공공 및 산업 데이터

분석 서비스 개발

프로젝트

(공공 데이터를 활용한

Matching School 서비스)

클래스 및 패키지 설계 보고서

개발 1팀 : 서정은(팀장), 최승빈(부팀장),

문도연, 장정은, 서정은

승인자 : 훈련교사 김찬식

**※ 공공 및 산업 데이터 분석 서비스 개발 프로젝트**

**공공데이터를 활용한 Matching School 서비스: 클래스 및 패키지 설계 보고서**

**공통 WebConfig.java**

**package com.springboot.react.config;**

**// 필요한 스프링 프레임워크 클래스들을 임포트합니다.**

**import org.springframework.context.annotation.Configuration;**

**import org.springframework.web.servlet.config.annotation.CorsRegistry;**

**import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;**

**import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;**

**// 스프링 부트에서 CORS 설정을 적용하기 위해 WebMvcConfigurer 인터페이스를 구현합니다.**

**@Configuration // 이 클래스를 스프링 구성(설정) 클래스로 선언합니다.**

**public class WebConfig implements WebMvcConfigurer {**

**// CORS 설정을 추가하는 메소드입니다.**

**@Override**

**public void addCorsMappings(CorsRegistry registry) {**

**registry.addMapping("/\*\*") // 모든 경로에 대하여 CORS 정책을 적용합니다.**

**.allowedOrigins("\*","http://192.168.10.19:3000", "http://localhost:3000") // 접근을 허용할 출처를 지정합니다.**

**// .allowedOrigins("http://현재PC IP 주소 입력:3000", "http://localhost:3000") // 다른 출처를 추가할 수 있습니다.**

**.allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS") // 허용할 HTTP 메소드를 지정합니다.**

**.allowedHeaders("\*") // 모든 HTTP 헤더를 허용합니다.**

**.allowCredentials(false); // 자격 증명(예: 쿠키)을 포함한 요청을 허용합니다.**

**}**

**// 정적 리소스의 처리를 위한 핸들러를 추가하는 메소드입니다.**

**@Override**

**public void addResourceHandlers(ResourceHandlerRegistry registry) {**

**registry.addResourceHandler("/react\_images/\*\*") // '/react\_images/'로 시작하는 모든 요청을 처리합니다.**

**.addResourceLocations("file:///C:/react\_images/"); // 이 경로에서 리소스를 찾습니다.**

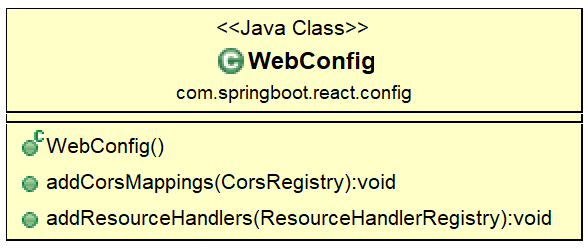
**registry.addResourceHandler("/downloadFile/\*\*")**

**.addResourceLocations("file:///C:/react\_images/");**

**}**

**}**

**WebConfig Diagram**

****

**공통 SecurityConfig.java**

**package com.springboot.react.config;**

**// 필요한 클래스들을 임포트합니다.**

**import java.util.Arrays;**

**import org.springframework.context.annotation.Bean;**

**import org.springframework.context.annotation.Configuration;**

**import org.springframework.security.config.annotation.web.builders.HttpSecurity;**

**import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;**

**import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;**

**import org.springframework.security.web.SecurityFilterChain;**

**import org.springframework.web.cors.CorsConfiguration;**

**@Configuration // 이 클래스를 스프링 구성(설정) 클래스로 선언합니다.**

**@EnableWebSecurity // 이 어노테이션을 사용하여 Spring Security 설정을 활성화합니다.**

**public class SecurityConfig {**

**@Bean // Spring IoC 컨테이너에 의해 관리되는 빈 객체를 생성합니다.**

**public BCryptPasswordEncoder passwordEncoder() {**

**// 비밀번호를 암호화하는 데 사용되는 BCryptPasswordEncoder의 인스턴스를 생성하고 반환합니다.**

**return new BCryptPasswordEncoder();**

**}**

**// SecurityFilterChain 객체를 생성하는 빈을 정의합니다.**

**// 이 빈은 HTTP 보안 구성을 담당합니다.**

**@Bean**

**public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {**

**http**

**.cors() // CORS (Cross-Origin Resource Sharing) 설정을 활성화합니다.**

**.configurationSource(request -> { // CORS 구성을 정의합니다.**

**CorsConfiguration config = new CorsConfiguration();**

**config.setAllowedOrigins(Arrays.asList("\*","http://localhost:3000", "http://192.168.10.19:3000")); // 접근을 허용할 출처 목록을 설정합니다.**

**config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS")); // 허용할 HTTP 메소드를 지정합니다.**

**config.setAllowedHeaders(Arrays.asList("\*")); // 허용할 HTTP 헤더를 지정합니다.**

**config.setAllowCredentials(false); // 자격 증명(쿠키 등)을 포함한 요청을 허용할지 여부를 설정합니다.**

**return config; // 구성된 CORS 설정을 반환합니다.**

**})**

**.and() // HttpSecurity 객체의 메소드 체인을 계속하기 위해 사용됩니다.**

**.csrf().disable(); // CSRF(Cross-Site Request Forgery) 보호 기능을 비활성화합니다.**

**// 여기에 추가적인 보안 구성을 정의할 수 있습니다.**

**return http.build(); // 구성된 HttpSecurity 객체를 반환합니다.**

**}**

**}**

**SecurityConfig.java Diagram**

****

**모델(Entity) 클래스 : Member.java**

**package com.springboot.react.entity;**

**// JPA 관련 어노테이션을 사용하기 위한 임포트**

**import javax.persistence.\*;**

**// Lombok 라이브러리의 어노테이션으로 getter, setter, 생성자를 자동으로 생성**

**import lombok.\*;**

**// 클래스를 데이터베이스의 엔티티로 선언합니다.**

**@Entity**

**// 엔티티에 해당하는 데이터베이스 테이블을 지정합니다.**

**@Table(name = "members")**

**// Lombok 어노테이션으로 getter 메소드 자동 생성**

**@Getter**

**// Lombok 어노테이션으로 setter 메소드 자동 생성**

**@Setter**

**// Lombok으로 기본 생성자 자동 생성**

**@NoArgsConstructor**

**// Lombok으로 모든 필드를 포함한 생성자 자동 생성**

**@AllArgsConstructor**

**public class Member {**

**// 식별자 필드를 선언하고 자동 생성 전략을 지정합니다. 시퀀스를 사용합니다.**

**@Id**

**@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "member\_seq")**

**@SequenceGenerator(name = "member\_seq", sequenceName = "MEMBER\_SEQ", allocationSize = 1)**

**private Long memberNumber;**

**// memberId 필드는 NULL을 허용하지 않습니다.**

**@Column(nullable = false)**

**private String memberId;**

**// memberPassword 필드는 NULL을 허용하지 않습니다.**

**@Column(nullable = false)**

**private String memberPassword;**

**// memberName 필드는 NULL을 허용하지 않습니다.**

**@Column(nullable = true)**

**private String memberName;**

**@Column(nullable = true)**

**private String memberPhone;**

**@Column(nullable = true)**

**private String address;**

**@Column(nullable = true)**

**private String addressDetail;**

**@Column(nullable = true)**

**private String gender;**

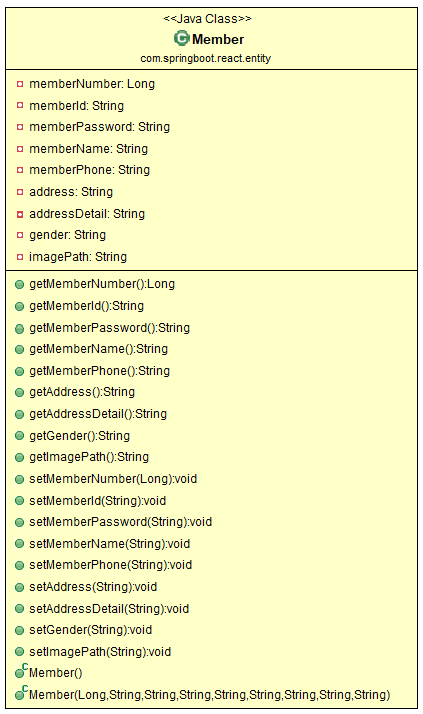
**// imagePath 필드는 NULL을 허용할 수 있으며, 이미지 파일의 저장 경로를 나타냅니다.**

**@Column(nullable = true)**

**private String imagePath; // 이미지 파일의 저장 경로**

**}**

**Member.java Diagram**

****

**리포지토리(Repository) 인터페이스 : MemberRepository.java**

**package com.springboot.react.repository;**

**// 필요한 클래스 임포트**

**import com.springboot.react.entity.Member; // Member 엔티티 클래스**

**import org.springframework.data.jpa.repository.JpaRepository; // Spring Data JPA의 JpaRepository**

**import java.util.Optional;**

**// MemberRepository 인터페이스는 JpaRepository를 확장합니다.**

**// JpaRepository에는 엔티티 클래스와 해당 ID 필드의 타입을 제네릭으로 지정합니다.**

**public interface MemberRepository extends JpaRepository<Member, Long> {**

**// 이 인터페이스는 JpaRepository를 상속받아 Member 엔티티에 대한 CRUD 및 페이징 처리를 자동으로 지원받습니다.**

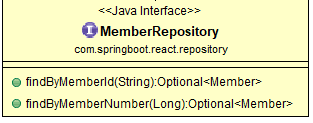
**// JpaRepository는 기본적인 CRUD 메서드와 페이징 및 정렬을 위한 메서드를 제공합니다.**

**// 여기에는 추가적인 메서드를 정의하지 않아도, Member 엔티티에 대한 기본적인 데이터베이스 작업이 가능합니다.**

**Optional<Member> findByMemberId(String memberId);**

**Optional<Member> findByMemberNumber(Long memberNumber);**

**}**

****

**서비스 인터페이스 : Memberservice.java**

**package com.springboot.react.service;**

**import com.springboot.react.entity.Member; // Member 엔티티 클래스**

**import java.util.Optional;**

**// MemberService 인터페이스 정의**

**public interface MemberService {**

**// saveMember 메소드는 Member 객체를 매개변수로 받아 데이터베이스에 저장합니다.**

**// 이 메소드는 Member 엔티티를 저장하거나 업데이트하는 로직을 처리할 수 있습니다.**

**// @param member Member 엔티티 인스턴스**

**// @return 저장된 Member 엔티티 인스턴스**

**Member saveMember(Member member);**

**// 특정 memberId를 사용하여 Member를 찾습니다.**

**Optional<Member> findByMemberId(String memberId);**

**// 특정 memberId를 사용하여 Member 정보를 업데이트합니다.**

**// @param memberId 업데이트할 사용자의 ID**

**// @param updatedMember 업데이트할 정보가 담긴 Member 객체**

**// @return 업데이트된 Member 객체**

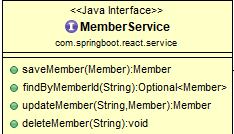
**Member updateMember(String memberId, Member updatedMember);**

**// 특정 memberId를 사용하여 회원 탈퇴(삭제)를 수행합니다.**

**// @param memberId 삭제할 사용자의 ID**

**void deleteMember(String memberId);**

**}**

****

**서비스 구현 클래스 : MemberServiceImpl.java**

**package com.springboot.react.service.impl;**

**import com.springboot.react.entity.Member;**

**import com.springboot.react.repository.MemberRepository;**

**import com.springboot.react.service.MemberService;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;**

**import org.springframework.stereotype.Service;**

**import org.springframework.transaction.annotation.Transactional;**

**import java.util.Optional;**

**@Service**

**public class MemberServiceImpl implements MemberService {**

**private final MemberRepository memberRepository;**

**private final BCryptPasswordEncoder passwordEncoder;**

**@Autowired**

**public MemberServiceImpl(MemberRepository memberRepository, BCryptPasswordEncoder passwordEncoder) {**

**this.memberRepository = memberRepository;**

**this.passwordEncoder = passwordEncoder;**

**}**

**@Override**

**@Transactional**

**public Member saveMember(Member member) {**

**member.setMemberPassword(passwordEncoder.encode(member.getMemberPassword()));**

**return memberRepository.save(member);**

**}**

**@Override**

**public Optional<Member> findByMemberId(String memberId) {**

**return memberRepository.findByMemberId(memberId);**

**}**

**@Override**

**@Transactional**

**public Member updateMember(String memberId, Member updatedMember) {**

**Member member =memberRepository.findByMemberId(memberId).orElseThrow(() -> new RuntimeException("Member not found with id " +memberId));**

**// 필드 업데이트**

**member.setMemberName(updatedMember.getMemberName());**

**member.setMemberPhone(updatedMember.getMemberPhone());**

**member.setAddress(updatedMember.getAddress());**

**member.setAddressDetail(updatedMember.getAddressDetail());**

**member.setGender(updatedMember.getGender());**

**member.setImagePath(updatedMember.getImagePath());**

**return updatedMember;**

**}**

**@Override**

**@Transactional**

**public void deleteMember(String memberId) {**

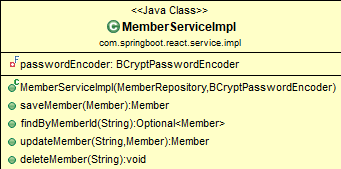
**Member member = memberRepository.findByMemberId(memberId)**

**.orElseThrow(() -> new RuntimeException("Member not found with id " + memberId));**

**memberRepository.delete(member); // 회원 삭제**

**}**

**}**

****

**컨트롤러(Controller) 클래스 : MemberController.java**

**package com.springboot.react.controller;**

**import com.springboot.react.entity.Member;**

**import com.springboot.react.service.MemberService;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.http.ResponseEntity;**

**import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;**

**import org.springframework.web.bind.annotation.\*;**

**import org.springframework.web.multipart.MultipartFile;**

**import org.springframework.web.servlet.support.ServletUriComponentsBuilder;**

**import com.fasterxml.jackson.databind.ObjectMapper;**

**import java.io.IOException;**

**import java.nio.file.Files;**

**import java.nio.file.Path;**

**import java.nio.file.Paths;**

**import java.nio.file.StandardCopyOption;**

**import java.util.Optional;**

**@RestController**

**@RequestMapping("/api/members")**

**public class MemberController {**

**private final MemberService memberService;**

**private final BCryptPasswordEncoder passwordEncoder;**

**@Autowired**

**public MemberController(MemberService memberService, BCryptPasswordEncoder passwordEncoder) {**

**this.memberService = memberService;**

**this.passwordEncoder = passwordEncoder;**

**}**

**private final Path rootLocation = Paths.get("C:/react\_images");**

**@PostMapping("/upload")**

**public ResponseEntity<?> uploadImage(@RequestParam("image") MultipartFile file,**

**@RequestParam("member") String memberStr) throws IOException {**

**ObjectMapper objectMapper = new ObjectMapper();**

**Member member = objectMapper.readValue(memberStr, Member.class);**

**String fileName = file.getOriginalFilename();**

**Files.copy(file.getInputStream(), this.rootLocation.resolve(fileName), StandardCopyOption.REPLACE\_EXISTING);**

**String fileDownloadUri = ServletUriComponentsBuilder.fromCurrentContextPath().path("/downloadFile/")**

**.path(fileName).toUriString();**

**member.setImagePath(fileDownloadUri);**

**Member savedMember = memberService.saveMember(member);**

**return ResponseEntity.ok("회원 가입 정보가 등록되었습니다. ID: " + savedMember.getMemberNumber());**

**}**

**@PostMapping**

**public Member saveMember(@RequestBody Member member) {**

**return memberService.saveMember(member);**

**}**

**@PostMapping("/login")**

**public ResponseEntity<?> login(@RequestBody Member loginRequest) {**

**Optional<Member> foundMember = memberService.findByMemberId(loginRequest.getMemberId());**

**if (!foundMember.isPresent()) {**

**return ResponseEntity.badRequest().body("로그인 아이디가 잘못 되었습니다!");**

**}**

**Member member = foundMember.get();**

**if (!passwordEncoder.matches(loginRequest.getMemberPassword(), member.getMemberPassword())) {**

**return ResponseEntity.badRequest().body("패스워드가 잘못 되었습니다!");**

**}**

**return ResponseEntity.ok(member);**

**}**

**@PutMapping("/update/{memberId}")**

**public ResponseEntity<?> updateMember(@PathVariable String memberId, @RequestParam("image") MultipartFile file,**

**@RequestParam("member") String memberStr)throws IOException {**

**ObjectMapper objectMapper = new ObjectMapper(); // JSON 매핑**

**Member memberToUpdate = objectMapper.readValue(memberStr, Member.class);**

**// 기존 게시글 조회**

**Member existingMember = memberService.findByMemberId(memberId)**

**.orElseThrow(() -> new RuntimeException("Member not found with id " + memberId));**

**String fileName = file.getOriginalFilename(); // 파일 이름 추출**

**Files.copy(file.getInputStream(), this.rootLocation.resolve(fileName), StandardCopyOption.REPLACE\_EXISTING);**

**// 파일 다운로드 URI 설정**

**String fileDownloadUri = ServletUriComponentsBuilder.fromCurrentContextPath().path("/downloadFile/")**

**.path(fileName).toUriString();**

**// Member 정보 업데이트**

**existingMember.setMemberName(memberToUpdate.getMemberName());**

**existingMember.setMemberPhone(memberToUpdate.getMemberPhone());**

**existingMember.setAddress(memberToUpdate.getAddress());**

**existingMember.setAddressDetail(memberToUpdate.getAddressDetail());**

**existingMember.setGender(memberToUpdate.getGender());**

**existingMember.setImagePath(fileDownloadUri);**

**Member updatedMember = memberService.saveMember(existingMember); // 수정된 게시글 저장 및 반환**

**return ResponseEntity.ok("게시글이 수정되었습니다. ID: " + updatedMember.getMemberNumber()); // 응답 반환**

**}**

**// 특정 ID로 게시글 조회 API**

**@GetMapping("/{memberId}")**

**public ResponseEntity<Member> getMemberByMemberId(@PathVariable String memberId) {**

**Optional<Member> member = memberService.findByMemberId(memberId); // ID로 게시글 검색**

**if (!member.isPresent()) { // 게시글이 존재하지 않을 경우**

**return ResponseEntity.notFound().build(); // 404 Not Found 응답 반환**

**}**

**return ResponseEntity.ok(member.get());**

**}**

**@DeleteMapping("/{memberId}")**

**public ResponseEntity<?> deleteMember(@PathVariable String memberId) {**

**try {**

**memberService.deleteMember(memberId);**

**return ResponseEntity.ok().body("회원 탈퇴가 성공적으로 처리되었습니다.");**

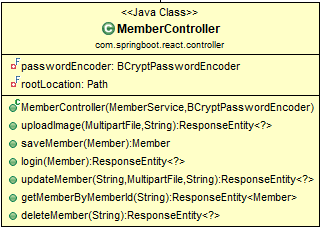
**} catch (RuntimeException e) {**

**return ResponseEntity.badRequest().body(e.getMessage());**

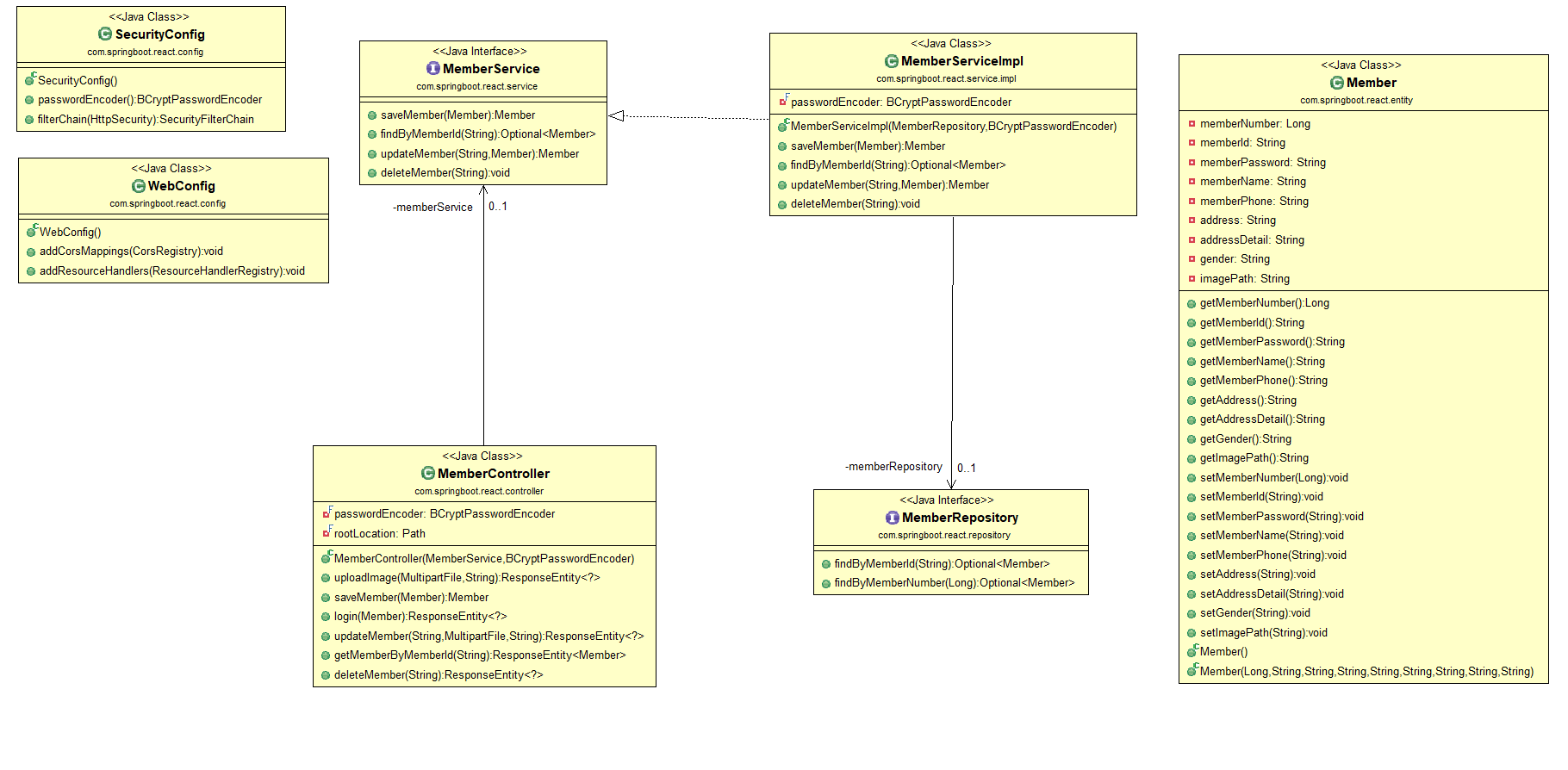
**}**

**}**

**}**

****

**Member Diagram**

****

**모델(Entity) 클래스 : Payment.java**

**package com.springboot.react.model;**

**import javax.persistence.\*;**

**import java.util.Date;**

**import lombok.Data;**

**@Data // Lombok 라이브러리를 사용하여 getter, setter, toString 등의 메소드를 자동으로 생성**

**@Entity // JPA 엔티티임을 나타냄**

**@Table(name = "pay\_import") // 엔티티와 매핑할 테이블 이름을 지정**

**public class Payment {**

**@Id // 기본 키(primary key)임을 나타냄**

**@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "payment\_seq\_gen") // 시퀀스를 사용하여 기본 키 값을 자동으로 생성**

**@SequenceGenerator(name = "payment\_seq\_gen", sequenceName = "idx\_seq", allocationSize = 1) // 시퀀스 제너레이터 설정, 시퀀스 이름과 할당 크기 설정**

**private Long pay\_id; // 결제 ID, 기본 키**

**private String imp\_uid; // 아임포트 결제 고유번호**

**private String merchant\_uid; // 가맹점 주문번호**

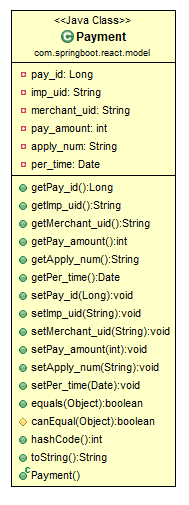
**private int pay\_amount; // 결제 금액**

**private String apply\_num; // 카드 승인번호**

**private Date per\_time; // 결제 시간**

**}**

**Payment Diagram**

****

**PaymentRepository.java**

**package com.springboot.react.repository;**

**import com.springboot.react.model.Payment;**

**import org.springframework.data.jpa.repository.JpaRepository;**

**import org.springframework.stereotype.Repository;**

**@Repository // 이 인터페이스가 스프링 데이터 JPA 레포지토리임을 나타냄**

**public interface PaymentRepository extends JpaRepository<Payment, Long> {**

**// JpaRepository를 상속받아 Payment 엔티티와 연동되는 레포지토리를 정의**

**// JpaRepository는 기본적인 CRUD 메소드(findAll, findById, save, delete 등)를 제공**

**// Payment 엔티티와 Long 타입의 ID를 사용**

**}**

**PaymentRepository Diagram**

****

**PaymentService.java**

**package com.springboot.react.service;**

**import com.springboot.react.model.Payment;**

**public interface PaymentService {**

**/\*\***

**\* 결제 성공 시 호출되는 메소드**

**\* 결제 정보를 데이터베이스에 저장하는 역할을 함**

**\***

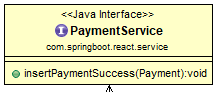
**\* @param payment 결제 정보를 담고 있는 Payment 객체**

**\*/**

**void insertPaymentSuccess(Payment payment);**

**}**

**PaymentService Diagram**

****

**PaymentServiceImpl.java  
package com.springboot.react.service.impl;**

**import com.springboot.react.model.Payment;**

**import com.springboot.react.repository.PaymentRepository;**

**import com.springboot.react.service.PaymentService;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.stereotype.Service;**

**import java.sql.Timestamp;**

**@Service**

**public class PaymentServiceImpl implements PaymentService {**

**// PaymentRepository 인스턴스를 자동 주입받아 사용**

**@Autowired**

**private PaymentRepository paymentRepository;**

**/\*\***

**\* 결제 정보를 데이터베이스에 저장하는 메소드**

**\***

**\* @param payment 결제 정보를 담고 있는 Payment 객체**

**\*/**

**@Override**

**public void insertPaymentSuccess(Payment payment) {**

**// 결제 정보를 로그에 출력하여 저장 시도를 확인**

**System.out.println("결제 정보 저장 시도: " + payment);**

**// 현재 시간을 결제 시간으로 설정**

**payment.setPer\_time(new Timestamp(System.currentTimeMillis()));**

**// Payment 객체를 데이터베이스에 저장**

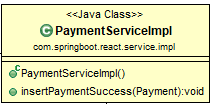
**paymentRepository.save(payment);**

**// 저장 성공 메시지를 로그에 출력**

**System.out.println("결제 정보 저장 성공");**

**}**

**}  
PaymentServiceImpl Diagram**

****

**PaymentController.java**

**package com.springboot.react.controller;**

**import com.springboot.react.model.Payment;**

**import com.springboot.react.service.PaymentService;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.http.ResponseEntity;**

**import org.springframework.web.bind.annotation.\*;**

**@RestController**

**@RequestMapping("/api/payment")**

**public class PaymentController {**

**// PaymentService 인스턴스를 자동 주입받아 사용**

**@Autowired**

**private PaymentService paymentService;**

**/\*\***

**\* 결제 정보를 처리하는 엔드포인트**

**\***

**\* @param payment 결제 정보를 담고 있는 Payment 객체**

**\* @return 결제 처리 결과를 포함한 ResponseEntity 객체**

**\*/**

**@PostMapping("/process")**

**public ResponseEntity<?> paymentDone(@RequestBody Payment payment) {**

**try {**

**// 결제 정보를 로그에 출력하여 수신 확인**

**System.out.println("Received payment info: " + payment);**

**// 결제 정보를 데이터베이스에 저장**

**paymentService.insertPaymentSuccess(payment);**

**// 결제 처리 성공 메시지를 포함한 ResponseEntity 객체 반환**

**return ResponseEntity.ok().body("{\"message\": \"Payment processed successfully\"}");**

**} catch (Exception e) {**

**// 결제 처리 중 발생한 오류를 로그에 출력**

**System.err.println("결제 처리 오류: " + e.getMessage());**

**// 결제 처리 실패 메시지를 포함한 ResponseEntity 객체 반환**

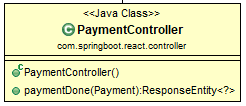
**return ResponseEntity.badRequest().body("{\"message\": \"Failed to process payment\"}");**

**}**

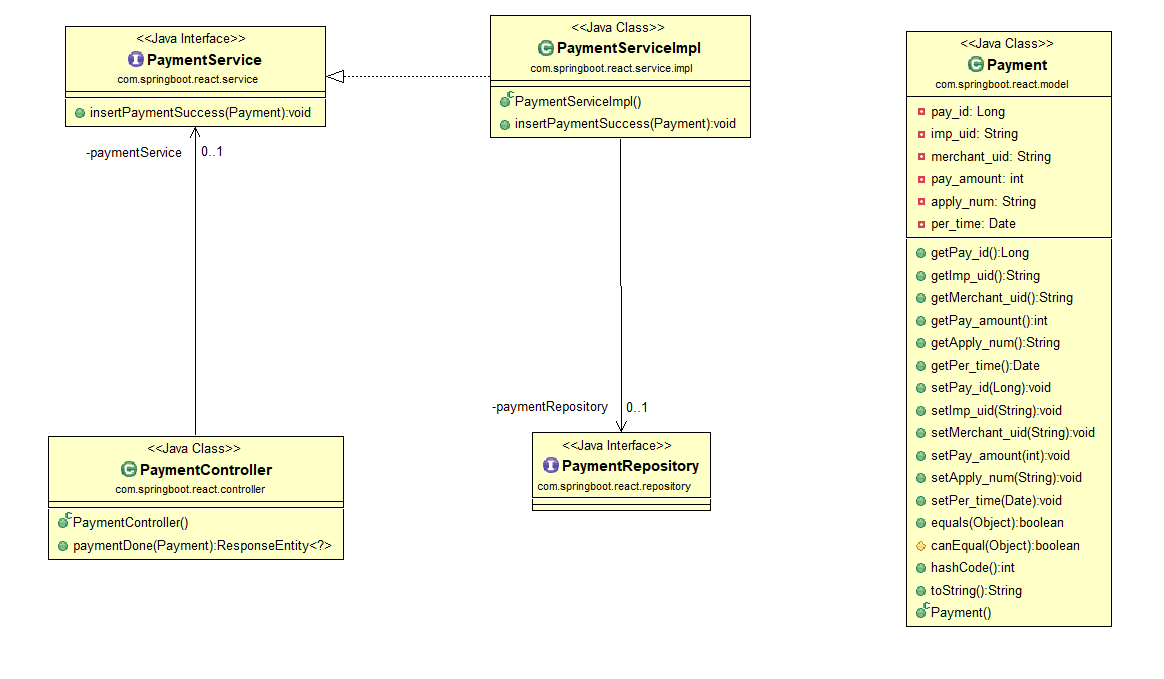
**}**

**}**

**PaymentController.java Diagram**

****

**Payment Diagram**

****