- 2024 전기 졸업과제 중간보고서 -

컨테이너 오케스트레이션 기반 서버리스 프로젝트 배포 시스템 구축

부산대학교 정보컴퓨터공학부
C분과 37조 zero

지도교수: 탁성우
201924501 신예준
201924515 유승훈
201924435 김선우

[목차]

1. 과제 개요

학생들은 텀프로젝트나 비교과 활동을 통해 다양한 프로젝트를 개발하지만 실행 가능한 형태로 결과물을 공유하는 경우는 드물다. 이처럼 결과물을 배포하여 제공하지 않는 주된 이유는 상용 클라우드 서비스의 비용 부담이나 배포 과정에서 기술적인 어려움이 있기 때문이다. 그렇기에 대부분 본인 개발 환경에서 실행한 화면을 녹화하여 보여주는 간접적인 방식을 통해 결과를 공유하고 있다.

이에 본 과제에서는 서버리스 프로젝트 배포 시스템을 개발하여 학생들이 비용이나 기술적 장벽 없이 자유롭게 자신의 프로젝트를 배포하고 공유할 수 있도록 하는 것을 지원하고자 한다. 시스템은 프로젝트를 배포 템플릿을 활용해 컨테이너로 쉽게 배포할 수 있도록 하고, Auto Scaling과 Scale to 0를 적용함으로써 평소에는 비활성 상태로 유지하여 자원 사용을 최소화한다. 이러한 구성을 통해 제한된 자원을 효율적으로 사용해 많은 프로젝트를 배포할 수 있다. 이를 통해 학생들은 자신의 프로젝트를 배포하여 실행 가능한 형태로 공유함으로써 실제 피드백을 받아볼 수 있고 포트폴리오로도 활용할 수 있습니다.

2. 요구조건 및 제약 사항 분석

2.1. 요구 조건

[프론트엔드]

- **프론트엔드 프레임워크**
  리액트(React) 라이브러리를 도입하여, 공통 컴포넌트의 분리를 통해 효율적인 코드 관리와 유지보수를 구현할 계획이다. 또한, 리액트의 SPA(Single Page Application) 렌더링 방식을 활용하여 페이지 간 전환 시 부드럽고 원활한 사용자 경험(UX)을 제공한다.

- **프론트엔드 데이터 관리**
  TanStack Query 라이브러리의 강력한 데이터 페칭, 캐싱, 동기화 기능을 통해 서버 상태와 클라이언트 상태 간의 일관성을 유지하고, 서버와의 데이터 동기화가 필요한 본인이 속한 그룹 및 프로젝트, 멤버 데이터를 효율적으로 관리한다.

- **프론트엔드 UI 설계**
  Chart.js 라이브러리를 사용해 실행 중인 컨테이너의 상태를 사용자에게 효과적으로 시각화하여 제공해야 해 사용자에게 직관적이고 이해하기 쉬운 UI를 제공한다.

- **(변경) 과제 기능 및 채점자 기능 제거**
  플랫폼의 주요 목적인 프로젝트 배포 및 컨테이너 관리 기능에 집중하기 위해 설계 시 계획하였던 과제 기능 및 채점자에 관련된 기능을 제거한다.

[백엔드]

- **백엔드 프레임워크**
  스프링 부트 프레임워크를 도입하여, 모듈화된 아키텍처와 의존성 주입을 통해 효율적인 코드 관리와 유지보수를 구현할 계획이다. 또한, 스프링 부트의 자동 구성(Auto-configuration) 기능과 내장 서버를 활용하여 빠른 개발과 배포를 가능하게 하고, RESTful API 구현을 통해 클라이언트와의 원활한 데이터 통신을 제공하려고 한다.

- **백엔드 인증 시스템**
  Spring Security와 JWT(JSON Web Token)를 활용하여 강력하고 확장 가능한 인증 및 인가 시스템을 구현할 계획이다. 스프링 시큐리티의 모듈화된 아키텍처와 의존성 주입을 통해 보안 로직을 효과적으로 분리하고 관리할 수 있다. 또한, JWT를 이용한 토큰 기반 인증 방식으로 서버의 상태를 저장하지 않는(stateless) 확장 가능한 인증 시스템을 구축하려고 한다.

- **백엔드 보안 시스템**
  스프링 부트의 자동 구성 기능을 활용하여 보안 설정을 간소화하고, RESTful API에 대한 세밀한 접근 제어와 권한 관리를 구현하여 안전하고 효율적인 데이터 통신을 제공한다.

- **백엔드 데이터베이스 시스템**
  Spring Data JPA를 도입하여 효율적이고 유지보수가 용이한 데이터 접근 계층을 구현할 계획이다. JPA(Java Persistence API)를 활용한 객체-관계 매핑(ORM)을 통해 데이터베이스 작업을 추상화하고, 객체 지향적인 방식으로 데이터를 관리하려고 한다.

[인프라]

- **사용자 비용 최소화**
  배포 및 운영을 위한 비용을 최소화하여 기존에 비용 문제로 배포가 어렵거나 지속적으로 운영되지 못하였던 문제를 해결한다. 이를 위해 시스템에서는 하위 도메인을 제공하여 기존에 발생하였던 도메인 구입 비용을 선택적인 요소로 변경하고, 서버리스 배포 환경을 제공하여 서버 구축 및 유지에 발생하는 비용을 최소화 할 수 있도록 설계한다.

- **자원 효율적 분배**
  저비용 고효율 시스템 구축하기 위해 Auto Scaling을 통해 트래픽에 따라 컨테이너를 자동으로 스케일링하여 원활한 서비스를 유지할 수 있도록 하고, Scale to 0를 적용하여 평소에는 컨테이너를 비활성화 상태로 유지하여 리소스 사용량을 최소화한다. 이후 요청이 발생한다면 다시 컨테이너를 특정 시간 동안 실행하여 요청을 처리한다.

- **사용자 친화적 배포 과정**
  사전에 프로젝트 유형별로 컨테이너 이미지 빌드를 위한 템플릿을 작성하여 배포 과정을 모르는 사용자도 프로젝트 파일을 업로드하고 환경 변수만 지정하면 자동으로 빌드 프로세스가 진행되고 컨테이너가 배포가 진행되어 쉽고 빠르게 진행할 수 있다.

- **다양한 유형의 프로젝트 지원**
  HTTP로 통신하는 웹 프로젝트 외에도 콘솔 프로그램 등 다양한 형식의 프로젝트 배포를 지원한다. 이를 위해 웹 터미널을 통해 콘솔 프로그램도 운영할 수 있도록 구현합니다. 또한 SQL 컨테이너 등을 지원하여 외부에서 접근하기 위한 프록시 서버를 구성하여 HTTP 이외 프로토콜을 사용하더라도 내부 컨테이너에 연결할 수 있어야 한다.

- **컨테이너 데이터 유지**
  사용자는 외부 파일 시스템의 도움 없이 자제적으로 프로젝트 내에 데이터를 유지하여 프로젝트가 지속적으로 운영될 수 있도록 한다.

- **프로젝트 접근 제어**
  사용자가 프로젝트를 외부에 공유하기 위해 공개적으로 배포할 수도 있지만 내부에서만 확인하거나 테스트를 위해 배포하는 경우 비공개 배포를 원하는 경우도 있다. 따라서 인증 시스템을 구성해 특정 사용자만 프로젝트에 접근이 가능하도록 인가 시스템을 구성한다.

- **(추가) 안전하고 신뢰성 있는 시스템 구축**
  프로젝트에는 민감한 정보들이 포함되어 있을 수 있으므로 기밀성을 유지할 수 있는 설계를 적용하고 가용성과 무결성을 보장하기 위해 백업이나 이중화 등의 조치가 가능하도록 설계한다.

2.2. 제약 조건

[프론트엔드]

- **사용자 데이터 입력 폼 제약 조건**
  플랫폼의 특성상 사용자로부터 입력되는 데이터가 많으므로, 정규 표현식 및 확장자명 확인과 같은 방법을 통해 사용자 입력 데이터를 제한한다.

[백엔드]

- **백엔드 인증 방식**
  인프라 서버와 스프링 부트 애플리케이션 서버 간에 공통 JWT secret을 사용하여 일관된 보안 체계를 구축하려고 한다. 이를 통해 마이크로서비스 아키텍처에서 서비스 사이 원활하게 인증 정보 공유와 권한을 검증할 수 있다.

- **(변경) 백엔드 이메일 인증 설계**
  회원가입 시 이메일 인증 방식이 인증 링크를 보내주는 방식에서 6자의 인증 코드를 보내주는 방식으로 변경하였다. 이 과정에서 인증코드의 수명(ttl)을 설정하였고 이를 저장하기 위해 인메모리 Key-Value 스토어인 Redis를 사용하기로 하였다.

[인프라]

- **내부 네트워크 구성**
  보안을 위해 컨테이너는 내부 네트워크에 구성되며 인바운트 트래픽은 게이트웨이 혹은 프록시 서버로만 접근할 수 있다. 또한 다른 프로젝트의 컨테이너에 상호 접근할 수 없도록 설정되어야 한다.

- **HTTPS 강제 적용**
  게이트웨이를 통해 접근하는 통신은 모두 HTTPS로 이루어져야 하며 HTTP로 접근 시 HTTPS 주소로 리다이렉트한다. 이를 위해 프로젝트에 사용되는 도메인에 대해 SSL 인증서가 발급되어야 하며 유효기간 이전에 갱신하도록 시스템을 구성한다.

- **(변경) 인프라 아키텍쳐 설계**
  초기 설계 시 인메모리 데이터베이스(redis)와 메시지큐(kafka)를 별개로 구성 요소로 설계하는 등 현재 규모에서는 과도한 설계가 적용되었다고 판단하여 유사한 기능을 제공하는 요소들을 통일하여 설계를 단순화한다.

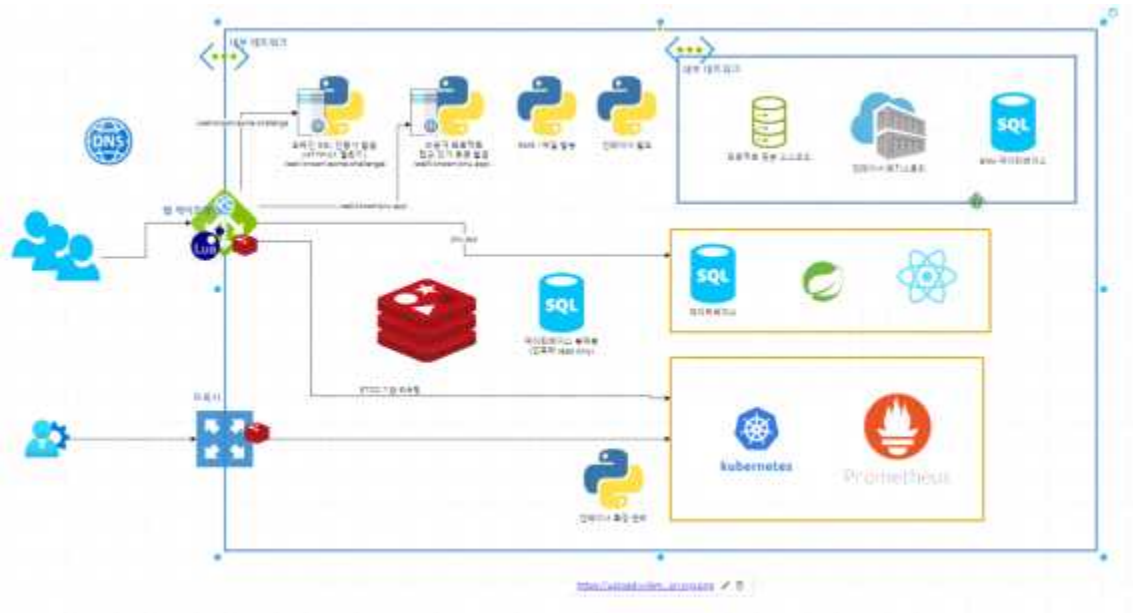
- **(변경) 사용자 프로젝트 환경 변수 관리**
  초기에는 환경 변수도 메인 데이터베이스에 함께 저장하였지만, API 키 등 민감한 정보들이 포함되어 유출될 경우 큰 피해가 발생할 수 있어 접근 권한 관리를 강화하고자 별도의 데이터베이스로 분리하고, 웹 서버에서는 읽기 권한 없이 쓰기 권한만 부여하여 데이터 기밀성을 보장하고자 하였습니다.

- **(변경) 사용자 프로젝트 컨테이너 데이터 유지**
  초기 설계 시 컨테이너 데이터 유지를 위해 변경사항을 이미지에 커밋하는 방식으로 설계하다. 하지만 사용자가 프로젝트 파일을 수정해 이미지를 다시 빌드할 경우 이전 데이터를 모두 손실하게 되었고 이를 해결하기 위해 별도로 데이터 유지를 위한 방안들을 설계하고 테스트하고 있다.

3. 설계 구체화

3.1. 인프라

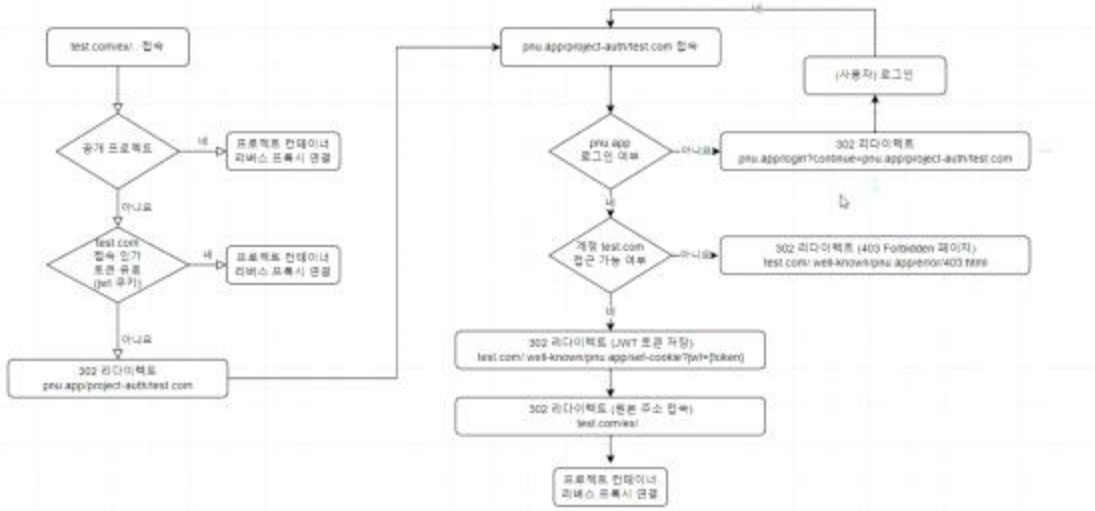
- 아키텍쳐 설계 및 내부 네트워크 구성
  전체적인 구성도는 아래와 같으며 사용자는 게이트웨이 혹은 프록시 서버를 통해 내부 네트워크에 접근할 수 있다.



- 게이트웨이 구조 설계
  게이트웨이는 HTTPS 접근만을 허용하며 HTTP 프로토콜에 요청한 host 기반으로 동적 라우팅을 수행한다. 이를 위해 OpenResty와 Lua 스크립트를 활용하여 ETCD에 저장된 포트를 기반으로 리버스 프록시를 연결하게 된다.

● 프로젝트 도메인 인증/인가 로직 구성

배포된 프로젝트는 서비스와는 다른 도메인을 가지게 된다. 따라서 웹 브라우저에서 JWT 토큰이 공유되지 못하기 때문에 인증이 필요한 경우 게이트웨이에서 프로젝트에 연결하는 것이 아닌 사전 구성된 내부 페이지로 리다이렉트하여 서비스에 로그인 된 인증 정보를 가져올 수 있도록 설계하였다.



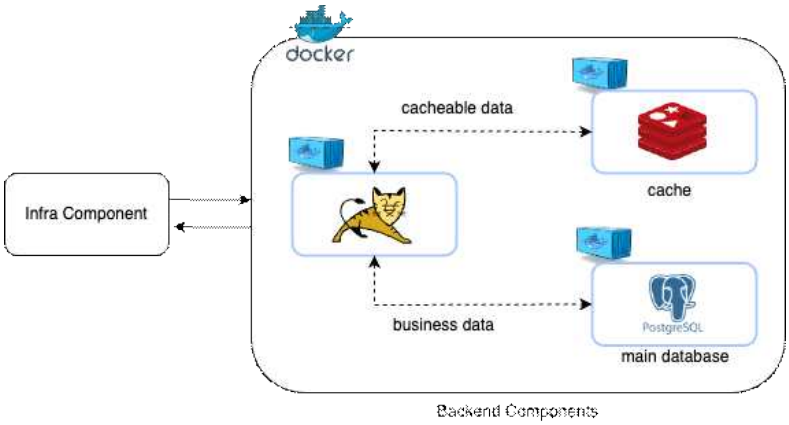
● SSL 인증서 발급 시스템

웹 브라우저를 통한 접속은 HTTPS로 강제하고 있으며, 이를 위해 도메인을 연결하면 자동으로 SSL 인증서를 발급받아야 한다. 이를 위한 시스템을 구성하였으며 관리자가 보유한 도메인에 대해서는 DNS-01 챌린지를 수행하여 하위 도메인에 대한 와일드카드 인증서를 발급하며, 사용자가 등록한 도메인은 HTTP-01 챌린지를 수행하여 해당 도메인에 대해서만 발급을 진행한다. 또한 Let's Encrypt 무료 인증서의 경우 유효기간이 3개월로 한정되기 때문에 연결된 도메인에 대해서는 주기적으로 갱신하는 로직을 포함하였다.

3.2. 백엔드

● 백엔드 아키텍처 설계

전체적인 구성도는 다음과 같으며, 백엔드 컴포넌트의 구성 요소는 WAS(Web Application Server), In Memory Cache, RDBMS로 구성되어 있으며, 각각 Tomcat, Redis, PostgreSQL을 사용한다.



백엔드 아키텍처

각각의 구성요소들은 도커라이징(Dockerizing)된 컨테이너들로 구성되어 있으며, Docker에서 지원하는 내부 네트워크를 통해 도메인 네임으로 서로 통신한다. 또한 인프라 자원 및 컴포넌트와는 메시지큐를 통해 비동기적으로 데이터를 주고받는다.

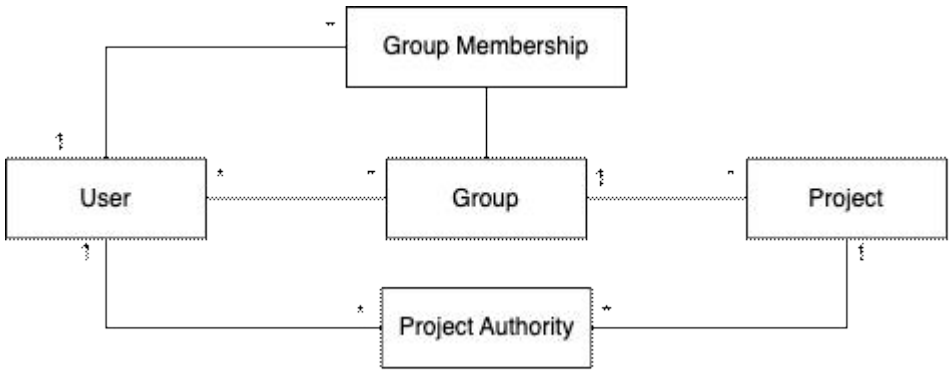
각각 구성요소들의 세부 역할은 다음과 같다.

구성요소	역할
Tomcat(WAS)	- Spring Security를 이용한 인증/인가 처리 - RESTful API Endpoint 제공 - Spring Data JPA를 이용한 데이터 접근 처리
Redis(Cache)	- Cacheable 데이터(ex. 회원가입 시 이메일 인증 코드) 저장 - 빠른 데이터 접근을 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

● **데이터베이스 설계**

데이터베이스는 개념적 설계를 우선적으로 진행하고, 논리적 설계를 진행하였다.

개념적 설계



개념적 설계

주요 엔티티에는 User(사용자), Group(사용자가 생성하는 그룹), Project(그룹에 배포하는 프로젝트), 이외 권한 처리를 위한 GroupMembership과, ProjectAuthority가 있다.

각 구성 요소에 대한 세부 내용은 다음과 같다.

구성요소	설명	주요 필드
User	시스템 사용자	email, password, name
Group	시스템 사용자의 집합, 사용자가 생성	name
Project	그룹 내에서 배포되는 프로젝트	name, subdomain
GroupMembership	그룹 내 멤버 정보 및 멤버들의 역할	id(user), role
ProjectAuthority	프로젝트를 보거나 편집할 수 있는 권한	permission

각 구성 요소 간 관계는 다음과 같이 설정하였다.

테이블	User, Group, Project, 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)

논리적 설계

- 주요 엔티티 설계

User	
PK	id INT AUTO_INCREMENT
	username VARCHAR(50) NOT NULL UNIQUE
	email VARCHAR(100) NOT NULL UNIQUE
	password_hash VARCHAR(256) 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)

유저, 그룹, 프로젝트에 대한 테이블 스키마

- 권한 처리와 연관된 엔티티 설계

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)

그룹 멤버십, 프로젝트 권한에 대한 테이블 스키마

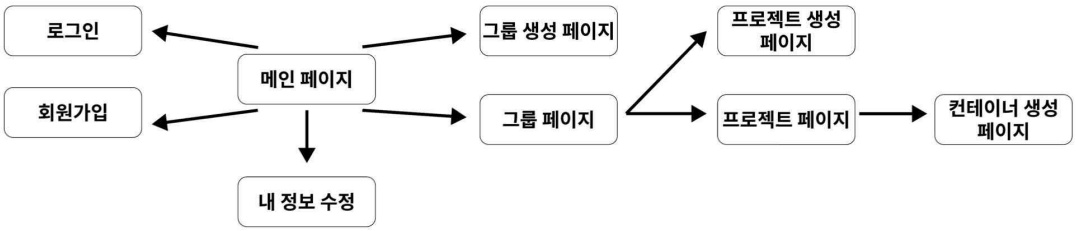
- 부가적 엔티티 설계

Invitation	
PK	id INT AUTO_INCREMENT
	sender_id INT
	receiver_id INT
	group_id INT
	status ENUM('pending', 'accepted', 'rejected') NOT N
	FOREIGN KEY (sender_id) REFERENCES User(id)
	FOREIGN KEY (receiver_id) REFERENCES User(id)
	FOREIGN KEY (group_id) REFERENCES Group(id)

사용자 그룹 초대와 관련된 테이블 스키마

3.3. 프론트엔드

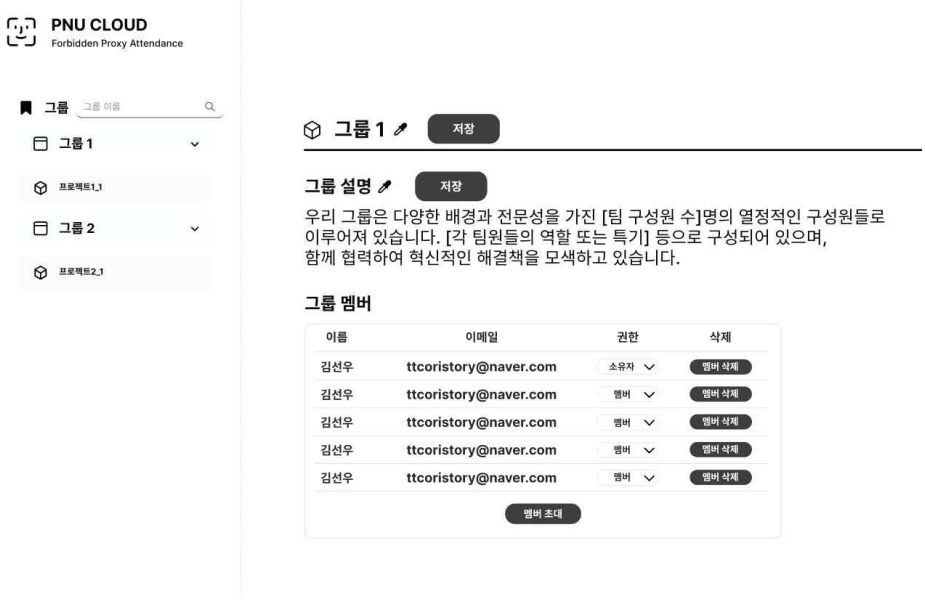
● 페이지 라우팅 구조



● 페이지 상세 정보

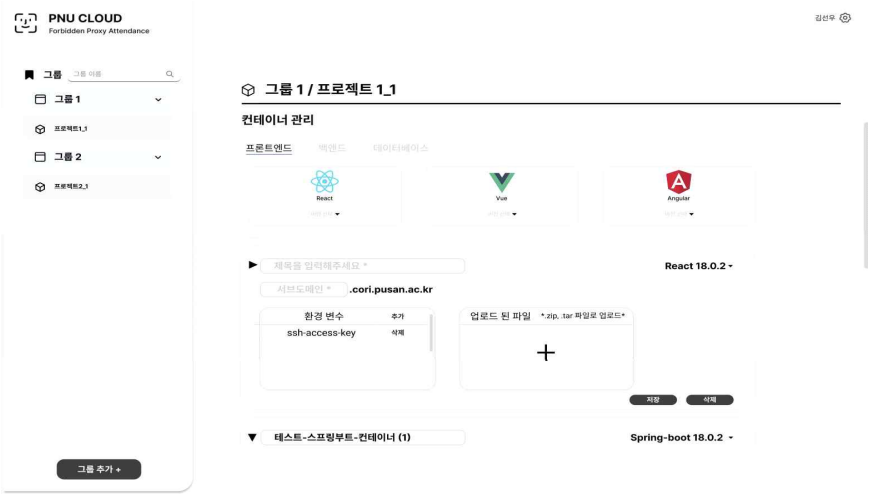
- 그룹 생성 / 관리 / 보기 페이지

사용자가 그룹을 생성하고, 생성된 그룹에 멤버를 초대 및 권한 관리를 하여 효율적으로 그룹을 관리할 수 있는 기능을 제공한다. 그룹 내에서는 프로젝트를 생성할 수 있으며, 생성된 프로젝트는 해당 그룹의 멤버를 자동으로 상속받아 관리의 편리성을 높인다. 또한, 멤버 초대 과정에서 최상의 사용자 경험(UX)을 제공하기 위해 모달 창을 통해 초대 기능을 구현하고, 이를 통해 직관적이고 원활한 초대 기능을 제공하고, 사용자의 편의성을 극대화하고자 한다.



- 프로젝트 생성 / 관리 / 보기 페이지

사용자가 프로젝트를 생성하고, 생성된 프로젝트에 멤버 권한 관리, 컨테이너 관리를하여 효율적으로 프로젝트를 관리할 수 있는 기능을 제공한다.. 그룹 내에서는 컨테이너를 생성할 수 있으며 생성된 컨테이너는 사용자가 미리 설정 해둔 url을 통해 접근하다..



● 사용자 입력 유효성 검사

- 이메일

이메일을 정규 표현식을 통해 이메일 형식인지 확인한다.

- 비밀번호

비밀번호는 영문 및 숫자 조합 8자 이항 15자 이하로 입력 받도록 확인한다.

- URL

URL은 예약된 문자 ':', '/', '?', '#', '[', ']', '@'를 포함하지 않으며 공백을 포함하지 않도록 확인한다.

- 프로젝트 파일

파일 형식을 .zip 이나 .tar로 제한하고, 추후 API와의 통신 연결을 하며 직접 테스르를 해 적절한 크기의 용량 제한도 한다.

4. 수행 현황

3.1. 인프라

● **서버 및 네트워크 구성**

해당 시스템은 클라우드 서비스로 다수의 서버가 필요하고 내부 네트워크가 필요한 아키텍쳐를 가지고 있다. 과제 수행 중에 이러한 자원을 물리적으로 구성하기 어려워 Hyper-V를 이용해 서버 및 네트워크를 구성하였다.

● **메시지큐 구축 및 연결**

내부 시스템 구성요소 간 통신은 메시지큐를 활용한 비동기 방식을 활용한다. 이를 위해 Redis를 활용하였고 메시지 손실을 방지하기 위해 AOF 방식을 활용해 디스크에 기록되어 의도치않은 다운이 발생해도 이전 내용을 복원할 수 있도록 구성하였다.

● **Gateway 기본 기능 구축**

HTTPS 인증서 적용 및 도메인 기반 리버스 프록시 연결을 통한 라우팅 기능에 대한 구현이 완료하였다. 아래 설정과 같이 lua 스크립트를 통해 할당된 내부 주소를 가져와 리버스 프록시로 연결하고 있다.

```
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;
}
```

● **SSL 인증서 발급 시스템 구축**

사용자가 도메인에 등록한 경우 /.well-known/acme-challenge/ 경로는 프로젝트 컨테인너가 아닌 SSL 발급 시스템으로 연결되도록 구성하여 HTTP-01 챌린지를 통해 인증서를 발급한다. 플랫폼이 보유중인 도메인은 와일드 카드 발급을 위해 DNS-01 챌린지를 수행한다.

● **배포 템플릿 작성**

백엔드, 콘솔 등 다양한 유형의 프로젝트를 지원하기 위해 컨테이너 이미지 빌드를 위한 템플릿을 작성하였다.

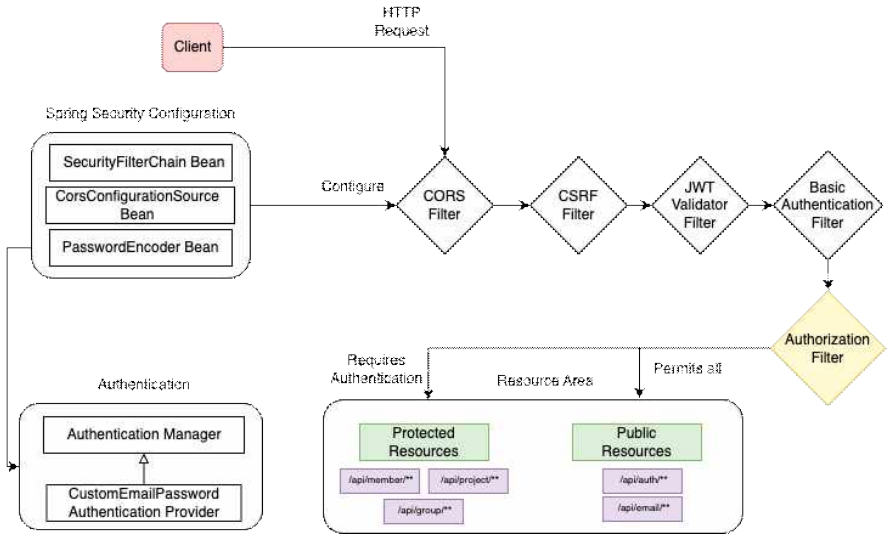
● **MVP 모델을 활용한 프로젝트 배포 시스템 검증**

프로젝트 배포가 가능한지 검증하기 위해 MVP 모델을 만들어 정적 파일과 node.js로 고정된 프로젝트 구성에 대해 검증하였다. 이를 통해 프로젝트 배포 및 도메인 연결 등 기본적인 기능에 대한 동작 여부를 확인하였다.

3.2. 백엔드

● **Spring Security 필터 체인 흐름 구현**

백엔드 핵심 구성요소 중 하나인 RESTful API에 세밀한 접근 제어와 권한관리를 위해 Spring Security 필터 체인을 아래 그림과 같이 구성하였다.



그룹 멤버십, 프로젝트 권한과 관련된 테이블 스키마

세부 인증/인가 동작과정은 다음과 같다.

[인증 과정]

1. CORS(Cross Origin Resource Sharing)처리

클라이언트 요청이 들어오면 CORS Filter가 작동하며, 설정된 CORS 정책에 따라 요청의 출처를 검사하고, 허용 여부를 검사한다.

2. JWT 토큰 검증

JWT Token Validator Filter가 요청 헤더에서 JWT 토큰을 추출하고 이의 디지털 서명 및 만료시간을 검증한다. 유효한 토큰이면 인증된 것으로 간주하고 다음 단계로 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 CustomEmailPasswordAuthenticationProvider를 호출하여 인증을 처리한다. 이 Provider는 회원관련 DAO인)ata Access ObjectD(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 접근권한을 확인한다.

3. 리소스 접근 허용/거부
인증된 사용자이고, 해당 리소스에 대한 접근 한권이 있으면 요청을 허용한다. 그렇지 않은 경우, 요청을 거부하고 적절한 오류 응답을 반환한다.

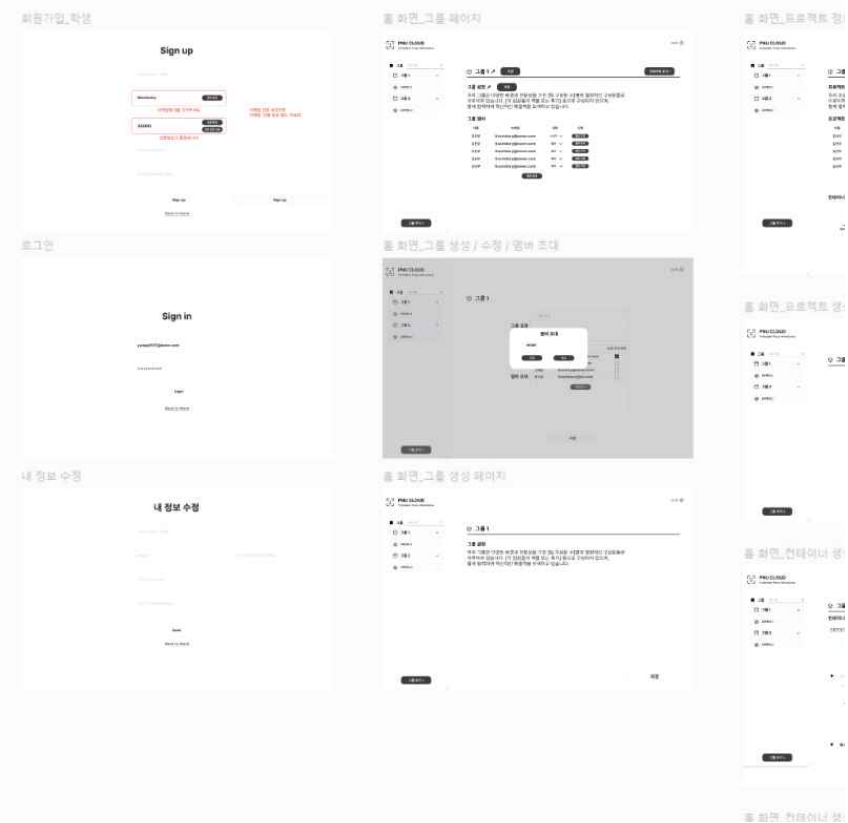
● **API 구현 현황**

REST 원칙에 맞게 현재까지 다음과 같은 API 엔드포인트 및 기능을 구현하였다.

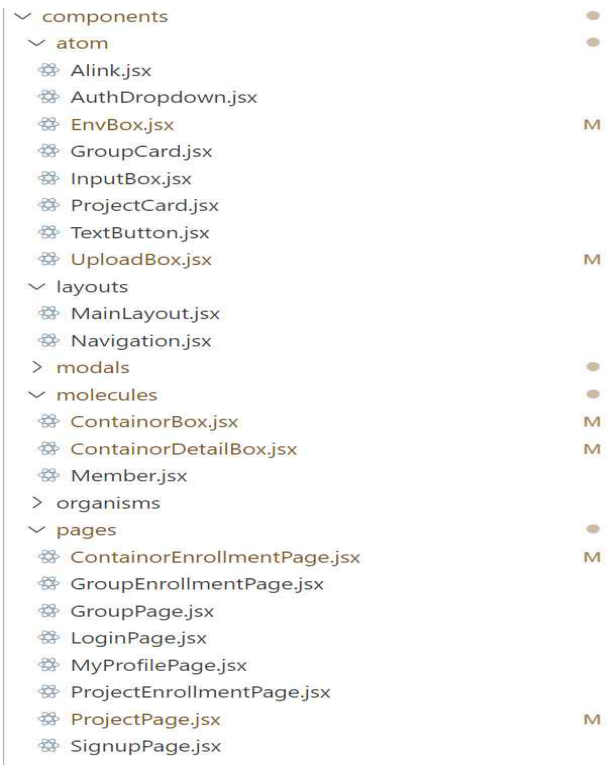
API 엔드포인트	기능 설명	HTTP Method	응답코드	권한
/api/auth/sign-up	회원가입	POST	201/400	Member
/api/auth/sign-in	로그인	POST	201/403	Member
/api/group	그룹 생성	POST	201/400	Admin
/api/group	그룹 조회	GET	200/400	Admin, Member
/api/group	그룹 수정	PATCH	200/400	Admin
/api/group	그룹 삭제	DELETE	203/400	Admin
/api/group/invite-member	그룹에 멤버 초대	POST	200/400	Admin
/api/email/check-valid	이메일 인증 확인	POST	200/400	Member
/api/email/code	이메일 인증 코드 발송	POST	200/400	Member

3.3. 프론트엔드

● 피그마를 통해 화면 UI 설계



● 컴포넌트 UI 및 상태 관리 구현



● 공통 컴포넌트 구현

- InputBox Component

- **Props**
  - type: <input /> 테그의 type을 결정
  - placeholder: <input /> 테그의 placeholder를 결정
  - isError: 컴포넌트가 에러 상태 인지 아닌지를 결정함에 따라 UI가 변경
  - moreStyle: 기존 고정된 Style에 추가 Style을 tailwind형식으로 전달 받음
  - onChange: <input /> 테그의 onChange 함수를 정의함
  - disabled: <input /> 테그의 disabled 속성을 결정
  - defaultValue: <input /> 테그의 defaultValue의 값을 결정

- TextButton Component

- **Props**
  - color: 'dark' | 'light' 두개의 값을 가지며 UI를 결정
  - children: <button>테그 내부의 요소를 받음
  - moreStyle: 기존 고정된 Style에 추가 Style을 tailwind형식으로 전달 받음
  - handleClick: <input /> 테그의 handleClick함수를 정의함
  - disabled: <button> 테그의 disabled 속성을 결정

5. 구성원별 진척도

5.1. 현재까지 진행 상황

이름	역할 분담
신예준	- Hyper-V 가상화를 활용한 환경 구성 - 인메모리 DB, 메시지큐 등 구성 요소 설정 - 도메인 기반 라우팅 게이트웨이, 인증서 발급 시스템 구현 - 프로젝트 빌드 시스템 구현 및 템플릿 작성
유승훈	- 데이터베이스 스키마 설계 및 ERD 작성 - JWT 기반 인증/인가 구현 - 사용자 및 그룹 도메인에 대한 비즈니스 로직 작성 및 API 개발
김선우	- 페이지의 라우팅 구조 설계 - 각 컴포넌트 정적/동적 데이터 상태 관리 - 팀원들과 회의를 통해 논의한 내용을 바탕으로, 최종 디자인안을 Figma로 설계

5.2. 향후 추진 계획

이름	역할 분담
신예준	- 컨테이너 오케스트레이션 적용 및 오토 스케일링 구현 - 게이트웨이 인가 시스템 및 프록시 서버 구성 - 네트워크 설정 고도화 및 방화벽 구성
유승훈	- 프로젝트 도메인에 대한 추가 백엔드 기능 및 API 개발 - 구현된 API에 대한 JUnit5 테스트 코드 작성 - 인프라 컴포넌트와 메시지큐를 통한 비동기 통신
김선우	- REST 통신을 통해 백엔드와 API 연동 - 사용자 입력 유효성 검사를 통한 런타임 에러 방지

6. 향후 추진 계획

업무	5				6				7				8				9				10			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
기획 및 설계																								
착수보고서 작성																								
MVP 개발																								
설계 고도화																								
백엔드 개발																								
인프라 개발																								
디버깅																								
시스템 연동																								
중간보고서 작성																								
개선 사항 분석																								
시스템 개선																								
인프라 구축																								
최종 테스트																								
시스템 배포																								
최종보고서 작성																								
발표 준비																								