2024 년 전기 졸업과제 중간보고서

MSA 기반의 클라우드 티켓팅 플랫폼 개발

팀 명: clove

202125292 조용진

201824410 강찬석

201824543 이동현

지도교수: 염근혁 (인)

목차

1. 과제의 목표

1) 과제 배경

마이크로서비스 아키텍처(MSA)는 시스템을 작은 독립적인 서비스들로 분리하고, 각 서비스는 모듈 또는 프로젝트 단위로 나누어 개발 및 관리를 진행하는 소프트웨어 아키텍처이다. 각 서비스는 느슨하게 결합(Loosely Coupling)되어 다른 서비스와의 의존성이 최소화된다는 특징이 있다.

이러한 특징을 바탕으로 마이크로서비스는 서비스 간 독립성이 보장된다. 서비스 독립성의 특성 상 일부 서비스에서 실패 지점이 발생하더라도 전체 시스템에 큰 영향을 미치지 않는다. 실패가 발생한 서비스에 대해서만 복구 작업을 수행하면 서비스 회복에도 빠르게 대처할 수 있다는 장점이 있다. 또한 서비스의 확장성과 유연성이 높아진다는 장점이 있다.

티켓 예매 시스템의 사용자는 공연을 주최하여 티켓을 판매하는 판매자, 티켓을 구매하는 구매자로 나뉜다. 현재 운영되고 있는 티켓 예매 시스템은 티켓 판매 회사와 시스템 운영 회사 간의 계약서에 기반하여 위탁 판매 형태를 이루며, 적게는 수일에서 많게는 수십일에 걸쳐 계약-판매-정산의 과정이 이행된다. 이러한, 복잡한 의뢰 과정은 티켓 판매자가 원하는 요구사항이 서비스에 신속하게 반영되기 어렵다는 단점이 있다.

따라서 본 과제에서는 티켓 판매자의 요구사항이 반영된 티켓 예매 시스템을 제공하기 위해 마이크로서비스 아키텍처가 반영된 티켓 예매 시스템 구축 방법을 제시한다. 본 과제는 1) 기 준비된 컨테이너 풀을 통해 티켓 시스템을 구축하는 컨테이너 기반 마이크로서비스 배포 지원, 2) 컨테이너 인프라 및 마이크로서비스의 모니터링을 통해 안정적인 컨테이너 운영 지원, 3) 티켓 판매자의 요구사항을 컨테이너 배포 시 반영할 수 있는 맞춤형 마이크로서비스 배포 시스템을 목표로 수행한다.

2) 과제 세부 목표

① 티켓 판매 마이크로서비스 제공

② 마이크로서비스 및 배포 인프라 모니터링

③ 배포 상태에 기반한 마이크로서비스 관리

2. 요구사항 및 제약사항 분석에 대한 수정사항

1) 기존 요구사항 및 수정사항

① 기능적 요구사항

표 1 은 시스템이 제공해야 할 기능들에 대한 요구사항을 나타낸다. 굵게 표시된 부분은 추가 및 수정된 요구사항이다.

표 1. 기능적 요구사항

기능		설명
시스템 접근 제어	회원가입	사용자 생성 기능이다. 회원가입시 판매자와 구매자로 구분되는 계정이 생성된다.
	로그인	사용자가 시스템 기능을 이용하기 위한 로그인 기능이다.
	로그아웃	사용자가 시스템 이용을 그만하기 위한 로그아웃 기능이다.
	회원탈퇴	생성한 사용자 정보를 영구 삭제하는 기능이다.
	인증 토큰 발급	**각 마이크로서비스 간에 로그인 정보를 유지할 수 있어야 한다.**
	인증 토큰 유효성 검사	**각 마이크로서비스에서 로그인 정보를 확인할 수 있어야 한다.**
마이크로서비스 배포 및 운영	서비스 배포	판매자는 공연에 관한 간단한 요구사항 입력을 통해 자동화된 티켓 예매 서비스를 배포할 수 있다.
	요구사항 업데이트	판매자는 배포한 공연의 서비스에 대해 요구사항이 변경되었을 경우 해당 내용을 업데이트할 수 있어야 한다.
	배포 템플릿 구성	마이크로서비스를 배포하기 위해 템플릿을 바탕으로 판매자의 요구사항 입력 정보를 추가하여 실제 배포되는 템플릿을 구성한다.
	서비스 삭제	판매자는 자신이 관리하는 서비스의 배포를 중단할 수 있어야 한다.
	서비스 모니터링	판매자는 자신이 배포한 서비스의 리소스 사용량을 조회할 수 있어야 한다.
마이크로서비스 관리	마이크로서비스 이미지 추가	컨테이너 풀에 새로운 마이크로서비스의 이미지를 등록한다.
	마이크로서비스 이미지 목록 조회	컨테이너 풀에 등록되어 있는 이미지 목록을 조회한다.
	마이크로서비스 이미지 명세 조회	**컨테이너 풀에 등록되어 있는 이미지의 마이크로서비스 명세를 조회한다.**
	마이크로서비스 이미지 삭제	컨테이너 풀에 등록되어 있는 마이크로서비스의 이미지를 삭제한다.
템플릿 관리	**템플릿 추가**	**공연 배포에 있어 필요한 이미지들을 묶어 기본적인 공연 배포가 가능한 템플릿을 추가한다.**
	템플릿 조회	**작성된 템플릿 목록을 조회한다.**
	템플릿 삭제	**작성된 템플릿을 삭제한다.**
마이크로서비스 상태 관리	마이크로서비스 리소스 이상 탐지	마이크로서비스의 리소스 사용량 모니터링 정보를 바탕으로 임계치를 넘는 리소스 사용량이 일어나는 컨테이너를 탐지하는 기능이다.
	마이크로서비스 리소스 스케일링	마이크로서비스의 리소스를 확장 또는 축소하여 안정적인 서비스가 제공될 수 있도록 지원하는 기능이다.
	마이크로서비스 재구조화	마이크로서비스의 서비스간 재결합 또는 재배치를 통해 안정적인 서비스가 제공될 수 있도록 지원하는 기능이다.
티켓팅 서비스 관리	**공연 정보 등록**	**판매자는 자신이 배포한 티켓팅 서비스의 정보를 추가할 수 있다,**
	공연 정보 수정	판매자는 자신이 배포한 티켓팅 서비스의 정보를 수정할 수 있다.
	예매자 목록 조회	판매자는 티켓을 예매한 구매자들의 정보를 확인할 수 있어야 한다.
	판매 내역 확인	**판매자는 티켓 판매에 대한 정보 및 통계를 확인할 수 있어야 한다.**
티켓팅 서비스	공연 목록 조회	구매자는 전체 공연의 목록을 조회할 수 있어야 한다.

이용	공연 정보 조회	구매자는 공연에 대한 상세 정보를 조회할 수 있어야 한다.
	잔여 좌석 조회	구매자는 잔여 좌석 정보를 조회할 수 있어야 한다.
	공연 예매	구매자는 잔여 좌석에 대해 예매를 할 수 있어야 한다.
	실시간 중복 예매 검사	구매자가 예매 정보를 입력하는 사이에 다른 사람이 같은 좌석에 대해 예매를 진행하는 것을 막기 위해 실시간 중복 예매 여부를 검사하는 기능이다.
	티켓 결제	구매자는 예매한 티켓을 확정하기 위해 외부 시스템에 결제를 요청한다.
	티켓 환불	구매자가 결제를 완료한 확정된 티켓을 환불 및 취소할 수 있어야 한다.
	구매 이력 조회	**구매자는 과거 자신이 구매한 이력에 대해 조회할 수 있어야 한다.**
	예매 티켓 조회	구매자는 자신이 예매한 티켓을 조회할 수 있다.

② 비기능적 요구사항

다음 표 2 는 시스템이 만족해야 할 비기능적 요구사항이다.

표 2. 비기능적 요구사항

요건	설명
가용성	시스템은 24 시간, 365 일 내내 사용 가능해야 함
안전성	사용자 정보가 외부로 노출되지 않아야 함
성능	시스템은 많은 사용자가 접속하더라도 성능 저하 없이 사용자에게 서비스를 제공하여야 함
	동시에 여러 명이 같은 티켓을 예매하는 경우가 없어야 함
보안성	등록된 사용자만이 시스템에 접근할 수 있어야 함
	등록된 사용자의 역할(권한)에 맞는 동작만 수행할 수 있어야 함

2) 기존 제약 사항

① 현실적 제약 사항

I. 티켓 예매 시 실제로 결제를 진행하는 데 어려움이 있다.

II. 모든 공연 장소에 대한 좌석 정보를 반영하는 데 어려움이 있다.

III. 테스트 시, 티켓을 예매하는 과정을 수행하는 대규모의 사람을 모으기 힘들다.

② 대안

I. 테스트 결제 시스템을 사용해, 결제 없이 예매 절차를 진행하여 테스트를 진행한다.

II. 모든 공연 장소가 아닌 일부 장소의 좌석 정보만을 선택하여 데이터 셋을 구성한다.

III. 실제 사람이 아닌 예매 봇 또는 매크로 작업을 수행하는 별도의 프로세스를 구성해 테스트를 진행한다.

3) 추가 제약 사항

① 현실적 제약 사항

I. 배포된 시스템의 장애 테스트 시 실제 시스템 장애가 발생하는 경우를 모두 재현하는 데 어려움이 있다.

② 대안

I. 재현할 수 있는 시스템 장애 시나리오 몇 가지를 구성하여 테스트를 진행한다.

3. 설계 상세화 및 변경 내역

1) 마이크로서비스 시스템 설계 및 명세

마이크로서비스 시스템을 설계하기 위해 각 마이크로서비스를 Boundary-Control-Entity 아키텍처 패턴으로 분류하여 클래스 다이어그램으로 나타낸 뒤[1], 해당 다이어그램을 바탕으로 마이크로서비스 명세를 작성하였으며[2], 클래스 다이어그램에서 메시지 큐를 이용하여 Control 에 해당하는 마이크로서비스 간 의존성을 최소화하였다.

① UML 클래스 다이어그램

- 시스템 배포 환경 관리

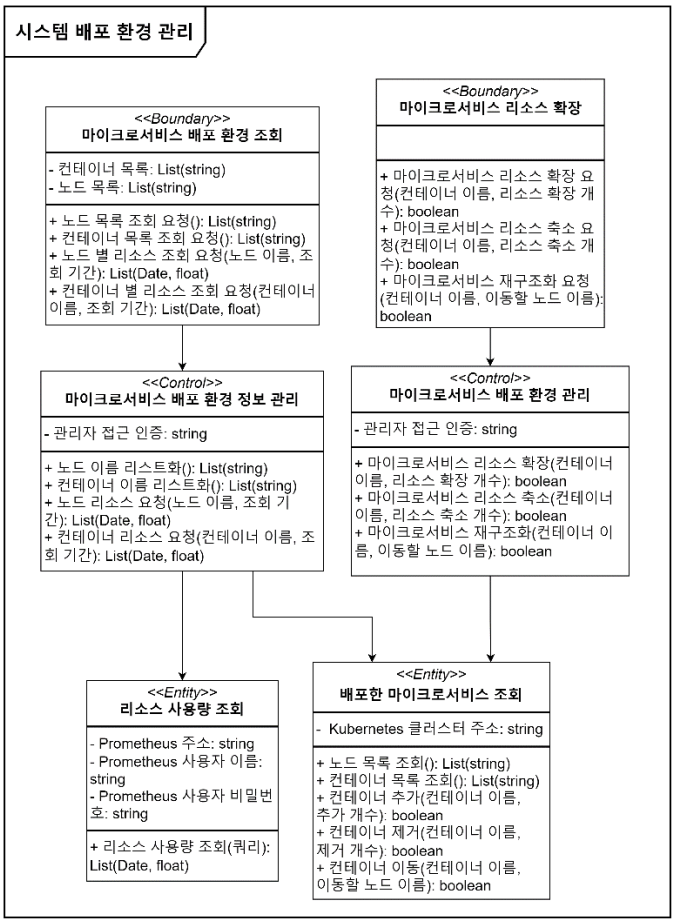

그림 1. 배포 환경 관리 클래스 다이어그램

그림 1 은 시스템 배포 환경 관리에 필요한 요소들을 클래스 다이어그램으로 나타낸 것이다.

마이크로서비스 배포 환경 조회 클래스는 관리자가 현재 쿠버네티스에 배포된 마이크로서비스에 대한 정보를 조회할 때 사용하는 인터페이스이다. 마이크로서비스 배포 환경 정보 관리에서 쿠버네티스 노드 목록과 배포된 컨테이너 목록 및 리소스를 가공하여 전달하는 로직을 수행한다. 배포한 마이크로서비스 조회 클래스를 통해 쿠버네티스에서 노드 목록 및 컨테이너 목록을 가져온다. 리소스 사용량 조회 클래스에서 프로메테우스에 저장된 리소스 정보들을 쿼리를 통해 가져온다.

마이크로서비스 리소스 확장 클래스는 관리자가 마이크로서비스 리소스를 관리할 때 사용하는 인터페이스이다. 마이크로서비스 배포 환경 관리 클래스에서 마이크로서비스 확장, 마이크로서비스 축소, 마이크로서비스 재구조화에 필요한 정보를 전달받아 수행한다. 배포한 마이크로서비스 조회 클래스를 통해 쿠버네티스에서 컨테이너 추가, 컨테이너 제거, 컨테이너를 다른 노드로 이동하는 동작을 수행한다.

- 시스템 배포 환경 모니터링

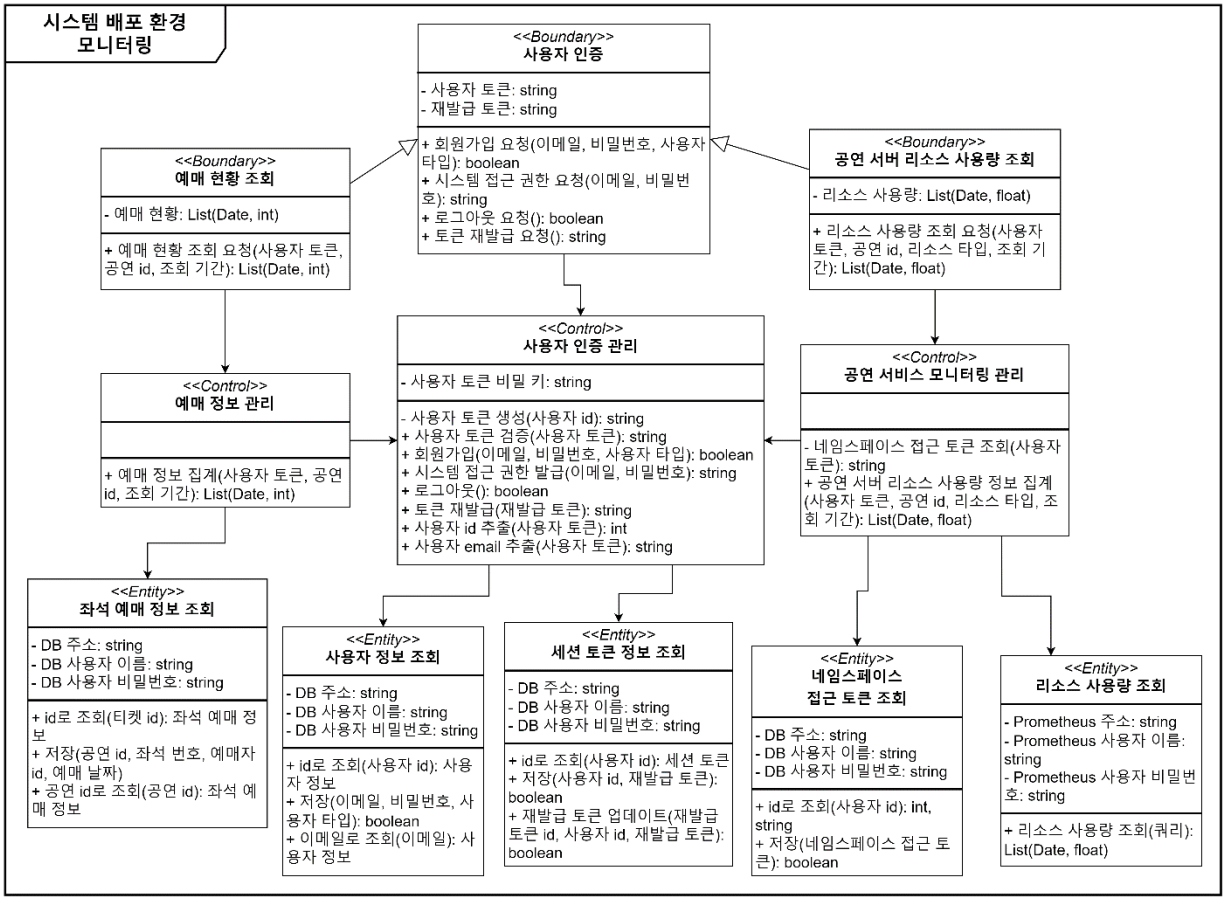

그림 2. 시스템 배포 환경 모니터링 클래스 다이어그램

그림 2 는 시스템 배포 환경 모니터링에 필요한 요소들을 나타낸 클래스 다이어그램이다.

사용자 인증 클래스는 사용자가 시스템에 접근하는 권한을 얻을 때 사용하는 인터페이스이다. 사용자 인증 관리 클래스에서는 사용자 정보 조회 클래스를 통해 사용자를 등록하거나 등록된 사용자의 정보를 조회하여 시스템 접근 권한을 발급한다. 사용자 정보는 사용자 id, 이메일, 비밀번호, 사용자 타입으로 구성된다. 세션 토큰 정보 조회 클래스는 사용자 토큰이 만료되어 재발급 요청 시 사용되는 재발급 토큰을 저장한다. 재발급 토큰은 사용자 별 재발급 토큰 관리를 위한 정보로 재발급 토큰 id, 사용자 id, 재발급 토큰 값으로 구성하였다.

예매 현황 조회 클래스는 현재 예매자 수, 일자 별 예매자 수 등 배포한 공연의 예매 통계를 조회할 때 사용하는 인터페이스이다. 예매 정보 관리 클래스에서 예매 정보 데이터를 사용하기 쉬운 구조로 가공하는 로직을 수행한다. 좌석 예매 정보 조회 클래스에서는 좌석 예매 정보를 저장하고 조회하는 역할을 수행한다. 좌석 예매 정보는 독립적인 좌석을 표현하기 위해 티켓 id, 공연 id, 좌석 번호, 예매자 id, 예매 날짜로 구성하였다.

공연 서버 리소스 사용량 조회 클래스는 배포한 공연 서버의 리소스 사용량 통계를 조회할 때 사용하는 인터페이스이다. 공연 서비스 모니터링 관리 클래스는 공연 서버 리소스 사용량 데이터를 사용하기 쉬운 구조로 가공하는 로직을 수행한다. 네임스페이스 접근 토큰 조회 클래스는 네임스페이스 단위로 사용자가 배포한 공연에 접근하기 위한 네임스페이스 접근 토큰을 DB 와의 상호작용을 통해 저장하고 조회하는 역할을 한다. 리소스 사용량 조회 클래스는 프로메테우스에 저장된 공연 서버의 리소스 사용량을 가져오는 역할을 수행한다.

- 마이크로서비스 인프라 관리

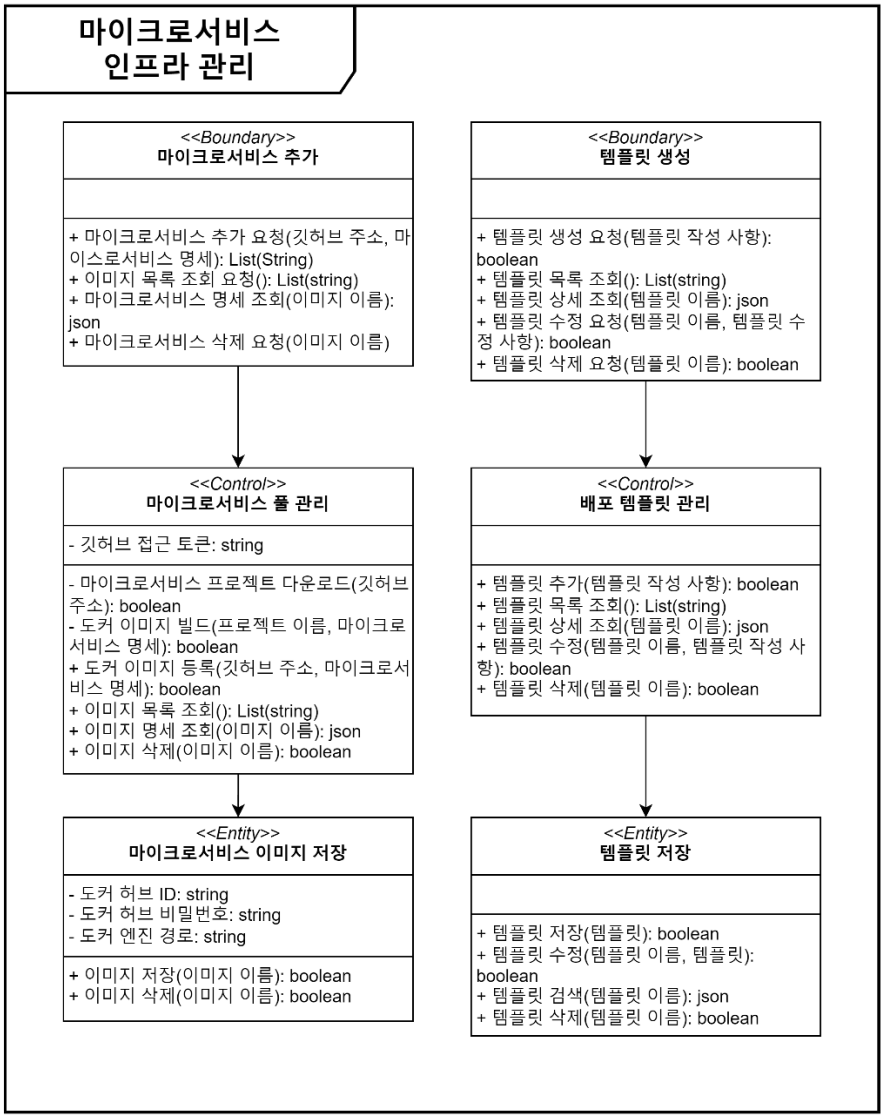

그림 3. 관리자 클래스 다이어그램

그림 3 은 관리자가 마이크로서비스 인프라 관리를 위해 사용하는 기능을 수행하는 클래스 다이어그램이다.

마이크로서비스 추가 클래스는 시스템 관리자가 개발한 마이크로서비스를 본 시스템에서 사용하기 위해 등록하는 인터페이스의 역할을 한다. 마이크로서비스 풀 관리 클래스는 새로운 마이크로서비스를 등록하기 위한 주요 로직들이 수행된다. 마이크로서비스 이미지 저장 클래스는 시스템에서 사용하는 도커 허브에 대한 정보를 가지고 있으며 도커 허브로의 이미지 저장 및 삭제의 역할을 수행한다.

템플릿 생성 클래스는 시스템 관리자가 운영 가능한 마이크로서비스 묶음 단위인 템플릿을 생성하기 위해 사용하는 인터페이스이다. 배포 템플릿 관리 클래스는 사용자 입력으로부터 템플릿 파일을 작성하는 등 주요 로직을 수행한다. 템플릿 저장 클래스는 시스템의 파일 시스템과 상호작용하며 템플릿 파일을 저장 및 관리하는 역할을 수행한다.

- 판매자 배포 서비스

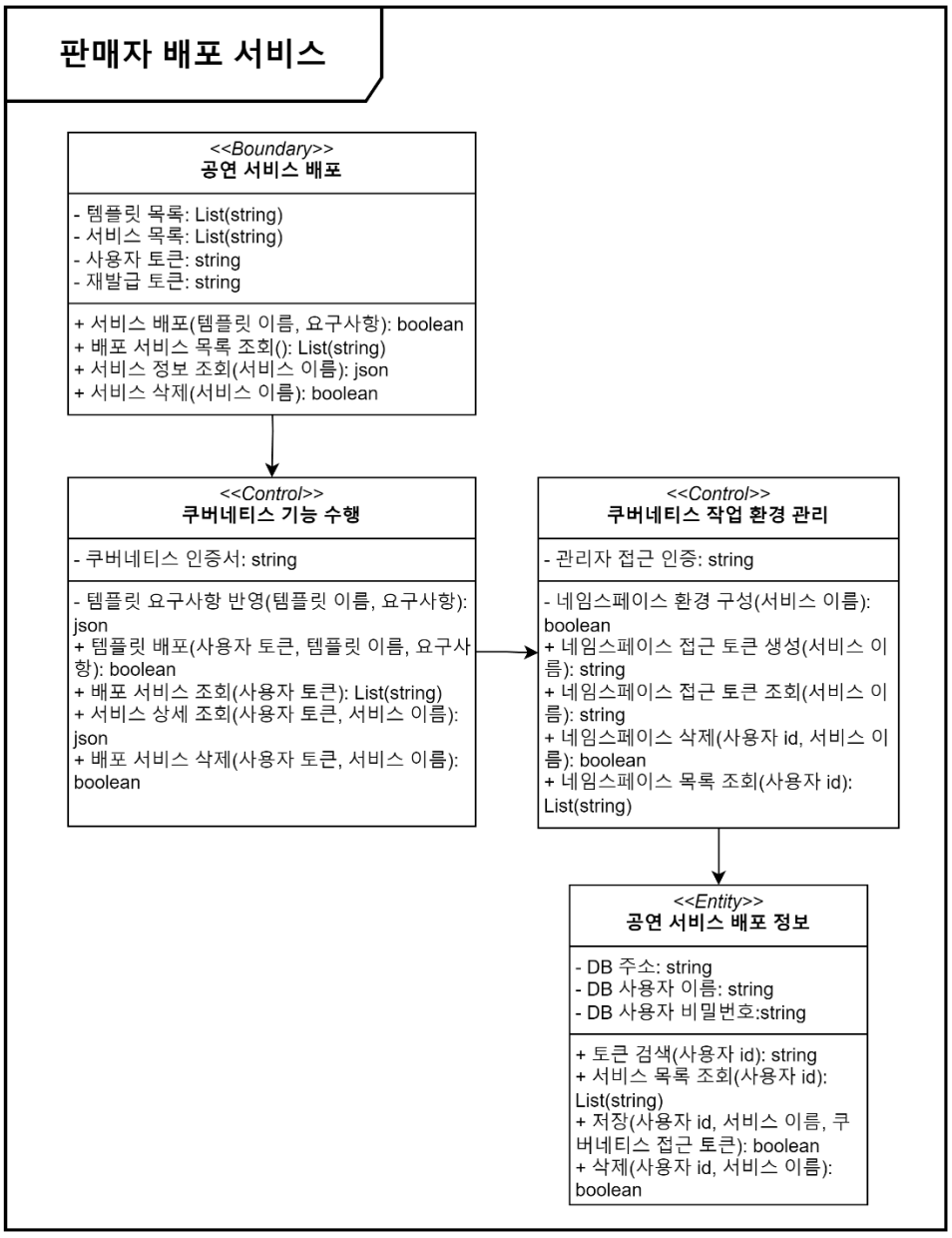

그림 4. 판매자 서비스 클래스 다이어그램

그림 4 는 판매자의 공연 서비스를 배포할 수 있도록 구성한 클래스 다이어그램이다.

공연 서비스 배포 클래스는 판매자가 공연 서비스를 배포하기 위해 사용하는 인터페이스이다. 쿠버네티스 기능 수행 클래스는 판매자가 자신이 배포하고자 하는, 또는 배포된 쿠버네티스 리소스를 제어하는 주요 로직을 수행한다. 쿠버네티스 작업 환경 관리 클래스는 쿠버네티스 관리자 접근 인증 파일을 가지고 있으며 판매자의 공연 서비스 운영을 위한 작업 공간인 네임스페이스를 생성하고 접근 권한이 담긴 토큰을 생성 및 조회하는 역할을 수행한다. 마지막으로 공연 서비스 배포 정보 클래스는 배포된 공연이나 해당 공연에 접근하기 위한 토큰 정보 등을 저장하고 있는 DB 와의 상호작용을 통해 데이터를 관리하는 클래스이다.

- 클라이언트 시스템

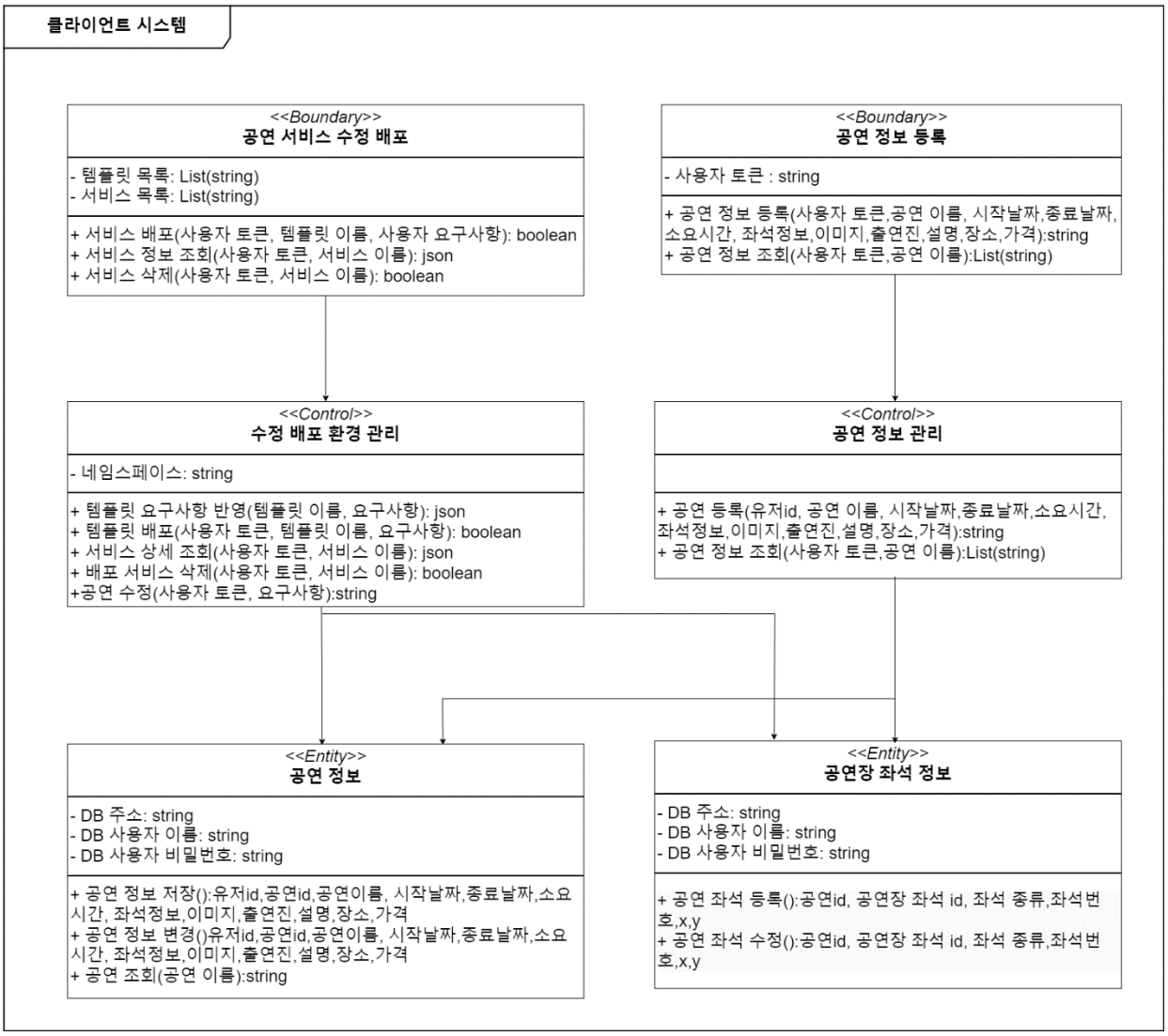

그림 5. 클라이언트 시스템 클래스 다이어그램 #1

그림 5 는 관리자가 배포된 공연 마이크로서비스의 정보를 관리하기 위한 클래스 다이어그램이다.

공연 서비스 수정 배포 클래스는 기 판매된 공연에 대해서 판매자가 임의로 수정할 때 사용되는 클래스이다. 수정할 공연을 선택하고 서비스 항목을 조회한다. 수정 배포 환경 관리 클래스는 사용자의 요구사항에 따라 템플릿을 반영하고 블루 그린 배포 방식에 따라 새로운 공연이 판매가 시작되는 순간 기존의 서비스는 삭제된다.

공연 정보 등록 클래스는 판매자가 마이크로서비스를 이용하여 공연을 판매하기 위해 사용되는 인터페이스의 역할을 한다. 공연 정보 관리 클래스는 판매되는 공연을 등록하기 위한 주요 로직들이 수행된다. 공연 정보 클래스는 시스템에서 관리되는 공연에 대한정보를 가지고 있으며 공연장 좌석 정보 클래스는 공연장 좌석에 대한 정보를 가지고 있다.

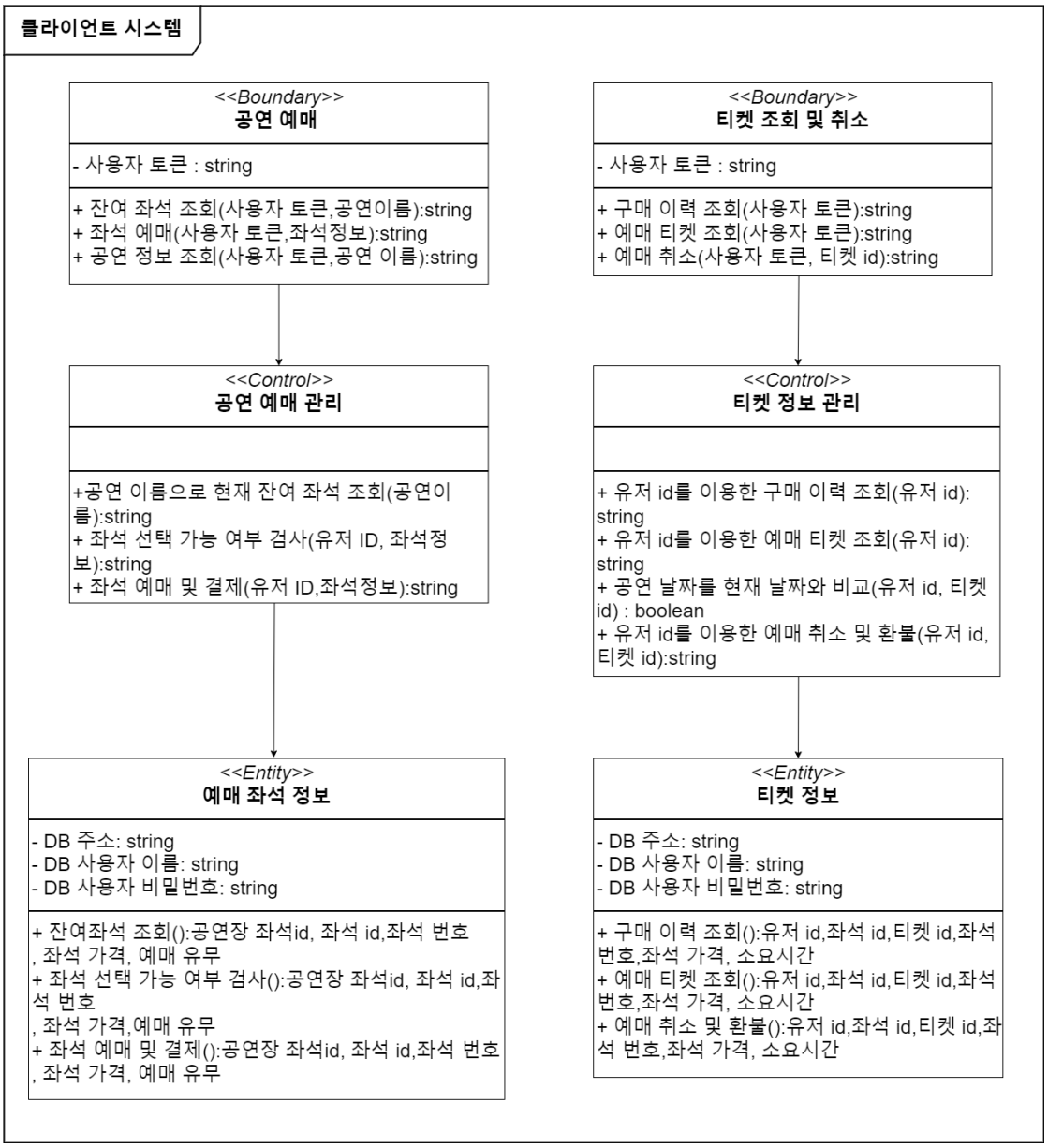

그림 6. 클라이언트 시스템 클래스 다이어그램 #2

그림 6 은 공연이 예매되는 과정에서 필요한 요소들을 클래스 다이어그램으로 나타낸 것이다.

공연 예매 클래스는 구매자가 특정 공연에 대해서 잔여 좌석을 조회하고 예매 및 결제를 할 때 사용되는 인터페이스이다. 공연 예매 관리 클래스에서 좌석중복선택 여부 메소드는 다수의 사용자가 하나의 좌석을 중복해서 선택하는 경우를 예방한다. 예매 좌석 정보 클래스에서는 공연장 좌석의 판매유무에 대한 정보를 저장하며 조회하는 역할을 수행한다.

티켓 조회 및 취소 클래스는 구매자가 마이크로서비스를 통해 구매한 티켓을 조회하고 특정티켓에 대해서 취소하는 인터페이스이다. 티켓 정보 관리 클래스에서 유저 id 를 이용하여 구매한 전체 티켓을 확인 가능하며, 현재 사용가능한 티켓 조회, 사용가능한 티켓에 한해서 취소 및 환불 로직을 처리할 수 있다.

② UML 시퀀스 다이어그램

- 회원가입

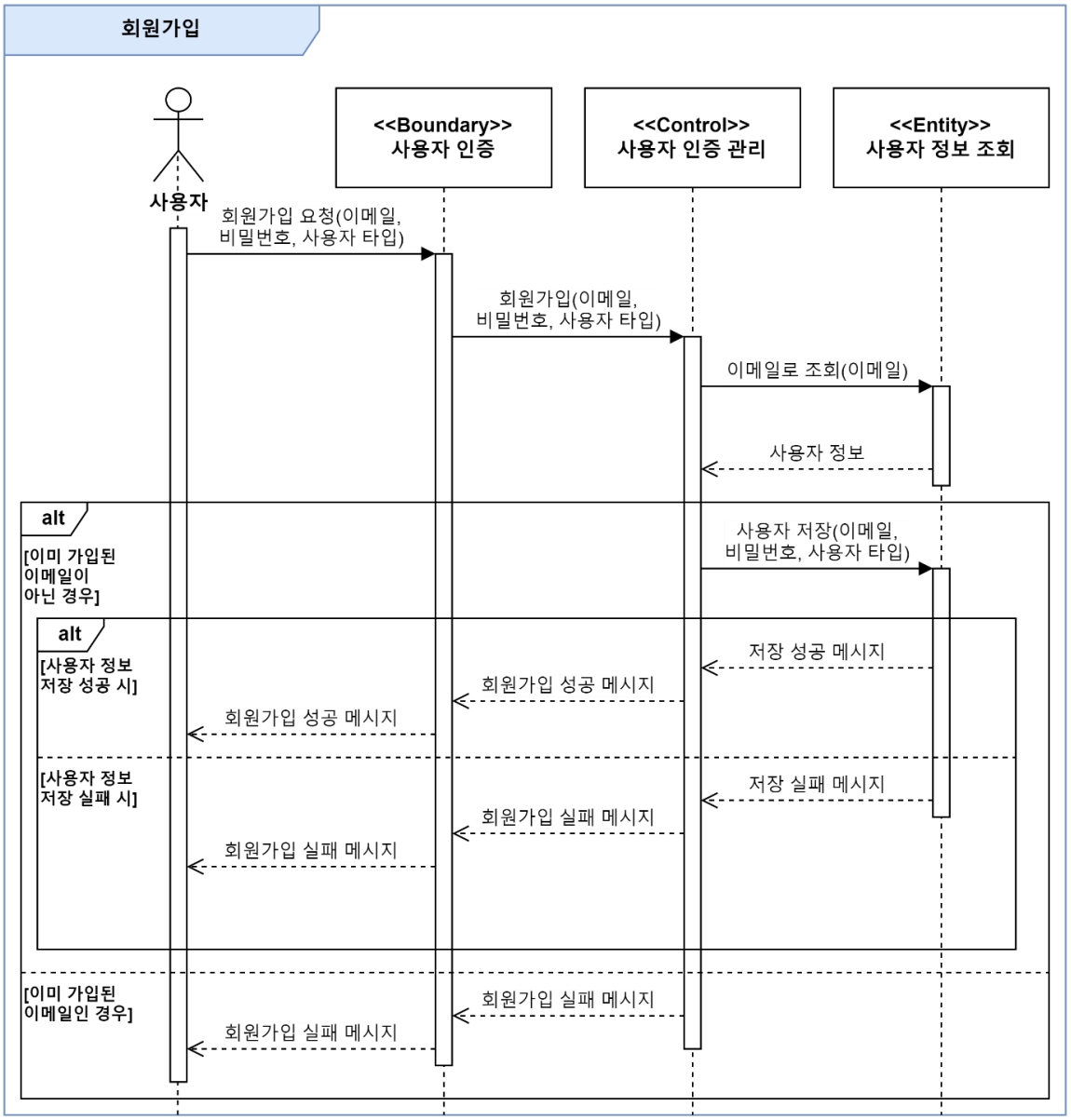

그림 7. 회원가입 시퀀스 다이어그램

그림 7 은 사용자가 회원가입을 진행하는 시퀀스 다이어그램이다.

사용자는 회원가입을 위해 이메일, 비밀번호, 사용자 타입을 입력하고 사용자 인증 클래스로 회원가입 요청을 보낸다. 사용자 인증 클래스에서는 사용자 인증 관리 클래스의 회원가입 메소드를 실행한다. 회원가입 메소드에서 사용자 정보 조회 클래스를 통해 이미 가입된 이메일인지 학인 한다. 가입된 이메일이라면 회원가입 실패 메시지를 반환한다. 가입된 이메일이 아니라면 입력 받은 회원가입 정보를 통해 사용자 정보 DB 에 사용자 정보를 저장한다. 사용자 정보 조회 클래스는 사용자 정보 저장 성공 시 저장 성공 메시지를, 사용자 정보 저장 실패 시 저장 실패 메시지를 반환한다. 사용자 인증 관리 클래스는 사용자 저장 메소드의 응답에 따라 회원가입 성공 메시지 또는 회원가입 실패 메시지를 반환한다.

- 로그인

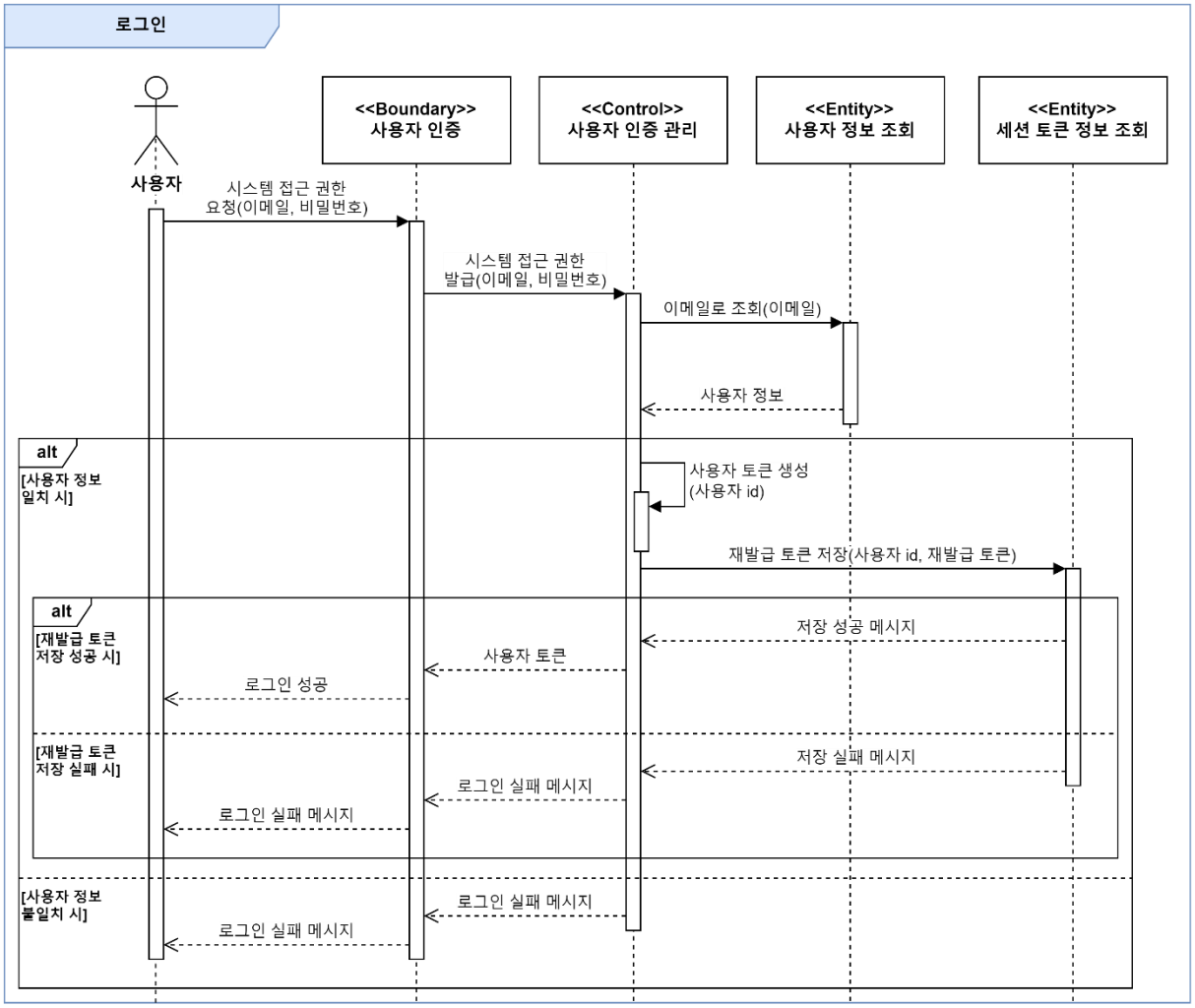

그림 8. 로그인 시퀀스 다이어그램

그림 8 은 사용자가 로그인을 진행하는 시퀀스 다이어그램이다.

사용자는 로그인을 위해 이메일, 비밀번호를 입력하고 사용자 인증 클래스의 시스템 접근 권한 요청을 실행한다. 사용자 인증 클래스는 사용자 인증 관리 클래스의 시스템 접근 권한 발급을 실행한다. 시스템 접근 권한 발급 메소드에서 전달받은 이메일로 사용자 정보를 조회한다. 이메일과 비밀번호가 사용자 정보 DB 와 일치하지 않는다면 로그인 실패 응답을 반환한다. 이메일과 비밀번호가 사용자 정보 DB 와 일치하면 사용자 토큰을 생성한다. 사용자 토큰 만료 시 재발급 로직을 수행하기 위해 재발급 토큰을 세션 토큰 정보 DB 에 저장한다. 세션 토큰 정보 조회 클래스는 재발급 토큰 저장 성공 시 저장 성공 메시지를, 재발급 토큰 저장 실패 시 저장 실패 메시지를 반환한다. 사용자 인증 관리 클래스는 재발급 토큰 저장 메소드의 응답으로 저장 성공 메시지를 반환 받으면 사용자 토큰을 반환하고 저장 실패 메시지를 반환 받으면 로그인 실패 메시지를 반환한다.

- 마이크로서비스 목록 조회

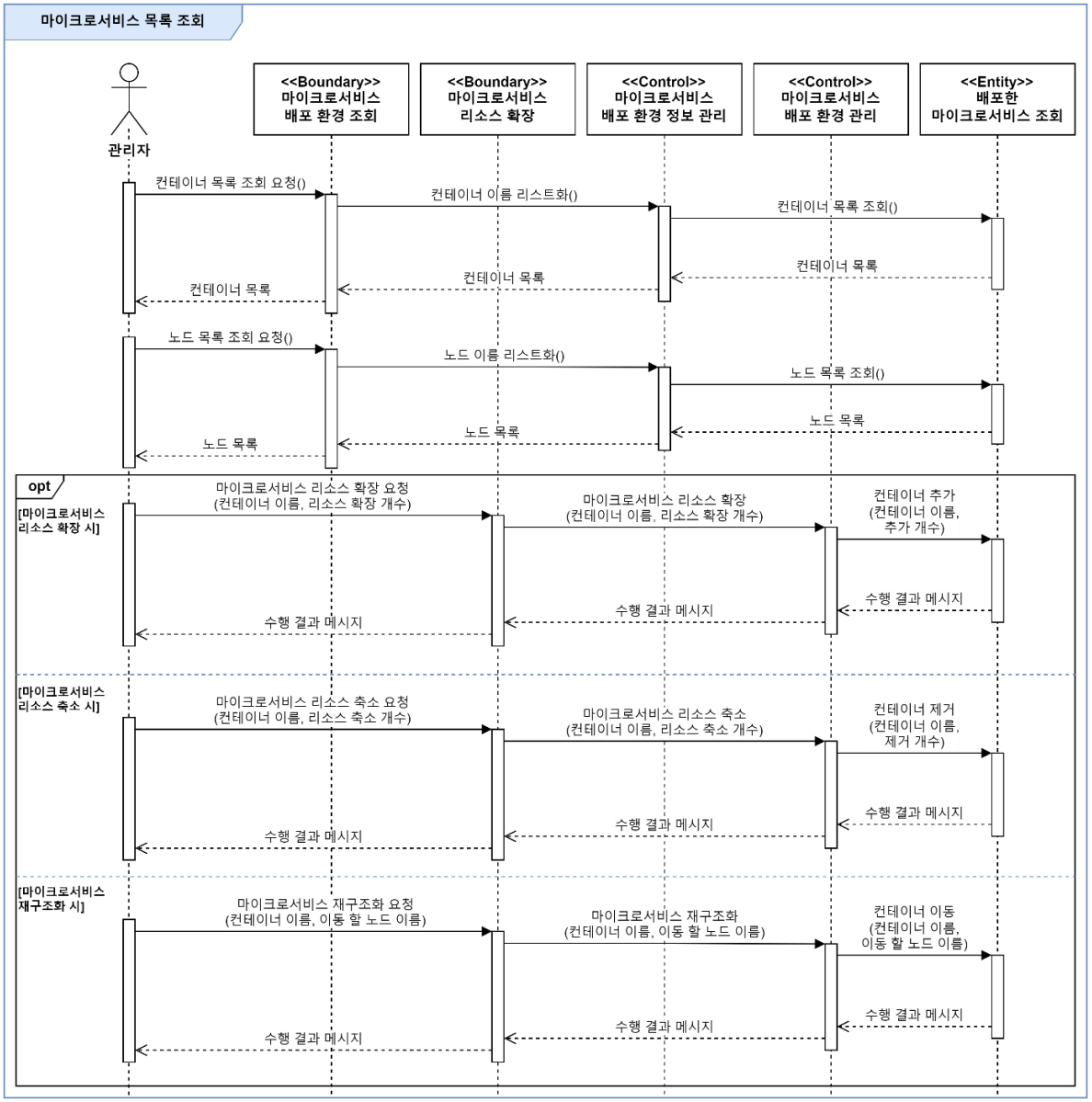

그림 9. 마이크로서비스 목록 조회 시퀀스 다이어그램

그림 9 는 마이크로서비스 목록 조회 기능을 시퀀스 다이어그램으로 나타낸 것이다.

관리자는 컨테이너 목록을 조회하기 위해 마이크로서비스 배포 환경 조회 클래스의 컨테이너 목록 조회 요청을 실행한다. 마이크로서비스 배포 환경 관리 클래스의 컨테이너 목록 조회를 통해 컨테이너 목록을 반환한다. 노드 목록 조회는 노드 목록 조회 요청을 호출하며 컨테이너 목록 조회와 같은 흐름으로 진행된다.

마이크로서비스 리소스 확장 시 관리자는 확장할 컨테이너 이름과 리소스를 확장할 개수를 입력한 뒤 마이크로서비스 리소스 확장 클래스의 마이크로서비스 리소스 확장 요청을 실행한다. 배포한 마이크로서비스 조회 클래스에서 컨테이너 추가 메소드까지 실행하게 되면 컨테이너 추가가 성공적으로 수행되었는지에 대한 수행 결과 메시지가 반환된다. 마이크로서비스 리소스 축소 및 마이크로서비스 재구조화의 경우 각각 리소스 축소 개수, 이동할 노드 이름을 입력하여 요청하며 마이크로서비스 리소스 확장과 동일한 흐름으로 진행된다.

- 공연 서버 모니터링

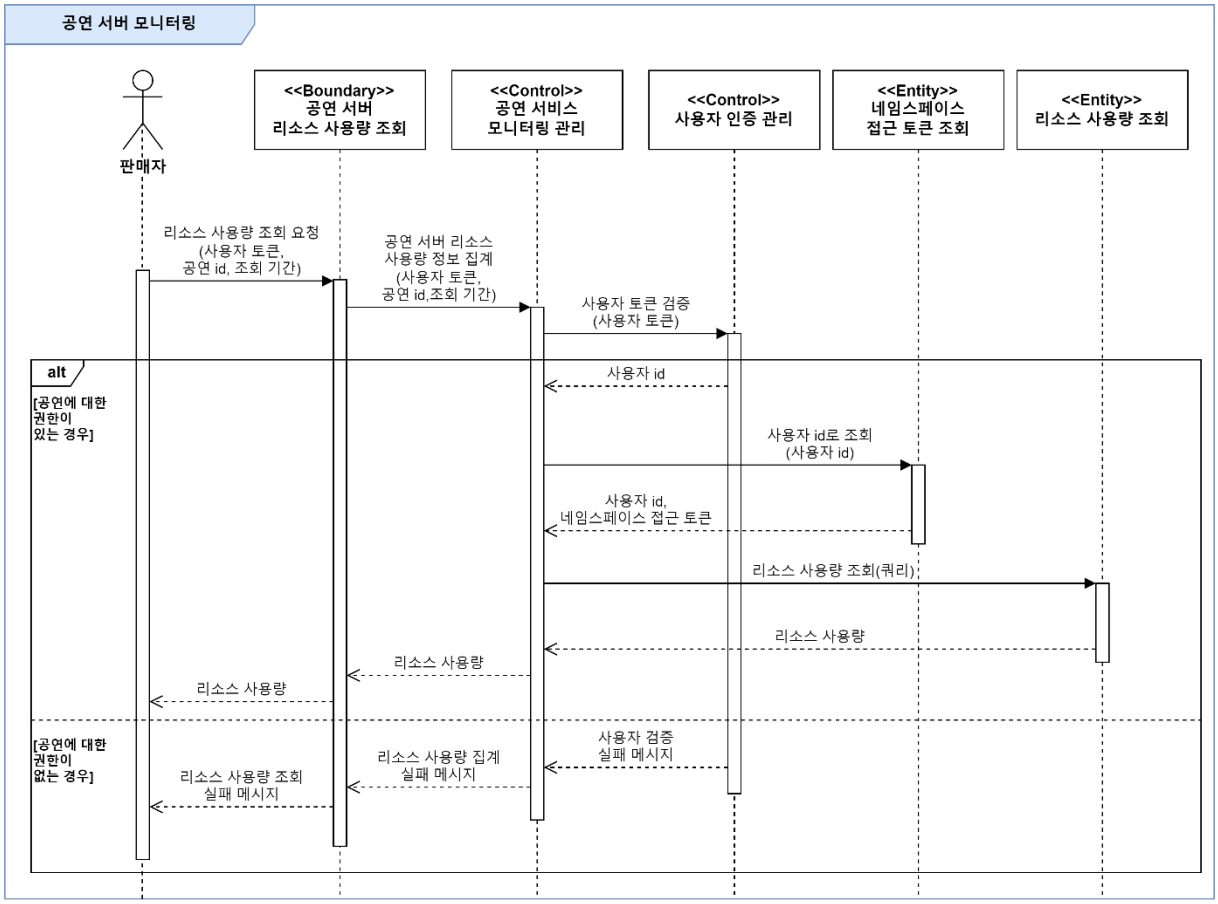

그림 10. 공연 서버 모니터링 시퀀스 다이어그램

그림 10 은 판매자가 배포된 공연 마이크로서비스에 대한 모니터링을 진행하는 시퀀스 다이어그램이다.

판매자는 배포한 공연 서버의 리소스 사용량을 조회하기 위해 조회할 공연과 조회 기간을 입력한 뒤 리소스 사용량 조회 요청을 실행한다. 공연 서버 리소스 사용량 조회 클래스에서는 공연 서비스 모니터링 관리 클래스의 공연 서버 리소스 사용량 정보 집계 메소드를 실행한다. 사용자 인증 관리 클래스는 사용자 토큰 검증 메소드의 파라미터를 통해 전달받은 사용자 토큰을 검증하여 공연에 대한 권한이 있는 경우 사용자 id 를 반환하며 권한이 없는 경우 사용자 검증 실패 메시지를 반환한다. 공연 서비스 모니터링 관리 클래스는 사용자 검증 실패 메시지를 반환 받으면 리소스 사용량 집계 실패 메시지를 반환한다. 사용자 id 를 반환 받으면 사용자 id 를 통해 네임스페이스 접근 토큰 DB 에서 네임스페이스 접근 토큰을 가져온다. 네임스페이스 토큰을 통해 프로메테우스에 리소스 사용량 조회 쿼리를 실행하여 공연 서버의 리소스 사용량을 가져온다. 공연 서비스 모니터링 관리 클래스는 공연 서버 리소스 사용량 조회 클래스에서 판매자에게 리소스 사용량을 시각화 하여 전달할 수 있도록 리소스 사용량 데이터를 가공하여 반환한다.

- 예매 현황 모니터링

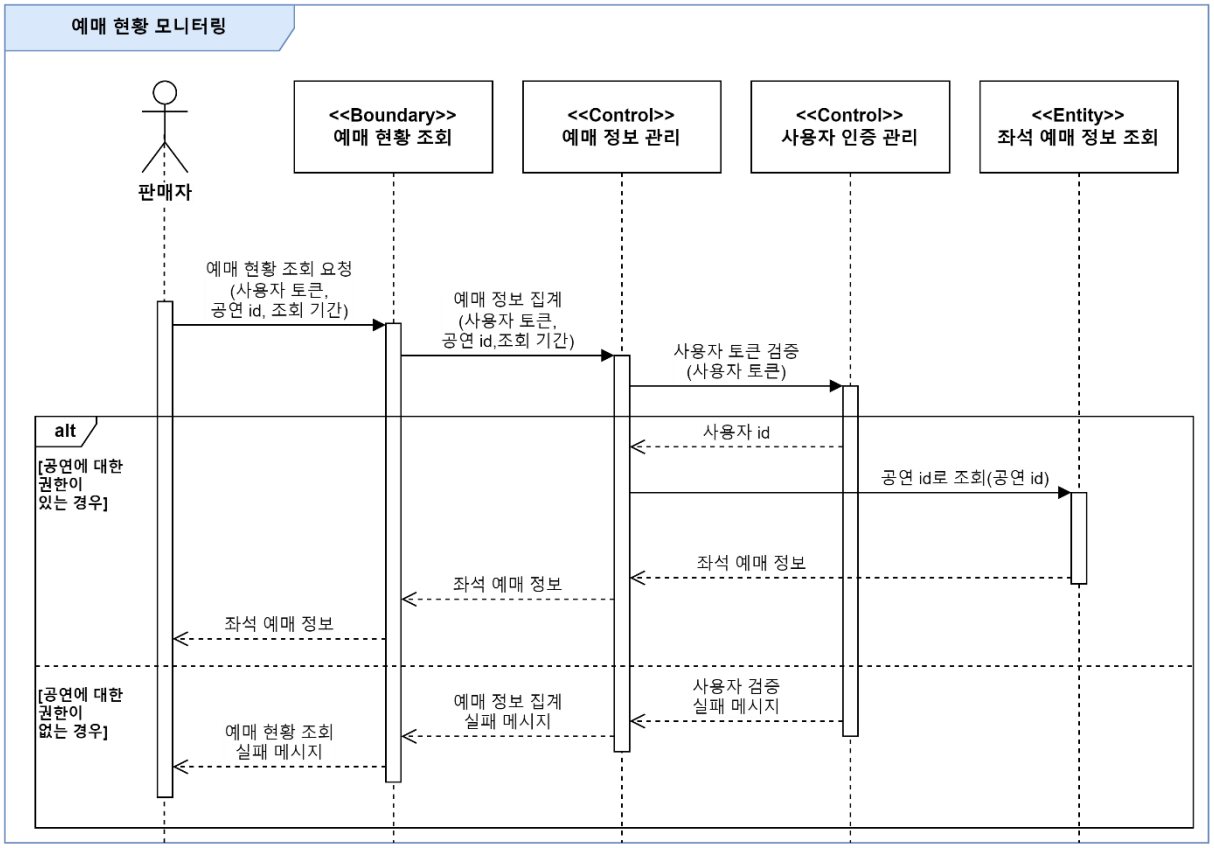

그림 11. 예매 현황 모니터링 시퀀스 다이어그램

그림 11 은 판매자가 자신이 배포한 공연에 대해 예매 현황을 조회하는 시퀀스 다이어그램이다.

판매자는 배포한 공연의 예매 현황을 조회하기 위해 조회할 공연과 조회 기간을 입력한 뒤 예매 현황 조회 요청을 실행한다. 예매 현황 조회 클래스에서는 예매 정보 관리 클래스의 예매 정보 집계 메소드를 실행한다. 사용자 인증 관리 클래스는 사용자 토큰 검증 메소드의 파라미터를 통해 전달받은 사용자 토큰을 검증하여 공연에 대한 권한이 있는 경우 사용자 id 를 반환하며 권한이 없는 경우 사용자 검증 실패 메시지를 반환한다. 예매 정보 관리 클래스는 사용자 검증 실패 메시지를 반환 받으면 예매 정보 집계 실패 메시지를 반환한다. 사용자 id 를 반환 받으면 예매 정보 집계 메소드의 파라미터를 통해 전달받은 공연 id 로 좌석 예매 정보를 가져온다. 예매 정보 관리 클래스는 예매 현황 조회 클래스에서 판매자에게 예매 현황을 시각화 하여 전달할 수 있도록 좌석 예매 정보 데이터를 가공하여 반환한다.

- 마이크로서비스 추가

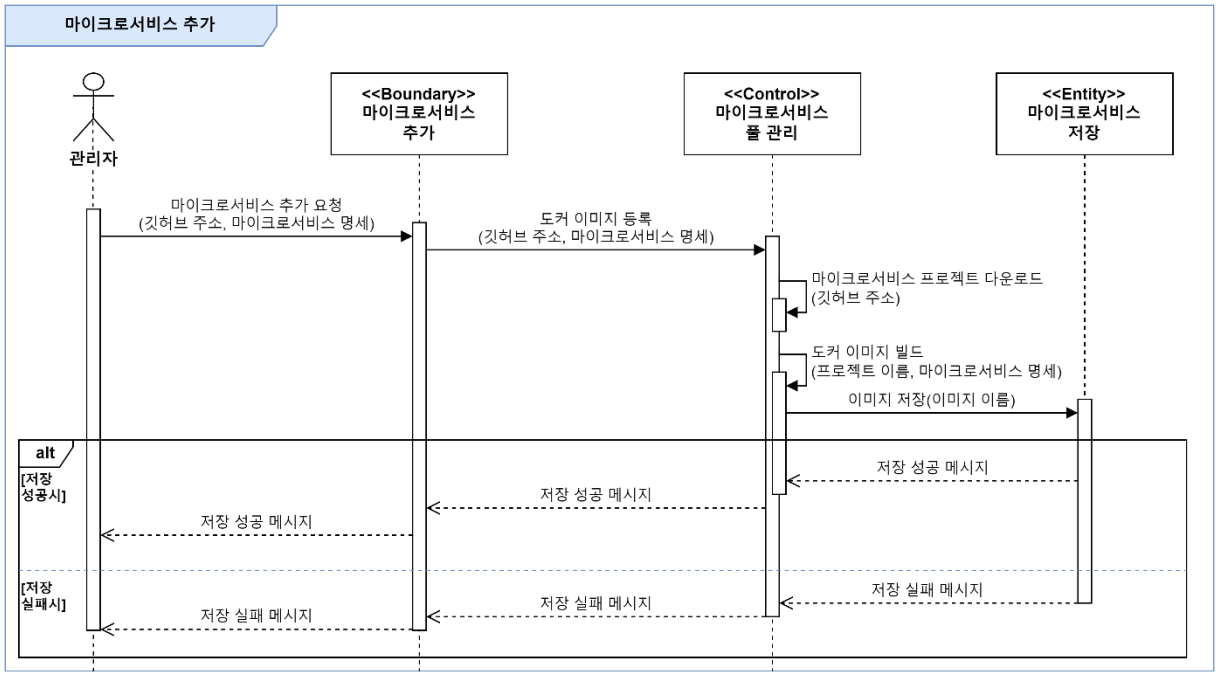

그림 12. 마이크로서비스 추가 시퀀스 다이어그램

그림 12 는 관리자가 시스템에 마이크로서비스를 추가하는 시퀀스 다이어그램이다.

마이크로서비스 추가를 위해서, 관리자는 자신이 개발한 프로젝트가 업로드 된 깃허브 주소를 입력하고, 클래스 이름, BCE 패턴, 연결 가능 서비스, 메서드, 약결합 URL 으로 구성된 마이크로서비스 명세를 입력한다. 사용자는 입력된 내용을 바탕으로 마이크로서비스 추가 클래스에 마이크로서비스 추가 요청을 호출한다. 마이크로서비스 추가 요청은 마이크로서비스 풀 관리 클래스에게 도커 이미지 등록을 요청하고, 도커 이미지 등록 수행을 위해 마이크로서비스 프로젝트 다운로드와 도커 이미지 빌드를 수행한다. 이후, 도커 이미지 등록은 마이크로서비스 저장 클래스에 이미지 저장을 요청하고, 마이크로서비스 이미지 저장이 성공한다면 성공 메시지를 반환하고, 실패한다면 실패 메시지를 반환하여 사용자에게 표시한다.

- 마이크로서비스 조회

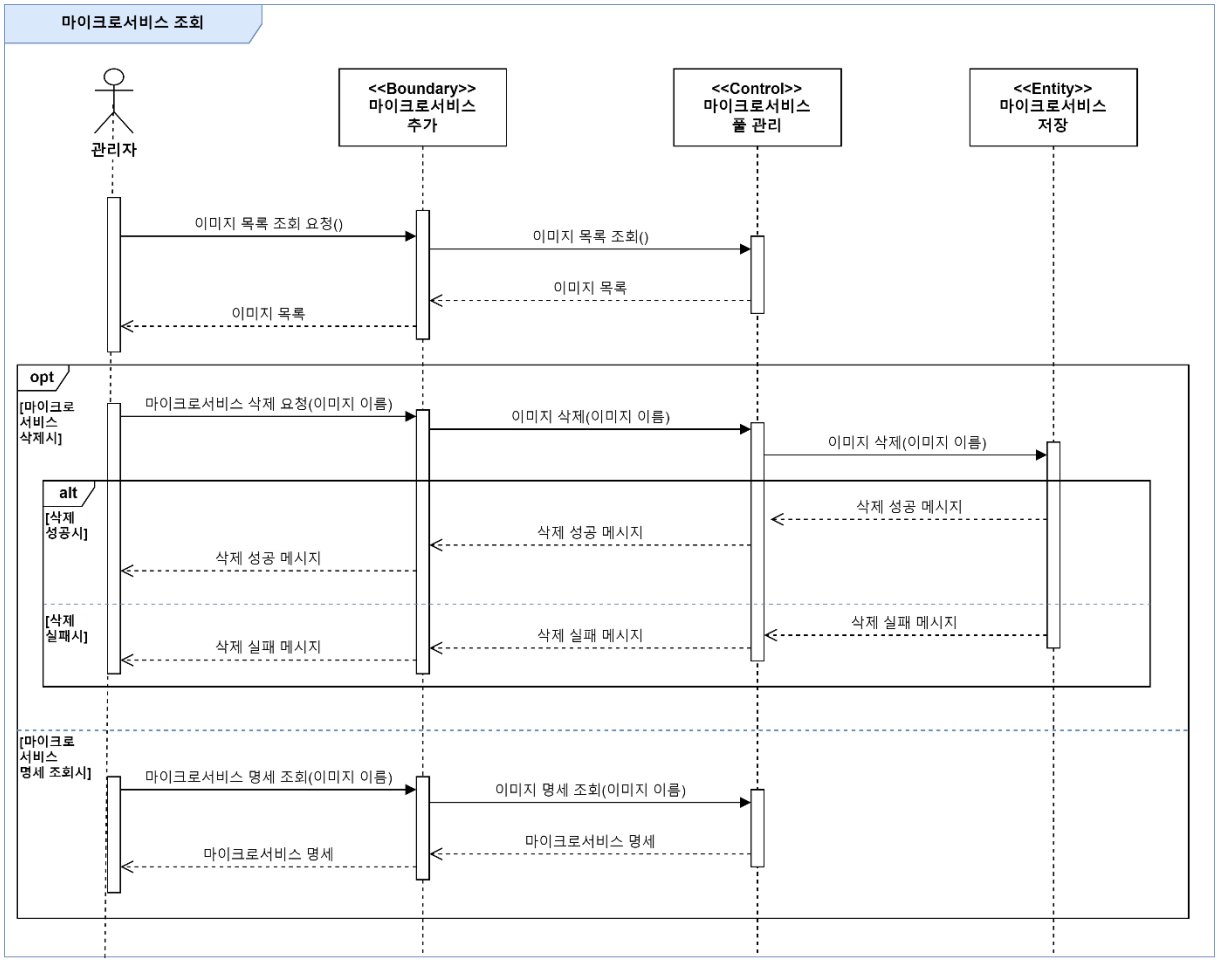

그림 13. 마이크로서비스 조회 시퀀스 다이어그램

그림 13 은 관리자가 시스템에 등록된 마이크로서비스를 조회하는 시퀀스 다이어그램이다.

마이크로서비스 조회를 위해서, 관리자는 마이크로서비스 추가 클래스의 이미지 목록 조회 요청을 호출한다. 이는 마이크로서비스 풀 관리 클래스의 이미지 목록 조회를 요청한다. 시스템 내에 등록된 이미지 목록을 반환하고 사용자에게 보여준다. 마이크로서비스를 삭제하고자 할 때, 관리자는 마이크로서비스 추가 클래스의 마이크로서비스 삭제 요청을 호출한다. 이는 마이크로서비스 풀 관리 클래스의 이미지 삭제를 요청하고 마이크로서비스 저장 클래스를 통해 등록된 이미지를 삭제한다. 이미지 삭제는 대상 이미지 삭제에 성공하면 성공 메시지를 반환하고, 실패한다면 실패 메시지를 반환하여 사용자에게 표시한다. 마이크로서비스의 명세를 조회하고자 할 때, 관리자는 마이크로서비스 추가 클래스의 마이크로서비스 명세 조회를 호출한다. 이는 마이크로서비스 풀 관리 클래스의 이미지 명세 조회를 요청하게 되고 입력한 이미지의 마이크로서비스 명세를 반환 받아 사용자에게 표시한다.

- 템플릿 추가

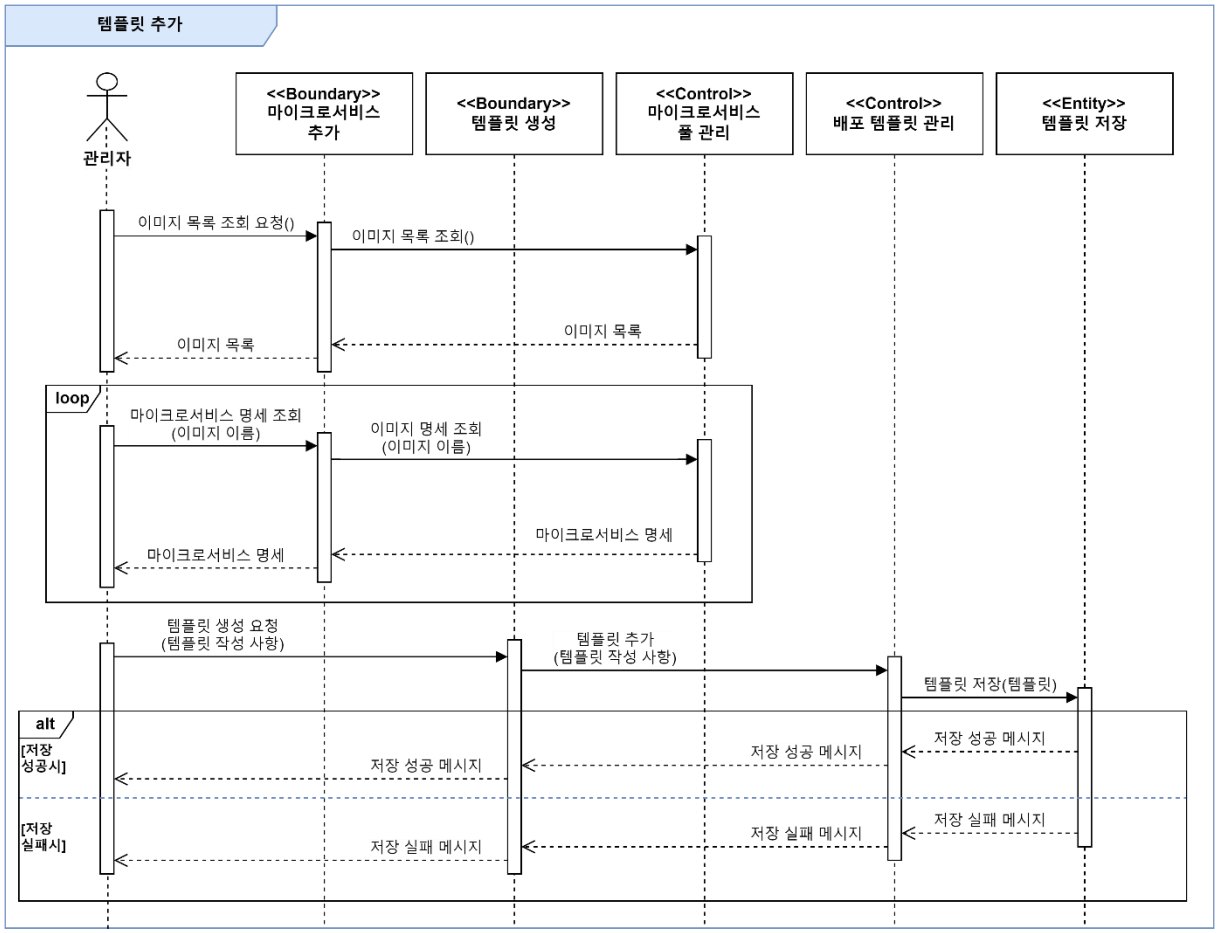

그림 14. 템플릿 추가 시퀀스 다이어그램

그림 14 는 관리자가 시스템에 템플릿을 추가하는 시퀀스 다이어그램이다.

템플릿 추가를 위해서, 관리자는 마이크로서비스 추가 클래스에 이미지 목록 조회 요청을 호출한다. 이는 마이크로서비스 풀 관리 클래스의 이미지 목록 조회를 요청하고, 시스템 내에 등록된 이미지 목록을 반환해주고 사용자 화면에 표시한다. 관리자가 템플릿에 이미지를 추가하고자 할 때, 템플릿에 추가할 이미지의 명세 정보를 확인해야 한다. 관리자는 마이크로서비스 추가 클래스의 마이크로서비스 명세 조회를 호출한다. 이는 마이크로서비스 풀 관리 클래스의 이미지 명세 조회를 호출하고, 해당 이미지의 마이크로서비스 명세를 반환해주고 사용자에게 보여준다. 관리자가 템플릿에 추가할 이미지가 더 없을 때까지 이미지 명세 정보를 확인하는 과정을 반복한다. 추가할 이미지 선택이 끝나면, 관리자는 템플릿 작성 요구사항을 입력하고, 템플릿 생성 클래스의 템플릿 생성 요청을 호출한다. 이는 배포 템플릿 관리 클래스의 템플릿 추가를 요청하게 되고 템플릿 저장 클래스를 통해 실제 템플릿을 저장한다. 템플릿 저장은 템플릿이 성공적으로 파일시스템에 저장이 되면 성공 메시지를 반환하고, 실패하면 실패 메시지를 반환하여 사용자에게 표시한다.

- 템플릿 조회

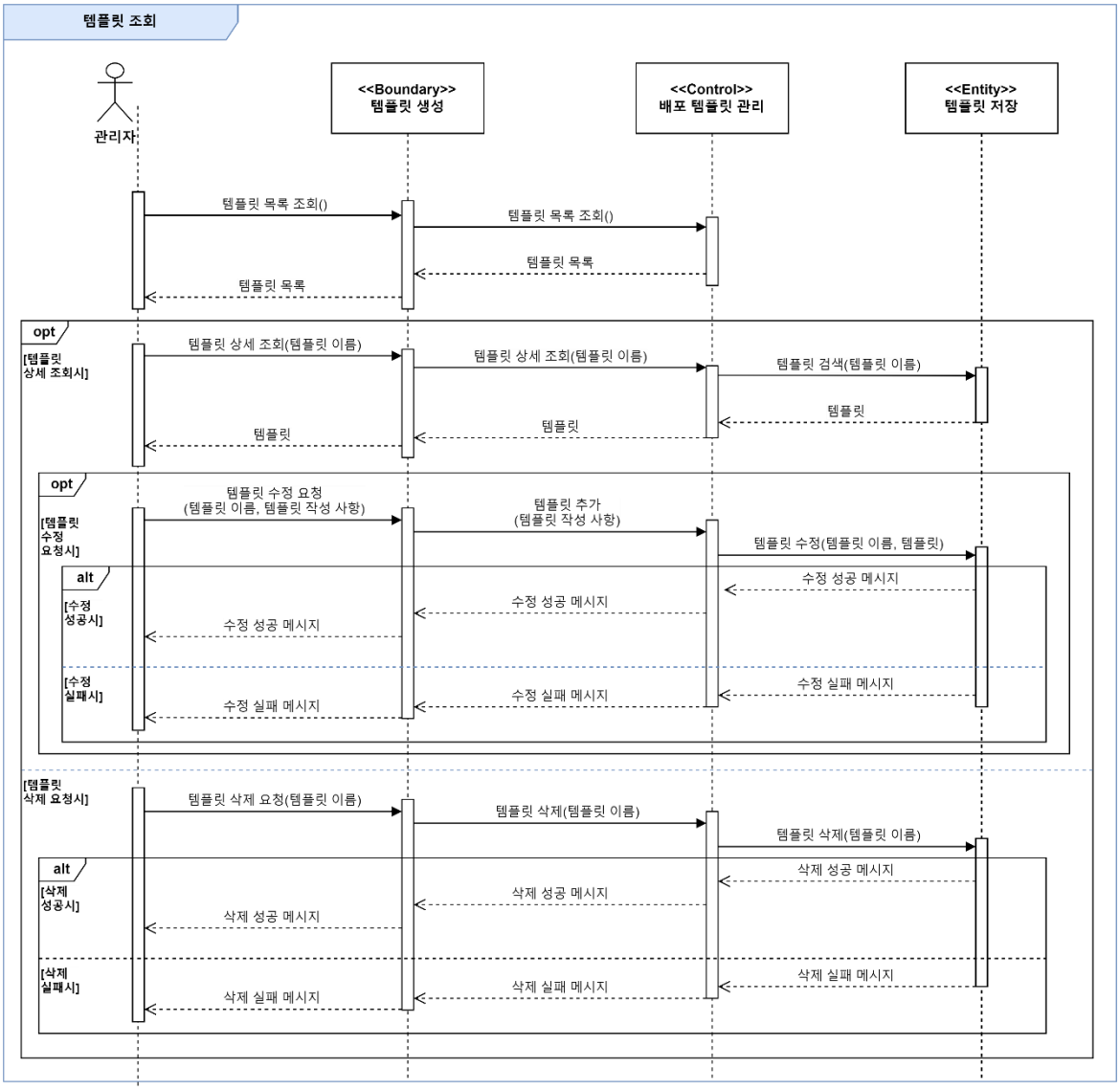

그림 15. 템플릿 조회 시퀀스 다이어그램

그림 15 는 관리자가 시스템에 등록된 템플릿을 조회하는 시퀀스 다이어그램이다.

템플릿 조회를 위해서, 관리자는 템플릿 생성 클래스의 템플릿 목록 조회를 호출한다. 이는 배포 템플릿 관리의 템플릿 목록 조회를 호출하고 시스템에 존재하는 템플릿 목록을 반환하여 사용자에게 표시한다. 템플릿의 상세 정보를 보고자 할 때, 관리자는 템플릿 생성의 템플릿 상세 조회를 호출하고, 배포 템플릿 관리의 템플릿 상세 조회를 통해 해당 템플릿의 정보를 반환 받아 사용자에게 표시한다. 반환 받은 템플릿 정보로부터 템플릿을 수정하고자 할 때, 관리자는 템플릿 생성의 템플릿 수정 요청을 호출한다. 이는 배포 템플릿 관리의 템플릿 추가를 호출하여 템플릿 저장의 템플릿 수정을 통해 저장된 템플릿 파일을 수정한다. 템플릿 수정은 수정사항이 반영되어 성공적으로 저장되면 성공 메시지를 반환하고, 실패한다면 실패 메시지를 반환하여 사용자에게 표시한다. 템플릿을 삭제하고자 할 때, 템플릿 수정 시와 같은 흐름으로 진행되며, 템플릿 삭제 요청, 템플릿 삭제를 호출하여 진행된다.

- 서비스 배포

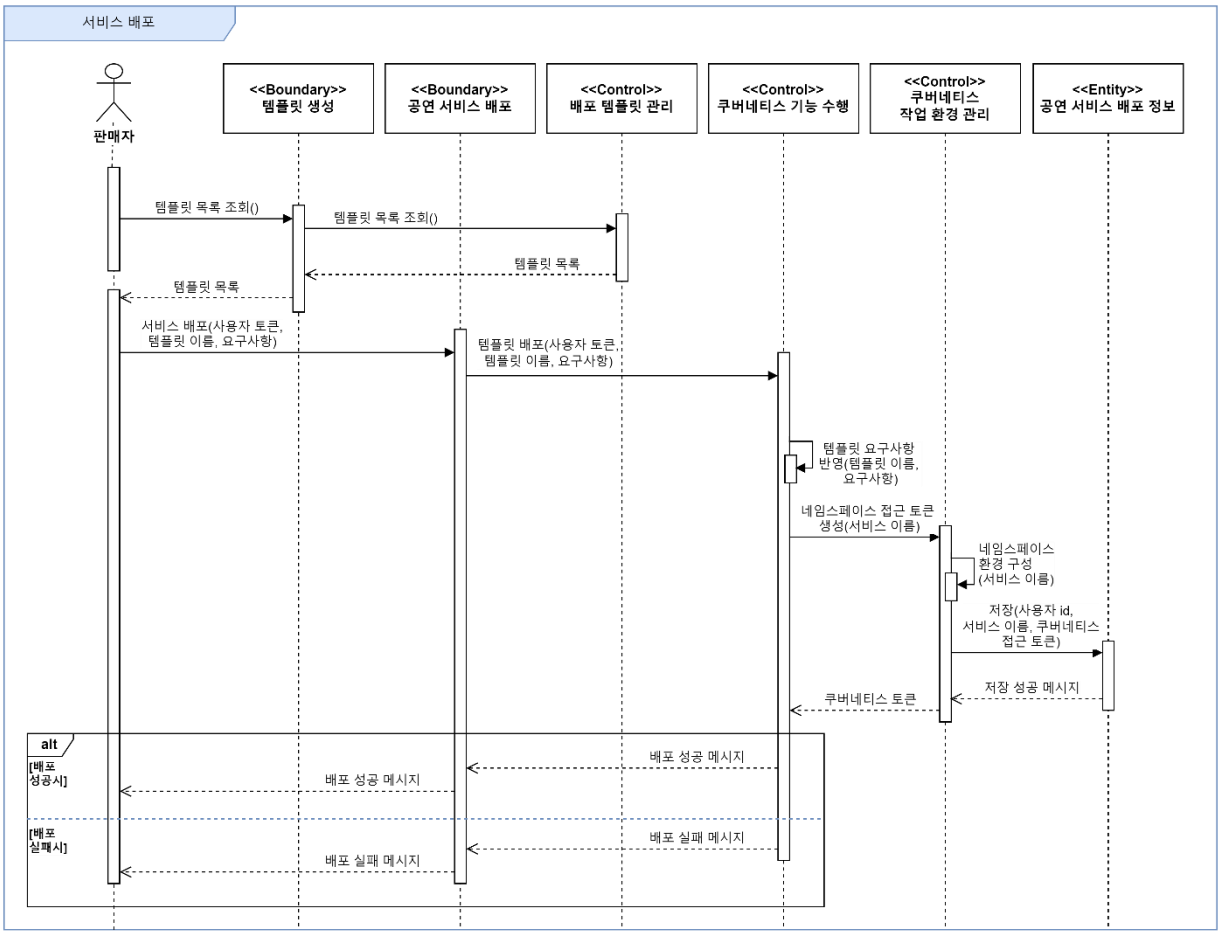

그림 16. 서비스 배포 시퀀스 다이어그램

그림 16 은 판매자가 공연 서비스를 배포하는 시퀀스 다이어그램이다.

서비스 배포를 위해, 판매자는 템플릿 생성 클래스의 템플릿 목록 조회를 호출하여 배포 템플릿 관리의 템플릿 목록 조회를 호출한다. 반환되는 템플릿 목록을 사용자에게 보여준다. 사용자는 템플릿 목록에서 원하는 템플릿을 선택하고 요구사항을 입력하여 공연 서비스 배포 클래스의 서비스 배포를 호출한다. 이는 쿠버네티스 기능 수행 클래스의 템플릿 배포를 호출한다. 템플릿 배포에서는 먼저 사용자 인증 컨트롤의 사용자 토큰 검증을 통해 사용자 검증 및 사용자 id 추출을 통해 사용자를 식별한다. 사용자가 판매자일 경우 템플릿 요구사항 반영을 통해 배포할 템플릿 문서를 완성하고 쿠버네티스 작업 환경 관리 클래스에게 네임스페이스 접근 토큰 생성을 요청한다. 쿠버네티스 작업 환경 관리는 먼저 네임스페이스 환경 구성을 통해 초기화를 수행하고 접근 토큰을 발행하여 공연 서비스 배포 정보 클래스를 통해 정보를 저장한다. 저장이 성공하면 쿠버네티스 토큰을 반환하여 템플릿 배포를 마친다. 위 과정을 포함해 배포가 완료되면 배포 성공 메시지를 반환하고, 배포에 실패하면 배포 실패 메시지를 반환하여 사용자에게 표시한다. 만약 사용자 토큰 검증 시 판매자가 아닐 경우 사용자 검증 실패 메시지를 반환하여 사용자에게 표시한다.

- 서비스 확인

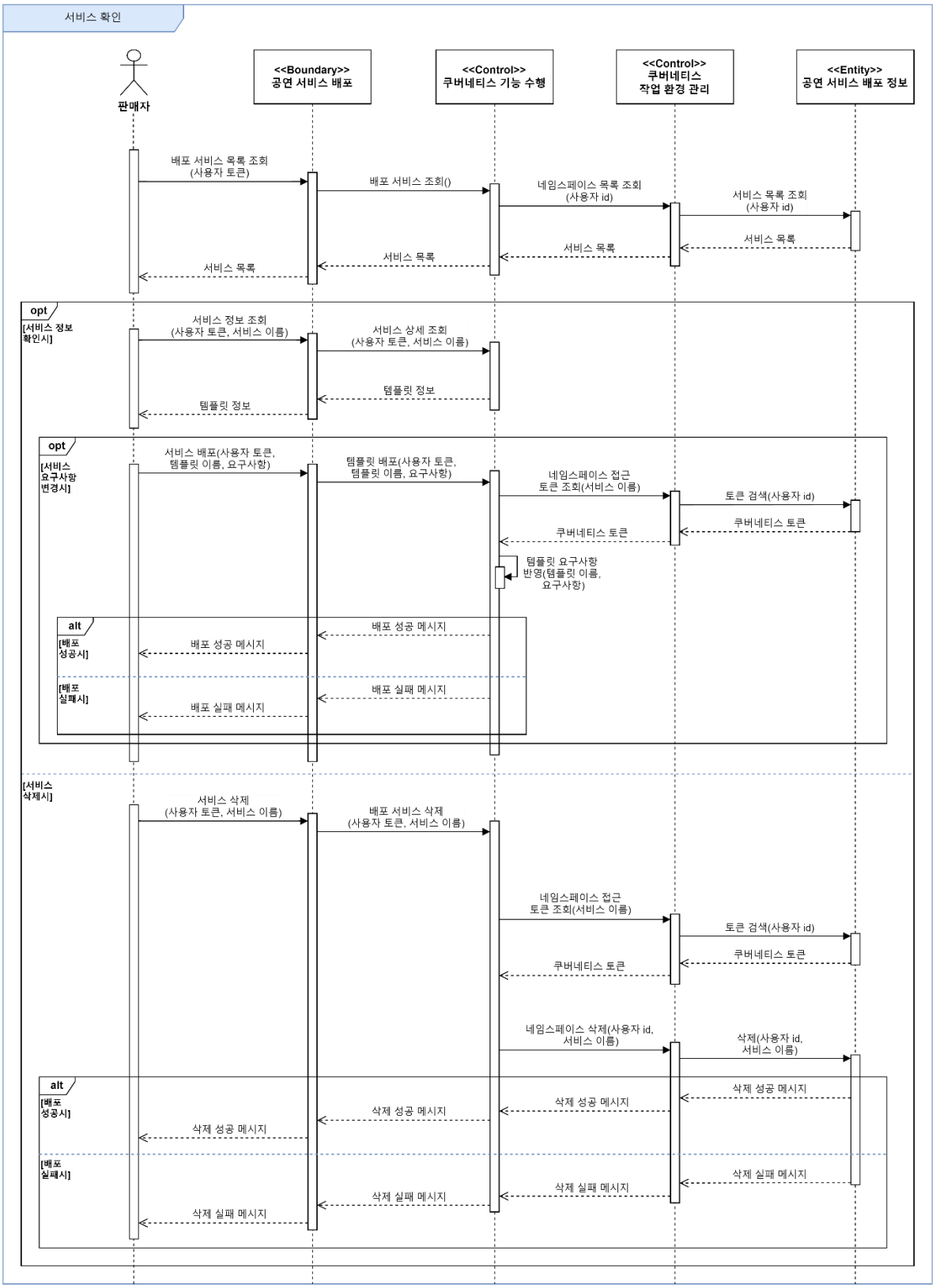

그림 17. 서비스 확인 시퀀스 다이어그램

그림 17 은 관리자가 시스템에 등록된 공연 서비스의 배포 상태를 확인하는 시퀀스 다이어그램이다.

서비스 확인을 위해서, 판매자는 공연 서비스 배포 클래스의 배포 서비스 목록 조회를 요청한다. 공연 서비스 배포는 쿠버네티스 기능 수행 클래스의 배포 서비스 조회를 요청한다. 쿠버네티스 기능 수행은 사용자 인증 클래스의 사용자 토큰 검증을 통해 사용자 검증 및 사용자 id 추출을 통해 사용자를 식별한다. 사용자 id 를 바탕으로 쿠버네티스 작업 환경 관리의 네임스페이스 목록 조회를 호출한다. 전달받은 사용자 id 를 바탕으로 공연 서비스 배포 정보를 통해 사용자 id 를 통해 배포된 서비스 목록을 조회하여 반환해준다. 반환된 서비스 목록은 사용자에게 표시된다. 만약, 사용자 토큰 검증 시 판매자가 아닐 경우 사용자 검증 실패 메시지를 반환하고 사용자에게 표시한다.

반환 받은 서비스 목록 중 배포된 서비스의 상세 정보를 확인하고자 할 때, 판매자는 공연 서비스 배포의 서비스 정보 조회를 호출한다. 이는 쿠버네티스 기능 수행의 서비스 상세 조회를 호출하고, 사용자 토큰 검증 과정을 거친다. 검증이 유효할 경우 해당 배포 시 사용된 템플릿 정보를 반환하여 사용자에게 표시한다. 이때 서비스 요구사항을 변경하고자 할 때, 템플릿 정보에서 요구사항을 변경한 뒤, 공연 서비스 배포의 서비스 배포를 호출한다. 요구사항이 변경될 시 쿠버네티스 상에 배포된 컨테이너들을 재 구동해야 하므로, 별도의 수정 작업이 아닌, 배포과정이 재실행 된다. 배포 시와의 차이점은 네임스페이스 및 환경은 미리 구성되어 있으므로, 쿠버네티스 작업 환경 관리의 네임스페이스 접근 토큰 생성을 요청하는 것이 아닌 네임스페이스 접근 토큰 조회를 호출해 공연 서비스 배포의 토큰 검색을 통해 쿠버네티스 토큰을 반환 받아 배포를 수행한다. 배포 성공 여부를 반환하여 사용자에게 표시한다.

반환 받은 서비스 목록 중 원하는 서비스를 삭제하고자 할 때, 판매자는 공연 서비스 배포의 서비스 삭제를 호출한다. 이는 쿠버네티스 기능 수행의 배포 서비스 삭제를 호출하고, 사용자 토큰 검증 과정을 거친다. 먼저 네임스페이스 접근 토큰 조회를 호출해 공연 서비스 배포 정보의 토큰 검색을 통해 삭제할 네임스페이스의 쿠버네티스 토큰을 반환 받는다. 해당 토큰을 통해 네임스페이스를 삭제하고 삭제 정보를 반영하기 위해 쿠버네티스 작업 환경 관리의 네임스페이스 삭제를 호출하여 공연 서비스 배포 정보를 통해 DB 에 삭제 정보를 반영한다. 성공적으로 삭제가 이루어졌다면 삭제 성공 메시지를 반환하고, 실패했다면 삭제 실패 메시지를 반환하여 사용자에게 표시한다.

- 공연정보 등록

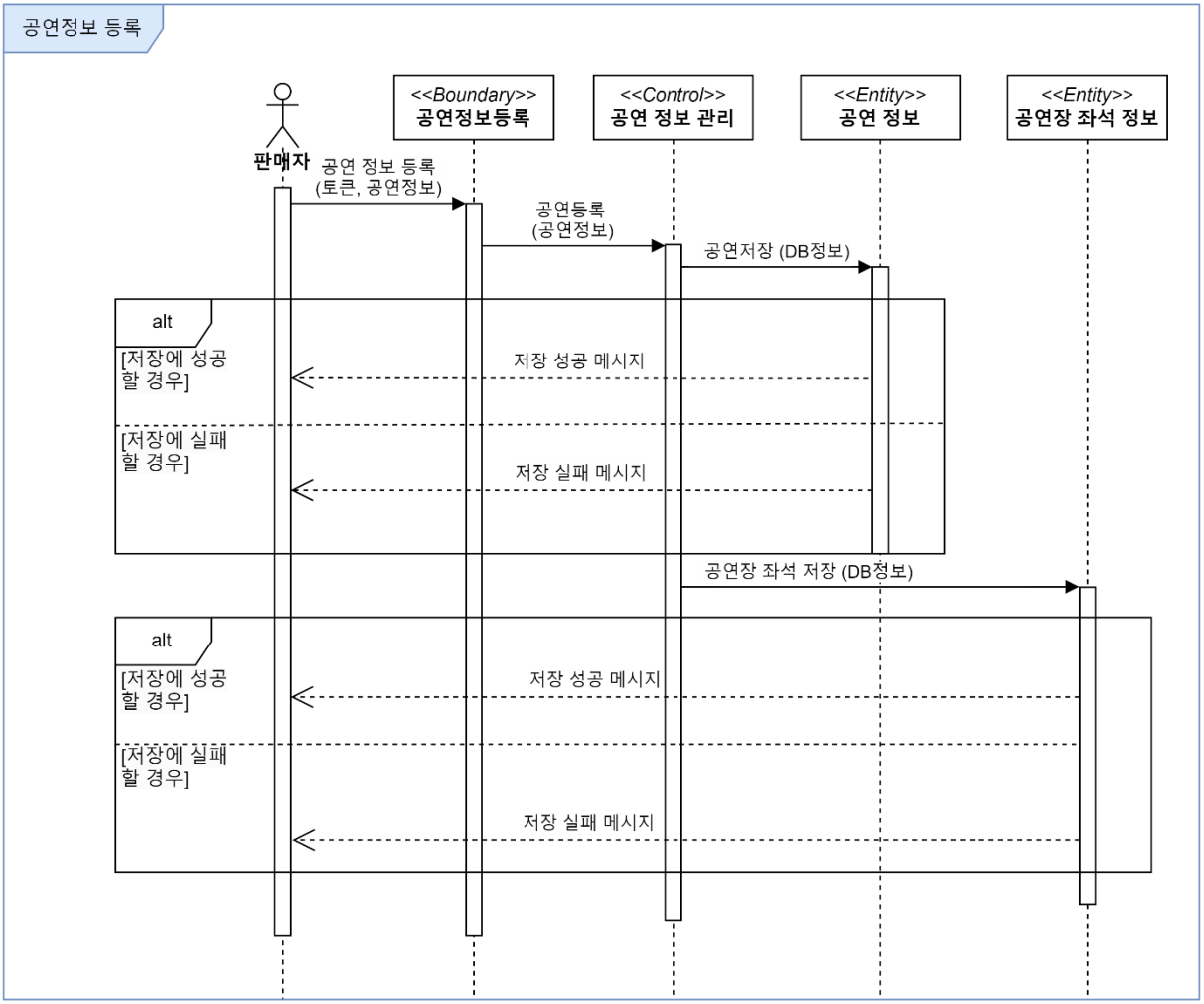

그림 18. 공연정보 등록 시퀀스 다이어그램

그림 18 은 판매자가 시스템에 배포한 공연 서비스의 정보를 등록하는 시퀀스 다이어그램이다.

판매자는 배포한 판매할 공연을 등록하기 위해 사용자 토큰과 공연 정보를 파라미터로 입력한 뒤 공연정보등록 메소드를 실행한다. 공연정보에는 공연이름, 공연날짜 공연 시작 시간, 공연 종료 시간, 공연 배우, 공연 장소, 가격 등이 해당된다. 공연 정보 등록 메소드가 실행되면 사용자 토큰 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 이때 토큰이 유효하지 않다면 권한 없음이라는 메시지를 응답을 받는다. 만약 토큰이 유효하다면 사용자 인증 클래스에서 사용자 권한 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 사용자 권한 검증을 통해 사용자가 판매자가 아닐 때는 권한 없음 메시지를 응답 받는다. 만약 사용자가 판매자일 때는 공연 정보를 파라미터로 하여 공연 등록 메소드가 실행되고 공연저장 메소드가 공연 정보 엔티티에서 실행된다. 공연 정보 DB 에는 우선 공연저장 메소스가 실행되고 공연 정보가 저장됨을 응답 받는다. 그 후 공연 좌석 저장 메소드가 실행되고 공연장 좌석이 저장되었음을 응답 받는다.

- 공연정보 수정

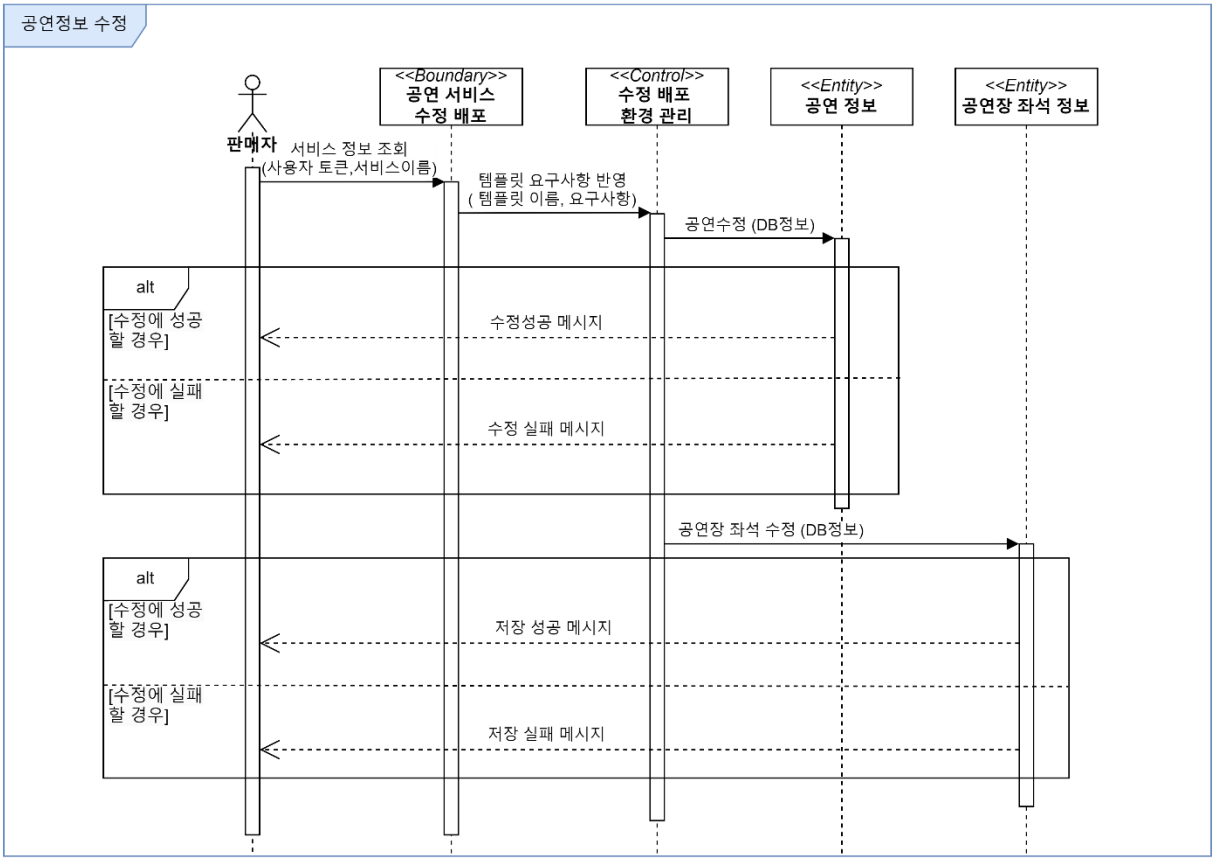

그림 19. 공연정보 수정 시퀀스 다이어그램

그림 19 는 판매자가 시스템에 배포한 공연 서비스의 정보를 수정하는 시퀀스 다이어그램이다.

판매자는 배포한 판매된 공연을 수정하기 위해 사용자 토큰과 공연 정보를 파라미터로 입력한 뒤 서비스 정보 조회 메소드를 실행한다. 공연정보에는 공연이름, 공연날짜 공연 시작 시간, 공연 종료 시간, 공연 배우, 공연 장소, 가격 등이 해당된다. 서비스 정보 조회 메소드가 실행되면 사용자 토큰 검증 메소드가 사용자 토큰을 파라미터로 하여 실행되는데 이때 토큰이 유효하지 않다면 권한 없음이라는 메시지를 응답 받는다. 만약 토큰이 유효하다면 사용자 인증 클래스에서 사용자 권한 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 사용자 권한 검증을 통해 사용자가 판매자가 아닐 때는 권한 없음이라는 메시지를 응답 받는다. 만약 사용자가 판매자일 때는 사용자 요구사항과 템플릿이름을 파라미터로 하여 템플릿 요구사항 반영 메소드를 실행시킨다. 판매자의 요구사항을 반영한 템플릿의 공연 정보를 파라미터로 하여 공연수정이라는 메소드를 실행시켜 공연을 수정한다. 수정에 성공하면 수정성공메시지를 실패하면 수정 실패 메시지를 응답 받는다.

- 공연정보 조회

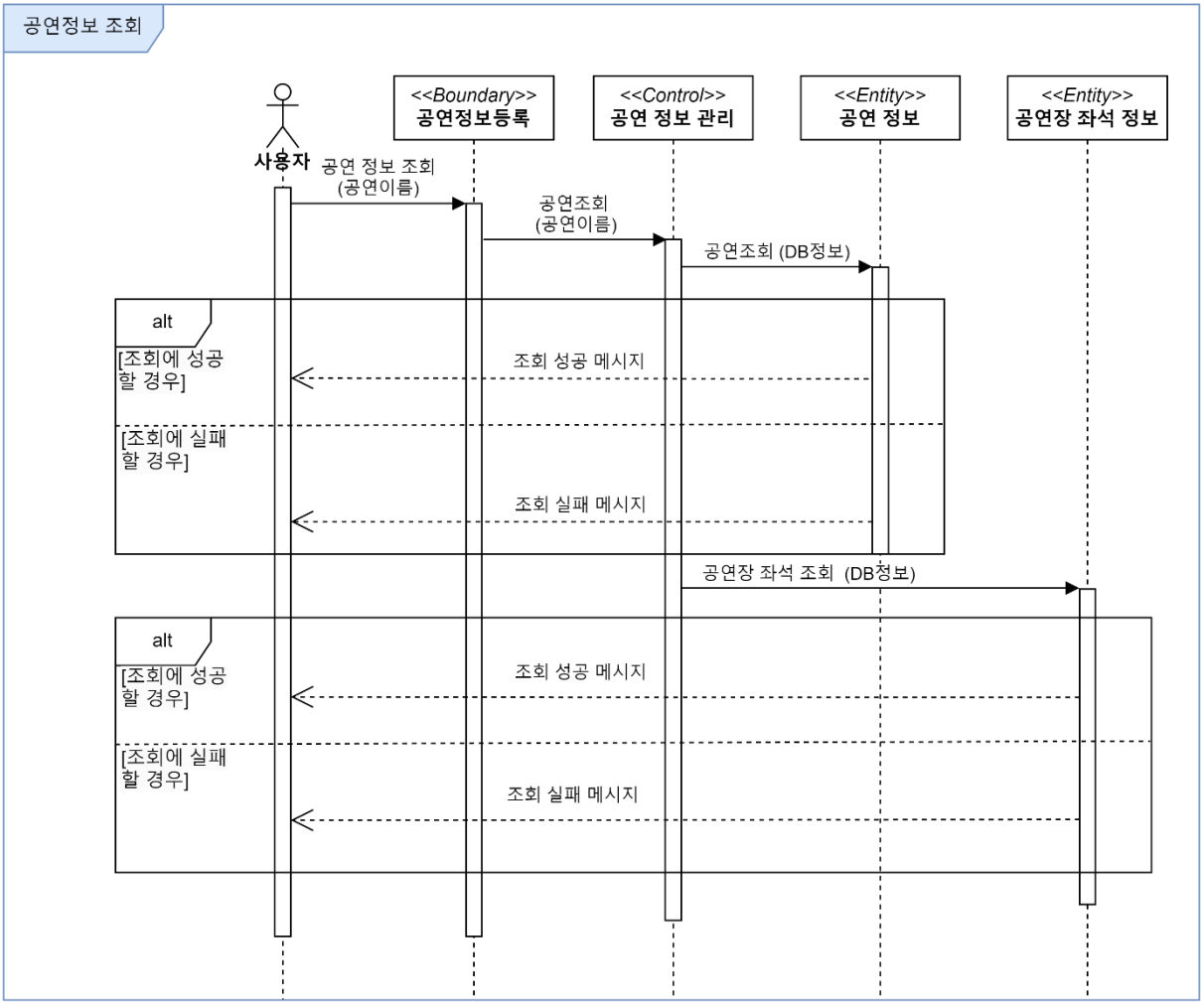

그림 20. 공연정보 조회 시퀀스 다이어그램

그림 20 은 판매자가 시스템에 배포한 공연 서비스의 정보를 수정하는 시퀀스 다이어그램이다.

사용자는 판매중인 공연을 조회하기 위해 사용자 토큰과 공연 이름을 파라미터로 입력한 뒤 공연 정보 조회 메소드를 실행한다. 공연 정보 조회 메소드가 실행되면 사용자 토큰 검증 메소드가 사용자 토큰을 파라미터로 하여 실행한다. 이때 토큰이 유효하지 않다면 권한 없음이라는 메시지를 응답 받는다. 만약 토큰이 유효하다면 사용자 인증 클래스에서 사용자 권한 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 공연 조회, 공연장 좌석 정보 조회 메소드가 DB 정보를 파라미터로 하여 실행되고 조회에 성공할 경우 조회 성공 메시지를 조회에 실패할 경우 조회 실패 메시지를 응답 받는다.

- 잔여 좌석 조회

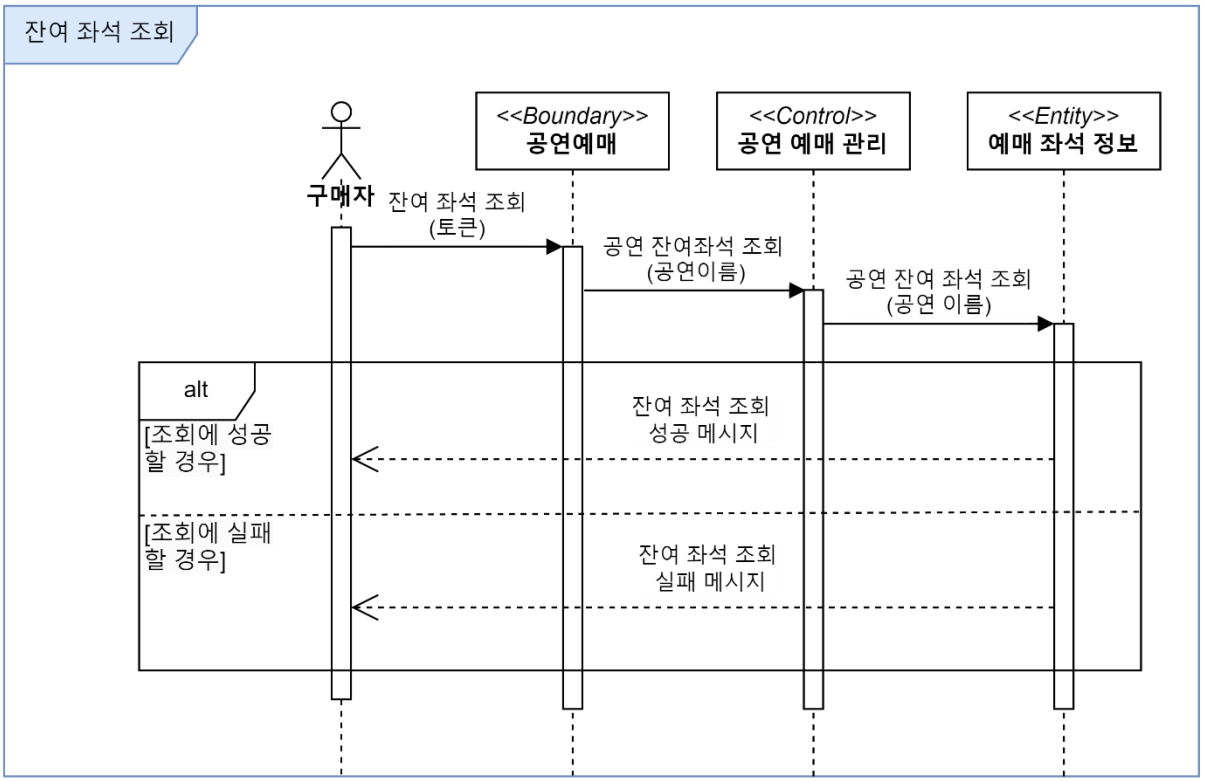

그림 21. 잔여 좌석 조회 시퀀스 다이어그램

그림 21 은 구매자가 공연의 잔여 좌석 정보를 조회하는 시퀀스 다이어그램이다.

구매자는 배포한 판매중인 공연에 대해서 잔여 좌석을 조회하기 위해 사용자 토큰과 공연 이름을 파라미터로 입력한 뒤 잔여 좌석 조회 메소드를 실행한다. 잔여 좌석 조회 메소드가 실행되면 사용자 토큰 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 이때 토큰이 유효하지 않다면 권한 없음이라는 메시지를 응답 받는다. 만약 토큰이 유효하다면 공연 잔여 좌석 조회 메소드가 공연이름을 파라미터로 하여 실행된다. 조회에 성공할 경우 잔여 좌석 조회 성공 메시지를 조회에 실패할 경우 잔여 좌석 조회 실패 메시지를 응답 받는다.

- 좌석 예매

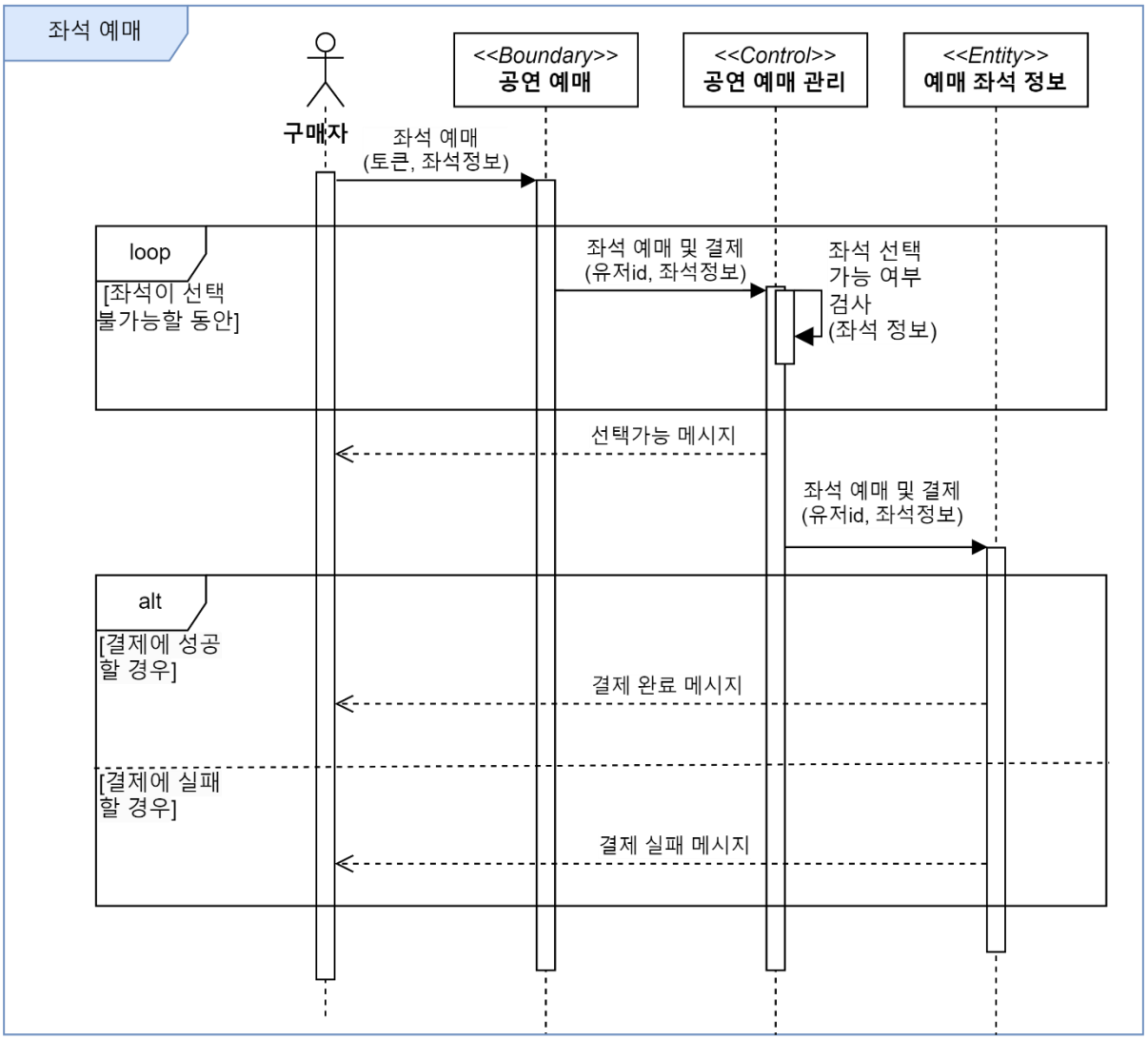

그림 22. 좌석 예매 시퀀스 다이어그램

그림 22 는 구매자가 티켓팅을 진행하며 좌석을 예매하는 시퀀스 다이어그램이다.

구매자는 판매 중인 공연에 대해서 예매하기 위해 사용자 토큰과 좌석 정보를 파라미터로 입력한 뒤 좌석 예매 메소드를 실행한다. 좌석 정보에는 좌석 종류, 좌석 가격, 좌석 번호가 해당한다. 좌석 예매 메소드가 실행되면 사용자 토큰 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 이때 토큰이 유효하지 않다면 권한 없음이라는 메시지를 응답 받는다. 만약 토큰이 유효하다면 사용자 권한 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 사용자가 구매자가 아닐 경우에는 권한 없음이라는 메시지를 응답 받는다. 만약 사용자가 구매자라면 사용자 ID 추출이라는 메소드가 사용자 토큰을 파라미터로 하여 수행된다.

구매자는 좌석 정보와 토큰을 파라미터로 하여 좌석 예매 및 결제 메소드를 수행한다. 공연 예매 관리 클래스에서 좌석 선택 가능 여부 검사 메소드가 실행되어 선택 가능하다는 응답을 받을 때까지 좌석 예매 및 결제 메소드를 실행시켜야 한다. 선택 가능이라는 응답을 받으면 좌석 예매 및 결제 메소드가 실행되고 결제에 성공할 경우 결제 완료 메시지를 결제에 실패할 경우 결제 실패 메시지를 응답 받는다.

- 구매 이력 조회

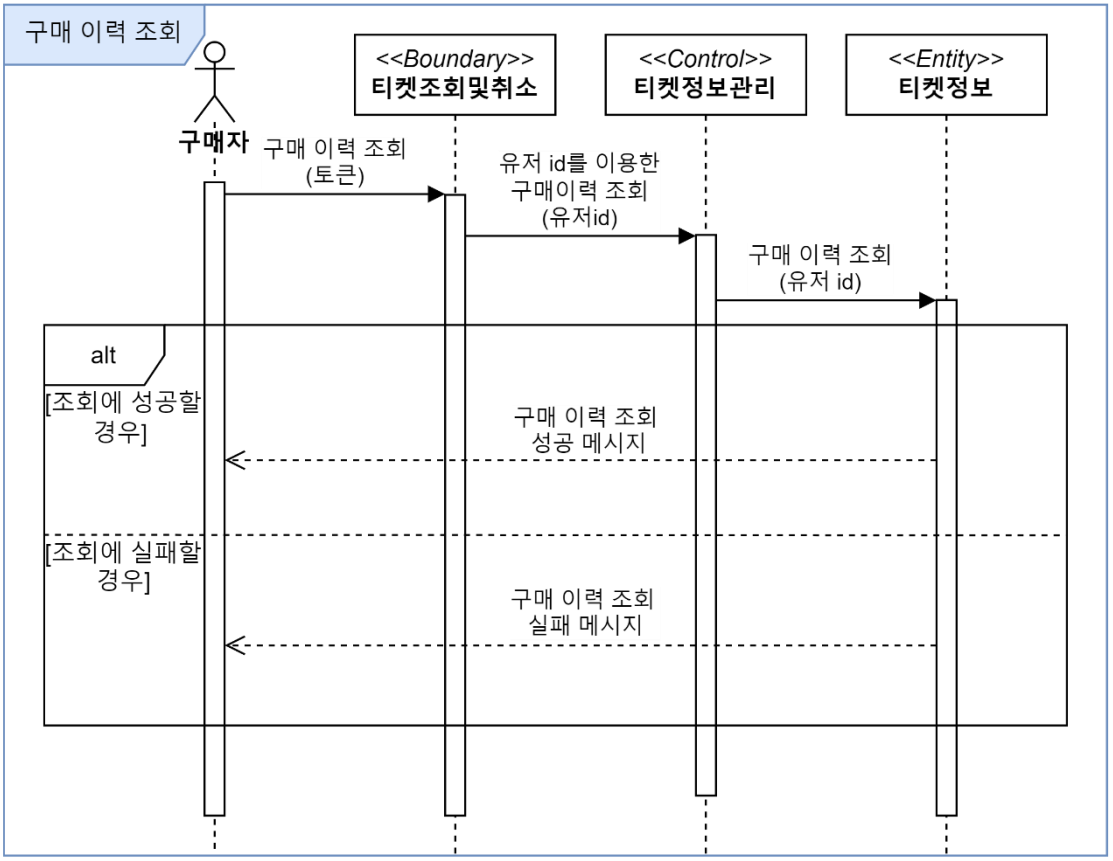

그림 23. 구매 이력 조회 시퀀스 다이어그램

그림 23 은 구매자가 티켓팅을 진행하며 좌석을 예매하는 시퀀스 다이어그램이다.

구매자는 구매한 티켓 전부에 대해 조회하기 위해서 구매이력 조회 메소드를 사용자 토큰을 파라미터로 하여 실행한다. 사용자 토큰 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 이때 토큰이 유효하지 않다면 권한 없음이라는 메시지를 응답 받는다. 만약 토큰이 유효하다면 사용자 권한 검증 메소드가 사용자 토큰을 파라미터로 하여 실행되고 사용자가 구매자가 아닐 경우 권한 없음이라는 메시지를 응답 받는다. 만약 사용자가 구매자일 경우 유저 ID 추출 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 유저 ID 를 이용한 구매이력 조회 메소드가 유저 ID 를 파라미터로 하여 실행된다. 티켓 정보 엔티티 클래스에서 구매이력 조회 메소드가 실행된다. 조회에 성공할 경우 구매이력 조회 성공 메시지를 조회에 실패할 경우 구매이력 조회 실패 메시지를 응답 받는다.

- 예매 티켓 조회

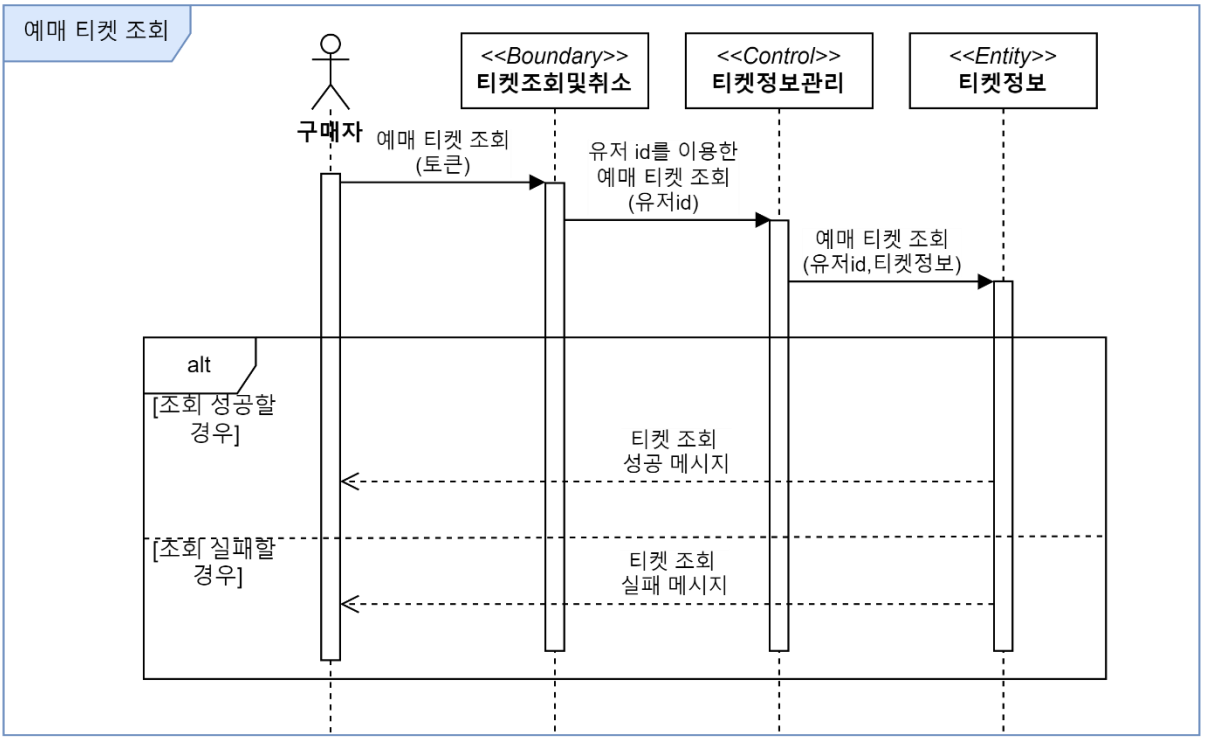

그림 24. 예매 티켓 조회 시퀀스 다이어그램

그림 24 는 구매자가 자신이 예매한 티켓을 조회하는 시퀀스 다이어그램이다.

구매자는 구매한 티켓 중에서 사용할 수 있는 티켓에 대해 조회하기 위해서 예매 티켓 조회 메소드를 사용자 토큰을 파라미터로 하여 실행한다. 사용자 토큰 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 이때 토큰이 유효하지 않다면 권한 없음이라는 메시지를 응답 받는다. 만약 토큰이 유효하다면 사용자 권한 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 사용자가 구매자가 아닐 경우 권한 없음이라는 메시지를 응답 받는다. 만약 사용자가 구매자일 경우 유저 ID 추출 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 유저 ID 를 이용한 예매 티켓 조회 메소드가 유저 ID 를 파라미터로 하여 실행된다. 조회에 성공하면 티켓 조회 성공 메시지를 실패하면 티켓 조회 실패 메시지를 응답 받는다.

- 예매 취소

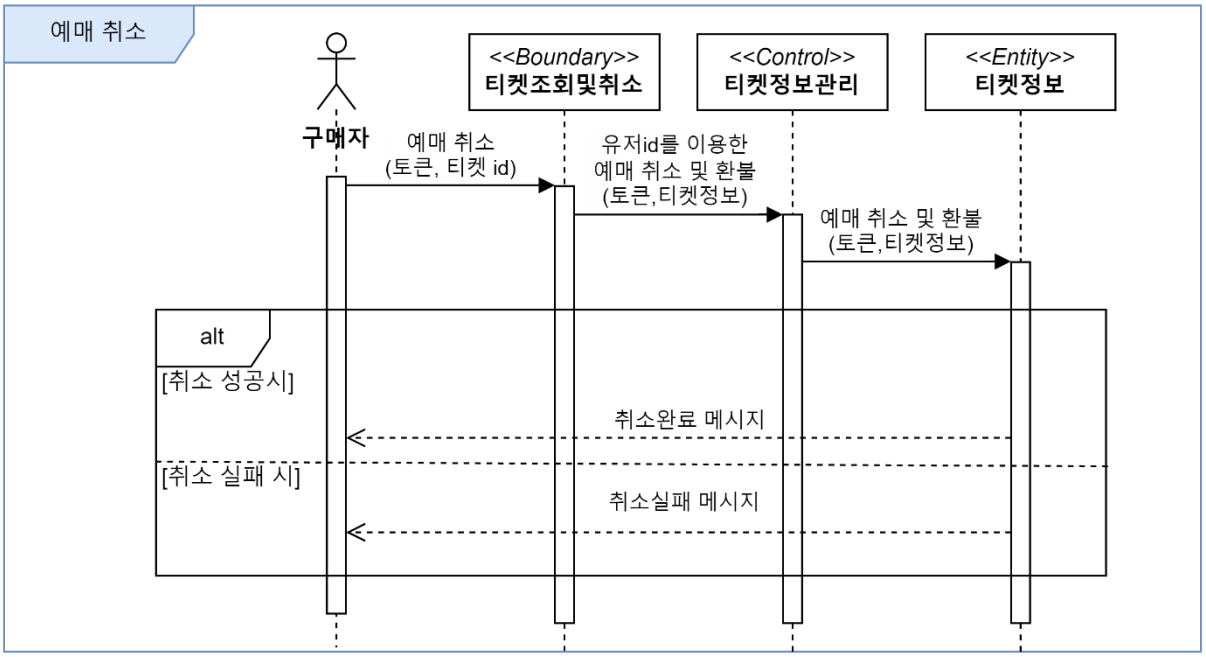

그림 25. 예매 취소 시퀀스 다이어그램

그림 25 는 구매자가 자신이 예매한 티켓을 취소하는 시퀀스 다이어그램이다.

구매자는 구매한 티켓 중에서 예매 취소를 하기 위해서 예매 취소 메소드를 사용자 토큰과 티켓정보를 파라미터로 하여 실행한다. 사용자 토큰 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 이때 토큰이 유효하지 않다면 권한 없음이라는 메시지를 응답 받는다. 만약 토큰이 유효하다면 사용자 권한 검증 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 사용자가 구매자가 아닐 경우 권한 없음이라는 메시지를 응답 받는다. 만약 사용자가 구매자일 경우 유저 ID 추출 메소드가 사용자 토큰을 파라미터로 하여 실행된다. 유저 ID 를 이용한 예매 티켓 취소 및 환불 메소드가 티켓정보를 파라미터로 하여 실행된다. 취소 성공할 경우 취소 완료 메시지를 취소에 실패할 경우 취소 실패 메시지를 응답 받는다.

③ 마이크로서비스 명세

클래스 다이어그램 기반의 기능들을 바탕으로 마이크로서비스 명세를 작성하였다. 클래스 이름과 BCE 패턴, 연결 가능 서비스 항목은 클래스 다이어그램을 기반으로 작성하였다. 메소드 항목은 각 마이크로서비스에서 호출하는 메소드의 HTTP 메소드 종류를 나타낸 것이다. 약결합 URL 항목은 구현 단계에서 각 마이크로서비스들 간 연결을 위한 경로를 지정하기 위한 주소를 마이크로서비스 기능에 맞는 이름으로 간단하게 명명한 것이다.

표 3. Boundary 마이크로서비스 명세

클래스 이름	BCE 패턴	연결 가능 서비스	메소드	약결합 URL
사용자 인증	Boundary	사용자 인증 관리	POST	/auth/view
예매 현황 조회	Boundary	사용자 인증, 예매 정보 관리	GET	/monitor/ticket/view
공연 서버 리소스 사용량 조회	Boundary	사용자 인증, 공연 서비스 모니터링 관리	GET	/monitor/server/view
마이크로서비스 배포 환경 조회	Boundary	마이크로서비스 배포 환경 정보 관리	GET	/system/monitor/view
마이크로서비스 리소스 확장	Boundary	마이크로서비스 배포 환경 관리	POST	/system/env/view
마이크로서비스 추가	Boundary	마이크로서비스 풀 관리	GET, POST, DELETE	/pool/view
템플릿 생성	Boundary	배포 템플릿 관리	GET, POST, DELETE, PUT	/template/view
공연 서비스 배포	Boundary	쿠버네티스 기능 수행	GET, POST, DELETE, PUT	/deploy/view
공연 서비스 수정 배포	Boundary	수정 배포 환경 관리	GET, POST, DELETE, PUT	/modify
공연 정보 등록	Boundary	공연 정보 관리	GET, POST	/apply
공연 예매	Boundary	공연 예매 관리	GET, POST	/buyticket
티켓 조회 및 취소	Boundary	티켓 정보 관리	GET, POST	/ticket

표 3 은 Boundary 마이크로서비스의 명세를 나타낸 것이다.

Boundary 마이크로서비스는 사용자와 상호작용하는 마이크로서비스로, 사용자의 입력을 받아 Control 마이크로서비스로 요청을 보내거나 Control 마이크로서비스에 요청한 메소드 응답을 사용자에게 보여주는 등의 기능을 수행한다.

표 4. Control 마이크로서비스 명세

클래스 이름	BCE 패턴	연결 가능 서비스	메소드	약결합 URL
사용자 인증 관리	Control	사용자 인증, 사용자 정보 조회, 세션 토큰 정보 조회, 예매 정보 관리, 공연 서비스 모니터링 관리	GET, POST, PUT	/auth
예매 정보 관리	Control	사용자 인증 관리, 예매 현황 조회, 좌석 예매 정보 조회	GET	/monitor/ticket
공연 서비스 모니터링 관리	Control	사용자 인증 관리, 공연 서버 리소스 사용량 조회, 네임스페이스 접근 토큰 조회, 리소스 사용량 조회	GET	/monitor/server
마이크로서비스 배포 환경 정보 관리	Control	마이크로서비스 배포 환경 조회, 리소스 사용량 조회, 배포한 마이크로서비스 조회	GET	/system/monitor
마이크로서비스 배포 환경 관리	Control	마이크로서비스 리소스 확장, 배포한 마이크로서비스 조회	POST	/system/env
마이크로서비스 풀 관리	Control	마이크로서비스 추가, 마이크로서비스 이미지 저장	GET, POST, DELETE	/pool
배포 템플릿 관리	Control	템플릿 생성, 템플릿 저장	GET, POST, DELETE, PUT	/template
쿠버네티스 기능 수행	Control	공연 서비스 배포, 쿠버네티스 작업환경 관리	GET, POST, DELETE, PUT	/deploy
쿠버네티스 작업 환경 관리	Control	쿠버네티스 기능 수행, 공연 서비스 배포 정보	GET, DELETE	/deploy/env
수정 배포 환경 관리	Control	공연 서비스 수정 배포, 공연 정보, 공연장 좌석 정보	POST, DELETE, PUT	/modify/management
공연 정보 관리	Control	공연 정보 등록, 공연 정보, 공연장 좌석 정보	GET, POST	/apply/management
공연 예매 관리	Control	공연예매, 예매 좌석정보	POST, PUT	/buyticket/management
티켓 정보 관리	Control	티켓 조회 및 취소, 티켓 정보	GET, POST	/ticket/management

표 4 는 Control 마이크로서비스의 명세를 나타낸 것이다.

Control 마이크로서비스는 Boundary 마이크로서비스 및 Entity 마이크로서비스와 연결되어 본 과제 시스템의 핵심적인 로직들을 수행한다. Boundary 마이크로서비스를 통해 전달받은 파라미터를 Entity 마이크로서비스에 전달하여 DB 에 저장하거나, Boundary 마이크로서비스가 요청한 데이터 정보를 Entity 마이크로서비스를 통해 조회하여 반환하는 등의 기능을 수행한다.

표 5. Entity 마이크로서비스 명세

클래스 이름	BCE 패턴	연결 가능 서비스	메소드	약결합 URL
사용자 정보 조회	Entity	사용자 인증 관리	GET, POST	/auth/data
세션 토큰 정보 조회	Entity	사용자 인증 관리	GET, POST, PUT	/auth/token
좌석 예매 정보 조회	Entity	예매 정보 관리	GET, POST	/monitor/ticket/data
네임스페이스 접근 토큰 조회	Entity	공연 서비스 모니터링 관리	GET, POST	/namespace/token/data
리소스 사용량 조회	Entity	공연 서비스 모니터링 관리, 마이크로서비스 배포 환경 정보 관리	GET	/monitor/server/data
배포한 마이크로서비스 조회	Entity	마이크로서비스 배포 환경 정보 관리, 마이크로서비스 배포 환경 관리	GET, POST	/system/env/data
마이크로서비스 이미지 저장	Entity	마이크로서비스 풀 관리	GET, POST, DELETE	/pool/data
템플릿 저장	Entity	배포 템플릿 관리	GET, POST, DELETE, PUT	/template/data
공연 서비스 배포 정보	Entity	쿠버네티스 작업환경 관리	GET, POST, DELETE	/deploy/data
공연 정보	Entity	수정 배포 환경 관리	GET, POST, PUT	/event
공연장 좌석 정보	Entity	공연장 좌석 정보	POST, PUT	/seat
예매 좌석 정보	Entity	공연 예매 관리	GET, POST	/buyticket/seat
티켓 정보	Entity	티켓 정보 관리	GET, POST, DELETE	/ticket/information

표 5 는 Entity 마이크로서비스의 명세를 나타낸 것이다.

Entity 마이크로서비스는 DB 나 쿠버네티스, Prometheus 와 같이 외부 시스템과 연결되어 Control 마이크로서비스의 요청을 전달한다. 주로 DB 와 연결되어 데이터 저장, 조회, 업데이트 등의 기능을 수행한다.

2) 전체 구성도

그림 26 은 해당 시스템의 배포도를 나타낸다.

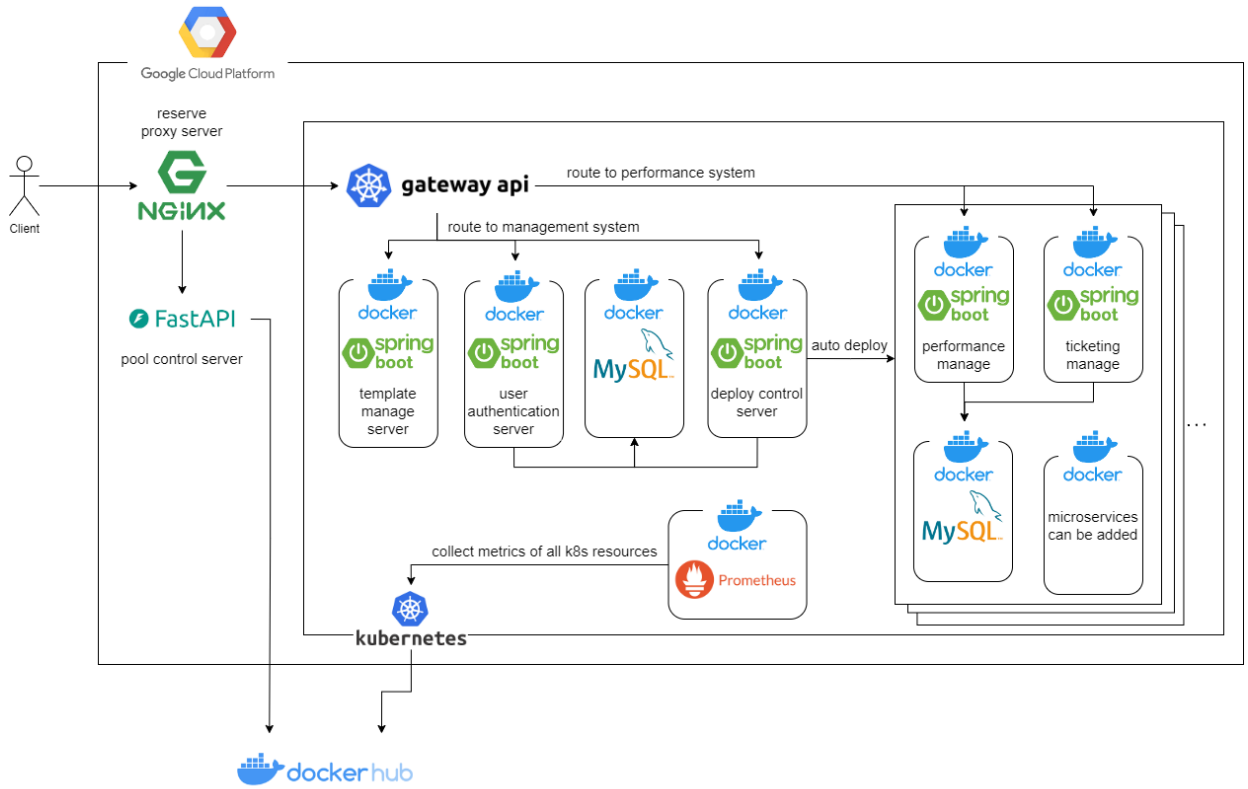

그림 26. 시스템 배포도

그림 26 에 나타낸 바와 같이 본 과제에서 제안하는 시스템은 구글 클라우드 플랫폼을 활용하여 구축하였다.

사용자는 시스템 활용을 위해 NGINX 기반의 서비스에 접근한다. 사용자가 활용하고자 하는 기능에 따라 NGINX 서비스가 요청하는 대상이 달라지게 된다. 쿠버네티스에 배포된 서비스로의 접근은 쿠버네티스 Gateway API 로 전달되어 각 서비스로 전달된다.

① 마이크로서비스 풀 관리

마이크로서비스 풀에 대한 관리를 수행하는 경우, 사용자 요청은 FastAPI 기반의 풀 관리 서버에 전달된다. 해당 서버는 도커 허브와 통신하여 마이크로서비스 이미지를 관리한다.

② 사용자 인증

사용자는 시스템 이용을 위해 로그인 및 세션 정보를 가지고 있어야 한다. 이때 사용자 요청은 Gateway API 를 통해 사용자 인증 서버로 전달된다. 해당 서버는 사용자 인증을 위해 JWT 기반의 토큰을 발행하여 로그인 세션을 관리한다.

③ 템플릿 관리

사용자가 템플릿을 관리하고자 할 경우, 사용자 요청은 Gateway API 를 통해 템플릿 관리 서버로 전달된다. 해당 서버는 자체 파일 시스템 상에 파일 형태로 템플릿 파일을 저장 및 관리한다.

④ 마이크로서비스 배포 관리

사용자가 서비스 배포 및 관리를 수행하고자 할 경우, 사용자 요청은 Gateway API 를 통해 배포 관리 서버로 전달된다. 배포 관리 서버는 사용자 입력으로부터 템플릿 및 요구사항을 전달받아 쿠버네티스 상에 배포를 진행한다.

⑤ 공연 서비스 사용

사용자가 기 배포된 공연 서비스를 사용하고자 할 경우, 사용자 요청은 Gateway API 를 통해 공연 관리, 티켓 관리 등의 서버로 전달된다. 해당 서버들은 공연 서비스 배포를 위해 구성된 템플릿에 따라 다른 마이크로서비스로 구성될 수 있으며, 공연에 대한 티켓팅 과정을 수행한다.

4. 구성원 별 개발 진척도

다음 표 6 은 구성원 별 개발 진척도이다.

표 6. 구성원 별 개발 진척도

이름	진척도
강찬석	1. 클라우드 인프라 구축 2. 컨테이너 풀 관리 서버 개발 - 도커 이미지 빌드, 도커 허브에 이미지 업로드 3. 템플릿 관리 서버 개발 - deployment, service, httproute 로 구성된 배포 템플릿 생성 4. 시스템 배포 관리 서버 개발 - 배포 템플릿을 통한 마이크로서비스 자동 배포
이동현	1. 클라이언트 개발 - 마이크로프론트엔드 아키텍처 적용하여 각 애플리케이션 독립적으로 개발 - 클라이언트 wireframe 작성 2. 사용자 인증 애플리케이션 개발 - 시스템 접근 권한 요청, 토큰 만료 시 갱신 요청 기능 3. 판매자 애플리케이션 개발 - 공연 배포 요청, 배포한 공연 목록 조회 요청 기능 4. 모니터링 시스템 개발 - 시스템 리소스 사용량 수집을 위한 Prometheus 및 cAdvisor 배포 - JavaScript 그래픽 라이브러리를 통해 리소스 사용량 정보 가시화
조용진	1. 사용자 인증 서버 개발 - JWT 토큰으로 사용자 검증 및 역할 분리 - ACCESS, REFRESH 토큰으로 분리, 로그인 유지 및 탈취방어 2. 유저 애플리케이션 - 판매자가 사용하는 공연 등록 및 필수 기능 구현 - 구매자가 사용하는 결제, 취소 및 필수 기능 구현 3. 쿠버네티스 트래픽 분산을 위한 네트워크 구축 - 유저 애플리케이션 서버 통신 간의 데이터 무결성 유지 - envoy proxy 를 활용한 sidecar 패턴 기반의 service mesh 구현

5. 과제 수행 내용 및 중간 결과

1) 구현 변경 내역

① 컨테이너 풀 저장소

본 과제에서는 컨테이너 기반의 오픈소스 가상화 플랫폼인 도커를 사용하여 컨테이너를 관리한다. 각 마이크로서비스는 도커 이미지의 형태로 컨테이너 풀 저장소에 저장 및 관리된다. 기존에는 도커에서 공식으로 제공하는 도커 레지스트리 이미지를 사용하여 프라이빗 레지스트리를 구축하여 컨테이너 풀 저장소로 사용하려 하였다. 하지만 클라우드 리소스를 사용함으로 인해 발생하는 비용 등의 효율성의 문제로 퍼블릭 레지스트리인 도커 허브를 컨테이너 풀 저장소로 사용하여 시스템을 구축하였다.

② 모니터링 정보 가시화

모니터링 정보 가시화는 착수 단계에서 Grafana 기반의 오픈소스 모니터링 도구를 활용하기로 계획하였다. 그러나, Grafana 의 경우 추가적인 인증 절차가 필요하며, 대시보드 임베딩 방식이 사용자 입장에서 비직관적인 형태를 띄고 있는 것을 실험 실습을 통해 확인하였다. 따라서, Prometheus API 를 직접적으로 활용하여 가져온 리소스 데이터를 JavaScript 의 그래픽 라이브러리로 시각화 하는 방식으로 모니터링 정보 가시화를 구현하였다.

③ 개발 언어

기존 개발 계획은 백 엔드 서버의 역할을 하는 마이크로서비스의 개발 언어로 spring boot 를 사용하여 개발하려고 하였다. 하지만 실제 개발을 진행하면서 언어에 대한 숙련도 문제와, 쿠버네티스 SDK 등 다양한 라이브러리를 적용하는 데 시간이 오래 걸리는 문제를 해결하고자 하였다. 따라서 일부 마이크로서비스는 spring boot 가 아닌 Flask, Fast-api 를 사용해 개발을 진행하였다.

2) 인프라 설계 및 구축

본 과제를 수행하기 위해서, 시스템을 구축하기 위한 인프라 환경으로 구글의 퍼블릭 클라우드 컴퓨팅 플랫폼(GCP)에서 제공하는 가상머신 환경을 이용하였다. 쿠버네티스 노드의 역할에 따라 클러스터 노드, 워커 노드를 각각 한 개씩 2 개의 가상머신으로 환경을 구성하였다.

사용자가 시스템으로 접근하고자 할 때, 시스템 앞 단에 배치된 NGINX 를 통해 각 서버 및 쿠버네티스 애플리케이션으로 접근한다. 쿠버네티스 애플리케이션으로의 접근은 게이트웨이 API 방식으로 접근이 가능하게 하였다. MetalLB, Istio, k8s Gateway API CRD(Custom Resource Definition)를 배치해 쿠버네티스 애플리케이션과 연결되는 게이트웨이 API 시스템을 구성하였다. MetalLB 는 쿠버네티스 외부와 통신할 수 있는 IP 를 할당하고, Istio 는 각 애플리케이션을 Sidecar 패턴으로 구성되는 Service Mesh 형태로 네트워크 환경을 구성한다. 그리고 k8s Gateway API CRD 는 통해 Gateway 및 HTTPRoute 에 대한 쿠버네티스 리소스 정의를 바탕으로 사용자 요청을 라우팅 해준다.

3) 공연 배포 템플릿 구성

판매자는 공연 예매 서비스를 배포하기 위해 기 작성된 배포 템플릿을 선택한다. 배포 템플릿은 공연 예매 서비스에 필요한 컨테이너들을 묶어서 한 번에 쿠버네티스에 배포할 수 있게 모아놓은 yaml 확장자 파일이다. 배포 템플릿의 내용으로는 컨테이너 배포를 위한 deployment 에 대한 내용과 배포된 컨테이너들을 네트워크 상에 노출시키기 위한 service, 그리고 Gateway 를 통해 외부와 연결하기 위한 HTTPRoute 로 구성된다

3) 배포 컨테이너 접근 권한 제어

사용자 인증, 배포 관리 서버와 같은 애플리케이션은 시스템 전체에 하나만 존재하는 반면, 자동 배포를 통해 배포된 공연 관련 마이크로서비스들은 공연마다 생성되며 각각의 애플리케이션은 관련된 공연의 마이크로서비스로만 연결되어야 한다.

쿠버네티스에서는 컨테이너 오케스트레이션을 위한 모든 리소스를 namespace 단위로 분리하여 관리한다. namespace 상의 리소스에 대한 접근 및 관리 권한은 RBAC(Role-Based-Access-Control) 기반으로 이루어진다.

판매자가 공연을 관리하는 상황을 상정했을 때, 판매자는 공연 관계자와 공동 관리를 원할 수 있다. 이때 판매자 계정 단위로 namespace 를 생성해 접근제어를 수행했을 때, 여러 공연을 관리하는 판매자는 공연 관계자에게 해당 공연만 접근할 수 있도록 권한을 주어야 하는데, 이를 위해서 별도로 리소스 정보를 관리해야 하는 등 번거로운 추가 작업이 동반될 수 있다는 단점이 있다. 따라서 본 과제에서는 사용자별 접근제어를 위해 배포 공연 당 namespace 를 할당하여 리소스를 관리하였다.

4) 모니터링 시스템

모니터링 시스템은 배포한 각 컨테이너들의 리소스 사용량을 수집 및 제공하는 기능을 수행한다.

모니터링 시스템에서 cAdvisor 를 활용하여 컨테이너들의 리소스 사용량을 수집하고 Prometheus 에 저장한다. 이후 Prometheus 에 저장된 리소스 사용량 정보를 Prometheus API 를 통해 가져와 클라이언트로 전달한다. 클라이언트에서는 JavaScript 그래픽 라이브러리를 통해 전달받은 리소스 사용량 정보를 사용자가 볼 수 있도록 시각화 한다.

```
[redacted]:~$ curl -G "http://localhost:9090/api/v1/query_range" \
> --data-urlencode 'query=rate(container_cpu_usage_seconds_total{id="/"}[5m])' \
> --data-urlencode 'start=1723538149' \
> --data-urlencode 'end=1723538449' \
> --data-urlencode 'step=30'
{"status":"success","data":{"resultType":"matrix","result":[{"metric":{"beta_kubernetes_io
_arch":"amd64","beta_kubernetes_io_os":"linux","cpu":"total","id":"/","instance":"docker-d
esktop","job":"kubernetes-cadvisor","kubernetes_io_arch":"amd64","kubernetes_io_hostname":
"docker-desktop","kubernetes_io_os":"linux"},"values":[[1723538149,"0.1306528822183371"],[
1723538179,"0.12246271469203339"],[1723538209,"0.1332194729525883"],[1723538239,"0.1489891
7020595695"],[1723538269,"0.16609629841822937"],[1723538299,"0.18335235031931368"],[172353
8329,"0.2011196827690204"],[1723538359,"0.2449106739420554"],[1723538389,"0.27121756750491
05"],[1723538419,"0.2856658841752"],[1723538449,"0.2174271985294371"]]}]}[redacted]
[redacted]:~$
```

그림 27. Prometheus API 실행 예시

그림 27 은 Prometheus API 를 사용하여 리소스 사용량 정보를 조회한 결과이다. 파라미터를 통해 쿼리와 조회 기간, 값 사이의 시간 간격을 입력하여 결과값을 얻는다. 결과값으로 컨테이너 이름과 리소스 종류 등이 나타나며, 시간과 리소스 값 쌍으로 데이터가 반환됨을 확인할 수 있다.

5) 유저 애플리케이션

사용자는 역할별 권한이 다르기 때문에 유저를 구분해야 하는데 마이크로소프트 아키텍처 특성 상 서비스의 추가 및 삭제하는데 불편함이 없어야 한다. 이와 같은 이유로 JWT 토큰을 이용하여 유저를 구분하며 접근 가능한 서비스를 분리한다.

공연별로 네임스페이스가 형성된다. 네임스페이스 내부에는 마이크로서비스 아키텍처를 따르기 위해 각 서비스들은 별도의 Pod 에서 배포된다. 이와 같은 환경에서 배포된 서비스간의 통신은 데이터 베이스의 무결성과 안정성을 유지하는 것이 어렵다. 이를 해결하기 위해, Istio 가 제공하는 Service Mesh 를 활용했다.

그림 28 은 kubectl describe pod 명령어를 통해 나타난 배포중인 Pod 들의 명세이다.

그림 28. Sidecar 패턴에 기반한 Proxy 주입

그림 28 은 Sidecar 패턴에 기반한 Istio-Proxy 주입을 보인다. 첫번째 Microservice 명세 박스는 판매자의 요구사항을 반영하여 배포중인 서비스를 뜻한다. 두번째 Istio-Proxy 명세 박스는 현재 배포중인 서비스에 Sidecar 패턴에 따라 Istio-Proxy 가 주입 되어있음을 보인다. 네임스페이스 별로 Pod 에 라벨을 설정하여 Istio-Proxy 주입하였다. 각 Istio-Proxy 는 Istio 의 Pilot 로부터 Service Discovery 를 제공받아 다른 인스턴스 정보를 기반으로 로드 밸런싱 된다.

6) 마이크로서비스 개발 내용 및 결과

① 컨테이너 풀 관리

컨테이너 풀 관리 기능은 마이크로서비스 개발자가 컨테이너를 관리하고 마이크로서비스를 배포하는 기반 기능이다. 관리자는 본 시스템에서 사용하기 위한 마이크로서비스를 등록 및 조회하는 기능을 수행할 수 있다. 마이크로서비스의 등록은 깃허브 주소 입력 받아 레포지토리를 다운로드하고 개발 내용물을 도커 이미지로 빌드하고 도커 허브에 업로드 하는 동작을 수행한다. 그림 29 는 현재 과제 수행을 위해 개발한 프로젝트 일부를 컨테이너 풀 관리 서버를 통해 마이크로서비스를 이미지화 하여 최종적으로 도커 허브에 업로드 된 것을 확인할 수 있다.

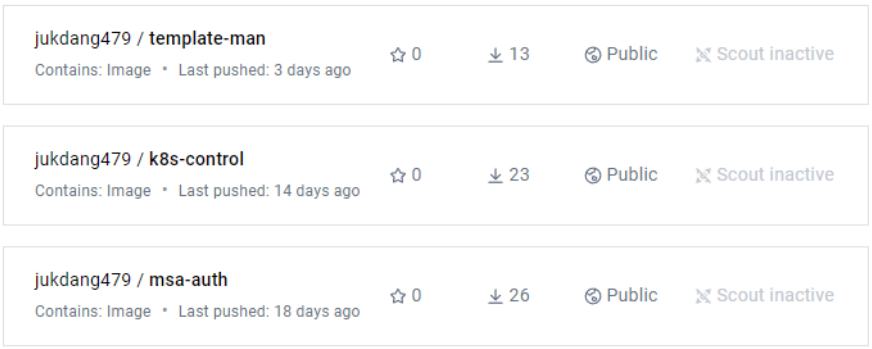

그림 29. 도커 허브에 업로드 된 이미지 목록

② 템플릿 관리

템플릿 관리 서버는 개발자가 공연 배포를 위해 필요한 마이크로서비스들을 하나로 묶는 역할을 수행한다. 컨테이너 풀 관리 서버를 통해 등록된 이미지 목록을 바탕으로, 여러 개의 이미지를 선택하고, 해당 이미지에서 사용되는 변수 값 등을 입력 받아 쿠버네티스에서 사용하는 리소스인 deployment, service, HTTProute 등에 해당하는 여러 개의 yaml 파일 형식의 문서를 포함한 하나의 템플릿을 생성한다. 생성된 템플릿은 공연 배포에 있어서 기본적인 양식의 역할로써, 약간의 사용자 요구사항만으로 쿠버네티스 상에 공연 배포가 가능하다.

③ 시스템 배포 관리

판매자는 시스템 배포 관리 서버를 통해 공연 마이크로서비스 배포에 관련된 CRUD 작업을 수행할 수 있다.

공연에 대한 배포는 템플릿 관리 서버에서 생성한 템플릿을 사용해 이루어진다. 사용자 입력을 통해 템플릿 파일에 사용자의 요구사항 내용을 반영하고, 템플릿의 부족한 부분을 채워 템플릿을 완성한다. 완성된 템플릿을 쿠버네티스 API 서버로 전달하여 시스템이 배포되도록 구성하였다.

배포 관리 서버에서는 자동 배포가 이루어질 때 해당 공연에 대한 컨테이너만으로 구성되는 namespace 를 생성한다. 해당 namespace 에 RBAC 기반의 접근제어 방식을 바탕으로 access token 을 발급하여 해당 token 을 가진 사용자의 권한을 제어하는 방식으로 구현하였다. 생성된 사용자별 token 의 관리는 배포 관리 서버에서 수행한다. 따라서 사용자는 개별적인 token 관리 없이 사용자 인증 서버를 통한 로그인 기능만으로 자신이 배포한 공연들에 대한 접근이 가능하다.

또한 판매자는 자신이 배포한 공연의 목록 조회, 배포된 컨테이너들의 정보 조회 등 배포와 관련된 정보들을 조회할 수 있고, 배포한 공연을 삭제할 수 있다.

④ 사용자 인증

모든 서버가 유저 권한에 따라 접근 여부가 다르다. 마이크로서비스 아키텍처에서 사용자인증은 서비스가 확대 및 축소하더라도 추가적인 리소스 사용 없이 이루어져야 한다. 그렇기 때문에 사용자 인증에는 JWT 토큰이 사용된다. 모든 서버에서 JWT 토큰 비밀키를 공유하면 JWT 토큰만으로 각 서버에서 자체적으로 사용자 토큰 검증이 가능하다.

유저가 로그인할 때 사용자 인증서버에서 클라이언트에게 액세스 토큰을 발급하고 리프레시 토큰은 데이터베이스에 저장된다. 액세스 토큰을 헤더에 담아 보내 탈취의 위험성을 줄였다. 만약 탈취된다 하더라도 액세스 토큰의 유효기간이 짧기 때문에 이를 보완하고 있다. 사용자가 서비스를 이용 중에 액세스 토큰이 만료된다면 사용자 인증 서버에서 만료된 액세스 토큰으로 데이터베이스에 저장되어 있는 리프레시 토큰과 대조하여 적합한 토큰이라 판단하면 새로운 액세스 토큰을 발급한다.

⑤ 판매자, 구매자 애플리케이션

본 과제에서는 마이크로서비스 아키텍처에 기반하여, 구매자가 이용 가능한 좌석 예매, 티켓 관리 판매자가 이용 가능한 공연 등록, 공연 판매 현황 등 다양한 서비스를 운영하고 있다. 각 서비스는 별개의 배포 단위로 운영되며, 데이터베이스 역시 개별로 배포되어 결합도를 줄였다. 서비스 간 데이터 일관성을 유지하기 위해 Istio 를 활용한 Service Mesh 를 사용하고 있다. Istio 는 Istio-Proxy 를 사이드카로 사용하여 각 Pod 에 배포한다. 이를 통해 서비스 간 트래픽을 효과적으로 관리하고, 관리자의 요청에 따라 라우팅 정책을 정의할 수 있다.

그림 30 은 외부 요청이 내부 Istio Service Mesh 를 통해 어떻게 처리되는지 나타내는 구성도이다.

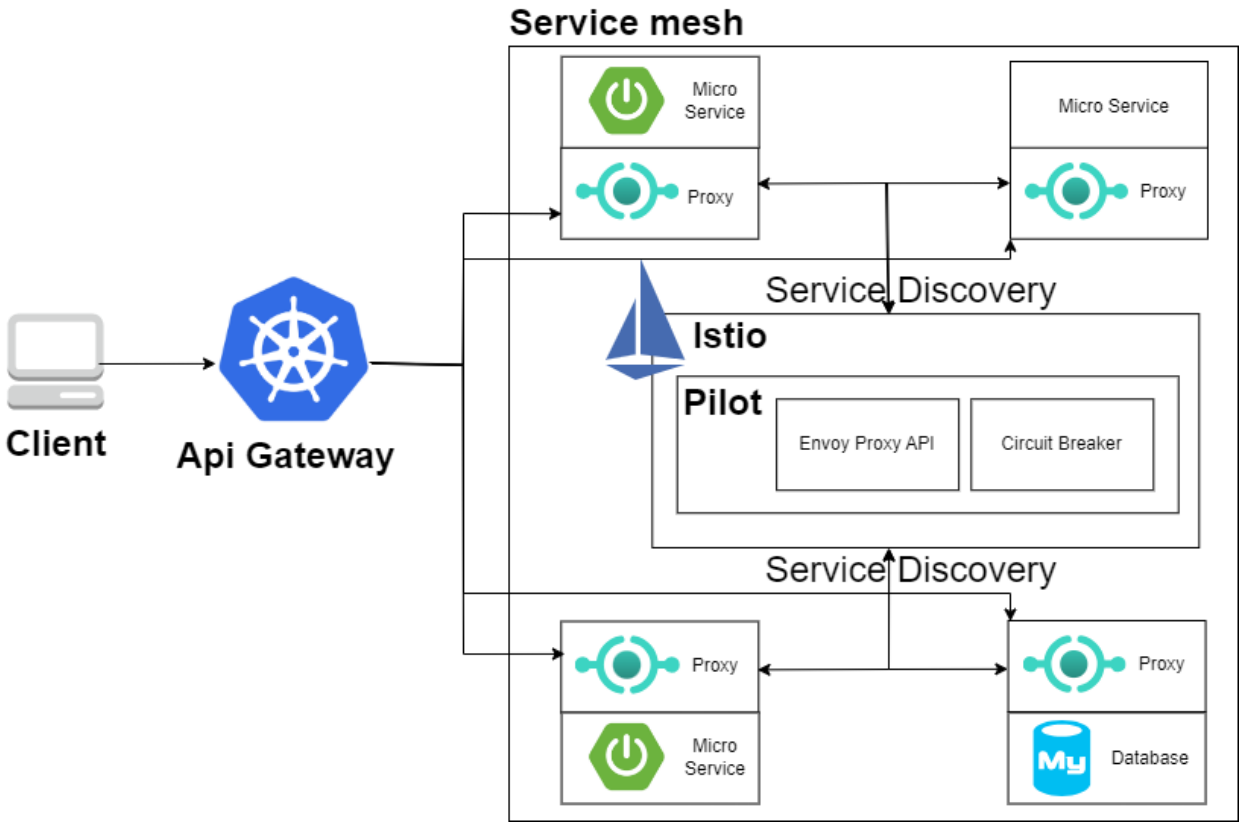

그림 30. Istio Service Mesh 구성도

그림 30 을 보면 사용자의 요청은 API Gateway 를 통해 클러스터 내부로 전달된다. 이때, API Gateway 는 클라이언트의 요청을 배포중인 Pod 옆의 Proxy 로 보낸다. 외부에서의 요청은 API Gateway 를 통해 처리되는 반면 내부에서의 요청은 API Gateway 의 트래픽 분산을 위해 Istio 의 Pilot 컴포넌트에서 정의한 라우팅 정책인 Service Discovery 를 통해 적절한 다른 Proxy 로 라우팅 되어 처리된다.

```java
@RestController
class BuyTicketController {

    1usage
    private final RestTemplate restTemplate = new RestTemplate();
    1usage
    private static final String PROVIDER_URL = "http://client-service/event";

    @GetMapping("/buyticket/event")
    public String buyEventTicket(@RequestParam String showName) {
        String url = UriComponentsBuilder.fromHttpUrl(PROVIDER_URL)
                .queryParam( name: "showName", showName)
                .toUriString();
        return restTemplate.getForObject(url, String.class);
    }
```

그림 31. 판매자 예시 코드

```java
class EventController {

    @GetMapping("/event")
    public String event(@RequestParam String showName) { return showName + " 공연의 티켓 구입완료"; }
}
```

그림 32. 구매자 예시 코드

그림 31, 그림 32 은 티켓 판매자 측면과 구매자 측면의 실험을 위한 코드 일부를 나타내었다. 그림의 코드를 배포하여 동작시킨 결과물이 그림 33 과 같다.

```
root@master-k8s:~# kubectl get pods -n istio-system
NAME                                READY   STATUS    RESTARTS   AGE
istio-egressgateway-88b9b6754-r6b2c     1/1     Running   0          3d12h
istio-ingressgateway-6469f77458-nwbrr   1/1     Running   0          3d12h
istiod-6d6b848688-h8g7l                 1/1     Running   0          3d12h
root@master-k8s:~# kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
client-6f5d8ccd9f-bzvb6           2/2     Running   0          13h
provider-644d5b64b5-f7pgp         2/2     Running   0          13h
test-test-nginx-86b578ff87-xqnh5  2/2     Running   0          14h
root@master-k8s:~# kubectl get service
NAME               TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
client-service     LoadBalancer   10.100.245.212   10.178.0.22   80:30108/TCP   13h
kubernetes         ClusterIP      10.96.0.1        <none>        443/TCP        5d20h
provider-service   LoadBalancer   10.109.234.56    10.178.0.23   80:32732/TCP   13h
test-test-nginx    LoadBalancer   10.110.131.228   10.178.0.20   80:30444/TCP   5d19h
root@master-k8s:~# curl http://10.178.0.23/buyticket/event?showName=event1
event1 공연의 티켓 구입완료root@master-k8s:~#
```

Istio 환경 구축

현재 배포중인 마이크로 서비스

배포중인 마이크로 서비스 명세

Service Mesh 환경의 네트워크 통신

그림 33. Istio Service Mesh 를 활용한 Pod 간 통신

그림 31 과 32 는 그림 33 의 Service Mesh 환경에서의 네트워크 통신에 사용되는 예제 코드이다. 판매자 애플리케이션에서 공연이름을 파라미터로 한 입력을 요청을 받는다. 구매자 애플리케이션에서 공연이름에 해당하는 공연이 구입 완료되었다는 응답을 보인다.

그림 33 의 kubectl get pods -n isito-system 을 통해 Istio Service Mesh 구축에 필요한 Istio 가 구축되어 있음을 보인다. Kubectl get pods 의 Ready 2/2 를 보아 현재 배포중인 마이크로 서비스가 Sidecar 패턴에 따라 배포되어 있음을 보인다. Kubectl get service 를 통해 마이크로서비스의 클러스터 IP, 외부 IP 를 확인할 수 있다. curl http://10.178.0.23/buyticket/event?shoName=event1 의 결과를 통해 판매자와 구매자 애플리케이션에서 event1 공연이 판매 및 결제 완료가 됨을 보인다.

⑥ 클라이언트

클라이언트는 마이크로프론트엔드 아키텍처를 적용하기 위해 webpack 의 Module Federation 기능을 이용하여 각 애플리케이션을 기능에 따라 공연 배포(Deployment), 공연 모니터링(Monitoring), 사용자 인증(Authentication), 공연 예매(Ticketing)로 분리하였다. 분리한 각 애플리케이션은 호스트(Host) 애플리케이션에서 통합한다.

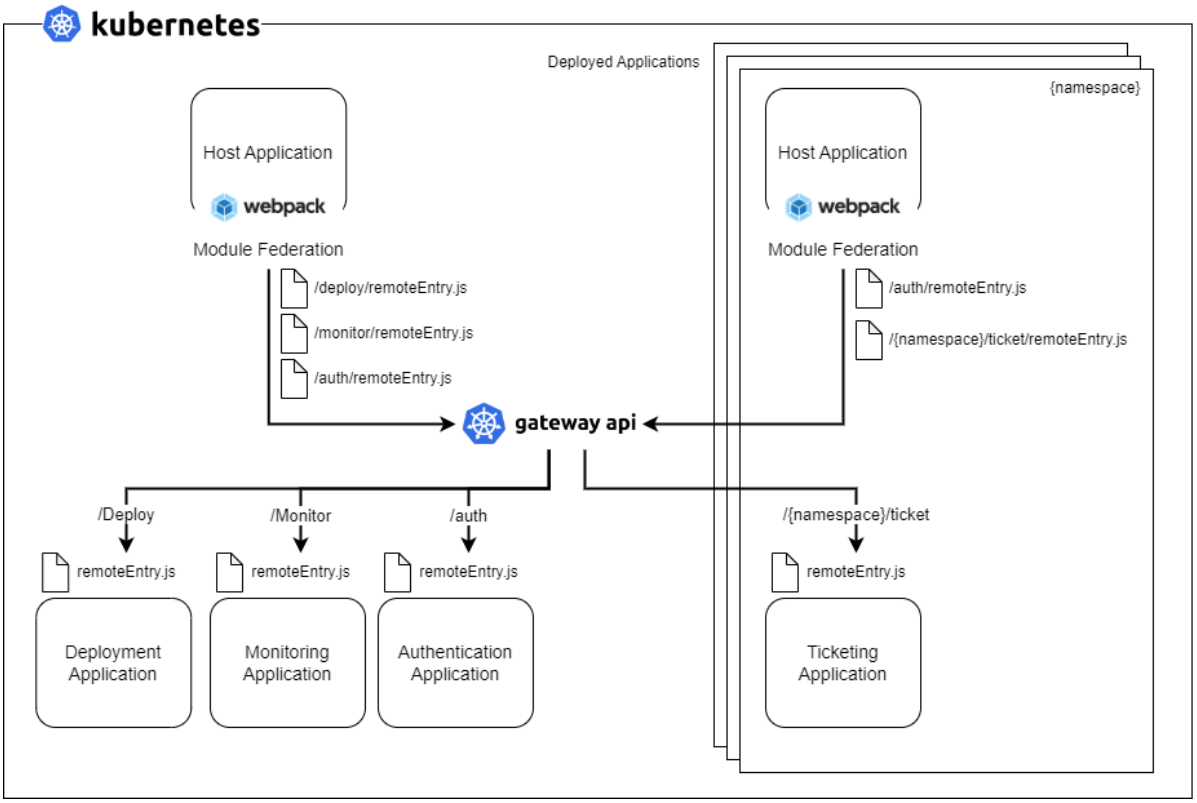

그림 34. 클라이언트 구성도

그림 34 는 독립적으로 개발된 각 애플리케이션이 통합되는 구조를 나타낸 클라이언트 구성도이다. 클라이언트 구성도에서 쿠버네티스에 배포한 각 애플리케이션은 모듈들을 webpack 설정을 통해 expose 하여 외부로 노출시킨다. 외부로 노출된 모듈들의 정보들은 각 애플리케이션의 remoteEntry.js 에 저장되며, 호스트 애플리케이션에서 remoteEntry.js 의 경로로 접근하여 해당 모듈을 가져올 수 있다. 쿠버네티스에 배포된 각 애플리케이션들은 gateway api 를 통해 엔드 포인트가 결정되고, 호스트 애플리케이션에서 해당 경로를 통해 모듈들을 가져와 사용한다. 클라이언트 구성도에서 공연 배포, 공연 모니터링, 사용자 인증 기능을 가져와 사용하는 호스트 애플리케이션과, 공연 배포를 통해 배포되어 공연 예매 기능, 사용자 인증 기능을 가져와 사용하는 호스트 애플리케이션을 확인할 수 있다.

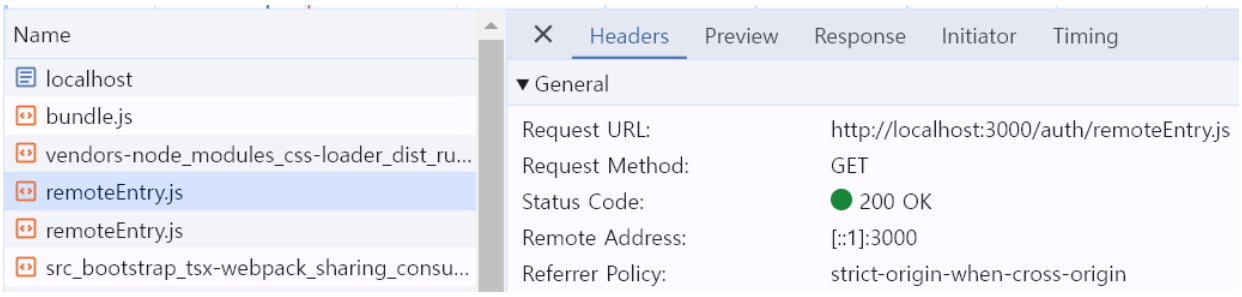

그림 35. remoteEntry.js 요청 결과

그림 35 에서 호스트 애플리케이션에서 사용자 인증 애플리케이션의 모듈을 가져오기 위해 '/auth' 경로를 통해 remoteEntry.js 파일을 요청하는 HTTP Request 의 결과를 확인할 수 있다.

그림 36. 호스트 애플리케이션 구현 결과

그림 36 은 호스트 애플리케이션의 공연 관리 페이지를 구현한 결과이다. 그림 영역에서 ①은 사용자 인증 애플리케이션의 사용자 인증 모듈이고, ②는 공연 배포 애플리케이션의 공연 배포 모듈이고, ③은 공연 모니터링 애플리케이션의 공연 모니터링 모듈이다.

I. 호스트 애플리케이션은 사용자 인증 모듈을 사용자 인증 애플리케이션의 remoteEntry.js 에서 가져와 사용한다. 사용자 인증 모듈은 시스템 접근 권한 요청, 로그아웃 요청, 사용자 토큰 재발급 요청 등의 기능을 수행한다.
II. 호스트 애플리케이션은 공연 배포 모듈을 공연 배포 애플리케이션의 remoteEntry.js 에서 가져와 사용한다. 공연 배포 모듈은 공연 배포 요청, 배포한 공연 목록 조회 요청 등의 기능을 수행한다.
III. 호스트 애플리케이션은 공연 모니터링 모듈을 공연 모니터링 애플리케이션의 remoteEntry.js 에서 가져와 사용한다. 공연 모니터링 모듈은 현재 예매된 좌석 수 등 예매 현황 정보를 조회 요청하는 기능을 수행한다.

마이크로프론트엔드 아키텍처를 적용하여 각 애플리케이션을 독립적으로 개발 및 배포하면 한 애플리케이션이 다른 애플리케이션에 의존하지 않아 오류가 발생하더라도 오류가 발생한 애플리케이션 외 다른 애플리케이션은 정상 동작한다는 장점과 빌드 및 배포 시 자원을 절약할 수 있다는 장점이 있다.

6. 향후 진행 계획

1) 공연 템플릿 구성

개발한 공연 마이크로서비스를 묶어 실제 서비스를 운영할 수 있을 수준의 템플릿을 구성한다. 구성한 템플릿을 통해 공연 배포가 잘 이루어지고, 사용자가 공연에 대한 정보 확인 및 예매가 가능한지 테스트를 진행하고자 한다.

2) 시스템 배포 환경 관리 기능 개발

시스템 배포 환경 관리 클래스 다이어 그램에 해당하는 기능을 개발한다. 쿠버네티스 API 를 이용하여 배포된 마이크로서비스의 배포 환경 정보를 조회 및 관리하고, 마이크로서비스의 리소스 확장 및 축소 기능을 수행하는 서버 및 UI 를 개발한다.

3) 공연 마이크로서비스 추가 개발

공연 배포 시 기본적으로 필요한 마이크로서비스로 템플릿이 구성된다. 필수적인 마이크로서비스 이외에 공연 관련 물품 구매와 같이 부가적인 마이크로서비스를 개발하고자 한다. 템플릿을 이용하여 기존 공연 서비스에 쉽게 추가 및 제거가 가능하게 하여 시스템의 확장성을 강조하고자 한다.

4) 마이크로서비스 배포 성능 평가

마이크로서비스 배포 시나리오를 작성하여 마이크로서비스 배포 시 소요 시간, 배포된 마이크로서비스 간 연결 안정성 등을 평가한다.

7. 참고 문헌

[1] 김대호, 박준석, 염근혁. (2018). 모놀리식 애플리케이션의 UML 설계 자료에 기반한 마이크로서비스 구성 방법. 한국차세대컴퓨팅학회 논문지, 14(5), 7-18.

[2] 송규원, 정진호, 정수민, 박준석, 염근혁. (2022). 명세와 BCE(Boundary-Control-Entity) 패턴을 활용한 컨테이너 기반의 마이크로서비스 배포 기법. 2022 한국차세대컴퓨팅학회 춘계학술대회, 417-420.