3. Design

자동 매칭 시스템 ZD

Student Number	Name	Email
22112054	김가빈	gabin5556@naver.com

[Revision history]

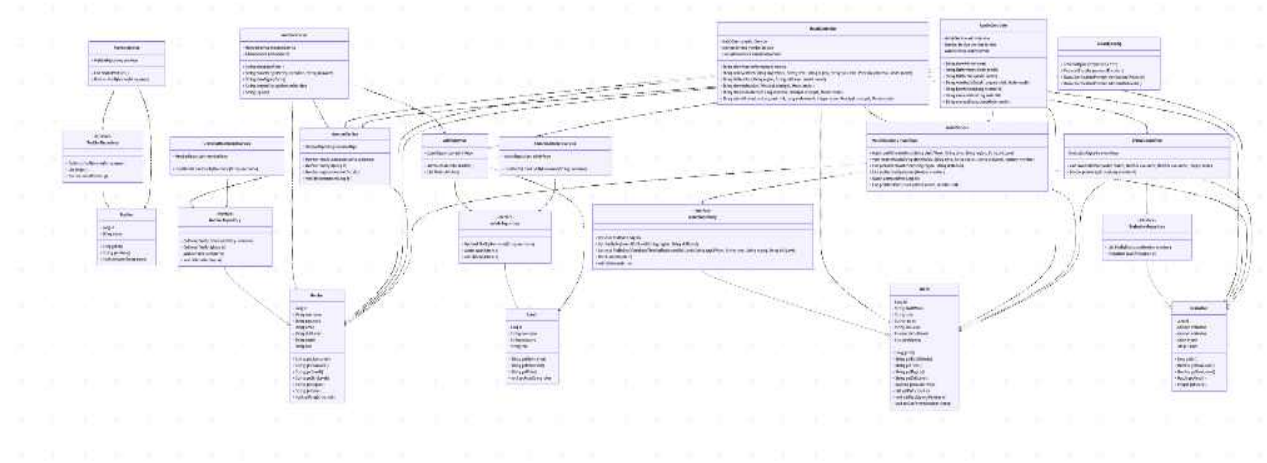
Revision date	Version #	Description	Author
2025/03/27	1.00	초기 버전	
2025/05/08	1.01	기능 수정	
2025/06/04	1.02	기능 추가	

= Contents =

1. Introduction

축구는 전 세계적으로 많은 사람들이 즐기는 대표적인 스포츠이며, 국내에서도 사회인 풋살 리그, 동호회 활동, 교내 체육 활동 등을 통해 널리 사랑받고 있습니다. 하지만 실제 경기를 성사시키기 위해서는 적절한 인원 구성, 경기장 확보, 일정 조율 등 복잡하고 번거로운 절차가 필수적이며, 이로 인해 경기가 자주 취소되거나 참여율이 낮아지는 문제가 빈번하게 발생합니다. 특히, 바쁜 일상 속에서 취미로 축구를 즐기고자 하는 직장인, 대학생들에게 이러한 과정을 일일이 관리하는 것은 큰 부담이 됩니다.
이러한 문제를 해결하고자 개발된 ZD 시스템은 '자동 매칭 기반 축구 경기 관리 플랫폼'으로, 사용자가 입력한 정보(실력, 포지션, 지역, 시간대 등)를 바탕으로 자동으로 팀을 구성하고, 조건에 맞는 경기를 추천하거나 새로 생성하여 예약할 수 있도록 지원합니다. 또한, 경기 후 팀원 간 평가 기능을 통해 사용자 간 신뢰를 형성하고, 비매너 유저를 필터링하는 데 활용함으로써 시스템 전체의 품질과 만족도를 높이는 구조를 갖추고 있습니다.
ZD는 단순한 경기 예약 시스템이 아니라, 포지션 기반 자동 팀 매칭 알고리즘, 경기 성사 조건 자동 판단(인원 12명 도달 시 확정), 평가 기반의 사용자 피드백 루프, 관리자 전용 제어 기능 등을 갖춘 지능형 경기 운영 플랫폼입니다. 유사한 풋살 예약 서비스들이 수동 기반의 인원 모집에 머무는 것과 달리, ZD는 실제 팀 구성과 경기 운영 흐름 전반을 자동화하여 사용자 편의성과 공정성을 동시에 확보하는 것이 특징입니다.
ZD 시스템은 직장인 풋살 동호회, 대학교 운동 소모임, 지역 기반 축구 커뮤니티 등 다양한 사용자층을 주요 대상으로 삼고 있으며, 누구나 손쉽게 접근할 수 있도록 직관적인 사용자 인터페이스를 제공하고, 브라우저 기반 웹 애플리케이션으로 다양한 기기에서 사용 가능하게 설계되었습니다.
기술적인 측면에서, ZD는 Java 17과 Spring Boot 프레임워크를 기반으로 개발되었으며, 사용자 인증 및 권한 관리는 Spring Security, 데이터 저장은 MySQL 혹은 H2 데이터베이스를 사용합니다. 웹 인터페이스는 HTML과 Thymeleaf 템플릿을 기반으로 구성되며, RESTful API를 통해 프론트엔드와 백엔드 간 효율적인 데이터 흐름을 구현하였습니다. 특히, 경기 매칭 알고리즘은 요일, 시간, 지역, 실력, 포지션을 고려한 다단계 필터링 로직을 기반으로 하여 공정한 팀 구성을 보장합니다.
또한 ZD 시스템은 관리자 계정 기능을 통해 전체 회원 관리, 경기 내역 조회 및 강제 조정, 평가 정보 검토 등 운영적인 요소까지도 통합하여, 안정적인 플랫폼 운영이 가능하도록 구성되어 있습니다.
결론적으로 ZD의 비즈니스 목표는 단순한 경기 예약이 아니라, 사용자 편의성, 자동화된 공정한 팀 구성, 피드백 기반 신뢰 시스템, 확장성 있는 아키텍처를 종합적으로 제공함으로써 축구를 좋아하는 사람들이 시간과 공간의 제약 없이 공정하고 즐거운 경기를 보다 쉽고 편리하게 즐길 수 있도록 하는 데에 있습니다. 이는 단순한 시스템 구현을 넘어, 지속 가능한 스포츠 커뮤니티를 구축하는 데에도 기여할 수 있습니다.

2. Class diagram



Class Name	설명(Explanation)
AutomatchApplication	AutomatchApplication Spring Boot 애플리케이션의 진입점(Main 클래스)이다. @SpringBootApplication 어노테이션이 붙어 있으며, main에서 SpringApplication.run(...)을 호출하여 내장 톰캣 서버를 띄우고, 스프링 컨텍스트를 초기화한다.
AdminInitializer	애플리케이션 시작 시 기본 관리자 계정을 초기화하거나 사전 데이터를 세팅하기 위한 컴포넌트 클래스이다. 보통 CommandLineRunner 또는 ApplicationRunner를 구현하여, 서버 구동 시점에 DB에 “admin” 계정을 미리 생성하거나, 관리자가 없으면 자동으로 추가하도록 로직이 포함된다.
SecurityConfig	관련 설정을 담당하는 시큐리티 설정 클래스이다. HttpSecurity를 이용해 URL별 접근 권한(ROLE_USER, ROLE_ADMIN), 로그인 페이지, 로그아웃 처리 방식을 정의하고, PasswordEncoder(BCryptPasswordEncoder) 및 DaoAuthenticationProvider 빈 등록을 통해 일반 회원/관리자 인증 프로바이더를 각각 설정한다.
AuthController	인증(로그인·회원가입·로그아웃) 기능을 처리하는 컨트롤러 클래스이다. • @GetMapping("/login"), @GetMapping("/signup") → 로그인·회원가입 폼 페이지 반환 • @PostMapping("/login") → 사용자 인증 로직 호출 후 성공 시 리다이렉트 • @PostMapping("/signup") → 회원가입 정보 처리 → “회원가입 성공” 메시지 출력 • @GetMapping("/logout") → 로그아웃 처리 후 홈으로 리다이렉트 MatchController 경기(매치) 예약·조회·참여·평가 기능을 처리하는 컨트롤러 클래스이다. • @GetMapping("/reserve") → 매치 예약

	폼(요일·시간·지역·실력 선택) 페이지 반환 • @PostMapping("/reserve") → 폼에서 넘어온 dayOfWeek, time, region, skillLevel 정보를 받아 MatchService.reserveMatch(...) 호출 • @GetMapping("/list") → 특정 지역+실력 레벨에 맞는 매치 목록 조회 후 반환 • @GetMapping("/my") → 현재 로그인한 사용자가 참여 중인 매치 목록 조회 후 반환 • @GetMapping("/evaluate/{matchId}") → 경기 평가 폼 페이지 반환(같이 뛴 동료 목록 제공) • @PostMapping("/evaluate") → 평가 대상자 ID, 점수를 받아 EvaluationService.saveEvaluation(...) 호출
AdminController	관리자 전용 화면(회원 관리·매치 관리·평가 관리 등)을 처리하는 컨트롤러 클래스이다. • @GetMapping("/admin/home") → 관리자 대시보드(홈) 페이지 반환 • @GetMapping("/admin/members") → 전체 회원 목록 조회 후 반환 • @GetMapping("/admin/matches") → 전체 매치 목록 조회 후 반환 • @GetMapping("/admin/match/{id}") → 특정 매치 상세 정보 조회 후 반환 • 회원 강제 탈퇴, 매치 강제 취소, 평가 관리 등의 추가 메서드 포함 가능
Member	일반 사용자(회원) 정보를 담는 엔티티 클래스이다. 주요 필드: Long id, String username(로그인 ID), String password, String email, String skillLevel(실력 레벨: 초급/중급/고급), String region(지역), String role(ROLE_USER). 주요 메서드: 회원 식별자 조회(getUsername), 비밀번호 조회(getPassword), 실력 레벨/지역 조회(getSkillLevel, getRegion), 권한 설정/조회(setRole, getRole) 등.
Admin	관리자 계정 정보를 담는 엔티티 클래스이다. 주요 필드: Long id, String username, String password, String role(ROLE_ADMIN). 주요 메서드: 관리자 식별자 조회(getUsername), 비밀번호 조회(getPassword), 권한 조회(getRole), 권한 설정(setRole) 등. AdminInitializer에서 초기 관리자로 저장하거나 관리자가 직접 DB에 추가할 때 사용된다.
Match	경기(매치) 정보를 담는 엔티티 클래스이다. 주요 필드: Long id(매치 고유 식별자), String dayOfWeek(요일:월~일), String time(시간: 09:00 ~ 23:00) String region(지역), String skillLevel(실력 레벨: A/B/C). 주요 메서드: 평가 조회(getScore), 평가자/피평가자/매치 조회(getEvaluator, getEvaluatee, getMatch).

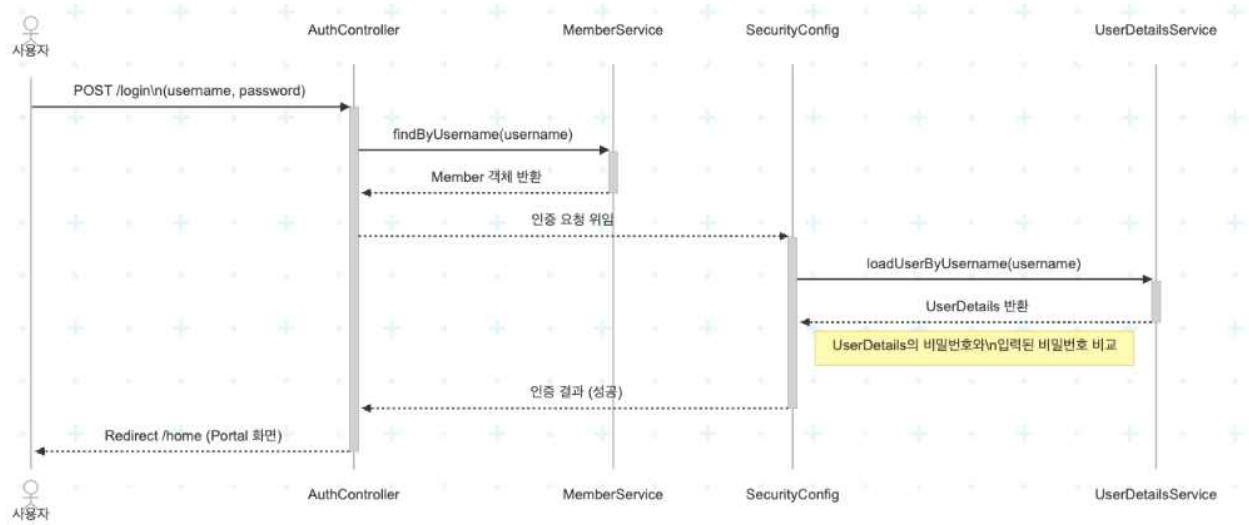
	Position 축구 포지션 정보를 담는 엔티티 클래스이다. 주요 필드: Long id(포지션 고유 식별자), String name(포지션 이름: GK, DF, MF, FW 등). 주요 메서드: 포지션 이름 조회(getName), 이름 설정(setName) 등. 회원가입 시 담당 포지션 지정, 매치 구성 시 포지션별 인원 제한 체크 등에 사용된다.
Evaluation	경기 후 참가자 간 평가(평점) 정보를 담는 엔티티 클래스이다. 주요 필드: Long id(평가 고유 식별자), Member evaluator(평가를 남긴 회원, @ManyToOne), Member evaluatee(평가 대상 회원, @ManyToOne), Match match(어떤 매치에 대한 평가인지, @ManyToOne), Integer score(평점: 예. 1~5). 주요 메서드: 평가 조회(getScore), 평가자/피평가자/매치 조회(getEvaluator, getEvaluatee, getMatch)
Position	축구 포지션 정보를 담는 엔티티 클래스이다. 주요 필드: Long id(포지션 고유 식별자), String name(포지션 이름: GK, DF, MF, FW 등). 주요 메서드: 포지션 이름 조회(getName), 이름 설정(setName) 등. 회원가입 시 담당 포지션 지정, 매치 구성 시 포지션별 인원 제한 체크 등에 사용된다
MemberRepository	Member 엔티티의 CRUD 및 조회 기능을 제공하는 JPA Repository(인터페이스)이다. 주요 메서드: Optional<Member> findByUsername(String username)(아이디로 회원 조회), Optional<Member> findById(Long id)(회원 고유 ID로 조회), Member save(Member m)(회원 저장/수정), void delete(Member m)(회원 삭제).
AdminRepository	Admin 엔티티의 CRUD 및 조회 기능을 제공하는 JPA Repository(인터페이스)이다. 주요 메서드: Optional<Admin> findByUsername(String username)(관리자 아이디로 조회), Admin save(Admin a)(관리자 저장/수정), void delete(Admin a)(관리자 삭제).
MatchRepository	Match 엔티티의 CRUD 및 조건별 조회 기능을 제공하는 JPA Repository(인터페이스)이다. 주요 메서드: Optional<Match> findById(Long id)(매치 ID로 조회), List<Match> findByRegionAndSkillLevel(String region, String skillLevel)(지역+실력별 매치 조회),

	Optional<Match> findByDayOfWeekAndTimeAndRegionAndSkillLevel(String dayOfWeek, String time, String region, String skillLevel)(동일 요일·시간·지역·실력별 매치 조회), Match save(Match m)(매치 저장/수정), void delete(Match m)(매치 삭제).
EvaluationRepository	Evaluation 엔티티의 CRUD 및 조회 기능을 제공하는 JPA Repository(인터페이스)이다. 주요 메서드: List<Evaluation> findByEvaluatee(Member member)(특정 회원이 받은 평가 리스트 조회), Evaluation save(Evaluation e)(평가 저장/수정).
PositionRepository	Position 엔티티의 CRUD 및 조회 기능을 제공하는 JPA Repository(인터페이스)이다. 주요 메서드: Optional<Position> findByName(String name)(포지션 이름으로 조회), List<Position> findAll()(모든 포지션 조회), Position save(Position p)(포지션 저장/수정).
MemberService	MemberRepository를 주입받아 회원 관련 비즈니스 로직을 수행하는 서비스 클래스이다. 주요 메서드: Member findByUsername(String username)(회원 조회), Member findById(Long id)(회원 고유 ID로 조회), Member register(MemberDto dto)(회원가입 처리), void deleteMember(Long id)(회원 탈퇴/삭제 처리).
AdminService	AdminRepository를 주입받아 관리자 관련 비즈니스 로직을 수행하는 서비스 클래스이다. 주요 메서드: Admin createDefaultAdmin()(최초 관리자 계정 생성), List<Admin> findAllAdmins()(모든 관리자 조회).
MatchService	MatchRepository를 주입받아 매치 예약·조회·확정·참여 취소 등 매치 관련 비즈니스 로직을 수행하는 서비스 클래스이다. 주요 메서드: Match getOrCreateMatch(String dayOfWeek, String time, String region, String skillLevel)(동일 요일·시간·지역·실력 매치를 조회하고, 없으면 새로 생성하여 반환), void reserveMatch(String dayOfWeek, String time, String region, String skillLevel, Member member)(회원의 실력 레벨이

	요청값과 일치하는지 확인 후, 참가자 추가·확정 여부 처리), List<Match> getAvailableMatches(String region, String skillLevel)(지역+실력별로 참가 인원 12명 미만인 매치 목록 조회), List<Match> getMatchesByMember(Member member)(특정 회원이 참여 중인 모든 매치 조회), Match getMatchById(Long id)(매치 고유 ID로 조회), List<Member> getOtherParticipants(Match match, Member self)(특정 매치에서 자기 자신 제외한 다른 참가자 목록 조회(평가 대상자 추출용)). |
EvaluationService	EvaluationRepository를 주입받아 경기 평가 정보 저장 및 조회 로직을 수행하는 서비스 클래스이다. 주요 메서드: void saveEvaluation(Match match, Member evaluator, Member evaluatee, Integer score)(평가 엔티티 생성 후 저장, 필요 시 평균 평점 갱신), Double getAverageScore(Long memberId)(특정 회원이 받은 평균 평점 계산하여 반환).
PositionService	PositionRepository를 주입받아 포지션 관련 비즈니스 로직을 수행하는 서비스 클래스이다. 주요 메서드: List<Position> findAllPositions()(모든 포지션 목록 조회), Position findByName(String name)(포지션 이름으로 단일 포지션 조회)
UniversalUserDetailsService	Spring Security의 `UserDetailsService`를 구현한 서비스로, 일반 회원(ROLE_USER) 인증을 담당한다. 주요 메서드: UserDetails loadUserByUsername(String username)(MemberRepository.findByUsername(...) 호출 후 UserDetails 객체 생성·반환).
AdminUserDetailsService	Spring Security의 `UserDetailsService`를 구현한 서비스로, 관리자(ROLE_ADMIN) 인증을 담당한다. 주요 메서드: UserDetails loadUserByUsername(String username)(AdminRepository.findByUsername(…) 호출 후 UserDetails 객체 생성·반환).

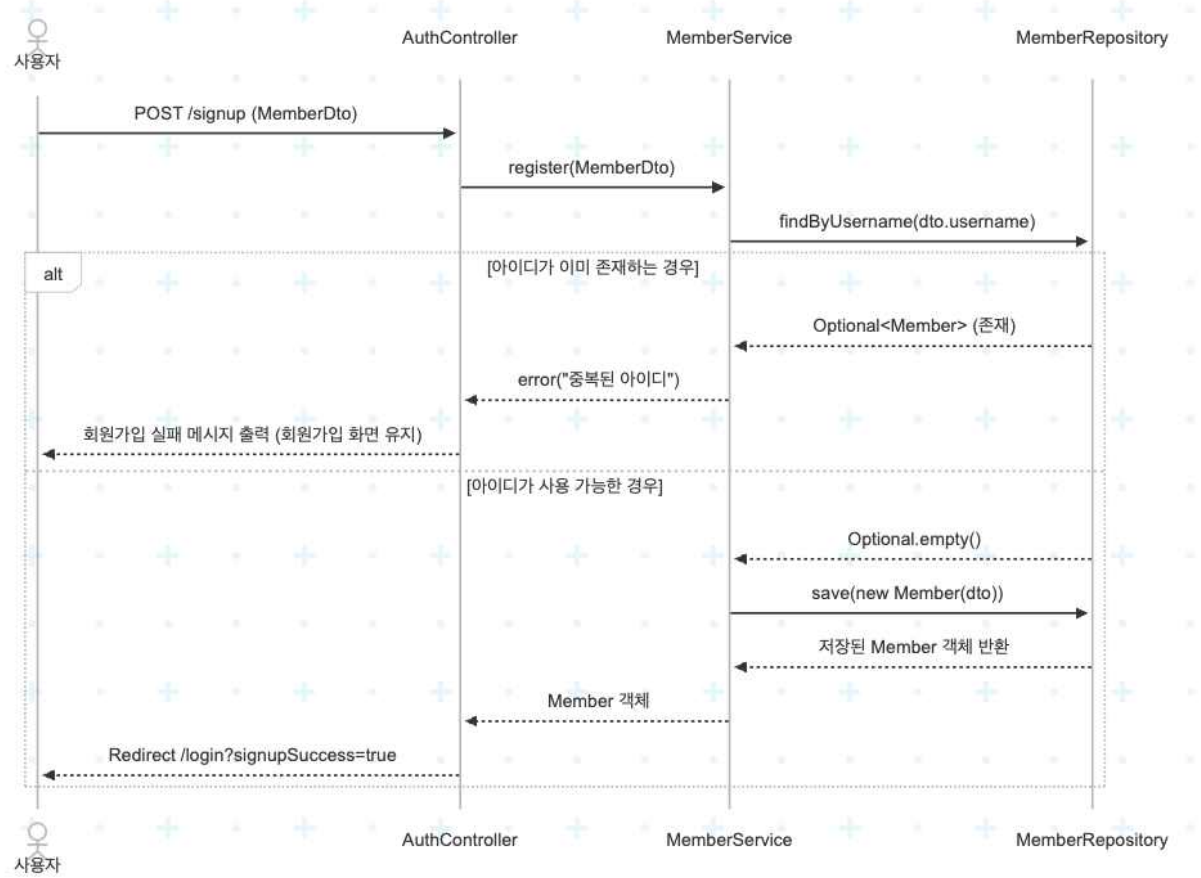
3. Sequence diagram

1. 로그인 기능



사용자가 로그인 화면에 아이디와 비밀번호를 입력하고 로그인 버튼을 누르면, 브라우저는 /login 경로로 POST 요청을 보낸다. 이 요청을 받은 AuthController는 먼저 사용자가 입력한 아이디(username)를 이용해 MemberService.findByUsername(username)을 호출하고, 데이터베이스에서 해당 아이디로 등록된 Member 객체를 조회한다. 조회된 Member 객체가 컨트롤러로 반환되면, AuthController는 실제 인증 절차를 스프링 시큐리티(SecurityConfig)에게 위임한다. 스프링 시큐리티의 설정 클래스인 SecurityConfig는 전달받은 아이디 정보를 바탕으로 UserDetailsService.loadUserByUsername(username)을 호출하여 UserDetails 객체를 얻는다. 이 UserDetails 객체에는 해당 사용자 계정의 암호화된 비밀번호와 권한 정보 등이 담겨 있다. 그런 다음 시큐리티는 사용자가 입력한 평문 비밀번호와 UserDetails에 저장된 암호화된 비밀번호를 비교하여 일치 여부를 확인한다. 비밀번호가 일치하면 인증이 성공했다는 결과를 AuthController로 돌려주고, AuthController는 사용자에게 로그인 성공을 알리기 위해 홈 페이지(예: /home 또는 대시보드)로 리다이렉트 응답을 보낸다. 만약 아이디가 존재하지 않거나 입력된 비밀번호가 틀린 경우에는 인증 실패 처리로 이어지며, 에러 메시지를 모델에 담아 로그인 화면으로 다시 이동시키거나 별도의 에러 페이지로 이동시킨다.

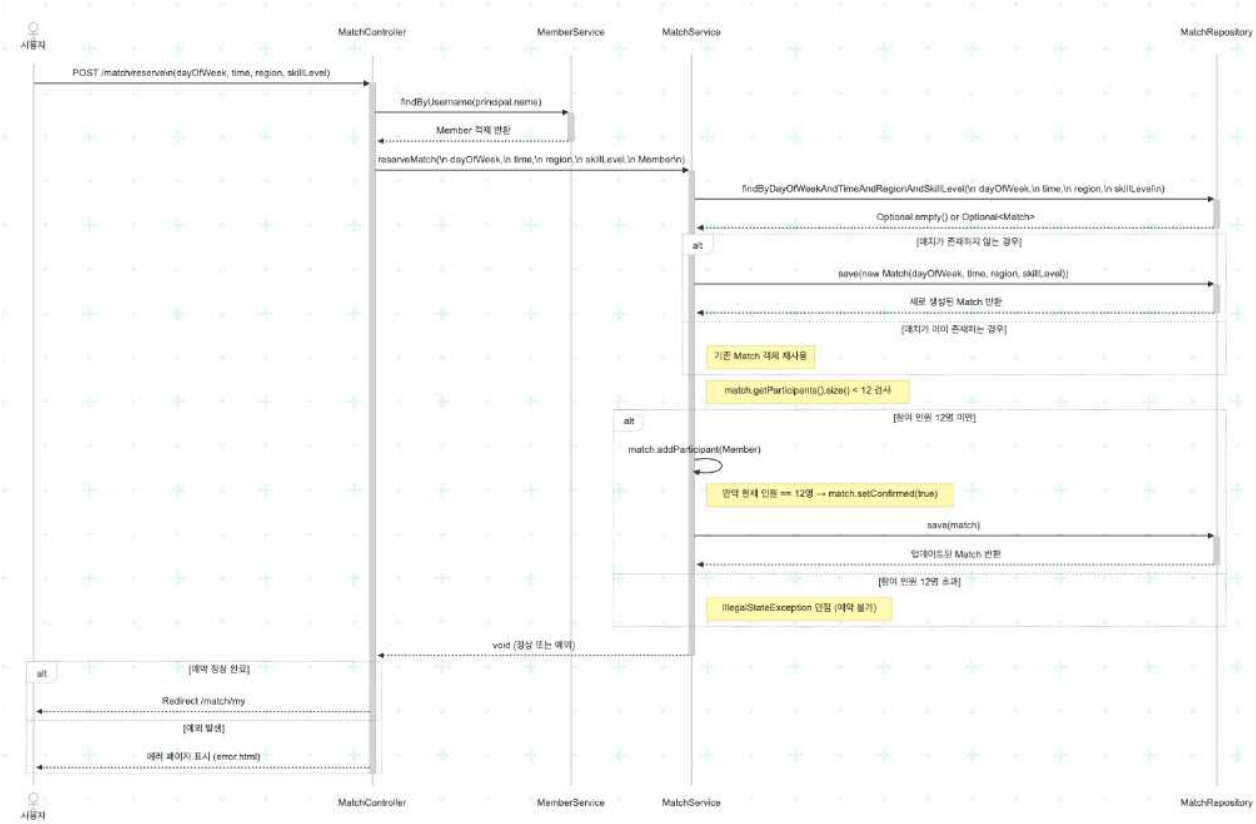
2. 회원가입 기능



사용자가 회원가입 화면에서 아이디, 비밀번호, 이메일, 실력 레벨, 지역 등의 정보를 입력한 뒤 회원가입 버튼을 누르면, 브라우저는 /signup 경로로 POST 요청을 보낸다. 이 요청을 받은 AuthController는 곧바로 MemberService.register(MemberDto) 메서드를 호출하며 회원가입 로직을 위임한다.
MemberService.register 메서드는 먼저 데이터 전송 객체(MemberDto)에 담긴 아이디(username)를 꺼내 MemberRepository.findByUsername(username)를 호출하여 데이터베이스에 동일한 아이디를 가진 사용자가 이미 있는지를 확인한다. 만약 이미 같은 아이디를 가진 회원 정보가 존재한다면(Optional<Member>가 non-empty로 반환된다면), MemberService는 중복된 아이디라는 예외 상태를 AuthController에게 전달하고 종료한다. 그러면 AuthController는 “중복된 아이디입니다”라는 에러 메시지를 모델에 담아 다시 회원가입 화면을 렌더링하도록 되돌려 보낸다.
반면, MemberRepository.findByUsername(username)가 빈 결과(Optional.empty())를 반환해서 아이디가 사용 가능한 상태라면, MemberService는 전송받은 MemberDto를 기반으로 새로운 Member 엔티티 객체를 생성한다. 이때 비밀번호는 BCryptPasswordEncoder 같은 암호화기를 사용해 반드시 암호화된 형태로 저장하도록 처리한 후, MemberRepository.save(new Member)를 호출하여 데이터베이스에 회원 정보를 저장한다. 저장이 완료되면, MemberService는 저장된 Member 객체를 다시 AuthController에게 반환한다.
AuthController는 회원가입이 성공했음을 확인하고, 최종적으로 사용자 브라우저에 Redirect /login?signupSuccess=true 응답을 보낸다. 이 리다이렉션을 통해 사용자는

로그인 페이지로 이동하게 되며, URL의 쿼리 파라미터(signupSuccess=true)를 이용해 “회원가입이 정상적으로 완료”라는 안내 메시지를 화면에 표시할 수 있게 된다. 만약 회원가입 과정에서 아이디 중복이나 기타 검증 오류가 발생한 경우에는 해당 오류 메시지를 담아 회원가입 폼을 다시 보여주도록 처리한다.

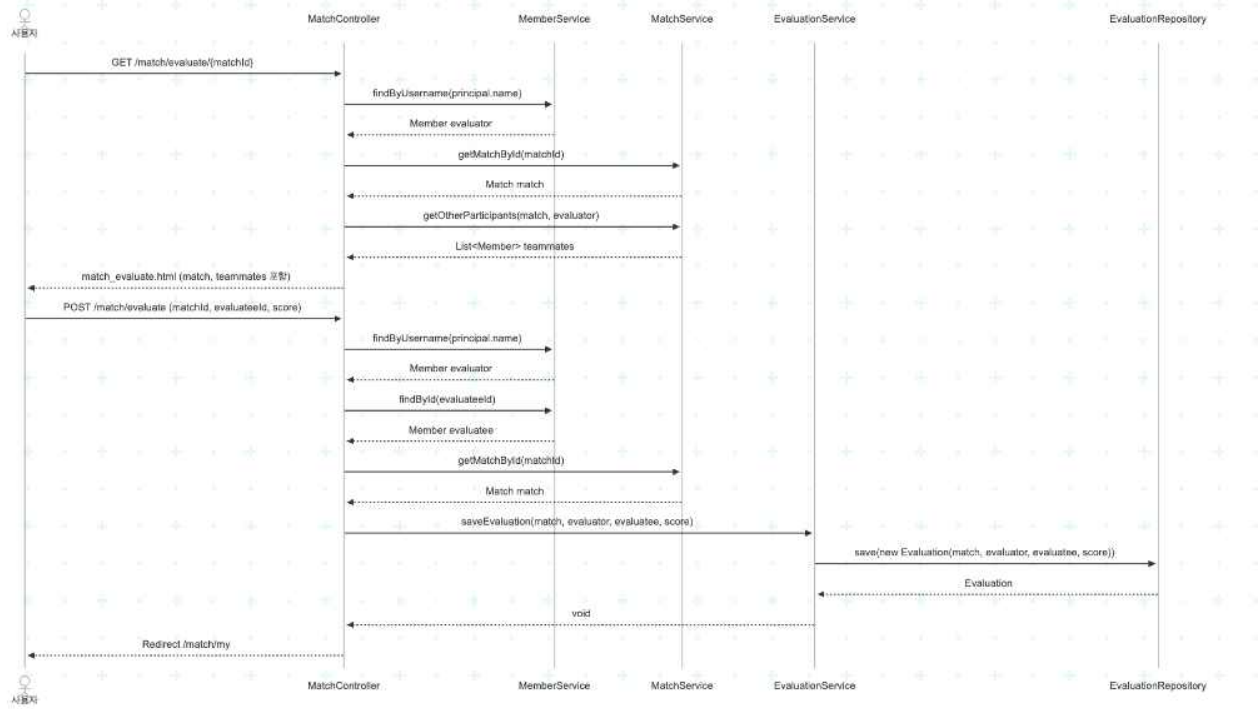
3. 경기 예약



사용자가 경기 예약 화면에서 요일, 시간, 지역, 실력 레벨을 선택하고 “예약하기” 버튼을 누르면, 브라우저는 /match/reserve 경로로 POST 요청을 보낸다. 이 요청을 받은 MatchController는 먼저 MemberService.findByUsername(principal.name)을 호출하여 현재 로그인된 사용자의 Member 객체를 데이터베이스에서 조회한다. 반환된 Member 객체를 바탕으로, 컨트롤러는 MatchService.reserveMatch(dayOfWeek, time, region, skillLevel, member) 메서드를 호출하며 실제 예약 로직을 수행하도록 위임한다.
MatchService.reserveMatch가 실행되면, 우선 동일한 요일·시간·지역·실력 레벨에 해당하는 매치가 이미 존재하는지 확인하기 위해 MatchRepository.findByDayOfWeekAndTimeAndRegionAndSkillLevel(dayOfWeek, time, region, skillLevel)을 호출한다. 이 단계에서 데이터베이스에 해당 조건의 매치가 하나도 없었다면 Optional.empty()가 반환되므로, reserveMatch는 새로운 Match 객체를 생성자에 dayOfWeek, time, region, skillLevel을 전달하여 만든 뒤 MatchRepository.save(new Match(...))를 호출해 저장하고, 저장된 Match 엔티티를 다시 받아온다. 반면 이미 같은 조건의 매치가 존재한다면, 저장소가 반환한 기존 Match 객체를 그대로 사용한다.
그다음 MatchService는 가져온 Match 객체를 조회하여 현재 참여자

수(match.getParticipants().size())가 12명 미만인지 확인한다. 만약 12명보다 적다면 match.addParticipant(member)를 호출해 사용자를 해당 매치의 참여자 목록에 추가한다. 추가한 뒤에 참여자 수가 딱 12명이 되었다면 match.setConfirmed(true)를 호출하여 매치가 성사되었음을 표시한다. 그리고 변경된 상태를 MatchRepository.save(match)로 다시 저장하여 데이터베이스에 반영한다.
반대로 참여자 수가 이미 12명을 넘어선 상태이거나 match.getParticipants().size() >= 12로 판단되면, MatchService는 IllegalStateException을 던지거나 별도의 오류 처리를 수행하여 "더 이상 예약할 수 없다"는 예외 상황을 발생시킨다.
이 과정을 마치고 제어가 MatchController로 다시 돌아오면, 정상적으로 예약이 완료된 경우 MatchController는 사용자에게 "내 매치 보기" 페이지로 이동하도록 redirect:/match/my 응답을 보낸다. 사용자는 브라우저에서 리다이렉트되어 자신이 참여 중인 경기 목록을 확인할 수 있다. 반면 예약 과정에서 예외가 발생했다면, MatchController는 모델에 오류 메시지를 담아 에러 전용 페이지(예: error.html)를 렌더링하거나 다시 예약 화면으로 돌아가도록 처리한다.

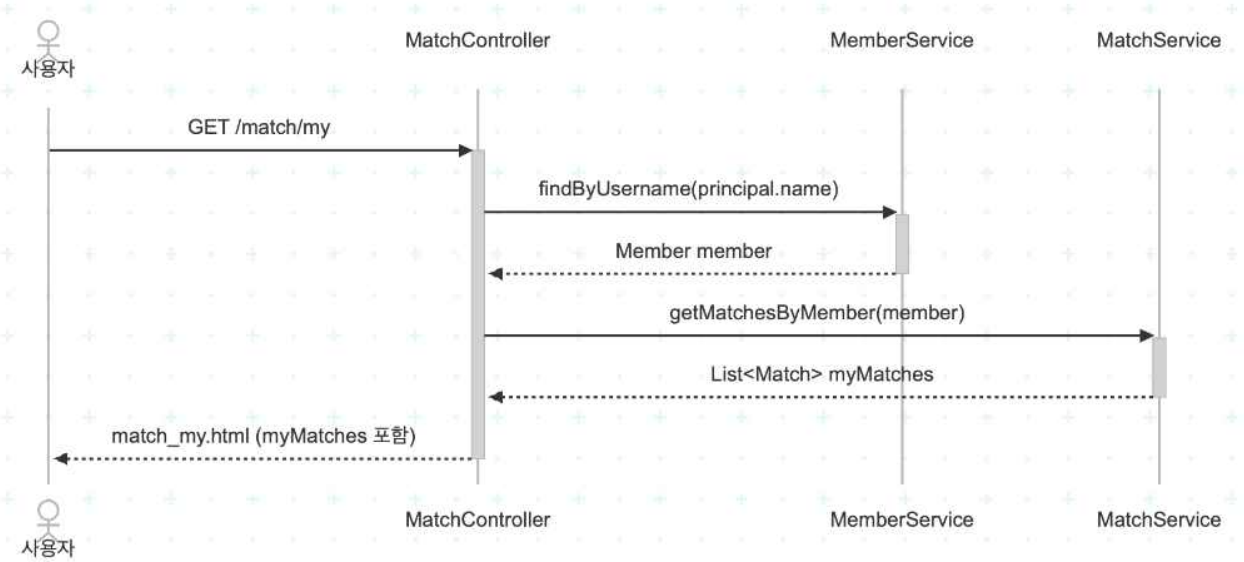
4. 경기 평가



사용자가 경기 평가 화면에서 평가할 경기의 ID를 포함해 "평가하기" 버튼을 누르면, 브라우저는 /match/evaluate/{matchId} 경로로 GET 요청을 보낸다. 이 요청을 받은 MatchController는 먼저 MemberService.findByUsername(principal.name)을 호출하여 현재 로그인된 회원(evaluator)의 Member 객체를 데이터베이스에서 조회한다. 이후 MatchService.getMatchById(matchId)를 호출해 해당 경기(Match) 정보를 가져오고, MatchService.getOtherParticipants(match, evaluator)를 호출해 자신을 제외한 동일 경기 참가자들(teammates) 목록을 반환받는다. 이렇게 조회된 match 정보와 teammates 리스트를 모델에 담아 match_evaluate.html 뷰로 렌더링하면, 사용자는 평가 대상자(동료 선수) 목록과 점수 입력 필드를 확인할 수 있다.
사용자가 평가 대상자 아이디(evaluateeId)와 평점(score)을 선택해 제출 버튼을 누르면, 브라우저는 /match/evaluate 경로로 POST 요청(파라미터: matchId, evaluateeId,

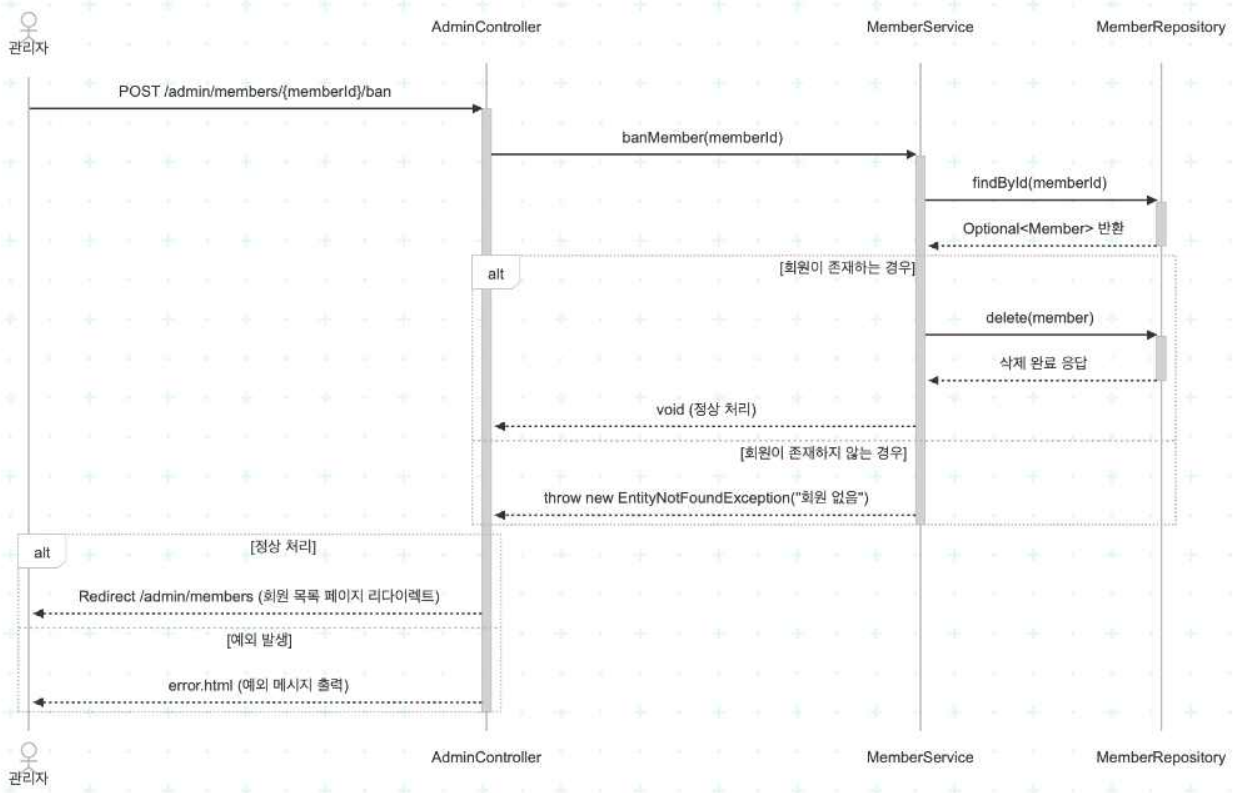
score)을 보낸다. MatchController는 다시 MemberService.findByUsername(principal.name)으로 평가자(evaluator) 정보를 가져오고, MemberService.findById(evaluateeId)를 호출해 피평가자(evaluatee) 정보를 조회한다. 그리고 MatchService.getMatchById(matchId)로 해당 경기 정보를 다시 가져온 뒤, EvaluationService.saveEvaluation(match, evaluator, evaluatee, score)를 호출하며 평가 저장 로직을 실행한다.
EvaluationService.saveEvaluation 내부에서는 new Evaluation(match, evaluator, evaluatee, score) 형태로 새로운 Evaluation 엔티티 객체를 생성하고, EvaluationRepository.save(new Evaluation(...))를 통해 데이터베이스에 평가 정보를 저장한다. 필요하다면 이 시점에 피평가자의 누적 또는 평균 평점을 다시 계산하도록 추가 로직이 포함될 수 있다. 저장이 완료되면 EvaluationService는 호출한 MatchController로 제어를 반환한다.
최종적으로 MatchController는 평가가 정상적으로 저장되었다면 사용자에게 redirect:/match/my 응답을 보내 "내가 참여한 경기 목록"으로 이동시키고, 만약 데이터 저장 중 예외가 발생했다면 에러 메시지를 모델에 담아 에러 페이지(error.html)로 렌더링한다.

5. 매치 조회



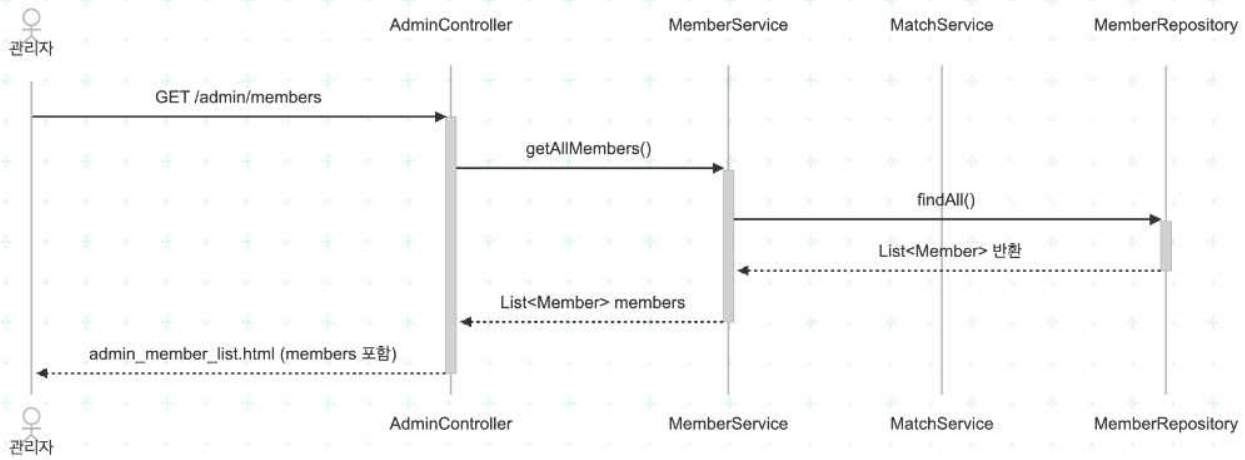
사용자가 브라우저에서 "내 매치 보기" 페이지를 요청하면, 클라이언트는 GET /match/my 경로로 요청을 보낸다. 이 요청을 받은 MatchController는 먼저 현재 로그인된 사용자의 아이디(Principal 객체)를 이용해 MemberService.findByUsername(principal.name)을 호출하고, 데이터베이스에서 해당 회원(Member) 정보를 조회한다. 조회된 Member 객체를 인자로 MatchController는 MatchService.getMatchesByMember(member)를 호출하여 그 회원이 참여 중인 모든 경기(Match) 목록을 가져온다.
MatchService는 내부적으로 MatchRepository를 사용해 해당 회원을 포함하는 매치들을 검색한 뒤, 결과 리스트(List<Match> myMatches)를 컨트롤러로 반환한다. 마지막으로 MatchController는 myMatches 리스트를 모델에 담아 match_my.html 뷰를 렌더링하여 클라이언트에 반환한다. 화면에는 사용자가 예약하거나 참여한 경기들이 테이블이나 카드 형식으로 나열되어, 사용자는 자신이 속해 있는 모든 경기 정보를 한눈에 확인할 수 있다.

6.1. 관리자 - 회원 탈퇴



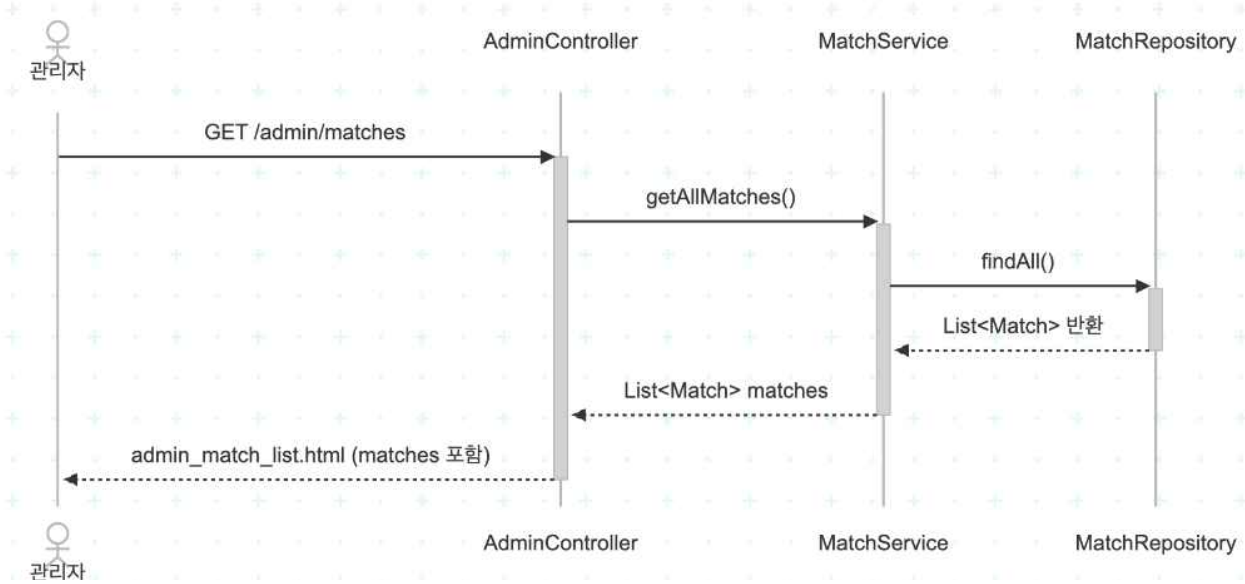
관리자가 특정 회원을 강제로 탈퇴시키기 위해 회원 목록 화면 등에서 “강제 탈퇴” 버튼을 누르면, 클라이언트는 해당 회원의 ID를 포함한 POST 요청 /admin/members/{memberId}/ban을 보낸다. 이 요청을 받은 AdminController는 내부에서 MemberService.banMember(memberId) 메서드를 호출하며 실제 탈퇴 처리를 요청한다.
MemberService는 전달받은 ID를 바탕으로 MemberRepository.findById(memberId)를 호출해 데이터베이스에서 해당 회원이 존재하는지를 조회한다. 회원이 존재할 경우, MemberRepository.delete(member) 메서드를 호출하여 해당 회원 정보를 데이터베이스에서 완전히 삭제한다. 반면, 회원이 존재하지 않을 경우에는 EntityNotFoundException 또는 유사한 커스텀 예외를 발생시켜 컨트롤러에 예외 상황을 알린다.
AdminController는 처리 결과에 따라 분기한다. 삭제가 정상적으로 완료된 경우에는 관리자를 /admin/members 경로로 리다이렉트하여 전체 회원 목록 화면을 새로고침한다. 반대로 예외가 발생했을 경우에는 오류 메시지를 포함해 error.html 페이지를 렌더링하거나 알림창을 통해 오류를 사용자에게 전달한다.

6.2 관리자 - 회원 목록 조회



관리자가 관리자 페이지에서 전체 회원 목록을 조회하고자 하면, 클라이언트는 GET /admin/members 요청을 보낸다. 이 요청은 AdminController가 받아 처리하며, 내부적으로 MemberService.getAllMembers()를 호출해 실제 데이터를 요청한다. MemberService는 MemberRepository.findAll()을 호출하여 데이터베이스에 등록된 전체 회원 엔티티 리스트(List<Member>)를 가져온다. 이 리스트는 다시 컨트롤러로 반환되고, AdminController는 해당 데이터를 모델에 담아 admin_member_list.html 뷰를 렌더링하여 관리자에게 화면을 제공한다. 관리자는 이 화면을 통해 각 회원의 정보(ID, 이름, 실력, 지역 등)를 확인하고, 필요 시 강제 탈퇴 등의 조치를 할 수 있게 된다.

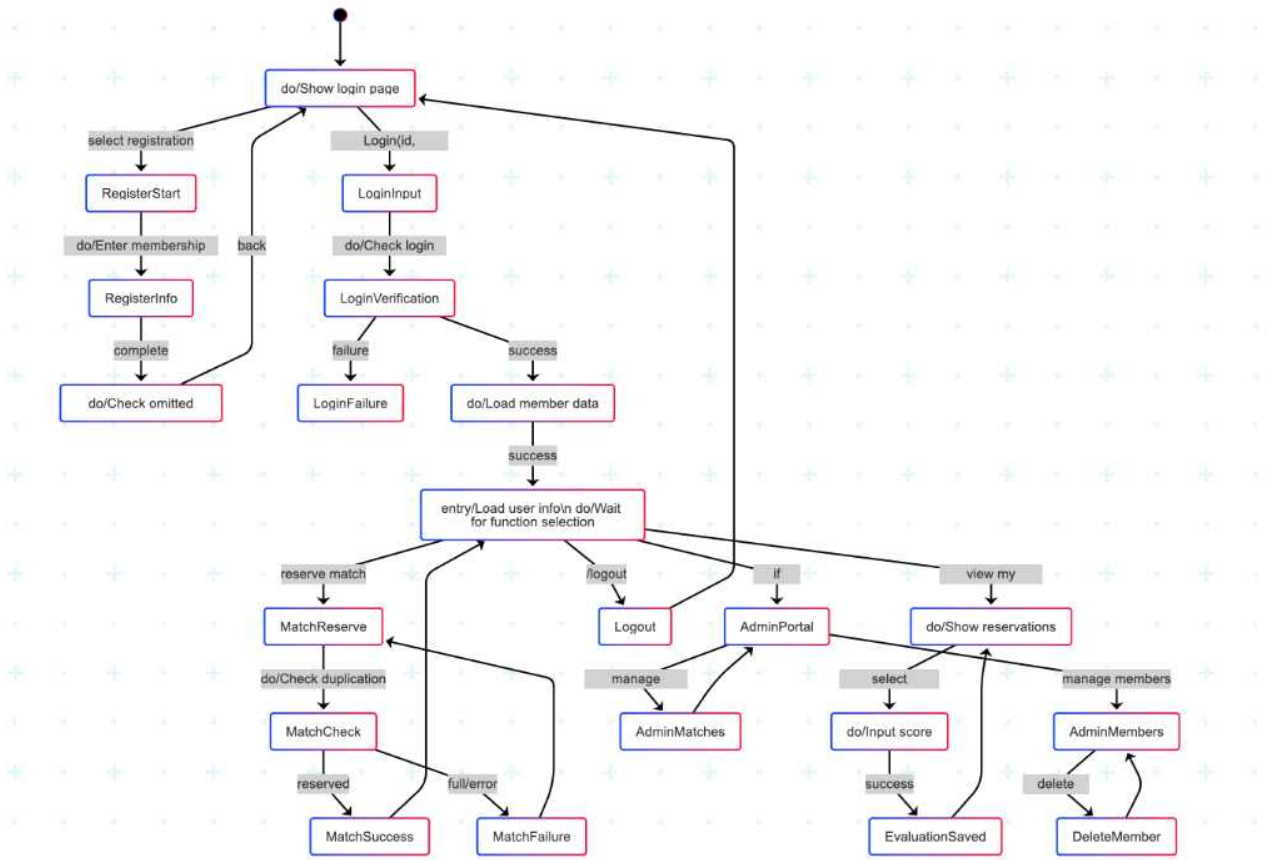
6.3 관리자 - 매치 조회



관리자가 등록된 모든 경기를 확인하고자 하면, 클라이언트는 GET /admin/matches 요청을 보낸다. 이 요청은 AdminController가 수신하고, 내부에서 MatchService.getAllMatches() 메서드를 호출하여 전체 매치 정보를 요청한다. MatchService는 MatchRepository.findAll()을 호출해 데이터베이스에 등록된 모든 매치 엔티티 객체(List<Match>)를 조회한다. 조회된 결과는 컨트롤러로 반환되며, AdminController는 이 데이터를 모델에 담아 admin_match_list.html 뷰를

렌더링한다.
이 페이지에는 각 경기의 ID, 지역, 요일, 시간, 실력 레벨, 현재 참가자 수, 확정 여부 등의 정보가 나열되며, 관리자는 이를 통해 경기 진행 현황을 파악하거나 필요 시 매치 상세보기 또는 강제 취소 등의 추가 조치를 할 수 있다.

4. State machine diagram



Status	Explanation
Launch System	시스템이 처음 실행되어 로그인 화면이 출력된 상태입니다. 사용자는 여기서 로그인하거나 회원가입을 선택할 수 있습니다.
Register Start	사용자가 “회원가입” 버튼을 클릭하여 등록을 시작하는 상태입니다.
Register Info	회원가입 입력 화면에서 사용자가 정보를 입력 중인 상태입니다.
Input Verification	입력된 회원 정보의 누락 여부를 검사하는 상태입니다. 입력 오류가 있을 경우 다시 입력하게 되며, 이상이 없으면 다음 단계로 넘어갑니다.
Login Input	사용자가 아이디와 비밀번호를 입력하는 로그인 화면 상태입니다.
Login Verification	입력한 아이디와 비밀번호가

	데이터베이스에 존재하는지 검증하는 상태입니다. 성공 시 다음 상태로 넘어가고, 실패 시 다시 처음으로 돌아갑니다.
Login failure	로그인에 실패한 상태로, 잘못된 정보로 인해 인증이 거부된 경우입니다. 다시 입력하거나 종료할 수 있습니다.
Init Member Info	로그인 성공 후 시스템이 현재 로그인한 사용자의 정보를 로딩하고 초기화하는 준비 상태입니다.
Portal	로그인된 사용자가 기능을 선택하거나 시스템을 종료할 수 있는 메인 상태입니다. 여기서 예약, 평가, 매치 조회, 로그아웃 등이 가능합니다.
Match Reserve	사용자가 경기 예약을 선택하여, 요일·시간·지역·실력 등을 입력하는 상태입니다.
Match Check	입력된 정보로 예약 가능한 경기(match)가 존재하는지 확인하는 상태입니다.
Match Success	예약이 성공적으로 이루어진 상태입니다
Match Failure	인원 초과 또는 예약 조건이 맞지 않아 예약에 실패한 상태입니다
Match List	로그인한 사용자가 본인이 예약한 경기 목록을 확인하는 상태입니다.
Evaluate Match	경기 후 함께한 팀원들에게 평점을 입력하는 상태입니다.
Evaluation saved	평가가 성공적으로 저장된 상태입니다.
Logout	사용자가 로그아웃 버튼을 누른 후, 다시 비로그인 상태로 전환되는 단계입니다.
Admin Portal	관리자 계정으로 로그인한 경우 진입 가능한 관리자 메뉴의 진입 상태입니다.
Admin Members	관리자 화면에서 전체 회원 목록을 조회하는 상태입니다.
Delete Member	관리자가 특정 회원을 강제 탈퇴시키는 기능을 실행하는 상태입니다.
Admin Matches	관리자 화면에서 전체 매치 목록을 조회하는 상태입니다.

5. Implementation requirements

1) Hardware Requirements

항목	최소 사양
CPU	Intel Core i3 이상
RAM	2GB 이상
HDD or SSD	200MB 이상의 여유 공간
Network	인터넷 연결 필수(경기 매칭 및 평가 등)

2) Software Requirements

항목	요구사항
OS	Windows 10 이상, macOS or Linux
Implementation Language	Java 17 이상, Spring Boot 사용

3) Nonfunctional Requirements

- Automatch 시스템은 웹 기반 서비스로, 별도의 물리 서버 없이 로컬 또는 클라우드 환경에서 실행될 수 있도록 설계되었다.
- 서버 역할은 Spring Boot 기반의 애플리케이션이 수행하며, 회원정보, 경기정보, 평가정보 등은 관계형 데이터베이스(MySQL 또는 H2)에 저장된다.
- 경기 매칭, 사용자 평가 등 주요 기능은 사용자 로그인 후 포털에서 선택적으로 수행되며, 클라이언트 요청에 따라 REST API가 호출되어 동작한다.
- 개발 중에는 내장 H2 데이터베이스를 사용할 수 있으며, 운영 환경에서는 MySQL 또는 PostgreSQL을 권장한다.
- 경기 매칭 로직은 실력, 지역, 요일, 시간 등의 기준에 따라 동적으로 팀을 구성하며, 매치 인원이 12명에 도달하면 자동으로 확정 처리된다.
- 사용자 인터페이스는 HTML + Thymeleaf 기반으로 설계되어, 별도의 브라우저 환경에서 접근 가능해야 하며, 크로스 브라우징을 고려하여 Chrome, Edge, Firefox를 권장한다.
- 관리자 기능(회원 강제 탈퇴, 매치 목록 확인 등)은 관리자 계정으로 로그인했을 때만 접근 가능하며, Spring Security로 보호된다.
- 모든 입력은 유효성 검사 후 서버에 반영되며, 부적절한 요청이나 비정상적인 접근 시 예외 처리와 오류 메시지를 통해 사용자에게 안내된다.

6. Glossary

Terms	Description
Match	Automatch 시스템에서 사용자들이 참여할 수 있는 축구 경기 단위를 의미한다. 요일, 시간, 지역, 실력 등의 속성을 가지며, 12명이 모이면 자동으로 확정된다.
Member	시스템에 가입한 사용자 객체로, 아이디, 비밀번호, 이름, 실력, 지역 등의 속성을 갖는다. 로그인, 예약, 평가 등의 모든 기능은 Member를 기준으로 동작한다.
MatchController	사용자의 요청(URL)을 처리하는 Spring MVC의 컨트롤러 클래스로, 경기 예약, 내 경기 보기, 평가 등의 흐름을 처리한다.
MatchService	경기 예약 및 평가와 같은 핵심 비즈니스 로직을 수행하는 서비스 계층 클래스이다. 컨트롤러와 저장소(repository) 사이에서 동작한다.
Evaluation	경기 종료 후 팀원에게 매기는 점수를 저장하는 객체이며, evaluator(평가자), evaluatee(피평가자), score를 포함한다.
@Entity	JPA에서 데이터베이스 테이블과 매핑되는 클래스를 의미한다. Member, Match, Evaluation 등의 클래스에 사용된다.
Repository	JPA를 통해 데이터베이스에 접근하기 위한 인터페이스 계층으로, `findById`, `save`, `findAll` 등의 메서드를 제공한다.
Principal	Spring Security에서 로그인된 사용자의 정보를 나타내는 인터페이스. 현재 접속 중인 사용자를 식별할 때 사용된다.
Thymeleaf	HTML과 Java 객체를 연결하는 템플릿 엔진으로, Automatch 시스템의 프론트엔드 뷰(View)를 구성하는 데 사용된다.
Spring Security	인증(Authentication)과 권한(Authorization) 처리를 담당하는 보안 프레임워크로, 로그인과 관리자 권한 제어에 활용된다.

7. References

1. Spring Security
   https://docs.spring.io/spring-security
2. Spring Boot Reference Documentation
   https://docs.spring.io/spring-boot
3. Spring Data JPA
   https://docs.spring.io/spring-data/jpa
4. Thymeleaf Documentation
   https://www.thymeleaf.org/documentation.html