**2025년 멋쟁이사자 프로젝트**

|  |
| --- |

**[창조스택\_PM\_001]**

**개발표준안**

**Ver. 1.1**

Revision History

| **날짜** | **버전** | **작성 / 변경 / 추가 내역** | **변경 요청자** | **승인자** |
| --- | --- | --- | --- | --- |
| 2025.07.14 | 1.0 | 문서 최초 작성 |  |  |
| 2025.07.16 | 1.1 | 디렉토리 및 파일 명명규칙 변경 |  |  |
| 20205.07.23 | 1.2 | 소스코드 추가 |  |  |

[**1. 개발 환경 iii**](#_heading=h.wbu39ria3bv0)

[**1.**](#_heading=h.ahjtfshtl6qq) **개발툴** iii

[**2.**](#_heading=h.qx7vy0ogcu7k) **버전관리계획** iii

[**3.**](#_heading=h.nmrjy36znhca) **데이터베이스 접근 계획** iii

[**2. 명명 규칙 iii**](#_heading=h.fqrdyz9f2q7j)

[**1.**](#_heading=h.7doonaq2n0ay) **디렉토리 명명 규칙** iii

[**2.**](#_heading=h.s78omyte0m05) **HTML 파일 명명 규칙** iii

[**4.**](#_heading=h.im5ikp2sqem0) **자바스크립트 XML 파일 명명 규칙** vi

[**5.**](#_heading=h.1b13em1j48xr) **클래스 명명 규칙** vi

[**6.**](#_heading=h.ieylkj4y6500) **객체 및 변수 명명 규칙** vii

[**7.**](#_heading=h.6wz7i3g02lst) **데이터베이스 명명 규칙** vii

[**3. 개발 구조 vii**](#_heading=h.jtacyiot8iwd)

[**1.**](#_heading=h.iohst6cs6m6w) **디렉토리 구조** viii

[**4. 코드 구조 viii**](#_heading=h.5yt2bihnv2jl)

[**1.**](#_heading=h.br4g9qokg4qb) **JAVA 코드 구조 (MVC단위로, front & back)** viii

[**2.**](#_heading=h.47ywltqim8ml) **React 구조** ix

# 1. 개발 환경

1. **개발툴**
   * Visual Studio Code 1.101.2를 기본 개발툴로 사용한다.
   * MySQL Community Server 8.0.41을 Database 관리툴로 사용한다.
   * Java 21.0.7 2025-04-15 LTS을 백엔드 기본 언어로 사용한다.
   * Spring-boot 3.5.3을 백엔드 기본 Framework로 사용한다.
   * React 19.1.0을 프론트엔드 기본 라이브러리로 사용한다.
   * npm 11.4.2를 프론트엔드 빌드 도구로 사용한다.
   * mvn 3.9.10을 백엔드 빌드 도구로 사용한다.
2. **버전관리계획**
   * 형상관리 툴인 Git(version 2.48.1)과 Git GUI인 SourceTree를 이용하여 단위 개발 및Version

관리를 하며 서버는 GitHub를 이용한다.

* + 개발자 로컬 PC의 웹환경을 기반으로 개발하는 것을 기본으로 하며, 타 개발자 작업 내용의

참조가 요구되는 경우에는 GitHub을 이용하여 해당 작업 내용을 개발자 로컬 PC로

이전받아 참조하며, 공용 라이브러리의 확장은 최종 개발 책임자에게 제작을 의뢰한다.

* + 테스트 서버의 관리 및 작업 내용 반영은 최종 개발 책임자가 작업한다.
  + 단위 개발 작업이 완료되면, 최종 개발 책임자는 해당 결과물을 받아 테스트 서버에 설치한다.
  + 개발 진행중인 작업 내용의 경우 개인 단위의 백업관리가 이루어지며, 개발 완료 작업

내용과 일(daily)단위의 백업은 최종 개발 책임자가 맡는다.

1. **데이터베이스 접근 계획**
   * 모든 개발자는 하나의 데이터베이스 서버(테스트 혹은 운영서버)를 공유하여 사용한다.
   * 데이터베이스의 접근은 WAS의 자체 Database Pooling모듈을 사용하며, DB 리소스는

독립적으로 획득할 수 없다.

* + JPA를 이용하여 메인서버와 DB서버간 데이터 전달을 제어한다.
  + MySQL 8.0.41의 커넥션 풀 설정은 spring.datasource를 통해 구성한다.

# 2. 명명 규칙

1. **디렉토리 명명 규칙**
   * 디렉토리 명칭은 영문으로 사용한다.
   * groupId에 정의된 패키지 깊이를 기준으로 디렉토리를 생성하며, 약어를 사용하지 않는다.
2. **HTML 파일 명명 규칙**
   * 해당 디렉토리 명칭 말머리로 갖고 파일의 주요 기능을 표기한다.

| XXXXX-YYYYY.html  ⓐ ⓑ ⓒ | | |
| --- | --- | --- |
| ⓐ | 디렉토리 명칭 | 역할별 분류   1. 공통 UI : components 2. 페이지 단위 : pages 3. API 통신: api 4. 정적 파일 : assets 5. 공통 로직 함수: utils 6. 페이지 공통 구조 : layouts 7. 라우팅 설정 : routes 8. 전역 스타일 설정 : styles |
| ⓑ | 구분자 | 연결 단어 첫 글자를 대문자로 시작 |
| ⓒ | 주요 기능 | 1) 게시판 : write(글작성), rewrite(수정), list(글목록), detail(글상세), delete(글삭제)  2) 댓글 : write(댓글작성), rewrite(댓글수정), list(댓글목록), delete(댓글삭제)  3) 회원: auth(인증), signup(회원생성), login(), logout()  4) 마이페이지 : auth(인증), subscriber-view(본인정보조회), creator-view(창작자정보조회), edit(정보수정)  5) 결제 : payment(결제진행), list(결제목록), cancel(결제예약취소)  6) 구독 : subscribe(구독생성), cancel(구독취소), list(구독목록) |

* 구분자를 제외한 모든 섹션은 영문자로 시작하며, 가급적 의미를 알 수 없는 약어를 사용하지 않는다.
* 페이지 구성은 모두 React 컴포넌트(JSX)로 구현한다.
  + 컴포넌트 파일명은 해당 기능과 역할을 명확히 알 수 있도록 의미 있는 단어로 작성한다.
  + 파일명은 첫 글자를 대문자로 하는 CamelCase 방식으로 작성하며, 특수 문자(-,\_, 공백 등)는 사용하지 않는다.
  + 각 컴포넌트는 하나의 파일로 구성하며, 컴포넌트 이름과 파일 이름은 동일하게 유지한다.
  + 공통 UI 컴포넌트는 components/ 디렉토리, 페이지 단위 컴포넌트는 pages/ 디렉토리에 저장한다.
  + 스타일 파일은 컴포넌트명.module.css로 작성한다.
  + 파일 확장자는 JavaScript 기반이므로 .jsx를 사용한다.
* React에서는 별도의 HTML 파일을 다수 생성하지 않고, 보통 public/index.html 한 개만 사용한다.
* 사용되는 확장자는 아래와 같다.

| **파일 형식** | **파일 확장자** |
| --- | --- |
| React Component(JSX) | .jsx |
| Cascading Style Sheet(CSS) | .css |
| JavaScript | .js |
| Hyper Text Markup Language | .html |
| Web Resource | .jpg, .svg, .png, .bmp, |
| JAVA | .java |
| Config file | application.yml, .env, pom.xml 등 |

1. **텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

   AI가 생성한 콘텐츠는 부정확할 수 있습니다.패키지 명명 규칙 및 자바 파일 명명 규칙**
   * 패키지명은 3자리 이상으로 정의한다.
     + com.creationstack.backend.auth : jwt, auth 파일 정의
     + com.creationstack.backend.config : 설정 파일 정의
     + com.creationstack.backend.controller : controller 파일 정의
     + com.creationstack.backend.domain : domain/enum 파일 정의
     + com.creationstack.backend.dto : dto 파일 정의
     + com.creationstack.backend.etc : 기타 파일 정의
     + com.creationstack.backend.exception : 예외처리 파일 정의
     + com.creationstack.backend.repository : repository 파일 정의
     + com.creationstack.backend.service : service 파일 정의
   * 자바 파일은 3자리 이상의 package명 하위에 작성한다.
2. **자바스크립트 XML 파일 명명 규칙**
   * 업무 및 기능 단위로 영문으로 명명하며 약어는 지양한다.
     + login : 로그인
     + slide : 이미지 슬라이드
     + modal : 모달
     + signUpFormCheck : 회원가입시 유효성 검사
     + navigo : 네비게이션바

예) mypageValidation.js : 마이페이지에서 유효성검사 기능에 사용되는 자바스크립트 파일

1. **클래스 명명 규칙**

* Domain의 경우 해당 데이터베이스 테이블명칭을 그대로 갖는 클래스 이름을 사용하며,

com.creationstack.backend.domain 을 상위 패키지로 갖는다.

예1) *com.creationstack.backend.domain.Academy.class*

예2) *com.creationstack.backend.domain.Board.class*

* DTO의 경우 사용되는 목적에 따라 Dto 앞에 Request/Response를 붙여서 클래스를 명명하며 com.creationstack.backend.dto를 상위 패키지로 갖는다.

예 1) com.creationstack.backend.dto.AcademyRequestDto

예 2) com.creationstack.backend.dto.RateResponseDto

* Repository, Controller, Service의 경우 기본적으로 해당 domain명칭에 각각 Repository, Controller, Service를 붙여서 클래스를 명명하며, 각각 com.creationstack.backend.repository, com.creationstack.backend.controller, com.creationstack.backend.service를 상위 패키지로 갖는다.

예 1) com.creationstack.backend.repository.AcademyRepository.class

예 2) com.creationstack.backend.controller.AcademyController.class

예 3) com.creationstack.backend.service.AcademyService.class

1. **객체 및 변수 명명 규칙**

* 객체의 경우 해당 객체가 페이지 내에서 유일할 경우 첫글자를 소문자로 바꾼 클래스명을 그대로 사용한다.

- 예1) 클래스 AcademyService의 객체 *academyService*

- 예 2) 클래스 CourseRepository의 객체 *courseRepository*

* 변수의 경우 단순 증감이며 for문 등의 제어문 내에서 선언되는 경우 <i,j,k> 등의 간단한 소문자를 사용한다. 또한 임시적인 변수로 <temp>를 사용한다.

- 예 1) *for(int j = 0; j <courses.size(); j++)*

- 예 2) *for(int temp = 0; temp< nList.getLength(); temp++)*

* 특정 정보를 담고 있는 변수인 경우, 해당 정보의 성격을 나타낼 수 있는 명칭을 사용한다

- 예) String *userId* = “사용자 ID 정보”;

- HTML내의 <input> 태그로 사용되는 필드 값을 전달받을 경우 동일한 이름의 변수에 할당한다.

1. **데이터베이스 명명 규칙**

* 테이블 명칭 : 테이블의 용도를 알수있도록 명명한다. 단, MySQL의 예약어(USER, ORDER, GROUP 등)와 중복될 경우에는 복수형 명칭을 사용하여 충돌을 방지한다.

예 1) 게시판 테이블 : BOARD

예 2) 사용자 테이블 : USERS

예 3) 학원 정보 테이블 : ACADEMY

* 필드 명칭: 외부참조키가 아닌 경우 <TABLE NAME>+’\_’를 붙이는 형태로 사용하며, 외부참조키(FK)인 경우는 해당 해당 FK의 원본 필드 명칭을 그대로 사용한다.

예 1) 게시판 테이블의 ID : BOARD\_ID

예 2) 댓글 테이블의 FK(게시판 ID) : BOARD\_ID

# 3. 개발 구조

1. **디렉토리 구조**

* 경로는 클래스 및 인터페이스 파일의 경우 <groudId> ‘com.gukbit’을 상위 패키지로 갖고, 그 외 파일들은resources를 상위로 갖는다.
* Directory 이름은 의미있는 하나의 영문 단어로 구성하고 전부 소문자로 표기하도록 함. 또한 혼동을 피하기 위해 가급적 약어를 사용하지 않도록 함.
* 다음은 백엔드 디렉토리 구조이다.
  + - JAVA Class 파일 : <groupId> ‘com.creationstack.backend + <package>
    - JAVA Source 파일 : <groupId> ‘com.creationstack.backend’ + <package>
    - JAVA Test 파일 : ‘/test/java’ + <groupId> + ‘com.creationstack.backend’
* 다음은 프론트엔드 디렉토리 구조이다.
  + - JavaScript XML 파일 : ‘/src/pages + ‘/페이지명’, ‘/src/layouts’, ‘src/components’ + ‘/기능명’
    - 정적 파일 : ‘/src/assets/images’
    - API 통신 파일 : ‘/src/api’
    - 비즈니스 로직 파일 : ‘/src/pages’ + ‘/페이지명’
    - 공통 유틸 함수 파일 : ‘/src/utils’
    - CSS 파일 : ‘/src/pages’ + ‘/페이지명’, ‘/src/components‘ + ‘/기능명’, ‘src/styles’

# 4. 코드 구조

1. **JAVA 코드 구조 (MVC단위로, front & back)**
   * Repository 구조 예제

| package com.creationstack.backend.repository;  import java.util.List;  import java.util.Optional;  import org.springframework.data.jpa.repository.JpaRepository;  import org.springframework.stereotype.Repository;  import com.creationstack.backend.domain.user.Job;  import com.creationstack.backend.domain.user.JobType;  @Repository  public interface JobRepository extends JpaRepository<Job, Integer> {  List<Job> findAllByOrderByNameAsc();  boolean existsByName(String name);  boolean existsByJobType(JobType jobType);  Optional<Job> findByName(String name);  Optional<Job> findByJobType(JobType jobType);  } |
| --- |

* + Service 구조 예제

| package com.creationstack.backend.service;  import java.time.LocalDateTime;  import java.util.List;  import java.util.Map;  import java.util.stream.Collectors;  import org.springframework.http.HttpStatus;  import org.springframework.stereotype.Service;  import com.creationstack.backend.domain.payment.Payment;  import com.creationstack.backend.domain.payment.PaymentMethod;  import com.creationstack.backend.domain.payment.PaymentStatus;  import com.creationstack.backend.domain.subscription.Subscription;  import com.creationstack.backend.domain.subscription.SubscriptionStatus;  import com.creationstack.backend.domain.subscription.SubscriptionStatusName;  import com.creationstack.backend.domain.user.User;  import com.creationstack.backend.dto.Subscription.SubscriptionRequestDto;  import com.creationstack.backend.dto.Subscription.SubscriptionResponseDto;  import com.creationstack.backend.dto.Subscription.UserSubscriptionDto;  import com.creationstack.backend.dto.member.PublicProfileResponse;  import com.creationstack.backend.etc.Role;  import com.creationstack.backend.exception.CustomException;  import com.creationstack.backend.repository.PaymentMethodRepository;  import com.creationstack.backend.repository.PaymentRepository;  import com.creationstack.backend.repository.SubscriptionRepository;  import com.creationstack.backend.repository.SubscriptionStatusRepository;  import com.creationstack.backend.repository.UserRepository;  import org.springframework.transaction.annotation.Transactional;  import lombok.RequiredArgsConstructor;  import lombok.extern.slf4j.Slf4j;  @Service  @RequiredArgsConstructor  @Slf4j  public class SubscriptionService {  private final SubscriptionRepository subscriptionRepository;  private final SubscriptionStatusRepository statusRepository;  private final PaymentMethodRepository paymentMethodRepository;  private final PaymentRepository paymentRepository;  private final UserRepository userRepository;  // 구독 생성 (PENDING 상태)  @Transactional  public SubscriptionResponseDto createPendingSubscription(Long subscriberId, SubscriptionRequestDto request) {  *log*.info("구독 생성 시작");  // 1. 구독 상태 PENDING 로딩  SubscriptionStatus pendingStatus = statusRepository.findByName("PENDING")  .orElseThrow(() -> new CustomException(HttpStatus.*INTERNAL\_SERVER\_ERROR*,  "PENDING 상태를 찾을 수 없습니다"));  // 2. 결제 수단 존재 확인  *log*.info("요청 userId: {}", subscriberId);  *log*.info("결제 수단 ID: {}", request.getPaymentMethodId());  PaymentMethod paymentMethod = paymentMethodRepository.findById(request.getPaymentMethodId())  .orElseThrow(() -> new CustomException(HttpStatus.*BAD\_REQUEST*, "결제 수단 정보가 없습니다."));  LocalDateTime now = LocalDateTime.*now*();  // 3. 자기 자신에게 구독 불가  if (subscriberId.equals(request.getCreatorId())) {  throw new CustomException(HttpStatus.*BAD\_REQUEST*, "자기 자신을 구독할 수 없습니다.");  }  // 4. 대상 크리에이터가 크리에이터인지 확인  User creator = userRepository.findById(request.getCreatorId())  .orElseThrow(() -> new CustomException(HttpStatus.*NOT\_FOUND*, "대상 사용자가 존재하지 않습니다."));  *log*.info("대상 role: {}", creator.getRole());  *log*.info("Role.CREATOR: {}", Role.*CREATOR*);  *log*.info("creator.getRole() 클래스: {}", creator.getRole().getClass().getName());  if (!"CREATOR".equals(creator.getRole().name())) {  throw new CustomException(HttpStatus.*BAD\_REQUEST*, "해당 사용자는 크리에이터가 아닙니다.");  }  // 5. 기존 구독 확인 및 처리  Subscription existing = subscriptionRepository  .findBySubscriberIdAndCreatorId(subscriberId, request.getCreatorId())  .orElse(null);  Subscription subscription;  if (existing != null) {  if (SubscriptionStatusName.*ACTIVE*.equals(existing.getStatus().getName())) {  throw new CustomException(HttpStatus.*CONFLICT*, "이미 활성화된 구독이 있습니다.");  }  // 기존 구독이 있으나 ACTIVE는 아님 → 상태 유지 (변경 없음), 이후 결제 성공 시 상태 갱신  subscription = existing;  } else {  subscription = new Subscription();  subscription.setSubscriberId(subscriberId);  subscription.setAmount(4900);  subscription.setCreatorId(request.getCreatorId());  subscription.setStatus(pendingStatus);  subscription.setStartedAt(now);  subscription.setNextPaymentAt(now.plusMonths(1));  subscription.setPaymentMethod(paymentMethod);  subscriptionRepository.save(subscription);  }  return SubscriptionResponseDto.*builder*()  .subscriptionId(subscription.getSubscriptionId())  .creatorId(subscription.getCreatorId())  .statusName(subscription.getStatus().getName())  .startedAt(subscription.getStartedAt())  .nextPaymentAt(subscription.getNextPaymentAt())  .lastPaymentAt(subscription.getLastPaymentAt())  .message("구독 요청이 접수되었습니다 (PENDING)")  .build();  }  // 결제 성공 후 구독 활성화  @Transactional  public void activateSubscription(Long subscriptionId, Long paymentId) {  Subscription subscription = subscriptionRepository.findById(subscriptionId)  .orElseThrow(() -> new CustomException(HttpStatus.*NOT\_FOUND*, "구독 정보를 찾을 수 없습니다."));  SubscriptionStatus activeStatus = statusRepository.findByName("ACTIVE")  .orElseThrow(() -> new CustomException(HttpStatus.*INTERNAL\_SERVER\_ERROR*,  "ACTIVE 상태 정보가 없습니다."));  Payment payment = paymentRepository.getReferenceById(paymentId);  if (payment.getPaymentStatus() != PaymentStatus.*SUCCESS*) {  throw new CustomException(HttpStatus.*BAD\_REQUEST*, "결제가 실패하였습니다.");  }  subscription.setStatus(activeStatus);  subscription.setLastPaymentAt(payment.getSuccessAt());  subscription.setPaymentMethod(payment.getPaymentMethod());  // 기존 nextPaymentAt이 없을 수도 있으니 null check  subscription.setNextPaymentAt(  subscription.getNextPaymentAt() == null ? subscription.getStartedAt().plusMonths(1)  : subscription.getNextPaymentAt());  subscriptionRepository.save(subscription);  }  // 결제 실패 처리: 이전 구독이면 EXPIRED, 신규면 삭제  @Transactional  public void handleSubscriptionFailure(Long subscriptionId) {  Subscription subscription = subscriptionRepository.findById(subscriptionId)  .orElseThrow(() -> new CustomException(HttpStatus.*NOT\_FOUND*, "해당 구독을 찾을 수 없습니다."));  String statusName = subscription.getStatus().getName();  if (SubscriptionStatusName.*PENDING*.equals(statusName)) {  // 결제 전 신규 구독 실패 → 삭제  subscriptionRepository.delete(subscription);  return;  }  // 기존 구독이었던 경우 → EXPIRED 처리  SubscriptionStatus expiredStatus = statusRepository.findByName(SubscriptionStatusName.*EXPIRED*)  .orElseThrow(() -> new CustomException(HttpStatus.*INTERNAL\_SERVER\_ERROR*,  "EXPIRED 상태를 찾을 수 없습니다."));  subscription.setStatus(expiredStatus);  subscriptionRepository.save(subscription);  }  // 구독 여부 조회  @Transactional(readOnly = true)  public boolean isActiveSubscriber(Long creatorId, Long SubscriberId) {  SubscriptionStatus activeStatus = statusRepository.findByName("ACTIVE")  .orElseThrow(() -> new CustomException(HttpStatus.*INTERNAL\_SERVER\_ERROR*,  "ACTIVE 상태 정보가 없습니다."));  return subscriptionRepository.existsByCreatorIdAndSubscriberIdAndStatus(  creatorId, SubscriberId, activeStatus);  }  // 사용자의 구독 목록 조회  @Transactional  public List<UserSubscriptionDto> getMySubscriptions(Long userId) {  List<UserSubscriptionDto> list = subscriptionRepository.findAllBySubscriberId(userId);  if (list.isEmpty()) {  return list;  }  List<Long> creatorIds = list.stream().map(UserSubscriptionDto::getCreatorId).distinct().toList();  List<Object[]> counts = subscriptionRepository.countActiveSubscriptionsByCreatorIds(creatorIds);  Map<Long, Long> subsCountMap = counts.stream()  .collect(Collectors.*toMap*(row -> (Long) row[0], row -> (Long) row[1]));  list.forEach(dto -> {  dto.setSubsCount(subsCountMap.getOrDefault(dto.getCreatorId(), 0L));  dto.setMessage(switch (dto.getStatusName()) {  case "ACTIVE" -> "다음 결제 예정일: " + format(dto.getNextPaymentAt());  case "CANCELLED" -> "만료 예정일: " + format(dto.getNextPaymentAt());  case "EXPIRED" -> "만료된 구독입니다.";  case "PENDING" -> "결제 대기 중입니다.";  default -> "";  });  });  return list;  }  private String format(LocalDateTime dt) {  return dt != null ? dt.toLocalDate().toString() : "";  }  // 사용자가 구독한 크리에이터 목록 조회  @Transactional(readOnly = true)  public List<PublicProfileResponse> getSubscribedCreators(String nickname) {  List<PublicProfileResponse> list = subscriptionRepository.findSubscribedCreatorsByNickname(nickname);  if (list.isEmpty()) {  return list;  }  List<Long> creatorIds = list.stream().map(PublicProfileResponse::getUserId).distinct().toList();  List<Object[]> counts = subscriptionRepository.countActiveSubscriptionsByCreatorIds(creatorIds);  Map<Long, Long> subsCountMap = counts.stream()  .collect(Collectors.*toMap*(row -> (Long) row[0], row -> (Long) row[1]));  list.forEach(dto -> dto.setSubsCount(subsCountMap.getOrDefault(dto.getUserId(), 0L)));  return list;  }  @Transactional  public void cancelSubscription(Long subscriptionId, Long userId) {  Subscription subscription = subscriptionRepository.findById(subscriptionId)  .orElseThrow(() -> new CustomException(HttpStatus.*NOT\_FOUND*, "구독 정보를 찾을 수 없습니다."));  if (!subscription.getSubscriberId().equals(userId)) {  throw new CustomException(HttpStatus.*FORBIDDEN*, "해당 구독을 해지할 권한이 없습니다.");  }  SubscriptionStatus cancelledStatus = statusRepository.findByName("CANCELLED")  .orElseThrow(() -> new CustomException(HttpStatus.*INTERNAL\_SERVER\_ERROR*,  "CANCELLED 상태 정보를 찾을 수 없습니다."));  subscription.setStatus(cancelledStatus);  // 크리에이터의 구독자 수 감소  User creator = userRepository.findById(subscription.getCreatorId())  .orElseThrow(() -> new CustomException(HttpStatus.*NOT\_FOUND*, "크리에이터를 찾을 수 없습니다."));  creator.setSubscriberCount(Math.*max*(0, creator.getSubscriberCount() - 1));  userRepository.save(creator);  }  } |
| --- |

* + Controller 구조 예제

| package com.creationstack.backend.controller;  import java.util.stream.Collectors;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.security.core.Authentication;  import org.springframework.validation.BindingResult;  import org.springframework.web.bind.annotation.DeleteMapping;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.RequestBody;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.RestController;  import com.creationstack.backend.dto.member.EmailCheckResponse;  import com.creationstack.backend.dto.member.NicknameCheckResponse;  import com.creationstack.backend.dto.member.SignupRequest;  import com.creationstack.backend.dto.member.SignupResponse;  import com.creationstack.backend.service.AuthService;  import com.creationstack.backend.service.UserService;  import jakarta.validation.Valid;  import lombok.RequiredArgsConstructor;  import lombok.extern.slf4j.Slf4j;  @RestController  @RequestMapping("/api/user")  @RequiredArgsConstructor  @Slf4j  public class UserController {  private final AuthService authService;  @PostMapping  public ResponseEntity<SignupResponse> signup(@Valid @RequestBody SignupRequest request,  BindingResult bindingResult) {  // 유효성 검사 오류 처리  if (bindingResult.hasErrors()) {  String errorMessage = bindingResult.getFieldErrors().stream()  .map(error -> error.getDefaultMessage())  .collect(Collectors.*joining*(", "));  SignupResponse response = SignupResponse.*builder*()  .success(false)  .message("입력 정보가 올바르지 않습니다: " + errorMessage)  .build();  return ResponseEntity.*badRequest*().body(response);  }  try {  SignupResponse response = authService.signup(request);  if (response.isSuccess()) {  return ResponseEntity.*status*(HttpStatus.*CREATED*).body(response);  } else {  return ResponseEntity.*badRequest*().body(response);  }  } catch (IllegalArgumentException e) {  *log*.error("회원가입 실패: {}", e.getMessage());  SignupResponse response = SignupResponse.*builder*()  .success(false)  .message(e.getMessage())  .build();  return ResponseEntity.*badRequest*().body(response);  } catch (Exception e) {  *log*.error("회원가입 중 예상치 못한 오류 발생", e);  SignupResponse response = SignupResponse.*builder*()  .success(false)  .message("서버 오류가 발생했습니다. 잠시 후 다시 시도해주세요.")  .build();  return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*).body(response);  }  }  @GetMapping("/check-email")  public ResponseEntity<EmailCheckResponse> checkEmailDuplicate(@RequestParam String email) {  try {  boolean isAvailable = authService.isEmailAvailable(email);  EmailCheckResponse response = EmailCheckResponse.*builder*()  .success(true)  .available(isAvailable)  .message(isAvailable ? "사용 가능한 이메일입니다." : "이미 사용 중인 이메일입니다.")  .build();  return ResponseEntity.*ok*(response);  } catch (Exception e) {  *log*.error("이메일 중복 확인 중 오류 발생", e);  EmailCheckResponse response = EmailCheckResponse.*builder*()  .success(false)  .available(false)  .message("이메일 중복 확인 중 오류가 발생했습니다.")  .build();  return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*).body(response);  }  }  @GetMapping("/check-nickname")  public ResponseEntity<NicknameCheckResponse> checkNicknameDuplicate(@RequestParam String nickname) {  try {  boolean isAvailable = authService.isNicknameAvailable(nickname);  NicknameCheckResponse response = NicknameCheckResponse.*builder*()  .success(true)  .available(isAvailable)  .message(isAvailable ? "사용 가능한 닉네임입니다." : "이미 존재하는 닉네임입니다.")  .build();  return ResponseEntity.*ok*(response);  } catch (Exception e) {  *log*.error("닉네임 중복 확인 중 오류 발생", e);  NicknameCheckResponse response = NicknameCheckResponse.*builder*()  .success(false)  .available(false)  .message("닉네임 중복 확인 중 오류가 발생했습니다.")  .build();  return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*).body(response);  }  }  private final UserService userService;  @DeleteMapping("/me")  public ResponseEntity<Void> unregisterUser(Authentication authentication) {  if (authentication == null) {  throw new IllegalStateException("인증된 사용자 정보가 없습니다.");  }  Long userId = Long.*parseLong*(authentication.getName());  userService.softDeleteUser(userId);  return ResponseEntity.*ok*().build();  }  } |
| --- |

1. **React 구조**

* 예제

| import { Routes, Route } from 'react-router-dom';  import { LoginSection } from '../pages/Login/LoginSection';  import { MemberRegister } from '../pages/Register/MemberRegister';  import AuthCallback from '../pages/Register/KakaoRegisterCallback';  import { ProfileEdit } from '../pages/ProfileSettings/ProfileEdit';  import PaymentMethodManagementPage from '../pages/Payment/PaymentMethodManagementPage';  import PaymentPage from '../pages/Payment/PaymentPage';  import PaymentSuccessPage from '../pages/Payment/PaymentSuccessPage';  import ContentFormPage from '../pages/ContentForm/ContentFormPage';  import ContentEditPage from '../pages/ContentForm/ContentEditPage';  import { ContentDetailPage } from '../pages/ContentDetail/ContentDetailPage';  import { MyPage } from '../pages/MyPage/MyPage';  import { SubscriptionManage } from '../pages/ManageSubscriptionPage/SubscriptionManage';  import { UserMainPage } from '../pages/MainPage/UserMainPage/UserMainPage';  import MainLayout from '../layouts/mainLayout';  import FavoriteContent from '../pages/FavoriteContent/FavoriteContentPage';  import { CreatorMainPage } from '../pages/MainPage/CreatorMainPage/CreatorMainPage';  import CreatorManagementPage from '../pages/CreatorManagement/CreatorManagementPage';  import CreatorNoticePage from '../pages/CreatorNoticePage/CreatorNoticePage';  import { CreatorSearchPage } from '../pages/CreatorSearchPage/CreatorSearchPage';  import { ContentSearchPage } from '../pages/ContentSearchPage/ContentSearchPage';  import { UnifiedSearchPage } from '../pages/UnifiedSearchPage/UnifiedSearchPage';  import { Home } from '../pages/Home/Home';  export default function AppRoutes() {  return (  <Routes>  <Route element={<MainLayout />}>  {/\* 홈 경로 \*/}  <Route path="/" element={<Home />} />  <Route path="/payments" element={<PaymentMethodManagementPage />} />  <Route path="/payments/summary" element={<PaymentPage />} />  <Route path="/payments/success" element={<PaymentSuccessPage />} />  <Route path="/favorites" element={<FavoriteContent />} />  <Route path="/creator-management" element={<CreatorManagementPage />} />  <Route path="/content-form" element={<ContentFormPage />} />  <Route path="/mypage" element={<MyPage />} />  <Route path="/subscription-manage" element={<SubscriptionManage />} />  <Route path="/user-main/:nickname" element={<UserMainPage />} />  <Route path="/creator-main/:creatorNickname" element={<CreatorMainPage />} />  <Route path="/creators" element={<CreatorSearchPage />} />  <Route path="/contents" element={<ContentSearchPage />} />  <Route path="/search" element={<UnifiedSearchPage />} />  <Route path="/content/:contentId" element={<ContentDetailPage />} /> {/\* 콘텐츠 상세 페이지 라우트 \*/}  <Route path="/content-edit/:contentId" element={<ContentEditPage />} /> {/\* 콘텐츠 수정 페이지 라우트 \*/}  <Route path="/payments/summary/:creatorNickname" element={<PaymentPage />} />  <Route path="/profile/edit" element={<ProfileEdit />} />  <Route path="/creator/notice/:creatorNickname" element={<CreatorNoticePage />} />  </Route>  <Route path="/login" element={<LoginSection />} />  <Route path="/register" element={<MemberRegister />} />  <Route path="/auth/callback" element={<AuthCallback />} />  {/\* 404 처리 \*/}  <Route path="\*" element={<div>404 - Page Not Found</div>} />  </Routes>  );  } |
| --- |