

*Real Man Proj*





구성원



개발 배경



구현 목표



요구사항 정의  
서



ERD



스토리보드



UseCase



클래스 / 시퀀  
스 다이어그램



코드리뷰

# Team Five Guys



이 주형  
Jonathan



박래오  
Desert



김대철  
Andy Polo



윤정호  
Philip



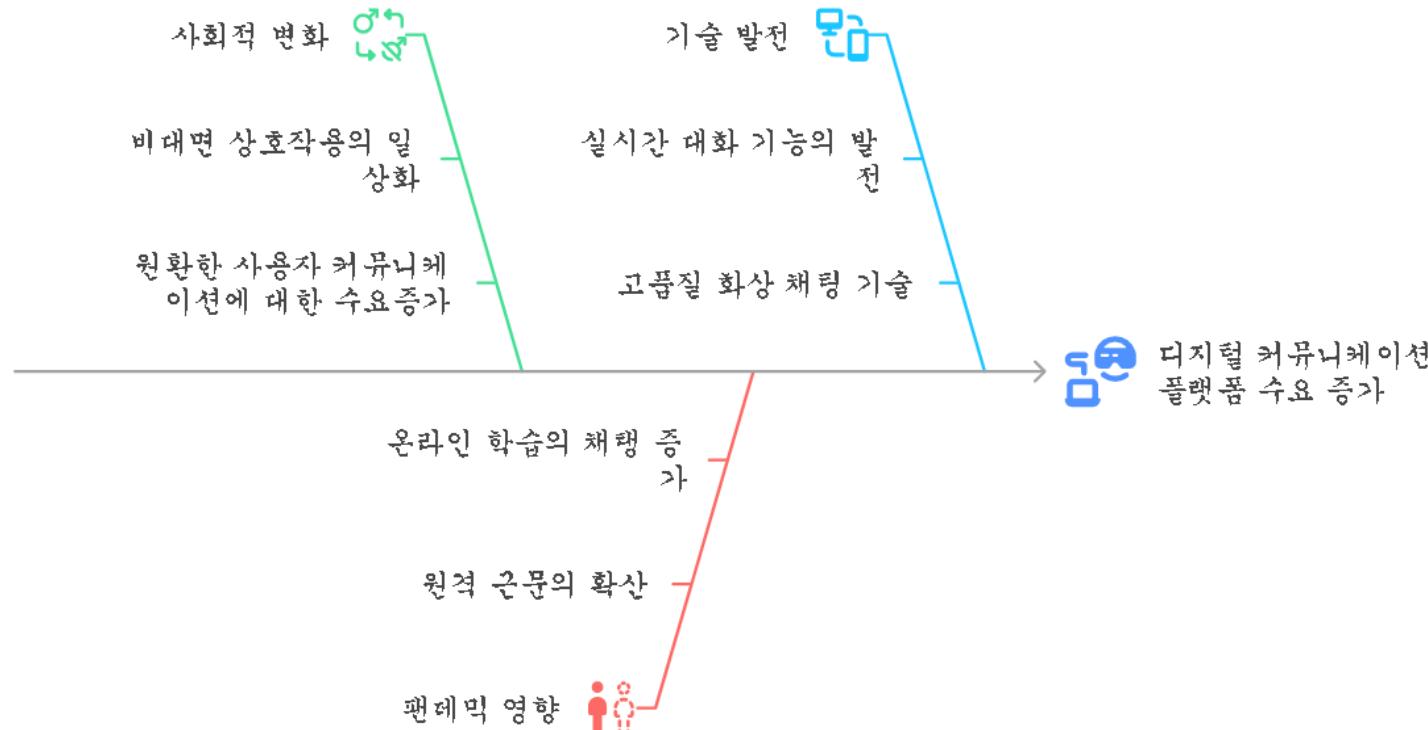
문성종  
Maddie

## 개 발 배 경

실시간 커뮤니케이션 플랫폼을 개발하게 된 배경

1. **사회적 변화**로 비대면 협업이 증가되고, 사용자 커뮤니티 대한 수요도 증가했습니다.
2. **기술 발전**을 통해 실시간 대화와 고품질 화상 기술이 가능해졌고, 사용자 기대 수준도 높아졌습니다.
3. **팬데믹** 이후 **원격 근무와 온라인 학습이 급증**하며, 안정적이고 실시간성이 높은 플랫폼의 필요성이 커졌습니다.

이러한 변화 덕에 사람들은 더나은 실시간 커뮤니케이션 플랫폼을 찾게되었고 그흐름에 맞춰 이 플랫폼을 개발하게 되었습니다.



# 팬데믹 이후 디지털 산업의 변화

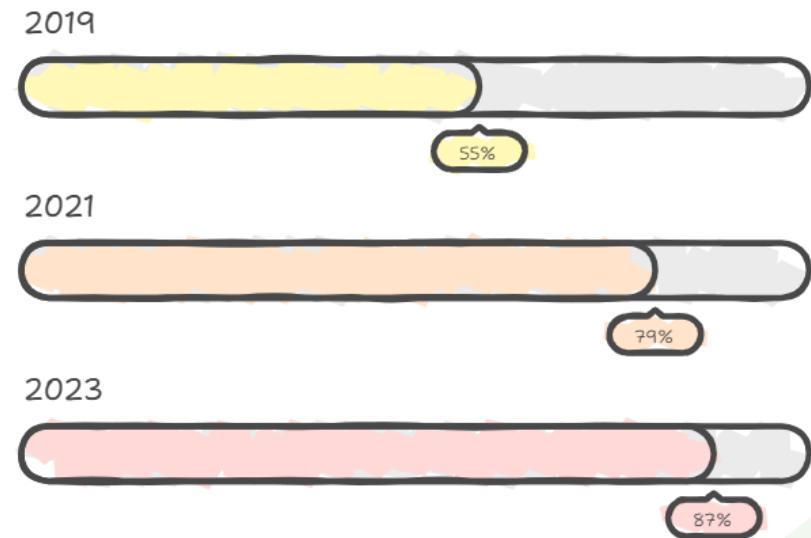
쇼핑 & 유통	오프라인 매장 감소, 온라인 쇼핑·배달 서비스 활성화 (예: 쿠팡, 마켓컬리)
교육	온라인 강의, MOOC(대규모 온라인 공개강좌), 메타버스 기반 교육 증가
금융	비대면 계좌 개설, AI 챗봇 활용한 금융 상담 (예: 카카오뱅크, 토스)
헬스케어	원격 진료, AI 진단, 스마트 헬스케어 기기 도입 증가
근무 환경	재택근무, 화상회의 플랫폼 사용 증가 (예: Zoom, Slack, MS Teams)
여가 & 문화	OTT 서비스(넷플릭스, 유튜브) 증가, 온라인 콘서트 & 전시회 개최



## 기사스크랩 및 점진적 증가 추이 그래프

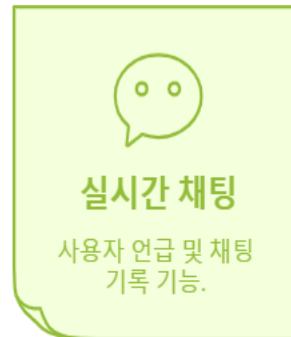
- 협업툴이 원격 근무를 위한 필수 도구처럼 인식되며 전염병 확산 이후 국내외에서 사용이 크게 증가했다.
- 글로벌 시장조사업체 가트너가 지난해 8월 발표한 조사결과에 따르면 협업툴을 사용하는 직장인은 2019년 55%에서 2021년 79%로 증가했다. 최근 거리 두기 해제로 사무실 출근이 늘어날 것으로 보이지만 협업툴 이용은 변화와 상관없이 계속 증가할 것으로 보인다.
- 지난 4월 7일 인크루트가 직장인 939명을 대상으로 '협업툴 활용 현황과 엔데믹 이후 수요 예상'을 주제로 설문조사를 한 결과, 응답자의 87.0%가 코로나19가 종식돼도 수요가 증가할 것으로 예상했다
  - 참고 자료 - 경향 신문 주영재 기자

팬데믹 이후 협업 툴 이용률 증가 추이





구현 목표



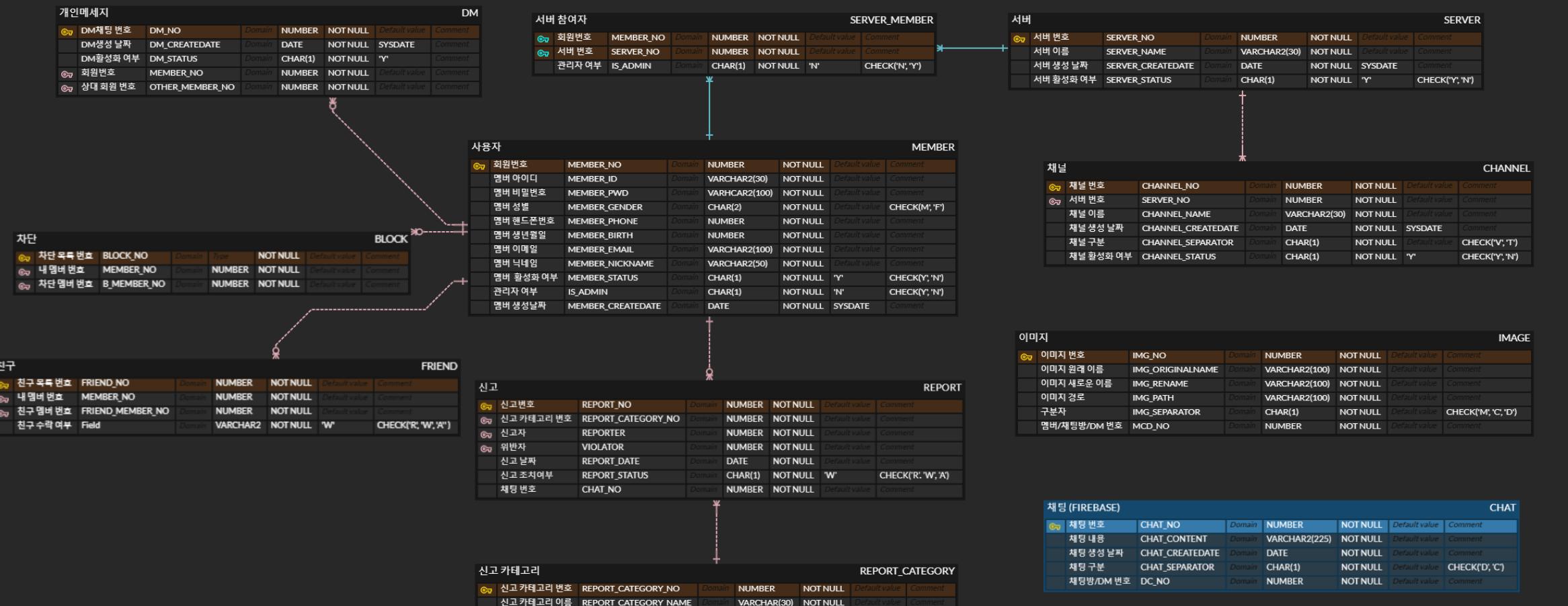
No	구분	요구사항 ID	요구사항 명	요구사항 상세 설명	비고
1	대화 화면	chat-001	채팅	채팅 - 채팅입력 api를 따와서 채팅창에서 보낼수있도록 구현 (굵기,기울임,첨부파일등등)	
				채팅시간 - 채팅이 입력되면 뒤에 자동으로 채팅올린 시간입력	
				언급기능 - @을 이용하여 사용자를 언급할수 있는 기능	
				채팅 전송 예약 - 채팅전송을 예약하는 기능 구현	
				#기능 - #채널명으로 삽입을 하면 클릭하면 해당 채널로 이동할수있는 기능	
				to do list - 체크리스트를 넣어서 체크를하면 선을 그어줌	
		chat-002	일정관리	날짜 선택기능 - 날짜를 선택할수있게하여 채팅방에 보이는 기능	
				날짜/시간 리마인더 - 특정날짜/시간에 알림을보내 사용자들에게 알림	
		chat-003	화상채팅	화상화면 - 캠코더 화면으로 소통할수있게함	
			책갈피	링크를 추가해서 즐겨찾는 링크를 설정	
2	사이드바	side-001	채널	채널 목록 / 검색 - 본인이 참여하고있는 채널 목록을 표시 / 서버내 채널을 검색 기능	3/26/2025
				채널 생성 및 제거 , 채널 수정, 선호하는 채널이 상단에 올라갈수있게 변경 기능	
		side-002	음성 채널	사용자를 음성채널에 참여 시킴 / 사용자 마이크 사용시 ui변경 및 마이크 비활성화, 듣기 비활성화 인원 ui 추가	
		side-003	타임 라인	서버 내 타임 라인 채널로 이동 / 사용자가 비로그인시 서버내 활동내역을 볼수있게 해줌	
		side-004	신고	서버내 불법적인 무언가를 제제 및 신고할수있는 창을 활성화	
		side-005	유저	서버내 모든 유저를 표기 / 친구추가된 유저를 최상단에 위치할수있게 만들고 , 그ㄴㄷ 순으로 정렬, 및 유저 이름, 직책 검색 기능	
3	탑바	side-006	화면 공유	서버내 사용자가 화면공유 버튼을 누르면 서버내 참여한 인원이 사용자의 화면을 볼수있게 활성화 , 및 비활성화	3/27/2025
		top-001	메세지 검색	검색시 키워드에 따라 해당 워크스페이스에 있는 모든 메시지, 파일을 출력	

3 내정보	member-001	내 정보 조회	내 계정 프로필 아이콘 클릭시 내 정보 조회 페이지로 이동.	
	member-002	다른 사람 정보 조회	다른 사람 프로필 클릭시 상대방 정보 조회 페이지로 이동.	
			다른 사람 정보 페이지에서 dm 버튼을 누르면 dm창으로 이동.	
			다른 사람 정보 페이지에서 통화 버튼을 누르면 음성/영상 통화 창으로 이동.	
	member-003	내 정보 조회 페이지	이름, 직책, 프로필 사진, 이메일, 핸드폰 번호 조회.	
	member-004	내 정보 수정	내 이메일 수정.	주요 정보는 관리자가 수정
			내 핸드폰 번호 수정.	
			내 이름 수정.	
	member-005	정보 수정 (관리자 권한)	직책 수정.	
	member-006	회원가입	아이디를 입력.	
			비밀번호를 입력.	
			이메일을 입력.	
			핸드폰 번호를 입력.	
	member-007	로그인	아이디를 입력.	
			비밀번호 입력.	
			아이디 찾기 버튼 클릭시 아이디 찾기 페이지로 이동.	
			비밀번호 찾기 버튼 클릭시 비밀번호 찾기 페이지로 이동.	
	member-008	아이디 찾기 (택1)	이메일로 아이디 조회.	
			핸드폰 번호로 아이디 조회.	
	member-009	비밀번호 찾기	아이디를 입력.	
			이메일로 비밀번호 찾기.	
			핸드폰 번호로 비밀번호 찾기	

친구	friend-001	친구 버튼 클릭 시	<ul style="list-style-type: none"> <li>- 친구 버튼 클릭 시 기존 채널사이드바에 DM목록 출력</li> <li>- 대화 화면 페이지 첫번째 상단에 온라인, 모두, 대기중, 차단목록, 친구추가 버튼 출력           <ul style="list-style-type: none"> <li>- 대화 화면 페이지 두번째 상단에 검색기능 출력</li> <li>- 대화 화면 페이지에 친구 목록 출력</li> </ul> </li> </ul>
	friend-002	특정 DM 클릭 시	DM목록에서 특정 DM 클릭 시 1:1 DM화면 출력
	friend-003	DM 목록	<p style="text-align: center;">프로필사진, 닉네임으로 표시 (프로필 사진 오른쪽 아래에 온라인이면 초록색, 오프라인이면 회색으로 표시)</p>
	friend-004	친구 목록	<ul style="list-style-type: none"> <li>- 프로필 사진, 닉네임으로 표시 (프로필 사진 오른쪽 아래에 온라인이면 초록색, 오프라인이면 회색으로 표시)</li> <li>- 왼쪽에 메시지보내기, 토클버튼 표시</li> </ul> <p>(토클 버튼 클릭 시 '영상통화 시작하기', '음성 통화 시작하기', '친구 삭제하기', '차단하기' 출력)</p>
	friend-005	친구 검색 기능	<ul style="list-style-type: none"> <li>- 검색 시 검색하는 키워드가 들어간 친구리스트 출력(닉네임 and 아이디)</li> </ul>
	friend-006	온라인 버튼 클릭 시	<ul style="list-style-type: none"> <li>- 온라인 버튼 클릭 시 온라인 중인 친구 목록을 출력 (친구 목록 화면과 동일하게 출력)</li> </ul>
	friend-007	모두 버튼 클릭 시	<ul style="list-style-type: none"> <li>- 모든 친구 목록 출력 (친구 목록 화면과 동일하게 출력)</li> </ul>
	friend-008	대기 중 버튼 클릭 시	<ul style="list-style-type: none"> <li>- 내가 추가한 친구중 대기중인 친구 목록 출력 (프로필 사진, 닉네임, 아이디, 삭제버튼으로 구성)</li> <li>- 상대방이 나를 추가한 목록 출력 (프로필 사진, 닉네임, 아이디, 수락, 거절버튼으로 구성)</li> </ul>
	friend-009	차단 목록 버튼 클릭 시	<ul style="list-style-type: none"> <li>- 내가 차단한 계정 목록 출력 (프로필사진, 닉네임, 차단 해제 버튼으로 구성)</li> </ul>
	friend-010	친구 추가 버튼 클릭 시	<ul style="list-style-type: none"> <li>- 대화 화면 페이지에 아이디 입력 및 친구요청 보내기 버튼 출력</li> </ul>
	friend-011	친구요청 보내기 버튼 클릭 시	<ul style="list-style-type: none"> <li>- 성공 시 아래 초록 글씨로 '000에게 성공적으로 친구 요청을 보냈어요.' 출력</li> <li>- 실패 시 아래 빨간 글씨로 '잘못 적으신것 같네요 다시적어주세요.' 출력</li> </ul>

	Preferences-001	폰트 크기	- 폰트 크기 설정 (작게 / 보통 / 크게 / 아주크게 등)
	Preferences-002	테마	- 다크 / 라이트 모드 등 테마 색상 설정
	Preferences-003	사이드바	- 사이드바에 표시할 메뉴 설정 (메세지, 채널, 홈, 내활동, 알림 목록 등 표시할 메뉴 버튼 개별 활성화 / 비활성화)
5	환경 설정	Preferences-004	메세지 - 메세지 표시 방식 설정 (프로필 사진, 시간, 이름, 내용 등 모두 표시 / 시간 이름 내용 등 간단한 표시) - 시간 표시 방식 (12시간제 / 24시간제)
	Preferences-005	오디오 및 비디오	- 카메라 연결 설정 - 오디오 입력 및 출력 장치 연결 설정 - 마이크 음소거 설정 - 마이크 / 스피커 볼륨 설정 - 오디오 및 비디오 테스트
	Preferences-006	알림	- 웹 알림 권한 요청 - 알림 끄기 / 켜기 - 표시할 알림 범위 설정

# ERD



# 요구 사항 정의서

## IMAGE

식별자	요구 사항 명	테이블 명	타입	Null 허용	기본값	코멘트
PK	이미지 번호	IMG_NO	NUMBER	N		
	이미지 원래 이름	IMG_ORIGINALNAME	VARCHAR2(100)	N		
	이미지 새로운 이름	IMG_RENAME	VARCHAR2(100)	N		
	이미지 경로	IMG_PATH	VARCHAR2(100)	N		
	구분자	IMG_SEPARATOR	CHAR(1)	N		CHECK('M', 'C', 'D')
	멤버/채팅방/DM 번호	MCD_NO	NUMBER	N		

## REPORT

식별자	요구사항 명	테이블 명	타입	Null 허용	기본값	코멘트
PK	신고번호	REPORT_NO	NUMBER	N		
FK	신고 카테고리 번호	REPORT_CATEGORY_NO	NUMBER	N		
FK	신고자	REPORTER	NUMBER	N		
FK	위반자	VIOLATOR	NUMBER	N		
	신고 날짜	REPORT_DATE	DATE	N		
	신고 조치여부	REPORT_STATUS	CHAR(1)	N	'W'	CHECK('R'. 'W', 'A')
	채팅 번호	CHAT_NO	NUMBER	N		

## SERVER\_MEMBER

식별자	요구사항 명	테이블 명	타입	Null 허용	기본값	코멘트
PK, FK	회원번호	MEMBER_NO	NUMBER	N		
PK, FK	서버 번호	SERVER_NO	NUMBER	N		
	관리자 여부	IS_ADMIN	CHAR(1)	N	'N'	CHECK('N', 'Y')

## CHANNEL

식별자	요구사항 명	테이블 명	타입	Null 허용	기본값	코멘트
PK	채널 번호	CHANNEL_NO	NUMBER	N		
FK	서버 번호	SERVER_NO	NUMBER	N		
	채널 이름	CHANNEL_NAME	VARCHAR2(30)	N		
	채널 생성 날짜	CHANNEL_CREATEDATE	DATE	N	SYSDATE	
	채널 구분	CHANNEL_SEPARATOR	CHAR(1)	N		CHECK('V', 'T')
	채널 활성화 여부	CHANNEL_STATUS	CHAR(1)	N	'Y'	CHECK('Y', 'N')

MEMBER						
식별자	요구 사항 명	테이블 명	타입	Null 허용	기본값	코멘트
PK	회원번호	MEMBER_NO	NUMBER	N		
	멤버 아이디	MEMBER_ID	VARCHAR2(30)	N		
	멤버 비밀번호	MEMBER_PWD	VARHCAR2(100)	N		
	멤버 성별	MEMBER_GENDER	CHAR(2)	N		CHECK('M', 'F')
	멤버 핸드폰번호	MEMBER_PHONE	NUMBER	N		
	멤버 생년월일	MEMBER_BIRTH	NUMBER	N		
	멤버 이메일	MEMBER_EMAIL	VARCHAR2(100)	N		
	멤버 닉네임	MEMBER_NICKNAME	VARCHAR2(50)	N		
	멤버 활성화 여부	MEMBER_STATUS	CHAR(1)	N	'Y'	CHECK('Y', 'N')
	관리자 여부	IS_ADMIN	CHAR(1)	N	'N'	CHECK('Y', 'N')
	멤버 생성날짜	MEMBER_CREATEDATE	DATE	N	SYSDATE	

화면명

로그인 화면

REAL MAN

로그인 화면

**Surprisingly,  
this service is not an exerciseservice,  
but a team cooperative  
communication service**



② Sign up

Enter Email

.....

③ Forgot ID?

④ Forgot Password?

1 Sign In

Or continue

G Apple F

Project

1

2

3

4

Real Man

로그인 버튼

회원가입 페이지로 넘어가는 버튼

아이디 찾기 페이지로 넘어가는 버튼

비밀번호 페이지로 넘어가는 버튼

화면 영

회원가입 화면

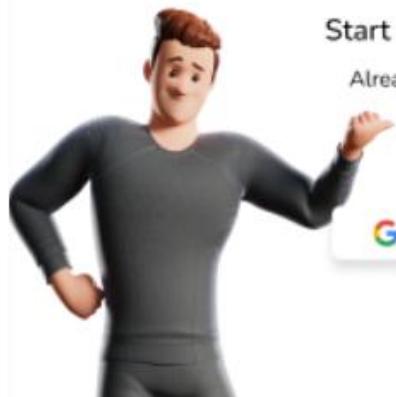
Real Man

회원가입



Start testing in minutes ! ...

Already have an account? [Log in](#)



ID	<input type="text"/>
Password	<input type="password"/> <small>•••</small>
Confirm Password	<input type="password"/> <small>•••</small>
Name	<input type="text"/>
Phone	<input type="text"/>
Email	<input type="text"/>

I agree to the [Terms of Service](#) and [Privacy Notice](#)

[Create Account](#)

Project

Real Man

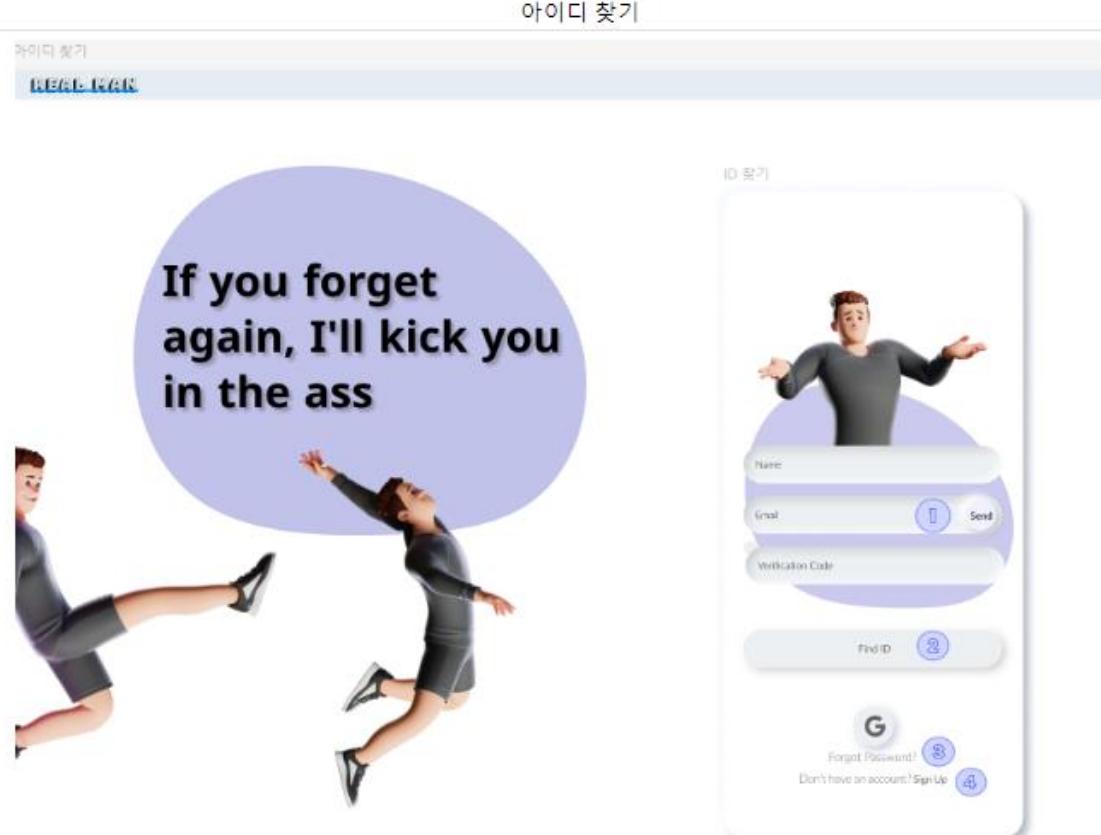
1

로그인 페이지로 이동

2

계정 생성이 되고 로그인 페이지로 이동

## 화면 명



## 아이디 찾기

## Project

## Real Man

1	이메일 인증 버튼을 누르면 인증 이메일이 날아가는 버튼
2	아이디 찾기 버튼 누르면 성공, 실패 화면으로 넘어가는 버튼
3	비밀번호 찾기 페이지로 넘어가는 버튼
4	회원가입 페이지로 넘어가는 버튼

화면 명



내 정보

Project

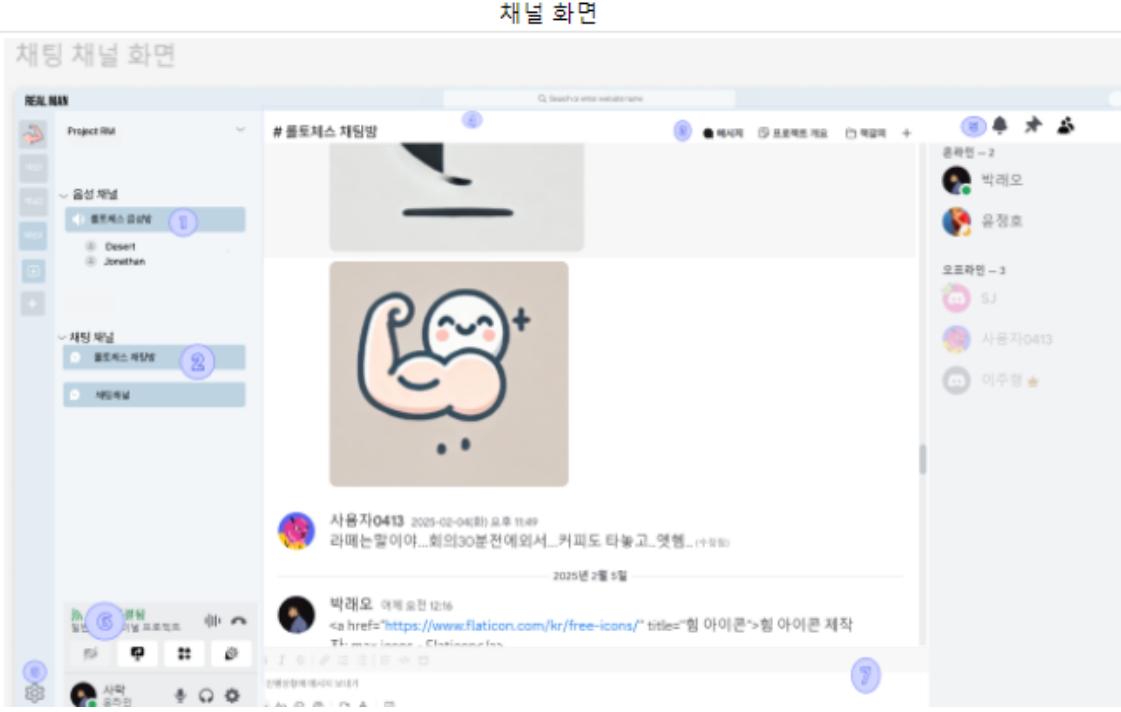
Real Man

- | Project | Real Man              |
|---------|-----------------------|
| 1       | 내 계정을 조회할수 있는 페이지로 이동 |
| 2       | 별명을 변경할 수 있는 창 활성화    |
| 3       | 사용자 명을 변경할 수 있는 창 활성화 |
| 4       | 이메일 변경할 수 있는 창 활성화    |
| 5       | 전화번호 변경할수 있는 창 활성화    |
| 6       | 비밀번호 변경할수 있는 창 활성화    |
| 7       | 계정 비활성화 할수 있는 창 활성화   |
| 8       | 계정 삭제 할수 있는 창 활성화     |

20

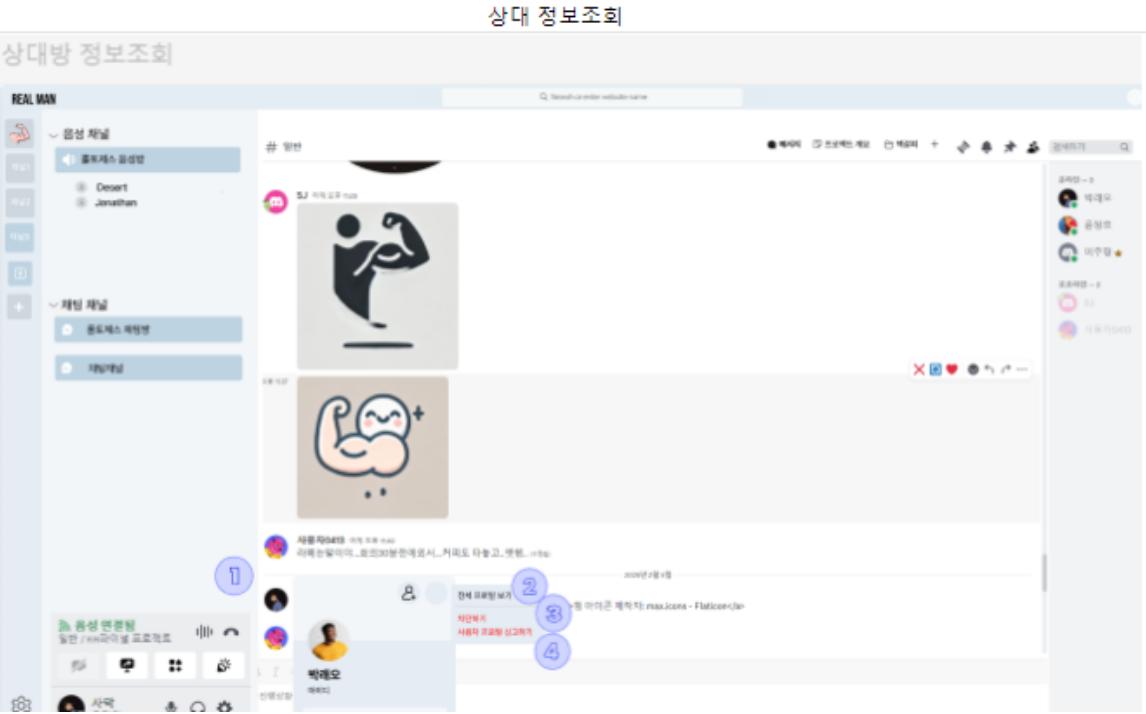
3/26/2025

## 화면 명



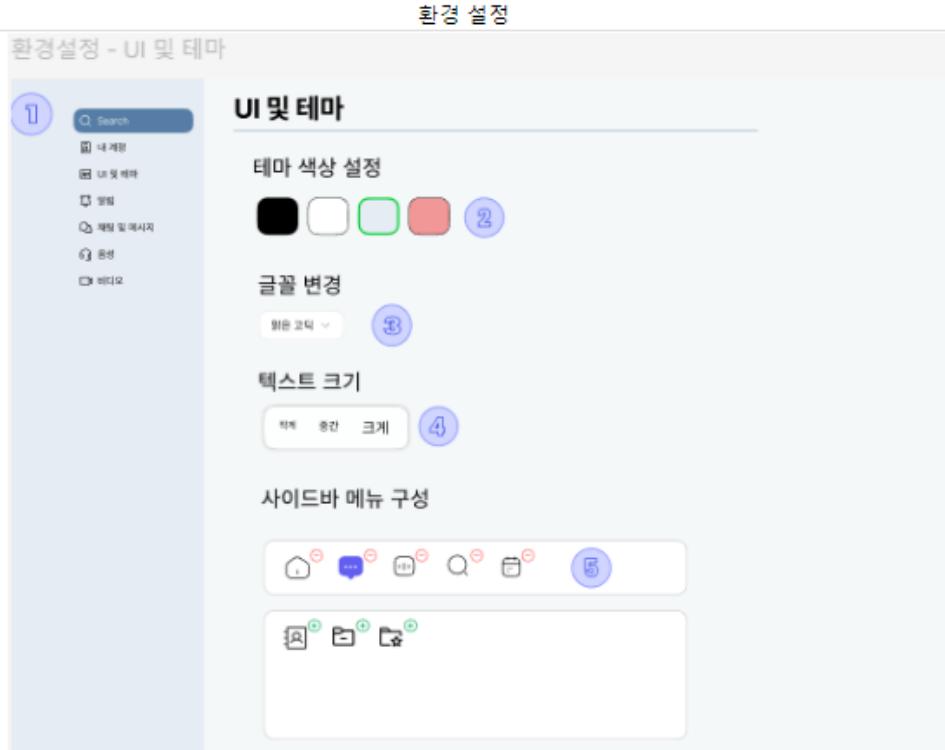
Project	Real Man
1	음성 채널 방으로 입장
2	채팅 채널을 볼 수 있는 창으로 넘어감
3	DM 메세지, 프로젝트 개요, 책갈피를 확인할 수 있는 창으로 넘어감
4	탭바 검색창 채널에 있는 메세지를 검색할 수 있게 함
5	알림, 고정pin, 유저 목록을 확인할 수 있는 창으로 넘어감
6	화면 공유, 화상통화, 마이크 활성화 / 비활성화와 같은 기능을 사용할 수 있습니다.
7	채팅, 첨부파일, 링크, 멘션 등의 기능을 사용할 수 있습니다.
8	설정 창 페이지로 이동

3/26/2025



Project	Real Man
1	상대 프로필에 마우스를 올리면 상대방 프로필 및 신고/차단할수있는 창 활성화
2	상대 사용자의 정보를 볼수있는 창 활성화
3	상대 사용자를 차단할 수 있는 창을 활성화
4	상대 사용자를 신고할 수 있는 창을 활성화

화면 명



환경 설정

## UI 및 테마

## 테마 색상 설정



2

## 글꼴 변경



3

## 텍스트 크기



4

## 사이드바 메뉴 구성



5



Project

	Real Man
1	환경 설정검색, UI 테마, 알림 등을 설정할 수 있는 페이지로 넘어감
2	테마 색상을 선택하여 선택한 색상에 맞게 테마를 변경
3	고를 수 있는 폰트를 스크롤로 표기
4	클릭 시 텍스트 크기에 맞게 변경
5	사이드 바 정렬 기능

## Use case

## Real man

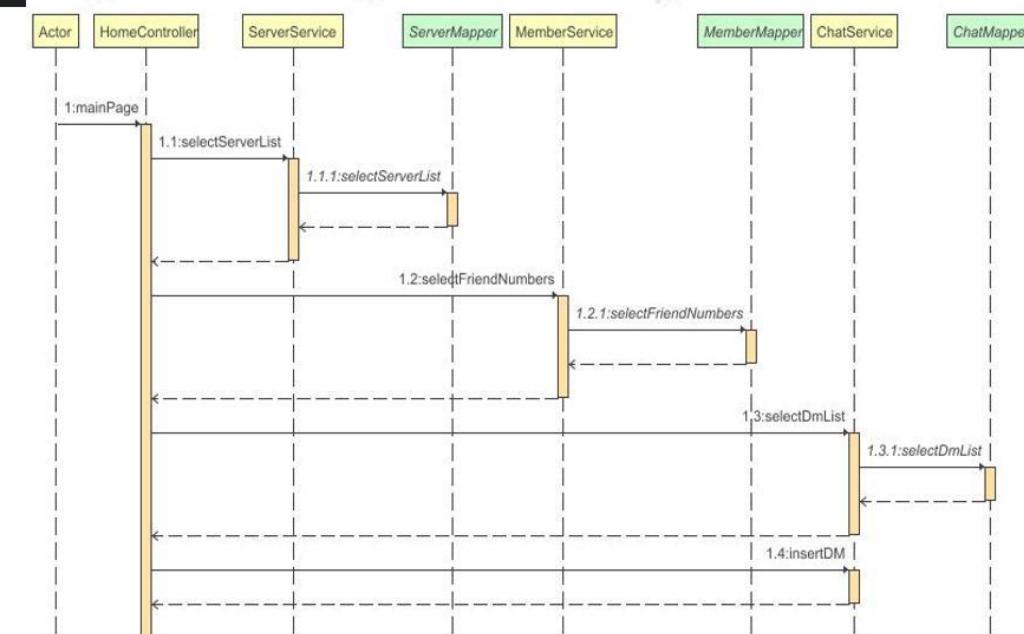
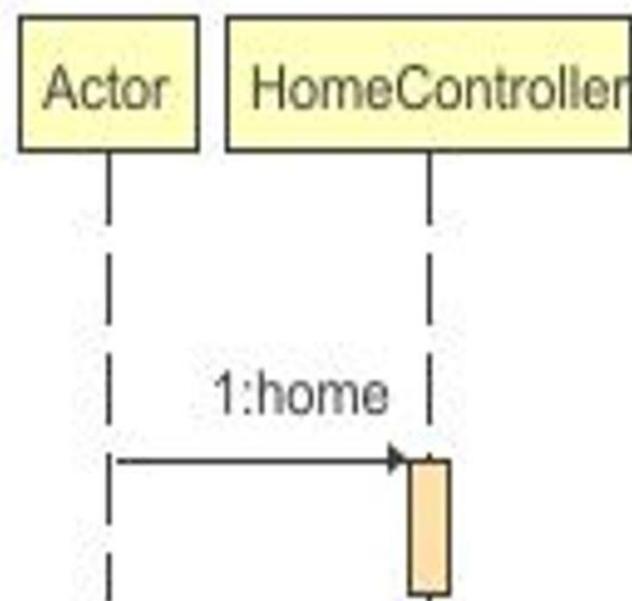
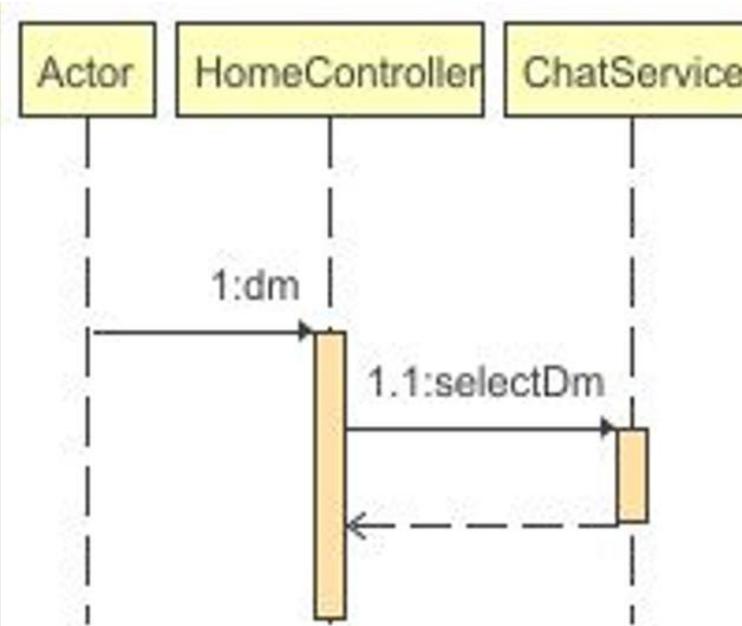
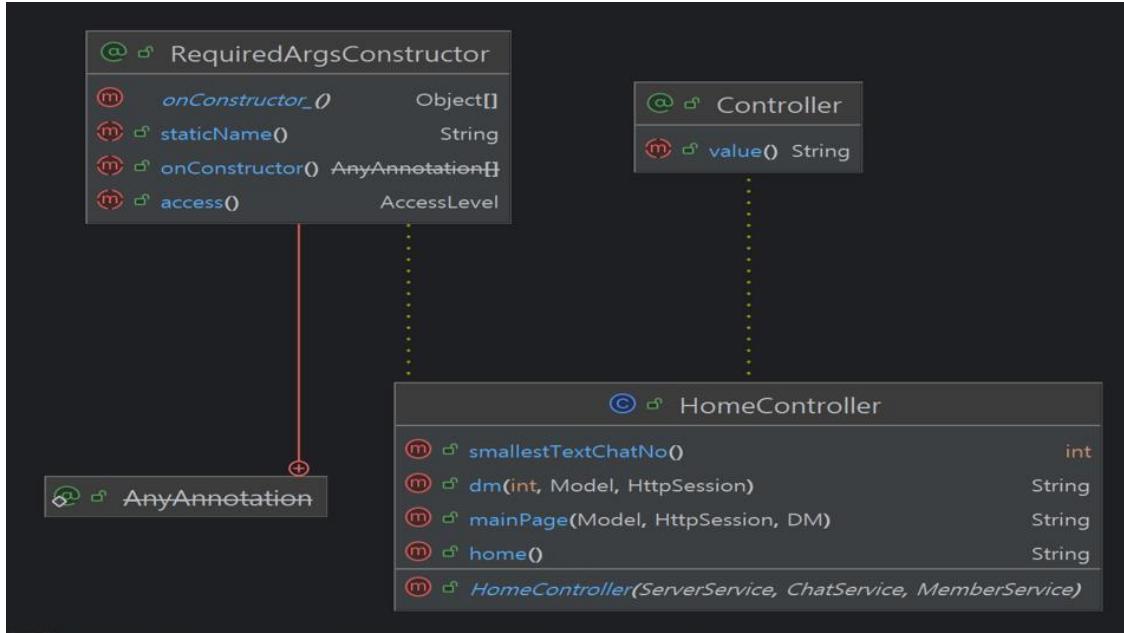


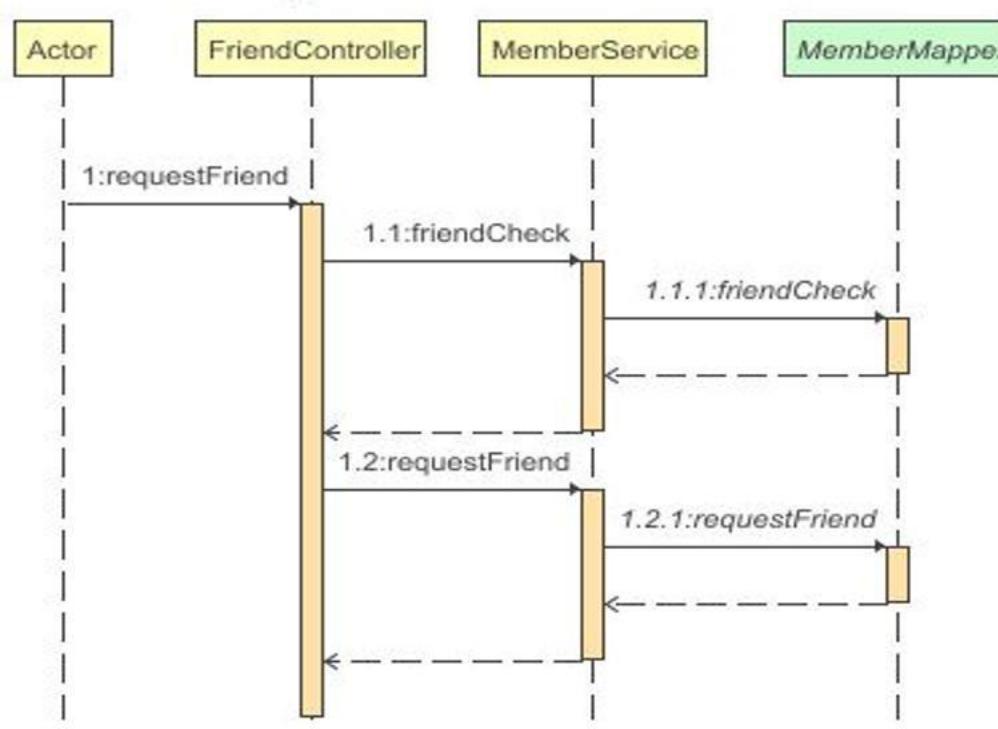
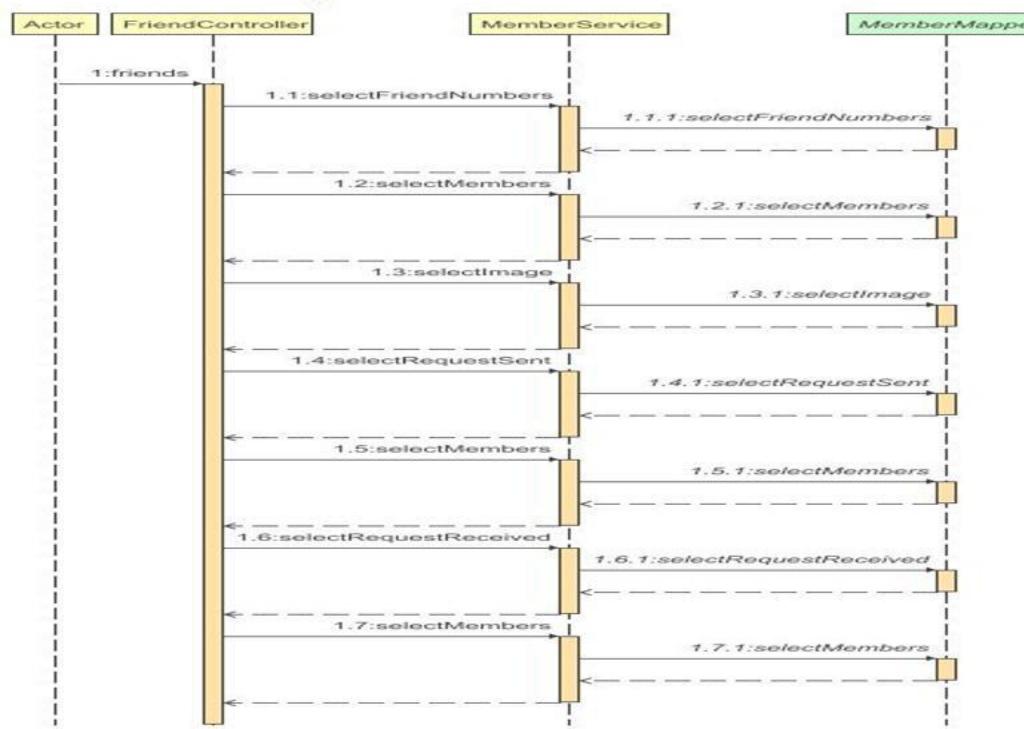
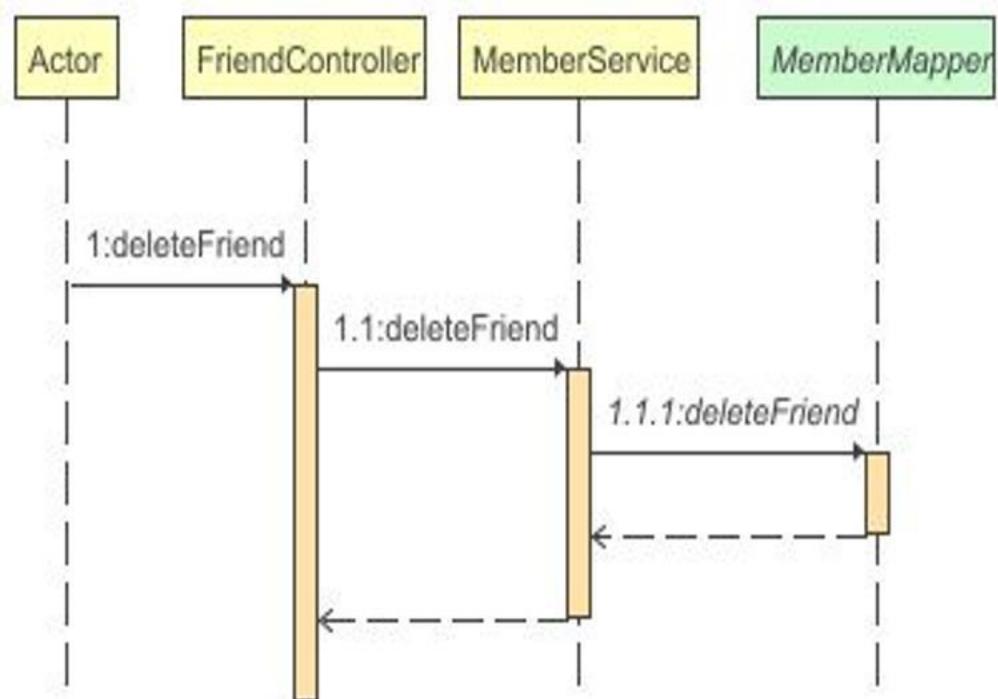
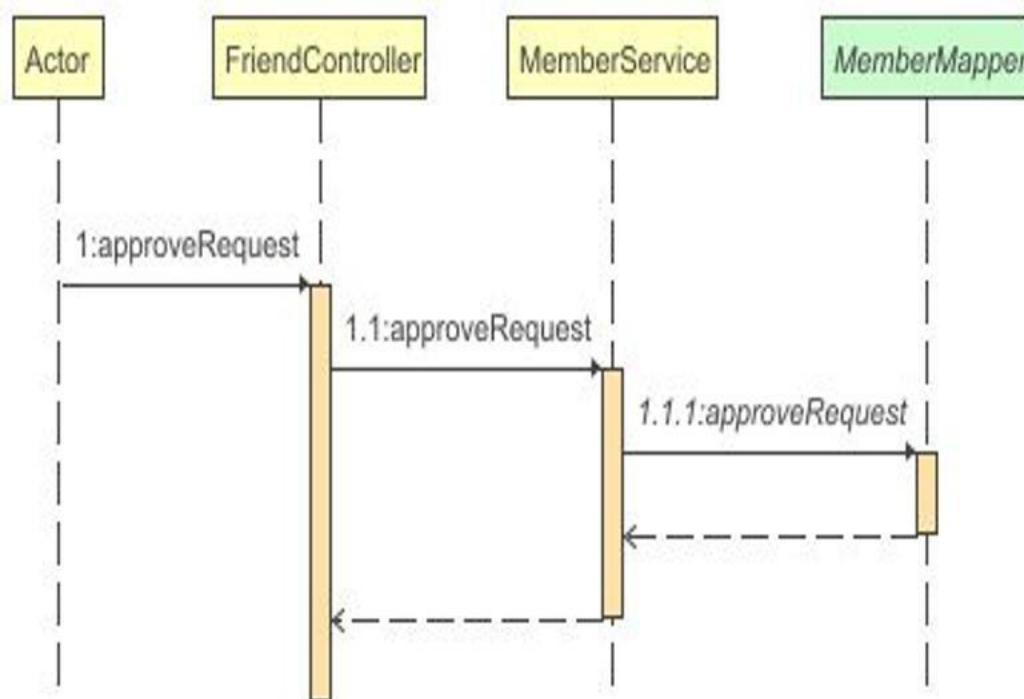
일반 사용자

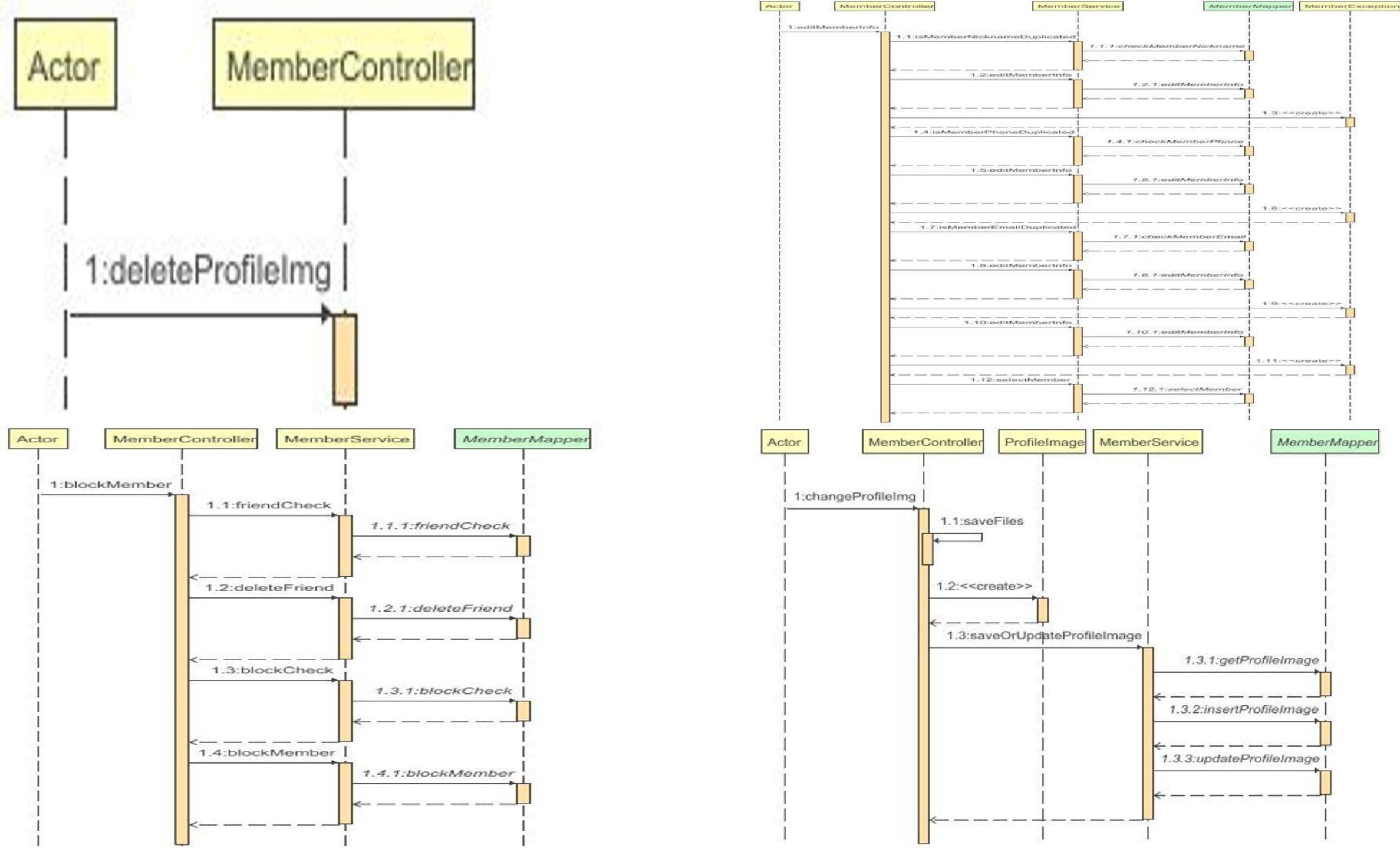


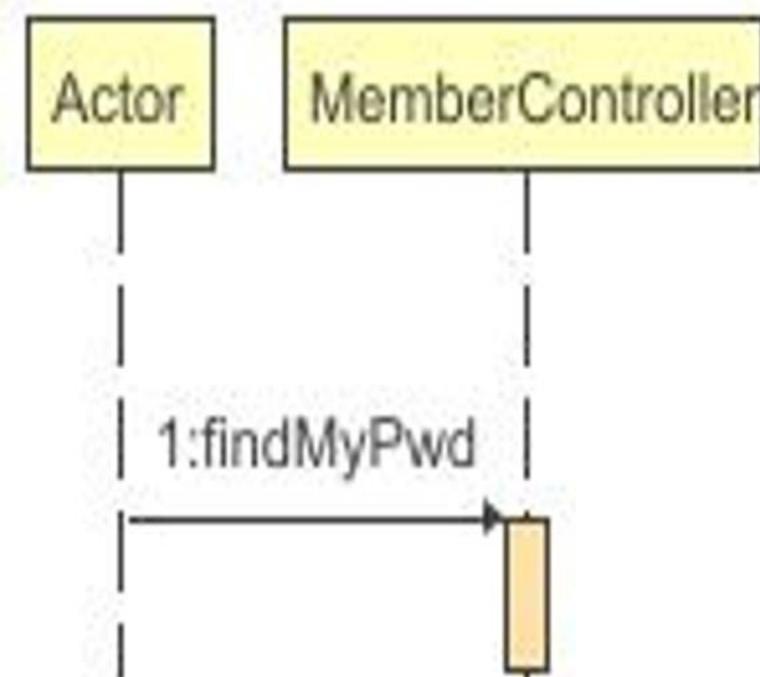
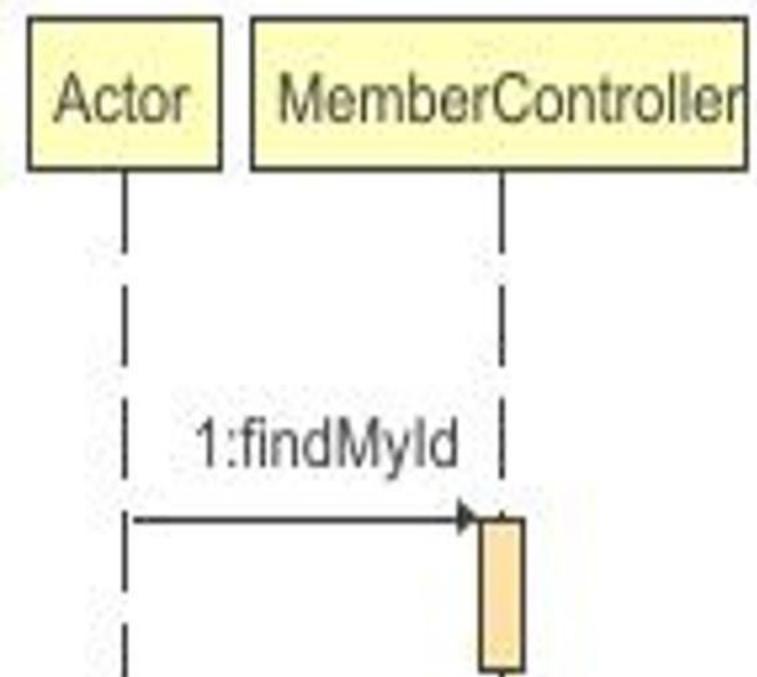
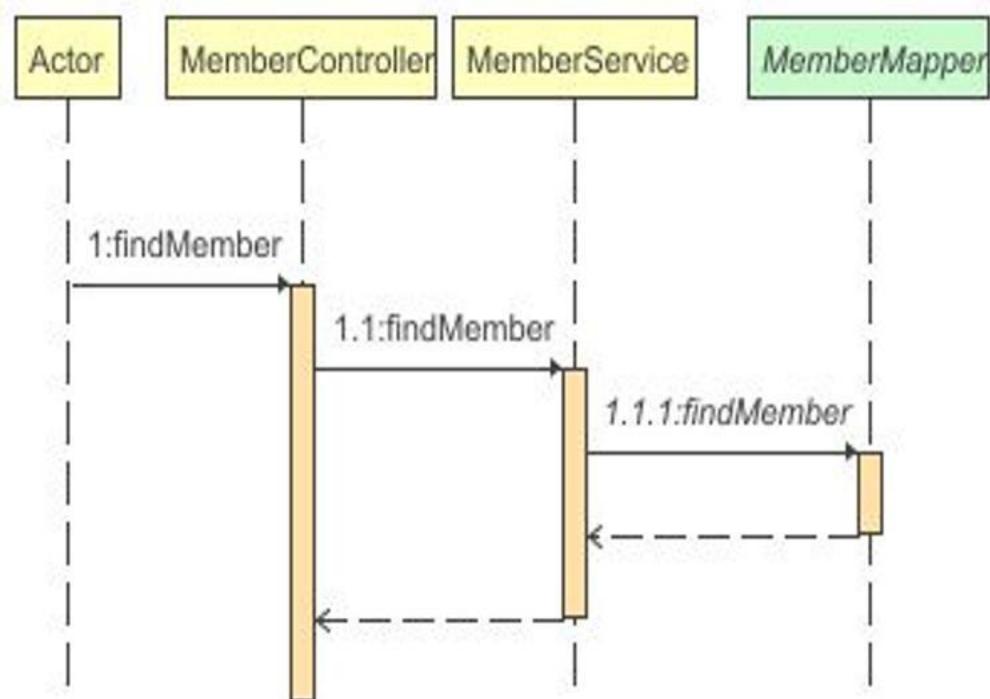
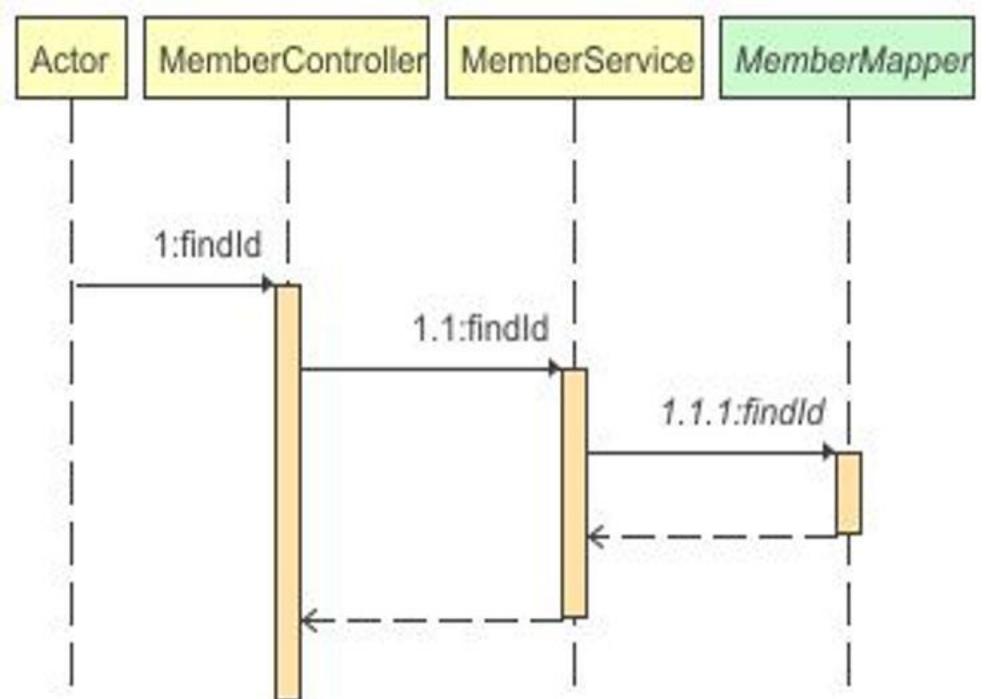
관리자

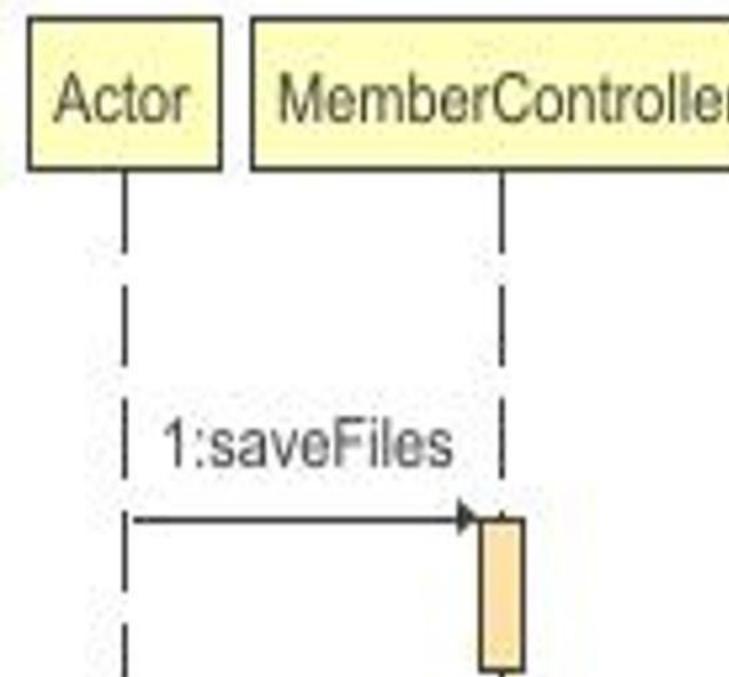
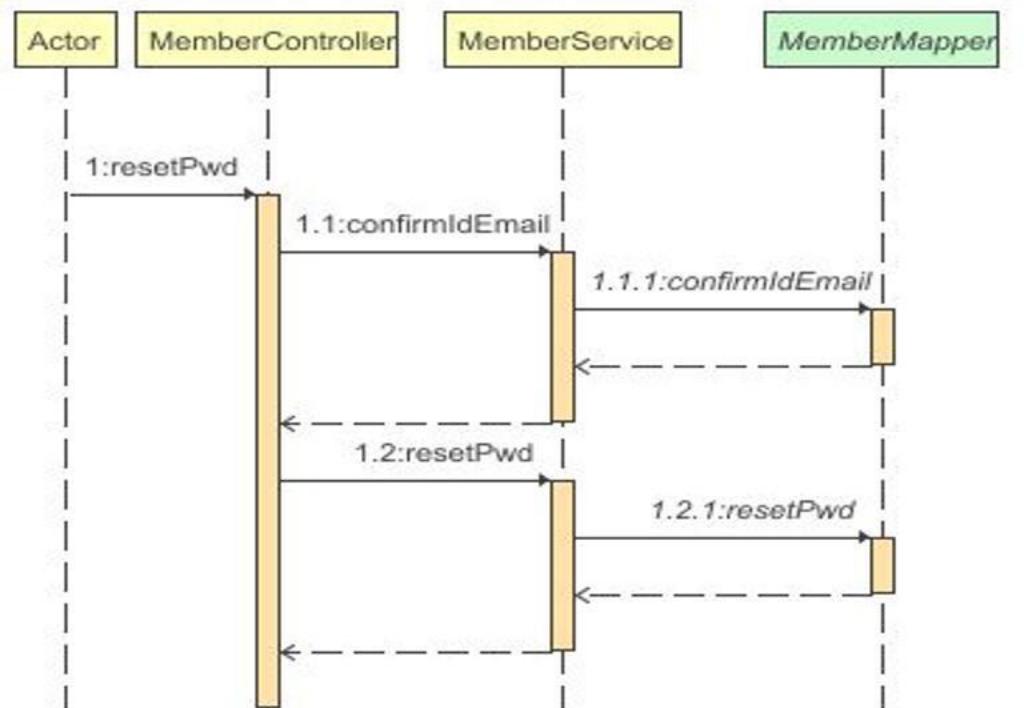
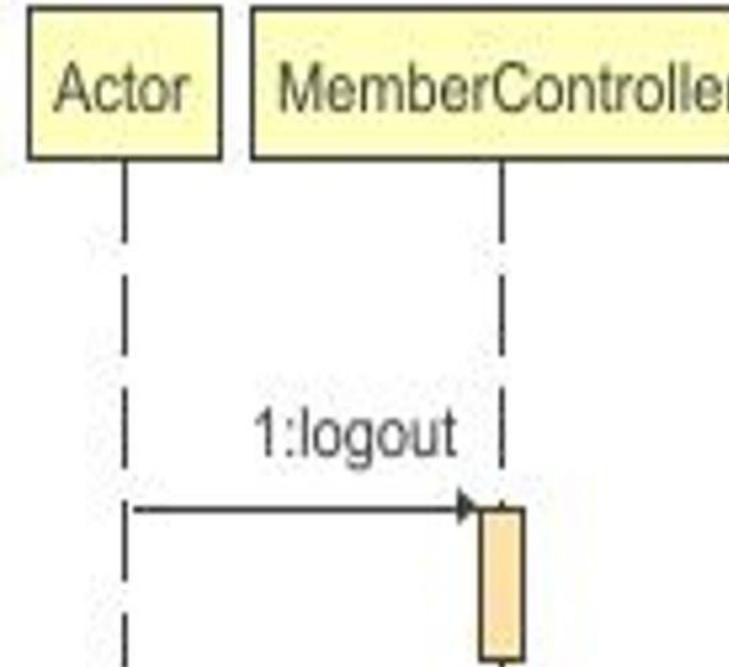
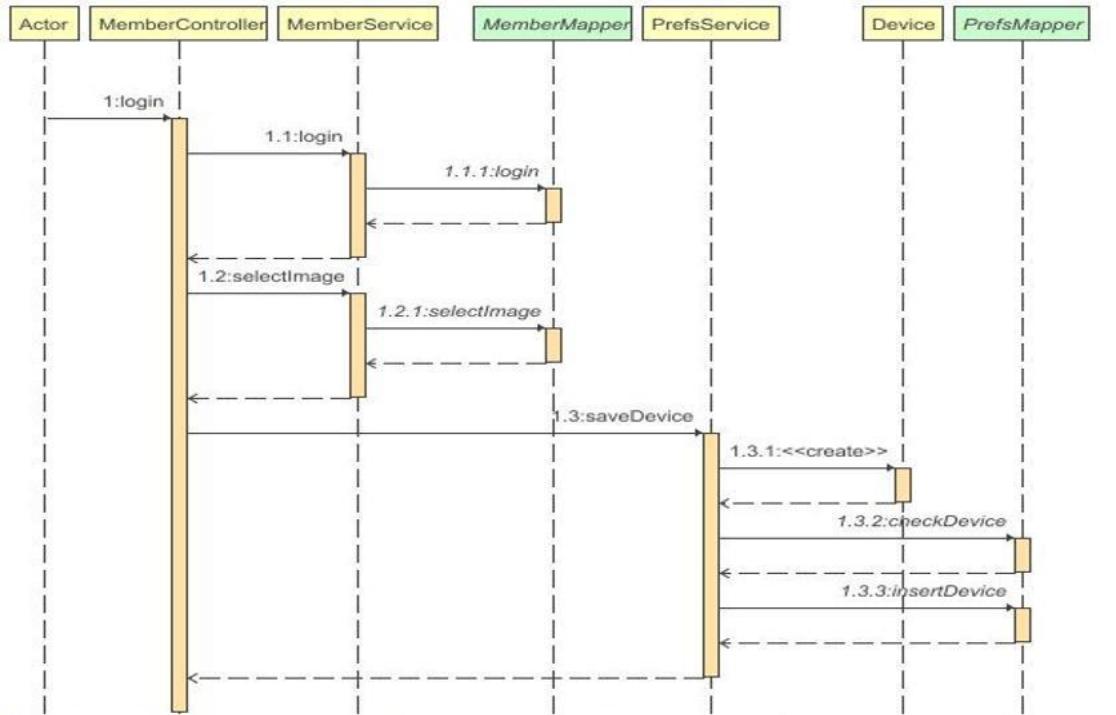
# 클래스/시퀀스 다이어그램

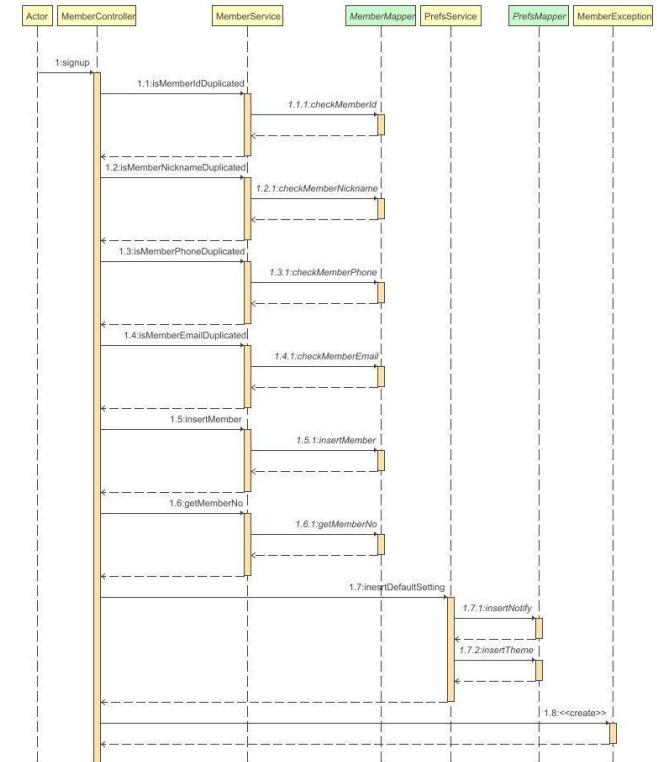
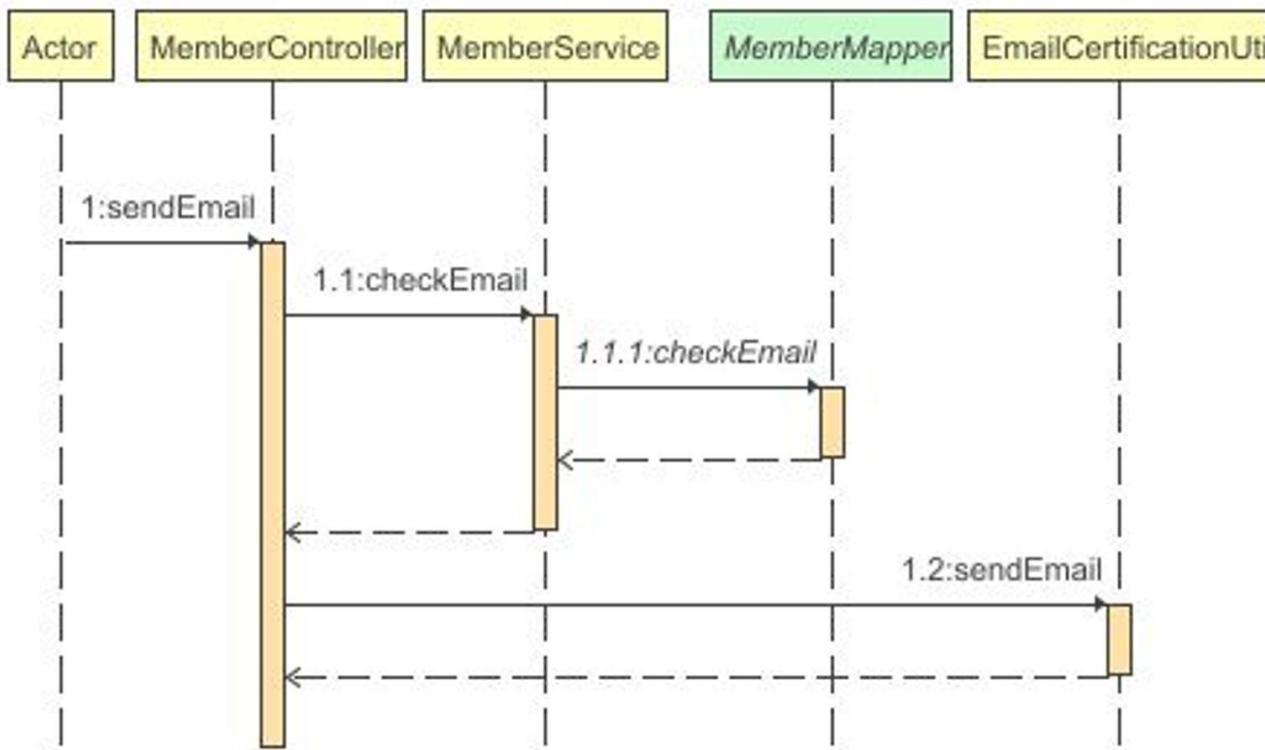


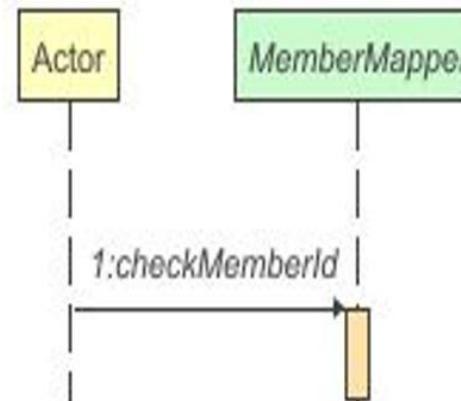
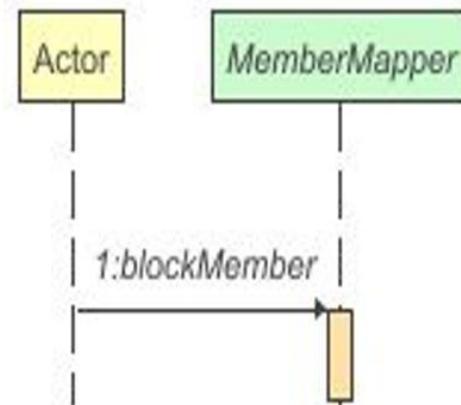
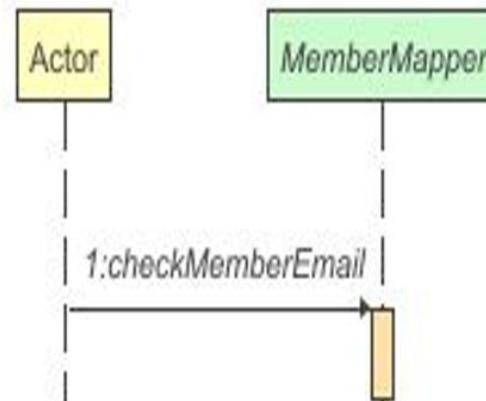
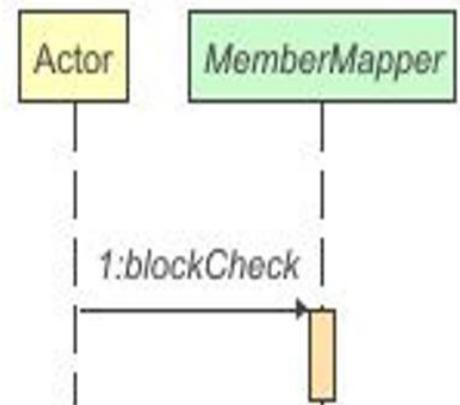
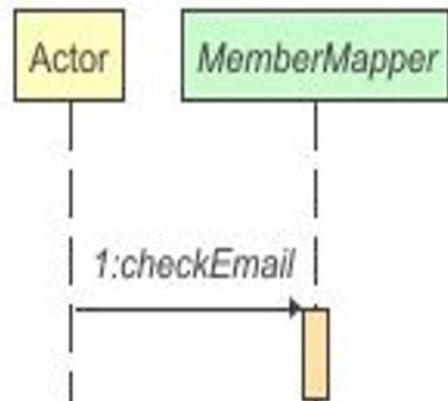
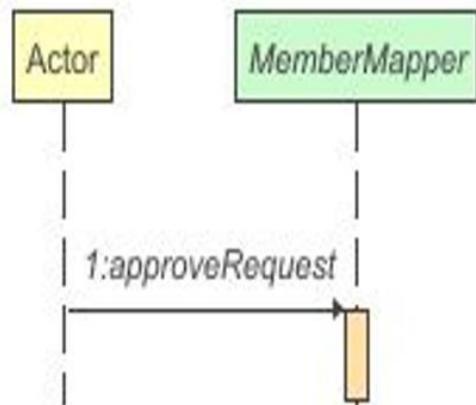


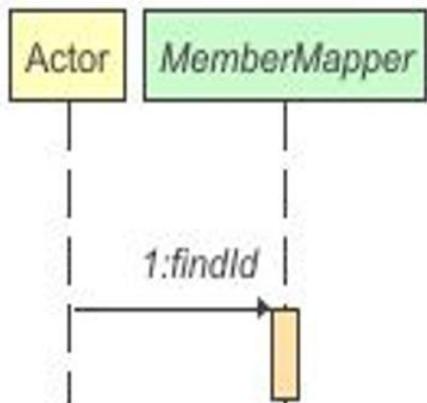
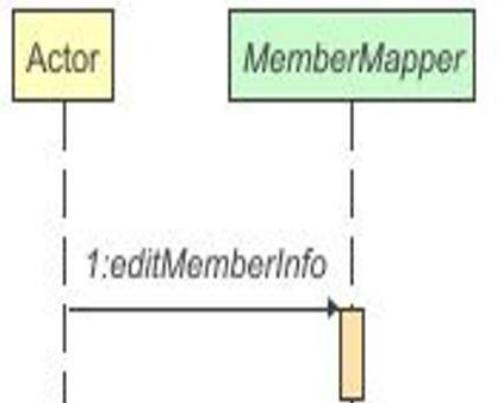
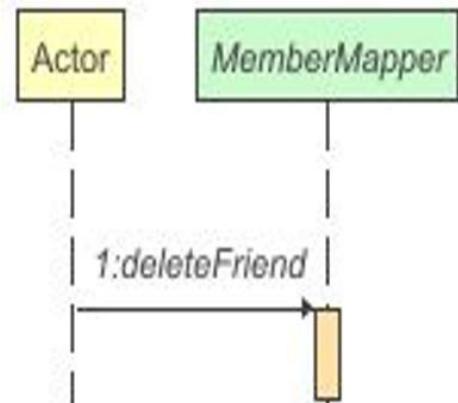
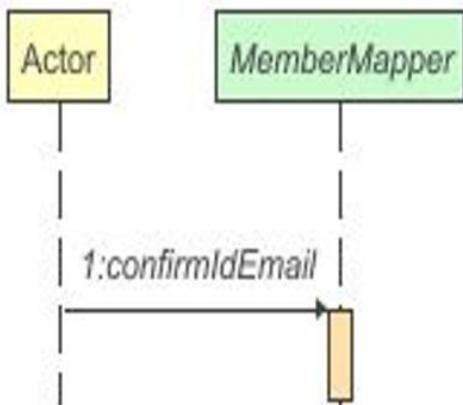
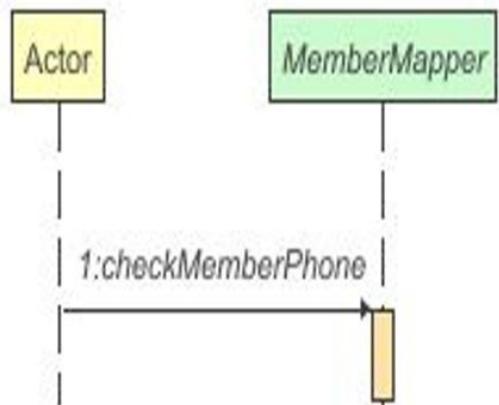
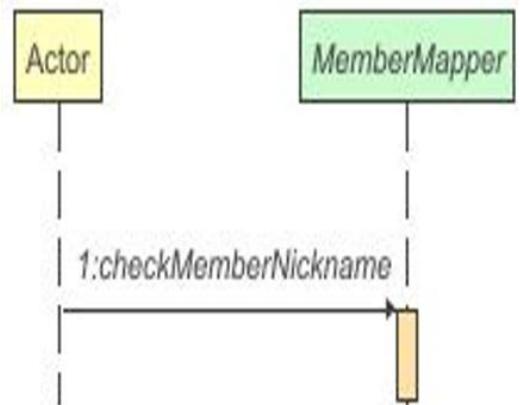


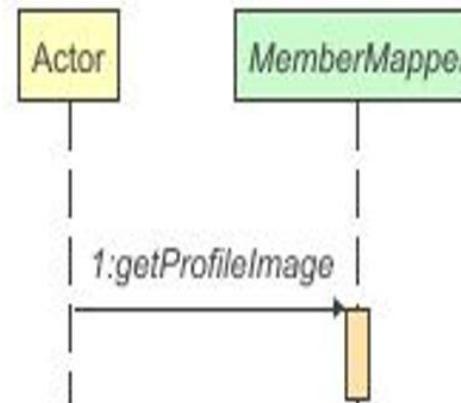
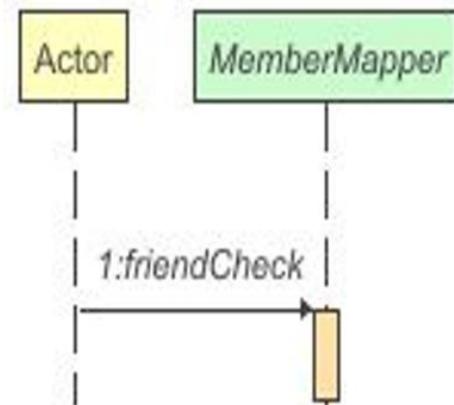
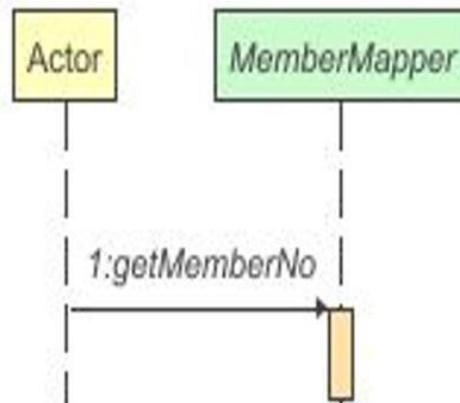
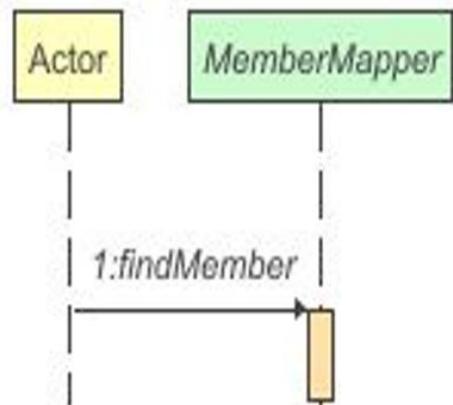
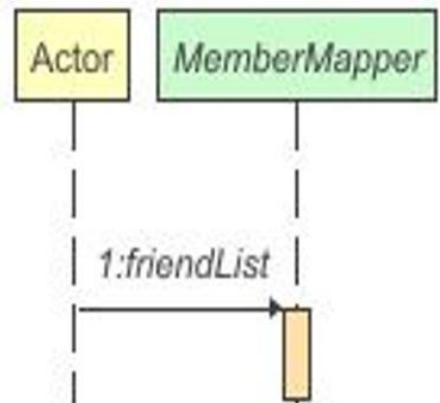
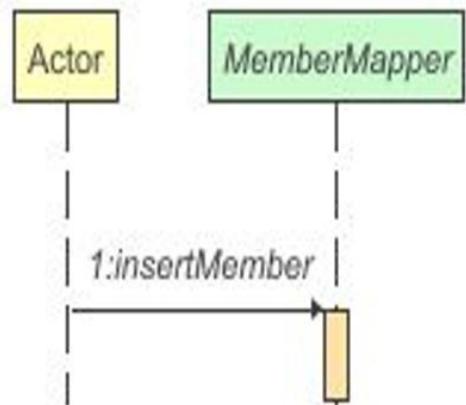


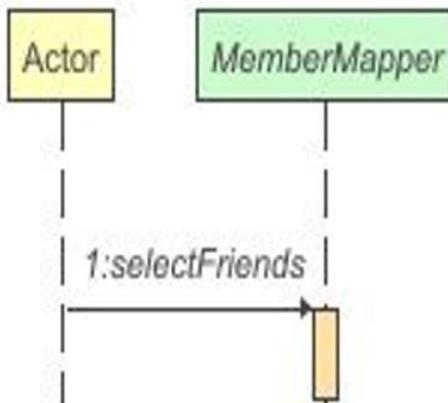
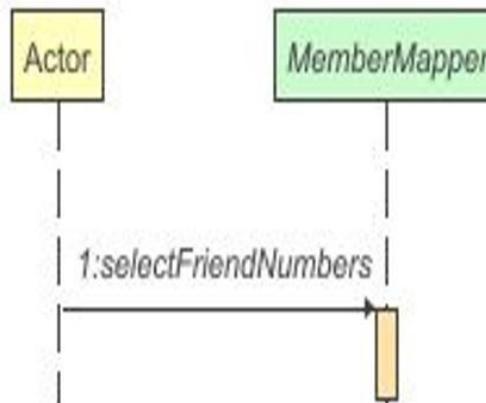
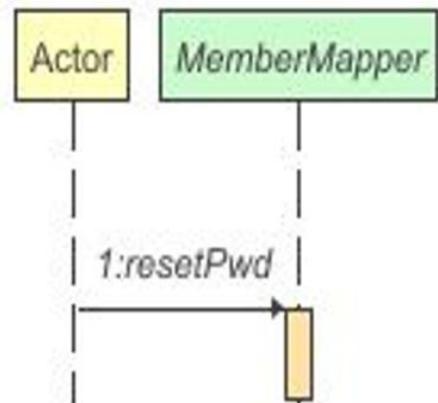
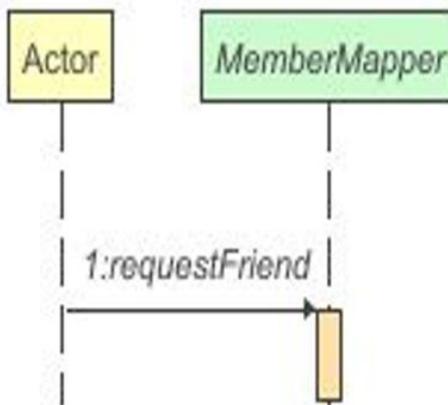
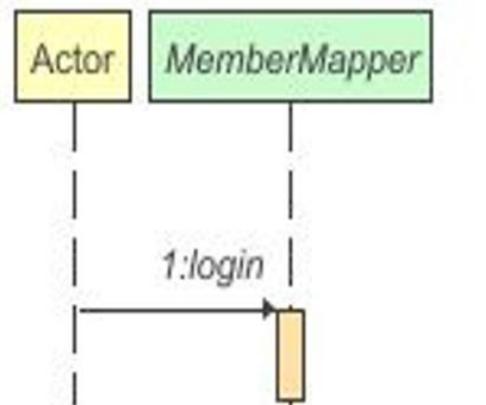
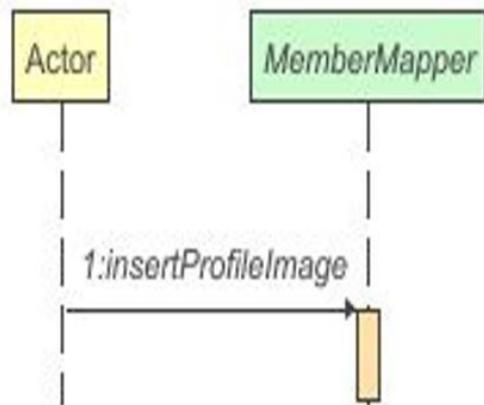


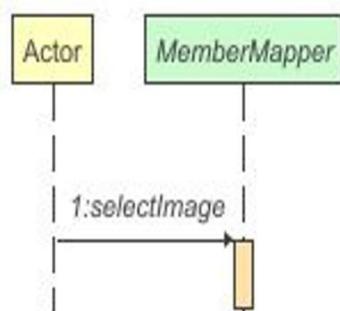
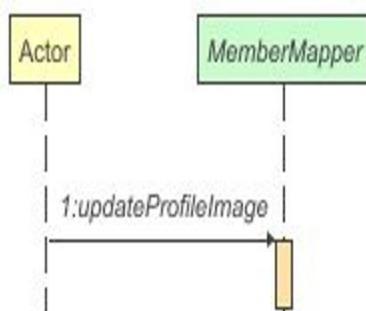
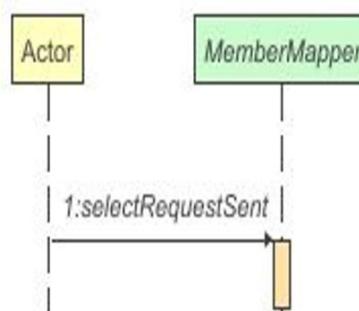
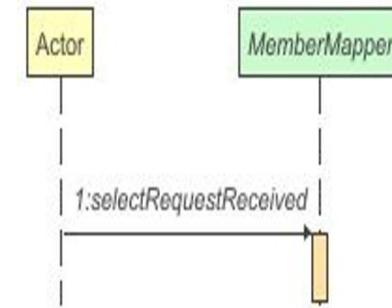
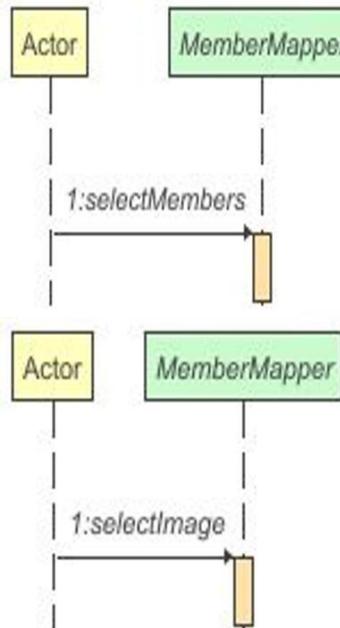
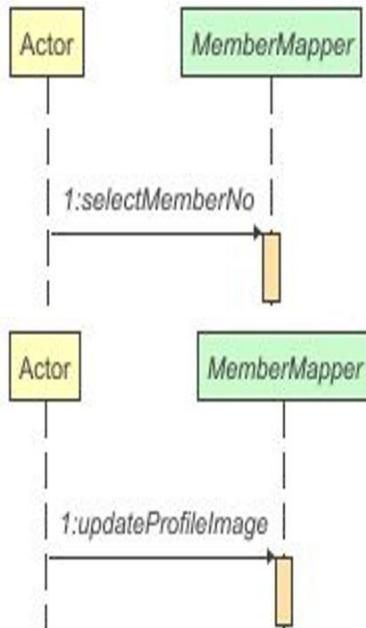
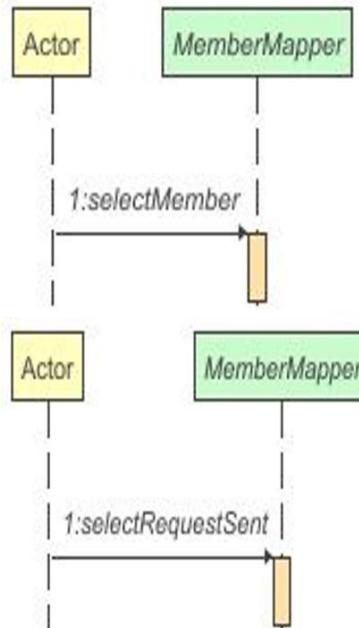


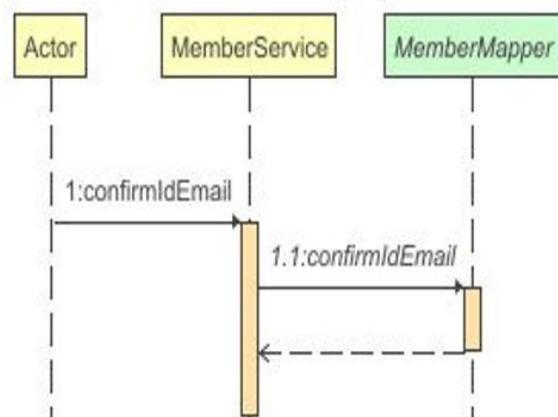
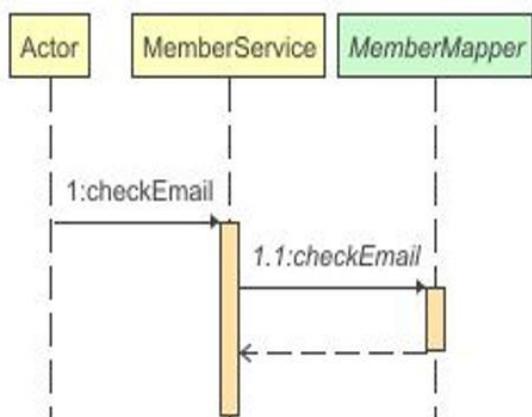
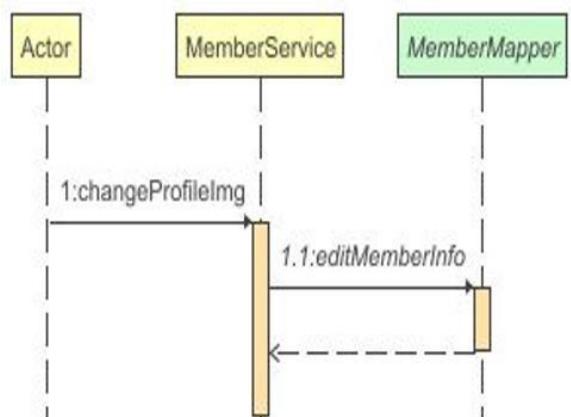
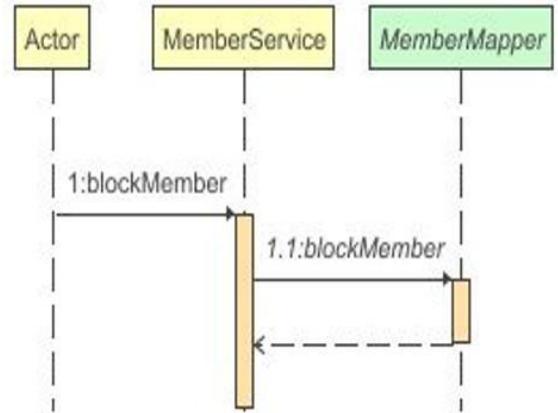
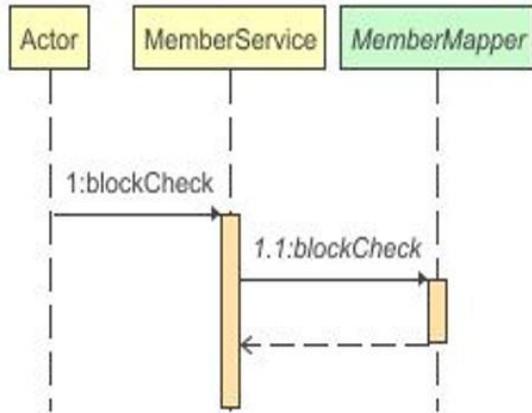
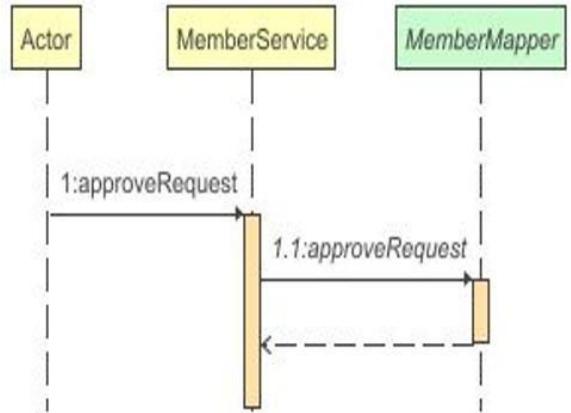


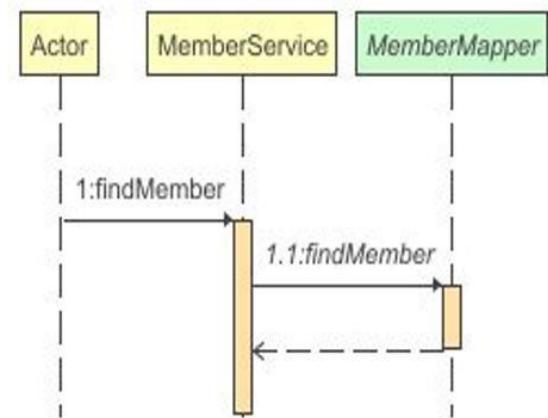
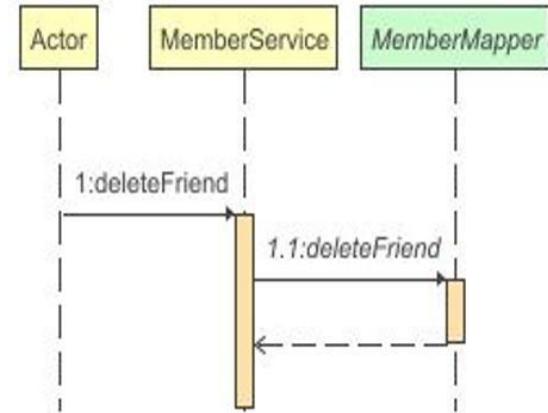
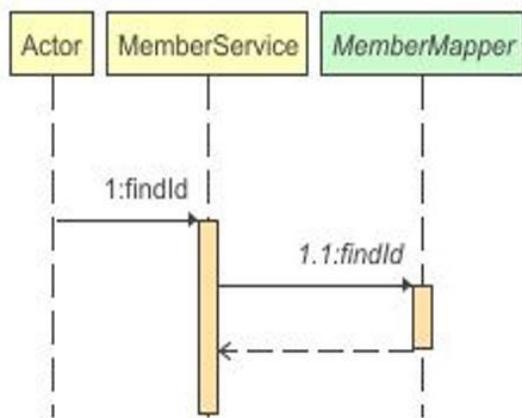
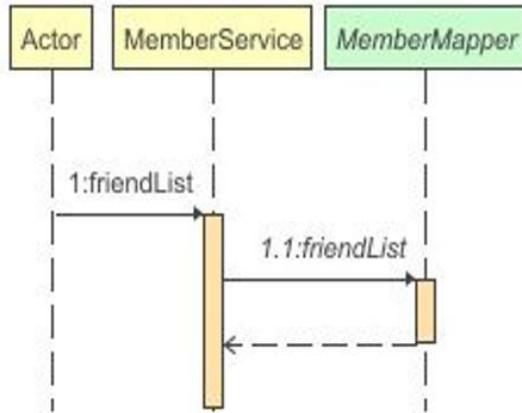
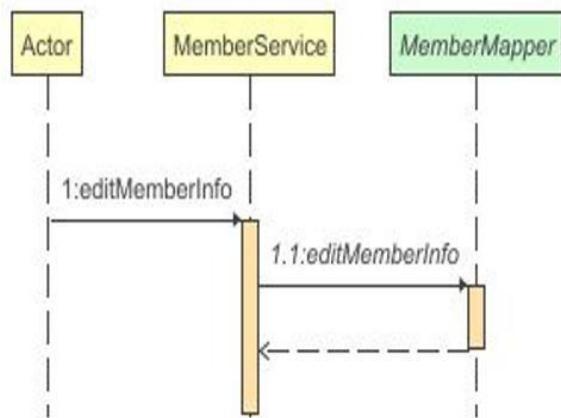
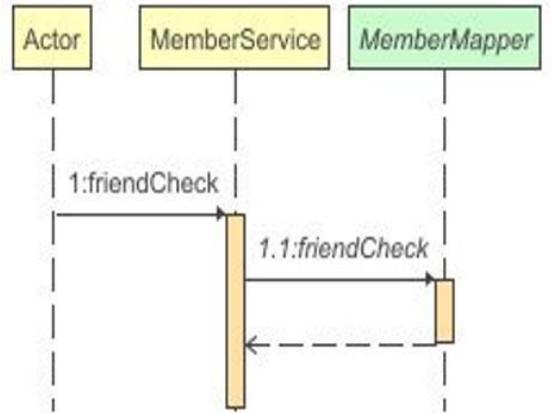


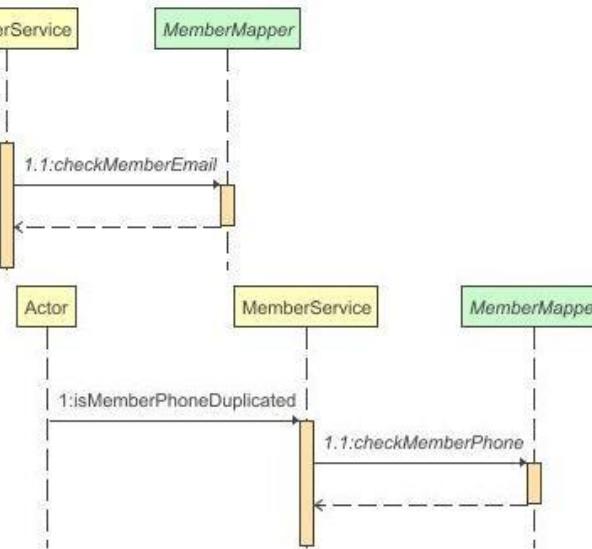
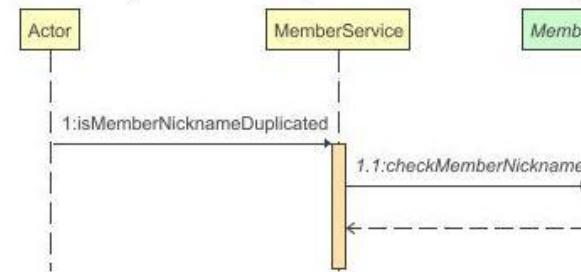
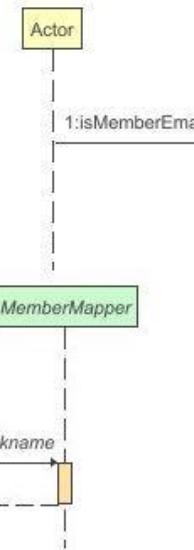
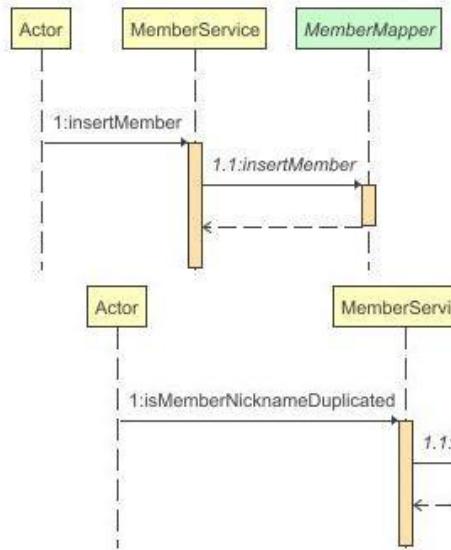
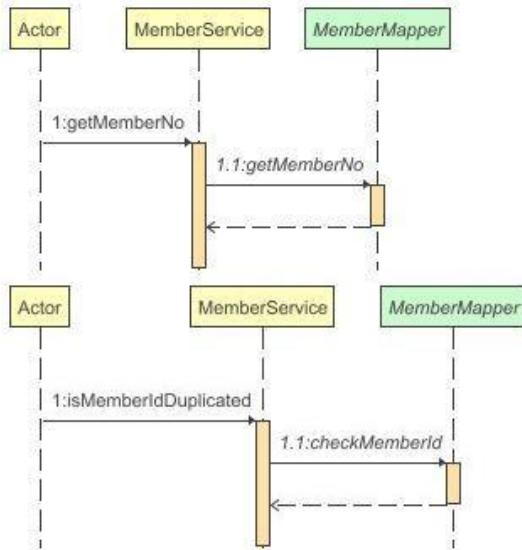


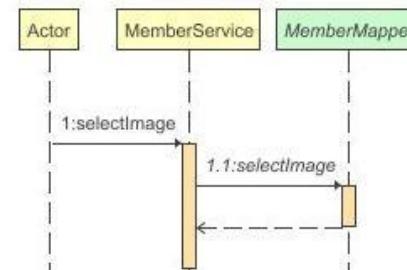
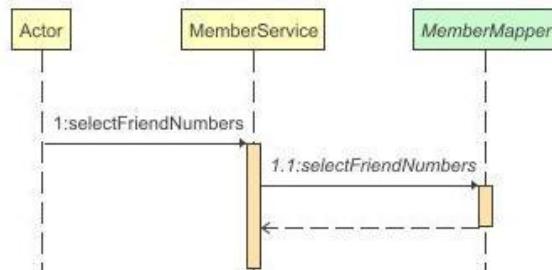
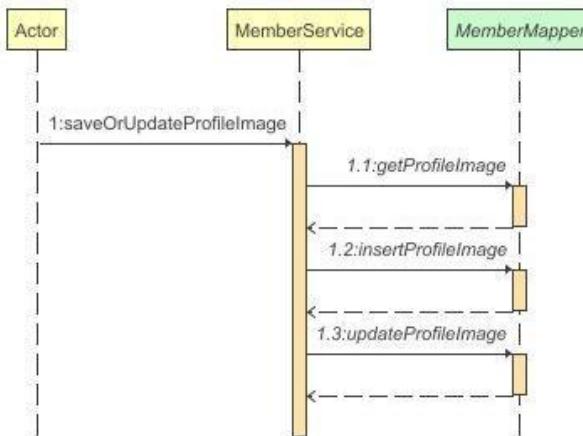
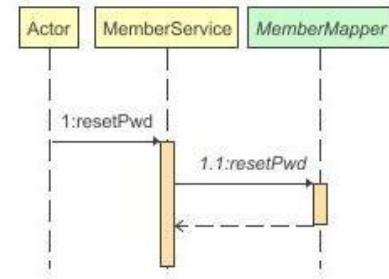
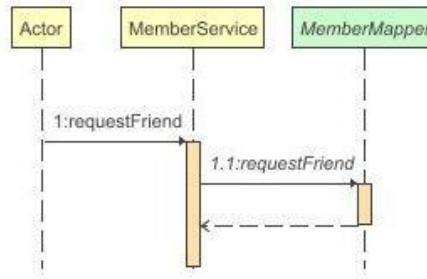
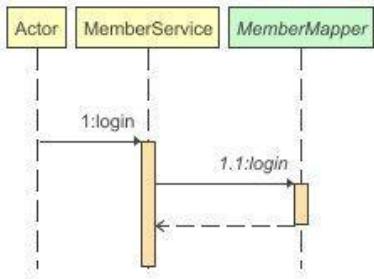


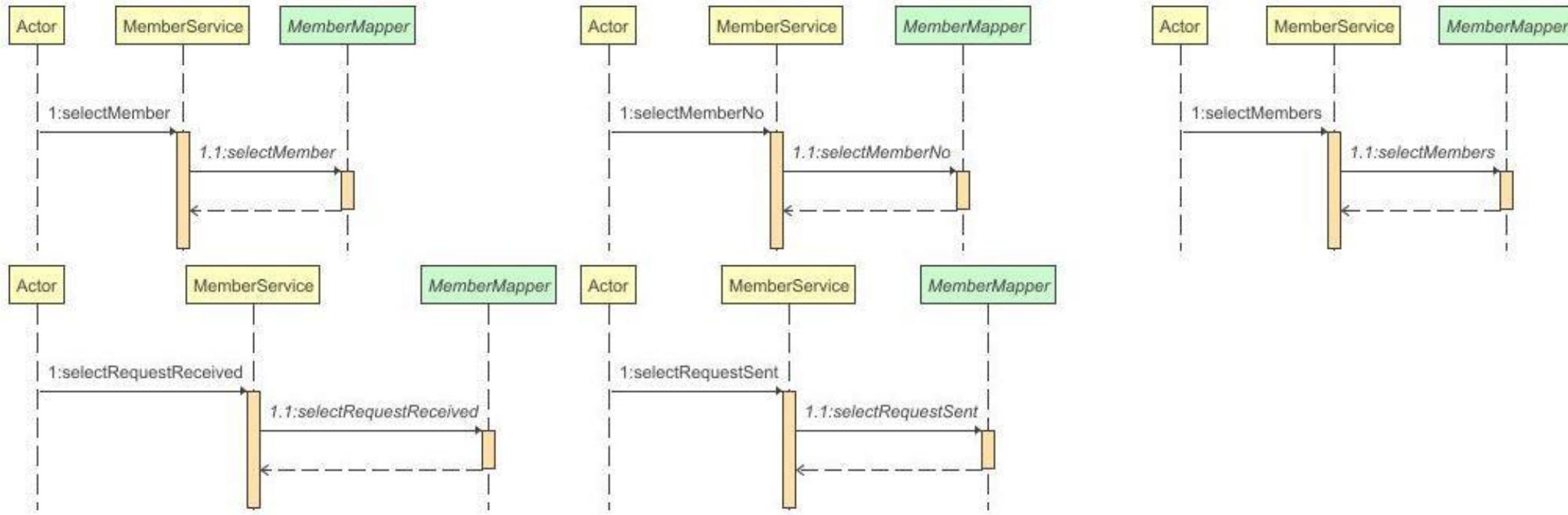


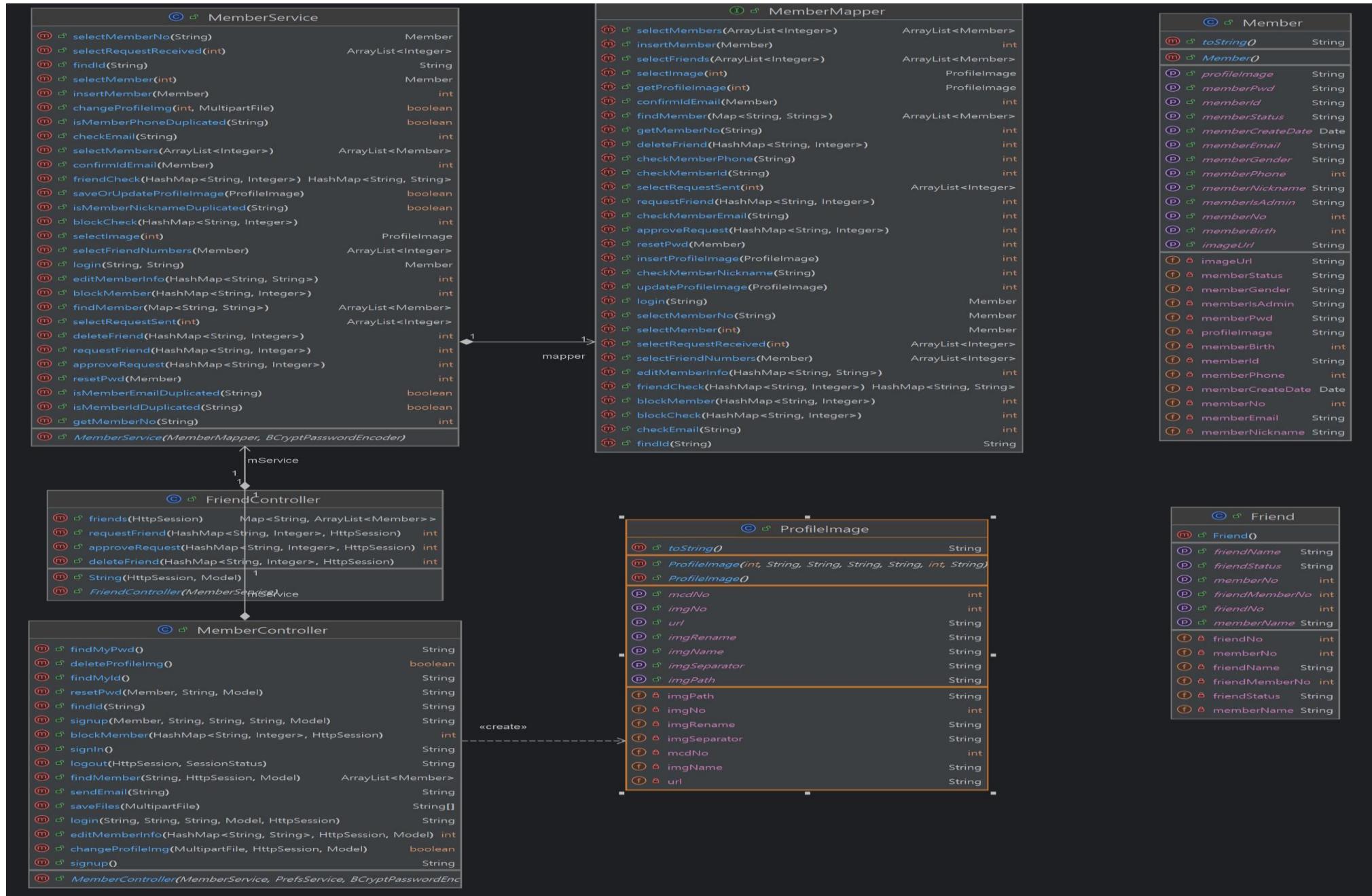


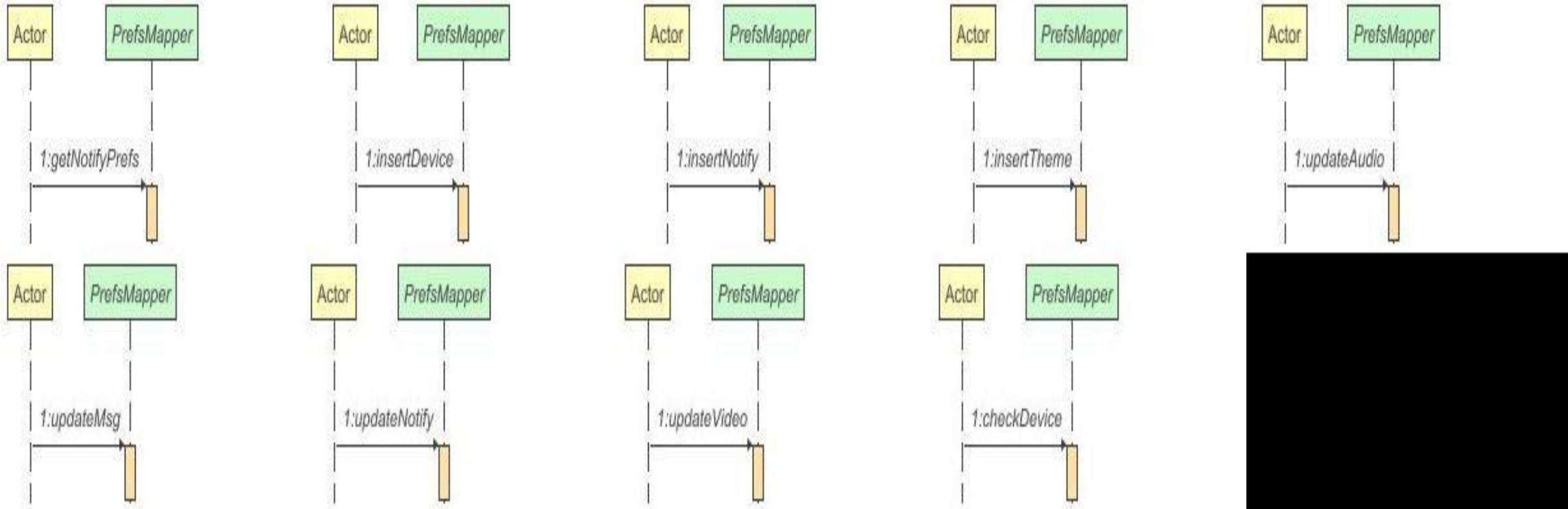


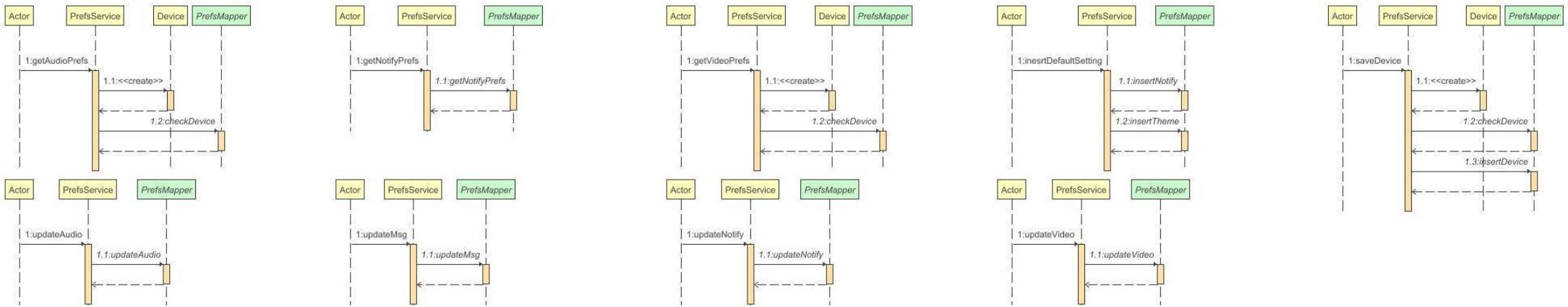


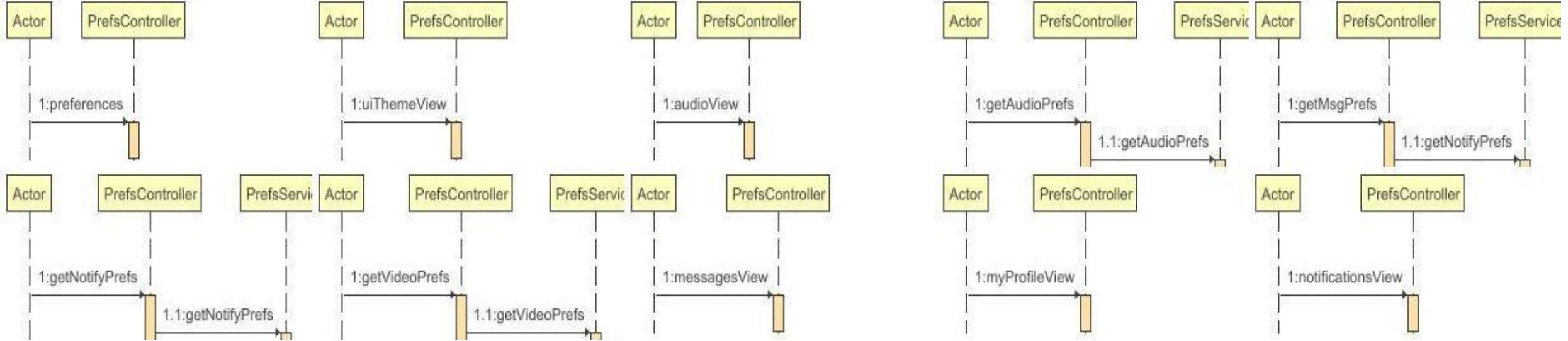


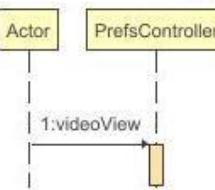
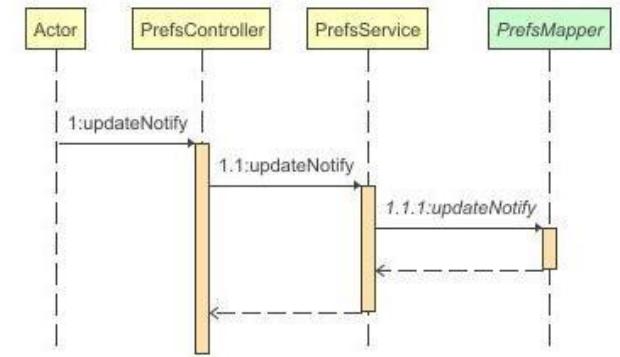
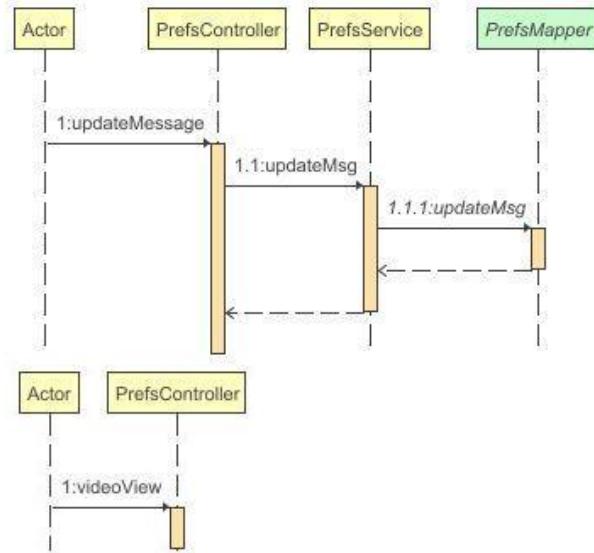
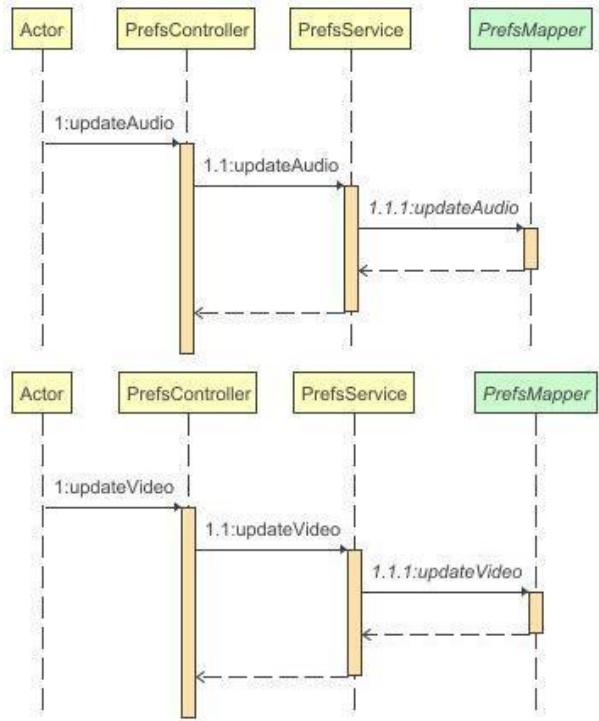


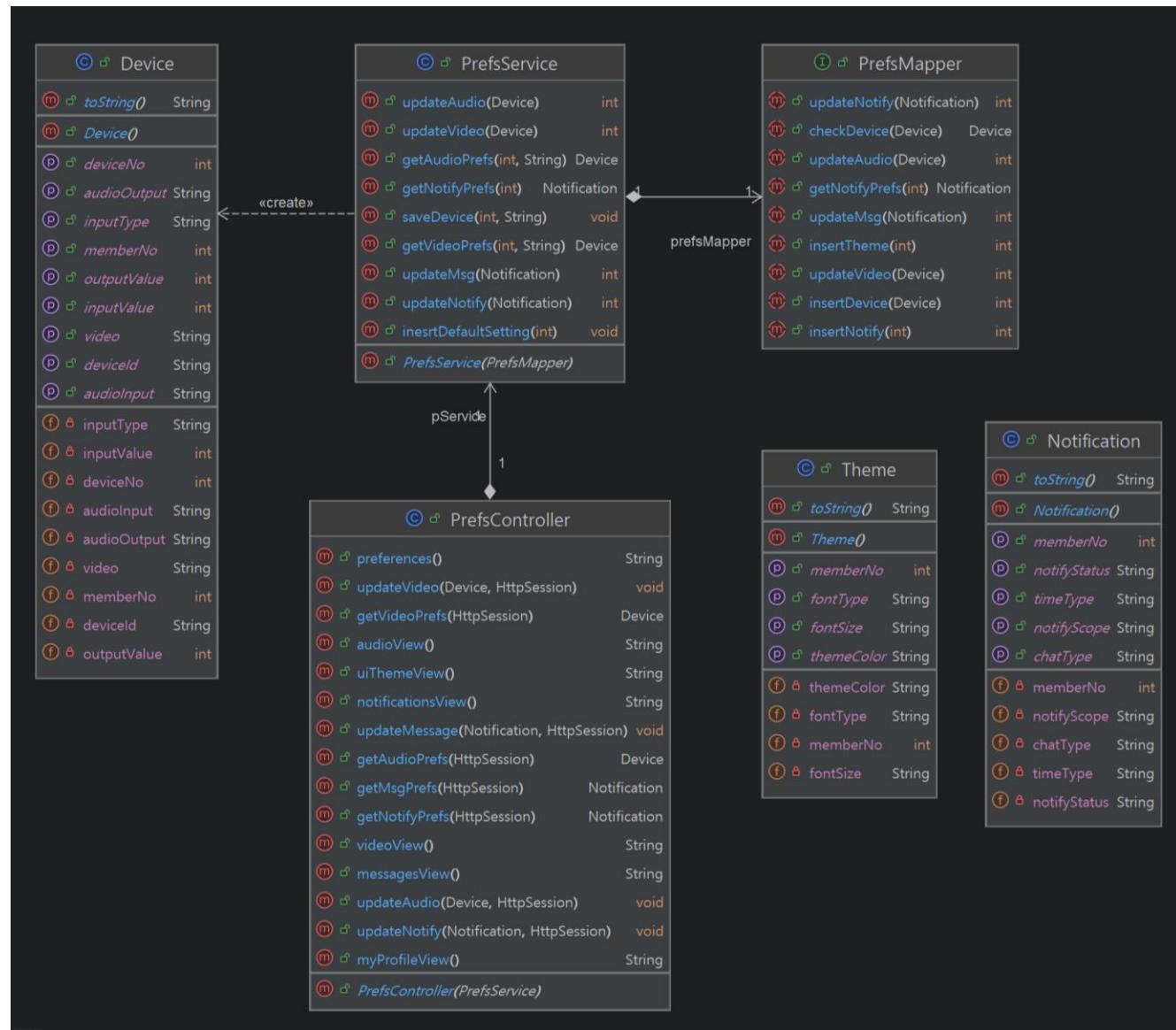


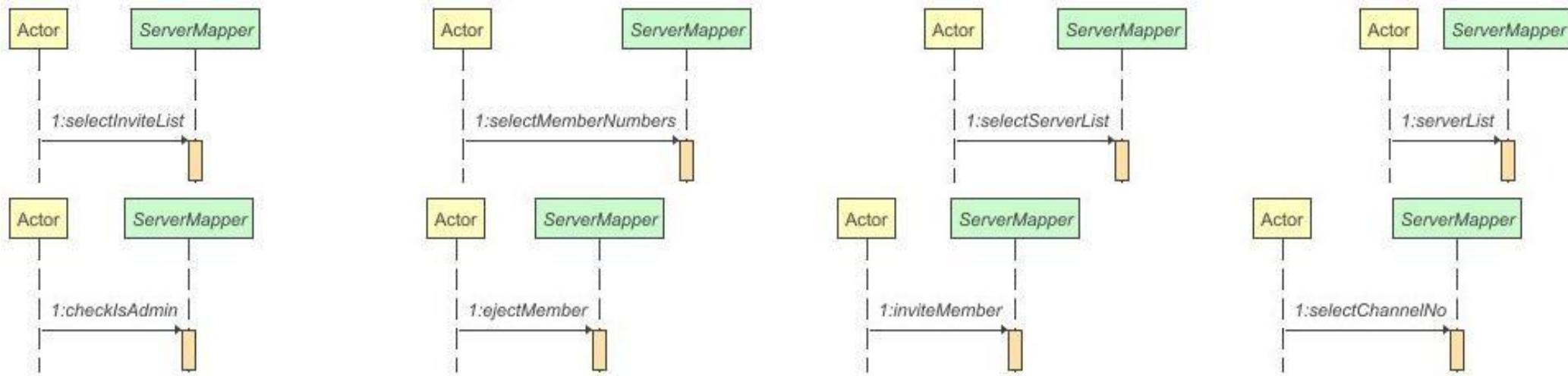


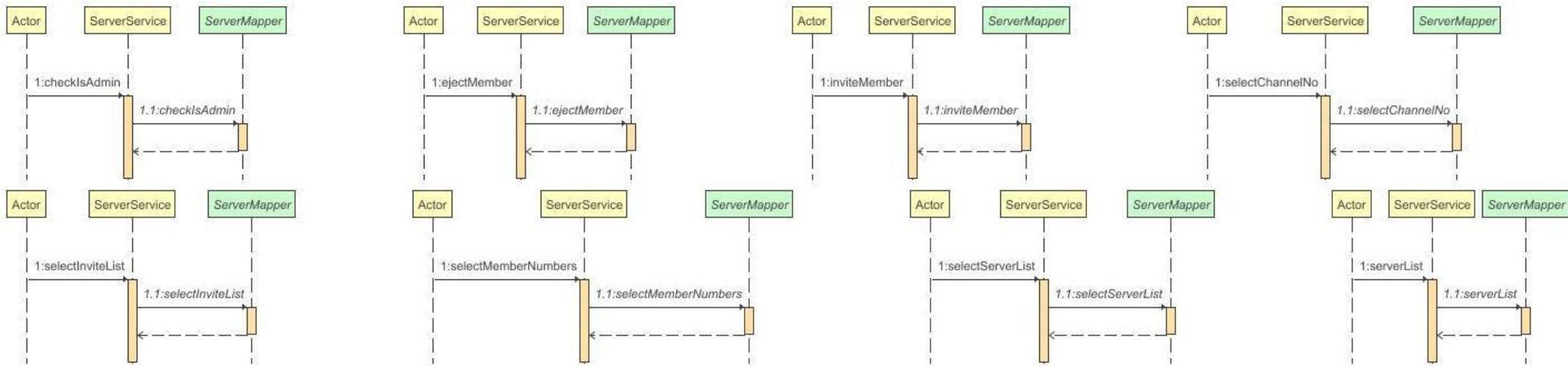


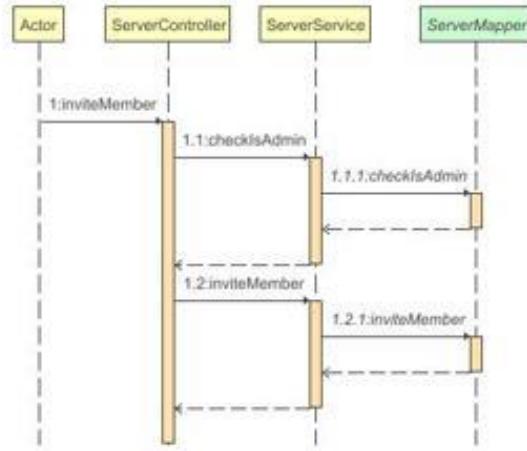
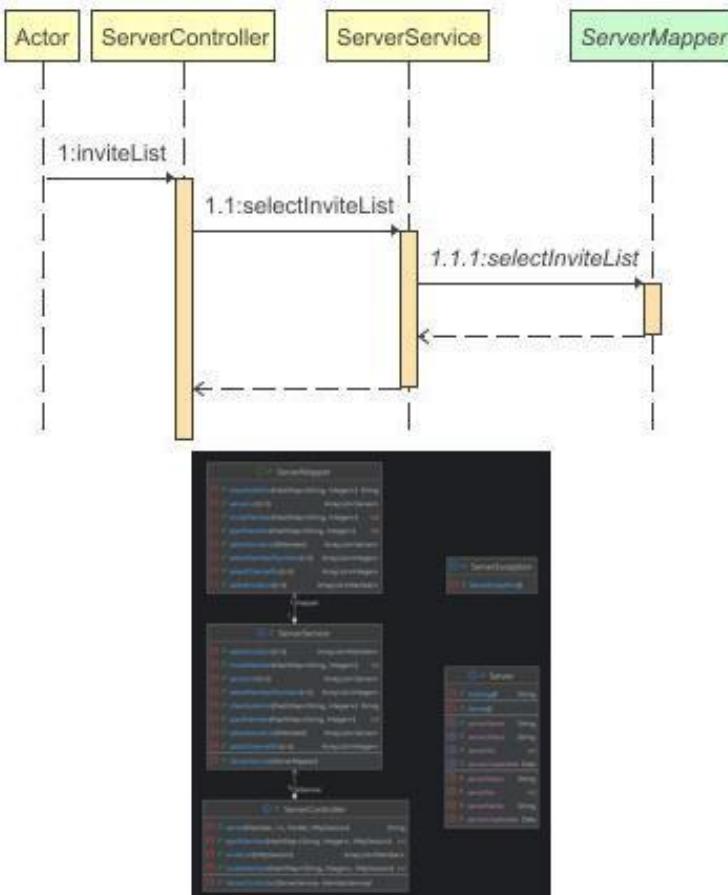
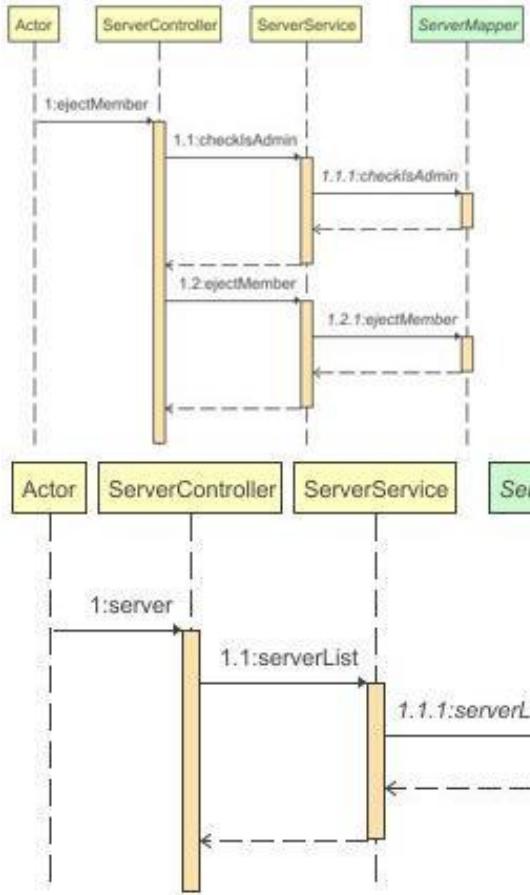


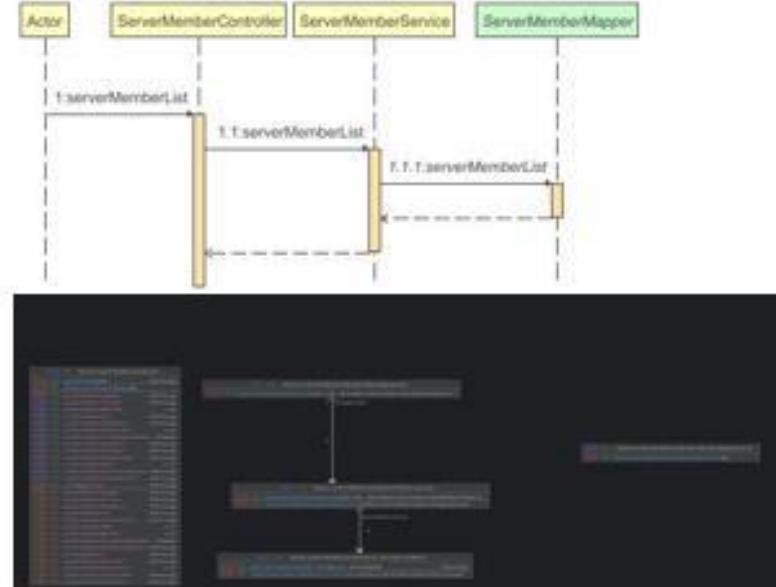
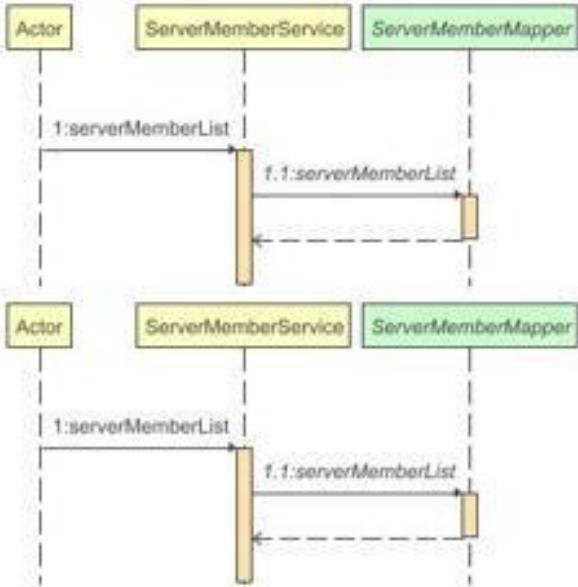
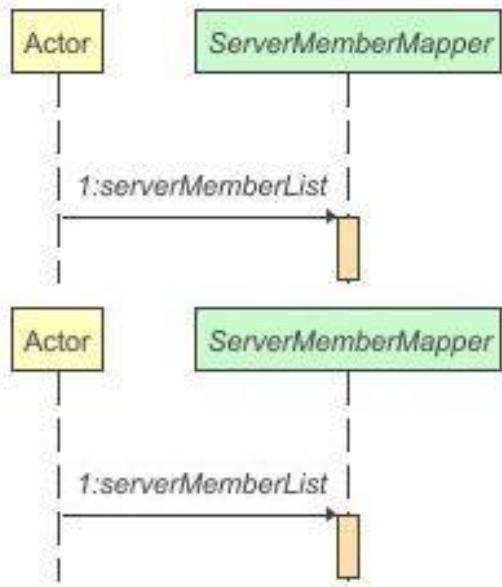


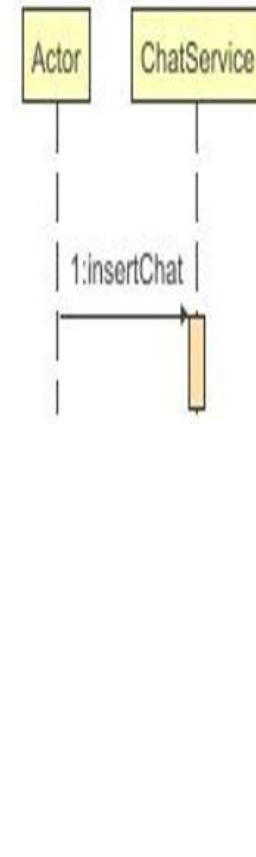
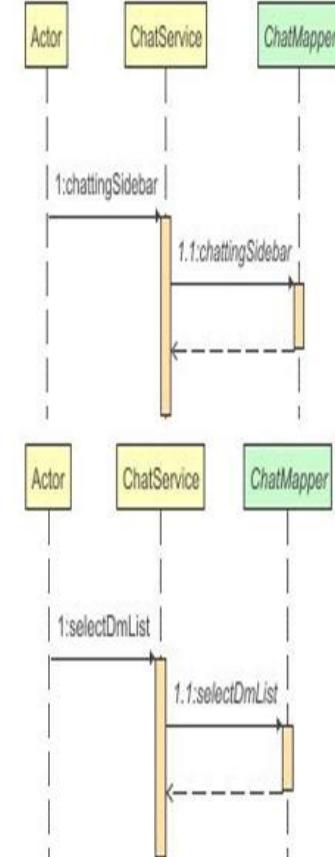
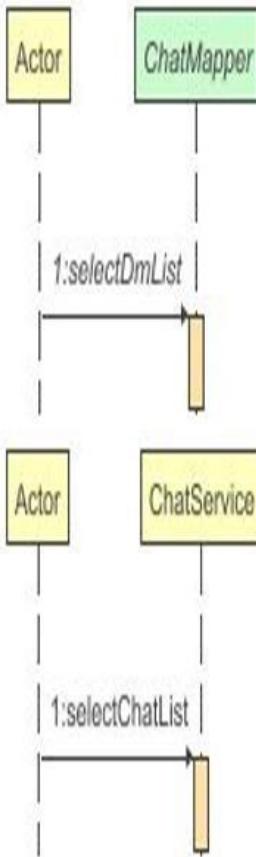
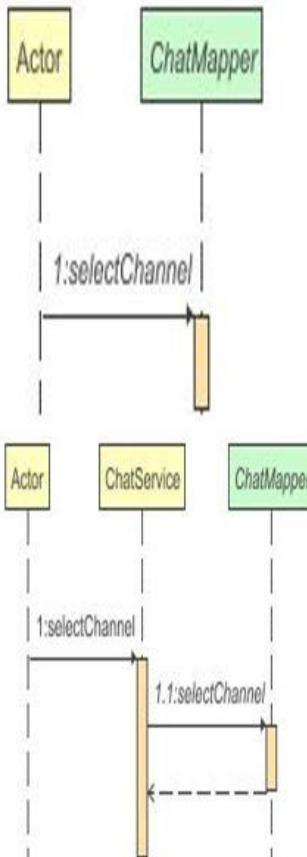
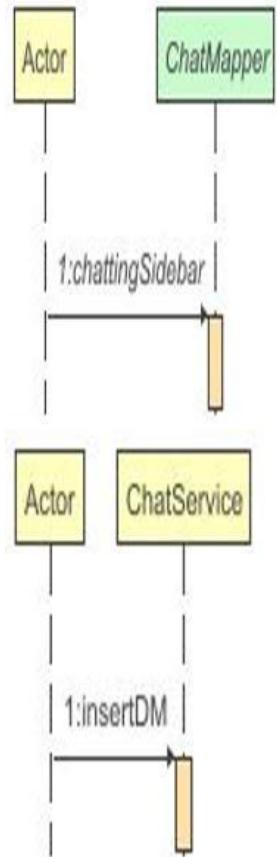


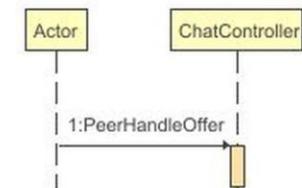
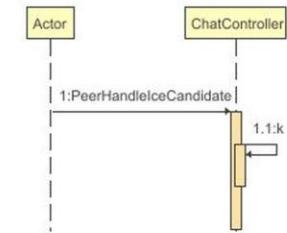
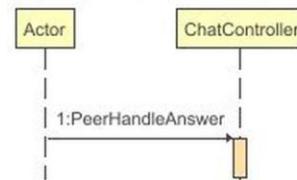
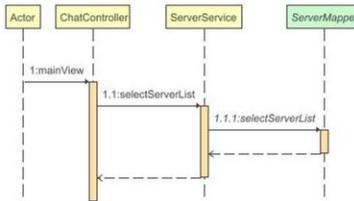
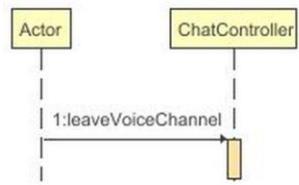
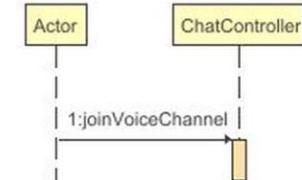
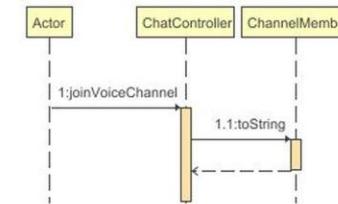
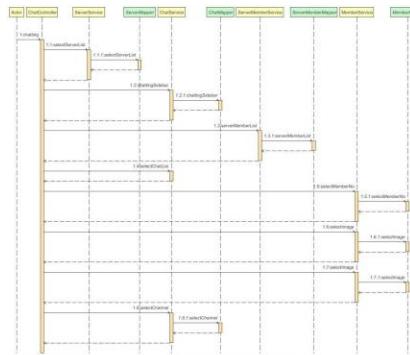
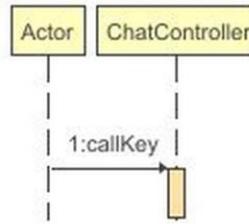
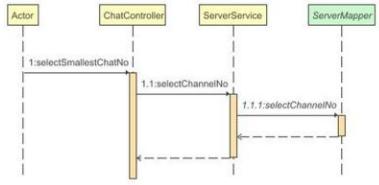


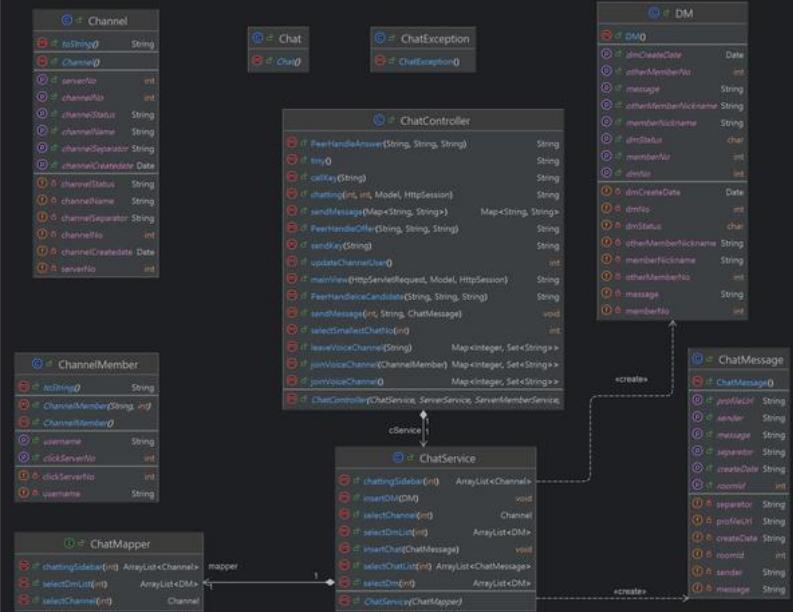
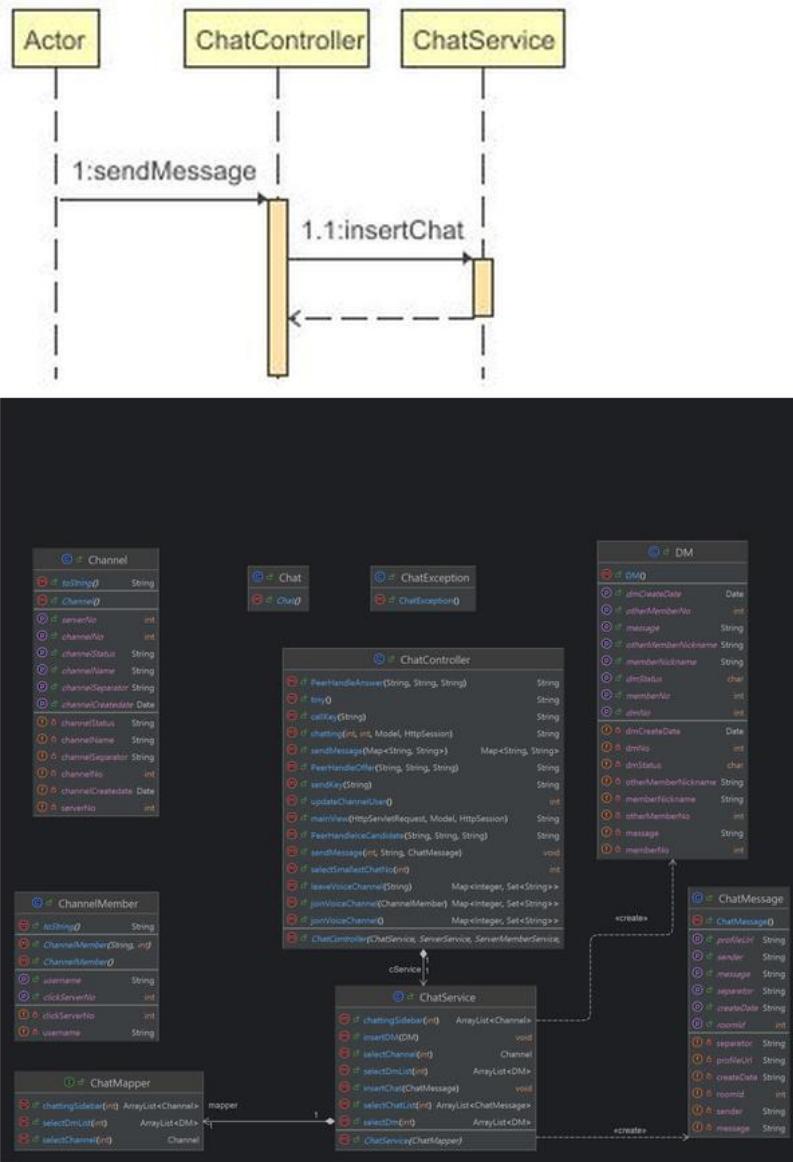
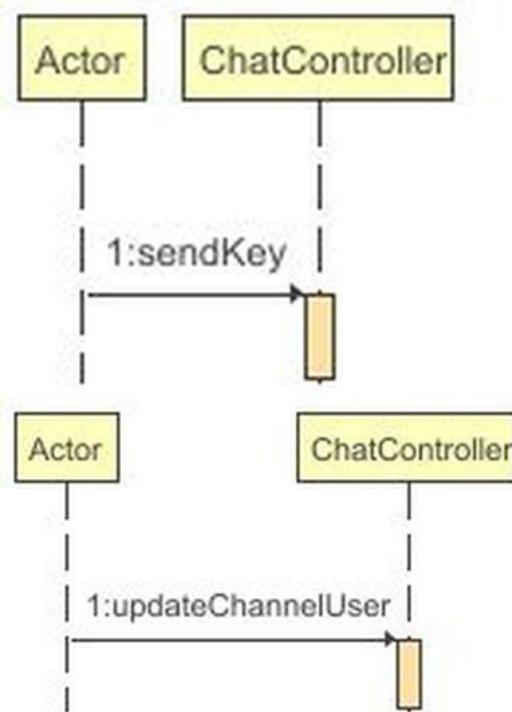












```
<form th:action="@{/member/signup}" method="post">
    <!-- 아이디 입력 -->
    <div class="input-box">
        <input type="text" id="memberId" name="memberId" placeholder=" " required>
        <label for="memberId">아이디</label>
        <p id="idCheck" style="font-size: 12px; margin-top: 5px;"></p>
    </div>

    <!-- 비밀번호 입력 -->
    <div class="input-box">
        <input type="password" id="memberPwd" name="memberPwd" placeholder=" " required>
        <label for="memberPwd">비밀번호</label>
    </div>

    <!-- 비밀번호 확인 -->
    <div class="input-box">
        <input type="password" id="confirmPassword" name="confirmPassword" placeholder=" " required>
        <label for="confirmPassword">비밀번호 확인</label>
        <p id="pwdCheck" style="font-size: 12px; margin-top: 5px;"></p>
    </div>

    <!-- 닉네임 입력 -->
    <div class="input-box">
        <input type="text" id="memberNickname" name="memberNickname" placeholder=" " required>
        <label for="memberNickname">닉네임</label>
    </div>

    <!-- 전화번호 입력 -->
    <div class="input-box">
        <input type="text" name="memberPhone" pattern="[0-9]{10,11}" placeholder=" " required />
        <label>전화번호</label>
    </div>
```

```
// 회원가입 진행
int result = mService.insertMember(m);
if (result > 0) {
    int getMemberNo = mService.getMemberNo(m.getMemberId());
    pService.inesrtDefaultSetting(getMemberNo);
    return "redirect:/member/signin";
} else {
    throw new MemberException("회원가입에 실패하였습니다.");
}

// 회원가입 처리 (성공 여부 반환)
public int insertMember(Member member) {
    if (member.getMemberBirth() == 0) {
        throw new IllegalArgumentException("생년월일은 필수 입력 값입니다.");
    }

    // 비밀번호 암호화 후 저장
    member.setMemberPwd(bcrypt.encode(member.getMemberPwd()));

    // 회원 정보 DB 저장 후 결과 반환
    return mapper.insertMember(member);
}
```

```
// 로그인 처리
@PostMapping("/signin")
public String login(@RequestParam("memberId") String memberId,
                    @RequestParam("memberPwd") String memberPwd,
                    @RequestParam("fingerprint") String fingerprint,
                    Model model, HttpSession session) {
    Member loginMember = mService.login(memberId, memberPwd);

    // 프로필 이미지도 세션에 저장하는게 좋을 듯
    ProfileImage userImage = mService.selectImage(loginMember.getMemberNo());
    if(userImage != null) {
        loginMember.setImageUrl(userImage.getImgRename().toString());
    }

    if (loginMember != null) {
        Theme theme = pService.getThemePrefs(loginMember.getMemberNo());
        Notification msg = pService.getNotifyPrefs(loginMember.getMemberNo());
        session.setAttribute("chatType", msg.getChatType());
        session.setAttribute("timeType", msg.getTimeType());
        session.setAttribute("theme", theme);
        session.setAttribute("loginMember", loginMember);
        session.setAttribute("fingerprint", fingerprint);
        pService.saveDevice(loginMember.getMemberNo(), fingerprint);
        return "redirect:/main";
    }
}

// 로그인 처리
public Member login(String memberId, String memberPwd) {
    Member member = mapper.login(memberId);
    if (member != null && bcrypt.matches(memberPwd, member.getMemberPwd())) {
        return member;
    }
    return null;
}
```

```
<div class="profile">
    
    <b id="profileName" th:text="${member.memberNickname}"></b>
    
    
    

    const imageSrc = profileImg.getAttribute("src") || "/images/default-avatar.png";
    const nickname = profileName.textContent || "사용자";
    const userId = profileName.getAttribute("data-user-id") || "unknown";

    openMiniProfile(imageSrc, nickname, userId);

    function openMiniProfile(imageSrc, nickname, userId) {
        document.getElementById("miniProfileImage").src = imageSrc;
        document.getElementById("miniProfileNickname").textContent = nickname;
        document.getElementById("miniProfileId").textContent = `#${userId}`;

        modal.style.display = "block";
        modal.style.opacity = "1";
        modal.style.visibility = "visible";
    }
}
```

# Jonathan

```
@MessageMapping("/chat/{channelNo}/{separetor}")
public void sendMessage(@DestinationVariable("channelNo") int channelNo, @DestinationVariable("separetor") String separetor, ChatMessage message) {
    // 특정 채팅방(roomId)에 메시지를 전송
    System.out.println("channelNo : " + channelNo);
    System.out.println("separetor : " + separetor);
    System.out.println("nickName : " + message.getSender());
    System.out.println("message : " + message.getMessage());
    message.setRoomId(channelNo);
    message.setSeparator(separetor);
    // firebaseStore
    cService.insertChat(message);
    messagingTemplate.convertAndSend(destination: "/sub/chatroom/" + channelNo, message);
}
```

```
// 메시지 전송하는 메서드
const sendMessage = (sender, message, roomId) => {
  if (!message || message.trim() === "" || message === "<p></p>") {
    alert("메시지를 입력하세요!");
    return;
  }

  // 객체화
  const sendData = { roomId, sender, message };

  console.log(`전송 데이터:`, sendData);

  stompClient.send(`/pub/chat/${roomId}/C`, {}, JSON.stringify(sendData));
};

enterText = () => {
  let msg = tinymce
    .get("mytextarea")
    .getContent({ format: "html" })
    .trim(); // HTML 가져오기

  let imgTagMatch = msg.match(/<img[^>]+src="([^>]+)"\>/);
  if (imgTagMatch) {
    let imgSrc = imgTagMatch[1];
    let imgTag = ``;
    let newContent = msg.replace(/<img[^>]+>/, imgTag);
    sendMessage(username, newContent, channelNo);
  } else {
    sendMessage(username, msg, channelNo);
  }

  tinymce.get("mytextarea").setContent("");
};
```

```
@Configuration
@RequiredArgsConstructor
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
    // private final ChatHandler chatHandler;
    // private final StompHandler stompHandler;

    @Override 0개의 사용위치
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.enableSimpleBroker(...destinationPrefixes: "/sub");
        registry.setApplicationDestinationPrefixes("/pub");
    }

    @Override 0개의 사용위치
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint(...paths: "/stomp/chat").setAllowedOriginPatterns("*").withSockJS();
        registry.addEndpoint(...paths: "/stomp/channel").setAllowedOriginPatterns("*").withSockJS();
        registry.addEndpoint(...paths: "/stomp/signaling").setAllowedOriginPatterns("*").withSockJS();
    }

    @Override 0개의 사용위치
    public void configureWebSocketTransport(WebSocketTransportRegistration registry) {
        registry.setMessageSizeLimit(160*64*1024);
    }
}
```

```
const socket = new SockJS("https://192.168.40.6:9090/stomp/chat");
//const socket = new SockJS("https://192.168.140.22:9090/stomp/chat");
const stompClient = Stomp.over(socket);

stompClient.connect({}, () => {
    console.log("WebSocket 연결 성공!");

    // 특정 채팅방(roomId) 구독
    stompClient.subscribe("/sub/chatroom/" + channelNo, (message) => {
        // chatMessage는 JSON으로 parse해서 가져옴
        const chatMessage = JSON.parse(message.body);

        // 현재 시간 받아오기
        let today = new Date();

        let hour = String(today.getHours());
        let minute = String(today.getMinutes());
        let second = String(today.getSeconds());

        if (hour.length == 1) {
            hour = "0" + hour;
        }

        if (minute.length == 1) {
            minute = "0" + minute;
        }

        if (second.length == 1) {
            second = "0" + second;
        }

        let HMS = hour + ":" + minute + ":" + second;
        ...
    });
});
```

```
// 받아온 메시지 채팅방에 출력하기
let str = `<div class='chatContent-1'>
    <img src=${userProfileImage != null ? userProfileImage : '/image/member/no-profile.svg'} width="40px" height="40px">
    <div class='chatContent-2'>
        <b>${chatMessage.sender}</b>
        <div style='display:flex; justify-content:space-between; align-items:flex-end;'>
            ${chatMessage.message}
            <p>${HMS}</p>
        </div>
    </div>
</div>`;
document.querySelector(".chatContent").insertAdjacentHTML("afterbegin", str);

// 스크롤 이동
document.querySelector(".chatContent").scrollTop = document.querySelector(".chatContent").scrollHeight;
});
```

```
//camKey 를 받기위해 신호를 보내는 webSocket
@MessageMapping("/call/key")
@SendTo("/sub/call/key")
public String callKey(@Payload String message) {
    System.out.println(message);
//    log.info("[Key] : {}", message);
    return message;
}
```

```
const connectSocket = async () =>{
    const socket = new SockJS('https://192.168.40.6:9090/stomp/signaling');
    //const socket = new SockJS('https://192.168.140.22:9090/stomp/signaling');
    stompClient = Stomp.over(socket);
    stompClient.debug = null;

    stompClient.connect({}, () => {
        console.log('Connected to WebRTC server');

        //iceCandidate peer 교환을 위한 subscribe
        stompClient.subscribe('/sub/peer/iceCandidate/' + myKey + '/' + roomId, candidate => {
            console.log("/sub/peer/iceCandidate/에 들어옴");
            const key = JSON.parse(candidate.body).key
            const message = JSON.parse(candidate.body).body;

            // 해당 key에 해당되는 peer 에 받은 정보를 addIceCandidate 해준다.
            pcListMap.get(key).addIceCandidate(new RTCIceCandidate({candidate:message.candidate,sdpMLineIndex:message.sdpMLineIndex,sdpMid:message.sdpMid}));

        });

        //offer peer 교환을 위한 subscribe
        stompClient.subscribe('/sub/peer/offer/' + myKey + '/' + roomId, offer => {
            console.log("sub/peer/offer/에 들어옴");
            console.log("Received Offer:", offer.body);
            const key = JSON.parse(offer.body).key;
            const message = JSON.parse(offer.body).body;

            // 해당 key에 새로운 peerConnection 을 생성해준후 pcListMap 에 저장해준다.
            pcListMap.set(key,createPeerConnection(key));
            // 생성한 peer 에 offer정보를 setRemoteDescription 해준다.
            console.log('Setting Remote Description for key: ' + key);
            pcListMap.get(key).setRemoteDescription(new RTCSessionDescription({type:message.type,sdp:message.sdp}));
            //sendAnswer 할수를 호출해준다.
            sendAnswer(pcListMap.get(key), key);

        });

    });
}
```

```
//answer peer 교환을 위한 subscribe
stompClient.subscribe('/sub/peer/answer/' + myKey + '/' + roomId, answer =>{
    const key = JSON.parse(answer.body).key;
    const message = JSON.parse(answer.body).body;

    console.log("sub/peer/answer/에 들어옴 : " + message);

    // 해당 key에 해당되는 Peer에 받은 정보를 setRemoteDescription 해준다.
    pcListMap.get(key).setRemoteDescription(new RTCSessionDescription(message));

});

//key를 보내라는 신호를 받은 subscribe
stompClient.subscribe('/sub/call/key', message =>{
    console.log("/sub/call/key에 들어옴 : " + message);
    //자신의 key를 보내는 send
    stompClient.send('/pub/send/key', {}, JSON.stringify(myKey));

});

//상대방의 key를 받는 subscribe
stompClient.subscribe('/sub/send/key', message => {
    console.log("/sub/send/key에 들어옴");
    const key = JSON.parse(message.body);

    //만약 중복되는 키가 otherKeyList에 있는지 확인하고 없다면 추가해준다.
    if(myKey !== key && otherKeyList.find((mapKey) => mapKey === key) === undefined){
        otherKeyList.push(key);
    }

    console.log(otherKeyList);
});

});
```

```
const createPeerConnection = (otherKey) =>{
    const pc = new RTCPeerConnection();
    try {
        pc.addEventListener('icecandidate', (event) =>{
            onIceCandidate(event, otherKey);
        });
        pc.addEventListener('track', (event) =>{
            onTrack(event, otherKey);
        });
        if(localStream !== undefined){
            localStream.getTracks().forEach(track => {
                pc.addTrack(track, localStream);
            });
        }
        console.log("otherKey는 아래와 같습니다.");
        console.log(otherKey);
        const remoteVideo = document.getElementById(otherKey); // 상대방 비디오 요소
        console.log("remoteVideo의 값은 아래와 같습니다.");
        console.log(remoteVideo);

        pc.addEventListener("connectionstatechange", () => {
            console.log("연결 상태:", pc.connectionState);
        });
    } catch (error) {
        console.error(error);
    }
}
```

```
        if (pc.connectionState === "disconnected" || pc.connectionState === "failed" || pc.connectionState === "closed") {
            console.log("상대방 연결이 끊어졌습니다. 비디오 제거!");
            remoteVideo.srcObject = null; // 비디오 스트림 제거
            remoteVideo.style.display = "none"; // 화면에서 숨기기 (선택)
        }
    });

pc.addEventListener("iceconnectionstatechange", () => {
    console.log("ICE 상태:", pc.iceConnectionState);

    if (pc.iceConnectionState === "disconnected" || pc.iceConnectionState === "failed") {
        console.log("상대방이 네트워크 문제로 연결이 끊어졌습니다!");
        console.log(remoteVideo);
        remoteVideo.srcObject = null; // 비디오 스트림 제거
        remoteVideo.style.display = "none"; // 화면에서 숨기기 (선택)
    }
});

console.log('PeerConnection created');
} catch (error) {
    console.error('PeerConnection failed: ' + error);
}
return pc;
}
```

```
// video 생성 및 삭제
let remoteVideo = null;
let onTrack = (event, otherKey) => {
    console.log('otherKey : ' + otherKey);
    console.log('event : ' + event);
    if(document.getElementById(otherKey) === null){
        console.log('otherkey가 null이 아니야!!!!');
        const video = document.createElement('video');

        video.autoplay = true;
        video.controls = true;
        video.muted = true;
        video.id = otherKey;
        video.srcObject = event.streams[0];
        const element = document.querySelector('#' + otherKey);
        // 같은 cameraKey값이 있는 video가 있으면 해당 video 삭제
        console.log('element 확인');
        console.log(element);
        if(element){
            console.log('존재하는 element');
            console.log(element);
            element.remove();
        }

        document.getElementById('remoteStreamDiv').appendChild(video);
        remoteVideo = document.getElementById(otherKey);
        console.log(remoteVideo);
    }
}

// 
// remoteStreamElement.srcObject = event.streams[0];
// remoteStreamElement.play();
};
```

```
@MessageMapping("/peer/answer/{camKey}/{roomId}")
@SendTo("/sub/peer/answer/{camKey}/{roomId}")

public String PeerHandleAnswer(@Payload String answer, @DestinationVariable(value = "roomId") String roomId,
                               @DestinationVariable(value = "camKey") String camKey) {
    System.out.println("answerroomId : " + roomId);
    System.out.println("answercamKey : " + camKey);
    System.out.println("answer : " + answer);
    log.info("[ANSWER] {} : {}", camKey, answer);
    return answer;
}
```

```
//iceCandidate 정보를 주고 받기 위한 webSocket
//camKey : 각 요청하는 캠의 key , roomId : 룸 아이디
@MessageMapping("/peer/iceCandidate/{camKey}/{roomId}")
@SendTo("/sub/peer/iceCandidate/{camKey}/{roomId}")

public String PeerHandleIceCandidate(@Payload String candidate, @DestinationVariable(value = "roomId") String roomId,
                                     @DestinationVariable(value = "camKey") String camKey) {
    System.out.println("iceCandidateroomId : " + roomId);
    System.out.println("iceCandidatecamKey : " + camKey);
    for(int key : videoInChannel.keySet()) {
        if(videoInChannel.get(key).contains(camKey)) {
            videoInChannel.get(key).remove(camKey);
        }
    }
}

videoInChannel.computeIfAbsent(memberInchannelNo, Integer k -> ConcurrentHashMap.newKeySet()).add(camKey);

System.out.println("[ICECANDIDATE] {} : {}", camKey, candidate);
return candidate;
}
```

```
//offer 정보를 주고 받기 위한 websocket
//camKey : 각 요청하는 캠의 key , roomId : 룸 아이디
@MessageMapping("/peer/offer/{camKey}/{roomId}")
@SendTo("/sub/peer/offer/{camKey}/{roomId}")

public String PeerHandleOffer (@Payload String offer, @DestinationVariable(value = "roomId") String roomId,
                               @DestinationVariable(value = "camKey") String camKey){
    System.out.println("roomId : " + roomId);
    System.out.println("offer : " + offer);
    System.out.println("camKey : " + camKey);
    // System.out.printf("[OFFER] {} : {}", camKey, offer);

    return offer;
}
```

```
//자신의 camKey 를 모든 연결된 세션에 보내는 webSocket
@MessageMapping("/send/key")
@SendTo("/sub/send/key")
public String sendKey (@Payload String message){
    System.out.println("sendmessage : " + message);
    return message;
}
```

```
let localStreamElement = document.querySelector('#localStream');
//자신을 식별하기위한 랜덤한 key
const myKey = Math.random().toString(36).substring(2, 11);
const myKey = userid;
let pcListMap = new Map();
let roomId = channelNo;
let otherKeyList = [];
let localStream = undefined;
console.log("듬чат 드림 : " + navigator.mediaDevices);
onload = () => {
    startCam();

}

const startCam = async () =>{
    await connectSocket();

    if(navigator.mediaDevices !== undefined){
        await navigator.mediaDevices.getUserMedia({ audio: true, video : true })
        .then(async (stream) => {
            console.log('Stream found');
            //웹캠, 마이크의 스트림 정보를 글로벌 변수로 저장한다.
            localStream = stream;
            // Disable the microphone by default
            stream.getAudioTracks()[0].enabled = true;
            localStreamElement.srcObject = localStream;
            localStreamElement.style.display = "block"; // 비디오 화면 표시

            // Connect after making sure that local stream is available

        }).catch(error => {
            console.error("Error accessing media devices:", error);
        });
    }
}
```

```
await stompClient.send('/pub/call/key', {}, {});
const connect = document.querySelectorAll('.connect')[0];
connect.innerHTML = `<b>음성 연결됨</b>
`;

// 음성연결 끊기 버튼 클릭 시 가장 상단 채팅채널로 이동
document.querySelectorAll('.connect')[0].querySelector('img').addEventListener('click', ()=>{
    console.log('나가기 버튼 클릭!');
    console.log(document.querySelectorAll('.profile')[0].querySelector('b').innerText);
    const myNickname = document.querySelectorAll('.profile')[0].querySelector('b').innerText;
    leaveVoiceChannel(myNickname);
    $.ajax({
        url: "/chat/selectSmallestChatNo",
        method: 'post',
        data: {serverNo:serverNo},
        success: data=>{
            console.log(data);
            if(data != 0){
                location.href='/chat/main/' + serverNo + "/" + data;
            }
        }
    })
})
console.log('들어왔당.');
setTimeout(() =>{

    otherKeyList.map((key) =>{
        console.log('들어왔당.');
        console.log(pcListMap);
        if(!pcListMap.has(key)){
            console.log('pcListMap.has(key)가 true다!!!!')
            pcListMap.set(key, createPeerConnection(key));
            sendOffer(pcListMap.get(key),key);
        }
    });

});,1000);
```

```
// ICE(Interactive Connectivity Establish)를 이용한 connection 연결
let onIceCandidate = (event, otherKey) => {
    if (event.candidate) {
        console.log('ICE candidate');
        stompClient.send('/pub/peer/iceCandidate/' + otherKey + '/' + roomId, {}, JSON.stringify({
            key : myKey,
            body : event.candidate
        }));
    }
};

// 상대방에서 풀신 걸기
let sendOffer = (pc ,otherKey) => {
    console.log('pc : ' + pc);
    console.log('otherKey : ' + otherKey);
    pc.createOffer().then(offer =>{
        setLocalAndSendMessage(pc, offer);
        stompClient.send('/pub/peer/offer/' + otherKey + '/' + roomId, {}, JSON.stringify({
            key : myKey,
            body : offer
        }));
        console.log('Send offer');
    });
};
```

```
// 등신에 대한 응답
let sendAnswer = (pc,otherKey) => {
    pc.createAnswer().then( answer => {
        setLocalAndSendMessage(pc ,answer);
        stompClient.send('/pub/peer/answer/' + otherKey + '/' + roomId, {}, JSON.stringify({
            key : myKey,
            body : answer
        }));
        console.log('Send answer');
    });
};

const setLocalAndSendMessage = (pc ,sessionDescription) =>{
    pc.setLocalDescription(sessionDescription);
}

// 등신 종료
window.addEventListener('beforeunload', ()=>{
    // webRTC 연결 종료
    if(peerConnection){
        console.log('연결이 끊깁니다.');
        peerConnection.close();
    }
})
```

```
tinymce.init({
    selector: "#mytextarea",
    plugins: "image paste",
    menubar: false,
    toolbar: false,
    statusbar: false,
    height: "70px",
    width: "100%",
    resize: false,
    images_upload_handler: function (blobInfo, success, failure) {
        let reader = new FileReader();
        reader.onload = function () {
            let base64URL = reader.result;

            // ✅ 이미지 크기 조절 태그 추가
            let imgTag = ` showMentionBox(editor), 10);
    } else if (mentionBox) {
        if (event.key === "ArrowDown" || event.key === "ArrowUp") {
            event.preventDefault();
            moveSelection(event.key === "ArrowDown" ? 1 : -1);
        } else if (event.key === "Enter" && mentionText) {
            event.preventDefault();
            selectItem();
        } else if (event.key === "Escape") {
            removeMentionBox();
        }
    } else{
        if(event.key == 'Enter' && !event.shiftKey){
            event.preventDefault();
            document.querySelector('#button-send').click();

// `input` 미벤트에서 mentionText 업데이트
editor.on("input", function () {
    let content = editor.getContent({ format: "text" }); // 현재 텍스트 가져오기
    let words = content.split(/\s+/); // 공백 기준으로 분리
    let lastWord = words[words.length - 1]; // 마지막 단어 가져오기

    if (lastWord.startsWith("@")) {
        mentionText = lastWord.slice(1); // @ 제외한 텍스트 저장
        console.log("현재 입력한 mentionText:", mentionText);
        updateMentionList();
    } else {
        mentionText = ""; // @로 시작하지 않으면 초기화
        removeMentionBox();
    }
});
```

```
function updateMentionList() { 사용 위치 표시 ✎ desertdevv +1 *
  mentionBox.innerHTML = "";

  if (!mentionText) {
    mentionBox.innerHTML = "<div>검색어를 입력하세요.</div>";
    return;
  }

  let filteredMembers = memberList.filter((member) =>
    member.memberNickname
      .toLowerCase()
      .includes(mentionText.toLowerCase())
  );

  if (filteredMembers.length === 0) {
    mentionBox.innerHTML = "<div>검색 결과가 없습니다.</div>";
    return;
  }

  filteredMembers.forEach((member) => {
    let item = document.createElement("div");
    item.classList.add("mention-item");
    item.dataset.nickname = member.memberNickname;
    item.innerHTML = member.memberNickname.replace(
      new RegExp(mentionText, "gi"),
      (match) =>
        `<span style="color: blue; font-weight: bold;">${match}</span>`
    );
    item.style.padding = "5px";
    item.style.cursor = "pointer";

    item.addEventListener("click", function () {
      insertMention(editor, member.memberNickname);
    });
    mentionBox.appendChild(item);
  });
}
```

```
function insertMention(editor, name) { 사용 위치 표시 🔍 desertdevv +1
  const range = editor.selection.getRange();
  const startNode = range.startContainer;
  const startOffset = range.startOffset;

  // 기존 '@ + 입력한 텍스트' 삭제
  const textContent = startNode.textContent;
  const atIndex = textContent.lastIndexOf("@"); // 마지막 '@' 위치 찾기
  if (atIndex !== -1) {
    startNode.deleteData(atIndex, mentionText.length + 1); // @ + 입력한 텍스트 길이만큼 삭제
  }

  // 새로운 멘션 삽입
  const newText = `@${name}`;
  startNode.insertData(atIndex, newText);

  // 커서 위치 조정 (멘션 삽입 후 올바른 위치로 이동)
  editor.selection.setCursorLocation(
    startNode,
    atIndex + newText.length
  );

  removeMentionBox();
}

function removeMentionBox() { 사용 위치 표시 🔍 desertdevv +1
  if (mentionBox) {
    mentionBox.remove();
    mentionBox = null;
    mentionText = "";
  }
}
```

```
public void insertChat(ChatMessage message) { 1개 사용 위치 ✅ success
    // 현재 날짜/시간
    LocalDateTime now = LocalDateTime.now();
    // 현재 날짜/시간 출력
    System.out.println(now);
    // 2021-06-17T06:43:21.419878100
    // 포맷팅
    String formatedNow = now.format(DateTimeFormatter.ofPattern("yyyy년 MM월 dd일 HH시 mm분 ss초"));
    // 포맷팅 현재 날짜/시간 출력
    System.out.println(formatedNow); // 2021년 06월 17일 06시 43분 21초

    Firestore db = FirestoreClient.getFirestore();

    DocumentReference docRef = db.collection("RealMan01").document();

    // Add document data with id "alovelace" using a hashmap
    Map<String, Object> data = new HashMap<>();
    data.put("chat_content", message.getMessage());
    data.put("chat_memberNickname", message.getSender());
    data.put("chat_createdate", Timestamp.now());
    data.put("dc_no", message.getRoomId());
    data.put("chat_separator", message.getSeparator());
    //asynchronously write data
    ApiFuture<WriteResult> result = docRef.set(data);
    // ...
    // result.get() blocks on response
    try {
        System.out.println("Update time : " + result.get().getUpdateTime());
    } catch (InterruptedException | ExecutionException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

```
public ArrayList<ChatMessage> selectChatList(int channelNo) { 1개 사용 위치 ✅ sucresucces *
```

```
    Firestore db = FirestoreClient.getFirestore();

    // asynchronously retrieve all users
    ApiFuture<QuerySnapshot> query = db.collection("RealMan01").whereEqualTo("dc_no", channelNo)
        .orderBy("chat_createdate", Query.Direction.DESCENDING).get();
    // ...
    QuerySnapshot querySnapshot;
    ArrayList<ChatMessage> chatList = new ArrayList<>();
    try {
        querySnapshot = query.get();
        List<QueryDocumentSnapshot> documents = querySnapshot.getDocuments();
        for (QueryDocumentSnapshot document : documents) {

            ChatMessage message = new ChatMessage();
            message.setMessage(document.getString("chat_content"));
            message.setRoomId(document.getLong("dc_no").intValue());
            message.setSender(document.getString("chat_memberNickname"));
            message.setCreateDate(document.getTimestamp("chat_createdate"));
            message.setSeparator(document.getString("chat_separator"));
            chatList.add(message);
        }
    } catch (InterruptedException | ExecutionException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return chatList;
}
```

```
// 프사 변경
@PutMapping("/profileImg")
@ResponseBody
public boolean changeProfileImg(@RequestParam("image") MultipartFile image, HttpSession session, Model model) {
    System.out.println("profileImg 들어옴.");
    Member loginMember = (Member)session.getAttribute("loginMember");

    if(image != null && !image.isEmpty()) {
        String fileName = image.getOriginalFilename();
        |
        String[] files = saveFiles(image);
        if (files[1] != null) {
            ProfileImage profileImage = new ProfileImage();
            profileImage.setImgName(fileName);
            profileImage.setImgPath(files[1]);
            profileImage.setImgRename(files[0]);
        }
    }
}
```

```
public String[] saveFiles(MultipartFile upload) {
    SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddHHmmssSSS");
    int ranNum = (int)(Math.random()*100000);
    String originFileName = upload.getOriginalFilename();
    String renameFileName = sdf.format(new Date()) + ranNum + originFileName.substring(originFileName.lastIndexOf("."));

    ObjectMetadata metadata = new ObjectMetadata();
    metadata.setContentLength(upload.getSize());
    metadata.setContentType(upload.getContentType());

    try {
        amazonS3.putObject(bucket, renameFileName, upload.getInputStream(), metadata);
    } catch (SdkClientException | IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    String[] returnArr = new String[2];
    returnArr[0] = amazonS3.getUrl(bucket, renameFileName).toString();
    returnArr[1] = renameFileName;

    return returnArr;
}
```

```
//타이머  
const timer = () => {  
    sendBtn.disabled = true;  
    let sec = 60;  
    const interval = setInterval(() => {  
        if (sec > 0) {  
            sendBtn.innerText = sec;  
            sendBtn.classList.add("send-btn-disabled");  
            sendBtn.disabled = true;  
            sec--;  
        } else {  
            clearInterval(interval);  
            sendBtn.innerText = "send";  
            sendBtn.classList.remove("send-btn-disabled");  
            sendBtn.disabled = false;  
        }  
    }, 1000);  
};
```

```
//이메일 전송 버튼
sendBtn.addEventListener("click", async function () {
    console.log("이메일 전송 버튼 클릭");

    if (validateEmail(email.value)) {
        console.log(email.value + "로 이메일 전송 시작");
        document.querySelector(".modal-container").style.display = "flex";

        fetch("/member/sendEmail?email=" + email.value)
            .then((response) => response.text())
            .then((data) => {
                document.querySelector(".modal-container").style.display = "none";
                switch (data) {
                    case "EmailNotFound":
                        alert("해당 이메일로 가입된 회원이 존재하지 않습니다.");
                        email.focus();
                        break;
                    case "MailException":
                        alert(
                            "이메일 전송 과정중 오류가 발생했습니다 잠시 후에 다시 시도해주세요."
                        );
                        break;
                    case "MessagingException":
                        alert("이메일 형식 오류(사실상 일어날 일 없음)");
                        break;
                    default:
                        document.querySelector(".modal-container").style.display = "none";
                        verificationCode = data;
                        console.log("verificationCode : " + verificationCode);
                        timer();
                        break;
                }
            });
    }
});
```

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT
1 DEVICE_NO	NUMBER	No	"REALMAN"."ISEQ\$\$_199094".nextval
2 MEMBER_NO	NUMBER	No	(null)
3 DEVICE_ID	VARCHAR2(255 BYTE)	No	(null)
4 AUDIO_INPUT	VARCHAR2(100 BYTE)	Yes	(null)
5 AUDIO_OUTPUT	VARCHAR2(100 BYTE)	Yes	(null)
6 VIDEO	VARCHAR2(100 BYTE)	Yes	(null)
7 INPUT_VALUE	NUMBER(4,1)	No	70
8 OUTPUT_VALUE	NUMBER(4,1)	No	70
9 INPUT_TYPE	VARCHAR2(20 BYTE)	No	'A'

```
// DOM이 로드된 후 fingerprint를 입력 필드에 설정
document.addEventListener("DOMContentLoaded", async function () {
    const fingerprint = await getClientFingerprint();
    console.log("Final Fingerprint:", fingerprint);

    // 로그인 폼 내부의 hidden인 fingerprint INPUT 값을 수정
    const fingerprintInput = document.getElementById("fingerprint");
    if (fingerprintInput) {
        fingerprintInput.value = fingerprint;
    }
});
```

```
// 클라이언트의 공용 IP 주소를 가져오는 함수
async function getClientIPAddress() { 사용 위치 표시

    return new Promise((resolve, reject) => {
        const connection = new RTCPeerConnection({
            iceServers: [{ urls: "stun:stun.l.google.com:19302" }],
        });

        // WebRTC를 통해 ICE 후보 생성
        connection.createDataChannel(""); // 연결을 위한 채널 생성
        connection.onicecandidate = (event) => {
            if (event.candidate) {
                // srflx 타입의 후보에서만 IP 추출 (공용 IP)
                if (event.candidate.candidate.includes("srflx")) {
                    const ipMatch = event.candidate.candidate.match(/\d+\.\d+\.\d+\.\d+/);
                    console.log(ipMatch); // 추출된 IP 확인용 로그
                    if (ipMatch) {
                        resolve(ipMatch[0]);
                    }
                }
            }
        };
        // Offer 생성 및 로컬 설명 설정
        connection
            .createOffer()
            .then((offer) => connection.setLocalDescription(offer))
            .catch(reject);
    });
}
```

```
// 사용자의 GPU 정보를 가져오는 함수  
  
function getGPUInfo() { 사용 위치 표시  
    const canvas = document.createElement("canvas");  
    const gl =  
        canvas.getContext("webgl") || canvas.getContext("experimental-webgl");  
  
    if (!gl) return "Unknown GPU";  
  
    // WebGL 디버그 확장을 통해 렌더러 정보 추출  
    const debugInfo = gl.getExtension("WEBGL_debug_renderer_info");  
    return debugInfo  
        ? gl.getParameter(debugInfo.UNMASKED_RENDERER_WEBGL)  
        : "Unknown GPU";  
}
```

```
// 클라이언트의 디바이스 정보를 기반으로 고유 식별자(Fingerprint) 생성
async function getClientFingerprint() { 사용 위치 표시
    try {
        const ip = await getClientIPAddress(); // 공용 IP 주소
        const gpu = getGPUInfo(); // GPU 모델명
        const cpuCores = navigator.hardwareConcurrency || "unknown"; // CPU 코어 수
        const memory = navigator.deviceMemory || "unknown"; // RAM 용량

        // 정보를 조합하여 fingerprint 생성
        const rawFingerprint = `${ip}-${gpu}-${cpuCores}-${memory}`;
        const fingerprint = btoa(rawFingerprint); // Base64 인코딩

        console.log("Fingerprint 생성됨:", fingerprint);
        return fingerprint;
    } catch (error) {
        console.error("Fingerprint 생성 중 오류 발생:", error);
        return "UNKNOWN_DEVICE";
    }
}
```

```
// signIn -> 로그시인시 접속환경이 다를 시 장치설정 DB 추가 생성
public void saveDevice(int memberNo, String fingerprint) { 1개 사용 위치

    Device device = new Device();
    device.setMemberNo(memberNo);
    device.setDeviceId(fingerprint);
    System.out.println("saveDevice 호출됨, memberNo: " + memberNo + ", fingerprint: " + fingerprint);

    // 동일한 fingerprint가 이미 존재하는지 확인
    Device checkDevice = prefsMapper.checkDevice(device);
    if (checkDevice == null) {
        // fingerprint가 다르면 새로운 디바이스 추가
        int result = prefsMapper.insertDevice(device);
        System.out.println("fingerprint 들어갔나? : "+result);
    }
    // 동일한 fingerprint가 있으면 아무 동작도 하지 않음 (중복 저장 방지)
}
```

```
// 마이크 권한을 요청하고 승인되면 페이지 새로고침
async function requestAudioPermission() { 사용 위치 표시
    try {
        const permissionStatus = await navigator.permissions.query({ name: "microphone" });

        if (permissionStatus.state !== "granted") {
            await navigator.mediaDevices.getUserMedia({ audio: true });
            alert("마이크 및 스피커 권한이 허용되었습니다!");
            location.reload();
        }
    } catch (error) {
        console.error("권한 요청 실패:", error);
        alert("마이크 및 스피커 권한을 허용해주세요.");
    }
}
```

```
// 사용 가능한 오디오 입력 및 출력 장치를 select 요소에 추가
async function getAudioDevices() { 사용 위치 표시
    const devices = await navigator.mediaDevices.enumerateDevices();
    micSelect.innerHTML = "";
    speakerSelect.innerHTML = "";

    devices.forEach(device => {
        const option = document.createElement("option");
        option.value = device.deviceId;
        option.textContent = device.label || "알 수 없는 장치";

        if (device.kind === "audioinput") {
            micSelect.appendChild(option);
        } else if (device.kind === "audiooutput") {
            speakerSelect.appendChild(option);
        }
    });
}
```

```
// 마이크 입력을 받아 시각화 및 테스트를 위한 오디오 분석기 초기화
async function startMicTest() { 사용 위치 표시
    try {
        if (micStream) stopMicTest();

        // 선택된 마이크 장치로부터 오디오 스트림 요청
        micStream = await navigator.mediaDevices.getUserMedia({
            audio: { deviceId: micSelect.value ? { exact: micSelect.value } : undefined }
        });

        // 오디오 관련 변수 초기화
        // 오디오 처리를 위한 컨텍스트 생성
        audioContext = new AudioContext();
        // 마이크 입력값을 받아오는 노드 생성
        analyser = audioContext.createAnalyser();
        // 오디오 스트림을 Web Audio API 입력 노드로 변환
        source = audioContext.createMediaStreamSource(micStream);
        // 입력 노드를 분석 노드에 연결
        source.connect(analyser);

        drawWaveform();
    } catch (error) {
        console.error("마이크 테스트 실패:", error);
        alert("마이크 테스트 중 오류가 발생했습니다.");
    }
}
```

```
// 현재 설정된 오디오 값을 서버에 저장
function sendAudioPreferences() { 사용 위치 표시
    const requestData = {
        audioInput: micSelect.value,
        inputValue: inputVolumeSlider.value,
        audioOutput: speakerSelect.value,
        outputValue: outputVolumeSlider.value,
        inputType: document.querySelector('input[name="inputType"]:checked').value,
    };

    const blob = new Blob([JSON.stringify(requestData)], { type: "application/json" });
    navigator.sendBeacon("/prefs/audio", blob);
}
```

```
// 마이크 입력을 실시간으로 캔버스에 시각화

function drawWaveform() { 사용 위치 표시
    const canvas = document.createElement("canvas");
    canvas.width = 1024;
    canvas.height = 40;
    waveform.innerHTML = "";
    waveform.appendChild(canvas);
    const ctx = canvas.getContext("2d");

    // 분석 데이터를 기반으로 파형 그리기
    function update() { 사용 위치 표시
        requestAnimationFrame(update);

        let dataArray = new Uint8Array(analyser.frequencyBinCount);
        analyser.getByteFrequencyData(dataArray);

        ctx.clearRect(0, 0, canvas.width, canvas.height);
        ctx.fillStyle = "#7c3aed";

        // 특정 주파수 구간의 평균값을 시각화
        let total = dataArray.slice(5, 41).reduce((sum, val) => sum + val, 0);
        let average = total / 36;
        ctx.fillRect(0, 0, average * 4, 100);
    }
    update();
}
```

```
// 버튼 및 이벤트 등록  
testButton.addEventListener("click", startMicTest);  
stopButton.addEventListener("click", stopMicTest);  
requestPermissionButton.addEventListener("click", requestAudioPermission);  
window.addEventListener("beforeunload", sendAudioPreferences);
```

```
# HTTPS ssl 인증키 설정  
server.ssl.key-store=classpath:keystore.jks  
server.ssl.key-store-password=realman  
server.ssl.key-store-type=JKS  
server.ssl.key-alias=tomcat  
server.ssl.key-password=realman  
server.forward-headers-strategy=native
```