**Technology (보유기술)**

|  |  |
| --- | --- |
| **기술분류** | **보유기술** |
| 언어 | * **Java (JDK 17)**: Spring Boot 백엔드 개발에 활용. 주요 도메인 처리 로직 구현. * **JavaScript** : React 프론트엔드 구성 및 비동기 통신(Axios) 처리에 사용. * **HTML5 / CSS3**: 사용자 인터페이스 마크업 및 스타일링에 활용. * **SQL**: MySQL 기반 테이블 생성 및 JOIN, 집계 쿼리 작성 경험 다수. |
| 프론트엔드 | **React**   * react-router-dom을 활용한 페이지 라우팅 구성 * useState, useEffect 기반 상태 관리 * Axios를 활용하여 백엔드 API 연동 * 상품 등록/수정/삭제, 회원가입, 로그인, 주문 및 장바구니 기능 구현 * 사용자/관리자 권한 분리된 화면 구성   **UI 구성 및 반응형 디자인**   * CSS 모듈화 및 컴포넌트 기반 스타일링 * 조건부 렌더링을 통한 로그인 상태/권한별 UI 제어 * 최근 본 상품, 쿠폰 다운로드, 주문 내역, 마이페이지 구현 경험 |
| 백엔드 | **Spring Boot 3.5.3 (Maven 기반)**   * Controller, Service, Repository 3계층 구조로 도메인 개발 * 상품, 회원, 주문, 배송, 쿠폰 등 도메인 설계 및 기능 구현 * @Transactional, @RequestMapping, @Valid 등 다양한 어노테이션 활용   **Spring Security 사용, 세션 기반 로그인 접근 제어 처리**  **파일 업로드**   * 상품/회원 이미지 파일을 MultipartFile로 처리하고 EC2에 저장   **RESTful API 개발**   * 프론트엔드와의 JSON 통신을 위한 API 다수 설계 및 배포 |
| 데이터베이스 | **MySQL 8.4**   * 회원, 상품, 주문, 등 테이블 설계 및 JPA 매핑 * RDS(AWS) 환경에서도 연동 경험 있음 * Entity 간 연관관계 설계 (1:N, N:1) 및 복합 조회 DTO 구성 경험 |
| 배포 | **AWS EC2, RDS**   * React와 Spring Boot EC2 배포 * WinSCP / PuTTY를 활용한 서버 파일 전송 및 배포 * 이미지 파일 저장 처리 |
| 협업 | **Git / GitHub**   * 코드리뷰 |

\* 위의 보유 기술은 `예시`로 작성한 것이니 반드시 고치셔야 합니다. (1장 이상 작성하시기 바랍니다)

개인의 역량에 맞추어 Skill을 수정해서 작성하시기 바랍니다!! (팀 친구들과 내용이 같으면 모두 탈락~)

**Project (프로젝트)**

**Project Info**

|  |  |
| --- | --- |
| **프로젝트 명** | 풀대접 |
| **개발 기간** | 25.05.07 ~ 25.06.25 |
| **참여인원** | 4명 |
| **담당업무** |  |

**Use Technology**

|  |  |
| --- | --- |
| **개발 환경** | * **운영체제**: Windows 10, Ubuntu 20.04 (AWS EC2) * **프론트엔드 IDE**: Visual Studio Code * **백엔드 IDE**: Eclipse IDE (Spring Boot용) * **JDK**: Java 17 * **빌드 도구**: Maven * **서버**: Tomcat 10 (내장 서버 및 EC2 배포 시 사용) * **데이터베이스**: MySQL 8.4 (로컬 및 AWS RDS) * **배포 환경**: AWS EC2 (Ubuntu), RDS (MySQL) 관리 |
| **사용 도구** | * **프론트엔드 라이브러리** * React (create-react-app) * react-router-dom (라우팅) * Axios (API 통신) * **백엔드 프레임워크** * Spring Boot 3.x * Spring MVC (Controller 기반 API 구성) * Spring Data JPA (MySQL 연동) * **UI 스타일링** * CSS3, 사용자 정의 스타일 시트 * **이미지 업로드 및 저장** * MultipartFile (Spring) |
| **사용 기술** | * **프론트엔드** * React + JSX 기반 컴포넌트 개발 * 상태 관리: useState, useEffect * 조건부 렌더링 및 세션 기반 접근 제어 * Axios를 통한 REST API 연동 * **백엔드** * Spring Boot 기반 RESTful API 개발 * 세션 기반 로그인 구현 (HttpSession, Interceptor 사용) * 회원, 상품, 주문, 쿠폰 등 도메인별 CRUD API 구현 * 파일 업로드, 데이터 유효성 검증 및 예외 처리 * **데이터베이스** * MySQL 기반 테이블 설계 (회원, 상품, 주문, 쿠폰, 리뷰, 배송 등) * JPA Entity 연관관계 설계 (1:N, N:1) * **배포 및 운영** * AWS EC2 (Ubuntu 서버에 Spring Boot 배포) |

**Introduction**

|  |
| --- |
| **프로젝트를 한마디로 설명할 수 있는 부분** |
| **소비자의 중심으로 밀키트 상품을 보다 더 쉽고 간편하게 탐색하고 주문할 수 있는 웹 기반 플랫폼으로 밀키트 전문 쇼핑몰을 기획하게 되었습니다.** |
| **Develop Detail**   |  | | --- | | **제안** | | * **프로젝트 주제 선정**   최근 비대면 소비문화의 확산과 함께 다양한 온라인 쇼핑몰 플랫폼들이 빠르게 성장하고 있으며, 사용자 경험 중심의 기능 구현과 안정적인 백엔드 시스템이 무엇보다 중요해졌습니다. 이에 따라 **실제 서비스 가능한 쇼핑몰을 직접 기획하고 구현**함으로써, 프론트엔드와 백엔드 전체 흐름을 아우르는 실전 경험을 쌓고자 본 프로젝트를 기획하게 되었습니다. 특히 로그인, 상품 등록, 장바구니, 주문, 쿠폰 등 **일반적인 전자상거래 서비스에 필수적인 기능들을 중심**으로 프로젝트를 구성하여, 실무에 가까운 개발 역량을 키우는 것이 목표였습니다.   * **프로젝트 목표**  1. **풀스택 개발 역량 강화**  * React 프론트엔드와 Spring Boot 백엔드를 연동하여 **RESTful 아키텍처 기반의 쇼핑몰**을 직접 구축 * 사용자 인증, 상품 관리, 주문 및 결제 등 실무에서 필요한 핵심 기능 구현  1. **데이터베이스 및 API 설계 경험 습득**  * 회원, 상품, 주문, 쿠폰, 리뷰, 배송지 등 다양한 도메인의 테이블과 연관관계 설계(JPA) * 이를 기반으로 한 CRUD API 구현 및 통합 테스트 경험  1. **파일 업로드 및 이미지 처리 실습**  * 상품/회원 이미지 업로드 기능 구현 * 개발 초기에는 로컬 폴더에 저장, 이후 AWS 배**포를 위한 구조로 전환**  1. **AWS 클라우드 환경에 배포 실습**  * EC2, RDS,를 연동하여 실제 운영 가능한 상태로 프로젝트 구성 * WinSCP, PuTTY 등을 활용한 실서버 배포 경험 | | **기획** | | * **텍스트, 스크린샷, 번호, 평행이(가) 표시된 사진    AI 생성 콘텐츠는 정확하지 않을 수 있습니다.프로젝트 일정**   **텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진  AI 생성 콘텐츠는 정확하지 않을 수 있습니다.텍스트, 스크린샷, 번호, 평행이(가) 표시된 사진  AI 생성 콘텐츠는 정확하지 않을 수 있습니다.**     * **프로젝트 업무 분담** * **프로젝트 업무 전략**   **1. 목표 명확화 전략**   * 실제 운영 가능한 쇼핑몰 구축을 목표로 설정하고, 단순한 CRUD를 넘어 회원가입, 상품 관리, 주문, 쿠폰, 리뷰 등 전자상거래 전반의 기능 구현을 포함 * 프론트엔드와 백엔드를 분리 구성하여 API 기반 통신 경험 강화 * AWS 클라우드를 활용해 개발 → 테스트 → 운영까지 전 과정 체험을 목표로 설정   **2. 사용자 역할 정의 전략**   * 사용자 유형을 구분하여 기능 설계, 각 역할에 따라 화면 구성 및 접근 권한 제어 전략 수립 * 일반회원: 장바구니, 주문, 리뷰, 쿠폰 다운로드 가능 * 관리자: 상품 등록/수정/삭제, 쿠폰 발급, 전체 주문/회원 관리 가능   **3. 핵심 기능 우선순위 전략**   * 회원가입/로그인/로그아웃 (세션 기반 인증) * 상품 목록/상세 조회 * 상품 등록/수정/삭제 (관리자) * 장바구니 및 주문 기능 * 쿠폰 기능, 마이페이지, 리뷰 기능 등은 후순위로 배치하여 점진적 구현   **4. 데이터 및 테이블 설계 전략**   * ERD 초안을 작성하여 회원-상품-주문-쿠폰-리뷰 간의 연관관계를 시각화 * 주문 시 사용하는 포인트, 쿠폰 할인, 배송지, 결제 금액 계산 로직을 데이터 구조 중심으로 설계 * 이미지 저장은 DB가 아닌 파일/URL 기반으로 관리하여 성능 및 확장성 고려   **5. UI/UX 전략**   * 사용자 입장에서 가장 자주 사용하는 기능(상품 조회, 장바구니, 주문)을 중심으로 UI 흐름 설계 * 관리자 페이지는 기능 중심의 간결한 구성, 사용자 페이지는 시각적 정보 우선 * 반응형 대응보다는 데스크탑 우선으로 설계하여 초기 개발 리스크 축소 | | **설계** | | **★ 설계에서는 기능을 세분화하면서 실제 코드 예시와 함께 구성**  **프론트엔드: React 컴포넌트 기반 설계**  **1) 페이지 라우팅 구성**  import { BrowserRouter, Routes, Route } from "react-router-dom";  import ProductList from "./pages/ProductList";  import ProductForm from "./pages/ProductForm";  function App() {  return (  <BrowserRouter>  <Routes>  <Route path="/products" element={<ProductList />} />  <Route path="/admin/products/new" element={<ProductForm />} />  </Routes>  </BrowserRouter>  );  }  **react-router-dom을 활용하여 페이지 이동과 권한별 접근 구성을 명확히 분리했습니다.**  **2) 상태 관리 및 조건부 UI 제어**  const [user, setUser] = useState(null);  useEffect(() => {  api.get("/auth/me", { withCredentials: true }).then(res => setUser(res.data));  }, []);  return (  <div>  {user?.role === "ADMIN" && <Link to="/admin/products/new">상품 등록</Link>}  </div>  );  **react-router-dom을 활용하여 페이지 이동과 권한별 접근 구성을 명확히 분리했습니다.**  **백엔드: Spring Boot 계층 구조 및 도메인 설계**  **1) 3계층 구조 (Controller → Service → Repository)**  @RestController  @RequiredArgsConstructor  @RequestMapping("/api/products")  public class ProductController {  private final ProductService productService;  @PostMapping("/create")  public ResponseEntity<?> register(@ModelAttribute ProductRequestDto dto) {  productService.save(dto);  return ResponseEntity.ok("상품 등록 완료");  }  }  @Service  @RequiredArgsConstructor  public class ProductService {  private final ProductRepository productRepository;  @Transactional  public void save(ProductRequestDto dto) {  Product product = dto.toEntity(); // DTO → Entity 변환  productRepository.save(product);  }  }  public interface ProductRepository extends JpaRepository<Product, Long> {  List<Product> findByNameContaining(String keyword);  }  **각 계층을 명확히 분리해 유지보수와 테스트가 용이한 구조로 설계했습니다.**  **데이터 전달 구조: DTO vs Entity**  @Getter @Setter  public class ProductRequestDto {  private String name;  private int price;  private MultipartFile image;  public Product toEntity(String imagePath) {  return Product.builder()  .name(this.name)  .price(this.price)  .imagePath(imagePath)  .build();  }  }  @Entity  @Getter @Builder  public class Product {  @Id @GeneratedValue  private Long id;  private String name;  private int price;  private String imagePath;  }  **클라이언트와 서버 간 데이터 전달은 DTO를 통해 처리하고, DB 저장에는 Entity를 사용하여 관심사를 분리했습니다.**  **이미지 업로드 처리**  # application.properties  upload.path=./upload  String fileName = UUID.randomUUID() + "\_" + file.getOriginalFilename();  Path savePath = Paths.get(uploadPath, fileName);  Files.copy(file.getInputStream(), savePath);  **로그인/보안 및 예외 처리**  **1) 세션 기반 로그인 + 인터셉터**  @Override  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {  Member loginUser = (Member) request.getSession().getAttribute("loginMember");  if (loginUser == null) {  response.sendRedirect("/login");  return false;  }  return true;  }  **2) 예외 처리**  @ExceptionHandler(MethodArgumentNotValidException.class)  public ResponseEntity<?> handleValidationErrors(MethodArgumentNotValidException e) {  return ResponseEntity.badRequest().body("입력값 오류");  } | | **개발** | | **★ 개발에서는 실제 코딩 및 구현중심으로 진해한 작업을 정리, 설계 단계에서 계획했던 내용을 어떻게 코드로**  **구현했는지를 보여주는 영역**  **1) 회원 기능 구현**   * 회원가입 시 Bcrypt를 통한 비밀번호 암호화 * 로그인 후 세션에 사용자 정보 저장, HttpSession 활용 * 마이페이지에서 사용자 정보 수정 가능, 관리자 여부 확인 포함   // MemberService.java  public void register(MemberDto dto) {  String encodedPw = passwordEncoder.encode(dto.getPassword());  Member member = dto.toEntity(encodedPw);  memberRepository.save(member);  }  // 로그인 시 세션 저장  session.setAttribute("loginMember", loginUser);  **2) 상품 기능 구현**   * 관리자가 상품을 등록, 수정, 삭제할 수 있는 기능 * 상품 목록 조회 및 상세 페이지 구현   @PostMapping("/admin/products")  public ResponseEntity<?> create(@ModelAttribute ProductRequestDto dto) {  String imageUrl = s3Uploader.upload(dto.getImage());  productService.save(dto.toEntity(imageUrl));  return ResponseEntity.ok("등록 완료");  }  // ProductForm.js (React)  const formData = new FormData();  formData.append("name", name);  formData.append("price", price);  formData.append("image", image);  axios.post("/api/admin/products", formData, { headers: { 'Content-Type': 'multipart/form-data' } });  **3) 장바구니 및 주문 기능 구현**   * 로그인한 사용자가 장바구니에 상품을 담고 수량 변경 가능 * 주문 시 쿠폰 및 포인트 사용, 총 결제 금액 자동 계산 * 주문 후 주문내역 페이지에서 목록 조회   @PostMapping("/order")  @Transactional  public ResponseEntity<?> placeOrder(@RequestBody OrderRequestDto dto, HttpSession session) {  Member member = (Member) session.getAttribute("loginMember");  orderService.createOrder(dto, member);  return ResponseEntity.ok("주문 완료");  }  **4) 쿠폰 기능 구현**   * 회원가입 시 자동 쿠폰 발급 * 관리자 페이지에서 쿠폰 등록 가능 (발행일 → 만료일 자동 계산) * 사용자 마이페이지에서 사용 가능한 쿠폰 조회 및 다운로드 처리   public Coupon createCoupon(CouponRequestDto dto) {  LocalDate expiredAt = dto.getCreatedAt().plusMonths(dto.getValidMonth());  return Coupon.builder()  .name(dto.getName())  .discountRate(dto.getDiscountRate())  .expiredAt(expiredAt)  .build();  }  **5) 이미지 업로드 처리**   * 초기엔 로컬 폴더(upload/)에 저장 * 저장된 이미지 URL은 DB에 저장하고, 프론트에서 경로로 접근   public String uploadToS3(MultipartFile file) {  String fileName = UUID.randomUUID() + "\_" + file.getOriginalFilename();  amazonS3Client.putObject(bucket, fileName, file.getInputStream(), metadata);  return amazonS3Client.getUrl(bucket, fileName).toString();  }  **6) 리뷰 / 문의 기능 구현**   * 사용자는 배송 완료된 주문 건에 대해 리뷰 작성 가능 * 리뷰는 마이페이지에서 목록 조회 및 수정 가능 * 관리자용 문의 목록 관리 페이지 구현 (사진 첨부 포함) | | **후기** | | 1. **전체적인 성장**  * 이번 쇼핑몰 프로젝트를 통해 프론트엔드와 백엔드를 연계한 실전형 풀스택 개발 역량을 키울 수 있었습니다. 특히 React를 활용한 컴포넌트 기반 UI 설계와 Spring Boot의 계층적 아키텍처를 함께 다루면서 개발 구조에 대한 이해와 실무 감각을 크게 향상시킬 수 있었습니다. * 또한 단순한 기능 구현이 아닌 로그인, 장바구니, 주문, 쿠폰, 파일 업로드, 관리자 기능 분리, 클라우드 배포까지 전 과정을 직접 다루면서 실제 서비스 흐름을 구현해낸 점에서 큰 성취감을 느꼈습니다.  1. **도전과 문제 해결**  * **1) 프론트엔드와 백엔드 간 CORS 이슈** * **문제 상황** React 프론트엔드에서 Spring Boot 백엔드로 로그인 요청을 보낼 때, CORS 정책에 의해 브라우저에서 요청이 차단됨. * **원인 분석** 브라우저 보안 정책에 따라, 서로 다른 도메인(포트) 간 요청 시 CORS 설정이 필요하며, 세션 쿠키를 주고받기 위해 credentials 허용이 필수였음**.** * **해결 방법**   프론트: axios 요청 시 withCredentials: true 옵션 추가  백엔드: @CrossOrigin(origins = "http://localhost:3000", allowCredentials = "true") 설정 추가  CORS 전역 설정 추가로 모든 API에서 적용되도록 확장   * **결과 및 배운 점** CORS는 단순히 허용만 한다고 해결되지 않으며, 쿠키·세션 기반 인증을 위한 추가 설정이 필요하다는 점을 체험함. 실무 웹 보안 정책의 이해도가 향상됨. * **2) 이미지 업로드 기능 구현 중 경로 및 파일명 충돌** * **문제 상황** 동일한 이름의 파일 업로드 시 이전 파일을 덮어쓰거나, 브라우저에서 이미지가 정상적으로 로딩되지 않음. * **원인 분석**   파일 이름 중복 발생  상대 경로 사용으로 인해 클라이언트에서 접근 경로가 일관되지 않음   * **해결 방법**   파일명에 UUID 접두어를 붙여 고유 이름 부여  upload.path를 application.properties에 정의하여 경로를 통합  프론트엔드에서는 이미지 URL을 <img src={imageUrl} />로 직접 렌더링   * **결과 및 배운 점** 파일 업로드는 단순 저장 이상의 문제(중복, 경로, 보안 등)를 포함하며, 파일명 정책과 저장소 분리가 중요함을 실감. 이후 AWS S3로 전환하여 문제 근본 해결**.** * **3) 세션 로그인 유지 실패 및 관리자 권한 분기 오류** * **문제 상황** 로그인은 되었지만 페이지 이동 후 사용자 정보가 사라지거나, 관리자가 아닌 사용자가 관리자 페이지에 접근 가능함. * **원인 분석**   프론트엔드에서 브라우저 세션을 유지하지 않거나, credentials 없이 요청  인터셉터에서 사용자 권한 체크가 누락됨   * **해결 방법**   Axios 요청마다 withCredentials: true 고정 적용  Spring에서 인터셉터로 모든 요청을 가로채고, 세션 정보 없으면 redirect  관리자 접근시 if (!member.getRole().equals("ADMIN")) 검사 추가   * **결과 및 배운 점** 세션 기반 인증은 프론트-백엔드 간 쿠키 통신을 전제로 하며, 역할 기반 접근 제어 로직은 프론트보다는 백엔드 중심으로 설계해야 보안이 강화됨을 확인. * **4) 쿠폰 만료일 계산 및 사용자 다운로드 오류** * **문제 상황** 발급된 쿠폰의 만료일이 잘못 계산되거나, 다운로드 후 중복 등록되는 현상 발생 * **원인 분석** 쿠폰 발급일 기준으로 만료일 계산 시 LocalDate 계산 로직 누락   중복 다운로드 체크 없이 insert 처리   * **해결 방법** expiredAt = createdAt.plusMonths(validMonth);로 계산식 명확화   UserCouponRepository**.existsByUserIdAndCouponId()로 중복 검사 후 처리**   * **결과 및 배운 점** 날짜 계산 로직과 사용자 조건 검사 등 작은 로직 하나가 전체 기능 신뢰도에 영향을 준다는 점을 인지하게 되었고, 향후엔 더 철저한 검증 로직을 설계할 예정  1. **배운 점과 개선점**  * DTO, Entity, Service의 분리 설계가 유지보수와 확장성에 얼마나 중요한지 체감했고, 다음 프로젝트에는 테스트 코드 작성까지 포함하여 더 견고하게 만들고 싶습니다. * React에서 상태가 많아질수록 useState만으로는 관리가 어려움을 느껴, 추후에는 Redux 또는 Context API 활용을 고려하고자 합니다. * 리뷰 기능, 관리자 페이지, 이미지 업로드 등 다양한 사용자 흐름을 처리하면서 실제 서비스의 구조와 유사한 프로젝트를 경험한 것은 향후 실무 투입에 있어서도 큰 밑거름이 될 것으로 생각합니다.  1. **향후 계획**  * 이번 프로젝트를 바탕으로 다음 단계에서는 다음과 같은 목표를 가지고 발전시키고자 합니다: * 결제 API(Toss 등) 연동 * Spring Security 기반 JWT 인증 구현 * CI/CD 도입 및 GitHub Actions 활용 자동 배포 실습 * 테스트 코드 작성 및 TDD 기반 개발 연습 | |