2023

Design Document

동아리 스케줄 공유를 통한 공동공간 예약 및 조회 시스템 WePlan

7 조 박종범, 정유환, 김동령, 이승주

목차

Table of Figures

Table of Tables

1. Revised Requirements

1.1. Use Case Model

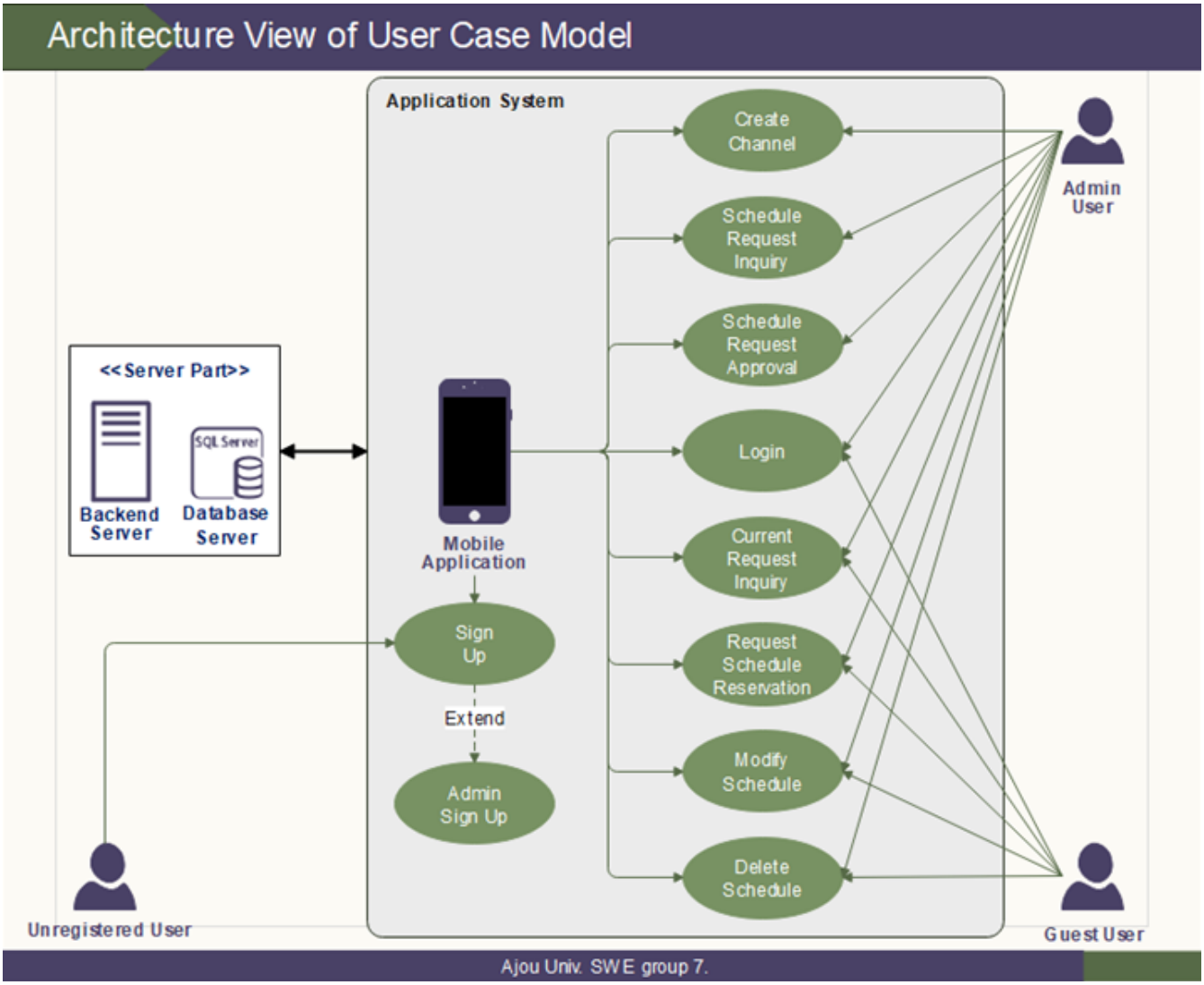

Figure 1-1

1.2. Use Case Scenario

1.2.1. Use Case: 로그인

Table 1-1

Use Case	1
Case Name	로그인
Actors	유저(게스트 및 관리자), 미가입 유저, 모바일 애플리케이션, DB 서버
Description	시스템을 이용하고자 하는 유저는 이전에 등록한 유저 ID와 유저 패스워드를 사용하여 로그인해야 한다.
Pre-Conditions	유저는 회원가입을 통해 유저의 정보와 유저 ID, 유저 패스워드를 미리 등록해야 한다.
Post-Conditions	시스템을 이용할 수 있게 된다.
Primary Flow	1. 유저는 로그인 창에 유저 ID와 유저 패스워드를 입력한다. 2. 서버는 모바일 애플리케이션으로부터 데이터를 수신한다.

	3. 서버는 수신한 데이터와 저장된 데이터의 비교를 수행한다. 3.1. 만약 데이터가 일치하지 않으면, 서버는 모바일 애플리케이션으로 불일치 데이터를 송신한다. 3.2. 일치하는 데이터가 있으면, 유저는 시스템을 이용할 수 있게 된다.

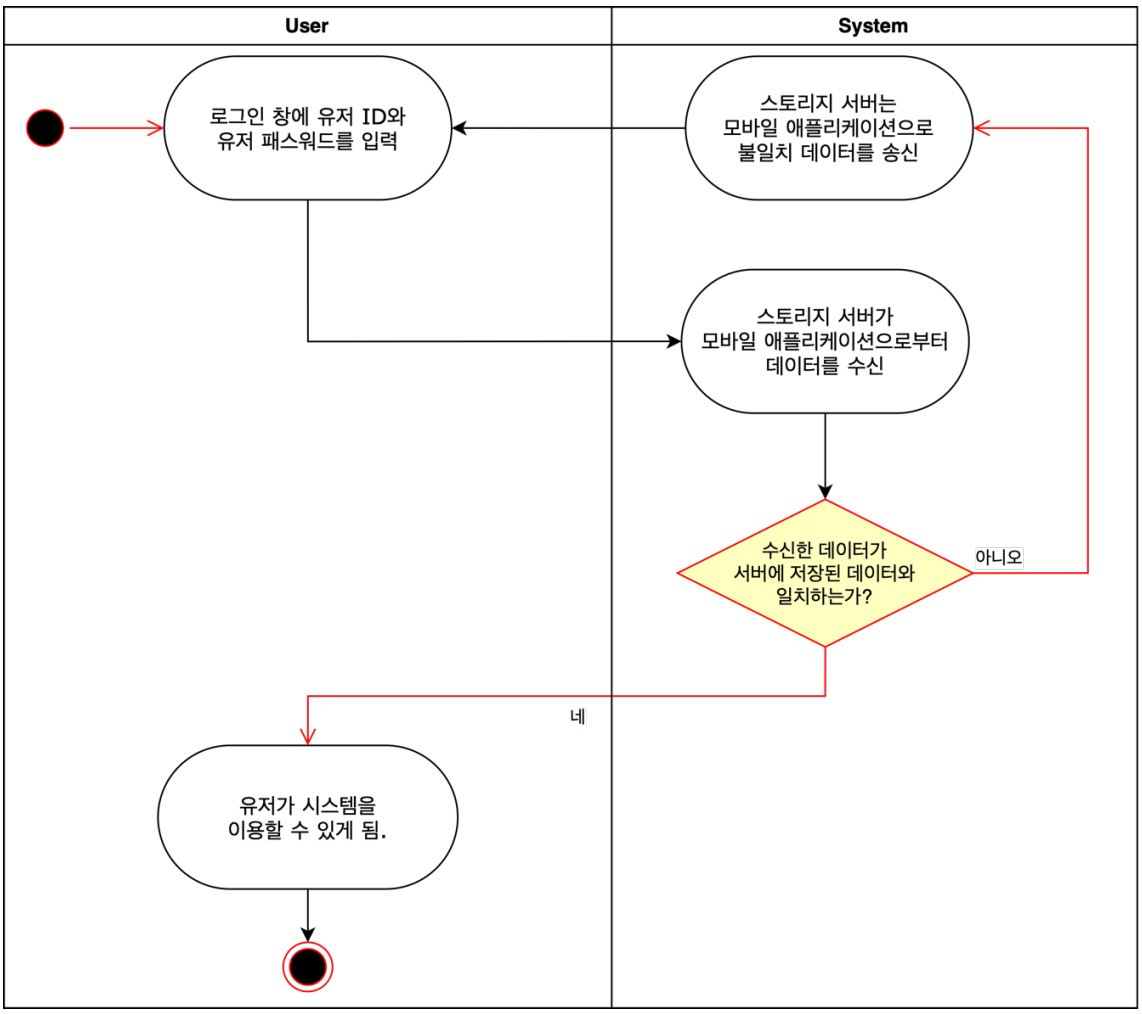

Figure 1-2

1.2.2. Use Case: 유저 등록

Table 1-2

Use Case	2
Case Name	유저 등록
Actors	사용자(게스트 및 관리자), 모바일 애플리케이션, DB 서버
Description	유저는 시스템을 이용하기 위해 등록이 필요하다.
Pre-Conditions	유저는 애플리케이션을 사용하기 위한 디바이스를 소유해야 하며, 디바이스에 애플리케이션을 설치해야 한다.
Post-Conditions	유저 정보가 DB 서버에 등록되고 유저는 시스템을 이용할 수 있게 된다.
Primary Flow	1. 유저가 애플리케이션을 실행한다. 2. 유저가 로그인 페이지에서 회원가입 페이지로 이동한다. 3. 유저는 회원가입 폼에 유저 ID, 유저 비밀번호, 이름, 전화번호 등의 정보를 입력한다. 3.1. 만약 회원가입 폼에 입력되지 않은 정보가 있다면, 애플리케이션은 유저에게 경고 메시

	지를 전달한다. 3.2. 등록 폼에 모든 필수 정보가 입력되면, 해당 정보는 스토리지 서버로 송신된다. 3.2.1. 서버는 유저에게 경고 메시지를 전달한다. 3.2.2. 만약 수신된 데이터가 이미 서버에 존재하면, 서버는 유저에게 경고 메시지를 전달한다. 3.2.3. 만약 수신된 데이터가 서버에 존재하지 않는 데이터라면, 해당 데이터를 서버에 저장한다. 3.2.3.1. 유저가 시스템에 로그인할 수 있게 된다.

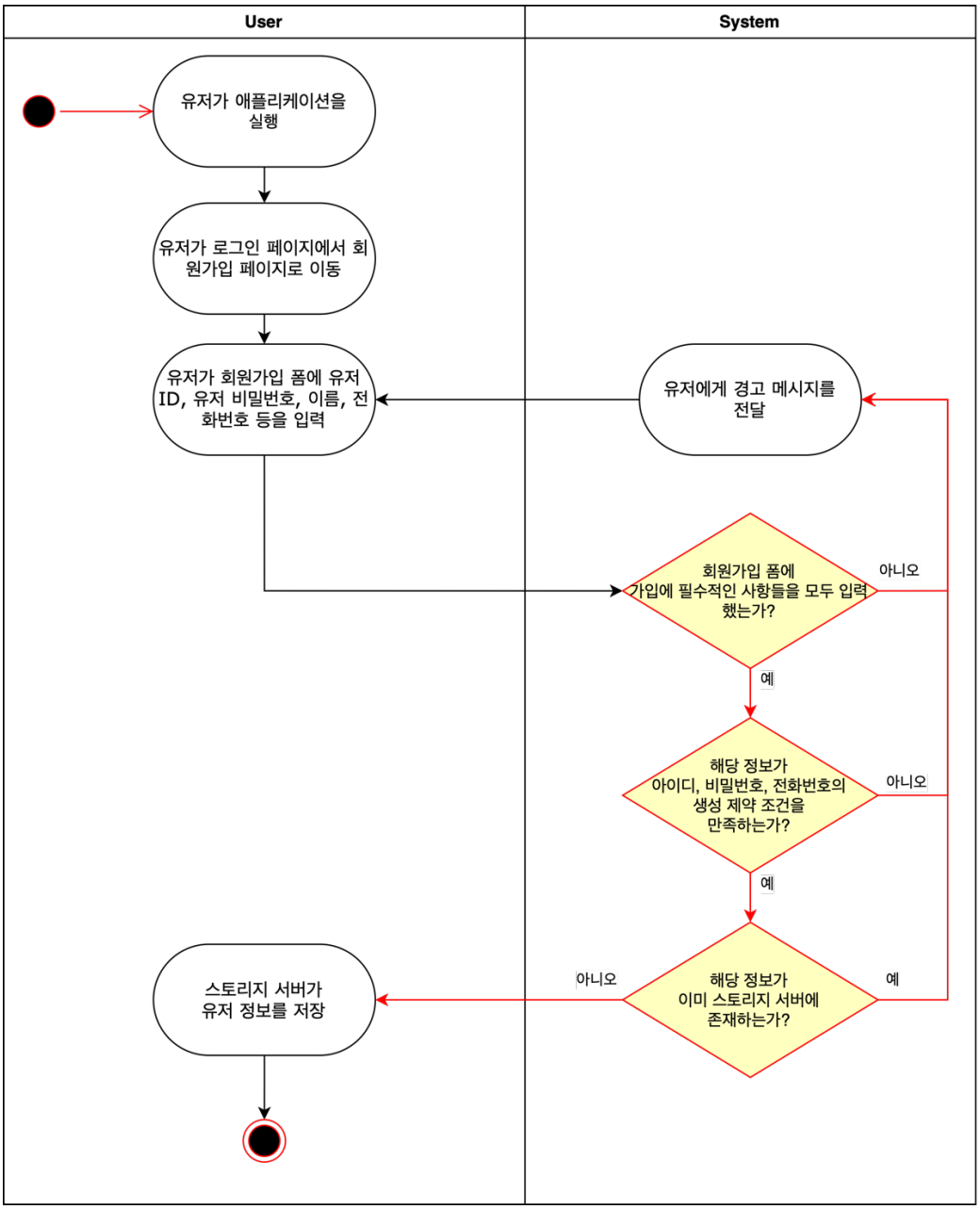

Figure 1-3

1.2.3. Use Case: 동아리 채널 생성

Table 1-3

Use Case	3

Case Name	동아리 채널 생성
Actors	유저(관리자), 모바일 애플리케이션, DB 서버
Description	관리자 권한을 가진 사용자가 동아리 채널을 생성한다.
Pre-Conditions	유저는 관리자 권한이 부여된 계정으로 로그인을 성공해야 한다.
Post-Conditions	채널 정보가 DB에 등록되고, 유저는 추가된 채널에 접근할 수 있게 된다.
Primary Flow	1. 관리자가 채널 리스트 보기 버튼을 눌러 리스트 메뉴를 연다. 1.1. 만약 현재 접속한 계정이 관리자 계정이 아니라면, 채널 추가 버튼을 활성화하지 않는다. 1.2. 관리자는 채널 추가 메뉴를 선택하고, 애플리케이션은 새 채널 추가 뷰를 보여준다. 1.3. 관리자는 채널 추가 폼에 추가하려는 채널의 이름과 장소 정보를 기입한다. 1.3.1. 만약 채널 추가 폼에 입력되지 않은 정보가 있다면, 애플리케이션은 유저에게 경고 메시지를 전달한다. 1.3.2. 등록 폼에 모든 필수 정보가 입력되면, 해당 정보는 DB 서버로 송신된다. 1.3.2.1. 만약 수신된 데이터와 일치하는 내용의 데이터가 이미 서버에 존재하면, DB 서버는 유저에게 경고 메시지를 전달한다. 1.3.2.2. 채널 정보를 DB 서버에 추가하여 새 채널을 추가하고, 서버는 애플리케이션을 통해 새 채널이 정상적으로 추가되었음을 보여주는 메세지를 전달한다.

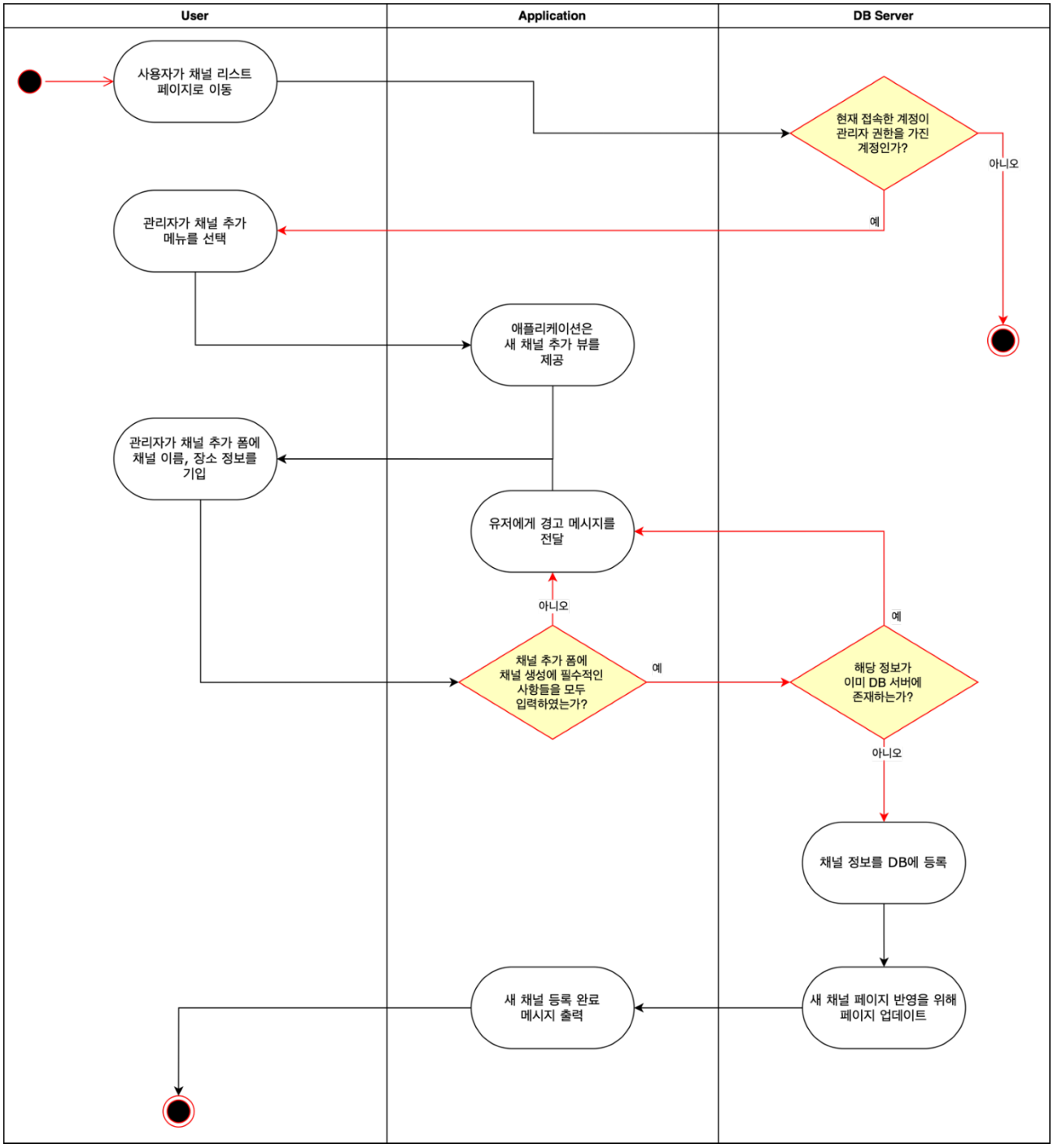

Figure 1-4

1.2.4. Use Case: 스케줄 예약 요청

Table 1-4

Use Case	4
Case Name	스케줄 예약 요청
Actors	유저(게스트), 모바일 애플리케이션, 백엔드 애플리케이션 서버, DB 서버
Description	게스트가 스케줄 예약을 요청한다.
Pre-Conditions	게스트가 현재 로그인 상태여야 한다.
Post-Conditions	게스트가 예약 요청한 스케줄이 관리자의 스케줄 승인 페이지로 이동한다
Primary Flow	1. 게스트가 동아리 채널의 타임 테이블 페이지로 이동한다.

	2. 게스트가 타임 테이블 페이지에서 스케줄 예약 버튼을 누른다. 3. 애플리케이션은 스케줄 예약 페이지를 보여준다. 4. 게스트는 스케줄 예약 폼에서 예약일자와 예약시간을 선택하고, 예약명과 비고사항을 입력한다. 5. 게스트가 확인 및 제출 버튼을 누른다. 5.1. 만약 비고사항을 제외한 예약일자, 예약시간이 선택되지 않았거나, 예약명이 입력되지 않았다면, 애플리케이션은 유저에게 경고 메시지를 전달한다. 5.2. 모든 필수 정보가 정상적으로 입력되었다면, 해당 예약 데이터가 DB 서버에 저장되고, 관리자용 예약 관리 페이지에 해당 정보가 업데이트된다. 5.3. 서버는 애플리케이션을 통해 새 채널이 정상적으로 추가되었음을 보여주는 메세지를 전달한다.

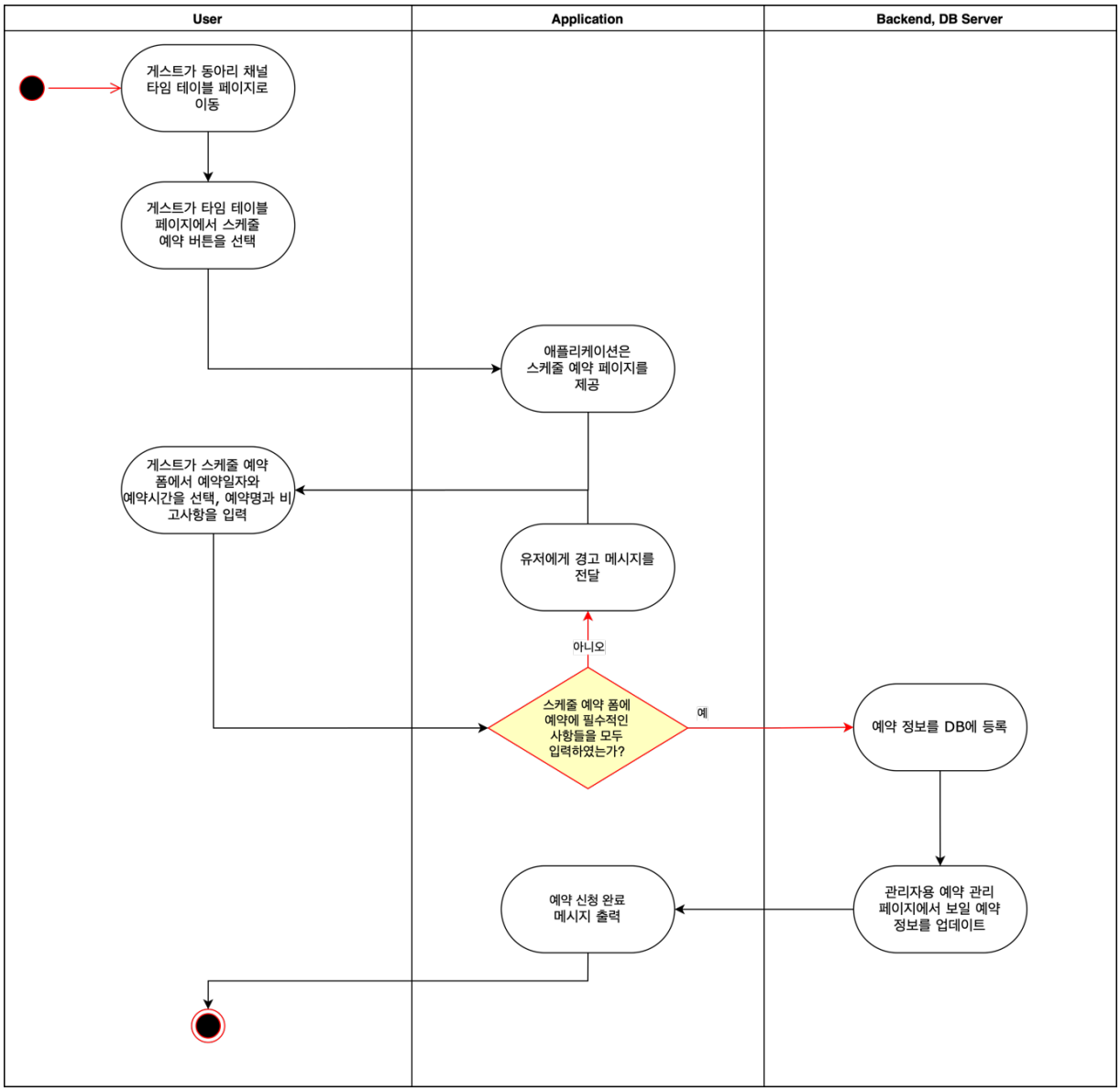

Figure 1-5

1.2.5. Use Case: 스케줄 상세 정보 조회 및 삭제

Table 1-5

Use Case	5
Case Name	스케줄 상세정보 조회 및 삭제
Actors	유저(관리자, 게스트), 모바일 애플리케이션, 백엔드 애플리케이션 서버, DB 서버
Description	유저가 예약 상세정보를 조회하고, 게스트가 현재 예약된 스케줄을 삭제한다.
Pre-Conditions	유저가 현재 로그인 상태이고, 현재 조회 가능한 예약이 존재해야 한다.
Post-Conditions	예약 삭제 신청에 성공하면 게스트의 예약이 삭제된다.
Primary Flow	1. 게스트가 동아리 채널의 타임 테이블 페이지로 이동한다. 2. 게스트가 현재 타임 테이블상의 예약 블록을 선택한다. 3. 애플리케이션은 스케줄 예약 상세 정보 페이지를 보여준다. 4. 게스트는 스케줄 예약 상세 정보 페이지에서 예약 삭제 버튼을 선택한다. 4.1. 만약 서버에 등록된 예약자 정보와 현재 유저의 정보가 일치하지 않으면, 애플리케이션은 유저에게 경고 메시지를 보낸다. 4.2. 만약 서버에 등록된 예약자 정보와 현재 유저의 정보가 일치하면, 해당 예약 데이터가 DB 서버에서 삭제된다. 4.2.1. 서버는 애플리케이션을 통해 예약 정보가 삭제되었음을 보여주는 메세지를 전달하고, 타임 테이블을 업데이트한다.

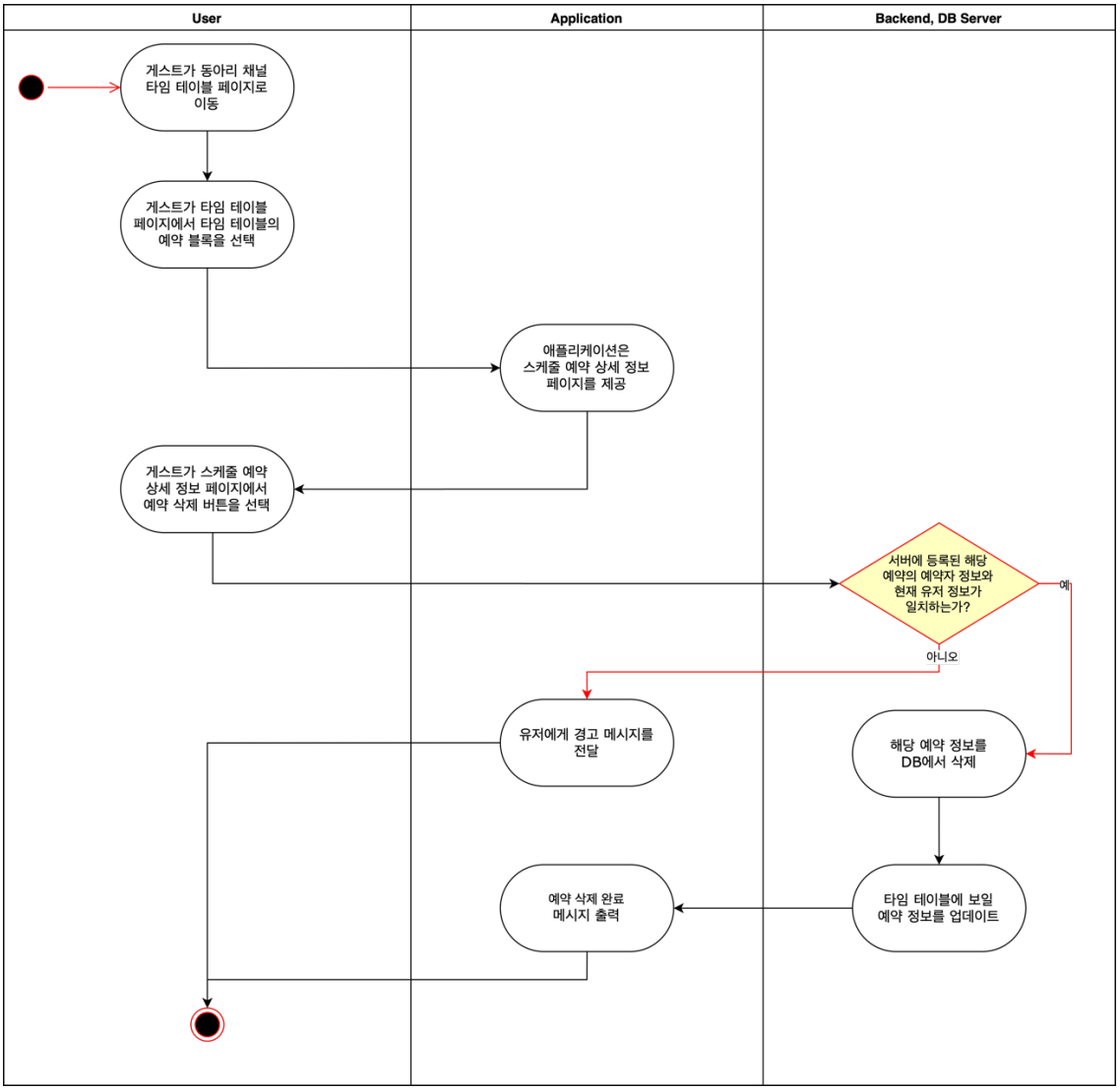

Figure 1-6

1.2.6. Use Case: 관리자의 스케줄 승인/거절

Table 1-6

Use Case	6
Case Name	관리자의 스케줄 승인/거절
Actors	유저(관리자), 모바일 애플리케이션, 백엔드 애플리케이션 서버, DB 서버
Description	관리자가 게스트가 요청한 스케줄을 승인하거나 거절할 수 있다
Pre-Conditions	유저는 관리자 권한이 부여된 계정으로 로그인을 성공해야 하며, 현재 승인을 대기 중인 예약이 존재해야 한다.
Post-Conditions	예약의 승인 여부에 따라 해당 예약 정보가 타임 테이블에 반영된다.
Primary Flow	1. 사용자가 로그인을 한다. 2. 접속한 계정의 권한을 확인한다. 2.1. 만약 현재 접속한 계정이 관리자 계정이 아닐 경우 2.1.1. 채널 예약 관리 페이지 버튼을 활성화하지 않는다. 2.2. 만약 현재 접속한 계정이 관리자 계정일 경우 2.2.1. 관리자는 채널 예약 관리 페이지에 접속하고, 애플리케이션은 채널 관리 페이지를 보여

	준다. 2.2.2. 관리자는 게스트의 예약 요청 정보들을 확인하고, 해당 예약을 승인하거나 거절한다. 2.2.3. 예약 요청의 승인 여부 2.2.3.1. 만약 관리자가 예약 요청을 승인하면, 백엔드 애플리케이션 서버에서 게스트에게 승인 알림을 전송한다. 2.2.3.1.1. DB 서버는 해당 스케줄 정보를 저장한다. 2.2.3.1.2. 백엔드 애플리케이션 서버에서 예약 정보를 업데이트한다. 2.2.3.2. 만약 관리자가 요청을 거절하면 모바일 애플리케이션 서버에서 게스트에게 거절 알림을 전송한다. 2.2.3.2.1. 백엔드 애플리케이션 서버에서 예약 정보를 업데이트한다.

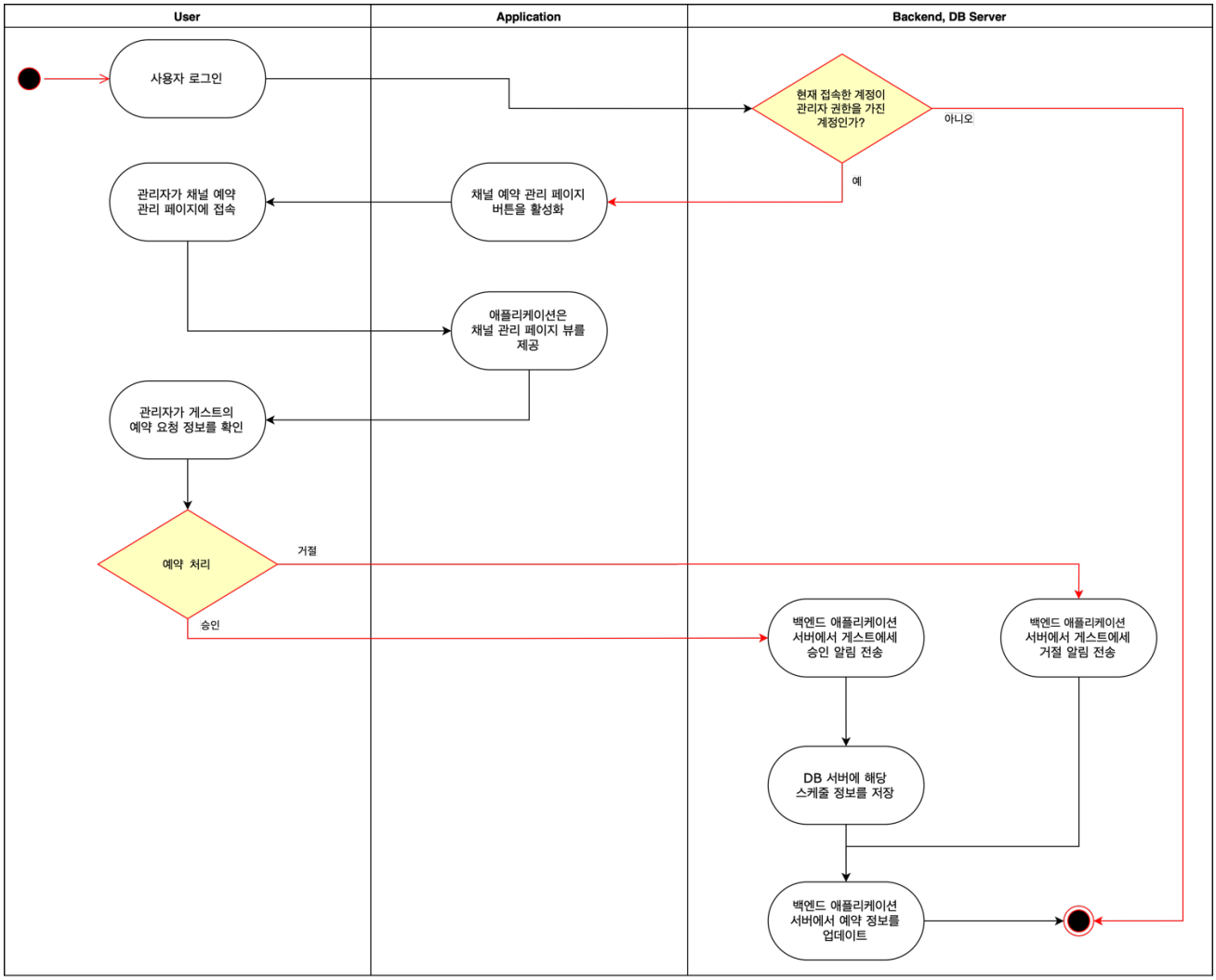

Figure 1-7

2. Architectural Design

2.1. Frontend Layers[1]

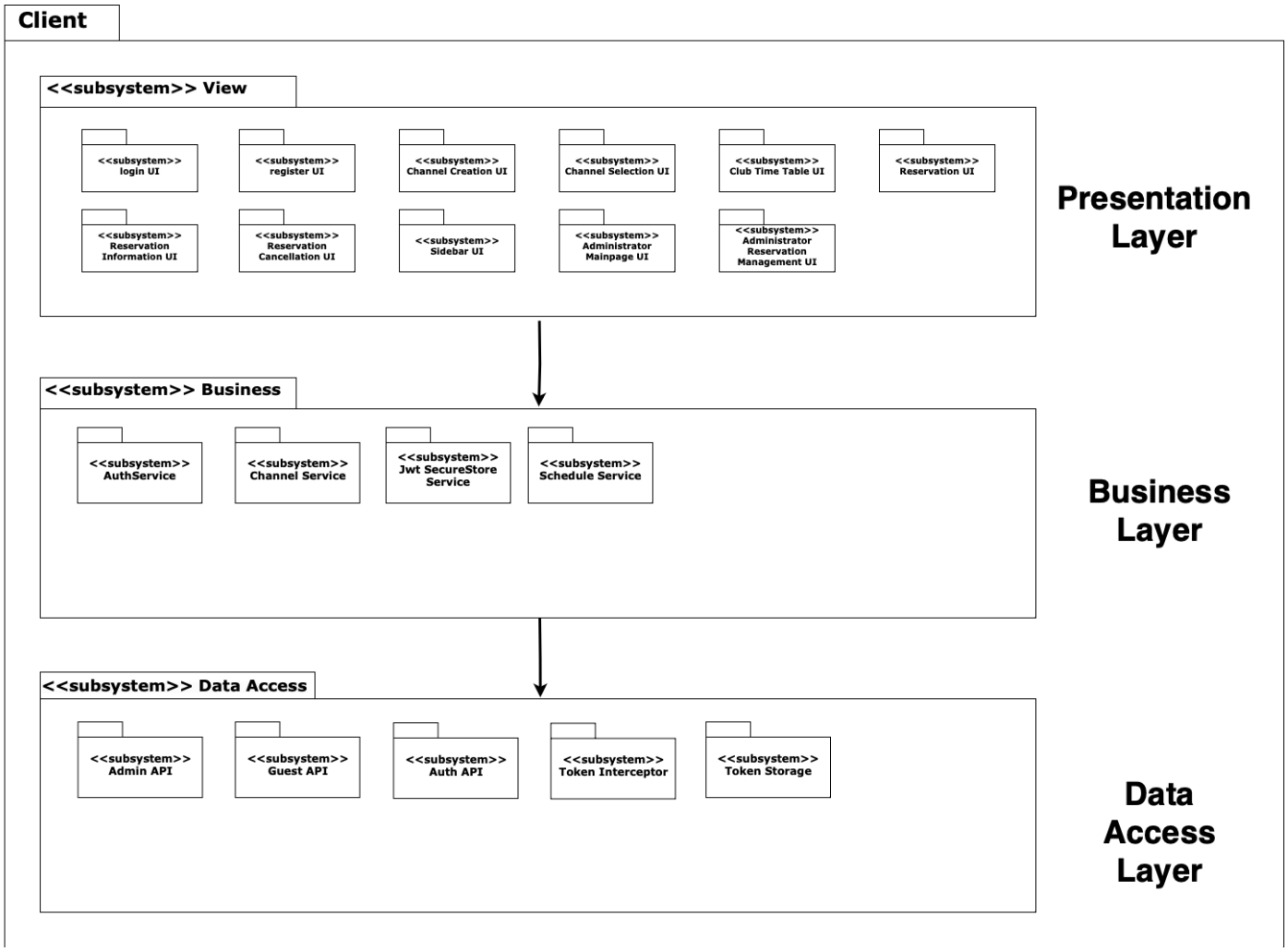

Figure 2-1 Client System Architecture Diagram

2.1.1. Presentation Layer

- 외부에 화면을 보여주는 View 영역
- 외부에서 사용자에게 입력을 받는 영역

실제 앱에서 보여지는 대부분의 영역으로 구성되는 계층이다.

2.1.2. Business Layer

- 화면에 나타낼 State를 저장해두는 영역
- 사용자의 입력에 따라 데이터를 불러와 State를 업데이트 하는 영역

앱을 표시할 때 필요한 데이터와 비즈니스 로직을 담아두는 계층이다.

2.1.3. Data Access Layer

- 서버에서 직접 요청을 받아오는 영역
- 클라이언트 디바이스에 저장된 데이터를 불러오는 영역

즉, 앱을 구동하기 위해 필요한 데이터를 받아오는 레이어이다.

---

[1] (O'Reilly Media, Inc.)

2.2. Backend Layers[2]

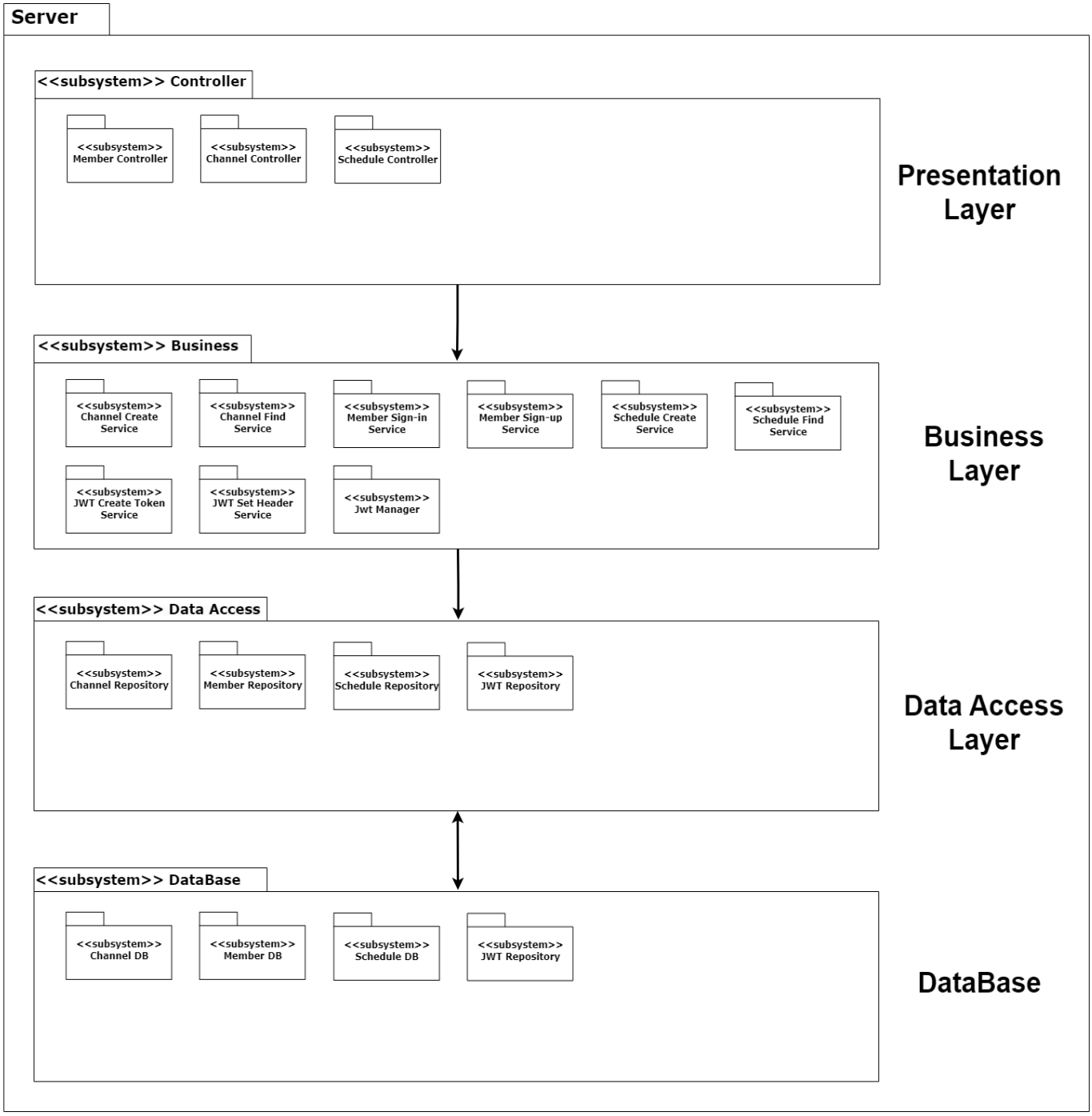

Figure 2-2 Server System Architecture Diagram

2.2.1. Presentation Layer

- 외부에 요청, 응답을 담당하는 계층
- 외부 변화에 민감한 외부 의존성이 높은 영역
- 외부 영역에 대한 처리를 담당하는 코드나 요청, 응답을 담당하는 클래스가 모두 여기에 속한다.

2.2.2. Business Layer

- 비즈니스 로직을 담당하는 클래스

[2] (지속 성장 가능한 소프트웨어를 만들어가는 방법, 날짜 정보 없음)

- 전체적인 비즈니스 로직 흐름을 읽을 수 있도록 작성한다.
- 상세한 구현은 각각의 도메인에 위임한다.

2.2.3. Data Access Layer

- Business Layer가 DB에 접근할 수 있도록 인터페이스를 제공한다.
- 이때 레이어는 위에서 아래로 순방향으로만 참조 되어야 한다. 참조 방향이 역류되지 않게 한다.
- 레이어의 참조가 하위 레이어를 건너뛰지 않는다. 동일 레이어 간에는 서로 참조하지 않아야 한다.

3. Design Analysis

3.1. Frontend Design Class Diagram

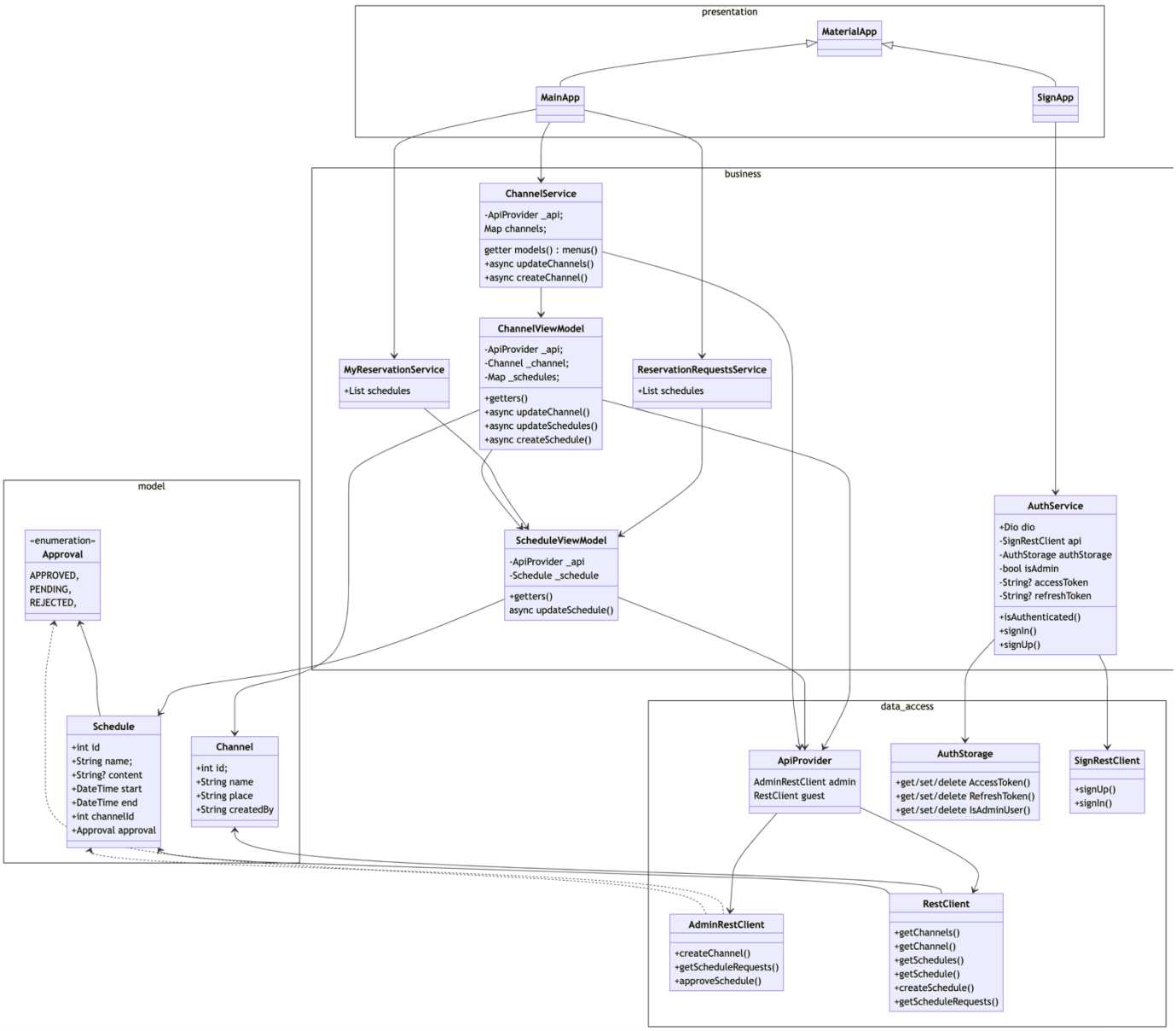

Figure 3-1 Frontend Design Class Diagram

해당 다이어그램은 프론트엔드 애플리케이션의 디자인 클래스 다이어그램으로, 크게 앞서 2.1의 Layered Architecture 에서 언급한 Data Access Layer, Business Layer, Presentation Layer로 구성되어 있다.

- Data Access Layer에서는 Retrofit 라이브러리로 제작된 API RestClient에 AuthService를 통해 사용자의

Token이 인증된 Dio 인스턴스를 Injection하여 ApiProvider 인터페이스를 제공한다.

- Business Layer에서는 ApiProvider 인터페이스를 통해서 백엔드 데이터에 액세스하고, 결과를 저장하는 역할을 수행한다.
- Presentation Layer에서는 최종 사용자에게 보여질 정보를 적합한 UI를 통해 제공하며, 보여질 정보를 Business Layer와 Binding하여 state가 변경되는대로 화면에 보여지게 한다.
- 추가적으로, Model은 서버 Domain과 일치하도록 하였다.

3.2. Backend Domain Diagram

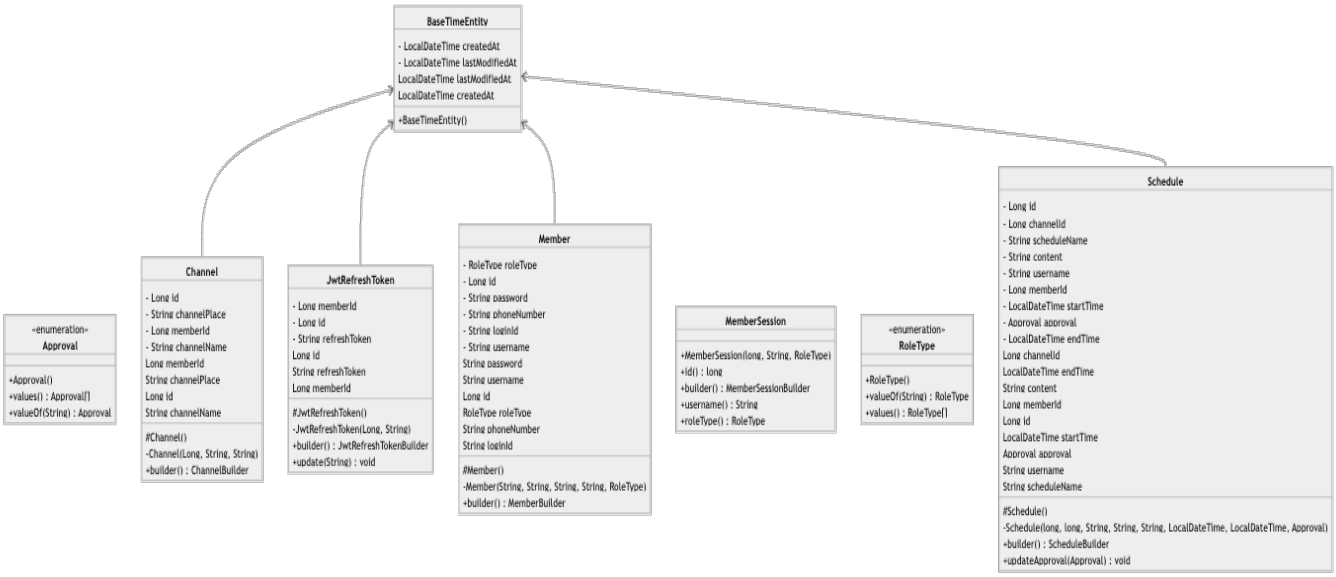

Figure 3-2 Backend Domain Diagram

이 클래스 다이어그램은 도메인 계층의 다이어그램을 나타낸다. 각 클래스에는 해당 클래스의 구조와 기능을 나타내는 메소드와 변수가 포함되어 있다.

- Approval, RoleType: 이들은 열거형 클래스로, 특정 변수에 허용되는 값의 집합을 정의한다.
- BaseTimeEntity: 이 클래스는 시간 관련 변수를 가지고 있으며, 다른 클래스들 (Channel, JwtRefreshToken, Member, Schedule)은 이 클래스를 상속받아 시간 정보를 활용한다.
- Channel: 이 클래스는 채널에 대한 정보를 나타낸다. 'id', 'channelPlace', 'memberId', 'channelName' 등의 변수를 가지고 있다.
- JwtRefreshToken: 이 클래스는 JWT(refresh token)에 대한 정보를 나타낸다. 'id', 'refreshToken', 'memberId'와 같은 변수를 가지고 있다.
- Member: 이 클래스는 회원에 대한 정보를 나타낸다. 'id', 'password', 'phoneNumber', 'loginId', 'username', 'roleType' 등의 변수를 가지고 있다.
- MemberSession: 이 클래스는 회원 세션에 대한 정보를 나타낸다. 'id', 'username', 'roleType' 등의 변수를 가지고 있다.
- Schedule: 이 클래스는 일정에 대한 정보를 나타낸다. 'id', 'channelId', 'scheduleName', 'content', 'username', 'memberId', 'startTime', 'endTime', 'approval' 등의 변수를 가지고 있다.

3.3. Backend Presentation/Business/Data Access Layer Diagram (Member, Channel, Schedule)

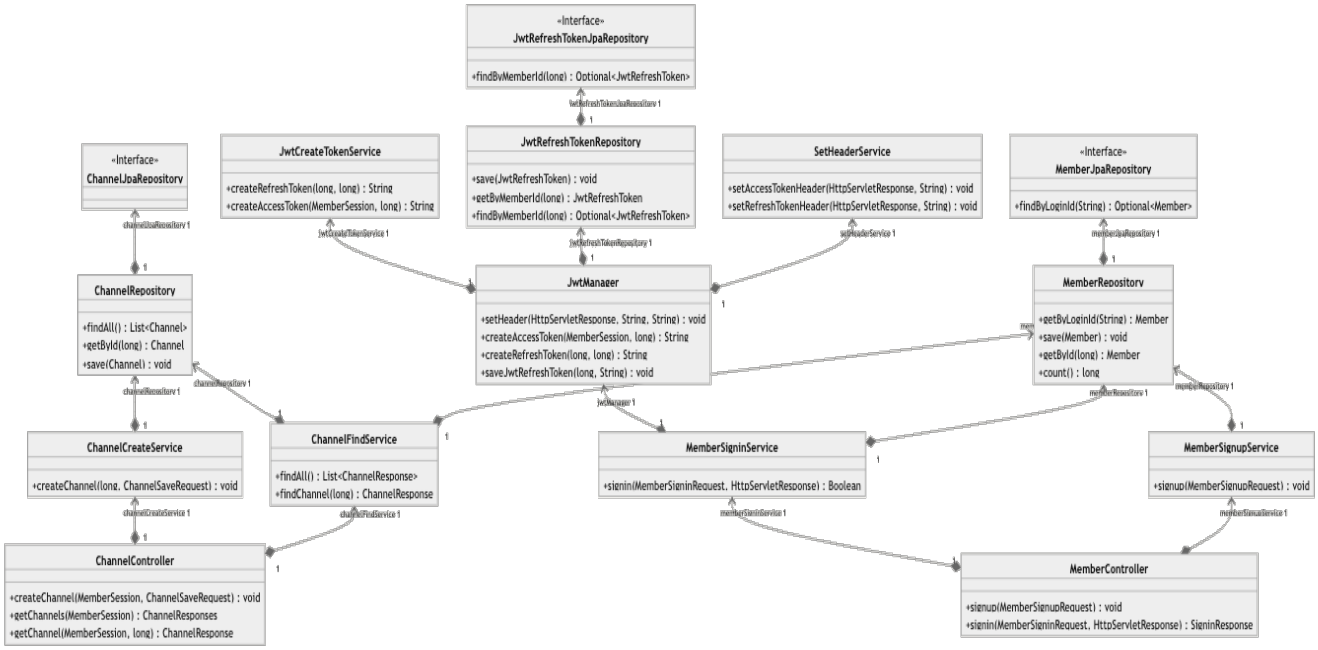

Figure 3-3 Backend Presentation/Business/Data Access Layer Diagram (Member, Channel)

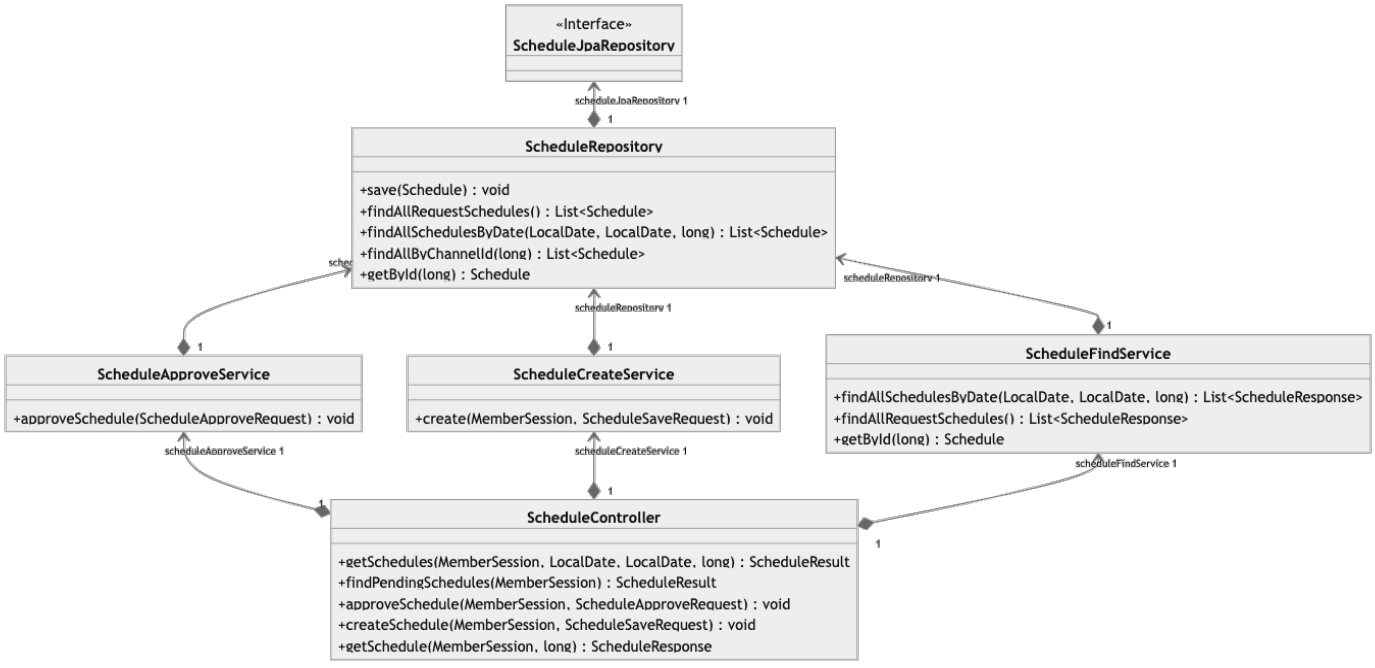

Figure 3-4 Backend Presentation/Business/Data Access Layer Diagram

이 클래스 다이어그램은 웹 애플리케이션의 RESTful API를 처리하는 여러 서비스와 컨트롤러, 그리고 이들이 상호작용하는 방식을 보여준다.

- Controller 클래스 (MemberController, ChannelController, ScheduleController) Presentation Layer에 해당

하는 클래스들로 사용자의 요청을 받아 처리하고, 적절한 서비스를 호출하여 비즈니스 로직을 실행한다. 이후 결과를 사용자에게 응답한다.

- Service 클래스 (ChannelCreateService, ChannelFindService, JwtManager, MemberSigninService, MemberSignupService 등) Business Layer에 해당하는 클래스들로 각각의 서비스 클래스는 특정 비즈니스 로직을 수행한다. 이 과정에서 이 클래스들은 필요한 데이터를 Repository로부터 가져올 수 있다.
- Repository 클래스 (MemberRepository, ChannelRepository, ScheduleRepository, JwtRefreshTokenRepository 등) 이들은 데이터베이스와의 직접적인 상호작용을 담당하며, 데이터를 저장하거나 검색하는 역할을 한다. 또한 JpaRepository는 Spring Data JPA의 인터페이스로 각 도메인의 CRUD 작업을 추상화한다. Repository 클래스들은 JpaRepository를 사용하여 데이터베이스에 접근한다.

4. Use Case Realization

4.1. Sign-up

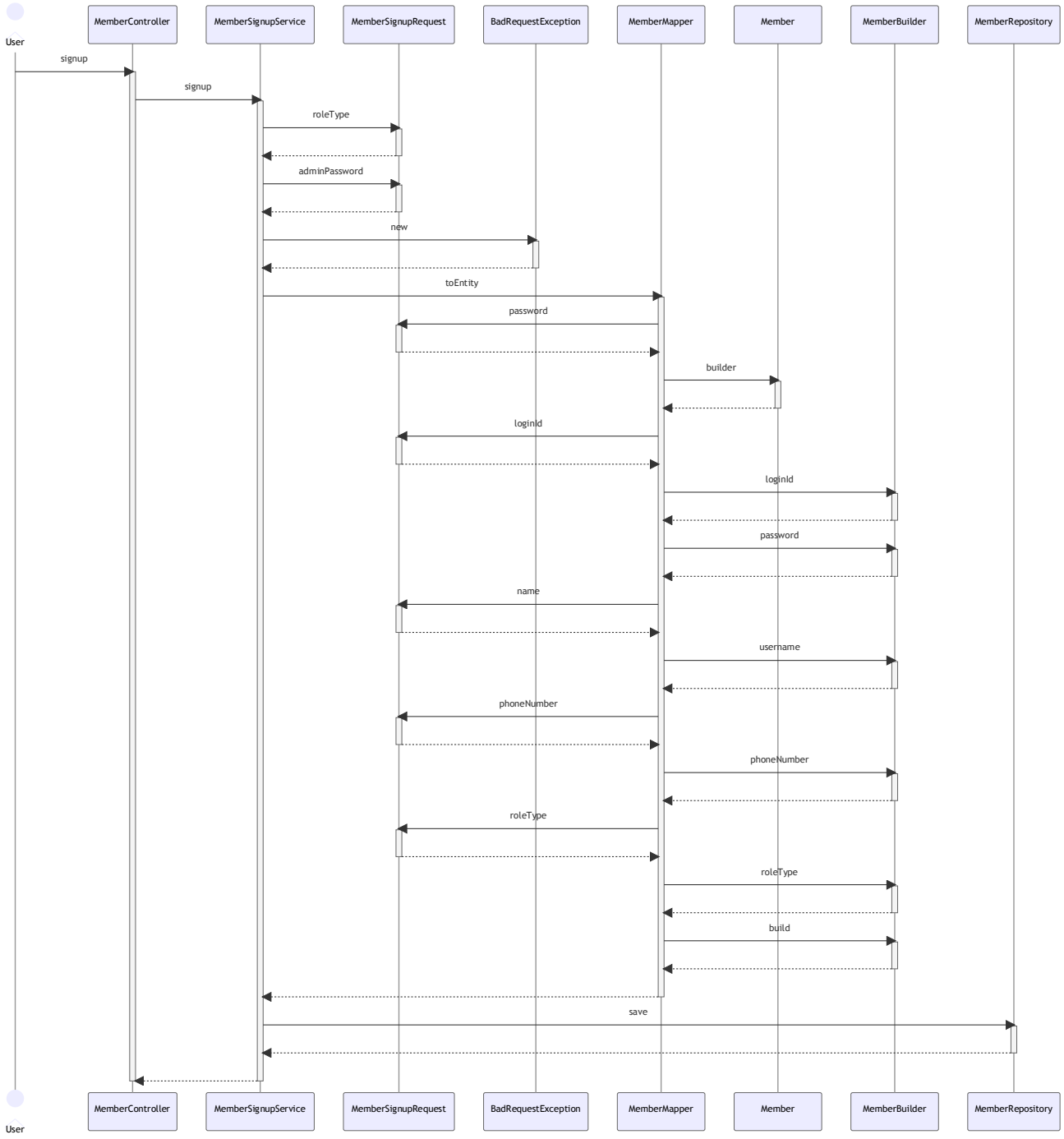

Figure 4-1 Sign-up sequence diagram

이 시퀀스 다이어그램은 유저가 서비스에 회원가입 시의 일련의 흐름을 나타낸다.

- MemberController: 이 클래스에서는 회원 가입 요청을 받고, MemberSignupService에 처리를 위임한다.
- MemberSignupService: 이 클래스에서는 signup() 메소드를 통해 회원 가입을 처리하는 역할을 한다.
  - signup(): 해당 메소드는 회원 가입 요청을 받아 처리한다.
- MemberSignupRequest: 이 클래스에서는 회원 가입 요청 정보를 담고 있다.
- BadRequestException: 이 클래스는 예외 클래스로, 잘못된 요청이 있을 경우 에러를 발생시키는 역할을 한다.
- MemberMapper: 이 클래스에서는 toEntity() 메소드를 통해 회원 가입 요청 정보를 회원 엔티티로 변환하는 역할을 수행한다.
  - toEntity(): 이 메소드에서는 회원 가입 요청 정보를 회원 엔티티로 변환한다.
- Member: 이 클래스는 회원 정보를 담고 있는 엔티티 클래스로, 회원 객체로 인스턴스화된다.
- MemberBuilder: 이 클래스는 회원 객체를 생성하고 필드 값을 설정하는 빌더 클래스로 기능한다.
- MemberRepository: 이 클래스는 save() 메소드를 통해 회원 정보를 저장하고 조회하는 역할을 수행하는 클래스이다.
  - save(): MemberRepository 클래스 내에서 회원 정보를 저장하는 역할을 수행하는 메소드이다.

각 클래스와 메소드는 회원 가입 요청을 처리하고, 필요한 정보를 변환하며, 회원 객체를 생성하여 저장한다. 이를 통해 사용자의 회원 가입 요청에 대한 처리를 수행한다.

4.2. Sign-in

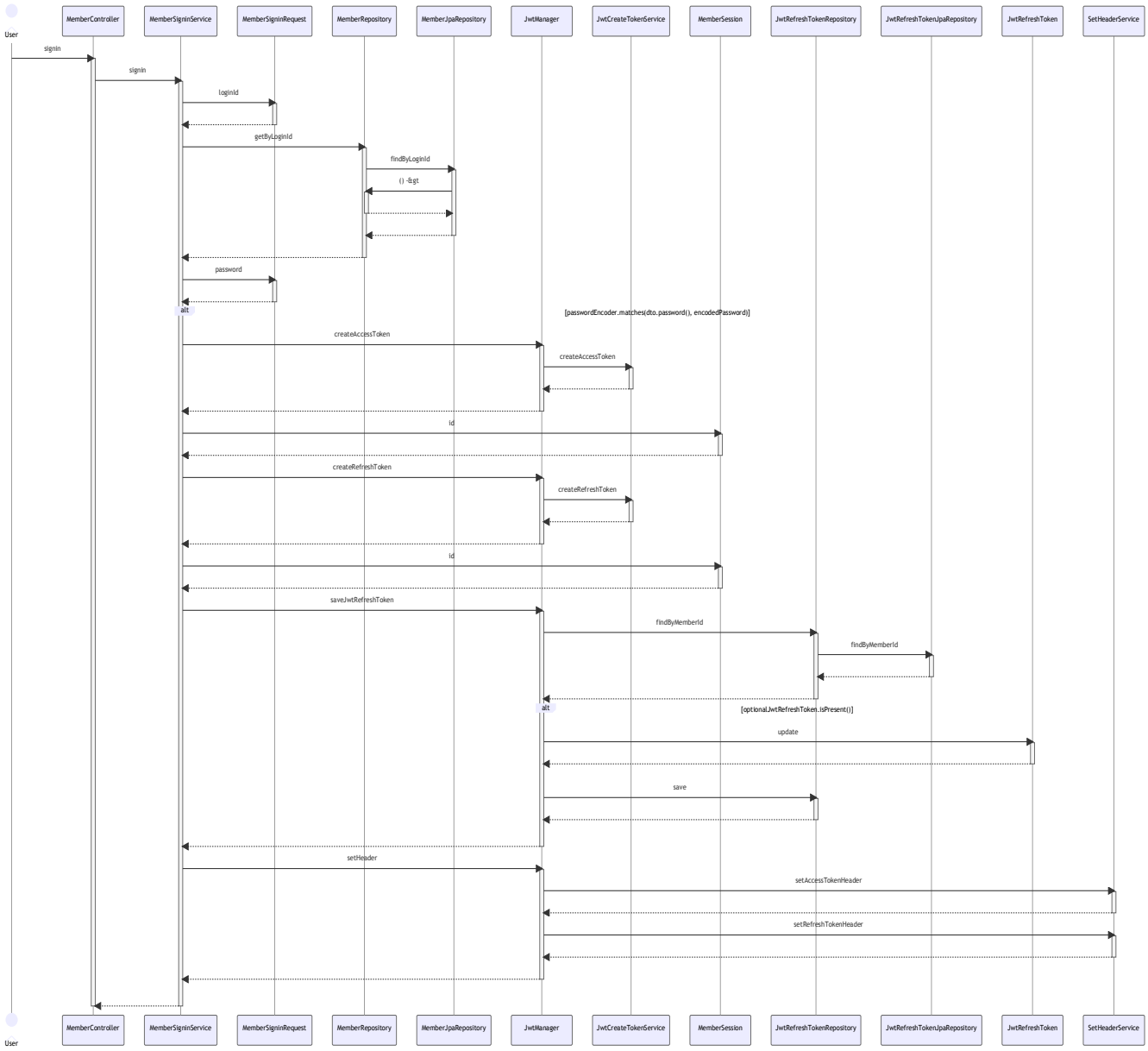

Figure 4-2 Sign-in sequence diagram

이 시퀀스 다이어그램은 사용자가 서비스에 로그인하는 일련의 과정을 나타낸다.

- MemberController: 이 클래스에서는 signin() 메소드로 로그인 요청을 받고, MemberSigninService에 그 처리를 위임한다.
- MemberSigninService: 이 클래스는 signin() 메소드를 통해 회원 로그인 요청을 처리하고, 회원 정보를 조회하고 인증을 수행하는 역할을 한다.
  - signin(): 해당 메소드는 회원 로그인 요청을 처리하는 역할을 한다.
- MemberRepository: 이 클래스는 getByLoginID() 메소드를 통해 회원 정보를 저장하고 조회하는 역할을 한다.
  - getByLoginId(): 이 메소드에서는 로그인 ID에 해당하는 회원 정보를 조회한다.
- NotFoundException: 이 클래스는 예외 클래스로, 조회된 회원 정보가 없을 경우 에러를 발생시키는 역할을 수행한다.
- Member: 이 클래스는 회원 정보를 담고 있는 엔티티 클래스로, 회원 객체로 인스턴스화된다.
  - getPassword(): 이 메소드는 회원의 비밀번호를 가져오는 역할을 수행한다.

- getRoleType(): 이 메소드는 회원의 역할 유형을 가져온다.

- MemberSigninRequest: 이 클래스는 회원 로그인 요청에서 필요한 정보를 포함한다.
- MemberMapper: 이 클래스에서는 toMemberSession 메소드를 통해 회원 정보를 MemberSession으로 변환하는 역할을 수행한다.
  - toMemberSession(): 이 메소드는 MemberMapper 클래스 내에서 회원 정보를 MemberSession으로 변환하는 역할을 수행한다.
- MemberSession: d이 클래스는 로그인 된 회원의 세션 정보를 담고 있다.
- JwtManager: 이 클래스는 JWT 토큰을 생성하고 처리하는 역할을 수행한다.
  - createAccessToken(): 액세스 토큰을 생성하는 메소드이다.
  - createRefreshToken(): 리프레시 토큰을 생성하는 메소드이다.
  - saveJwtRefreshToken(): 리프레시 토큰을 저장하는 메소드이다.
  -
- JwtRefreshTokenRepository: 이 클래스는 리프레시 토큰을 저장하고 조회하는 역할을 수행한다.
  -
- JwtCreateTokenService: 이 클래스에서는 JWT 토큰 생성에 관련된 기능을 수행한다.
- BadRequestException: 이 클래스는 잘못된 요청 예외를 처리하는 역할을 수행하는 클래스이다.
- SigninResponse: 이 클래스는 로그인 응답을 담고 있는 클래스다.

이와 같이 각 클래스와 메소드는 로그인 과정에서 필요한 역할을 수행하며 사용자의 로그인 요청을 처리하고, 액세스 토큰과 리프레시 토큰을 생성하여 반환한다.

4.3. Channel creation

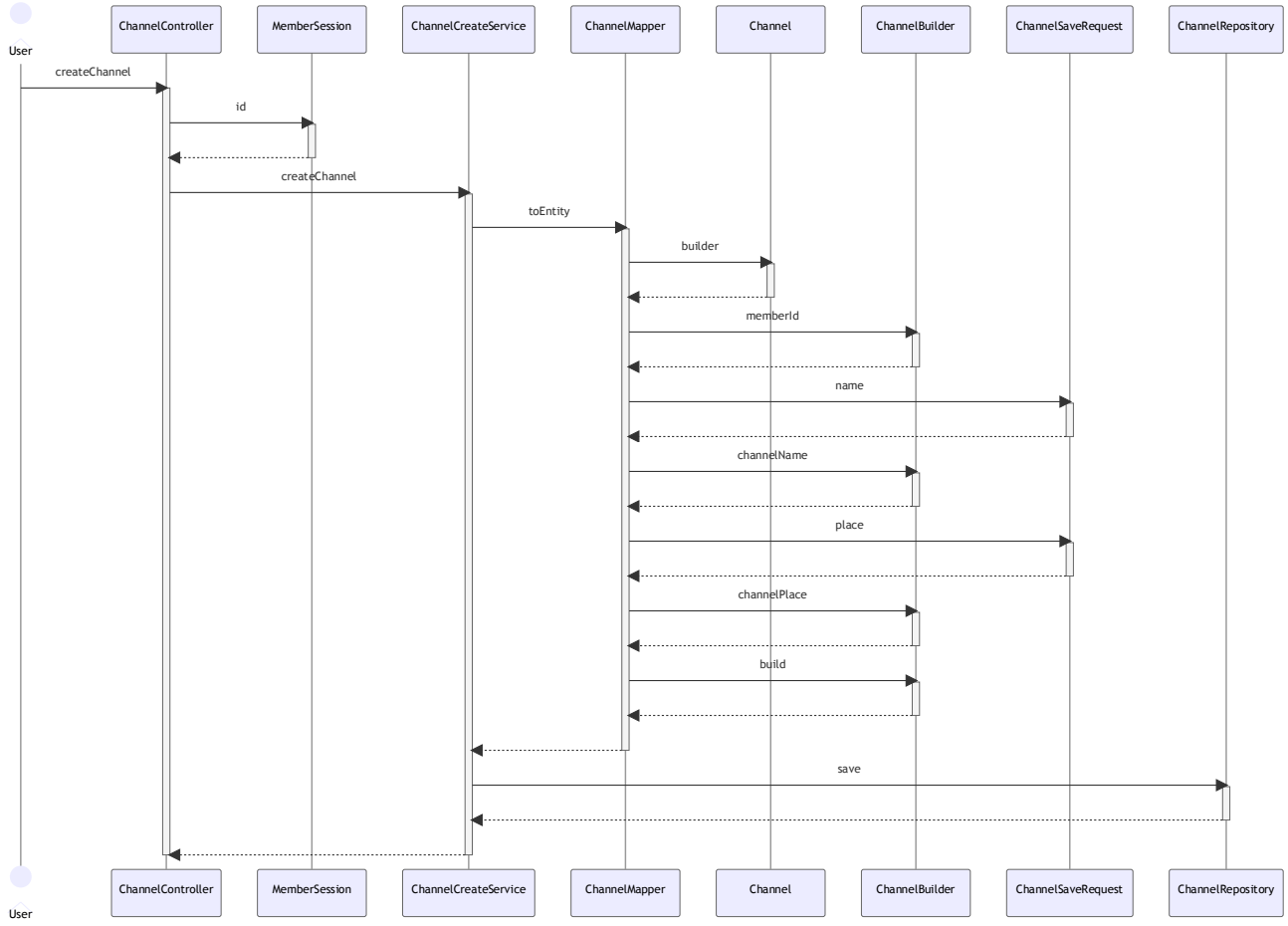

Figure 4-3 Channel creation sequence diagram

이 시퀀스 다이어그램은 관리자가 채널을 생성하는 일련의 과정을 나타낸다.

- ChannelController: 이 클래스에서는 채널 생성 요청을 받고, MemberSession에서 회원 ID를 가져와 처리한다.
- MemberSession: 이 클래스는 회원 세션 정보를 담고 있다.
- ChannelCreateService: 이 클래스에서는 createChannel() 메소드를 통해 채널 생성 요청을 처리하고 ChannelMapper를 사용하여 채널 엔티티를 생성한다.
  - createChannel(): 이 메소드는 채널 생성 요청을 처리한다.
- ChannelMapper: 이 클래스를 통해 채널 정보를 채널 엔티티로 변환하는 역할을 수행한다.
  - toEntity(): 이 메소드는 채널 정보를 채널 엔티티로 변환하는 역할을 수행한다.
- Channel: 이 클래스는 채널 정보를 담고 있는 엔티티이다.
- ChannelBuilder: 이 클래스는 채널 엔티티를 생성하는 빌더 클래스이다.
- ChannelSaveRequest: 이 클래스는 채널 생성 요청에서 필요한 정보를 담고 있다.
- ChannelRepository: 이 클래스를 통해 채널 정보를 저장하고 조회하는 역할을 수행하는 클래스이다.
  - save(): 이 메소드에서는 채널 정보를 저장한다.

이와 같이 각 클래스와 메소드는 채널 생성 과정에서 필요한 역할을 담당하며, 사용자의 요청을 처리하여 채널 엔티티를 생성하고 저장한다.

4.4. Single channel find service

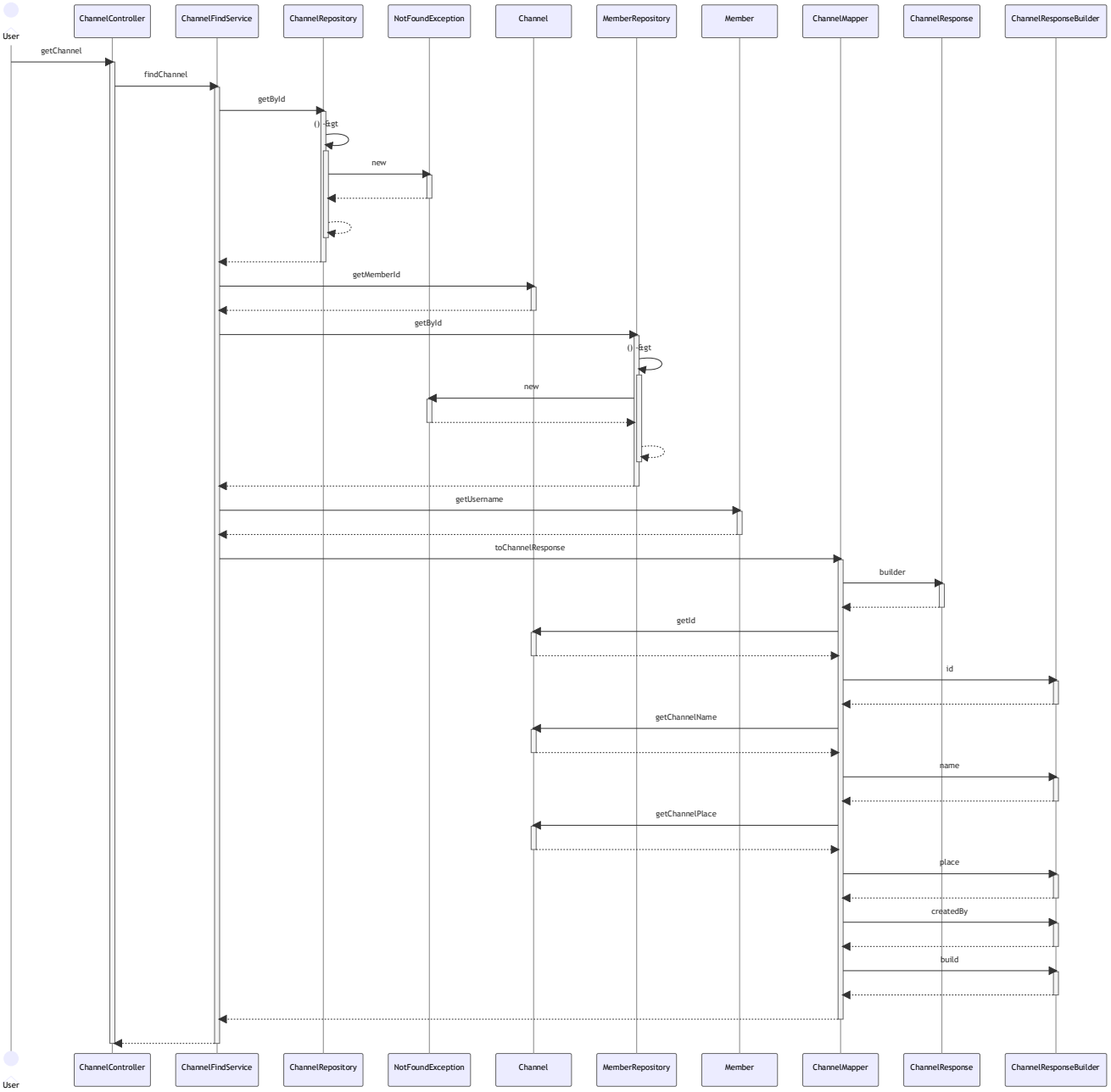

Figure 4-4 Single channel find service sequence diagram

이 시퀀스 다이어그램은 사용자가 채널 정보를 조회하는 일련의 과정을 나타낸다.

- ChannelController: 이 클래스는 채널 정보 조회 요청을 받고 ChannelFindService에 처리를 위임하는 역할을 한다.
- ChannelFindService: 이 클래스에서는 채널 정보 조회를 처리하고 ChannelRepository와 MemberRepository를 사용하여 채널과 회원 정보를 조회한다.
  - findChannel(): 이 메소드는 채널 정보 조회 요청을 처리하는 역할을 수행한다.
- ChannelRepository: 이 클래스는 getById() 메소드를 통해 채널 정보를 저장하고 조회하는 역할을 수행한다.
  - getById(): 이 메소드는 ChannelRepository 클래스 내에서 채널 ID에 해당하는 채널 정보를 조회한다.
- NotFoundException: 이 클래스는 예외 클래스로, 조회된 채널이나 회원 정보가 없을 경우 에러를 발생

시킨다.

- MemberRepository: 이 클래스에서는 getById() 메소드를 통해 회원 정보를 저장하고 조회하는 역할을 수행한다.
  - getById(): 이 메소드는 MemberRepository 클래스 내에서 회원 ID에 해당하는 회원 정보를 조회한다.
- Member: 이 클래스는 회원 정보를 담고 있는 엔티티 클래스로, 회원 객체로 인스턴스화된다.
  - getUsername(): 이 메소드는 회원의 사용자명을 가져오는 역할을 수행한다.
- ChannelMapper: 이 클래스는 채널 정보를 채널 응답(ChannelResponse)로 변환하는 역할을 수행한다.
  - toChannelResponse(): 이 메소드는 ChannelMapper 클래스 내에서 채널 정보를 채널 응답으로 변환하는 역할을 수행한다.
- ChannelResponse: 이 클래스는 채널 정보를 담고 있는 응답 클래스이다.
- ChannelResponseBuilder: 이 클래스는 채널 응답을 구성하는 빌더 클래스이다.

이와 같이 각 클래스와 메소드는 채널 정보 조회 과정에서 필요한 역할을 담당하며, 사용자의 요청에 따라 채널과 회원 정보를 조회하여 채널 응답을 생성하여 반환한다.

4.5. Total channel find service

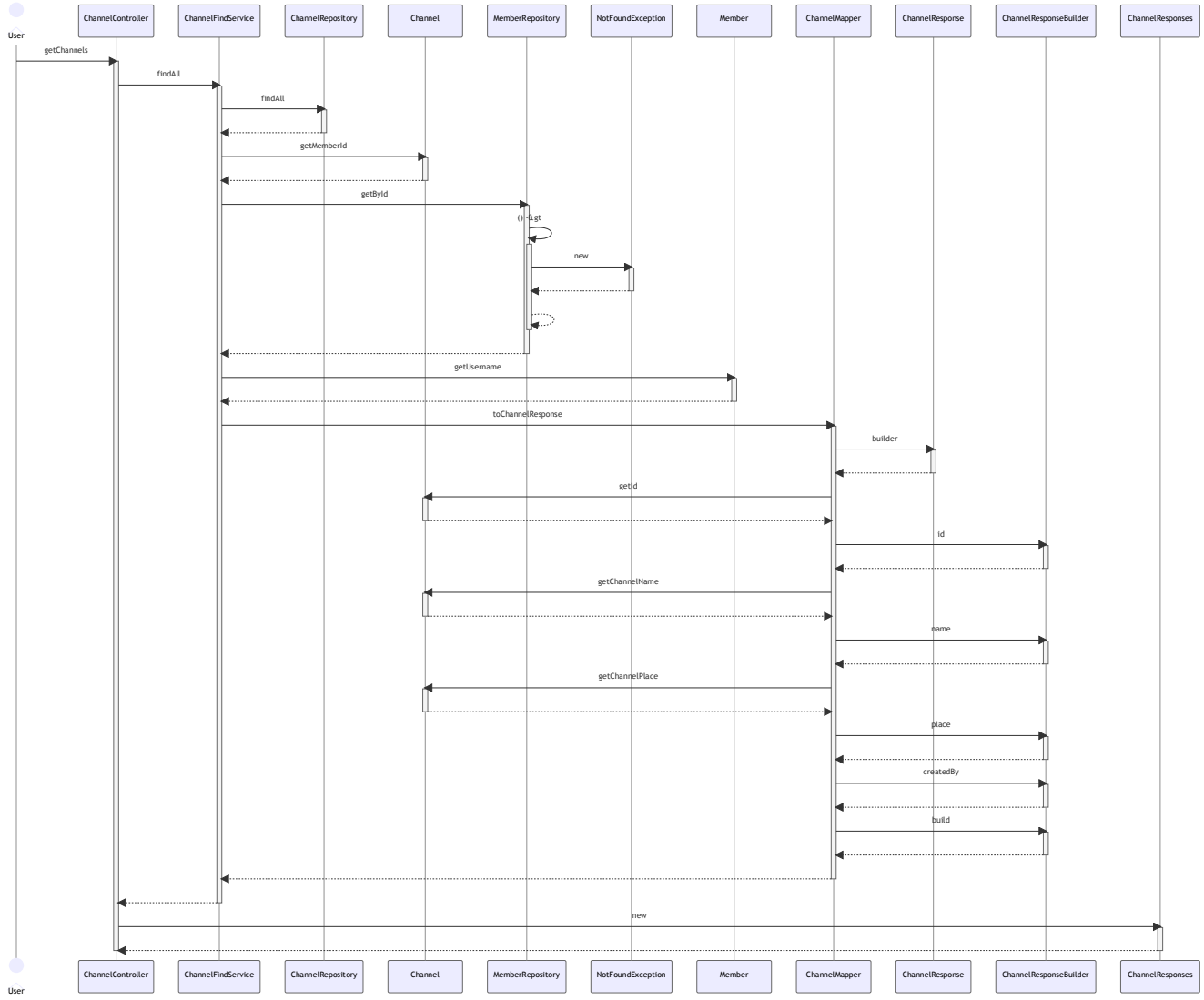

Figure 4-5 Total channel find service sequence diagram

이 시퀀스 다이어그램은 사용자가 모든 채널 정보를 조회한다.

- ChannelController: 이 클래스는 채널 정보 조회 요청을 받고 ChannelFindService에 처리를 위임하는 역할을 한다.
- ChannelFindService: 이 클래스에서는 findAll() 메소드를 통해 모든 채널 정보를 조회하고, MemberRepository를 사용하여 회원 정보를 조회한다.
  - findAll(): 이 메소드는 모든 채널 정보를 조회하는 기능을 수행한다.
- ChannelRepository: 이 클래스에서는 findAll() 메소드를 통해 채널 정보를 저장하고 조회하는 역할을 수행한다.
  - findAll(): 이 메소드는 모든 채널 정보를 조회하는 기능을 수행한다.
- NotFoundException: 이 클래스는 예외 클래스로, 조회된 회원 정보가 없을 경우 에러를 발생시킨다.
- MemberRepository: 이 클래스는 getById() 메소드를 통해 회원 정보를 저장하고 조회하는 역할을 수행한다.
  - getById(): 이 메소드는 회원 ID에 해당하는 회원 정보를 조회하는 기능을 수행한다.
- Member: 이 클래스는 회원 정보를 담고 있는 엔티티 클래스이다.
  - getUsername(): 이 메소드는 회원의 사용자명을 가져오는 역할을 수행한다.
- ChannelMapper: 이 클래스에서는 toChannelResponse() 메소드를 통해 채널 정보를 채널 응답(ChannelResponse)로 변환하는 역할을 수행한다.
  - toChannelResponse(): 이 메소드는 채널 정보를 채널 응답으로 변환하는 역할을 수행한다.
- ChannelResponse: 이 클래스는 채널 정보를 담고 있는 응답 클래스이다.
- ChannelResponseBuilder: 이 클래스는 채널 응답을 구성하는 빌더 클래스이다.
- ChannelResponses: 이 클래스는 채널 응답들을 담고 있다.

이와 같이 각 클래스와 메소드는 모든 채널 정보 조회 과정에서 필요한 역할을 담당하며, 채널과 회원 정보를 조회하여 채널 응답들을 생성하여 반환한다.

4.6. Schedule creation

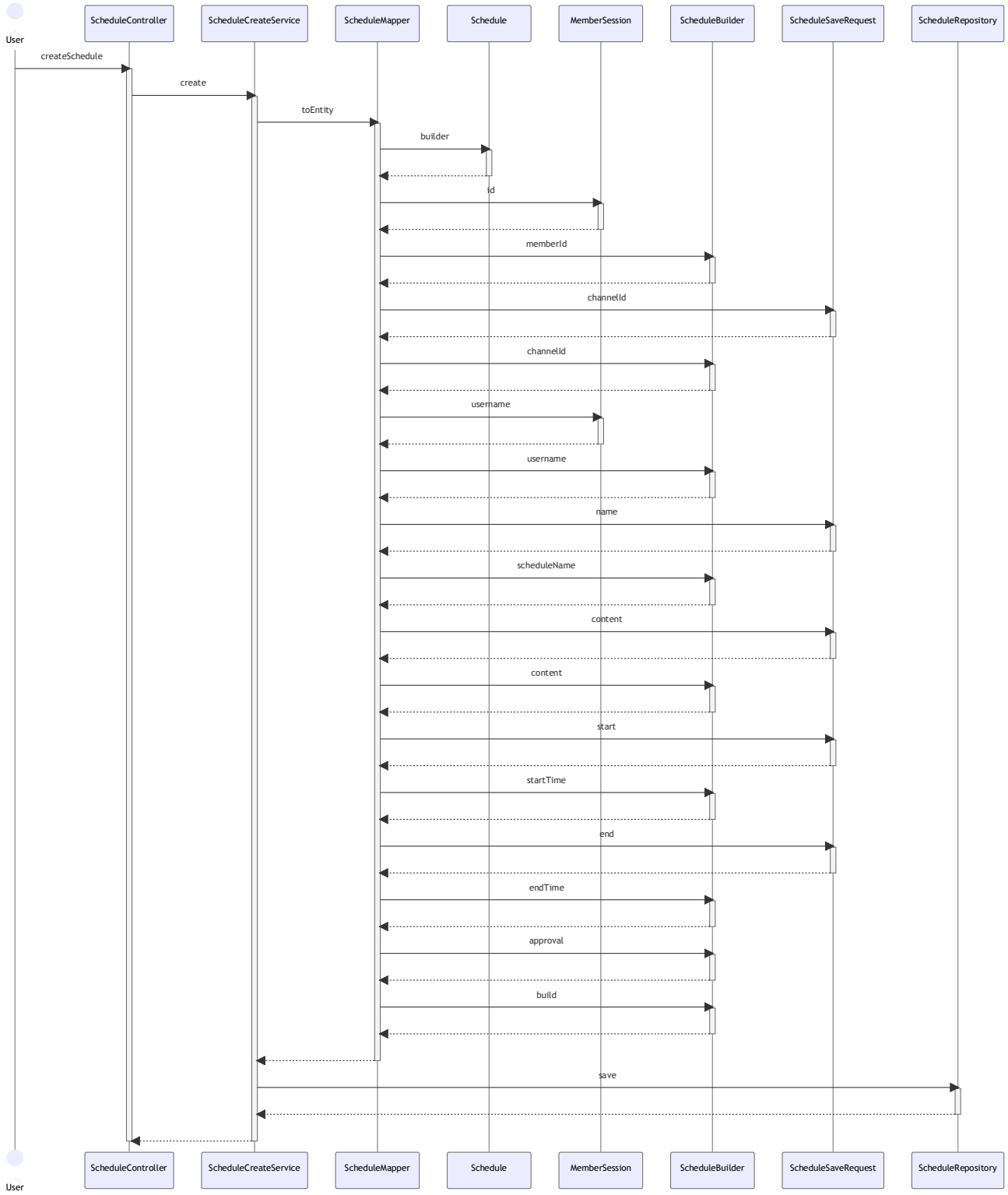

Figure 4-6 Schedule creation sequence diagram

이 시퀀스 다이어그램은 사용자가 스케줄을 생성하는 일련의 과정을 나타낸다..

- ScheduleController: 이 클래스는 일정 생성 요청을 받고, ScheduleCreateService에 처리를 위임하는 역할을 한다.
- ScheduleCreateService: 이 클래스에서는 create() 메소드를 통해 일정 생성 요청을 처리하고, ScheduleMapper를 사용하여 일정 엔티티를 생성한다.
  - create(): 이 메소드는 ScheduleCreateService 클래스에서 일정 생성 요청을 처리하는 역할을 한다.

- ScheduleMapper: 이 클래스에서는 toEntity() 메소드를 통해 일정 정보를 일정 엔티티로 변환하는 역할을 수행한다.
  - toEntity(): 이 메소드는 ScheduleMapper 클래스 내에서 일정 정보를 일정 엔티티로 변환하는 역할을 수행한다.
- Schedule: 이 클래스는 일정 정보를 담고 있는 엔티티 클래스이다.
- MemberSession: 이 클래스는 회원 세션 정보를 담고 있다.
- ScheduleBuilder: 이 클래스는 일정 엔티티를 생성하는 빌더 클래스이다.
- ScheduleSaveRequest: 이 클래스는 일정 생성 요청 시 필요한 정보를 포함하고 있다.
- ScheduleRepository: 이 클래스는 save() 메소드를 통해 일정 정보를 저장하고 조회하는 역할을 수행한다.
  - save(): 이 메소드는 일정 정보를 저장하는 기능을 수행한다.

이와 같이 각 클래스와 메소드는 일정 생성 과정에서 필요한 역할을 담당하여 사용자의 요청을 처리하고 일정 엔티티를 생성하여 저장한다.

4.7. Single schedule find service

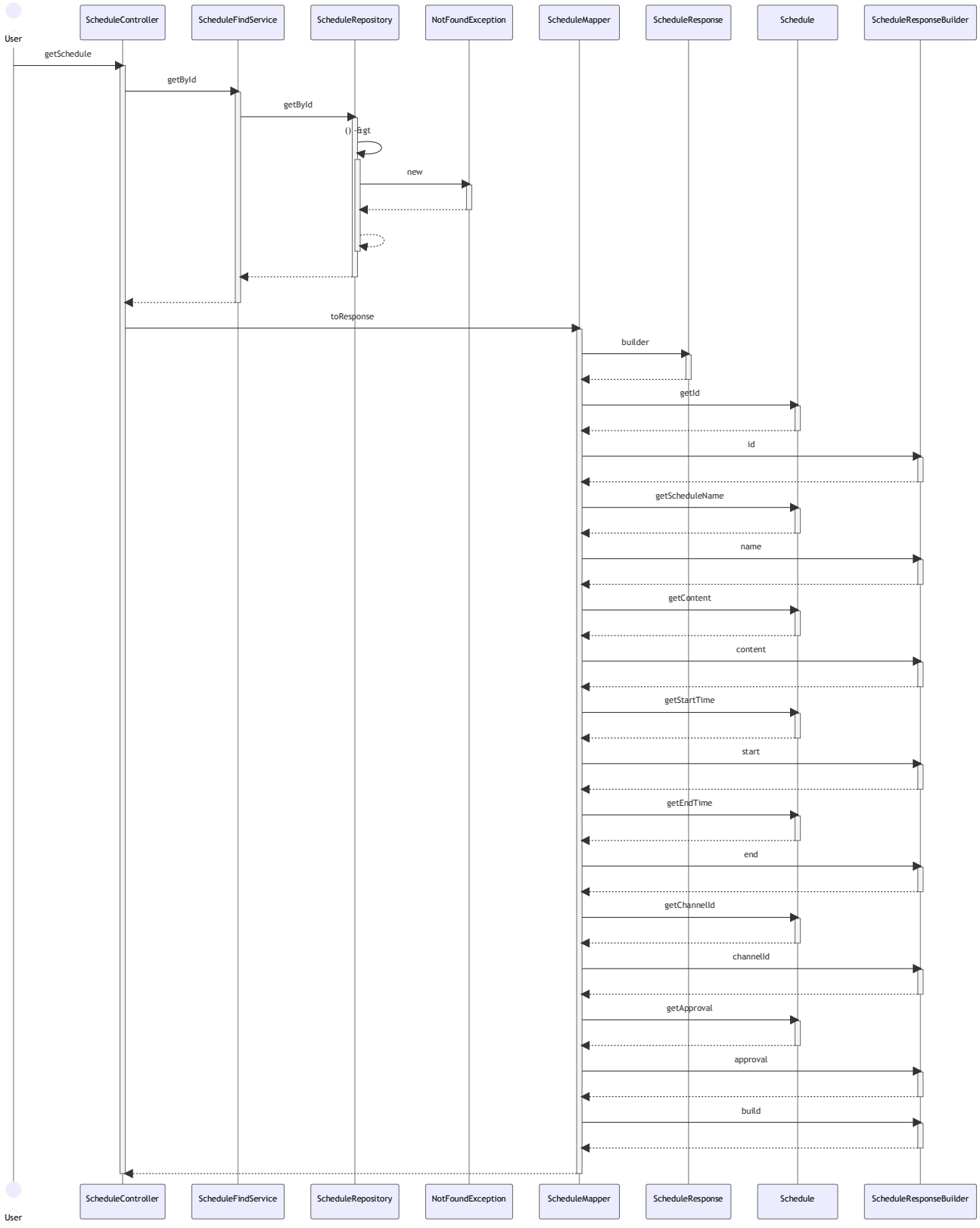

Figure 4-7 Single schedule find service sequence diagram

이 시퀀스 다이어그램은 사용자가 특정 스케줄을 조회 시의 일련의 과정을 나타낸다.

- ScheduleController: 이 클래스에서는 일정 정보 조회 요청을 받고, ScheduleFindService에 처리를 위임하는 역할을 한다.
- ScheduleFindService: 이 클래스에서는 getById() 메소드를 통해 특정 일정의 정보를 조회하고

ScheduleRepository를 사용하여 일정 정보를 조회한다.

- getById(): 이 메소드는 특정 일정 ID에 해당하는 일정 정보를 조회하는 기능을 수행한다.

- ScheduleRepository: 이 클래스에서는 getById() 메소드를 통해 일정 정보를 저장하고 조회하는 역할을 수행한다.
  - getById(): 이 메소드는 일정 ID에 해당하는 일정 정보를 조회하는 기능을 수행한다.
- NotFoundException: 이 클래스는 예외 클래스로, 조회된 일정 정보가 없을 경우 에러를 발생시킨다.
- ScheduleMapper: 이 클래스에서는 toResponse() 메소드를 통해 일정 정보를 일정 응답(ScheduleResponse)로 변환하는 역할을 수행한다.
  - toResponse(): 이 메소드는 일정 정보를 일정 응답으로 변환하는 역할을 수행한다.
- Schedule: 이 클래스는 일정 정보를 담고 있는 엔티티 클래스이다.
- ScheduleResponse: 이 클래스는 일정 정보를 담고 있는 응답 클래스이다.
- ScheduleResponseBuilder: 이 클래스는 일정 응답을 구성하는 빌더 클래스이다.

이와 같이 각 클래스와 메소드는 특정 일정 정보 조회 과정에서 필요한 역할을 담당하여 사용자의 요청을 처리하고 일정 응답을 생성하여 반환한다.

4.8. Total schedule find service

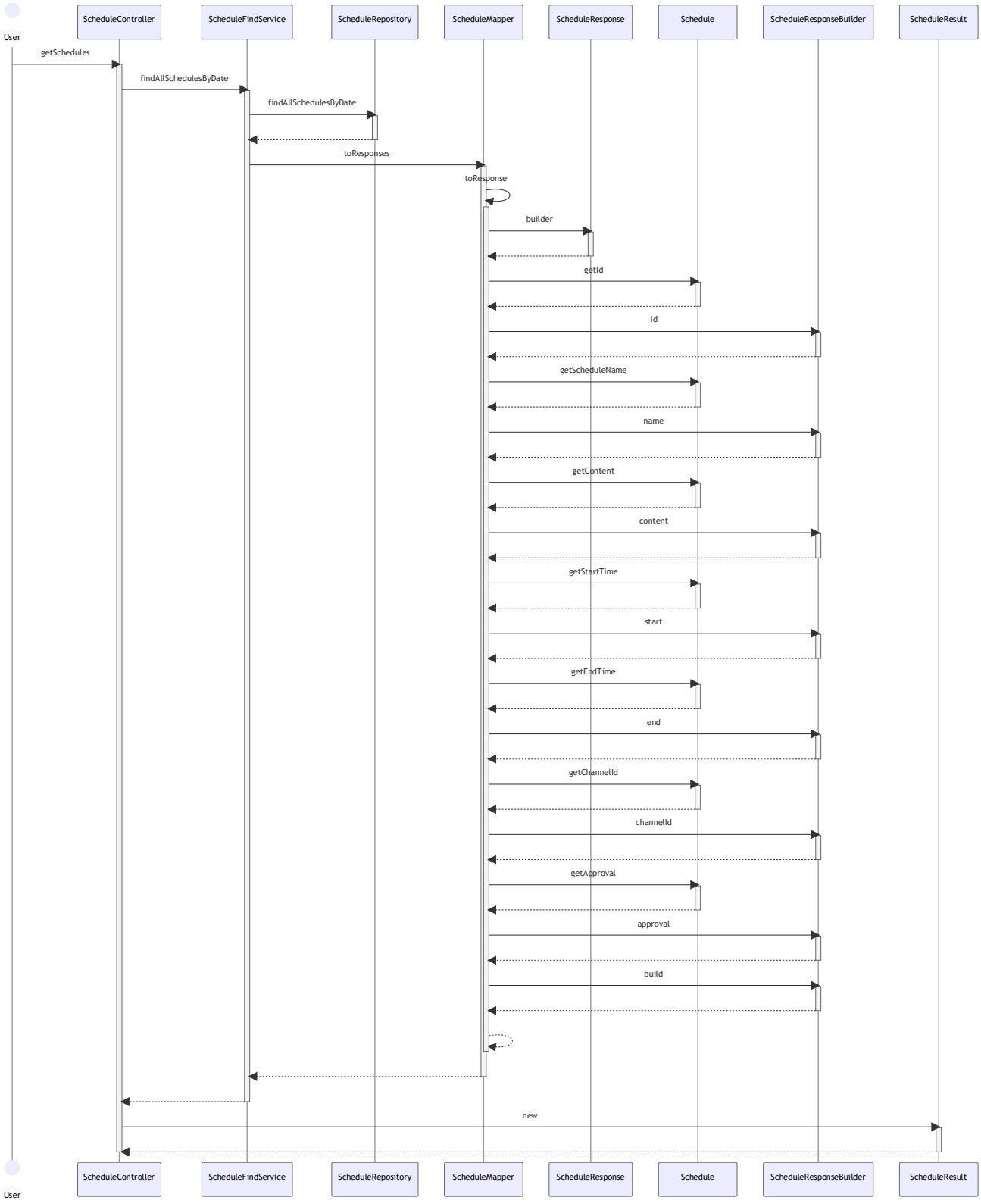

Figure 4-8 Total schedule find sevice sequence diagram

이 시퀀스 다이어그램은 사용자가 특정 날짜의 모든 일정 정보를 조회 시의 일련의 과정을 나타낸다.

- ScheduleController: 이 클래스에서는 일정 정보 조회 요청을 받고, ScheduleFindService에 처리를 위임한다.
- ScheduleFindService: 이 클래스에서는 findAllSchedulesByDate() 메소드를 통해 특정 날짜에 해당하는 모든 일정 정보를 조회하고, ScheduleRepository를 사용하여 일정 정보를 조회한다.

- findAllSchedulesByDate(): 이 메소드는 특정 날짜에 해당하는 모든 일정 정보를 조회하는 기능을 수행한다.

- ScheduleRepository: 이 클래스에서는 findAllSchedulesByDate() 메소드를 통해 일정 정보를 저장하고 조회하는 역할을 수행한다.
  - findAllSchedulesByDate(): 이 메소드는 특정 날짜에 해당하는 모든 일정 정보를 조회하는 기능을 수행한다.
- ScheduleMapper: 이 클래스는 일정 정보를 일정 응답(ScheduleResponse)으로 변환하는 역할을 수행한다.
  - toResponse(): 이 메소드는 일정 정보를 일정 응답으로 변환하는 기능을 수행한다.
- Schedule: 이 클래스는 일정 정보를 담고 있는 엔티티 클래스이다.
- ScheduleResponse: 이 클래스는 일정 정보를 담고 있는 응답 클래스이다.
- ScheduleResponseBuilder: 이 클래스는 일정 응답을 구성하는 빌더 클래스이다.
- ScheduleResult: 이 클래스에서는 조회된 일정 정보들을 담고 있다.

이와 같이 각 클래스와 메소드는 특정 날짜의 모든 일정 정보 조회 과정에서 필요한 역할을 담당하며, 사용자의 요청을 처리하고 일정 응답들을 생성하여 반환한다.

4.9. Schedule creation request find service

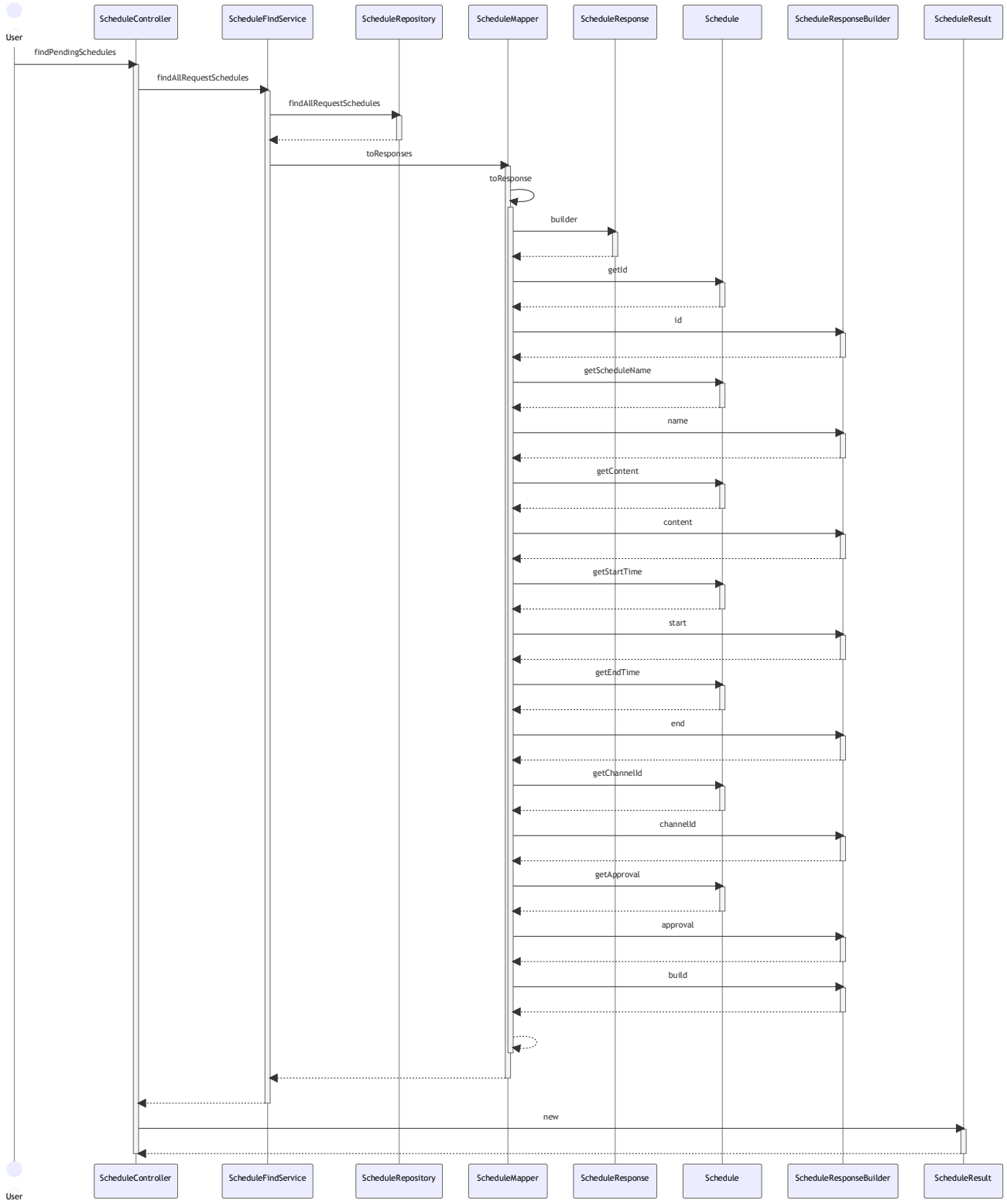

Figure 4-9 Schedule creation request find service sequence diagram

이 시퀀스 다이어그램은 관리자가 스케줄 생성 요청 목록을 조회하는 일련의 과정을 나타낸다.

- ScheduleController: 이 클래스에서는 일정 정보 조회 요청을 받고, ScheduleFindService에 처리를 위임한다.
- ScheduleFindService: 이 클래스에서는 findAllRequestSchedules() 메소드를 통해 보류 중인 모든 일정 정보를 조회하고, ScheduleRepository를 사용하여 일정 정보를 조회한다.
  - findAllRequestSchedules(): 이 메소드는 보류 중인 모든 일정 정보를 조회하는 기능을 수행한다.

- ScheduleRepository: 이 클래스는 findAllRequestSchedules() 메소드를 통해 일정 정보를 저장하고 조회하는 역할을 수행한다.
  - findAllRequestSchedules(): 이 메소드는 보류 중인 모든 일정 정보를 조회하는 기능을 수행한다.
- ScheduleMapper: 이 클래스는 toResponse() 메소드를 통해 일정 정보를 일정 응답(ScheduleResponse)로 변환하는 역할을 수행한다.
  - toResponse(): 이 메소드는 일정 정보를 일정 응답으로 변환하는 기능을 수행한다.
- Schedule: 이 클래스는 일정 정보를 담고 있는 엔티티 클래스이다.
- ScheduleResponse: 이 클래스는 일정 정보를 담고 있는 응답 클래스이다.
- ScheduleResponseBuilder: 이 클래스는 일정 응답을 구성하는 빌더 클래스이다.
- ScheduleResult: 이 클래스는 조회된 일정 정보들을 담고 있다.

이와 같이 각 클래스와 메소드는 보류 중인 모든 일정 정보 조회 과정에서 필요한 역할을 담당하며, 사용자의 요청을 처리하고 일정 응답들을 생성하여 반환한다.

4.10. Schedule approval

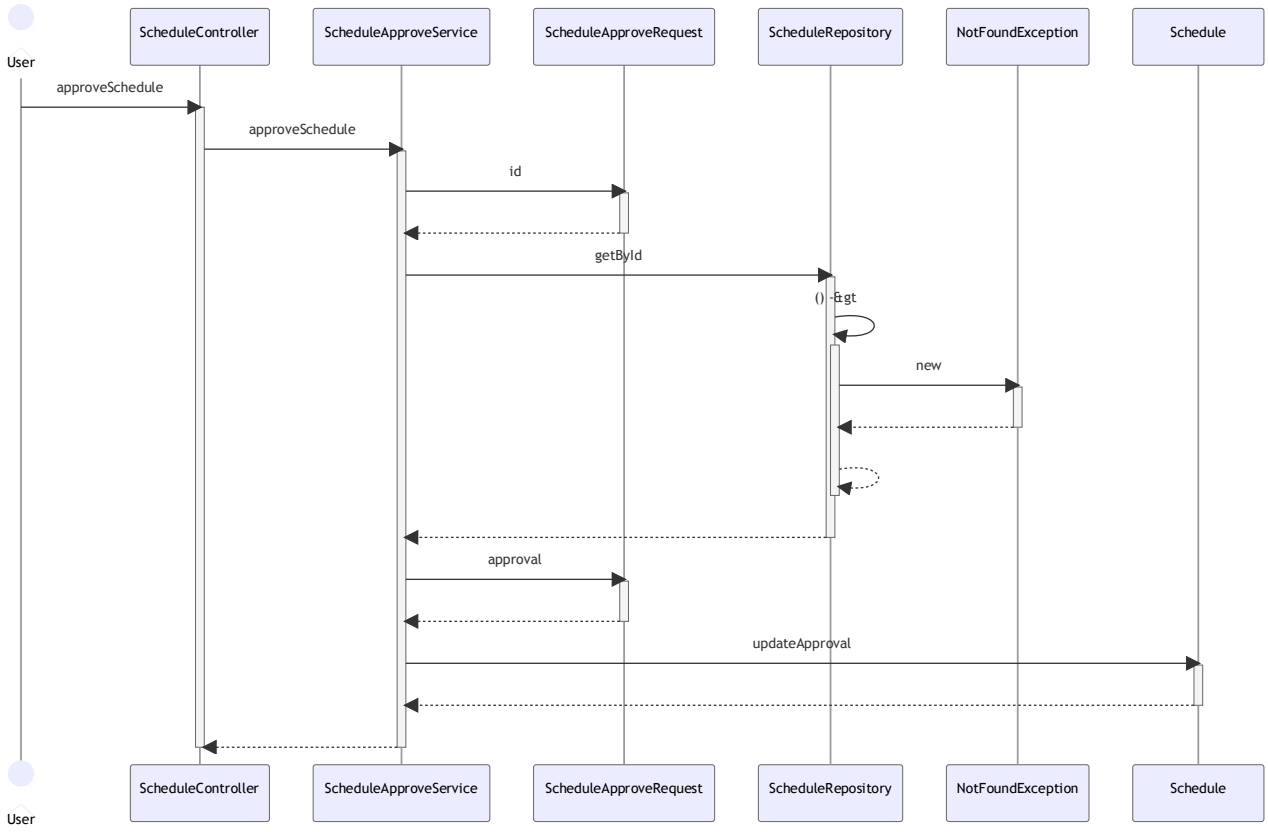

Figure 4-10 Schedule approval sequence diagram

이 시퀀스 다이어그램은 관리자가 특정 일정을 승인하는 일련의 과정을 나타낸다.

- ScheduleController: 이 클래스에서는 일정 승인 요청을 받고, ScheduleApproveService에 처리를 위임한다.
- ScheduleApproveService: 이 클래스는 aproveSchedule() 메소드를 통해 일정 승인을 처리한다.
  - approveSchedule(): 이 메소드는 특정 일정의 승인을 처리하는 기능을 수행한다.
- ScheduleApproveRequest: 이 클래스는 일정 승인 요청 정보를 담고 있다.
- ScheduleRepository: 이 클래스는 getById() 메소드를 통해 일정 정보를 저장하고 조회하는 역할을 수행한다.

- getById(): 이 메소드는 특정 일정 ID에 해당하는 일정 정보를 조회하는 기능을 수행한다.

NotFoundException: 이 클래스는 예외 클래스로, 조회된 일정 정보가 없을 경우 에러를 발생시킨다.

Schedule: 이 클래스는 일정 정보를 담고 있는 엔티티 클래스이다.

- updateApproval(): 이 메소드는 일정의 승인 상태를 업데이트하는 기능을 수행한다.

이와 같이 각 클래스와 메소드는 일정 승인 요청을 처리하고, 필요한 정보를 조회하며, 일정의 승인 상태를 업데이트한다. 또한 이를 통해 사용자의 승인 요청에 대한 처리를 수행한다.

5. Appendix

5.1. User Interfaces

5.1.1. SignIn / SignUp Menu

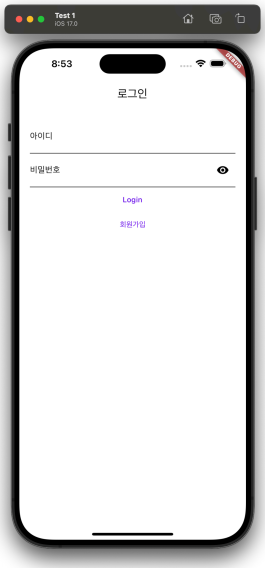

Figure 5-1

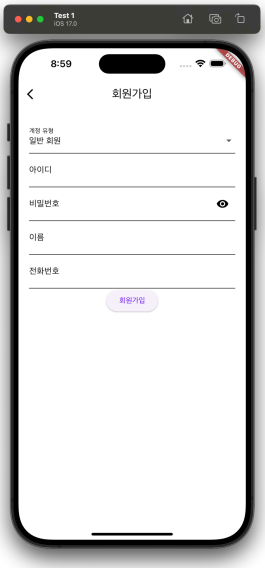

Figure 5-2

5.1.2. Admin Menu

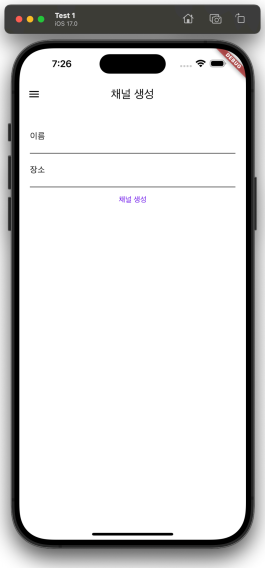

Figure 5-3

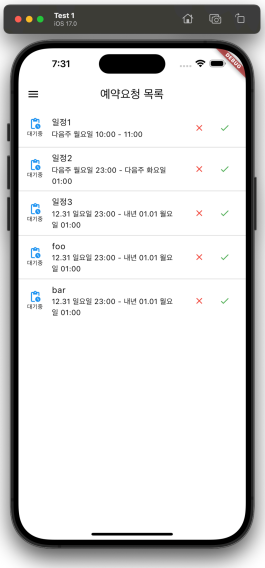

Figure 5-4

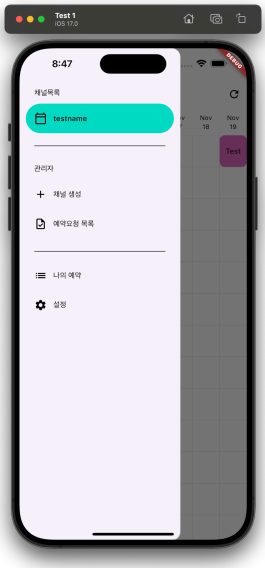

Figure 5-5

5.2. Guest Menu

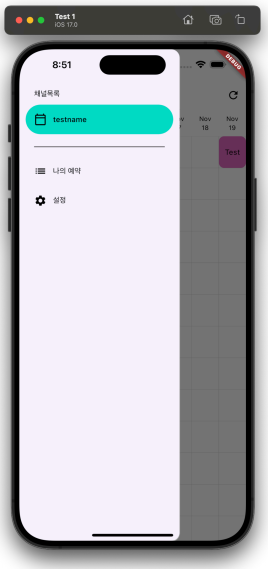

Figure 5-6

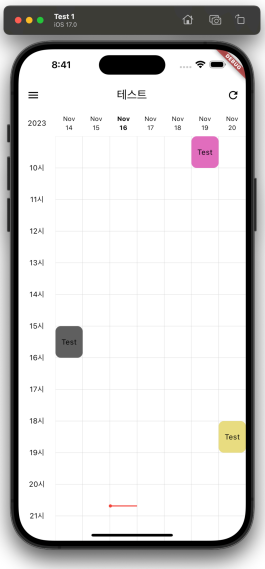

Figure 5-7

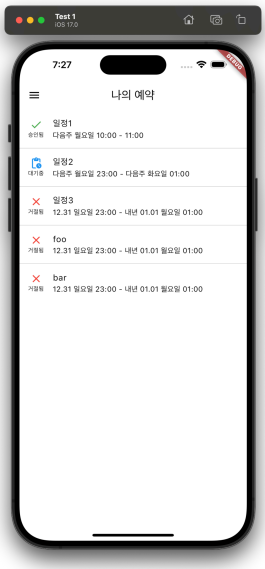

Figure 5-8

Figure 5-9

- Project Status

박종범: 백엔드 서버 설계 및 관리, 프론트엔드 애플리케이션 개발

정유환 : DB 설계, API 설계 및 문서화, 백엔드 서버 배포, 백엔드 애플리케이션 개발/테스트 코드 작성

김동령: Prototyping, 프론트엔드 View 제작 및 문서화

이승주: Prototyping, 백엔드 애플리케이션 개발

- Prototype

Figure 5-10

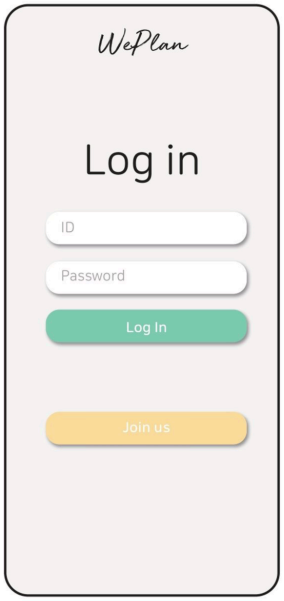

Figure 5-11

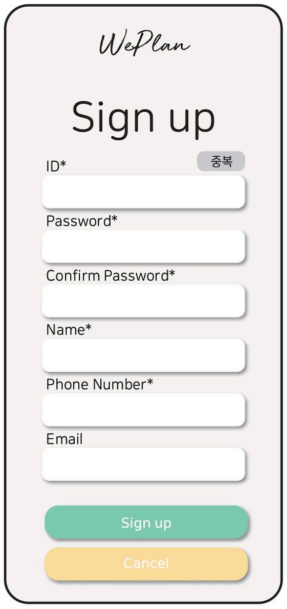

Figure 5-12

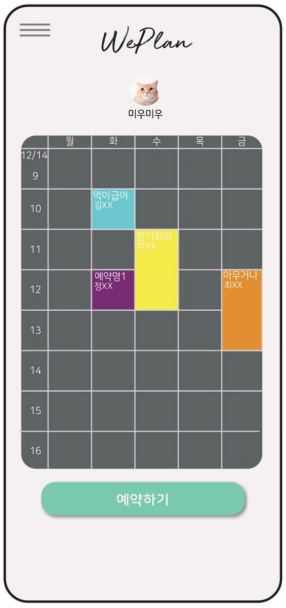

Figure 5-13

Figure 5-14

Gantt Chart

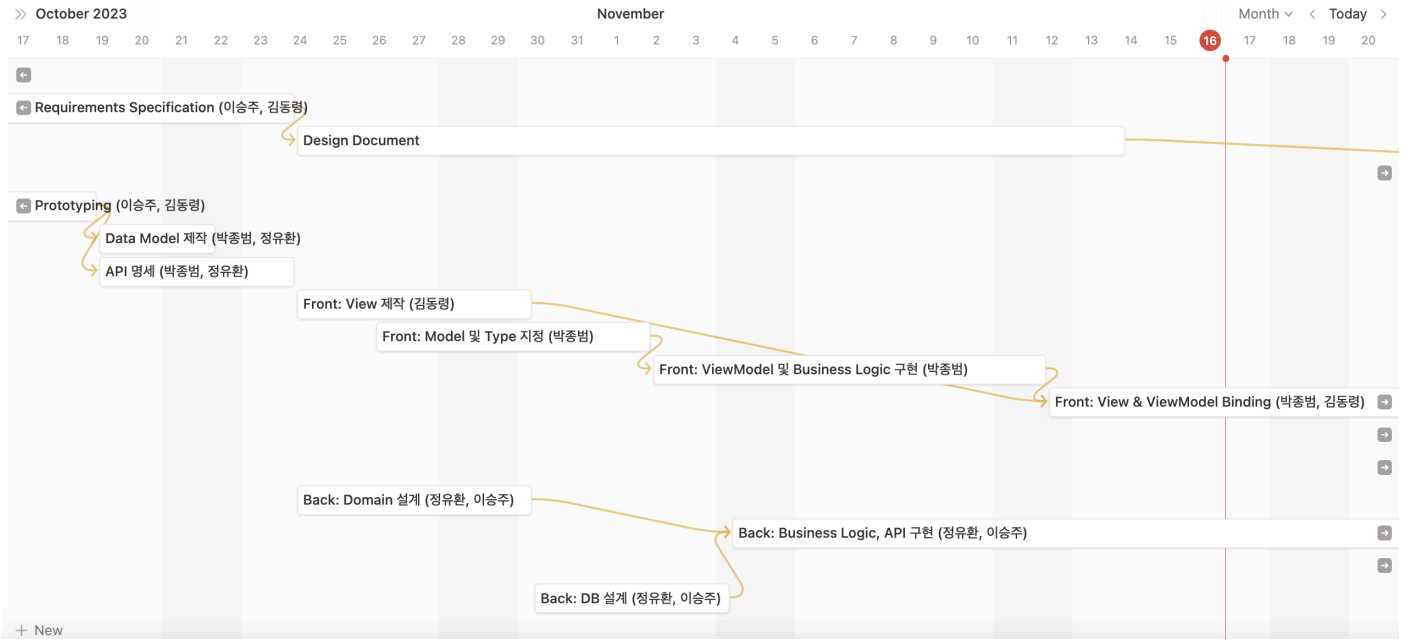

Figure 5-15

Swagger API Document

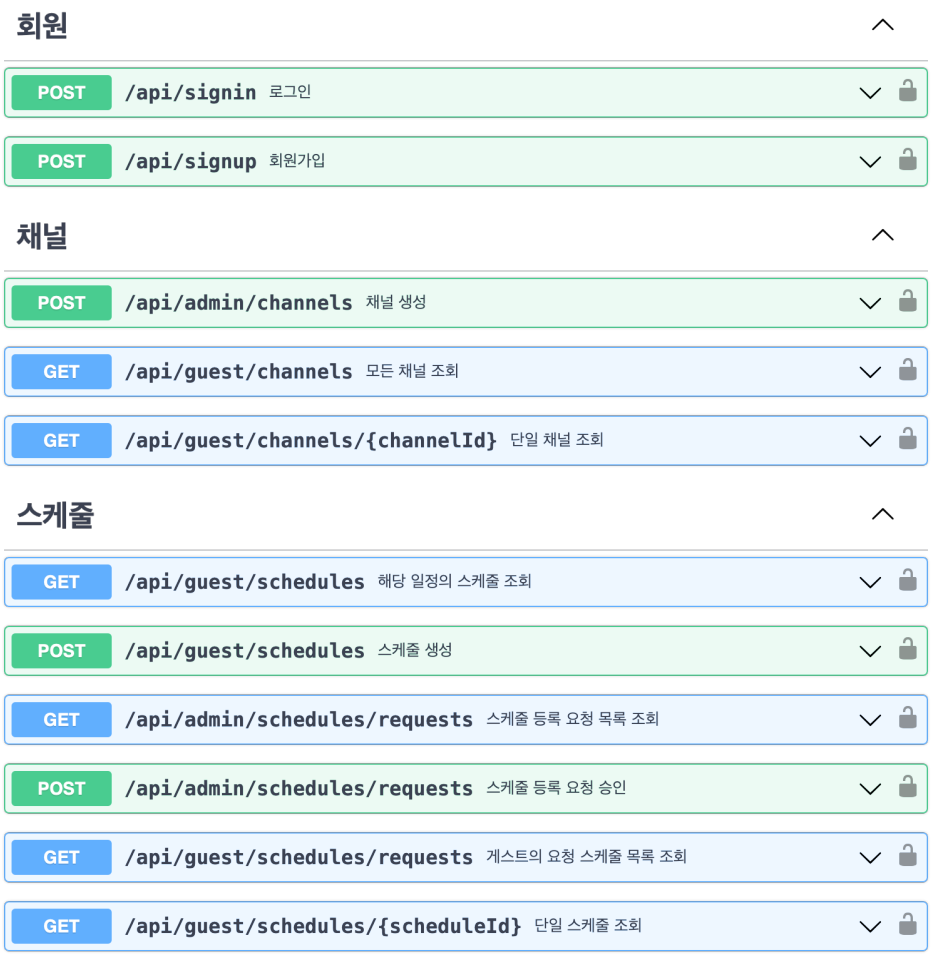

Figure 5-16

- FrontEnd Repository Commit Log

URL: https://github.com/softwar7/weplan

```
*   f3acfff (HEAD -> main, origin/main) Merge remote-tracking branch 'origin/feature/signup'
|\
| * 26ff317 (origin/feature/signup) RoleType field added in signup
* | 0df502c feat: add admin reservation requests view fixme: Appoval enum to string
* | 5aaed68 fix: handle signout only when refreshtoken failure 401
* | 4536f24 feat: add my reservations view
* | 9b5bee1 feat: add function to get relative date
* | 688e205 refactor: rename ChannelsViewModel to ChannelService
* | 3fda03e fix: GET /schedules channelId required field
* | 5270d4e refactor: viewmodel shows viewmodel
* | 54bfa30 fix: signout when refresh token is not checked properly
* | f33c27b feat: modifiy baseuri to deployed server
* | 5803c8b feat: navigate to signuppage
* | e74a4a4 feat: add signup api
* | 81653f4 fix: notifylistener not specified
* | 405f83e Merge remote-tracking branch 'origin/feature/signup'
|\|
| * 3f7dc54 feat: add signup page
* | 971f408 (refactor/mvvms) refactor: complete viewmodel
* | 4086789 refactor: mvvms
* | c086cec refactor: change file name & Menu class schema for the main scaffold
* | eb1d940 refactor: add settings page & move logout to settings page
* | 3d5b294 fix: error handling when server not connected & save nullable token
* | 16ac96b chore: change logger configs
* | 46d3e4b FEAT!: add ChannelsViewModel & create channel form
* | 4a6c6a2 fix: signout on refreshtoken expiration
* | 93a7144 fix: start & end time as ISO8601 string
* | 64bac64 move: validation logic
* | 081b879 refactor: merge tokenInterceptor & updateTokenInterceptor
* | 9d434de refactor: encapsulate authstorage
* | 8d25f55 refactor: navigationdrawer selectIndex logic
* | 0b36663 fix: bug getting token from header
* | 52a2289 fix: store isadmin info
|/
* 39eabaa chore: github action command
* 51ba9dd fix: bug while formatting
* 0c023a8 chore: handle error on login failure
* a846eea feat: show channels & handle logout
* 2ac11f7 feat: add signOut
* 5883025 feat: add timetable api calls
* e5df3a4 feat: add timetable default component
* e1ac96e build: add customized flutter_timetable (PR in progress)
* 9907c1a feat: add drawer components
* 0385f29 feat: add snackbar component
* 27e0d95 refactor: api id as int type
* 67e4fbf remove: unused type
* c945b54 feat: login onpress, apiservice fix: notifyListener not invoked after accesstoken is updated
* a97933e minor change: simplify password regexr
* 56d41b2 build: ios
* 30c002d fix: minor bug - isAuth App changed
* ece02d5 ios_build: securestorage
* a365554 chore: remove pc platform support
* 663d4d7 chore: allow explicit `this`
* d944ff6 feat: manage auth tokens automatically
* 7f5a9e3 feat: error handling in auth_service
* 7e0d467 refactor: api services args as fields
* ac666e9 dev_feat: add logger util
```

```
* 8960d58 feat: signup prototype & minor fix
* cc03335 feat: wrap materialapp with authservice provider
* 5b6660f feat: authService
* dc25703 remove: SignRequestType & add: dio error interceptor
* 0004347 feat: add login form & add validators for sign forms
```

- Backend Repository Commit Log

URL: https://github.com/softwar7/weplan-server

```
* f5d8c0c (HEAD -> main, origin/main, origin/HEAD) test : 쿼리 파라미터 문서화 추가
* 979296a test : 게스트의 요청 스케줄 목록 조회
* ea1b195 refactor : 스케줄 조회 동적 쿼리로 변경
* 9ece3a3 fix : url 오류 수정
* 535fc86 fix : 스케줄 DateTimeFormat 수정
* deeaa3d fix : 스케줄 생성시 채널 검증
* a2f82a9 refactor : ISO 8601 형식으로 통일
* 5be77b4 feat : 사용자의 예약 정보 가져오기
* d5926ff refactor : url에 guest 추가
* 67e1777 feat : 회원가입 중복 체크 추가
* f7c9ba5 fix : LocalDate ISO.DATE_TIME 오류 및 query null 해결
* 8ae6bab fix : AccessToken 주입 설정 변경
* 6a8193b schedule 변수명 수정
* 05f0404 스케줄 등록 요청 승인 테스트 작성 및 API 문서화
* 7efeb72 스케줄 등록 요청 승인 로직 구현
* 52849de 스케줄 등록 요청 리스트 가져오기 테스트 작성 및 API 문서화
* ba59133 스케줄 등록 요청 리스트 가져오기 구현
* d9dce50 ArgumentResolver로 관리자 권한 관리
* f23f495 일정에 맞는 스케줄 조회 테스트 작성 및 API 문서화
* 845f3ee 일정에 맞는 스케줄 조회
* 8a027e2 README 추가
* 74fa7e8 초기화 Dummy 데이터 생성
* f7e01f4 단일 스케줄 조회 테스트 작성 및 API 문서화
* 3f57a73 단일 스케줄 조회 로직 구현
* f8fe9dc 스케줄 생성 테스트 코드 작성 및 API 문서화
* c9ff1cf 채널 생성, 조회 테스트 작성 및 API 문서화
* 9160f84 로그인 테스트 작성 및 API 문서화
* b1a90ec 회원가입 API 문서화
* 0dd3b17 스케줄 생성 로직 구현
* c5207eb 모든 채널 조회 로직 구현
* dc3babd 단일 채널 조회 로직 구현
* cb28e0c 채널 생성 로직 구현
* 12dcb31 Access, Refresh Token 관리 방식 변경
* 9739779 회원 로그인 구현
* f53e27f 회원가입 입력 값 검증 추가 및 도메인 필드 수정
* 87f00e0 회원 가입 로직 구현
* f530e3c 테스트 초기 설정
* e04c1a1 도메인 초기 모델 구현
* 41f61ff JWT 토큰 발급 로직 (AccessToken, RefreshToken)
* feead8b 공통 예외 처리 및 Jwt - ArgumentResolver 처리
* a84f5cd global config 세팅
* a75c29a build.gradle 및 yml 설정 추가
* d8b5fb9 초기 설정
```

6. Reference

O'Reilly Media, Inc. (날짜 정보 없음). "Chapter 1. Layered Architecture." O'REILLY Software Architecture Patterns: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html에서 검색됨

"지속 성장 가능한 소프트웨어를 만들어가는 방법". (날짜 정보 없음).
https://geminikims.medium.com/%EC%A7%80%EC%86%8D-%EC%84%B1%EC%9E%A5-%EA%B0%80%EB%8A%A5%ED%95%9C-%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4%EB%A5%BC-%EB%A7%8C%EB%93%A4%EC%96%B4%EA%B0%80%EB%8A%94-%EB%B0%A9%EB%B2%95-97844c5dab63 에서 검색됨