3. Design

Busk Spot

Student No	22112014
Name	김연웅
E-Mail	skfafa1234@naver.com

[Revision history]

Revision date	Version #	Description	Author
2025/03/27	1.00	계획 수립	
2025/05/11	1.01	초기 기능 구현	
2025/06/05	1.02	기능 추가	

= Contents =

1. Introduction

1) Summary

현대 사회에서 예술과 문화는 점점 더 대중과 가까워지고 있으며, 그 중심에는 거리에서 자유롭게 이루어지는 버스킹(Busking) 문화가 있다. 음악, 퍼포먼스, 댄스, 마술 등 다양한 장르의 예술가들이 사람들과 직접 소통하며 자신의 재능을 펼칠 수 있는 장으로서, 버스킹은 도시의 문화적 다양성을 풍요롭게 하고 있다. 그러나 이러한 긍정적인 측면에도 불구하고, 실제로 거리 공연을 준비하거나 참여하려는 아티스트와 관객 모두에게는 적지 않은 불편함이 존재한다.

예를 들어, 버스킹을 진행하기 위해서는 각 지역별 규정이나 허가 요건을 사전에 알아야 하며, 공연 가능한 장소와 일정을 확보하는 것도 쉽지 않다. 또한 공연 장비의 대여나 다른 공연자와의 협업, 관객과의 실시간 소통 등 다양한 요소들이 유기적으로 작동해야 함에도 이를 지원하는 통합 플랫폼은 매우 제한적이다. 이러한 문제를 해결하고자 본 프로젝트에서는 Busk Spot이라는 웹 기반 시스템을 기획하게 되었다.

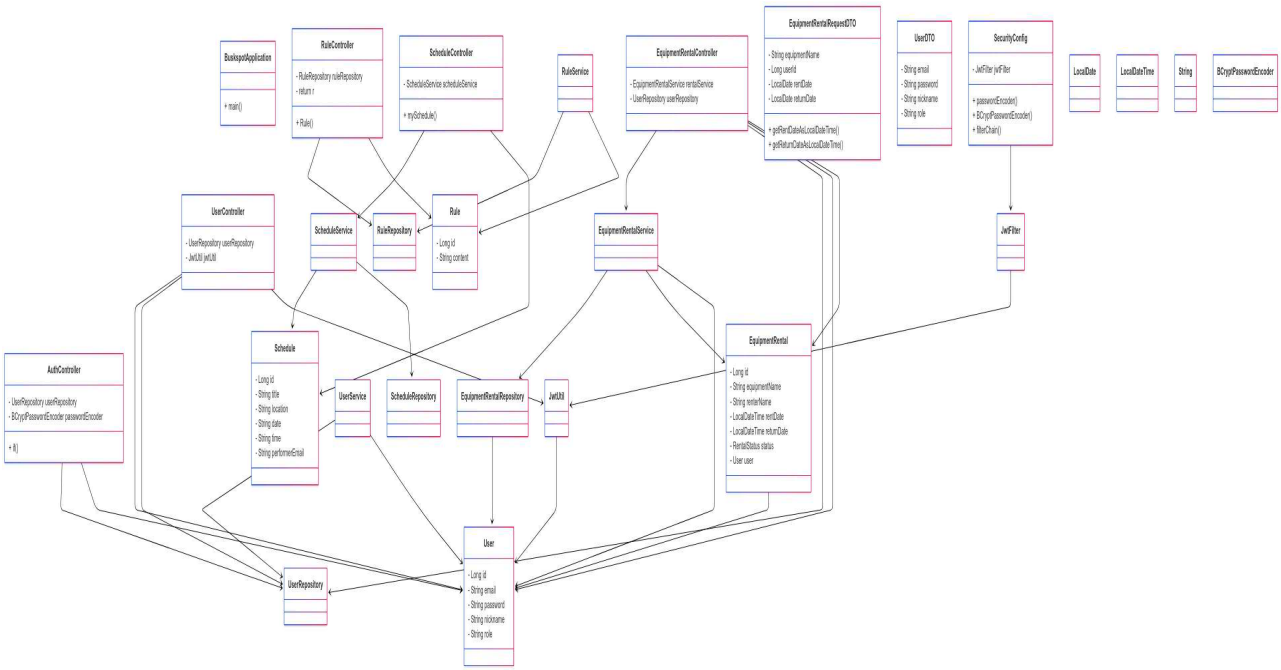
Busk Spot은 거리 공연자(버스커)와 관객이 쉽고 빠르게 공연 정보를 확인하고, 직접 공연 일정을 등록하거나 관람 계획을 세울 수 있도록 돕는 플랫폼이다. 또한 지역별 공연 규정, 공연 장비 대여, 아티스트 등록 및 프로필 관리 기능을 포함하여, 버스킹 문화 활성화에 실질적인 도움을 주는 방향으로 설계되었다.

2) Business Goals

Busk Spot 시스템은 버스킹 문화의 진입장벽을 낮추고, 공연자와 관람객 모두에게 유용한 서비스를 제공하는 것을 목표로 한다. 사용자는 본 시스템을 통해 실시간으로 지역별 공연 일정을 조회할 수 있고, 공연자는 자신의 공연 정보를 간편하게 등록하여 홍보 효과를 높일 수 있다. 나아가 공연 장비가 필요한 사용자들은 시스템 내 대여 기능을 통해 타 공연자들과 자원을 공유할 수 있으며, 공연 허가 관련 법령 및 규정은 관리자에 의해 정기적으로 업데이트되어 제공된다.

또한 단순히 정보 제공에 그치지 않고, 공연자와 관객 간 커뮤니케이션을 증진하는 기능도 포함할 예정이다. 예를 들어, 사용자는 공연 후 리뷰를 남기거나 아티스트의 다른 공연 정보를 팔로우할 수 있으며, 장기적으로는 팬 커뮤니티 기능도 계획되어 있다. 이러한 과정을 통해 Busk Spot은 단순한 공연 정보 사이트를 넘어서, 버스킹 생태계를 연결하는 허브 역할을 하게 될 것이다.

2. Class diagram



Class Name	설명(Explanation)
BuskspotApplication	□ 메서드: - main(): 애플리케이션을 시작하는 메인 진입점으로, SpringApplication.run을 호출합니다.
SecurityConfig	□ 필드: - JwtFilter jwtFilter □ 생성자: - SecurityConfig(JwtFilter jwtFilter) 생성자입니다. □ 메서드: - passwordEncoder(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다. - BCryptPasswordEncoder(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다. - filterChain(): 필터 체인에서 요청의 토큰을 확인하고 인증 여부를 판단합니다.
AuthController	□ 필드: - UserRepository userRepository - BCryptPasswordEncoder passwordEncoder □ 메서드: - if(): 조건문 내에서 인증 또는 상태를 분기 처리하는 메서드 흐름입니다.
EquipmentRentalController	□ 필드:

	- EquipmentRentalService rentalService - UserRepository userRepository □ 생성자: - EquipmentRentalController(EquipmentRentalService rentalService, UserRepository userRepository) 생성자입니다.
RuleController	□ 필드: - RuleRepository ruleRepository - return r □ 생성자: - RuleController(RuleRepository ruleRepository) 생성자입니다. □ 메서드: - Rule(): 규칙 데이터를 반환하거나 특정 규칙을 처리하는 API 엔드포인트입니다.
ScheduleController	□ 필드: - ScheduleService scheduleService □ 생성자: - ScheduleController(ScheduleService scheduleService) 생성자입니다. □ 메서드: - mySchedule(): 현재 로그인된 사용자의 스케줄 정보를 조회합니다.
UserController	□ 필드: - UserRepository userRepository - JwtUtil jwtUtil □ 생성자: - UserController(UserRepository userRepository, JwtUtil jwtUtil) 생성자입니다.
EquipmentRental	□ 필드: - Long id - String equipmentName - String renterName - LocalDateTime rentDate - LocalDateTime returnDate - RentalStatus status - User user
Schedule	□ 필드: - Long id - String title - String location - String date - String time - String performerEmail
EquipmentRentalRequestDTO	□ 필드: - String equipmentName - Long userId - LocalDate rentDate

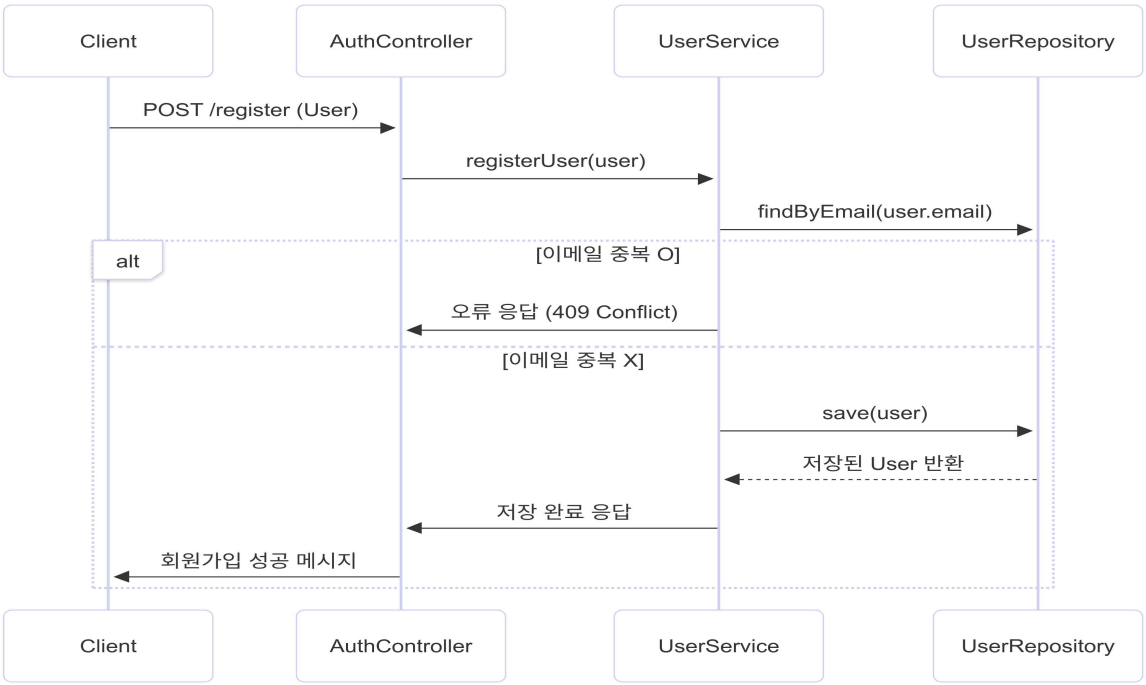
	- LocalDate returnDate □ 메서드: - getRentDateAsLocalDateTime(): 해당 정보를 검색하여 반환합니다. 대부분 read-only 로직입니다. - getReturnDateAsLocalDateTime(): 해당 정보를 검색하여 반환합니다. 대부분 read-only 로직입니다.
UserDTO	□ 필드: - String email - String password - String nickname - String role
Rule	□ 필드: - Long id - String content
User	□ 필드: - Long id - String email - String password - String nickname - String role
EquipmentRentalRepository	□ 메서드: - findByUserId(): 조건에 맞는 엔티티를 조회하는 데이터 접근 메서드입니다. - findByStatus(): 조건에 맞는 엔티티를 조회하는 데이터 접근 메서드입니다.
RuleRepository	□ 메서드: - findTopByOrderByIdDesc(): 조건에 맞는 엔티티를 조회하는 데이터 접근 메서드입니다.
ScheduleRepository	□ 메서드: - findByPerformerEmail(): 조건에 맞는 엔티티를 조회하는 데이터 접근 메서드입니다.
UserRepository	□ 메서드: - findByEmail(): 조건에 맞는 엔티티를 조회하는 데이터 접근 메서드입니다.
JwtFilter	□ 필드: - JwtUtil jwtUtil □ 생성자: - JwtFilter(JwtUtil jwtUtil) 생성자입니다. □ 메서드: - doFilter(): 필터 체인에서 요청의 토큰을 확인하고 인증 여부를 판단합니다. - if(): 조건문 내에서 인증 또는 상태를 분기 처리하는 메서드 흐름입니다. - if(): 조건문 내에서 인증 또는 상태를 분기 처리하는 메서드 흐름입니다. - if(): 조건문 내에서 인증 또는 상태를 분기 처리하는 메서드 흐름입니다. - isAdminByFakeToken(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다. - isPublicPath(): 일반적인 기능 수행

	메서드로, 클래스 내 주요 로직을 처리합니다.
JwtUtil	□ 필드: - return true - return false □ 메서드: - createToken(): 새로운 리소스를 생성하는 동작을 수행하며, 종종 save와 연결됩니다. - getRole(): 해당 정보를 검색하여 반환합니다. 대부분 read-only 로직입니다. - validate(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다. - getUsername(): 해당 정보를 검색하여 반환합니다. 대부분 read-only 로직입니다.
EquipmentRentalService	□ 필드: - EquipmentRentalRepository rentalRepository □ 생성자: - EquipmentRentalService(EquipmentRentalRepository rentalRepository) 생성자입니다. □ 메서드: - rentEquipment(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다. - getRentalsByUserId(): 해당 정보를 검색하여 반환합니다. 대부분 read-only 로직입니다. - getPendingRentals(): 해당 정보를 검색하여 반환합니다. 대부분 read-only 로직입니다. - approveRental(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다. - IllegalArgumentException(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다. - rejectRental(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다. - IllegalArgumentException(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다.
RuleService	□ 필드: - RuleRepository ruleRepository □ 생성자: - RuleService(RuleRepository ruleRepository) 생성자입니다. □ 메서드: - getLatestRule(): 해당 정보를 검색하여 반환합니다. 대부분 read-only 로직입니다. - saveRule(): 엔티티를 데이터베이스에 저장하는 JPA save 메서드를 호출합니다.
ScheduleService	□ 필드: - ScheduleRepository scheduleRepository

	□ 생성자: - ScheduleService(ScheduleRepository scheduleRepository) 생성자입니다. □ 메서드: - createSchedule(): 새로운 리소스를 생성하는 동작을 수행하며, 종종 save와 연결됩니다. - getSchedulesByEmail(): 해당 정보를 검색하여 반환합니다. 대부분 read-only 로직입니다.
UserService	□ 필드: - UserRepository userRepository - BCryptPasswordEncoder passwordEncoder - return userOpt □ 메서드: - registerUser(): 사용자의 회원가입을 처리하고, 정보를 데이터베이스에 저장합니다. - loginUser(): 사용자의 로그인 요청을 처리하며, 이메일과 비밀번호를 기반으로 인증을 수행합니다.
BuskspotApplicationTests	□ 메서드: - contextLoads(): 일반적인 기능 수행 메서드로, 클래스 내 주요 로직을 처리합니다.

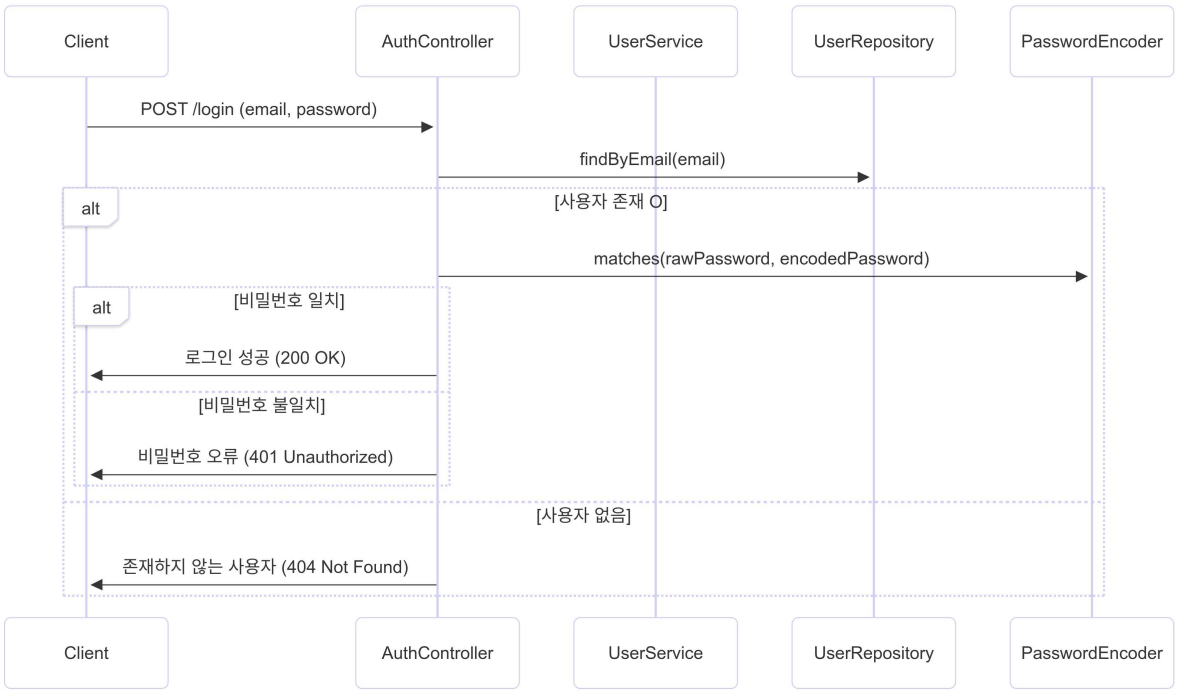
3. Sequence diagram

1) 회원가입



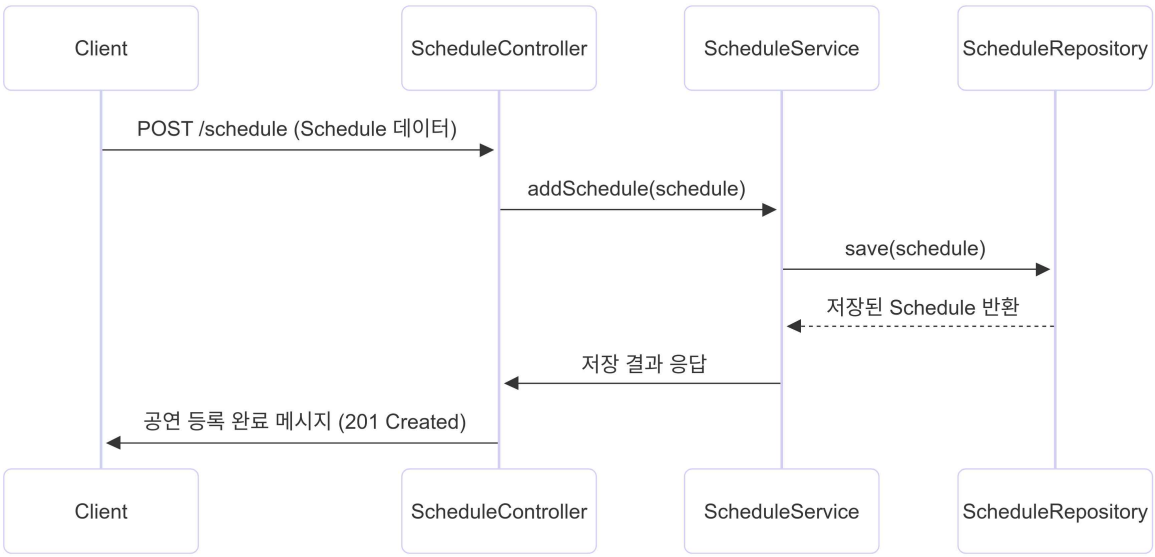
시스템에서 회원가입을 요청하면, 클라이언트는 사용자 정보를 담은 POST 요청을 /register 엔드포인트로 전송한다.
이 요청은 AuthController의 register() 메서드에 의해 처리되며, 사용자 정보는 User 객체 형태로 받아진다.
AuthController는 내부적으로 UserService의 registerUser() 메서드를 호출하여 본격적인 가입 처리를 위임한다.
UserService는 먼저 UserRepository를 이용해 주어진 이메일로 이미 가입된 사용자가 있는지를 검사한다.
이메일이 이미 존재할 경우, 중복 가입을 방지하기 위해 409 Conflict 응답을 반환하며, 클라이언트는 "이미 가입된 이메일입니다"와 같은 메시지를 받는다.
반면, 이메일이 존재하지 않을 경우, 새로운 사용자 정보를 save() 메서드를 통해 DB에 저장하고, 저장된 사용자 객체를 반환한다.
최종적으로 AuthController는 회원가입 성공 메시지를 클라이언트에게 응답으로 보낸다

2) 로그인



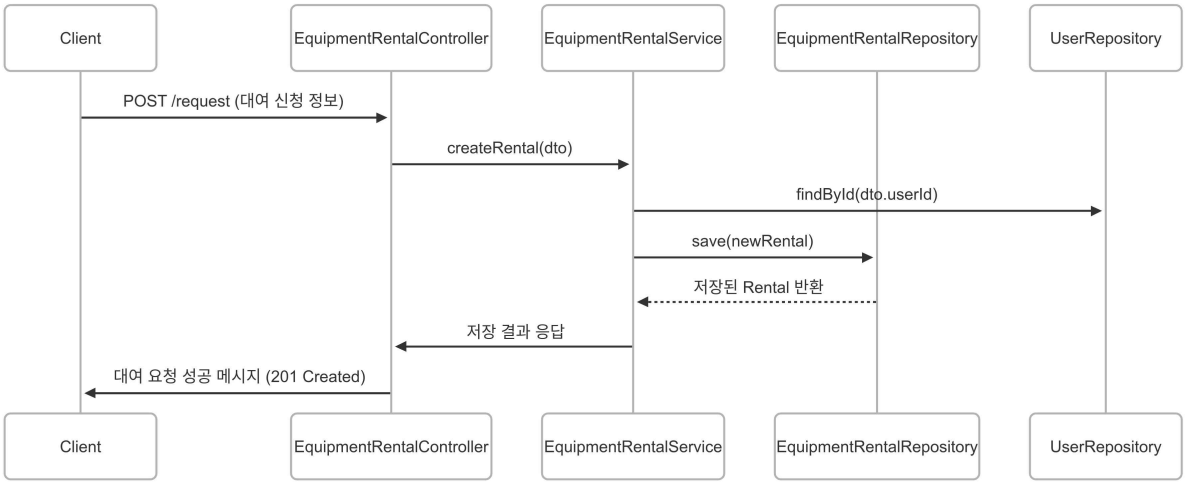
로그인 요청이 발생하면, 클라이언트는 사용자의 이메일과 비밀번호를 포함한 POST 요청을 /login 엔드포인트로 보낸다.
이 요청은 AuthController의 login() 메서드가 처리한다.
먼저, AuthController는 주어진 이메일을 기반으로 UserRepository를 이용해 사용자가 실제로 존재하는지를 확인한다.
해당 이메일로 등록된 사용자가 존재하면, 시스템은 사용자가 입력한 비밀번호를 DB에 저장된 암호화된 비밀번호와 비교하기 위해 PasswordEncoder.matches() 메서드를 호출한다.
비밀번호가 일치하면 로그인 성공으로 간주되어 200 OK 응답과 함께 사용자 정보를 반환하거나 토큰 등을 제공한다.
반대로 비밀번호가 일치하지 않을 경우, 401 Unauthorized 응답과 함께 비밀번호 오류 메시지를 반환한다.
만약 이메일 자체로 등록된 사용자가 없다면, 404 Not Found 응답을 통해 존재하지 않는 사용자임을 알린다.

3) 버스킹 공연 등록



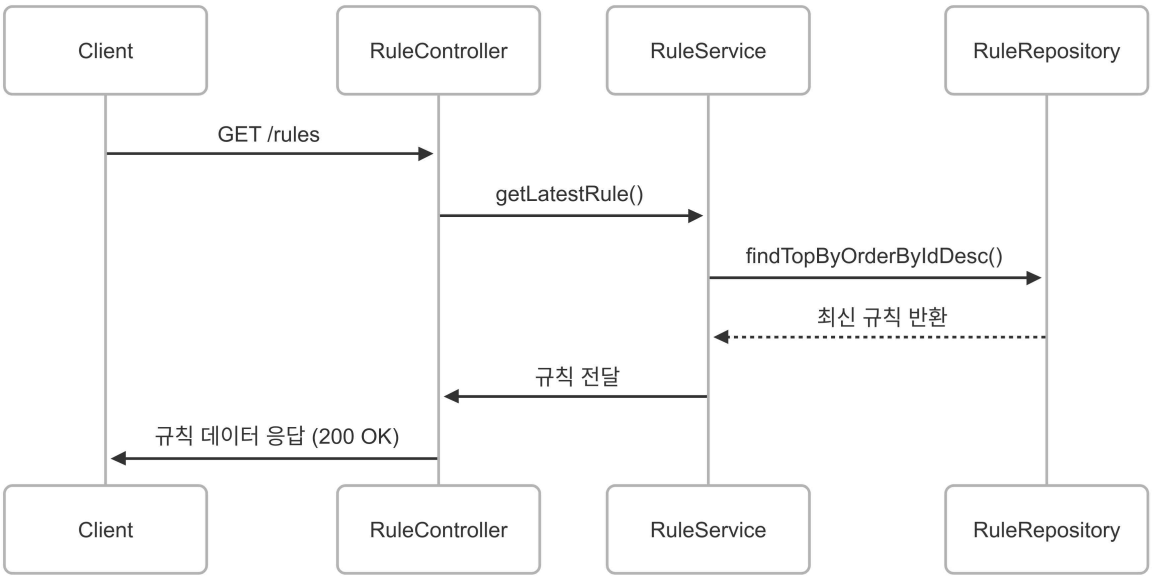
사용자가 버스킹 공연 등록 요청을 하면, 클라이언트는 공연 제목, 위치, 날짜, 시간 등의 정보를 포함하여 POST /schedule 요청을 전송한다.
이 요청은 ScheduleController에 전달되며, 해당 컨트롤러는 공연 등록을 위한 ScheduleService의 addSchedule() 메서드를 호출한다.
ScheduleService는 전달받은 Schedule 객체를 ScheduleRepository의 save() 메서드를 통해 데이터베이스에 저장한다.
정상적으로 저장이 완료되면, DB에서 저장된 공연 정보를 다시 반환하며, 이 정보를 ScheduleController로 전달한다.
컨트롤러는 최종적으로 사용자에게 “공연 등록이 완료되었습니다”와 같은 메시지를 포함한 HTTP 201 Created 응답을 반환한다.

4) 장비 대여



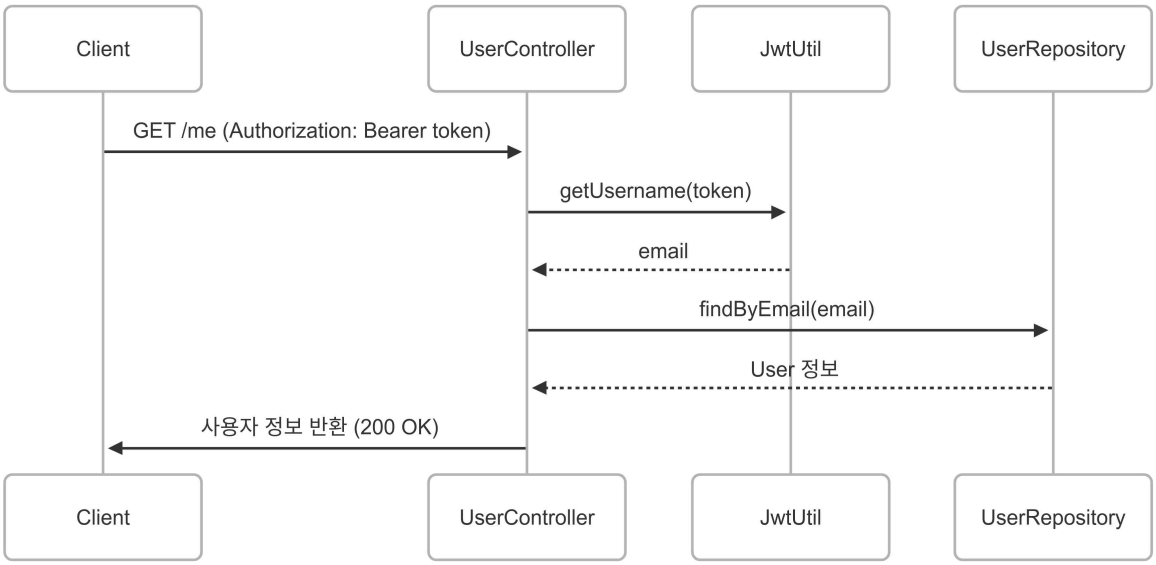
사용자가 장비를 대여하기 위해 대여 요청 정보를 입력한 후 POST /request 엔드포인트로 요청을 전송하면, EquipmentRentalController는 이를 받아 내부적으로 EquipmentRentalService의 createRental() 메서드를 호출한다.
createRental() 메서드는 먼저, 요청 정보에 포함된 사용자 ID를 기반으로 UserRepository.findById()를 호출하여 해당 사용자를 조회한다.
그 후 요청된 장비 이름, 대여 날짜, 반납 날짜 등을 바탕으로 새로운 EquipmentRental 객체를 생성한 뒤 EquipmentRentalRepository.save()를 통해 데이터베이스에 저장한다.
저장이 완료되면 저장된 객체를 컨트롤러에 반환하고, 컨트롤러는 최종적으로 사용자에게 **“장비 대여 요청이 완료되었습니다”**라는 메시지와 함께 201 Created 응답을 반환한다.

4) 규칙 보기



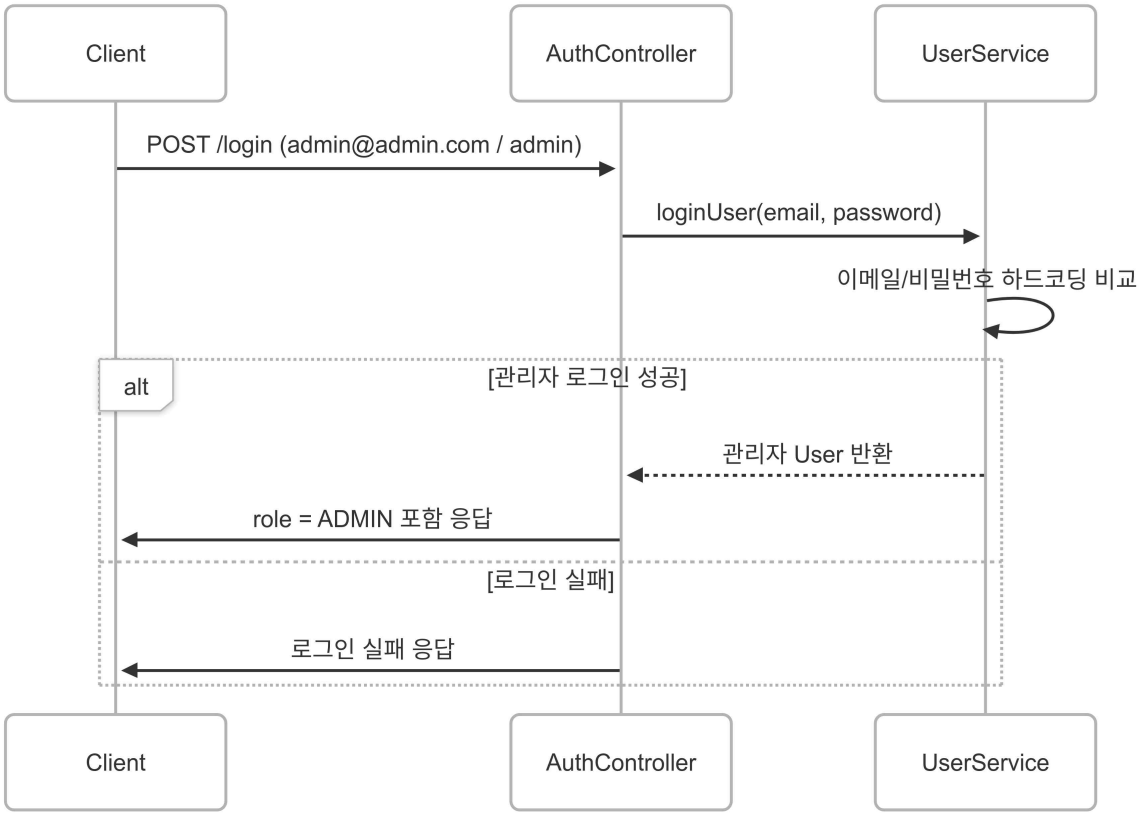
사용자가 규칙을 보기 위해 GET /rules 요청을 전송하면, 이 요청은 RuleController의 getRules() 메서드에 의해 처리된다.
RuleController는 내부적으로 RuleService의 getLatestRule() 메서드를 호출하여 가장 최근의 규칙 정보를 조회한다.
RuleService는 RuleRepository를 통해 규칙 테이블에서 최신 규칙을 찾기 위해 findTopByOrderByIdDesc() 같은 쿼리 메서드를 호출하고, 이 결과로 최신 규칙 하나를 반환받는다.
해당 규칙 객체는 다시 컨트롤러로 전달되며, RuleController는 최종적으로 이 규칙 데이터를 클라이언트에게 JSON 형식으로 응답하며, HTTP 상태 코드 200 OK가 함께 전송된다.

5) 정보 확인



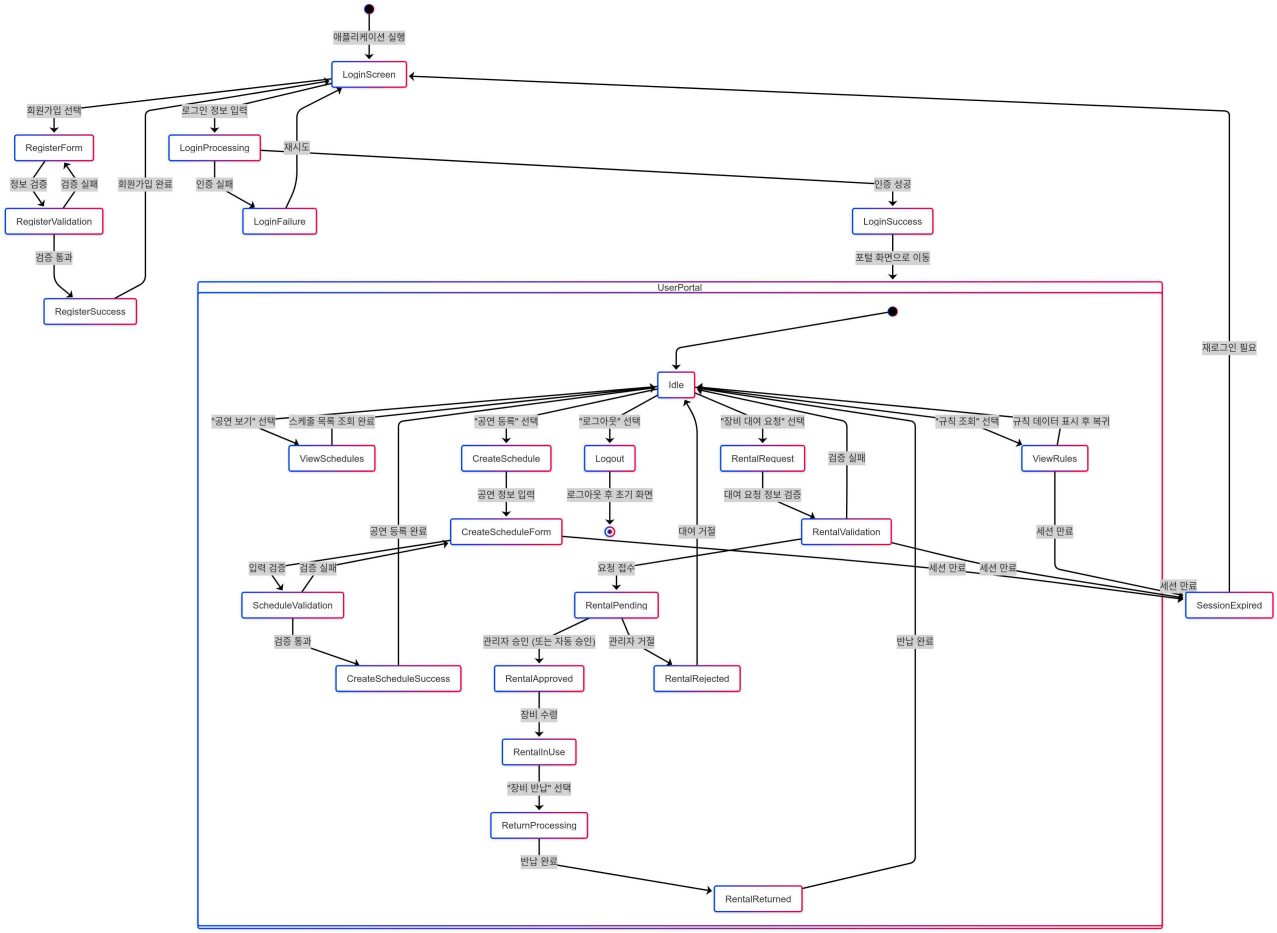
클라이언트가 자신의 정보를 확인하고자 할 때, GET /me 요청을 보낸다.
이 요청에는 JWT 토큰이 포함된 Authorization 헤더가 함께 전송된다.
UserController는 이 토큰을 JwtUtil.getUsername() 메서드에 전달하여,토큰 내부에서 사용자의 이메일 주소를 추출한다.
이메일이 확인되면, 해당 이메일을 기준으로 UserRepository.findByEmail() 메서드를 호출하여 사용자 정보를 조회한다.
정상적으로 사용자가 조회되면, 컨트롤러는 사용자 객체에서 필요한 정보(이메일, 닉네임, 역할 등)를 JSON 형태로 변환하여 클라이언트에게 200 OK 응답과 함께 반환한다.

6) 관리자 기능



관리자 기능은 기본적으로 admin@admin.com 이메일과 admin이라는 비밀번호로 로그인했을 때 동작한다.
사용자가 로그인 요청을 보내면, AuthController는 입력된 이메일과 비밀번호를 검사하기 위해 UserService.loginUser() 메서드를 호출한다.
UserService 내부에는 관리자 이메일과 비밀번호를 하드코딩한 조건문이 존재하며,
해당 조건이 만족되면 관리자 권한을 가진 User 객체를 직접 생성하여 반환한다.
AuthController는 반환된 사용자 객체의 정보를 바탕으로 role: ADMIN을 포함한 응답을 생성해 클라이언트에게 전달한다.
이를 통해 프론트엔드는 사용자가 관리자임을 인지하고, 관리자 전용 UI나 기능을 활성화할 수 있다.

4. State machine diagram



Status	Explanation
LoginScreen	애플리케이션 실행 시 처음 진입하는 로그인 화면 상태입니다. 사용자는 여기서 "로그인" 또는 "회원가입"을 선택할 수 있습니다.
RegisterForm	사용자가 "회원가입"을 선택했을 때 회원가입 입력 폼(Form)이 표시되는 상태입니다.
RegisterValidation	회원가입 폼에서 입력된 데이터를 검증하는 상태입니다. 이메일 중복, 비밀번호 일치 여부 등을 확인하여 검증 실패 시 다시 입력 폼으로 돌아갑니다.
RegisterSuccess	회원가입 검증을 모두 통과하고 실제로 데이터베이스에 사용자를 저장한 후, "회원가입 완료" 메시지를 보여주는 상태입니다. 이후 로그인 화면으로 이동합니다.
LoginProcessing	로그인 폼에서 입력된 아이디·비밀번호를 받아 인증 로직을 처리하는 상태입니다. 사용자 정보가 유효한지 확인하고, 성공/실패

	여부를 결정합니다.
LoginSuccess	로그인 인증이 성공하여 사용자 정보를 모두 로딩한 상태입니다. 이 상태가 되면 포털 화면(UserPortal)으로 전환됩니다.
LoginFailure	로그인 인증에 실패한 상태입니다(아이디 미존재 또는 비밀번호 불일치). 오류 메시지를 보여주고 다시 로그인 화면으로 돌아갑니다.
UserPortal.Idle	로그인 후 메인 포털에 진입하여 대기 중인 상태입니다. 여기서 사용자에게 "공연 보기, 공연 등록, 장비 대여, 규칙 조회, 로그아웃" 등의 옵션이 표시됩니다.
ViewSchedules	포털에서 "공연 보기"를 선택하여 공연 스케줄 목록을 조회하고 결과를 화면에 표시하는 상태입니다. 조회가 끝나면 다시 UserPortal.Idle로 복귀합니다.
CreateScheduleForm	포털에서 "공연 등록"을 선택했을 때, 사용자가 공연 정보(제목·장소·날짜·시간·공연자 이메일)를 입력하는 폼이 열린 상태입니다.
ScheduleValidation	공연 정보 입력 폼에서 제출된 데이터를 검증하는 상태입니다. 필수 항목 누락, 날짜 형식 오류 등을 체크하여 검증 실패 시 다시 입력 폼으로 돌아갑니다.
CreateScheduleSuccess	공연 정보 검증을 통과하고 데이터베이스에 공연 정보를 저장한 상태입니다. 저장이 완료되면 UserPortal.Idle로 복귀합니다.
RentalValidation	포털에서 "장비 대여 요청"을 선택했을 때, 입력된 장비명·대여자 ID·대여일자·반납일자 등이 유효한지 검증하는 상태입니다. 검증 실패 시 UserPortal.Idle로 돌아갑니다.
RentalPending	장비 대여 요청이 검증을 통과하여 서버에 접수된 후, "대기(PENDING)" 상태로 진입한 상태입니다. 이 상태에서 관리자의 승인 또는 자동 승인 로직을 기다립니다.
RentalApproved	장비 대여가 승인된 상태입니다. 대여 승인 후 사용자에게 "장비 대여 준비 완료"를 알리고, 실제 장비 수령(RENTED) 상태로 전이됩니다.
RentalRejected	장비 대여 요청이 거절된 상태입니다. 이유(재고 부족, 권한 문제 등)를 사용자에게 알리고 UserPortal.Idle로 돌아갑니다.
RentalInUse	사용자가 실제로 장비를 수령하여 사용 중인

	상태입니다(PENDING→APPROVED 후 RENTED). 이 상태에서 "장비 반납"을 선택할 때까지 유지됩니다.
ReturnProcessing	사용자가 "장비 반납"을 선택하여 반납 처리 로직(대여 상태 → RETURNED)으로 진입하는 상태입니다.
RentalReturned	장비 반납이 완료된 상태입니다. 장비 반납 후 "장비 반납 완료" 메시지를 보여주고 UserPortal.Idle로 복귀합니다.
ViewRules	포털에서 "규칙 조회"를 선택했을 때, 현재 시스템에 저장된 규칙(예: 규칙 내용, 가이드라인)을 화면에 표시하는 상태입니다. 조회가 끝나면 UserPortal.Idle로 복귀합니다.
Logout	포털에서 "로그아웃"을 선택하여 현재 로그인 세션을 종료하는 상태입니다. 세션을 삭제하고 초기 로그인 화면(LoginScreen)으로 이동합니다.
SessionExpired	사용자가 일정 시간 동안 활동하지 않아 세션이 만료된 상태입니다. UserPortal의 모든 하부 상태(공연 조회, 공연 등록, 장비 대여 등)에서 이 상태로 전이될 수 있으며, 재로그인이 필요함을 알립니다.

5. Implementation requirements

구분	요구사항
Hardware Requirements	- CPU: Intel Core i3 이상 - RAM: 4GB 이상 - 저장 공간: 20GB 이상 HDD 또는 SSD 권장
Software Requirements	- 운영체제: Windows 10 이상 또는 macOS 10.15 이상 - JDK: Java 17 이상 - IDE: IntelliJ IDEA 또는 Eclipse - 빌드 도구: Gradle 또는 Maven - 프레임워크: Spring Boot 3.x
Network Requirements	- 네트워크: 인터넷 연결 필수 (회원가입/로그인 및 장비 대여 시 REST API 요청 필요)
Database Requirements	- DBMS: MySQL 8.0 이상 - 드라이버: JDBC 또는 Spring Data JPA - 테이블: User, Schedule, EquipmentRental, Rule 등 사용
Security Requirements	- 사용자 인증: JWT 기반 인증 - 비밀번호 암호화: BCrypt 사용 - 관리자 구분: Role 기반 권한 분리
기타 (Etc.)	- 배포 환경: AWS EC2 또는 로컬 Tomcat 사용 가능 - 문서화: Swagger 또는 Postman Collection으로 API 문서화

6. Glossary

Terms	Description
JWT	JSON Web Token. 사용자 인증을 위해 클라이언트-서버 간에 사용되는 토큰 방식 인증 수단입니다.
Spring Boot	Java 기반 웹 애플리케이션 개발을 위한 프레임워크로, 빠른 개발과 설정 자동화를 지원합니다.
Entity	데이터베이스 테이블과 매핑되는 Java 클래스. 시스템의 주요 데이터 모델을 정의합니다.
Controller	클라이언트의 요청을 받아 처리하고, 서비스 계층으로 전달하거나 응답을 반환하는 계층입니다.
Service	비즈니스 로직을 처리하는 계층. 데이터 처리 흐름의 중심 역할을 수행합니다.
Repository	DB와 직접 통신하는 계층으로, CRUD 연산을 수행합니다. Spring Data JPA를 주로 사용합니다.
DTO	Data Transfer Object. 데이터 전송을 위한 객체로, Entity와는 구분됩니다.
Schedule	공연 스케줄 정보를 나타내는 도메인 객체. 제목, 날짜, 장소 등의 정보를 포함합니다.
User	회원 정보를 나타내는 객체. 이메일, 비밀번호, 닉네임, 권한(Role) 등의 필드를 포함합니다.
EquipmentRental	장비 대여 정보를 관리하는 엔티티. 대여 날짜, 반납 날짜, 대여자 등의 정보 포함.
RentalStatus	장비 대여의 현재 상태를 나타내는 열거형(Enum). 예: PENDING, APPROVED, REJECTED
BCrypt	비밀번호를 암호화하기 위한 해싱 함수. 보안성을 높이기 위해 사용됩니다.
Swagger	REST API 명세를 문서화하고 시각적으로 테스트할 수 있는 도구입니다.
Authentication	사용자가 시스템에 접근할 수 있는지를 확인하는 절차입니다. 로그인 처리 시 사용됩니다.
Authorization	사용자가 특정 리소스나 기능을 사용할 권한이 있는지 검증하는 절차입니다.

7. References

Spring Security : https://docs.spring.io/spring-security/reference/
Spring 공식 가이드 :
https://spring.io/guides/topicals/spring-security-architecture/
StarUML 사용법 : https://docs.staruml.io/
Mermaid Live Editor : https://mermaid.live