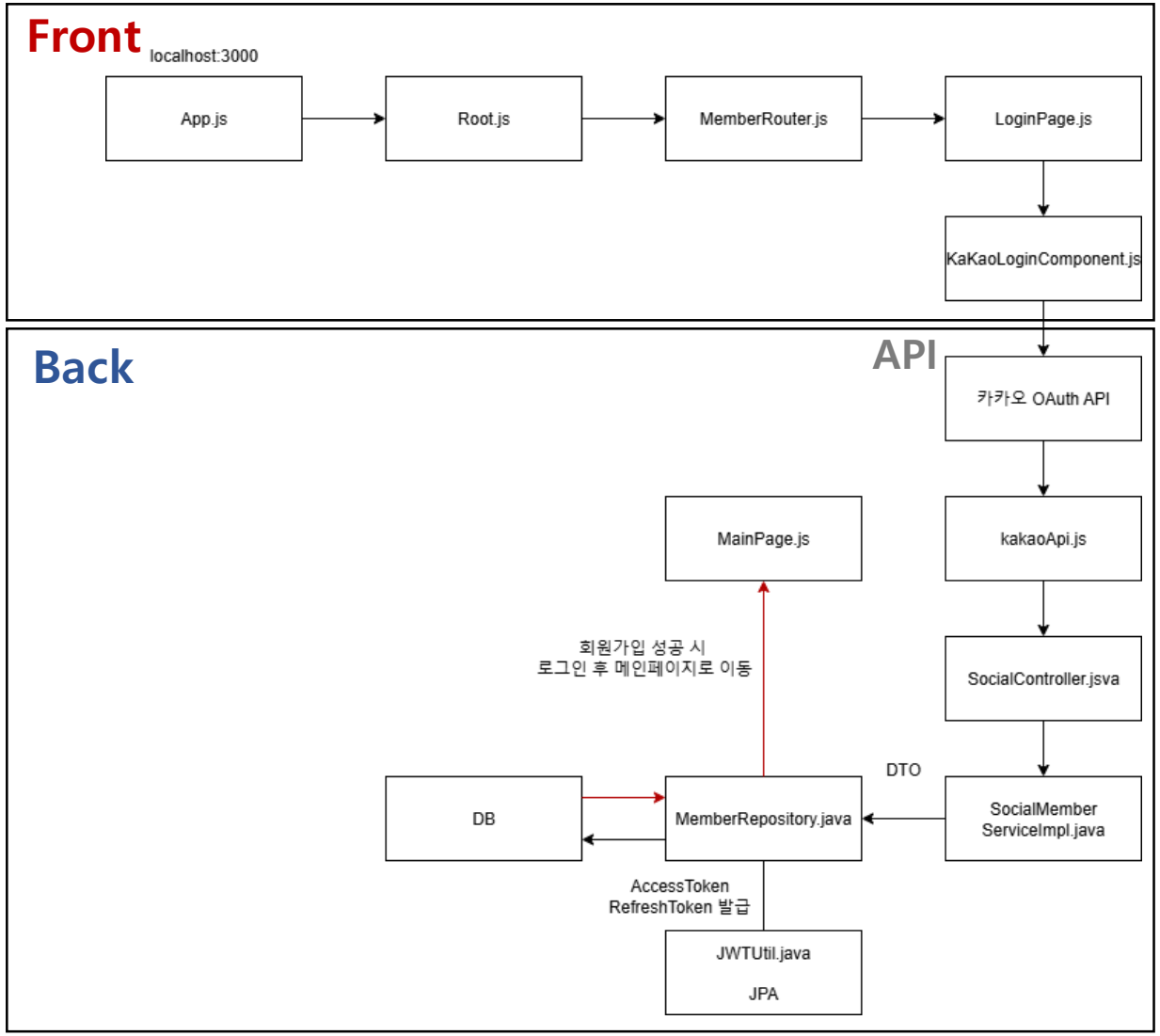
클레임 정보와 만료 시간을 받아
JWT 토큰을 생성

```
public String generateToken(Map<String, Object> claims, Duration expiration) {
    return Jwts.builder().setClaims(claims).setIssuedAt(new Date())
    .setExpiration(new Date(System.currentTimeMillis() + expiration.toMillis()))
    .signWith(key).compact();
}
```

사용자 인증 후 JWT 토큰을
발급하여 클라이언트에 반환

```
@PostMapping("/login")
public ResponseEntity<Map<String, Object>> login(@RequestBody MemberLoginDTO loginDTO) {
    MemberDTO memberDTO = memberService.login(loginDTO);
    String accessToken = jwtUtil.generateToken(claims, Duration.ofMinutes(30));
    String refreshToken = jwtUtil.generateToken(claims, Duration.ofDays(7));
    return ResponseEntity.ok(Map.of("accessToken", accessToken, "refreshToken", refreshToken, "member", memberMap));
}
```

카카오 회원가입



카카오 회원가입

카카오 OAuth 인증 URL을 생성하여
사용자를 카카오 인증 페이지로 이동

```
const redirect_uri = `http://localhost:3000/member/kakao`;
export const getKakaoLoginLink = () => {
  return `${auth_code_path}?client_id=${rest_api_key}&redirect_uri=
  ${encodeURIComponent(redirect_uri)}&response_type=code`;
};
```

카카오 Access Token을 백엔드로
전송하여 회원 정보 처리

```
export const getMemberWithAccessToken = async (accessToken) => {
  const res = await axios.get(`${API_SERVER_HOST}/api/member/kakao?accessToken=${accessToken}`);
  return res.data;
};
```

카카오 액세스 토큰으로 회원 정보를
조회하고 JWT 토큰을 발급

```
@GetMapping("/api/member/kakao")
public Map<String, Object> getMemberFromKakao(String accessToken) {
    MemberDTO memberDTO = socialMemberService.getKakaoMember(accessToken);
    String jwtAccessToken = jwtUtil.generateToken(claims, Duration.ofMinutes(60 * 24));
    return Map.of("accessToken", jwtAccessToken, "refreshToken", jwtRefreshToken, "member", memberDTO);
}
```

카카오 API로 이메일 조회 후 기존
회원이면 반환, 없으면 소셜 회원 자동 생성

```
public MemberDTO getKakaoMember(String accessToken) {
    String email = getEmailFromKakaoAccessToken(accessToken);
    Optional<Member> result = memberRepository.findById(email);
    if(result.isPresent()) {
        return entityToDTO(result.get(), memberBodyInfoRepository);
    }
    Member socialMember = makeSocialMember(email);
    memberRepository.save(socialMember);
    return entityToDTO(socialMember, memberBodyInfoRepository);
}
```

카카오 액세스 토큰으로 카카오 API를 호출하여
사용자 이메일 정보를 조회

```
private String getEmailFromKakaoAccessToken(String accessToken) {
    RestTemplate restTemplate = new RestTemplate();
    HttpHeaders headers = new HttpHeaders();
    headers.add("Authorization", "Bearer " + accessToken);
    ResponseEntity<LinkedHashMap> response = restTemplate.exchange(kakaoGetUserURL, HttpMethod.GET
    , entity, LinkedHashMap.class);
    LinkedHashMap<String, String> kakaoAccount = bodyMap.get("kakao_account");
    return kakaoAccount.get("email");
}
```

소셜 회원가입 성공 시 DB 내용

email	nickname	pw	social
refit@kakao.com	소셜회원	$2a$10$6ccmYmoDiFKCxuLkkC7SP.0j1B4T/4xEx4S47s/YW2Uxsq1eeaRY.	1

REDUCE

REUSE RECYCLE

커뮤니티

찬찬찬메바 2025. 7. 30. 오후 3:27:13 수정 삭제

옷이 이쁘네요!

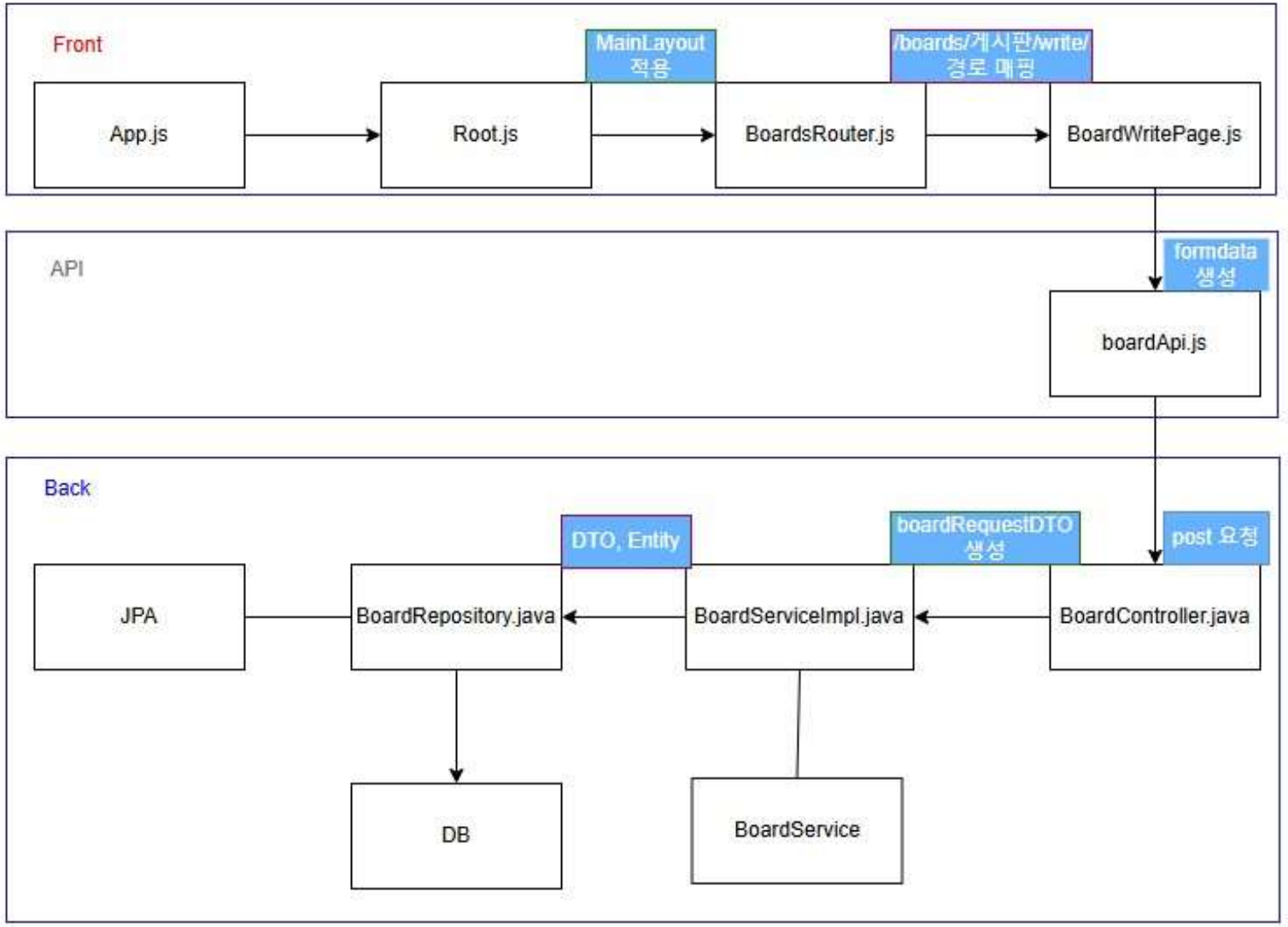
답글

댓글 등록

REDUCE

REUSE RECYCLE

커뮤니티



REDUCE

REUSE RECYCLE

커뮤니티

게시글 등록

```
// 게시글 작성 (기존 호환성)
export const createPost = (boardRequestDTO, images) => {
  const formData = new FormData();
```

Formdata 생성

```
// 2. Board 엔티티 생성
LocalDateTime now = LocalDateTime.now();
Board board = Board.builder()
        .title(requestDTO.getTitle())
        .content(requestDTO.getContent())
        .writer(member.getNickname()) // writer는 nickname으로 설정
        .boardType(requestDTO.getBoardType())
        .createdAt(now)
        .updatedAt(now)
        .build();
```

```
// 게시글 작성
@PostMapping(consumes = {"multipart/form-data"})
public ResponseEntity<BoardResponseDTO> createBoard(
        @RequestParam(value = "title", required = false) String title,
        @RequestParam(value = "content", required = false) String content,
        @RequestParam(value = "boardType", required = false) String boardType,
        @RequestPart(value = "images", required = false) List<MultipartFile> images)
```

Multipart 를 통한 파일처리

게시글 목록

페이징 처리

```
// 게시글 전체 목록 조회 (페이징)
export const fetchPostsWithPaging = (page = 0, size = 10) =>
  axios.get(`${API_BASE}/boards/list/paging?page=${page}&size=${size}`, {
    headers: getAuthHeaders(),
  });
```

```
@Override
public List<BoardResponseDTO> getAllBoards() {
    List<Board> boards = boardRepository.findAllByOrderByCreatedAtDesc();
    return boards.stream()
            .map(BoardResponseDTO::fromEntity)
            .collect(Collectors.toList());
}
```

```
// 게시글 목록 조회
@GetMapping
public ResponseEntity<List<BoardResponseDTO>> getAllBoards() {
    List<BoardResponseDTO> boards = boardService.getAllBoards();
    return ResponseEntity.ok(boards);
}
```

게시글 댓글

```
// 댓글 작성 (기존 호환성)
export const createComment = (postId, commentData) =>
  axios.post(`${API_BASE}/boards/${postId}/comments`, commentData, {
    headers: getAuthHeaders(),
  });
```

```
Comment comment = Comment.builder()
        .content(commentRequestDTO.getContent())
        .writer(commentRequestDTO.getWriter())
        .board(board)
        .build();

Comment savedComment = commentRepository.save(comment);
System.out.println("[DEBUG] Comment saved with ID: " + savedComment.getId());
```

```
@GetMapping("/{boardType}/{boardId}/comments")
public ResponseEntity<List<CommentResponseDTO>> getComments(@PathVariable String boardType, @PathVariable Long boardId) {
    List<CommentResponseDTO> comments = commentService.getCommentsByBoardId(boardId);
    return ResponseEntity.ok(comments);
}
```

DB에 저장된 게시글

id	content	created_at	title	updated_at	writer	member_email
7	사고싶었던 옷을 나눔을통해 얻을수 있어서 행...	2025-07-30 15:23:59.579048	퓰절인 옷을 나눔받았어요	2025-07-30 15:23:59.579048	잔잔잔메바	kim37734135@gmail.com

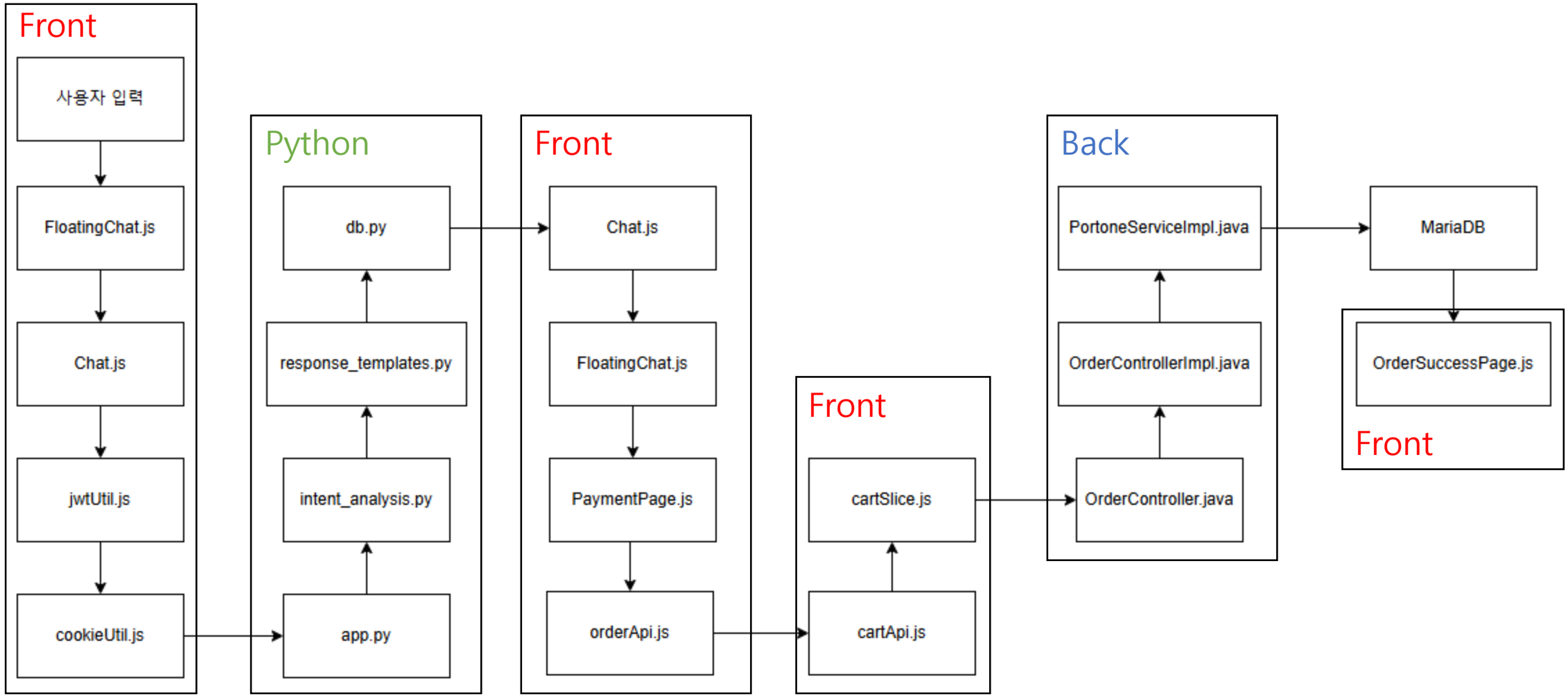
DB에 저장된 댓글

id	content	created_at	updated_at	writer	board_id
2	옷이 이쁘네요!	2025-07-30 15:27:13.466061	2025-07-30 15:27:13.466061	잔잔잔메바	7

REDUCE

REUSE RECYCLE

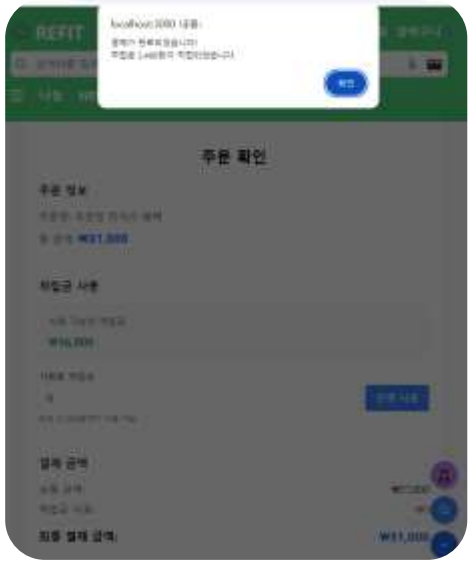
챗봇 AI 를 활용한 결제



REDUCE

REUSE RECYCLE

챗봇 AI 를 활용한 결제



사용자 의도 분석

```
# 결제 방식 감지
payment_method = detect_payment_method(user_message)

print(f"장바구니 구매 요청 감지: {user_message}")
print(f"감지된 결제 방식: {payment_method}")

if payment_method == "card":
    response = "장바구니 상품을 카드로 구매하시겠습니까? 결제 페이지로 이동합니다."
elif payment_method == "point":
    response = "장바구니 상품을 적립금으로 구매하시겠습니까? 적립금 결제 페이지로 이동합니다."
else:
    response = "장바구니 상품을 구매하시겠습니까? 결제 페이지로 이동합니다."
```

결제 수단 선택

```
} else if (paymentMethod === "card") {
  // 카드 결제 요청
  console.log("카드로 결제 시작");
  setSelectedPayment("card");
  setShowPaymentModal(true);
  return;
}
```

```
// 1. 결제 수단 선택
handlePaymentSelect("card")

// 2. 포트원 SDK로 결제창 띄우기
initializePayment("card")

// 3. 포트원 결제창에서 실제 카드 결제
window.IMP.request_pay(paymentData, (response) => {
  // 결제 성공 시 백엔드로 검증 요청
  verifyPayment(response, "card")
})
```

결제 검증 및 주문 처리

1. 백엔드에서 처리 (OrderServiceImpl.java)

```
// 1. 결제 검증
PortoneService.verifyPayment(impUid, amount)

// 2. 주문 데이터 저장
Order order = new Order();
orderRepository.save(order);

// 3. 재고 차감
productService.updateStock(productId, quantity);

// 4. 적립금 처리
memberPointRepository.save(new MemberPoint(...));

// 5. 장바구니 비우기
cartService.clearCart(memberId);
```

주문 완료

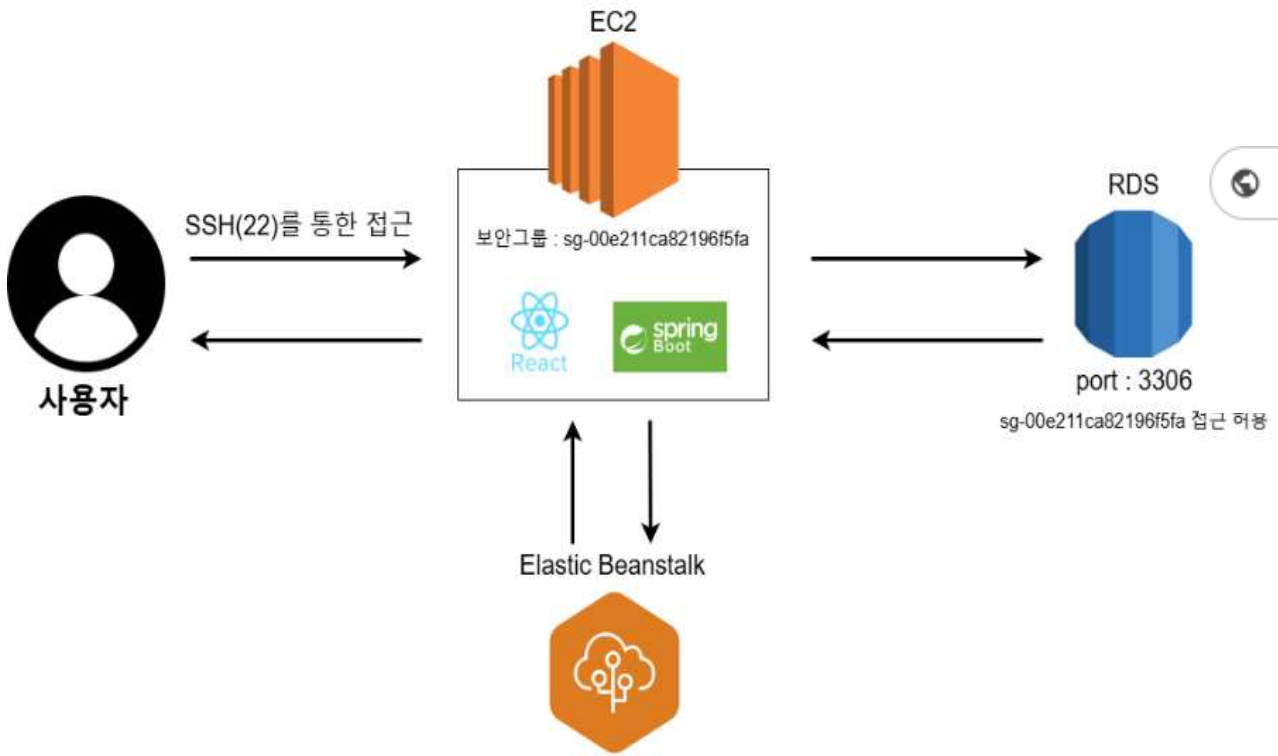
2. 프론트엔드에서 처리 (PaymentPage.js)

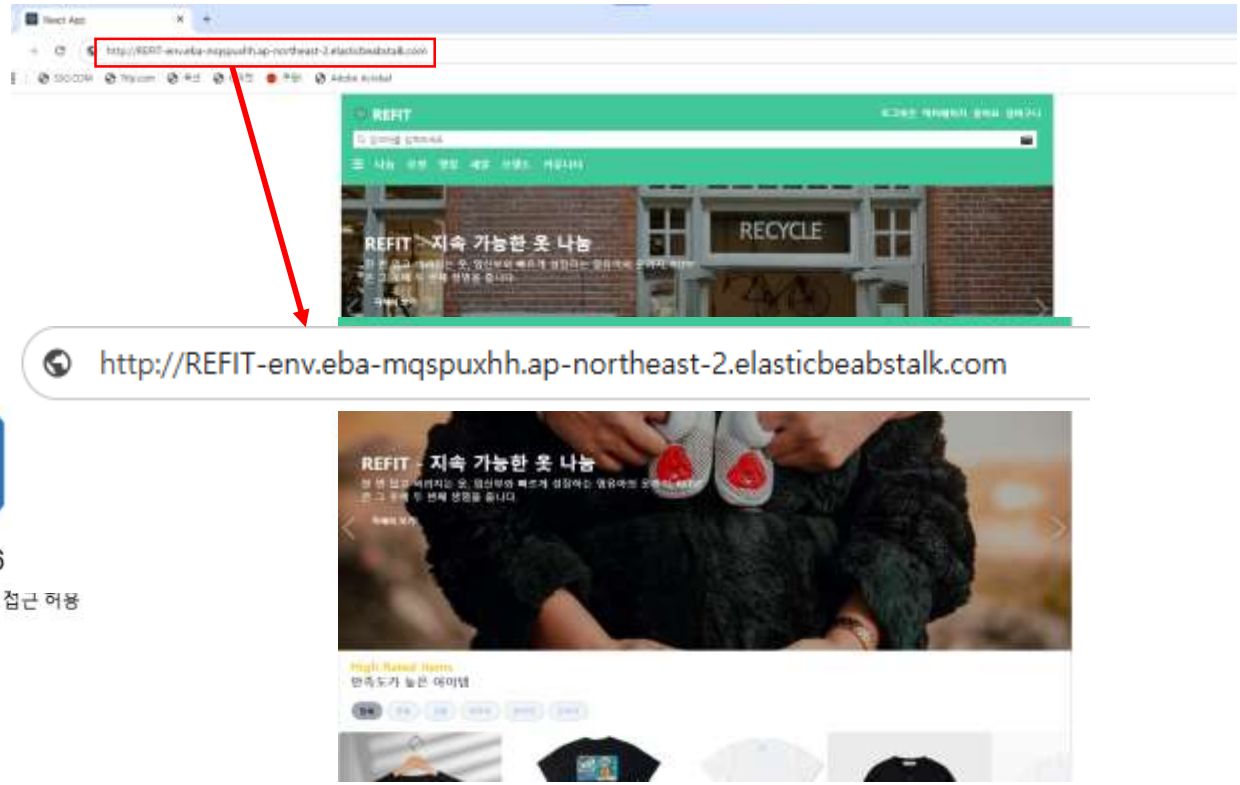
```
// 1. 백엔드 API 호출 결과 확인
const result = await placeOrderWithPayment(orderRequestDTO);

// 2. 쿠키 업데이트 (적립금 정보)
setCookie("member", updatedMember, 7);

// 3. 주문 완료 페이지로 이동
navigate("/order/success", {
  state: {
    orderInfo: {
      totalAmount: orderData.totalAmount,
      usedCredit: usedCredit,
      finalAmount: finalAmount,
      earnedCredit: earnedCredit,
      orderName: orderData.orderName,
      paymentMethod: selectedPayment
    }
  }
});
```

REDUCE

REUSE RECYCLE

AWS를 통한 배포





오류 해결1 (이미지 경로 중복 오류)

백엔드 경로 반환 로직 수정

기존 코드

```
// ❌ 팀원이 겪은 오류 코드
return "/" + subDirectory + "/" + newFileName; // /uploads/파일명.jpg
```

수정 코드

```
// ✅ 수정된 코드
if ("uploads".equals(subDirectory)) {
    return newFileName; // 파일명만 반환
}
return "/" + subDirectory + "/" + newFileName;
```

백엔드에서 파일명.jpg 반환
→ 프론트엔드에서 /uploads/ 추가
→ /uploads/파일명.jpg (정상)

프론트 경로 반환 로직 수정

기존 코드

```
// ❌ 팀원이 겪은 오류 코드
src={`http://localhost:8080${image.imageUrl}`}
```

수정 코드

```
// ✅ 수정된 코드
src={`http://localhost:8080/uploads/${image.imageUrl}`}
```

프론트엔드 호환성: 프론트엔드에서 http://localhost:8080/uploads/${imageUrl}
형태로 사용

오류 해결1 (이미지 경로 중복 오류)

정적 리소스 매핑 재 설정

기존 코드

```
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
    // uploads 폴더에 대한 정적 리소스 매핑
    registry.addResourceHandler("/uploads/**")
            .addResourceLocations("file:build/resources/main/static/uploads/")
            .setCachePeriod(3600)        // 1시간 캐싱
            .resourceChain(true);        // 리소스 체인 활성화
}
```

수정 코드

```
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
    // uploads 폴더에 대한 정적 리소스 매핑
    registry.addResourceHandler("/uploads/**")
            .addResourceLocations("file:build/resources/main/static/uploads/")
            .setCachePeriod(3600)        // 1시간 캐싱
            .resourceChain(true);        // 리소스 체인 활성화
}
```

Spring Boot의 정적 리소스 처리 방식

저장 경로 변경

초기 설계: refit_backend/src/main/resources/static/uploads/

현재 설계: refit_backend/build/resources/main/static/uploads/

JWT 필터에서 정적 리소스 제외

```
// ❌ 팀원이 겪은 오류: 정적 리소스도 JWT 인증 필요
// ✅ 수정된 코드
if (path.startsWith("/uploads")) {
    return true; // 인증 없이 접근 허용
}
```

해결된 문제

경로 중복 문제 해결: 정확한 이미지 URL 생성

성능 최적화: 정적 리소스 캐싱 설정

보안 강화: 정적 리소스 접근 시 불필요한 인증 제거

일관성 확보: 모든 게시판에서 동일한 이미지 표시 방식

Spring Boot 표준 준수: 빌드 시스템과의 완전한 통합

팀 개발 효율성 향상: 커뮤니티 기능 개발 지연 해결

오류 해결2 (댓글/답글 결합에 따른 함수 충돌)

문제점

- 댓글과 답글 로직 및 상태가 한 컴포넌트에 몰려 복잡하고 가독성 저하
- 상태 충돌 및 관리 어려움으로 유지보수성 낮음
- UI가 중첩되어 확장 및 기능 추가 시 코드 난이도 급증

수정 전 코드

```
const CommentItem = ({ comment }) => {
  const hasReplies = replies[comment.id] && replies[comment.id].length > 0;
  const isShowingReplies = showReplies[comment.id];
```

해결 방안

- CommentSection(전체 관리), CommentItem(댓글 단위), ReplyItem(답글 단위)로 컴포넌트 분리
- 각 컴포넌트가 독립적으로 상태 관리 및 UI 담당 (SRP 적용)
- API 호출과 상태 업데이트를 최상위에서 집중 처리, 자식은 UI와 개별 동작만 담당

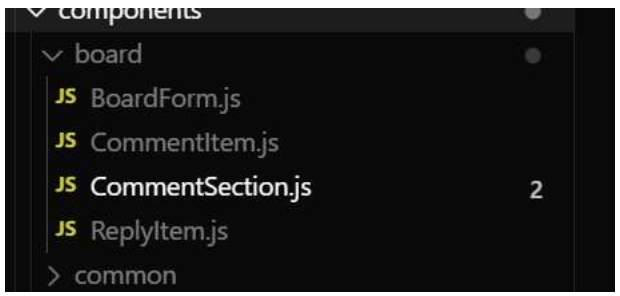
```
//3. CommentItem.jsx 댓글 수정/삭제/답글 상태 처리
{editingComment === comment.id ? (
  <textarea value={editContent} onChange={(e) => setEditContent(e.target.value)} />
) : (
  <p>{comment.content}</p>
)}

// 댓글 수정/삭제 버튼
<button onClick={() => handleStartEdit(comment)}>수정</button>
<button onClick={() => handleDeleteComment(comment.id)}>삭제</button>

// 답글 버튼 및 토글
<button onClick={() => handleStartReply(comment)}>답글</button>
{hasReplies && (
  <button onClick={() => toggleReplies(comment.id)}>
    {isShowingReplies ? "답글 숨기기" : `답글 보기 (${replies[comment.id]?.length || 0})`}
  </button>
)}
```

수정 후 코드

```
//3. ReplyItem.jsx 답글 내용 및 수정폼 처리
{editingReply === reply.id ? (
  <textarea value={editReplyContent} onChange={(e) => setEditReplyContent(e.target.val
) : (
  <p>{reply.content}</p>
)}

// 답글 수정/삭제 버튼
<button onClick={() => handleStartEditReply(reply)}>수정</button>
<button onClick={() => handleDeleteReply(reply.id)}>삭제</button>
```

JS CommentSection.js

컴포넌트 분리



오류 해결3 (결제 API 응답 타입 오류)

오류 상황

개발자가 계속 Object로 설정하고 있어서 String으로 못받고 있었음
`imp_uid`를 Object 타입으로 정의했는데 실제 API는 String을 반환

기존 코드

```
// OrderRequestDTO.java - 잘못된 타입 정의
private Object impUid; // Object로 잘못 정의
private Object merchantUid; // Object로 잘못 정의

// OrderServiceImpl.java - 오류 발생
Object impUidObj = orderRequestDTO.getImpUid(); // Object로 받음
String impUid = (String) impUidObj; // ClassCastException 발생!

boolean isPaymentValid = portoneService.verifyPayment(
    impUid,
    merchantUid,
    finalAmount
);
```

오류 원인

DTO에서 `imp_uid`를 Object 타입으로 잘못 정의
실제 포트원 API는 String을 반환하는데 Object로 받으려고 함
타입 불일치로 인한 `ClassCastException` 발생

수정 코드

```
// OrderRequestDTO.java - 올바른 타입으로 수정
private String impUid; // String으로 수정
private String merchantUid; // String으로 수정

// OrderServiceImpl.java - 안전한 처리
String impUid = orderRequestDTO.getImpUid(); // String으로 받음
String merchantUid = orderRequestDTO.getMerchantUid(); // String으로 받음

boolean isPaymentValid = portoneService.verifyPayment(
    impUid,
    merchantUid,
    finalAmount
);
```

해결 결과

타입 일치: 실제 API 응답 타입과 일치하도록 수정

런타임 오류 방지: ClassCastException 발생 방지

코드 안정성: 타입 안전성 확보

결과: 결제 API 응답을 올바른 String 타입으로 안전하게 처리 완료

느낀 점

구분	내용
잘한 점	초반 설계를 잘 잡아 놓아서 각자 구현한 기능들을 합쳤을 때 큰 오류 문제들이 없이 서버가 잘 돌아갔습니다. 챗봇 AI기능을 추가하여 사용자 결제 과정 편의성을 높였습니다. 카카오 로그인,카카오 맵 API를 활용하여 로그인 편의성 및 간편한 주소지 설정이 가능하도록 개선했습니다. 1시간 캐시 설정으로 반복 요청 시 성능 향상하였습니다.
어려움 극복	커뮤니티 기능 구현중 파일 업로드 시 이미지가 보이지 않는 문제를 팀원들끼리 협력하여 오류를 발견하고 문제를 해결하였습니다. 기능개발 중 Securtity관련 코드로 인해 데이터를 받아오지 못하는 에러를 로그를 찍어가며 발견하고 Security관련 기능을 맡은 팀원들과 함께 권한을 받는 방법을 터득하여 문제를 해결했습니다.
아쉬웠던 점	초반에 들어가야할 기능들을 빠르게 구현하고 미흡한 UI부분을 개선했으면 좀 더 완성도 있는 프로젝트가 나올 수 있었으면 하는 아쉬움이 남았습니다.
깨달은 점	개발과정 중 해결되지 않는 문제들은 최대한 빠르게 팀원들에게 도움을 요청해야 시간을 절약할 수 있다는 것을 알게 되었고, 알지못한 라이브러리들을 접해보면서 기능에 새로움을 발견하여 다음 프로젝트에는 더 좋은 결과물을 만들어 내야겠다고 깨달았습니다.

REDUCE

REUSE RECYCLE

감사합니다.

1조 REFIT

GitHub

고인한: https://github.com/inhan99

김찬영: https://github.com/chanO4135

고윤호: https://github.com/KessokuMAS

송승찬: https://github.com/bannana-key