Pusan National University
Computer Science and Engineering
Technical Report 2024-10

컨테이너 오케스트레이션 기반 서버리스 프로젝트 배포 시스템 구축

정보컴퓨터공학부 신예준

정보컴퓨터공학부 김선우

정보컴퓨터공학부 유승훈

지도교수 탁성우

목 차

1. 서론

1.1. 연구 배경

학생들은 텀프로젝트나 비교과 활동을 통해 다양한 프로젝트를 개발하지만, 실행 가능한 형태로 결과물을 공유하는 경우는 드물다. 그 이유는 주로 상용 클라우드 서비스의 비용 부담이나 기술적 어려움 때문이다. 대부분의 학생들은 본인의 개발 환경에서 실행한 화면을 녹화하여 결과물을 간접적으로 공유하는 방식을 택하고 있다.

이러한 문제를 해결하기 위해 본 프로젝트는 서버리스 프로젝트 배포 시스템을 개발하여, 학생들이 비용 부담 없이 기술적 장벽을 넘어서 자신의 프로젝트를 손쉽게 배포하고 공유할 수 있도록 지원하는 것을 목표로 한다. 클라우드 사용이 점차 활성화되고 있는 가운데, 다양한 산업에서 클라우드 인프라를 기반으로 한 서비스 개발 및 배포가 보편화되고 있다. 그러나 학생들이 상용 클라우드 서비스를 직접 활용하기에는 경제적 부담이 크고, 관리 복잡성도 높은 진입장벽이 되고 있다.

또한 최근 소프트웨어 교육이 초·중·고등학교에서부터 대학, 성인 교육에 이르기까지 강화되고 있어 많은 학생들이 실제 프로젝트를 개발할 기회를 얻고 있다. 하지만 그 결과물을 실질적으로 배포하고 다른 사람들과 공유하는 경우는 여전히 적은 실정이다. 더불어 부트캠프 등 집중형 SW 교육 프로그램의 증가로 인해 실전형 프로젝트의 중요성이 부각되면서, 프로젝트 결과물의 배포 및 공유 능력은 취업 경쟁력을 높이는 중요한 요소로 떠오르고 있다.

이에 본 프로젝트는 학생들이 별도의 비용이나 복잡한 기술적 지식 없이도 자유롭게 프로젝트를 배포하고 공유할 수 있는 환경을 제공함으로써, 학습 경험을 더욱 풍부하게 하고 실질적인 성과물을 외부에 공유할 수 있는 기회를 확대하고자 한다.

1.2. 기존 문제점

1.2.1. 프로젝트 배포 및 운영의 어려움

매년 학생들은 동아리 활동이나 학기 중 텀 프로젝트를 통해 다양한 프로젝트를 개발하고, 이 과정에서 보고서를 작성하거나 개발 과정을 발표하며 그들이 학습한 기술적 내용을 공유한다. 하지만 이러한 프로젝트의 결과물이 실제로 외부에서 접속 가능하도록 배포되는 경우는 매우 드물다. 이는 많은 학생들이 프로젝트를 배포하는 과정에 대한 경험이 부족하기 때문이다.

특히 클라우드 플랫폼을 활용해 프로젝트를 배포하는 과정은 학생들에게 생소하게 느껴지거나 기술적 복잡성으로 인해 어려움을 겪는 경우가 많다. 클라우드 배포는 프로젝트의 성과를 외부에 직접적으로 보여줄 수 있는 중요한 단계이지만, 상용 클라우드 서비스의 비용 부담이나 플랫폼 설정 과정에서의 복잡성 때문에 학생들이 이를 시도하지 않거나, 시도하더라도 도중에 포기하는 경우가 적지 않다.

결과적으로, 많은 학생들은 자신이 개발한 프로젝트를 단순히 개발 환경에서 실행한 후, 화면을 녹화해 간접적으로 공유하는 방식에 그치는 경우가 대부분이다. 이렇게 프로젝트의 성과가 외부에서 접속 가능하도록 배포되지 않음으로써, 학생들은 중요한 기술적 경험을 놓치고 있으며, 프로젝트의 가치를 완전히 실현하지 못하는 한계를 겪고 있다.

1.2.2. 프로젝트 배포 이후 비용 문제

프로젝트를 배포하여 운영할 경우 도메인 구입 비용, 클라우드 사용 비용 다양한 요금이 발생한다. 클라우드를 처음 사용한다면 무료 요금제 내에서 사용이 가능하지만 보통 1년의 기간을 가지고 있어 이전에 사용한 적이 있다면 비용이 발생하게 되며, 서버리스 시스템을 이용해 문제를 해결하려 했지만, 이는 일반적으로 상태를 유지하지 않기 때문에 별도의 데이터베이스(DB)를 생성해야 했다.

이로 인해 결국 가상 머신(VM)이나 클라우드 DB를 사용하게 되어 비용이 발생했다. 특히 클라우드 DB의 경우, 최소 요금제를 사용하더라도 고가용성, 백업 등의 기능이 포함되어 있어 1만 원 이상의 높은 비용이 발생한다.

하지만 이러한 고가용성이나 백업 등의 기능은 학생들이 진행하는 프로젝트에서는 중요하지 않은 경우가 많고 그렇다면 이러한 설정이나 비용을 줄이고 간단하게 배포해서 공유하거나 포트폴리오로 활용할 수 있다면 학생들에게 유용한 서비스가 될 수 있다.

1.2.3 프로젝트 결과물을 공유할 수 있는 플랫폼의 부재

많은 동아리와 텀 프로젝트에서 다양한 결과물이 개발되지만, 이를 공유할 수 있는 플랫폼이 부재한 상황입니다. 본 프로젝트에서는 학생들이 지속 가능한 배포를 할 수 있도록 지원할 뿐만 아니라, 그룹 기능과 프로젝트 기능을 통해 학생들 간의 프로젝트 결과물 공유를 원활하게 하는 플랫폼의 역할도 할 수 있을 것으로 기대된다.

1.3. 연구 목표

교내 서버리스 배포 시스템은 프로젝트를 배포 템플릿을 활용해 컨테이너로 쉽게 배포할 수 있도록 하고, 서버리스로 개발하여 접속자가 없는 경우 Auto Scaling과 Scale to 0를 적용함으로써 평소에는 비활성 상태로 유지하여 자원 사용을 최소화한다. 이러한 구성을 통해 교내 클라우드 시스템은 제한된 자원으로도 다수의 프로젝트를 효과적으로 배포할 수 있는 기반을 마련한다.

학생들은 자신의 프로젝트를 실행 가능한 형태로 배포하고, 이를 쉽게 공유할 수 있기 때문에 피드백을 즉각적으로 받을 수 있는 기회가 제공된다.

나아가, 이렇게 배포된 프로젝트는 포트폴리오로도 활용될 수 있어, 학생들이 실제 프로젝트 결과물을 통해 성과를 구체적으로 보여줄 수 있는 중요한 자료가 될 수 있다. # 자료를 활용할 수 있다.

또한, 본 시스템은 그룹 기능과 프로젝트 기능을 제공하여 학생들 간의 협업을 보다 원활하게 할 수 있도록 돕는다. 이를 통해 팀 프로젝트를 관리하고 성과를 공유하는 과정이 체계화되며 프로젝트 결과물 공유가 더욱 손쉽고 효율적으로 이루어진다. 프로젝트의 지속 가능성과 협업의 용이성을 동시에 추구하는 본 시스템은, 학생들이 개발한 다양한 프로젝트가 교내외에서 더 널리 활용되고 확산될 수 있는 환경을 조성한다.

2. 연구 배경

2.1. 배경 지식

2.1.1. 클라우드 컴퓨팅

클라우드 컴퓨팅은 인터넷을 통해 컴퓨팅 리소스를 제공하는 기술이다. 사용자는 물리적 하드웨어를 직접 소유하지 않고도 IT 자원을 필요에 따라 사용한다. 이 모델은 유연성, 확장성, 비용 효율성을 제공한다. 기업과 개인 모두에게 혁신적인 IT 솔루션을 가능하게 한다. 클라우드 서비스는 주로 IaaS, PaaS, SaaS 형태로 제공된다.

2.1.2. 마이크로 서비스 아키텍처(Micro Service Architecture)

마이크로서비스 아키텍처는 소프트웨어 개발 접근 방식이다. 큰 애플리케이션을 작고 독립적인 서비스로 나누어 구성한다. 각 서비스는 특정 비즈니스 기능을 담당하며 독립적으로 개발, 배포, 확장된다. 이 서비스들은 API를 통해 서로 통신한다. 이 아키텍처는 유연성, 확장성, 유지보수성을 향상시키지만, 복잡성과 관리 오버헤드가 증가할 수 있다. 대규모 분산 시스템에 적합하며 DevOps 및 지속적 배포 방식과 잘 어울린다.

2.1.3. 컨테이너

컨테이너 기술은 애플리케이션과 그 의존성을 하나의 패키지로 묶는 가상화 방식이다. 호스트 OS의 커널을 공유하면서 격리된 환경을 제공한다. 가상 머신보다 가볍고 빠르며 리소스 효율성이 높다. 애플리케이션의 일관된 실행 환경을 보장하여 개발과 운영 간의 환경 차이 문제를 해결한다. 마이크로서비스 아키텍처, DevOps, CI/CD 파이프라인과 잘 어울린다. Docker, Kubernetes 등의 도구로 관리되며 클라우드 네이티브 애플리케이션 개발의 핵심 기술이다.

2.1.4. Docker

Docker는 애플리케이션을 컨테이너화하는 플랫폼이다. 소프트웨어를 표준화된 유닛으로 패키징하여 개발, 배포, 실행을 간소화한다. 컨테이너는 애플리케이션 코드, 런타임, 시스템 도구, 라이브러리 등을 포함한다. 이를 통해 환경에 관계없이 일관된 실행을 보장

한다. Docker는 가상화보다 가볍고 효율적이며, 마이크로서비스 아키텍처와 DevOps 프랙티스를 지원한다. 이미지를 통해 버전 관리가 가능하며, 스케일링과 배포가 용이하다.

2.1.5. Kubernetes

Kubernetes는 컨테이너화된 애플리케이션의 자동 배포, 스케일링, 관리를 위한 오픈소스 플랫폼이다. 구글에서 개발했으며 현재 클라우드 네이티브 컴퓨팅 재단에서 관리한다. 대규모 분산 시스템과 마이크로서비스 아키텍처를 효과적으로 운영할 수 있게 한다. 로드 밸런싱, 스토리지 오케스트레이션, 자동 롤아웃과 롤백, 자가 복구 기능을 제공한다. 선언적 구성과 자동화를 통해 애플리케이션 배포와 관리를 간소화한다. 클라우드 프로바이더와 온프레미스 환경 모두에서 사용 가능하다.

2.1.6. Auto Scaling

Auto Scaling은 클라우드 컴퓨팅 환경에서 시스템 리소스를 자동으로 조정하는 기술이다. 애플리케이션의 부하에 따라 컴퓨팅 리소스를 동적으로 늘리거나 줄여 성능을 최적화하고 비용을 효율적으로 관리한다. 주로 CPU 사용률, 메모리 사용량, 네트워크 트래픽 등의 메트릭을 모니터링하여 미리 정의된 규칙에 따라 자동으로 인스턴스를 추가하거나 제거한다. 이를 통해 트래픽 급증 시 서비스 중단을 방지하고, 사용량이 적을 때는 불필요한 리소스 낭비를 줄일 수 있다. 대부분의 주요 클라우드 제공업체들이 Auto Scaling 서비스를 제공하며, 이는 클라우드 네이티브 애플리케이션의 핵심 기능 중 하나로 자리잡았다.

2.1.7. Scale to zero

Scale to zero는 클라우드 컴퓨팅의 중요한 개념이다. 이는 애플리케이션이 사용되지 않을 때 자원 소비를 완전히 중단하는 능력을 말한다. 트래픽이 없을 때 인스턴스 수를 0으로 줄여 비용을 절감한다. 요청이 들어오면 시스템이 자동으로 필요한 리소스를 프로비저닝하여 응답한다. 이 접근 방식은 서버리스 컴퓨팅과 밀접한 관련이 있으며, 리소스 효율성과 비용 최적화를 크게 향상시킨다.

2.1.8. SSL 인증서

SSL 인증서는 웹사이트의 신원을 확인하고 암호화된 연결을 제공하는 디지털 문서다. HTTPS 프로토콜의 핵심 요소로, 클라이언트와 서버 간 통신을 암호화한다. 신뢰할 수 있는 인증 기관(CA)에서 발급되며, 공개키와 개인키 쌍을 사용한다. 사용자의 데이터를 중간자 공격으로부터 보호하고, 웹사이트의 신뢰성을 보장한다. 다양한 종류가 있으며, 도메인 검증(DV), 조직 검증(OV), 확장 검증(EV) 인증서 등이 포함된다. 전자상거래, 온라인 뱅킹 등 민감한 정보를 다루는 웹사이트에서 필수적이다.

2.1.9. LetsEncrypt

Let's Encrypt는 무료로 제공되는 자동화된 공개 인증 기관(CA)이다. HTTPS를 위한 TLS 인증서를 무료로 발급하고 갱신할 수 있게 해준다. 이 서비스는 웹 보안을 증진시키고 암호화된 연결을 표준화하는 것을 목표로 한다. Let's Encrypt는 사용하기 쉬운 클라이언트 소프트웨어를 제공하여 인증서 발급과 갱신 과정을 자동화한다. 90일마다 인증서를 갱신해야 하지만, 이 과정도 자동화할 수 있어 관리가 용이하다. 많은 웹 호스팅 제공업체와 서버 소프트웨어가 Let's Encrypt를 기본적으로 지원하여, HTTPS 채택을 크게 촉진시켰다.

2.1.10. Nginx

Nginx는 고성능 웹 서버이자 리버스 프록시, 로드 밸런서로 널리 사용되는 오픈 소스 소프트웨어다. 비동기 이벤트 기반 아키텍처를 채택하여 높은 동시성과 낮은 메모리 사용량을 자랑한다. 정적 파일 서빙에 특히 뛰어나며, 동적 콘텐츠 처리를 위해 다양한 애플리케이션 서버와 연동할 수 있다. 또한 HTTP/2, SSL/TLS, WebSocket 등 최신 웹 기술을 지원하며, 설정이 간단하고 유연해 다양한 환경에서 활용된다. 대규모 트래픽을 처리하는 웹사이트와 애플리케이션에서 널리 채택되고 있다.

2.1.11. OpenResty

OpenResty는 Nginx HTTP 서버와 LuaJIT을 기반으로 하는 고성능 웹 애플리케이션 플랫폼이다. 이 플랫폼은 Nginx의 모든 기능을 제공하면서도 Lua 프로그래밍 언어를 통해 웹 애플리케이션을 쉽고 빠르게 개발할 수 있게 한다. OpenResty는 높은 동시성과 저지연 처리를 지원하며, 다양한 모듈과 라이브러리를 제공하여 확장성이 뛰어나다. 주로 API 게이트웨이, 웹 애플리케이션 방화벽, 동적 웹 서비스 등의 구현에 사용되며, 대규모 트래픽 처리가 필요한 환경에서 특히 효과적이다.

2.1.12. Spring Boot

스프링 부트는 자바 기반의 프레임워크다. 독립 실행형 프로덕션 급 스프링 애플리케이션을 쉽게 만들 수 있게 한다. 복잡한 설정을 자동화하고 개발자가 빠르게 애플리케이션을 구축할 수 있도록 돕는다. '설정보다 관례'를 강조하며 기본 설정을 제공한다. 내장 서버를 포함하여 별도의 웹 서버 없이 독립적으로 실행 가능하다. 마이크로서비스 개발에 적합하며 클라우드 네이티브 애플리케이션 구축을 지원한다.

2.1.13. Spring Security

Spring Security는 스프링 기반 애플리케이션을 위한 강력한 보안 프레임워크다. 인증과 권한 부여를 위한 포괄적인 솔루션을 제공한다. 다양한 인증 방식(폼 로그인, OAuth, LDAP 등)을 지원하며 쉽게 커스터마이즈할 수 있다. 웹 요청, 메소드 호출, 도메인 객체에 대한 접근 제어가 가능하다. 크로스 사이트 스크립팅(XSS), 크로스 사이트 요청 위조(CSRF) 등의 보안 위협에 대한 보호 기능을 제공한다. 스프링 생태계와 잘 통합되어 있어 스프링 프로젝트에서 쉽게 적용할 수 있다.

2.1.14. PostgreSQL

PostgreSQL은 강력한 오픈 소스 객체-관계형 데이터베이스 시스템이다. 높은 신뢰성, 데이터 무결성, 정확성으로 알려져 있다. 복잡한 쿼리, 외래 키, 트리거, 뷰 등 고급 기능을 지원한다. ACID 속성을 완벽히 준수하며 트랜잭션 처리에 강점을 가진다. 대규모 데이터 워크로드를 처리할 수 있으며, 지리 정보 시스템(GIS)을 위한 PostGIS 확장 기능도 제공한다. 기업용 데이터베이스로 널리 사용되며 클라우드 환경에서도 잘 작동한다.

2.1.15. Redis

Redis는 고성능 인메모리 데이터 구조 저장소다. 키-값 데이터베이스로 작동하며 다양한 데이터 구조를 지원한다. 문자열, 해시, 리스트, 셋, 정렬된 셋 등을 처리할 수 있다. 주로 캐싱, 세션 관리, 실시간 분석 등에 사용된다. 매우 빠른 읽기/쓰기 속도를 제공하며 분산 시스템에서의 데이터 공유에 적합하다. 퍼블리시/섭스크라이브 메시징 시스템도 구현 가능하다. 데이터의 영속성도 지원하여 디스크에 데이터를 저장할 수 있다.

2.1.16. Harbor

Harbor는 오픈소스 컨테이너 레지스트리 플랫폼이다. 엔터프라이즈급 보안, 신원 관리, 정책 제어 기능을 제공하여 Docker 이미지를 저장, 관리, 배포하는 데 사용된다. 주요 기능으로는 역할 기반 접근 제어(RBAC), 이미지 취약점 스캐닝, 이미지 서명 및 검증, 프로젝트 격리 등이 있다. Harbor는 Kubernetes와 같은 컨테이너 오케스트레이션 플랫폼과 잘 통합되며, 멀티 테넌시를 지원하여 여러 팀이나 프로젝트에서 안전하게 사용할 수 있다. 또한 복제 기능을 통해 여러 레지스트리 간 이미지 동기화가 가능하여 고가용성과 재해 복구를 지원한다.

2.1.17. React

리액트(React)는 SPA(Single Page Application) 구축에 최적화된 자바스크립트 라이브러리이다. SPA는 단일 페이지에서 모든 콘텐츠를 동적으로 로드하여 페이지 전환 없이 빠른 사용자 경험을 제공한다. 컴포넌트 기반 구조로, 상태 변화에 따라 필요한 부분만을 업데이트하여 성능을 극대화하기 때문에 사용자는 더 빠르고 매끄러운 인터페이스를 경험할 수 있다.

2.2. 서비스 기획

2.2.1. 기능 정의

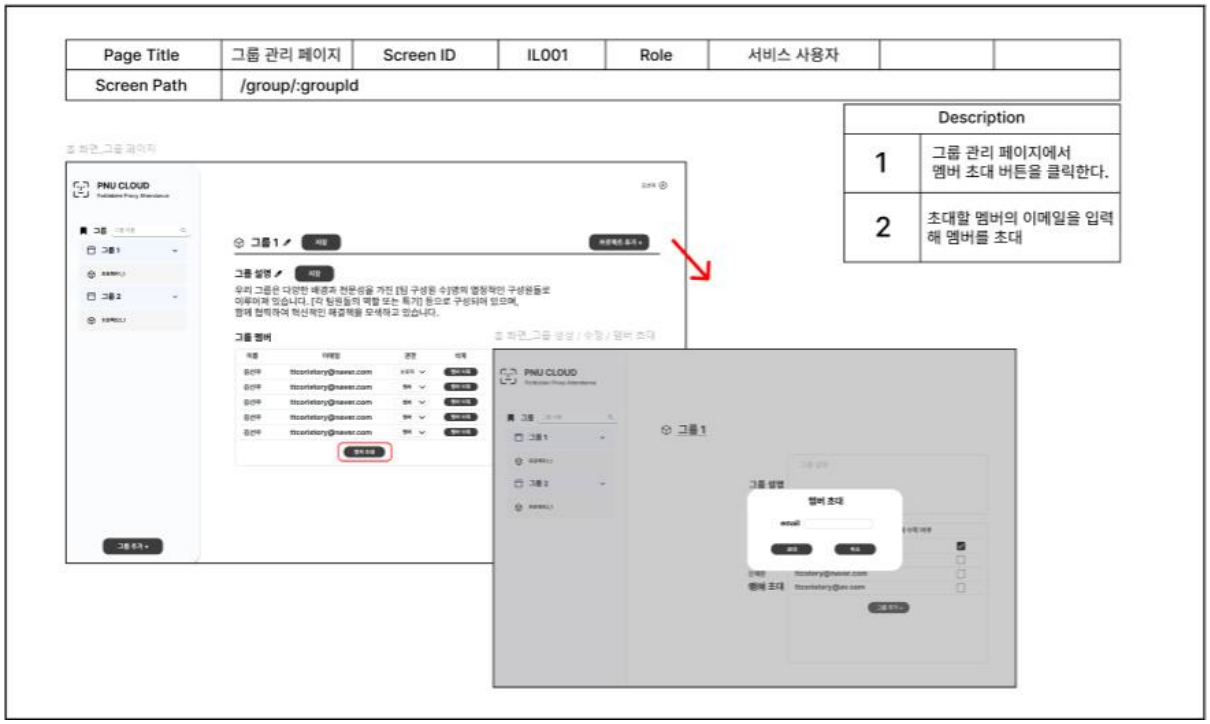
프로젝트 요구 사항을 명확히 하고, 서비스의 주요 기능을 정의하며, 프로젝트 진행의 기준을 설정하기 위해 기능을 정의한 정의서이다.

요구사항명	기능명	상세 설명	데이터 (*는 필수)
로그인	자체 로그인 기능	서비스를 사용하기 위해 로그인 필수	*ID, *비밀번호
회원가입	자체 회원가입 기능	서비스 사용을 위해 기본 정보를 입력해 회원가입 필수	*실명, *이메일, *비밀번호
	이메일 중복 확인 기능	이메일 중복 확인을 위해 필수	
	인증 번호 확인 기능	유효한 이메일을 확인하기 위해 필수	*인증 번호
그룹 생성	그룹 생성	네비게이션 바 하단의 '그룹 생성' 버튼을 눌러 프로젝트를 관리할 수 있는 그룹을 생성하는 기능	*그룹 이름, 그룹 설명
그룹 멤버 초대	그룹 멤버 초대	서비스를 이용하는 다른 유저를 해당 그룹에 이메일을 통해 초대하는 기능	*초대할 유저 이메일
그룹 멤버 권한 관리	그룹 멤버 권한 관리	그룹의 ADMIN 유저가 그룹에 속한 유저들의 권한을 관리할 수 있는 기능	*권한
프로젝트 생성	프로젝트 생성	그룹 리스트 중 프로젝트를 생성할 그룹을 선택해 '프로젝트 추가' 버튼을 통해 컨테이너들의 배포를 할 수 있는 프로젝트를 생성할 수 있는 기능	*프로젝트 명, *도메인 주소, *프로젝트 권한
	도메인 주소 중복 확인	프로젝트의 대표 도메인 주소의 중복 여부를 확인하는 기능	
프로젝트 멤버 권한 관리	프로젝트 멤버 권한 관리	프로젝트의 ADMIN 유저가 프로젝트에 속한 유저들의 권한을 관리할 수 있는 기능	*권한
프로젝트 삭제	프로젝트 삭제	프로젝트 목록 중 하나의 프로젝트를 선택 후 오른쪽 상단의 '프로젝트 삭제' 버튼을 누르면 팝업 박스로 한 번 더 확인하고 프로젝트 삭제	

컨테이너 생성	컨테이너 생성	컨테이너를 생성할 프로젝트를 선택 후 원하는 기술 스택을 선택하고 컨테이너 필수 정보를 입력하고 '저장' 버튼을 눌러 컨테이너 생성	*컨테이너 이름, *컨테이너 파일, *서브 도메인(SQL 기술 스택은 필요 없음), 환경 변수
컨테이너 삭제	컨테이너 삭제	컨테이너 목록 중 삭제할 컨테이너의 오른쪽 '삭제' 버튼을 누르면 팝업 박스로 한 번 더 확인하고 컨테이너 삭제	

2.2.2. 화면 설계

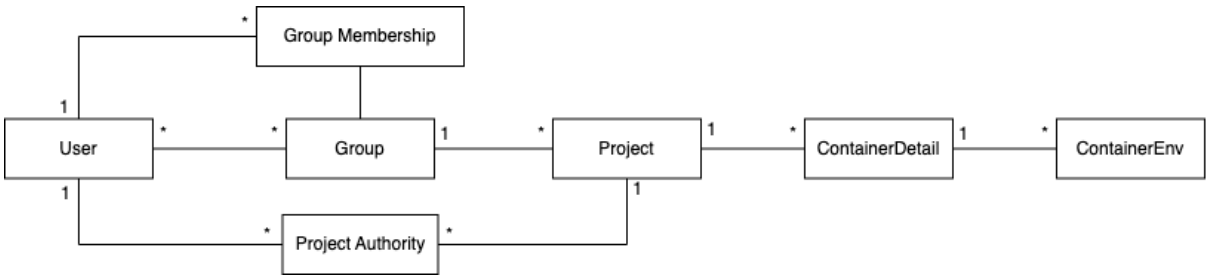
사용자 경험을 고려한 직관적이고 효율적인 인터페이스를 제고하기 위해 UI/UX 디자인을 작성한다. (화면 설계서 예시)



2.2.3 데이터베이스 설계

데이터베이스는 개념적 설계 및 논리적 설계를 진행하였다.

[개념적 설계]



주요 엔티티에는 User(사용자), Group(사용자가 생성하는 사용자들의 집합), Project(그룹 내에서 배포되는 프로젝트)가 있으며, 권한 관련 엔티티에는 GroupMembership, ProjectAuthority가 있다.

전체 엔티티에 대한 세부 내용은 다음과 같다.

엔티티	설명	주요 필드
User	시스템 사용자	이름, 이메일, 비밀번호
Group	사용자가 생성하는 시스템 사용자의 집합	그룹 이름
Project	그룹 내에서 배포되는 프로젝트	프로젝트 이름, 서브도메인
GroupMembership	그룹 내 사용자 정보 및 사용자들의 역할	그룹 내 역할(관리자, 멤버)
ProjectAuthority	프로젝트 정보를 보거나, 편집할 수 있는 권한	프로젝트 권한(읽기, 편집)
ContainerDetail	프로젝트 내 컨테이너 정보	서브도메인, 컨테이너 이름
ContainerEnv	프로젝트 내 컨테이너의 환경변수 정보	컨테이너의 환경변수 Key 값

각 엔티티 간 관계는 다음과 같이 설정하였다.

엔티티	**주요 엔티티:** User, Group, Project, ContainerDetail, ContainerEnv **부가적 엔티티:** GroupMemberShip, ProjectAuthority
관계 설정	한 사용자는 여러 그룹을 가진다. (1:N)
	한 그룹은 여러 사용자를 가진다. (1:N)
	한 그룹엔 여러 프로젝트를 가진다. (1:N)
	한 그룹은 여러 그룹 멤버십을 가진다. (1:N)
	한 프로젝트는 여러 프로젝트 권한을 가진다. (1:N)
	한 사용자는 여러 프로젝트 권한을 가진다. (1:N)
	한 프로젝트는 한 개 이상의 컨테이너 세부 정보를 가진다. (1:N)
	한 컨테이너 세부 정보는 여러 환경변수 정보를 가질 수 있다. (1:N)

[논리적 설계: 주요 테이블]

주요 테이블인 User, Group, Project의 테이블 스키마는 다음과 같이 설계하였다.

User	
PK	id INT AUTO_INCREMENT
	username VARCHAR(50) NOT NULL UNIQUE
	email VARCHAR(100) NOT NULL UNIQUE
	password_hash VARCHAR(255) NOT NULL

Group	
PK	id INT AUTO_INCREMENT
	name VARCHAR(100) NOT NULL
	creator_id INT NOT NULL
	FOREIGN KEY (creator_id) REFERENCES User(id)

Project	
PK	id INT AUTO_INCREMENT
	name VARCHAR(100) NOT NULL
	owner_id INT
	group_id INT
	FOREIGN KEY (owner_id) REFERENCES User(id)
	FOREIGN KEY (group_id) REFERENCES Group(id)

테이블 설명

- **User(사용자 테이블)**
  - **username:** 사용자 닉네임
  - **email:** 사용자 이메일
  - **password_hash:** sha256 encrypt된 비밀번호 해시값

- **Group(그룹 테이블)**
  - **name:** 그룹 이름
  - **creator_id:** 그룹을 생성한 사용자 id
- **Project(프로젝트 테이블)**
  - **name:** 프로젝트 이름
  - **owner_id:** 프로젝트를 생성한 사용자 id
  - **group_id:** 프로젝트가 속해 있는 그룹의 id

주요 관계 사항

- **Group 테이블의 creator_id는 User 테이블의 기본키를 외래키로 가짐**
- **Project 테이블의 owner_id는 User 테이블의 기본키를 외래키로 가짐**
- **Project 테이블의 group_id는 Group 테이블의 기본키를 외래키로 가짐**

[논리적 설계: 프로젝트 관련 테이블]

ContainerDetail	
PK	id INT AUTO_INCREMENT
	container_domain VARCHAR(100) NOT NULL
	container_template_name VARCHAR(255) NOT NULL
	container_stack VARCHAR(100) NOT NULL
	code_file_name VARCHAR(100) NOT NULL
	code_file_url VARCHAR(255) NOT NULL
	port INT NOT NULL
	FOREIGN KEY (project_id) REFERENCES Project(id)

ContainerEnv	
PK	id INT AUTO_INCREMENT
	env_key VARCHAR(100)
	FOREIGN KEY (container_id) REFERENCES ContainerDetail(id)

테이블 설명

- **ContainerDetail(프로젝트 내 컨테이너 정보를 저장하는 테이블)**
  - **container_domain:** 컨테이너에 할당된 도메인(또는 서브도메인)
  - **container_template_name:** 컨테이너 내에서 배포 템플릿 이름
  - **container_stack:** 컨테이너 배포 템플릿의 환경 및 프레임워크
  - **code_file_name:** 컨테이너 내에서 실행될 코드 파일 이름
  - **code_file_url:** 컨테이너 내에서 실행될 코드 파일 서버 경로
  - **port:** 컨테이너에 할당된 포트
- **ContainerEnv(컨테이너 내 환경변수의 키 값을 저장하는 테이블)**
  - **env_key:** 컨테이너 내 환경변수의 키 값

주요 관계 사항

- ContainerDetail 테이블의 project_id는 Project 테이블의 기본키를 외래키로 가짐
- ContainerEnv 테이블의 container_id는 ContainerDetail 테이블의 기본키를 외래키로 가짐

[논리적 설계: 권한 관련 테이블]

GroupMembership	
PK	id INT AUTO_INCREMENT
	user_id INT
	group_id INT
	role ENUM('admin', 'member') NOT NULL
	FOREIGN KEY (user_id) REFERENCES User(id)
	FOREIGN KEY (group_id) REFERENCES Group(id)
	UNIQUE (user_id, group_id)

ProjectAuthority	
PK	id INT AUTO_INCREMENT
	user_id INT
	project_id INT
	permission ENUM('view', 'edit') NOT NULL
	FOREIGN KEY (user_id) REFERENCES User(id)
	FOREIGN KEY (project_id) REFERENCES Project(id)
	UNIQUE (user_id, project_id)

테이블 설명

- **GroupMembership**
  - **user_id:** 그룹 멤버십을 가진 사용자 id
  - **group_id:** 그룹 멤버십이 속한 그룹 id
  - **role:** 사용자의 그룹 내 역할(관리자 또는 멤버)
- **ProjectAuthority**
  - **user_id:** 프로젝트 권한을 가진 사용자 id
  - **project_id:** 프로젝트 권한이 속한 프로젝트 id
  - **permission:** 프로젝트 권한 종류(읽기 가능 또는 편집 가능)

주요 관계 사항

- GroupMembership 테이블의 user_id는 User 테이블의 기본키를 외래키로 가짐
- GroupMembership 테이블의 group_id는 Group 테이블의 기본키를 외래키로 가짐
- ProjectAuthority 테이블의 user_id는 User 테이블의 기본키를 외래키로 가짐
- ProjectAuthority 테이블의 project_id는 Project 테이블의 기본키를 외래키로 가짐

주요 제약 사항

- GroupMembership 테이블에서 user_id와 group_id의 조합은 고유함
- ProjectAuthority 테이블에서 user_id와 project_id의 조합은 고유함

2.3. 개발 환경

분야	사용 기술 및 도구
운영체제	- Ubuntu 24.04 LTS Server
컨테이너	- Docker 20.10.23 - Kubernetes 1.28.14
프로그래밍 언어	- JDK 22 - ECMA Script 6 - Python 3
프레임워크	- Spring boot 3.3.2 - React 18.3.1
빌드 도구	- Gradle 8.8 - Babel 7.24.7 - Webpack 0.5.15
데이터베이스	- PostgreSQL 10.0 - Redis 7.2.4

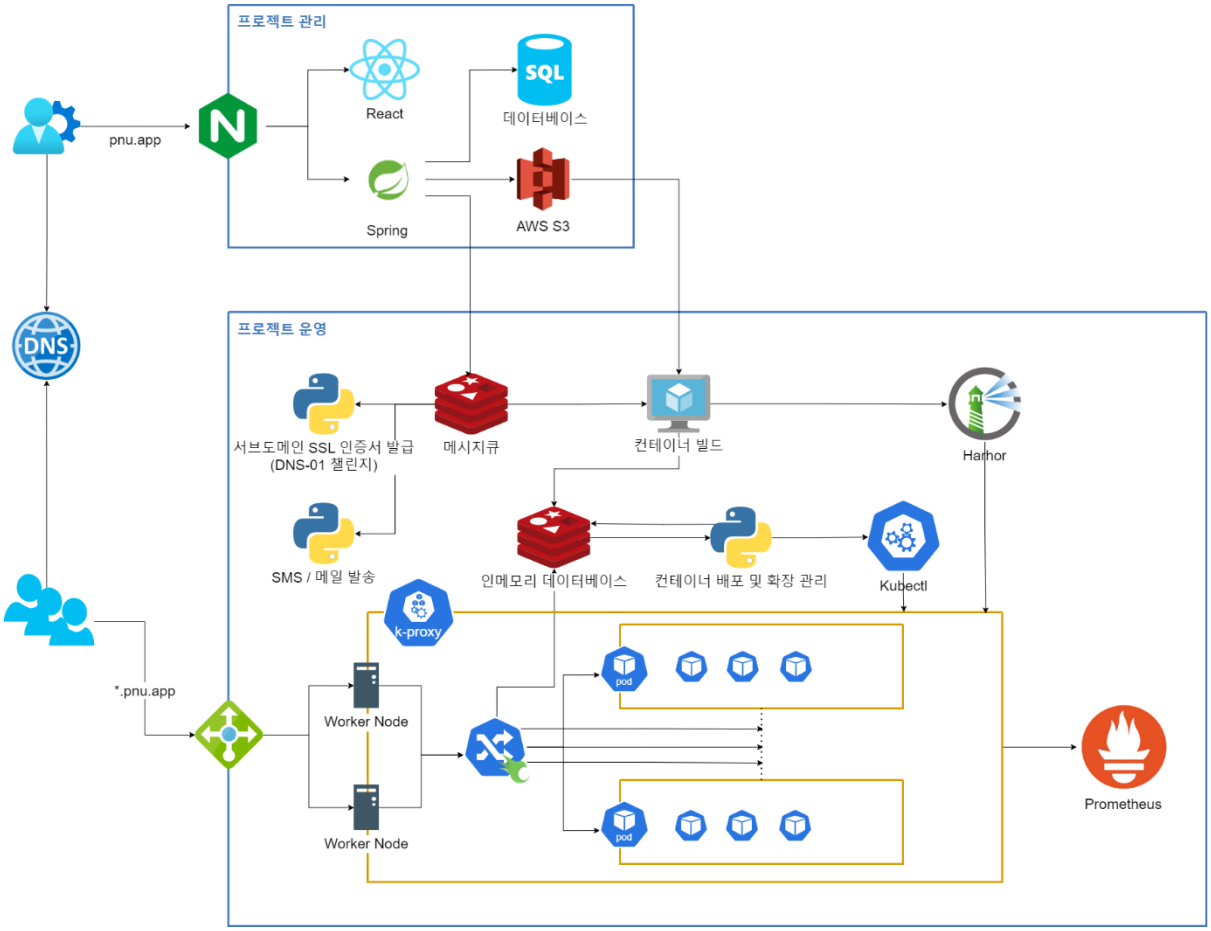
2.4. 용어 정리

보고서의 이해를 돕기 위해, 연구 내용을 소개하기에 앞서 프로젝트와 관련된 주요 용어들을 정의한다. 용어 설명을 통해 연구의 전반적인 내용을 쉽게 이해할 수 있도록 한다.

용어	정의
사용자	해당 서비스를 이용하는 모든 개인
그룹	함께 프로젝트들을 관리하는 사용자의 모임
프로젝트	하나의 목표를 달성하기 위해 독립적인 컨테이너의 모임
기본 도메인	SLD 도메인 주소, 예를 들어"pnu.app"이 기본 도메인
서브 도메인	기본 도메인의 하위 도메인을 나타내는 주소 예를 들어, "sub.pnu.app"가 "pnu.app"의서브 도메인
환경 변수	시스템이나 프로그램이 필요로 하는 설정 값을 동적으로 저장하고 관리하는 키-값 쌍
배포 템플릿	DB, 프론트엔드, 백엔드 등 프로젝트의 주요 구성 요소를 컨테이너로 쉽게 배포할 수 있도록 사전 정의된 파일
컨테이너	애플리케이션과 그 실행에 필요한 라이브러리, 종속성 등을 패키징하여 독립적으로 실행할 수 있도록 하는 가상화된 환경

3. 연구 내용

3.1. 서비스 전체 구성도



본 연구의 전체적인 아키텍처는 위와 같이 구성되어 있다.

프로젝트 관리를 위한 웹 서비스와 프로젝트를 구동하는 부분으로 나눌 수 있으며 사용자가 프로젝트를 새로 업로드한 경우 메시지큐를 통해 인증서 발급과 컨테이너 빌드 및 배포 설정을 메시지를 통해 전송한다.

해당 작업이 완료될 경우 Worker Node에 Pod가 생성되어 요청을 처리하게 되며 Ingress에서 연결된 세션 개수 및 마지막 요청 시간을 기록하여 요청이 없을 경우 Auto Scale을 수행하여 크기를 0으로 축소한다. 이후 크기가 0인 상태에서 요청이 오면 다시 컨테이너를 실행시켜 요청을 수행하게 된다.

3.1.1 인프라 아키텍처

프로젝트 운영을 위한 인프라의 주요 구성요소로는 컨테이너 관리를 위한 Kubernetes, HTTP 요청을 전달하고 Auto Scale에 필요한 정보를 관리하는 Ingress, 컨테이너가 있다.

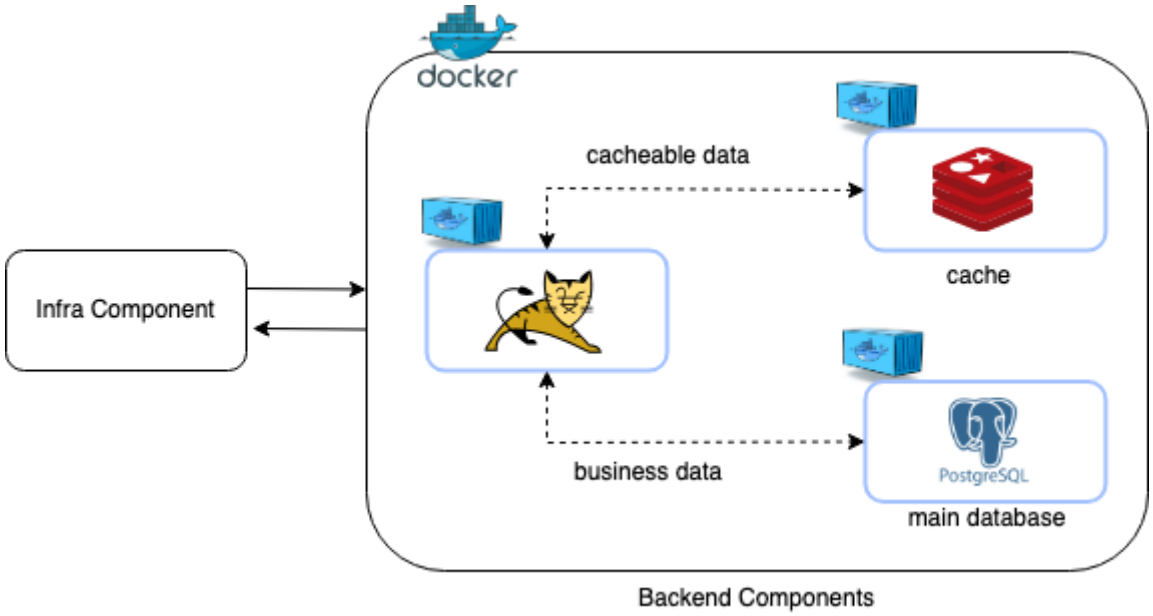
Kubernetes의 경우 다수의 서버의 사용하는 환경을 가정하여 개발하기 위해 2대의 Worker Node를 사용하였으며, 이를 위해 로드밸런서와 KubuPorxy를 사용하여 네트워크가 구성되어 있다.

```
zero@hyper-v:/usr/local/bin$ calicoctl ipam show --show-block
+----------+-----------------+-----------+------------+--------------+
| GROUPING |       CIDR      | IPS TOTAL | IPS IN USE |   IPS FREE   |
+----------+-----------------+-----------+------------+--------------+
| IP Pool  | 172.16.0.0/16   |     65536 | 6 (0%)     | 65530 (100%) |
| Block    | 172.16.219.64/26 |        64 | 1 (2%)     | 63 (98%)     |
| Block    | 172.16.247.0/26 |        64 | 1 (2%)     | 63 (98%)     |
| Block    | 172.16.84.128/26 |        64 | 4 (6%)     | 60 (94%)     |
+----------+-----------------+-----------+------------+--------------+
```

Ingress는 Lua Script 실행이 가능한 OpenResty로 구성되어 있으며 Redis에 현재 연결된 세션 개수와 마지막 접속시간을 기록하고 이를 바탕으로 컨테이너를 관리한다.

3.1.2 웹서비스 백엔드 아키텍처

백엔드 아키텍처의 전체적인 구성은 다음과 같다.



백엔드 컴포넌트의 구성요소는 WAS(Web Application Server), In Memory Cache, RDBMS로 구성되어 있으며, 각각 Tomcat, Redis, PostgreSQL을 사용한다.

각각의 구성요소들은 도커라이징된 컨테이너들로 구성되며, Docker에서 지원하는 내부 네트워크를 통해 도메인 네임으로 서로 통신한다. 또한 인프라 자원 및 컴포넌트와는 메시지큐를 통해 비동기적으로 데이터를 주고 받는다.

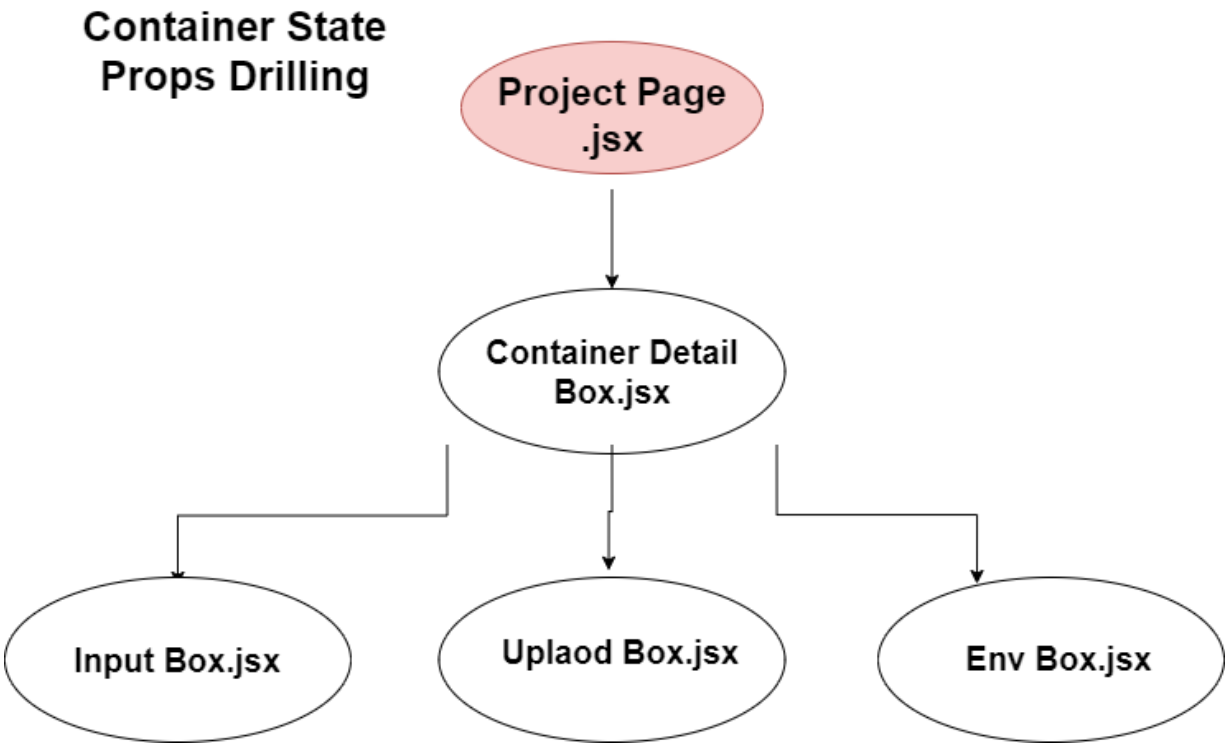
전체 구성요소의 세부 역할은 다음과 같다.

구성요소	역할
Tomcat(WAS)	- Spring Security를 이용한 인증/인가 처리 - REST API Endpoint 제공 - Spring Data JPA를 이용한 데이터 접근 처리
Redis(Cache)	- Cacheable 데이터(ex. 회원가입 시 이메일 인증코드) 저장 - API 성능 튜닝을 위한 인메모리 캐싱
PostgreSQL(DB)	- 영구적 데이터 저장 및 관리 - 트랜잭션 처리 및 데이터 무결성 보장
Docker	- 어플리케이션 및 의존성 컨테이너화 - 인프라 구성요소와의 환경 일관성 유지

3.1.3 웹서비스 프론트엔드 아키텍처

프론트엔드 상태 관리 구조는 크게 2가지로 나뉜다.

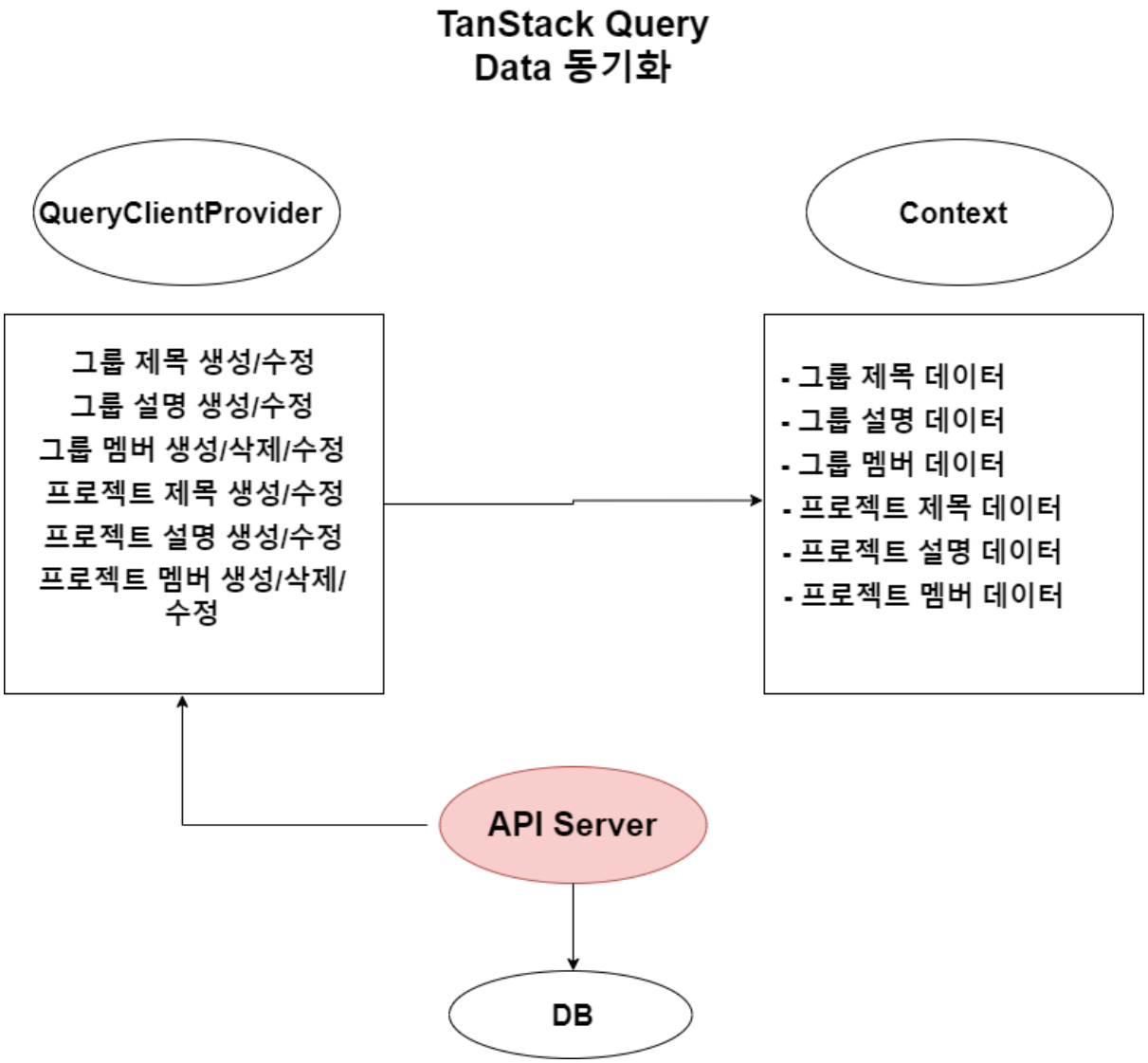
Props Drilling



Props Drilling의 대표적인 예시로 Project Page컴포넌트에서 Containor State가 있다.

이 상태 값을 Proejct Page -> Containor Detail -> Input Box, Uplaod Box, Env Box 순으로 Props를 통해 전달한다. 이 방식은 상태를 전달해야 할 컴포넌트가 많아질수록 코드 가독성이 떨어지고 유지보수가 어려워질 수 있다는 단점이 있다. 그러나 본 프로젝트에서는 Depth가 최대 3이기 때문에, 전역 상태 관리 라이브러리를 사용하지 않고, Props Drilling을 사용하여 상태 관리를 하였다.

서버와 클라이언트 간 데이터 동기화

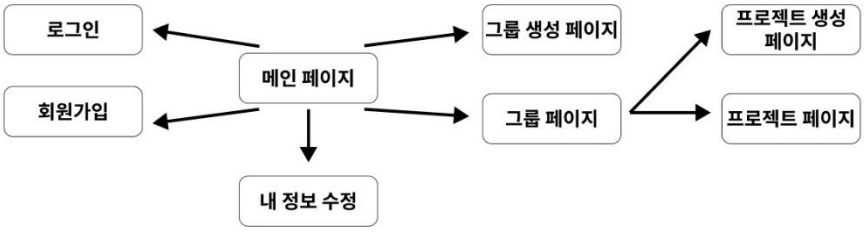


자주 수정되고 삭제될 수 있는 그룹 및 프로젝트 데이터를 TanStack Query를 통해 효율적으로 관리했다. TanStack Query는 데이터의 캐싱, 갱신, 그리고 무효화를 자동으로 처리하여 서버와의 데이터 동기화를 간편하게 만든다. 특히 그룹과 프로젝트와 같은 자주 변경되는 데이터를 다룰 때, `invalidateQueries` 기능을 사용하여 사용자가 데이터를 수정하거나 삭제할 때마다 실시간으로 최신 데이터를 동기화할 수 있다. 이로 인해 데이터 일관성을 유지하면서도 API 호출을 최적화하여 불필요한 요청을 줄일 수 있다.

3.2 사용자 서비스

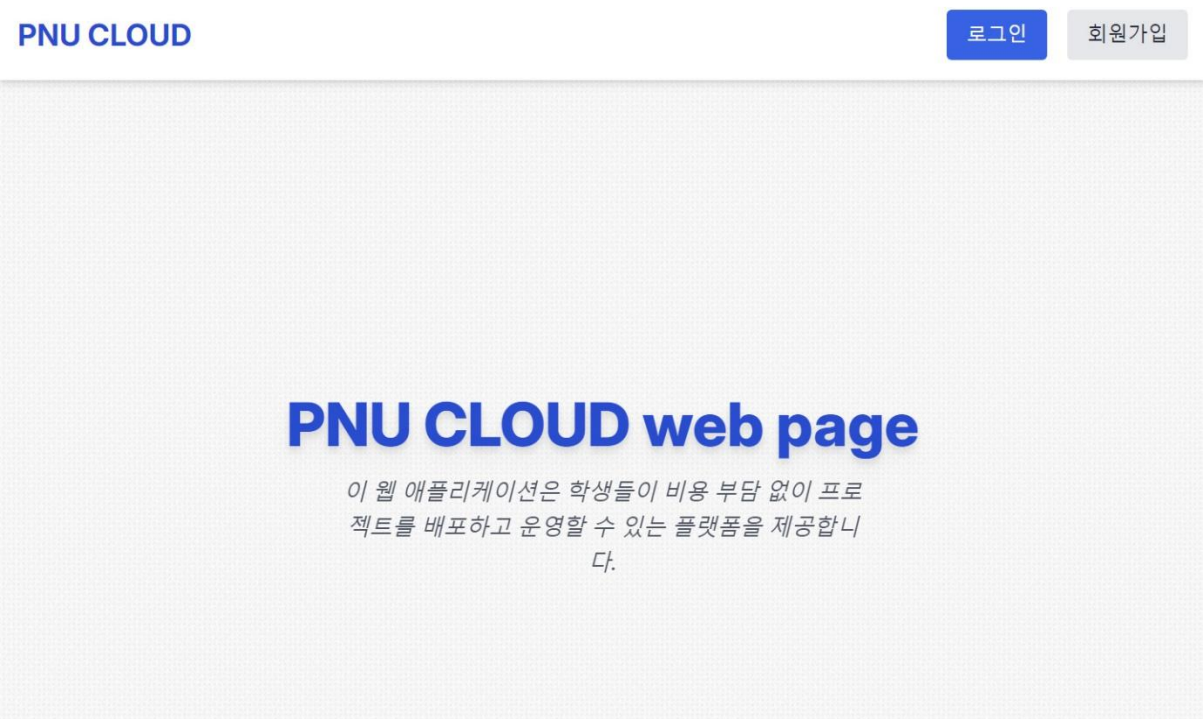
3.2.1 라우팅 구조

서비스 전체 라우팅 구조를 나타내는 그림이다.



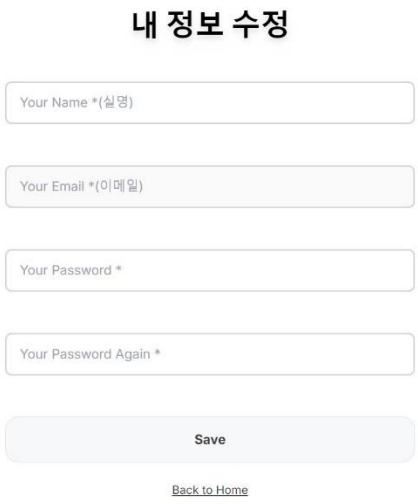
Name	Router	설명
메인페이지	/home	홈 화면으로 사용자가 주요 콘텐츠에 접근할 수 있는 페이지
로그인	/login	사용자가 로그인할 수 있는 페이
회원가입	/signup	사용자가 회원가입을 할 수 있는 페이지
내 정보 수정	/my-profile	사용자가 자신의 프로필 정보를 수정할 수 있는 페이지
그룹 생성 페이지	/group	새로운 그룹을 생성하는 페이지
그룹 페이지	/group/groupId	특정 그룹의 세부 정보를 확인하는 페이지
프로젝트 생성 페이지	/group/:groupId/project	특정 그룹에서 새로운 프로젝트를 생성하는 페이지
프로젝트 페이지	/group/:groupId/project/:projectId	특정 프로젝트의 세부 정보를 확인하는 페이지

3.2.2 메인 페이지



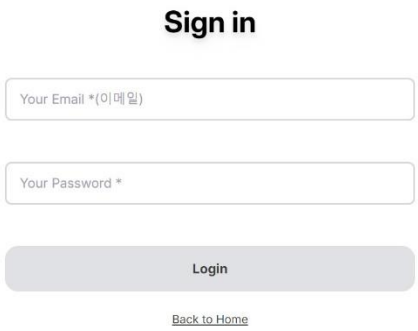
설정한 도메인을 통해 외부 사용자는 웹 브라우저를 통해 접근할 수 있다.

3.2.3 회원 가입



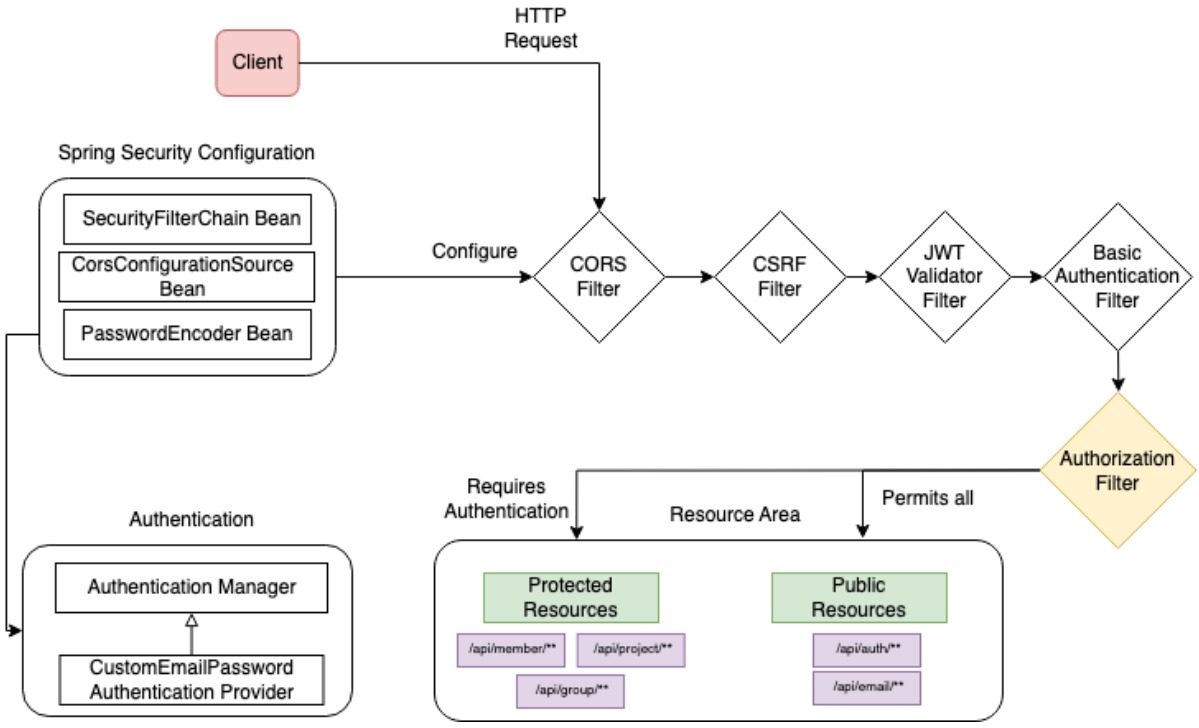
사용자는 이메일, 이름, 비밀번호를 입력한 후 회원가입을 할 수 있다.

3.2.3 로그인



사용자는 가입된 이메일과 비밀번호로 로그인을 할 수 있다.

백엔드 어플리케이션에서 REST API를 통해 세밀한 접근 제어와 권한 관리를 할 수 있도록 Spring Security Filter Chain을 아래 그림과 같이 구성하였다.



세부 인증/인가 동작 과정은 다음과 같다.

[인증 과정]

1. CORS(Cross Origin Resource Sharing) 처리

클라이언트 요청 시, 우선적으로 CORS Filter가 작동하며, 설정된 CORS 정책에 따라 요청의 출처를 검사하고, 허용 여부를 검사한다.

2. JWT 토큰 검증

JWT Token Validator Filter가 요청 헤더에서 JWT토큰을 추출하고 이 디지털 서명 및 만료 시간을 검증한다. 유효한 토큰이라면 인증된 것으로 간주고하고 다음 Filter Chain으로 진행한다.

3. 사용자 이메일/비밀번호 인증

JWT Token이 없거나, 유효하지 않은 경우 Basic Authentication Filter가 동작한다. 이는 Spring Security에서 제공하는 권한 제공자인 Authentication Manager를 상속한 CustomEmail-PasswordAuthenticationProvider를 호출하여 인증을 처리한다. 이 Provider는 회원관련 DAO(Data Access Object)인 MemberRepository를 사용하여 사용자 정보를 조회하고, PasswordEncoder Bean을 사용하여 제공된 비밀번호와 저장된 비밀번호를 비교한다.

[인가 과정]

1. 요청 경로 검사

Authorization Filter가 요청된 리소스의 URL을 검사한다.

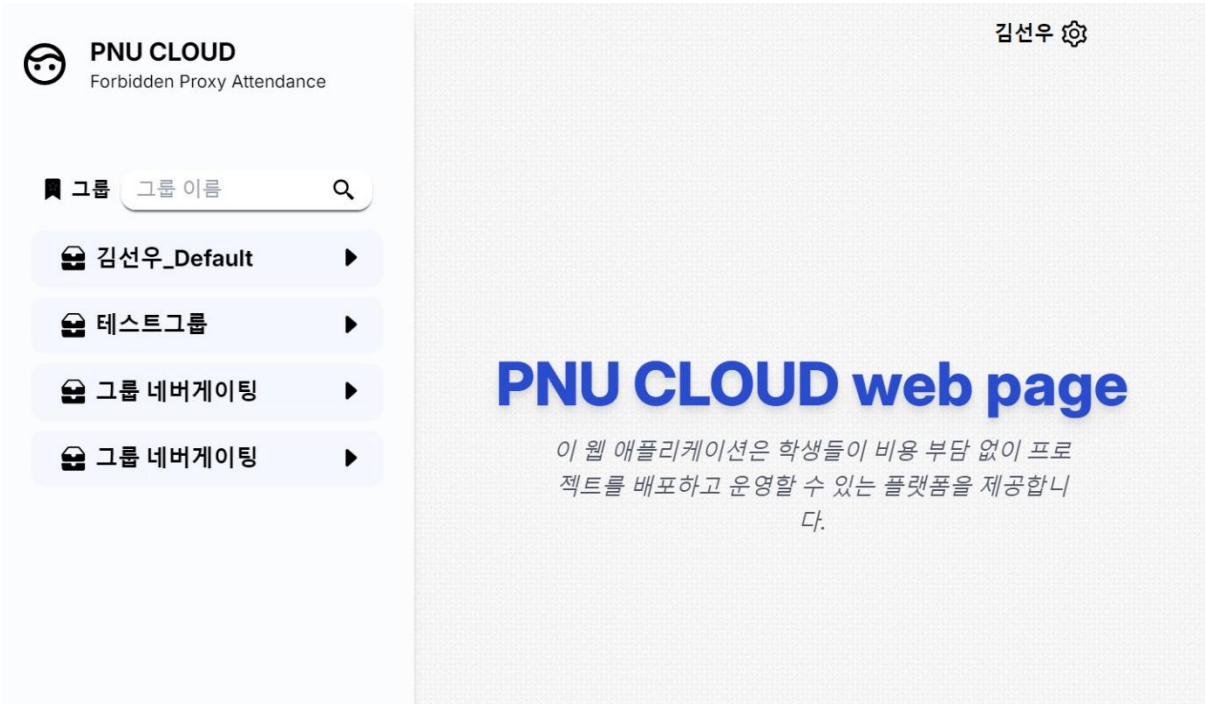
2. 접근 권한 확인

지정된 API 엔드포인트에 따라 접근 권한을 확인한다.

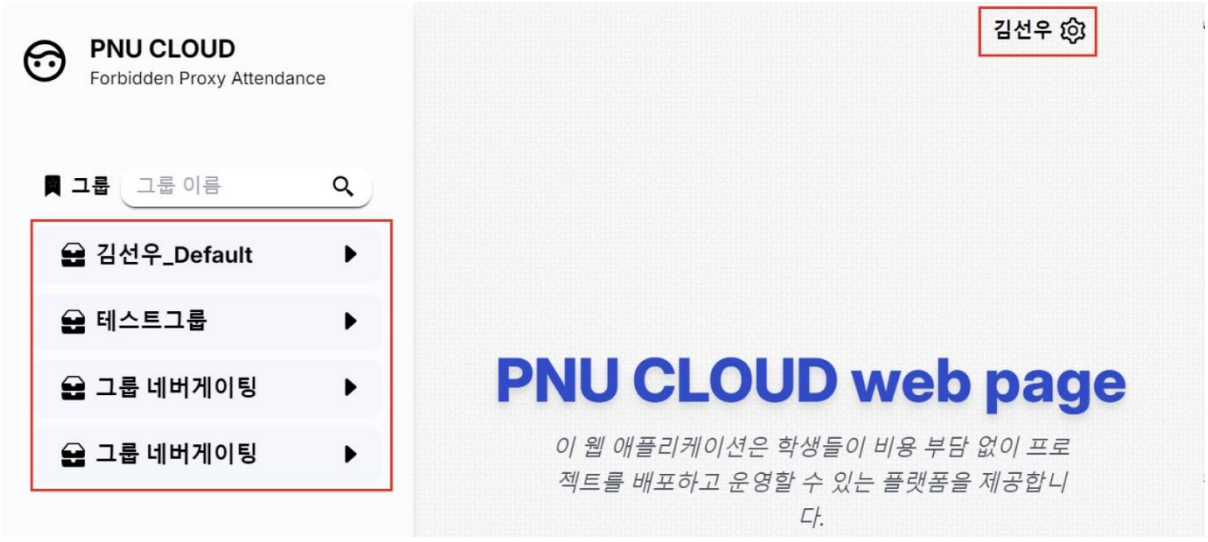
3. 리소스 접근 허용/거부

인증된 사용자이고, 해당 리소스에 대한 접근 권한이 있다면 요청을 허용한다. 그렇지 않은 경우, 요청을 거부하고 적절한 오류 응답을 반환한다.

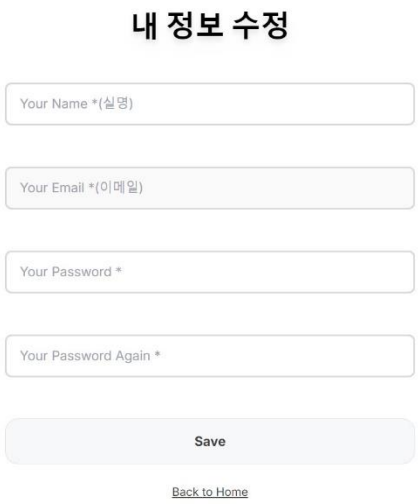
3.2.4 메인 화면



처음 서비스를 이용하는 사용자들을 위해 메인 화면에서 서비스에 대한 간략한 소개와 설명을 제공한다. 또한 왼쪽에 사용자가 가지고 있는 그룹들과 프로젝트들로 이동할 수 있는 네비게이션 바가 존재하고, 오른쪽 위 톱니바퀴 모양의 아이콘을 클릭하면 로그아웃과 내 정보 페이지로 이동할 수 있다.

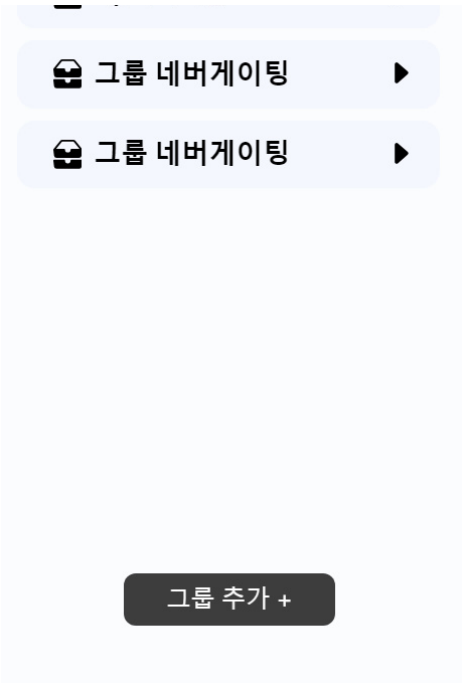


3.2.4 내 정보 수정

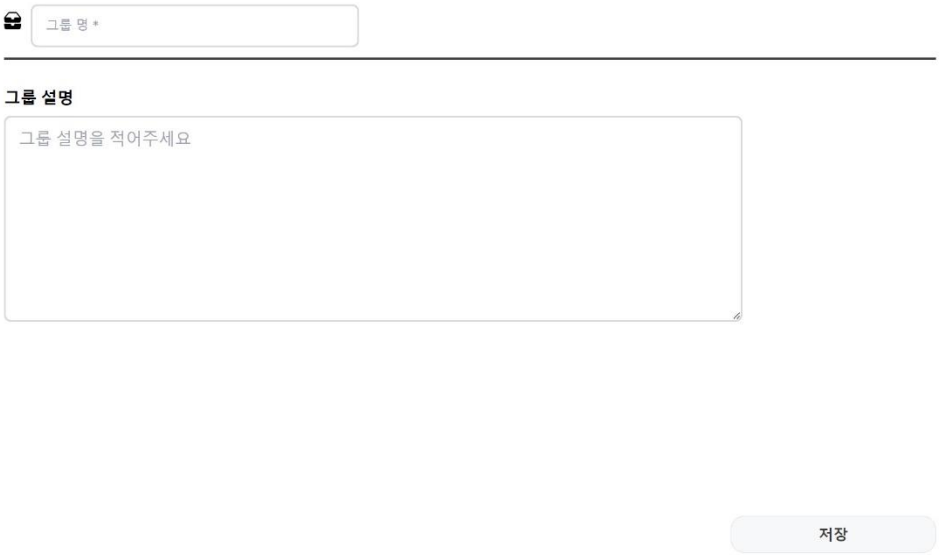


사용자는 내정보페이지에서 이메일, 이름 등의 정보를 변경할 수 있다.

3.2.5 그룹 생성



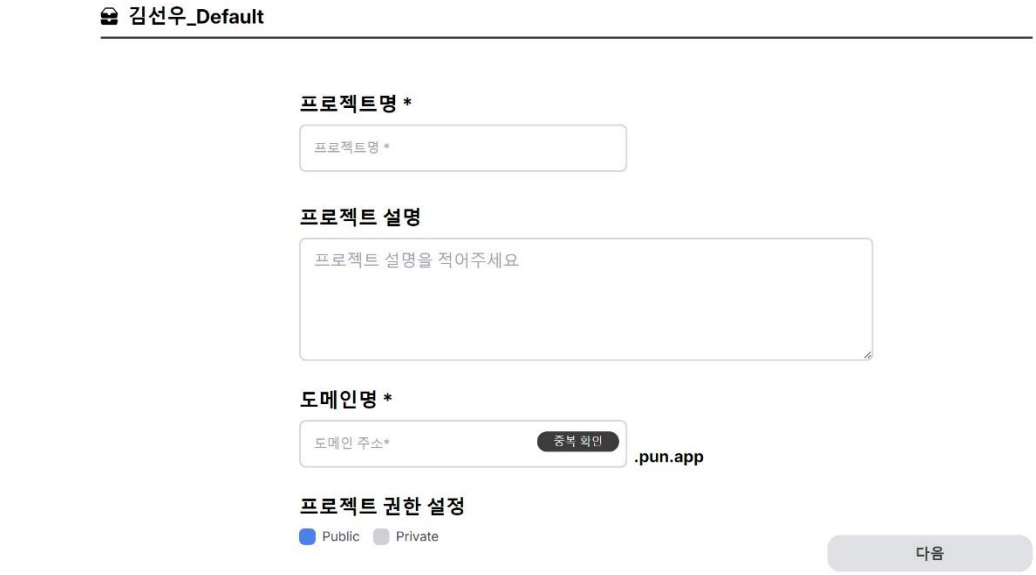
네비게이션 바 하단의 "그룹 추가+" 버튼을 눌러 그룹 생성 페이지로 이동할 수 있다.



사용자는 그룹 명과 그룹 설명을 입력한 후 저장 버튼을 눌러 그룹을 생성할 수 있다.

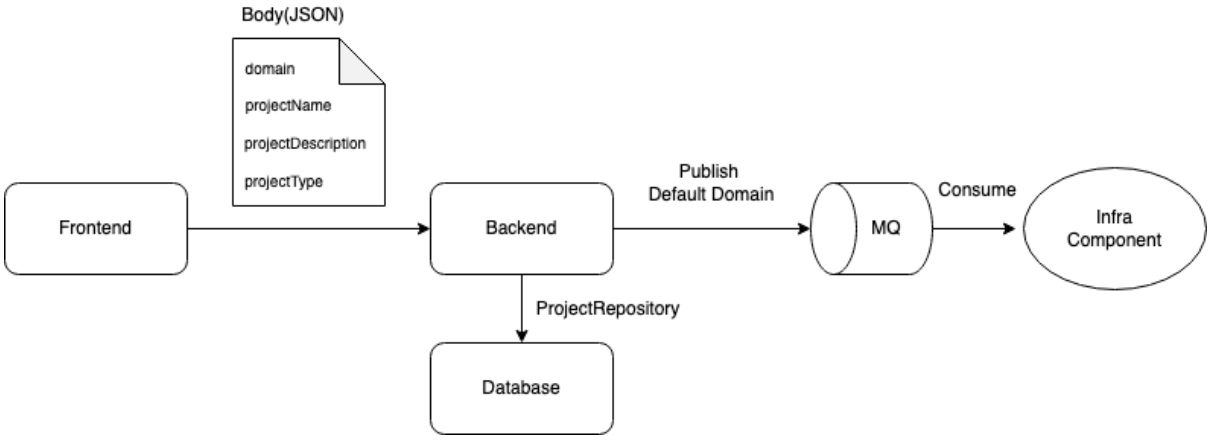
3.2.6 프로젝트 생성

사용자는 [그룹 페이지]에서 "프로젝트 추가" 버튼을 통해 프로젝트 생성 페이지로 이동한다.



사용자는 프로젝트 명, 프로젝트 설명, 도메인 명, 프로젝트 권한 설정을 입력한 후 "다음" 버튼을 통해 프로젝트를 생성한다.

[프로젝트 개요 등록 및 기본 도메인 발급]



프론트엔드는 아래와 같은 JSON 형식 데이터를 백엔드 어플리케이션(Spring Boot)으로 전송한다.

```
{
	"domain": "test.app", # 프로젝트 도메인
	"projectName": "test project", # 프로젝트 이름
	"projectDescription": "this is test project", # 프로젝트의 짧은 설명
	"projectType": "PUBLIC" # 프로젝트 공개/비공개 여부
}
```

백엔드 어플리케이션은 위 데이터를 역직렬화하고, ProjectRepository를 거쳐 데이터베이스 상에 프로젝트를 생성한다.

프로젝트 생성 이후, 인프라 컴포넌트를 통해 프로젝트 도메인의 SSL인증서를 발급받는다.

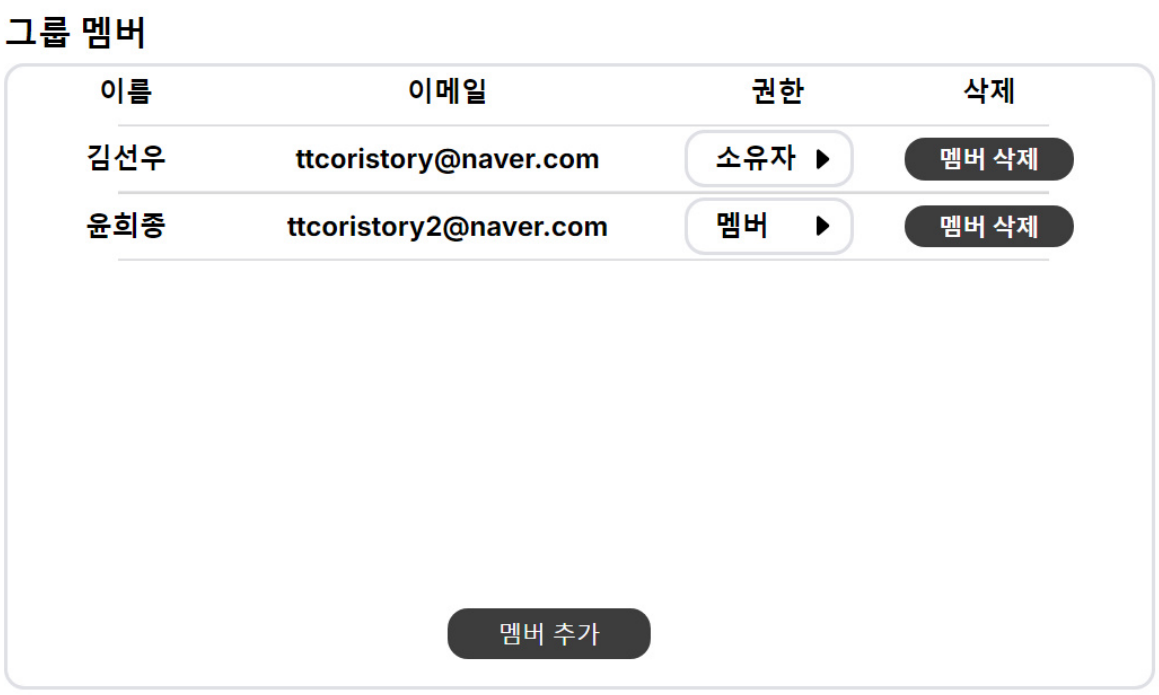
SSL 인증서를 발급받기 위하여, Spring boot는 메세지 큐(Message Queue)로 사용 중인 Redis에 아래의 Key-Value쌍을 List Operation을 사용하여 Publish한다.

```
“domain”: “test.app”
```

메세지 큐를 폴링 중인 인프라 컴포넌트는 Consume하고, Let’sEncrypt를 통해 SSL 인증서를 발급한다.

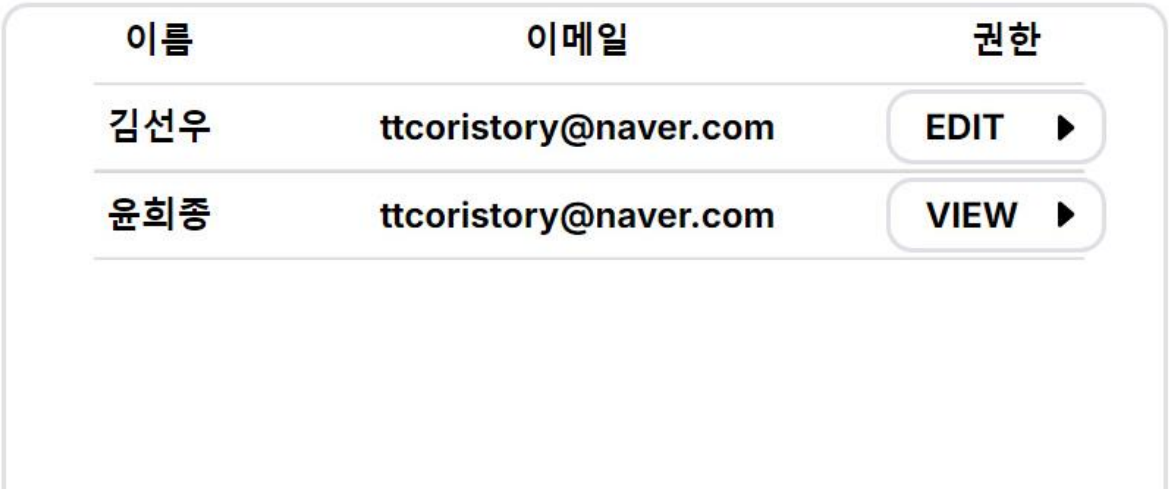
3.2.7 멤버 관리

- **그룹 멤버 관리**



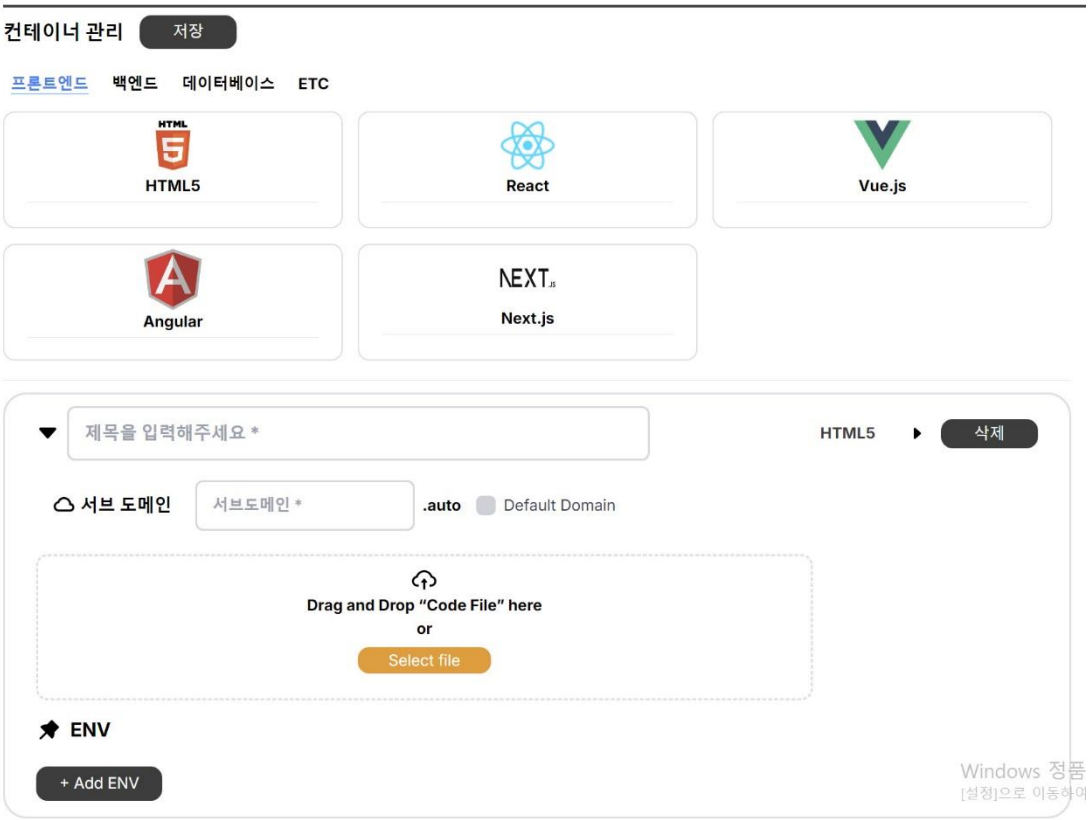
그룹 소유자는 멤버들의 권한을 관리하고, 삭제 할 수 있다. 그룹 권한의 종류는 소유자와 멤버가 있다.

- **프로젝트 멤버 관리**



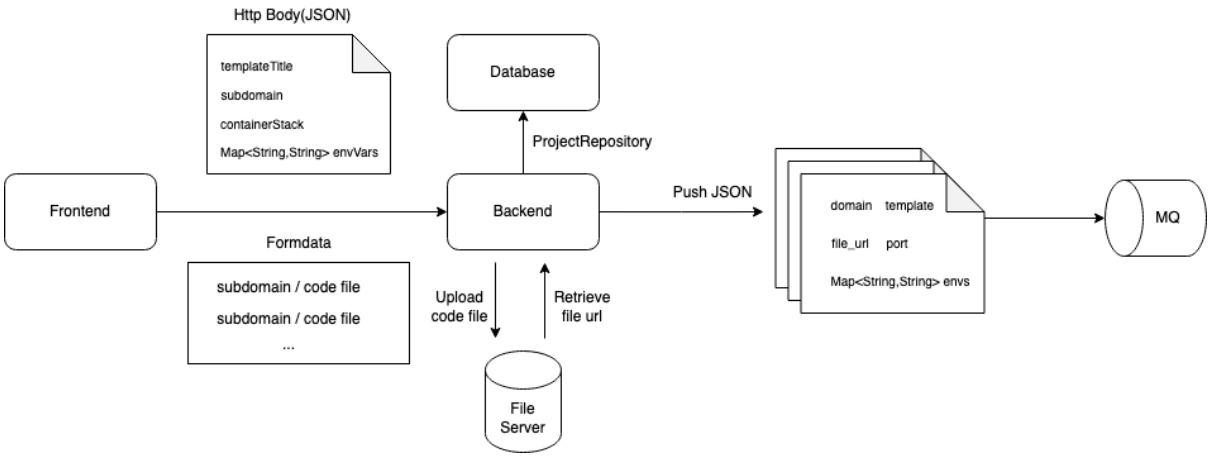
프로젝트의 Edit 권한을 가진 사용자는 멤버들의 권한을 관리할 수 있고 그룹의 멤버들은 프로젝트의 멤버들로 상속된다. 프로젝트 권한의 종류는 EDIT, VIEW가 있다.

3.2.8 프로젝트 내 컨테이너 생성 및 배포



사용자는 컨테이너 스택, 컨테이너 제목, 서브 도메인, 파일, 환경 변수를 입력해 컨테이너를 생성할 수 있다.

[프로젝트 내 컴포넌트(컨테이너) 등록]



프론트엔드는 프로젝트 내에 배포될 컨테이너의 메타데이터와, 코드 파일을 전송하기 위해 http의 Content-Type을 Multipart/formdata로 설정하여 전송한다.

```
# Formdata
{
   "projectData" : JSON, # 프로젝트 내에 배포될 전체 컨테이너들의 메타데이터를 지정한다. (필수)

   "subdomain-1": BinaryFile,
   # 프로젝트 내 개별 컨테이너의 서브도메인을 key로, 구동될 코드파일을 value로 지정한다.

   "subdomain-2": BinaryFile,
   ... # 프로젝트 내 배포될 컨테이너 개수에 따라 Formdata내 subdomain 개수는 가변적이다.(선택)
}
```

Formdata는 key-value 쌍으로 이루어져 있으며,

projectData(key)에 대응하는 value에는 프로젝트 내 컨테이너들의 메타데이터를 담고,

이후 key-value 쌍에는 프로젝트 내 배포될 컨테이너들의 서브도메인을 key로, 해당 컨테이너에서 구동될 코드 파일을 value로 담는다.

상세 내용은 다음과 같다.

(1) 프로젝트 메타데이터(JSON) 지정

프론트엔드는 아래와 같은 JSON 데이터를 백엔드 어플리케이션(Spring Boot)으로 전송한다.

```
# formdata[“projectData”]
{
   "projectId": 9, # 컨테이너를 배포할 프로젝트 ID
   "templates": [
      {
         "templateTitle": "Frontend - React", # 컨테이너의 이름
         "subdomain": "fe.test.pnu.app", # 컨테이너의 서브도메인
         "containerStack": "React v0.1.1", # 컨테이너 배포 템플릿의 환경 또는 프레임워크 스택
         "envVars": { # 컨테이너 내부 환경변수
            "ENV": "production"
         }
      },
      {
         "templateTitle": "Backend - spring",
         "subdomain": "be.test.pnu.app",
         "containerStack": "Spring v0.1.1",
         "envVars": {
            "API_KEY": "TEST_API_KEY",
            "ENV": "production"
         }
      }
```

```
    ]
}
```

(2) 프로젝트 실행 코드 파일 지정

위 JSON 형식 데이터는 프로젝트 내에 배포될 컨테이너의 정보들이며,

컨테이너 내 실행될 코드 파일은 formdata의 **key**(subdomain):**value**(file) 형식으로 함께 전송된다.

```
"fe.test.pnu.app" : Binary File,
"be.test.pnu.app" : Binary File
```

백엔드 어플리케이션은 위 데이터를 역 직렬화하여, ContainerDetailRepository를 통해 컨테이너의 정보를 데이터베이스에 저장하고, 각 컨테이너의 실행파일은 파일서버에 저장한 후, File URL을 반환받는다.

이후 인프라 컴포넌트에게 실제 프로젝트 내에 컨테이너들의 정보를 전송하기 위해, 반환받은 File URL과 프론트엔드로부터 받은 메타데이터를 조합하여, 아래의 JSON 형식 데이터를 만든다.

```
{
   "domain": "test.pnu.app", # 프로젝트의 도메인
   "container": [ # 프로젝트에 속한 컨테이너 정보 리스트
      {
         "id": 1,
         "domain": "fe.test.pnu.app", # 컨테이너에게 할당된 서브도메인
         "template": "React v0.1.1", # 컨테이너 배포 템플릿 환경 또는 프레임워크
         "file_url": https://file-url.com/23n1kj3n24, # 컨테이너에서 실행될 코드 파일 URL
         "port": 3234, # 컨테이너에 할당된 포트
         "env": { # 컨테이너 내 환경변수 리스트
            "ENV": "production"
         }
      },
      {
         "id": 2,
         "domain": "be.test.pnu.app",
         "template": "Spring boot v0.2.1",
         "file_url": https://file-url.com/2nn1kj31354,
         "port": 3235,
         "env": {
            "API_KEY": "TEST_API_KEY",
            "ENV": "production"
      }
   }
]
}
```

최종적으로, 위 JSON을 String으로 변환하고 메세지 큐인 Redis에 List Operation을 사용해 Publish한다.

```
"projectDetail": JSON(String)
```

메세지 큐를 폴링 중인 인프라 컴포넌트는 위 JSON을 consume하면 사전에 정의된 프레임워크별 템플릿을 통해 컨테이너 이미지 빌드하여 Harbor(Private Docker Registry)에 업로드하고, Kubernetes에 deployment한다.

```
#콘솔 프로젝트 배포 템플릿 (파일 복사 이전 작업은 사전에 이미지로 빌드하여 활용)
FROM ubuntu:latest

RUN apt-get update && apt-get install -y ₩
    cmake ₩
    git ₩
    g++ ₩
    pkg-config ₩
    libwebsockets-dev ₩
    libjson-c-dev ₩
    libssl-dev ₩
    vim ₩
    language-pack-ko ₩
    && rm -rf /var/lib/apt/lists/*
RUN locale-gen ko_KR.utf8
RUN dpkg-reconfigure locales
RUN git clone https://github.com/tsl0922/ttyd.git /ttyd && ₩
    cd /ttyd && ₩
    mkdir build && cd build && ₩
    cmake .. && ₩
    make && make install
COPY {file_name} /usr/local/bin/{file_name}
RUN chmod +x /usr/local/bin/{file_name}

CMD ["ttyd", "-W", "-t", "disableReconnect=true", "-t", "fontSize=20", "-t", "titleFixed=UNIX", "/usr/local/bin/chat"]
```

3.2.9 컨테이너 삭제하기

컨테이너 목록에서 삭제 버튼을 누르면 해당 컨테이너가 삭제된다.

3.2.10 배포된 프로젝트 접속

사용자가 프로젝트에 할당된 도메인으로 접속하면 OpenResty에서 Lua 스크립트를 활용하여 Redis 인메모리 데이터베이스에 저장된 정보를 바탕으로 연결 정보를 설정하여 리버스 프록시로 컨테이너를 연결하고 Auto Scaling을 위한 정보를 전달한다

```
location / {
    set $backend_url "";
    access_by_lua_file /etc/nginx/conf.d/vhost_lookup.lua;

    proxy_pass $backend_url;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    log_by_lua_block {
        ngx.log(ngx.ERR, "cert_path: ", ngx.var.cert_path)
        ngx.log(ngx.ERR, "key_path: ", ngx.var.key_path)
    }
}
```

배포된 서브 도메인에 맞는 인증서를 적용하여 HTTPS 통신을 지원하기 위해 Lua 스크립트에서 요청의 SNI 를 확인하여 사전에 발급한 SSL 인증서를 적용한다.

각인증서에는 서브도메인과 서브도메인 하위 와일드카드 도메인으로 (sub).pnu.app, *.(sub).pnu.app에 대해 구성되어 있다.

```
server {
    listen 443 ssl;
    server_name _;
                                                ...
    ssl_certificate_by_lua_block {
        local ssl = require "ngx.ssl"
        local domain, err = ssl.server_name()

        if not domain then
            ngx.log(ngx.ERR, "failed to get SNI: ", err)
            return
        end

        local domain = domain:match("^%w+%.(.+)")
        local cert_path = "/etc/letsencrypt/live/" .. domain .. "/fullchain.pem"
        local key_path = "/etc/letsencrypt/live/" .. domain .. "/privkey.pem"
                                                ...
        assert(ssl.clear_certs())
        assert(ssl.set_der_cert(cert_der))
        assert(ssl.set_der_priv_key(key_der))
    }
}
```

4 연구 결과 분석 및 평가

본 연구는 **사용자 편의성 관점**에서 상용 클라우드 서비스를 이용한 배포 방식과 비교 분석 할 수 있다.

사용자가 프로젝트를 상용 클라우드 서비스를 통해 배포하는 방법은 다양하지만,

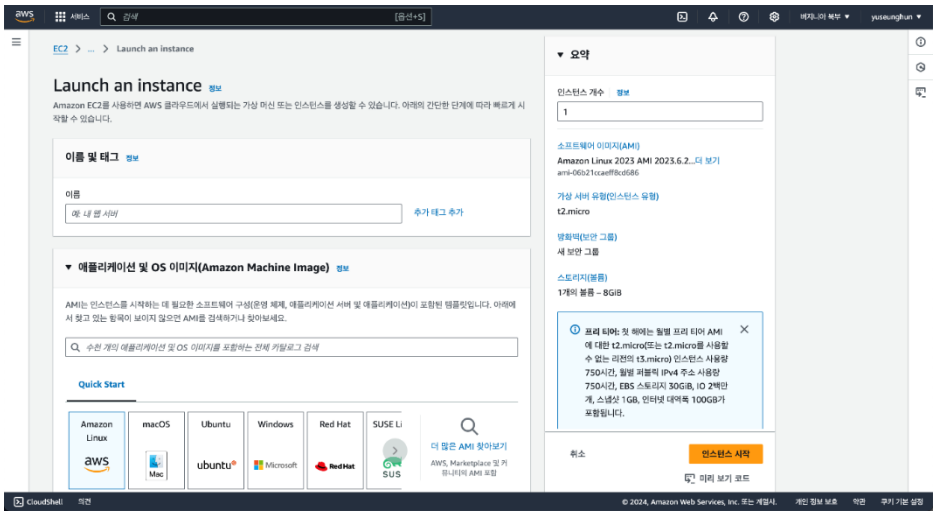
AWS의 EC2를 대여하여 직접 Spring boot 프로젝트를 배포하는 방식과 본 연구의 배포 방식을 비교하고자 한다.

[AWS EC2를 통한 Spring boot 프로젝트 배포]

상용 클라우드 서비스를 이용한 기존 배포 방식은 아래의 과정을 거친다.

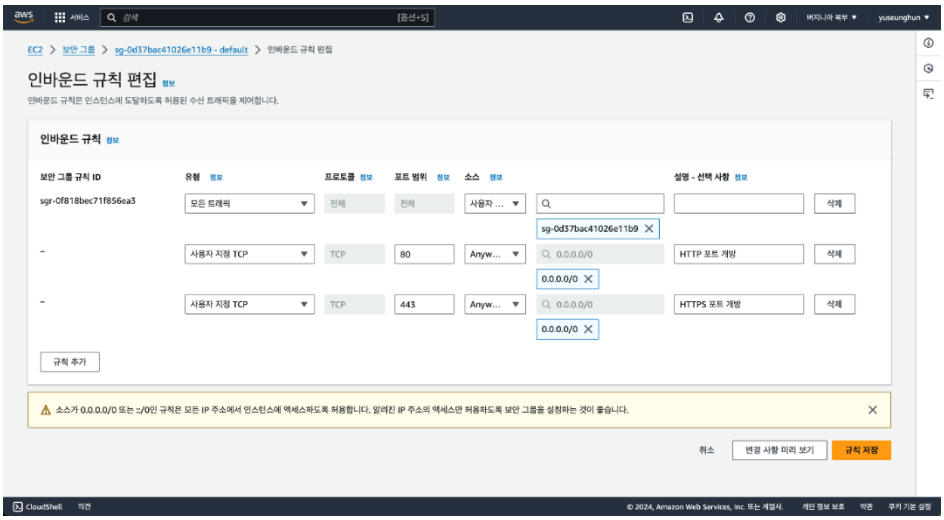
(1) AWS EC2 대여

- 리소스 사양 및 운영체제를 선택하고, EC2 인스턴스를 시작한다.



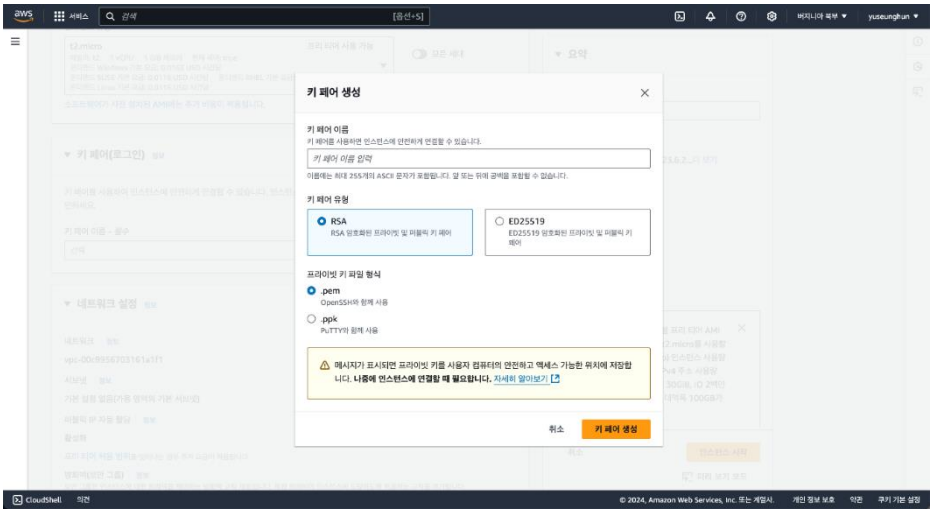
(2) 인바운드 규칙 설정

- 웹서버(80, 443)으로의 인바운드 트래픽을 허용하기 위해, 보안 그룹의 규칙을 수정해 특정 포트를 개방한다.



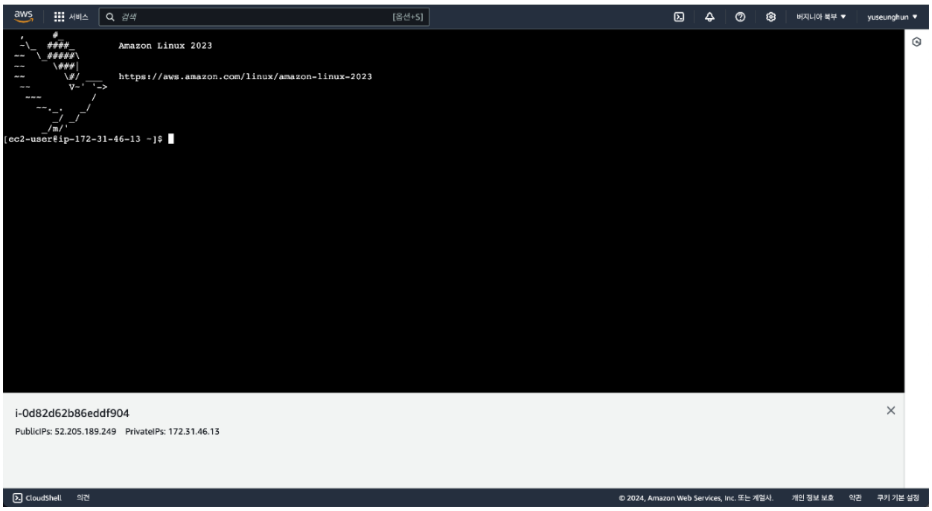
(3) SSH keypair 발급

- EC2 인스턴스에서의 원활한 작업을 위해 SSH keypair를 발급한다.



(4) EC2에 SSH 접속

- 발급받은 SSH 키를 이용해 AWS EC2 인스턴스에 접속한다.



(5) OpenJDK 설치

- JRE(Java Runtime Environment)가 포함된 JDK를 설치하여 자바 프로그램이 구동되게 한다.

```
# 패키지 업데이트
sudo apt-get update && sudo apt-get upgrade


# JDK 설치
sudo apt-get install openjdk-11-jdk


# 환경변수 설정
vim ~/.bashrc
export JAVA_HOME=$(dirname $(dirname $(readlink -f $(which java))))
export PATH=$PATH:$JAVA_HOME/bin
source ~/.bashrc
```

(6) 프로젝트 Git Clone

- 프로젝트의 소스코드를 EC2 인스턴스 내에 복제한다.

```
# 프로젝트 clone
git clone https://github.com/[Organization]/[Project].git
```

(7) Gradle을 이용해 프로젝트 빌드

- 복제된 자바 프로젝트에서 각종 의존성을 설치하고, 프로젝트를 빌드한다.

```
# 프로젝트 디렉토리로 이동
cd [project directory]

# gradlew를 이용해 자바 프로젝트 빌드
./gradlew build
```

(8) systemctl에 서비스로 등록 후 프로젝트 구동

- 자바 어플리케이션을 서비스로 등록하여 백그라운드 데몬으로 구동한다.

```
# 서비스 파일 생성
vi /etc/systemd/system/${service name}.service
```

이처럼 상용 클라우드 서비스를 사용하는 배포 방식은 아래 과정의 책임이 모두 사용자에게 있으며, 이 과정을 모두 개발자가 직접 수행하고, 운영 도중 발생할 수 있는 인프라적 결함을 모니터링 할 수 있는 관제시스템의 필요성도 요구된다.

1. 리소스 대여
2. 의존성 및 런타임 설치
3. 네트워크 및 방화벽 설정
4. 서비스 구동 및 관리

뿐만 아니라, 웹 서비스 운영을 위해 도메인 제공 업체에서 도메인을 구입하고 DNS 레코드 설정 및 SSL 인증서를 도메인에 연결하는 것 또한 사용자가 담당한다.

상용 클라우드 서비스 이용 시 비용 부담
클라우드 리소스 대여, 도메인 구입 및 유지, SSL 인증서 구입 및 유지

이처럼 직접 리소스를 대여하여 프로젝트를 배포하는 방식은 각종 기술적 난관과 비용을 수반한다.

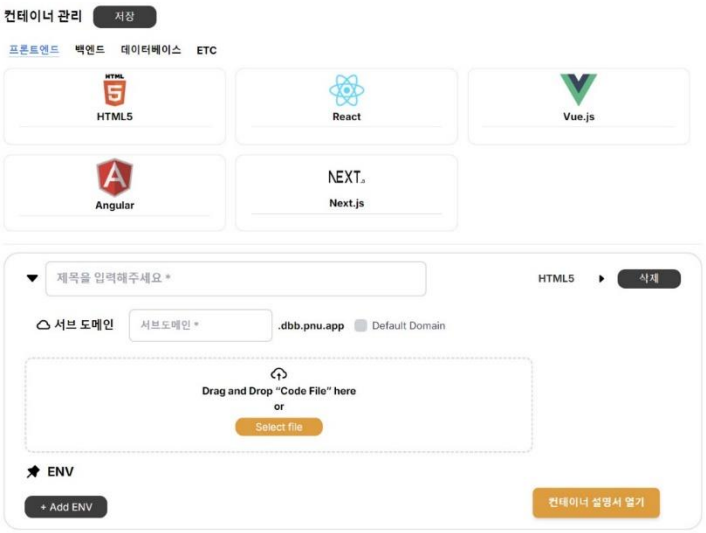
[본 연구의 웹 서비스를 통한 프로젝트 배포]

(1) 프로젝트 도메인 등록

- 프로젝트의 기본 도메인을 발급받는다.

(2) 프로젝트 컨테이너 등록 및 배포

- 프로젝트 내 배포할 컨테이너들의 정보(환경 및 프레임워크, 서브도메인)와 컨테이너 내 구동될 소스코드를 첨부하여 배포한다.



상용 클라우드 서비스와 비교하여 본 연구의 웹 서비스를 활용하면, 리소스 할당부터 서비스 구동 및 관리까지 모두 시스템이 책임지며, 사용자는 기술적 난이도가 있는 배포 과정을 GUI 인터페이스로 쉽고 빠르게 수행이 가능하다.

뿐만 아니라, 시스템이 제공하는 1차 도메인(ex. pnu.app)의 하위 도메인으로 프로젝트의 도메인을 설정하므로, 도메인 발급과 유지를 모두 시스템이 담당하여 기술적, 비용적인 부담을 경감한다.

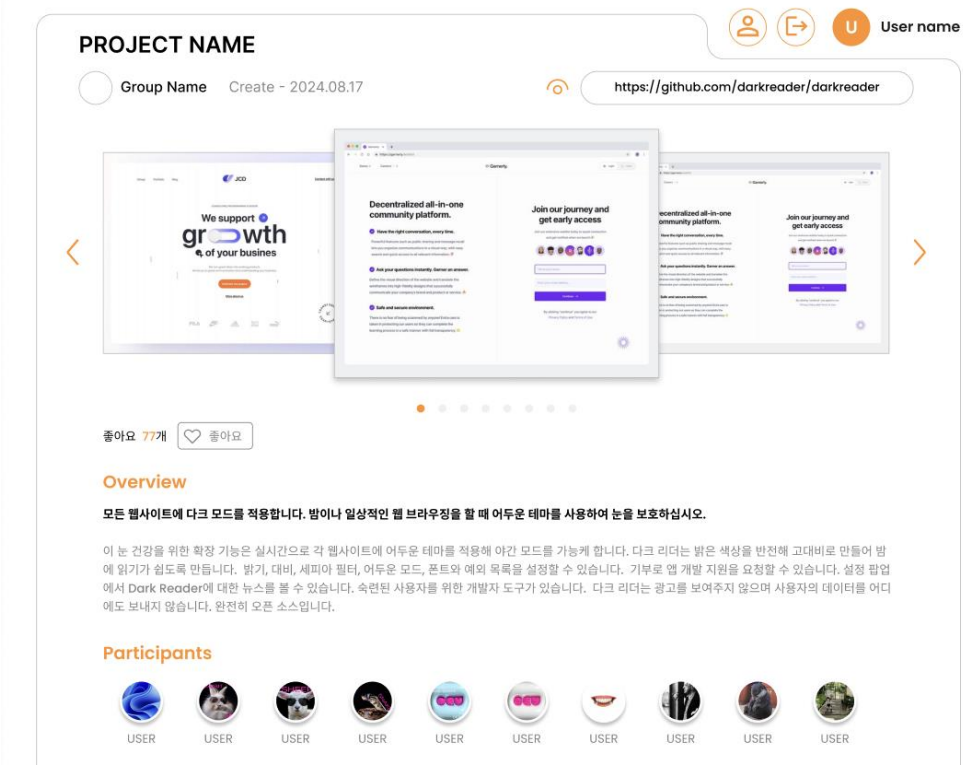
또한 시스템 적인 관점에서, 접속이 있을 때만 컨테이너를 구동시키는 scale to zero를 적용하여, 제한적인 리소스 내에 더 많은 프로젝트를 배포할 수 있다.

이는 학생들이 간단한 서비스를 배포하는 본 연구의 웹서비스 특성 상, 동시 접속이 많지 않은 환경에서 효율적으로 리소스를 운용할 수 있다는 장점이 있다.

5 결론 및 향후 연구 방향

우리는 학생들에게 서버리스 프로젝트 배포 시스템을 제공함으로써, 단순화된 프로젝트 배포 과정으로 쉽고 빠르게 프로젝트를 배포할 수 있는 환경을 구축한다. 이 시스템은 학생, 교수자, 그리고 학교 모두에게 다양한 이점을 제공한다.

학생들은 웹 서비스를 통해 복잡한 인프라 운영 및 관리 없이도 자신의 프로젝트를 쉽고 빠르게 배포할 수 있으며, 이를 통해 프로젝트 성과를 효과적으로 공유할 수 있다. 교수자 입장에서는 학생들의 프로젝트를 보다 쉽게 평가할 수 있는 환경이 조성되어 교육의 질을 높일 수 있다. 학교 측면에서는 학생들의 성과를 체계적으로 관리할 수 있으며, 서버리스 아키텍처를 통해 최소한의 비용으로 시스템을 운영할 수 있는 이점이 있다.



향후 연구 방향으로는 프로젝트 개요 페이지의 고도화를 통해 성과 공유를 더욱 활성화하는 것을 고려한다. 이를 통해 학생들의 프로젝트가 더 널리 알려지고, 교내외에서 더 많은 관심과 피드백을 받을 수 있는 환경을 조성하고자 한다.

또한, 이 시스템을 바탕으로 다양한 교육 기관이나 기업에서 활용할 수 있는 확장 가능성도 모색한다. 예를 들어, 공공기관이나 기업에서 진행하는 클라우드 전환 사업에 맞춰 이 시스템을 적용하여, 더 넓은 범위의 사용자들에게 서비스를 제공할 수 있을 것으로 기대한다. 궁극적으로 이 프로젝트는 학습과 개발의 연속성을 보장하고, 물리적 제약을 뛰어넘는 효율적인 프로젝트 관리 및 공유 플랫폼으로 발전해 나갈 것이다.

6 역할 분담

팀원	역할
신예준	**서비스 기획** - 서비스 기획 및 기능 정의 **아키텍쳐 설계 및 인프라 모듈 개발** - 전체 아키텍쳐 설계 및 구성 - 인프라 모듈 개발
유승훈	**데이터베이스 설계** - 엔티티 개념적 설계 및 논리적 설계 **백엔드 개발** - Spring Security를 활용한 인증/인가 처리 - REST API 제공
김선우	**UI 설계 및 디자인** - 요구 사항 기능서를 바탕으로 UI 설계서 작성 **프론트엔드 개발** - 웹 서비스에 필요한 UI 구현 - API 서버와 Rest 통신 연결 및 오류 예외 처리