​​**기능 요구사항 작성 템플릿 (팀 전체용)**

📝 이 문서는 팀원들이 각자 맡은 기능의 동작 방식을 명확히 정의하고, 협업 중 충돌을 방지하기 위해 작성됩니다.  
각자 맡은 기능에 대해 아래 항목을 채워 주세요.

## **✅ 1. 기능 이름**

[메시지]

텍스트 메시지 전송

실시간 메시지 수신

이전 메시지 목록 조회

메시지 읽음 처리

메시지 삭제

메시지 답장

메시지 검색

{메시지 전달}

[채팅방]

채팅방 생성

그룹 채팅방 사용자 초대

그룹 채팅방 나가기

채팅방 정보 수정 (이름 변경)

{채팅방 알림 설정}

## **✅ 2. 기능 설명 (한 줄 요약)**

제목과 상동

## **✅ 3. 주요 흐름 / 시나리오**

### **2.1. Use Case 1: 텍스트 메시지 전송 (UC-CHAT-001)**

* **ID:** UC-CHAT-001
* **Actor:** 사용자 (로그인 상태)
* **Description:** 사용자가 참여 중인 채팅방에 텍스트 메시지를 입력하고 전송합니다.
* **Trigger:** 사용자가 채팅방 화면에서 메시지 입력 후 '전송' 액션을 수행 (버튼 클릭, 키보드 Enter 등)
* **Preconditions:**
  + 사용자는 시스템에 로그인되어 유효한 인증 상태를 가집니다.
  + 사용자는 대상 채팅방의 멤버입니다.
  + 클라이언트는 서버와 정상적인 네트워크 연결 상태입니다 (HTTP/HTTPS 및 WebSocket).
* **Postconditions:**
  + 전송된 메시지는 서버 데이터베이스에 영구 저장됩니다.
  + 메시지는 해당 채팅방에 참여 중인 모든 사용자(본인 포함)에게 실시간으로 전달됩니다.
  + 전송한 클라이언트 화면에 해당 메시지가 표시됩니다 (전송 상태 표시 포함 가능).
* **Main Success Scenario:**
  1. 사용자가 채팅방 UI에서 텍스트 입력 필드에 메시지를 입력합니다. (입력 시 '@'를 통해 사용자 멘션 가능)
  2. 사용자가 '전송' 액션을 트리거합니다.
  3. 클라이언트는 입력된 텍스트 내용, 대상 roomId, (멘션 시) mentionedUserIds를 포함하여 서버의 메시지 전송 API(POST /api/v1/chatrooms/{roomId}/messages)를 호출합니다. (인증 정보는 헤더에 포함)
  4. 서버는 요청을 수신하고 다음을 검증합니다:
     + 사용자 인증 유효성
     + 사용자의 해당 roomId 채팅방 참여 권한
     + 메시지 내용 유효성 (최대 길이 제한 등)
  5. 검증 통과 시, 서버는 다음 정보를 포함하는 메시지 객체를 생성합니다:
     + 고유 messageId (UUID 등)
     + roomId
     + senderId (인증된 사용자 ID)
     + 메시지 content (텍스트)
     + 서버 timestamp (메시지 생성 시각, UTC)
     + (멘션 시) mentionedUserIds
     + (선택) 초기 status (예: 'sent')
  6. 서버는 생성된 메시지 객체를 데이터베이스 (Messages 테이블 등)에 저장합니다.
  7. 서버는 해당 roomId에 연결된 WebSocket 클라이언트들에게 새 메시지 정보(멘션 정보 포함)를 포함하는 이벤트(예: new\_message)를 Broadcast 합니다.
  8. 서버는 API 요청을 보낸 클라이언트에게 성공 응답 (201 Created)을 반환합니다. 응답 본문에는 생성된 messageId, timestamp 등이 포함될 수 있습니다.
  9. (Client) API 요청을 보낸 클라이언트는 UI에 메시지를 즉시 표시합니다. (Optimistic Update). 초기 상태를 '전송 중'으로 표시할 수 있습니다.
  10. (Client) 해당 채팅방에 연결된 모든 클라이언트(전송자 포함)는 WebSocket을 통해 new\_message 이벤트를 수신합니다.
  11. (Client) 클라이언트 UIs는 수신된 메시지를 렌더링하고, 필요한 경우 스크롤을 조정합니다. 멘션된 사용자는 강조 표시를 확인합니다.
* **Error/Alternative Scenarios:**
  + **4.1 (입력값 오류):** 메시지 길이 초과 시 400 Bad Request. 클라이언트는 오류 표시.
  + **4.2 (인증 실패):** 401 Unauthorized. 클라이언트는 재로그인 유도.
  + **4.3 (권한 없음):** 403 Forbidden. 클라이언트는 접근 불가 메시지 표시.
  + **4.4 (채팅방 없음):** 404 Not Found.
  + **6.1 (DB 저장 실패):** 500 Internal Server Error. 클라이언트는 오류 메시지 표시 및 재시도 유도 가능. WebSocket Broadcast 미수행.
  + **7.1 (WebSocket 연결 끊김):** API 호출은 성공 가능하나 실시간 수신 실패. 클라이언트는 연결 재시도 및 누락 메시지 동기화 필요.
  + **Network Error:** 클라이언트-서버 간 통신 오류 시 재시도 또는 오류 알림.

### **2.2. Use Case 2: 실시간 메시지 수신 (UC-CHAT-002)**

* **ID:** UC-CHAT-002
* **Actor:** 사용자 (로그인 상태)
* **Description:** 사용자가 참여 중인 채팅방에서 다른 사용자가 보낸 새 메시지를 실시간으로 수신하여 확인합니다.
* **Trigger:** 서버가 사용자가 참여 중인 채팅방의 WebSocket 채널로 새 메시지 이벤트를 Broadcast 함.
* **Preconditions:**
  + 사용자는 시스템에 로그인되어 유효한 인증 상태를 가집니다.
  + 사용자는 해당 채팅방에 참여 중입니다.
  + 클라이언트는 해당 채팅방의 WebSocket 채널에 연결(Subscribe)되어 있습니다.
* **Postconditions:**
  + 새 메시지가 클라이언트 UI의 채팅 목록 최하단에 표시됩니다.
  + (선택) 채팅방 목록 등에서 안 읽은 메시지 카운트가 업데이트됩니다.
  + (선택) 사용자가 현재 채팅방을 보고 있지 않다면, 특정 알림(뱃지 등)이 표시될 수 있습니다.
* **Main Success Scenario:**
  1. 클라이언트는 채팅방에 진입 시 해당 roomId에 대한 WebSocket 연결 및 구독을 수행합니다.
  2. 다른 사용자가 보낸 메시지가 서버에 의해 처리되고, 해당 roomId 채널로 new\_message 이벤트가 Broadcast 됩니다. (UC-CHAT-001의 7단계)
  3. 클라이언트는 구독 중인 채널을 통해 new\_message 이벤트를 수신합니다. 이벤트 Payload에는 메시지 정보(messageId, roomId, senderId, senderNickname, content, timestamp, mentionedUserIds 등)가 포함됩니다.
  4. 클라이언트는 수신된 메시지 데이터를 파싱합니다.
  5. 클라이언트는 채팅 UI의 메시지 목록 최하단에 새 메시지를 렌더링합니다. (보낸 사람 정보, 메시지 내용, 시간 등 포함). 자신을 향한 멘션은 강조 표시합니다.
  6. 사용자가 현재 화면 스크롤 최하단에 위치해 있다면, 새 메시지가 보이도록 스크롤을 자동으로 내립니다. 그렇지 않다면, '새 메시지 도착' 알림 등을 표시하여 사용자가 직접 스크롤하도록 유도할 수 있습니다.
  7. (선택) 해당 메시지가 내가 보낸 메시지가 아니라면, 안 읽음 카운트를 업데이트하고 읽음 처리 로직을 준비합니다. (UC-CHAT-004 참고)
* **Error/Alternative Scenarios (타이핑 인디케이터 포함):**
  + **WebSocket 연결 끊김:** 메시지 수신 중 연결이 끊어진 경우, 클라이언트는 재연결 로직을 수행합니다. 재연결 후에는 마지막으로 수신한 메시지 이후의 누락된 메시지를 가져오기 위해 서버 API(GET /api/v1/chatrooms/{roomId}/messages?since={lastMessageTimestamp})를 호출해야 합니다.
  + **메시지 파싱 오류:** 수신된 WebSocket 메시지 형식이 예상과 다를 경우, 오류를 로깅하고 해당 메시지는 표시하지 못할 수 있습니다. (개발 중 디버깅 필요)
  + **타이핑 인디케이터:**
    - 다른 사용자가 입력을 시작하면 서버로부터 user\_typing 이벤트를 수신하여 'OOO님이 입력 중입니다...' 표시.
    - 해당 사용자가 입력 중지/메시지 전송 시 user\_typing\_end 이벤트를 수신하여 인디케이터 숨김.

### **2.3. Use Case 3: 이전 메시지 목록 조회 (UC-CHAT-003)**

* **ID:** UC-CHAT-003
* **Actor:** 사용자 (로그인 상태)
* **Description:** 사용자가 채팅방에 입장하거나, 채팅 목록을 위로 스크롤하여 이전 대화 내용을 조회합니다.
* **Trigger:**
  + 사용자가 채팅방 화면에 처음 진입함.
  + 사용자가 채팅 메시지 목록을 위로 스크롤하여 최상단에 도달함.
* **Preconditions:**
  + 사용자는 시스템에 로그인되어 유효한 인증 상태를 가집니다.
  + 사용자는 대상 채팅방의 멤버입니다.
* **Postconditions:**
  + 요청된 범위의 이전 메시지 목록이 클라이언트 UI에 표시됩니다.
* **Main Success Scenario (채팅방 최초 진입 시):**
  1. 사용자가 특정 채팅방 UI로 진입합니다.
  2. 클라이언트는 해당 채팅방의 가장 최근 메시지 N개(예: 30개)를 가져오기 위해 서버 API (GET /api/v1/chatrooms/{roomId}/messages?limit=30)를 호출합니다.
  3. 서버는 요청을 검증 (인증, 권한)하고, 해당 채팅방의 메시지를 최신순으로 정렬하여 limit 개수만큼 조회합니다.
  4. 서버는 조회된 메시지 목록(각 메시지는 messageId, senderId, content, timestamp 등 포함)을 JSON 배열 형태로 200 OK 응답과 함께 반환합니다.
  5. 클라이언트는 응답받은 메시지 목록을 파싱하여 시간 순서대로 (오래된 메시지가 위로) UI에 렌더링합니다. 스크롤은 가장 최신 메시지가 보이도록 최하단으로 이동합니다.
* **Main Success Scenario (스크롤 기반 로딩 시):**
  1. 사용자가 메시지 목록을 위로 스크롤하여 현재 로드된 메시지 중 가장 오래된 메시지에 근접하거나 도달합니다.
  2. 클라이언트는 현재 로드된 가장 오래된 메시지의 timestamp 또는 messageId를 기준으로, 그 이전 메시지 N개를 요청하는 API (GET /api/v1/chatrooms/{roomId}/messages?limit=30&before={oldestMessageTimestamp})를 호출합니다.
  3. 서버는 요청을 검증하고, before 파라미터 값보다 오래된 메시지를 최신순으로 limit 개수만큼 조회합니다.
  4. 서버는 조회된 메시지 목록을 JSON 배열 형태로 200 OK 응답과 함께 반환합니다.
  5. 클라이언트는 응답받은 메시지 목록을 파싱하여 현재 UI 목록의 *상단에* 시간 순서대로 추가합니다. 스크롤 위치는 사용자가 보고 있던 위치를 유지하도록 조정합니다. 더 이상 로드할 메시지가 없으면 로딩 인디케이터를 숨깁니다.
* **Error/Alternative Scenarios:**
  + **인증/권한/채팅방 없음 오류:** UC-CHAT-001과 유사하게 401, 403, 404 에러 처리.
  + **네트워크 오류:** 재시도 로직 또는 오류 메시지 표시.
  + **더 이상 메시지 없음:** 서버는 빈 배열 []과 200 OK를 반환합니다. 클라이언트는 더 이상 이전 메시지를 로드하지 않음을 인지하고 로딩 인디케이터를 표시하지 않습니다.

### **2.4. Use Case 4: 메시지 읽음 처리 (UC-CHAT-004)**

* **ID:** UC-CHAT-004
* **Actor:** 사용자 (로그인 상태), 시스템
* **Description:** 사용자가 특정 메시지를 읽었음을 시스템에 알리고, 다른 사용자에게 해당 메시지의 안 읽은 사람 수가 업데이트됨을 표시합니다.
* **Trigger:**
  + 사용자가 채팅방에 들어와 특정 메시지까지 확인(스크롤 등)했을 때.
  + 다른 사용자가 보낸 새 메시지가 화면에 표시되었을 때 (자동 읽음 처리).
* **Preconditions:**
  + 사용자는 시스템에 로그인되어 유효한 인증 상태입니다.
  + 읽음 처리 대상 메시지가 존재합니다.
* **Postconditions:**
  + 서버에 해당 사용자가 특정 메시지까지 읽었음이 기록됩니다.
  + 해당 채팅방의 다른 사용자들에게 특정 메시지의 '안 읽음 수'가 실시간으로 업데이트됩니다.
* **Main Success Scenario:**
  1. 사용자가 채팅방 화면에서 특정 메시지(예: targetMessageId)까지 내용을 확인합니다. (UI 상 가시적 영역 진입 기준 등)
  2. 클라이언트는 사용자가 해당 메시지까지 읽었음을 서버에 알리는 API (POST /api/v1/chatrooms/{roomId}/read) 또는 WebSocket 메시지(예: mark\_read)를 전송합니다. 요청에는 마지막으로 읽은 메시지의 messageId 또는 timestamp가 포함될 수 있습니다.
  3. 서버는 요청을 검증하고, 해당 사용자의 roomId에 대한 '마지막 읽은 시점(lastReadTimestamp)' 정보를 데이터베이스(ChatRoomMembers 테이블 등)에 업데이트합니다.
  4. 서버는 해당 채팅방의 각 메시지에 대해 안 읽은 사용자 수를 다시 계산합니다. (메시지 timestamp > 사용자의 lastReadTimestamp 인 경우 안 읽음)
  5. 서버는 변경된 안 읽음 수 정보를 해당 roomId에 연결된 WebSocket 클라이언트들에게 실시간 이벤트(예: read\_update)로 Broadcast 합니다. Payload에는 각 messageId별 안 읽음 수가 포함될 수 있습니다.
  6. 클라이언트는 read\_update 이벤트를 수신하고, UI에 표시된 각 메시지 옆의 안 읽음 카운트를 업데이트합니다. (0이 되면 카운트 숨김)
* **Error/Alternative Scenarios:**
  + API/WS 요청 실패 시 재시도 로직 수행.
  + 서버 처리 실패 시 로깅 및 오류 응답. 실시간 업데이트 실패 가능.

### **2.5. Use Case 5: 채팅방 생성 (UC-CHAT-005)**

·

**ID:** UC-CHAT-005

·

**Actor:** 사용자 (로그인 상태)

·

**Description:** 사용자가 새로운 1:1 채팅방 또는 그룹 채팅방을 생성합니다.

·

**Trigger:**

o (1:1) 사용자 목록에서 특정 사용자를 선택하고 '채팅하기' 액션 수행.

o (그룹) 사용자 목록에서 2명 이상의 사용자를 선택하고 '그룹 채팅 시작' 액션 수행.

·

**Preconditions:** 사용자는 로그인 상태여야 합니다. 그룹 채팅 생성을 위해 최소 1명 이상의 다른 사용자를 선택해야 합니다.

·

**Postconditions:**

o 새로운 채팅방 정보가 서버에 생성됩니다.

o 채팅방 참여자 정보가 서버에 기록됩니다.

o 채팅방 생성자 및 참여자에게 새 채팅방이 생성되었음이 (목록 갱신 등으로) 표시됩니다.

·

**Main Success Scenario (그룹 채팅 기준):**

1.

사용자가 채팅방에 초대할 사용자 목록(최소 1명 이상)을 선택합니다.

2.

사용자가 '그룹 채팅 생성' 액션을 트리거합니다. (선택적으로 초기 채팅방 이름을 입력할 수 있습니다.)

3.

클라이언트는 선택된 사용자 ID 목록(memberIds)과 초기 채팅방 이름(roomName, 옵션)을 포함하여 서버의 채팅방 생성 API (POST

/api/v1/chatrooms)를 호출합니다.

4.

서버는 요청을 검증하고 다음을 수행합니다:

§ 새로운 고유 roomId 생성.

§ ChatRooms 테이블에 채팅방 정보 저장 (생성자 ID, 초기 이름 - 없으면 참여자 이름 조합, 타입='GROUP', 생성 시각 등).

§ ChatRoomMembers

테이블에 생성자를 포함한 모든 참여자 정보 저장.

5.

서버는 생성된 채팅방 정보 (roomId,

roomName, members, createdAt 등)를 포함하여 201 Created 응답을 반환합니다.

6.

(Client) API 응답 성공 시, 클라이언트는 새 채팅방 UI로 이동하거나 채팅 목록을 갱신합니다.

7.

(Real-time) 서버는 WebSocket을 통해 참여자들에게 new\_chatroom 이벤트를 전송하여 클라이언트들이 채팅 목록을 실시간으로 갱신하도록 할 수 있습니다.

·

**Error/Alternative Scenarios:**

o (1:1 생성 시) 이미 해당 사용자와의 1:1 채팅방이 존재하면, 새 방을 만들지 않고 기존 채팅방 정보를 반환하며 200 OK 또는 303 See Other 응답을 줄 수 있습니다.

o 선택한 사용자가 유효하지 않거나 탈퇴한 사용자일 경우 400 Bad

Request 응답.

o 서버 오류 시 500 Internal

Server Error.

### **2.6. Use Case 6: 그룹 채팅방 사용자 초대 (UC-CHAT-006)**

·

**ID:** UC-CHAT-006

·

**Actor:** 그룹 채팅방 참여자

·

**Description:** 기존 그룹 채팅방에 새로운 사용자를 초대합니다.

·

**Trigger:** 채팅방 설정 메뉴 등에서 '사용자 초대' 액션 수행 후 초대할 사용자 선택.

·

**Preconditions:** 사용자는 해당 그룹 채팅방의 멤버여야 합니다. 초대 대상 사용자는 아직 해당 채팅방 멤버가 아니어야 합니다.

·

**Postconditions:** 초대된 사용자가 채팅방 멤버로 추가됩니다. 채팅방 내 모든 참여자에게 사용자 초대 알림 메시지가 시스템 메시지로 표시됩니다.

·

**Main Success Scenario:**

1.

사용자가 채팅방 설정에서 초대할 사용자 목록을 선택하고 '초대' 액션을 트리거합니다.

2.

클라이언트는 초대할 사용자 ID 목록(inviteeIds)을 포함하여 서버의 사용자 초대 API (POST /api/v1/chatrooms/{roomId}/members)를 호출합니다.

3.

서버는 요청자 권한 및 초대 대상 유효성을 검증합니다.

4.

서버는 ChatRoomMembers 테이블에 새로운 멤버 정보를 추가합니다.

5.

서버는 채팅방에 "OOO님이 XXX님, YYY님을 초대했습니다." 와 같은 시스템 메시지를 생성하여 저장하고, 해당 메시지를 new\_message 이벤트로 Broadcast 합니다.

6.

서버는 200 OK 응답을 반환합니다.

7.

(Real-time) 서버는 WebSocket을 통해 기존/신규 참여자들에게 member\_joined 이벤트를 전송하여 클라이언트들이 참여자 목록 및 시스템 메시지를 갱신하도록 합니다.

·

**Error/Alternative Scenarios:** 권한 없음(403), 이미 참여 중인 사용자 포함(400), 사용자 없음(404), 서버 오류(500).

### **2.7. Use Case 7: 그룹 채팅방 나가기 (UC-CHAT-007)**

·

**ID:** UC-CHAT-007

·

**Actor:** 그룹 채팅방 참여자

·

**Description:** 사용자가 참여 중인 그룹 채팅방에서 나갑니다.

·

**Trigger:** 채팅방 설정 메뉴 등에서 '채팅방 나가기' 액션 수행.

·

**Preconditions:** 사용자는 해당 그룹 채팅방의 멤버여야 합니다.

·

**Postconditions:** 사용자는 더 이상 해당 채팅방의 멤버가 아니며, 메시지를 받거나 보낼 수 없습니다. 채팅방 내 다른 참여자에게 사용자 퇴장 알림 메시지가 시스템 메시지로 표시됩니다.

·

**Main Success Scenario:**

1.

사용자가 '채팅방 나가기'를 확인합니다.

2.

클라이언트는 서버의 채팅방 나가기 API (DELETE

/api/v1/chatrooms/{roomId}/members/me)를 호출합니다.

3.

서버는 요청자 권한을 검증하고

ChatRoomMembers 테이블에서 해당 사용자 정보를 삭제(또는 비활성화)합니다.

4.

서버는 채팅방에 "OOO님이 나갔습니다." 와 같은 시스템 메시지를 생성하여 저장하고, 해당 메시지를 new\_message 이벤트로 Broadcast 합니다.

5.

서버는 204 No Content 응답을 반환합니다.

6.

(Client) 클라이언트는 해당 채팅방 UI를 닫거나 채팅 목록에서 제거합니다.

7.

(Real-time) 서버는 WebSocket을 통해 남은 참여자들에게 member\_left 이벤트를 전송하여 클라이언트들이 참여자 목록 및 시스템 메시지를 갱신하도록 합니다.

·

**Error/Alternative Scenarios:** 권한 없음(403), 서버 오류(500). 채팅방에 마지막 남은 사용자일 경우 채팅방 비활성화/삭제 로직 추가 고려.

### **2.8. Use Case 8: 채팅방 정보 수정 (이름 변경) (UC-CHAT-008)**

·

**ID:** UC-CHAT-008

·

**Actor:** 그룹 채팅방 참여자 (또는 특정 권한자)

·

**Description:** 그룹 채팅방의 이름을 변경합니다.

·

**Trigger:** 채팅방 설정 메뉴에서 '채팅방 이름 변경' 액션 수행 후 새 이름 입력.

·

**Preconditions:** 사용자는 해당 그룹 채팅방의 멤버여야 합니다. (이름 변경 권한 정책 필요 - 예: 누구나 가능 또는 방장만 가능)

·

**Postconditions:** 채팅방 이름이 변경됩니다. 참여자들에게 이름 변경 알림 메시지가 시스템 메시지로 표시됩니다.

·

**Main Success Scenario:**

1.

사용자가 새 채팅방 이름을 입력하고 '확인' 액션을 트리거합니다.

2.

클라이언트는 새 이름(newName)을 포함하여 서버의 채팅방 정보 수정 API (PATCH /api/v1/chatrooms/{roomId})를 호출합니다.

3.

서버는 요청자 권한 및 새 이름 유효성을 검증합니다.

4.

서버는 ChatRooms 테이블의 해당 roomId의 이름을 업데이트합니다.

5.

서버는 "OOO님이 채팅방 이름을 'XXX'로 변경했습니다." 와 같은 시스템 메시지를 생성/저장하고 new\_message 이벤트로 Broadcast 합니다.

6.

서버는 200 OK 응답을 반환합니다.

7.

(Real-time) 서버는 WebSocket을 통해 참여자들에게 room\_updated 이벤트(변경된 이름 포함)를 전송하여 클라이언트들이 채팅방 제목 등을 갱신하도록 합니다.

·

**Error/Alternative Scenarios:** 권한 없음(403), 이름 형식 오류(400), 서버 오류(500).

### **2.9. Use Case 9: 채팅방 알림 설정 (UC-CHAT-009)**

·

**ID:** UC-CHAT-009

·

**Actor:** 사용자 (로그인 상태)

·

**Description:** 사용자가 특정 채팅방의 알림 수신 여부를 설정합니다. (켜기/끄기)

·

**Trigger:** 채팅방 설정 메뉴 또는 채팅 목록에서 특정 채팅방의 '알림 끄기/켜기' 토글 액션 수행.

·

**Preconditions:** 사용자는 해당 채팅방의 멤버여야 합니다.

·

**Postconditions:** 사용자의 해당 채팅방에 대한 알림 설정 상태가 변경됩니다. (실제 푸시 알림 발송 로직은 이 설정 값을 참조)

·

**Main Success Scenario:**

1.

사용자가 알림 켜기/끄기 액션을 트리거합니다.

2.

클라이언트는 변경할 알림 상태(notificationEnabled:

true/false)를 포함하여 서버의 알림 설정 API (PUT

/api/v1/chatrooms/{roomId}/members/me/notification)를 호출합니다.

3.

서버는 요청을 검증하고, 해당 사용자의 ChatRoomMembers 테이블 또는 별도의 사용자 설정 테이블에 알림 상태를 업데이트합니다.

4.

서버는 200 OK 응답을 반환합니다.

5.

(Client) 클라이언트는 UI에 변경된 알림 상태(예: 음소거 아이콘 표시)를 반영합니다.

·

**Error/Alternative Scenarios:** 권한 없음(403), 서버 오류(500).

### **[신규] 메시지 관련 추가 기능**

### **2.10. Use Case 10: 메시지 삭제 (UC-CHAT-010)**

·

**ID:** UC-CHAT-010

·

**Actor:** 메시지 발신자

·

**Description:** 사용자가 자신이 보낸 메시지를 채팅방에서 삭제합니다.

·

**Trigger:** 자신이 보낸 메시지를 길게 누르거나 우클릭하여 '삭제' 옵션 선택.

·

**Preconditions:** 사용자는 삭제하려는 메시지의 발신자여야 합니다. 메시지 삭제 가능 시간 제한이 있을 수 있습니다 (예: 전송 후 5분 이내).

·

**Postconditions:** 해당 메시지 내용은 "삭제된 메시지입니다." 와 같은 문구로 대체됩니다. 메시지 자체가 DB에서 물리적으로 삭제되지는 않을 수 있습니다 (상태 변경).

·

**Main Success Scenario:**

1.

사용자가 삭제할 자신의 메시지를 선택하고 '삭제' 액션을 확인합니다.

2.

클라이언트는 삭제할 messageId를 포함하여 서버의 메시지 삭제 API (DELETE /api/v1/chatrooms/{roomId}/messages/{messageId})를 호출합니다.

3.

서버는 요청자 권한(메시지 발신자 일치 여부) 및 삭제 가능 시간 등을 검증합니다.

4.

서버는 Messages 테이블에서 해당 메시지의 상태를 'DELETED'로 변경하거나, 내용을 "삭제된 메시지입니다."로 업데이트합니다. (원본 내용 보존 여부 정책 결정 필요)

5.

서버는 204 No Content 응답을 반환합니다.

6.

(Real-time) 서버는 WebSocket을 통해 해당 채팅방 참여자들에게 message\_deleted 이벤트 (Payload:

{"messageId": "uuid", "roomId":

"uuid"})를 Broadcast 합니다.

7.

클라이언트는 이 이벤트를 수신하여 UI에서 해당 메시지 내용을 "삭제된 메시지입니다."로 변경합니다.

·

**Error/Alternative Scenarios:** 권한 없음(403), 시간 초과(400), 메시지 없음(404), 서버 오류(500).

### **2.11. Use Case 11: 메시지 수정 (UC-CHAT-011)**

·

**ID:** UC-CHAT-011

·

**Actor:** 메시지 발신자

·

**Description:** 사용자가 자신이 보낸 메시지의 내용을 수정합니다.

·

**Trigger:** 자신이 보낸 메시지를 길게 누르거나 우클릭하여 '수정' 옵션 선택 후 새 내용 입력.

·

**Preconditions:** 사용자는 수정하려는 메시지의 발신자여야 합니다. 메시지 수정 가능 시간 제한이 있을 수 있습니다.

·

**Postconditions:** 메시지 내용이 수정되고, '수정됨' 표시가 추가됩니다.

·

**Main Success Scenario:**

1.

사용자가 수정할 자신의 메시지를 선택하고 새 내용을 입력한 후 '확인' 액션을 트리거합니다.

2.

클라이언트는 수정할 messageId와 새로운 내용(newContent)을 포함하여 서버의 메시지 수정 API (PATCH

/api/v1/chatrooms/{roomId}/messages/{messageId})를 호출합니다.

3.

서버는 요청자 권한, 수정 가능 시간, 새 내용 유효성을 검증합니다.

4.

서버는 Messages 테이블에서 해당 메시지의 content를 업데이트하고, is\_edited 플래그를 true로 설정합니다. 수정 시각(updated\_at)도 기록합니다.

5.

서버는 200 OK 응답 (수정된 메시지 정보 포함 가능)을 반환합니다.

6.

(Real-time) 서버는 WebSocket을 통해 해당 채팅방 참여자들에게 message\_updated 이벤트 (Payload:

{"messageId": "uuid", "roomId": "uuid",

"newContent": "string", "isEdited": true,

"updatedAt": "iso8601"})를 Broadcast 합니다.

7.

클라이언트는 이 이벤트를 수신하여 UI에서 해당 메시지 내용을 업데이트하고 '수정됨' 표시를 추가합니다.

·

**Error/Alternative Scenarios:** 권한 없음(403), 시간 초과(400), 메시지 없음(404), 내용 형식 오류(400), 서버 오류(500).

### **2.12. Use Case 12: 메시지 답장 (UC-CHAT-012)**

·

**ID:** UC-CHAT-012

·

**Actor:** 사용자 (로그인 상태)

·

**Description:** 사용자가 특정 메시지를 인용하여 답장 메시지를 보냅니다.

·

**Trigger:** 특정 메시지를 스와이프하거나 길게 눌러 '답장' 옵션 선택 후 답장 내용 입력 및 전송.

·

**Preconditions:** 사용자는 로그인 상태이며 해당 채팅방 멤버입니다. 원본 메시지가 존재해야 합니다.

·

**Postconditions:** 답장 메시지가 전송되며, 메시지 상단에 인용된 원본 메시지의 일부(발신자, 내용 미리보기)가 표시됩니다.

·

**Main Success Scenario:**

1.

사용자가 답장할 원본 메시지를 선택하고 답장 내용을 입력한 후 '전송' 액션을 트리거합니다.

2.

클라이언트는 답장 내용(content)과 함께 원본 메시지 ID(originalMessageId)를 포함하여 서버의 메시지 전송 API (POST /api/v1/chatrooms/{roomId}/messages)를 호출합니다. (기존 전송 API 재활용 또는 확장)

3.

서버는 요청을 검증하고 메시지를 생성/저장 시, original\_message\_id 필드에 원본 메시지 ID를 기록합니다.

4.

이후 메시지 전송/수신 흐름은 UC-CHAT-001, UC-CHAT-002와 유사하게 진행됩니다.

5.

(Real-time) WebSocket new\_message 이벤트 Payload에

originalMessage 정보 (ID,

senderNickname, previewContent)를 포함하여 전송합니다.

6.

클라이언트는 new\_message 이벤트 수신 시,

originalMessage 정보가 있으면 메시지 UI 상단에 인용 UI를 함께 렌더링합니다. 인용 UI 클릭 시 원본 메시지 위치로 스크롤 이동하는 기능 추가 가능.

·

**Error/Alternative Scenarios:** 원본 메시지 없음(404), 기타 전송 오류는 UC-CHAT-001과 동일.

### **2.13. Use Case 13: 메시지 검색 (UC-CHAT-013)**

·

**ID:** UC-CHAT-013

·

**Actor:** 사용자 (로그인 상태)

·

**Description:** 사용자가 현재 채팅방 내에서 특정 키워드가 포함된 메시지를 검색합니다.

·

**Trigger:** 채팅방 내 검색 아이콘 클릭 후 검색 키워드 입력.

·

**Preconditions:** 사용자는 해당 채팅방의 멤버여야 합니다.

·

**Postconditions:** 검색 키워드가 포함된 메시지 목록이 시간 역순으로 표시됩니다. 검색 결과 클릭 시 해당 메시지 위치로 채팅 스크롤이 이동합니다.

·

**Main Success Scenario:**

1.

사용자가 검색 키워드를 입력하고 검색 액션을 트리거합니다.

2.

클라이언트는 검색 키워드(query)를 포함하여 서버의 메시지 검색 API (GET

/api/v1/chatrooms/{roomId}/search?query={query})를 호출합니다.

3.

서버는 요청을 검증하고, 해당 채팅방의 Messages 테이블에서 content 필드에 키워드가 포함된 메시지를 검색합니다. (Full-Text Search 엔진 사용 권장)

4.

서버는 검색된 메시지 목록(messageId,

senderId, content, timestamp 등 포함)을 시간 역순으로 정렬하여 200 OK 응답과 함께 반환합니다. (페이징 처리 필요)

5.

클라이언트는 검색 결과를 별도의 UI 또는 현재 채팅 UI 상단에 표시합니다.

6.

사용자가 검색 결과를 클릭하면, 클라이언트는 해당 messageId를 기준으로 이전/이후 메시지를 로드하고(UC-CHAT-003 활용) 해당 메시지 위치로 스크롤을 이동시킨 후 하이라이트 효과를 줍니다.

·

**Error/Alternative Scenarios:** 권한 없음(403), 검색어 없음/부적절(400), 서버 오류(500). 검색 성능 최적화 필요.

### **2.14. Use Case 14: 메시지 전달 (UC-CHAT-014)**

·

**ID:** UC-CHAT-014

·

**Actor:** 사용자 (로그인 상태)

·

**Description:** 사용자가 특정 메시지를 다른 채팅방(들)으로 전달합니다.

·

**Trigger:** 메시지를 길게 누르거나 우클릭하여 '전달' 옵션 선택 후 전달 대상 채팅방(들) 선택.

·

**Preconditions:** 사용자는 원본 메시지가 있는 채팅방과 전달 대상 채팅방 모두의 멤버여야 합니다.

·

**Postconditions:** 선택된 메시지가 지정된 다른 채팅방(들)에 새로운 메시지로 전송됩니다. 전달된 메시지임을 나타내는 표시가 있을 수 있습니다.

·

**Main Success Scenario:**

1.

사용자가 전달할 메시지를 선택하고 '전달' 액션을 트리거합니다.

2.

클라이언트는 전달 대상 채팅방 목록 선택 UI를 표시합니다.

3.

사용자가 대상 채팅방(들)을 선택하고 '확인' 액션을 트리거합니다.

4.

클라이언트는 원본 메시지 ID(originalMessageId)와 대상 채팅방 ID 목록(targetRoomIds)을 포함하여 서버의 메시지 전달 API (POST /api/v1/messages/forward)를 호출합니다.

5.

서버는 요청을 검증 (사용자의 각 채팅방 참여 권한, 원본 메시지 유효성 등)합니다.

6.

서버는 각 targetRoomId에 대해 원본 메시지 내용을 복사하여 새로운 메시지를 생성/저장합니다. (이때 senderId는 전달자인 현재 사용자로 설정, is\_forwarded 플래그 true 설정 가능)

7.

서버는 각 대상 채팅방에 대해 new\_message WebSocket 이벤트를 Broadcast 합니다. (Payload에 isForwarded: true 포함 가능)

8.

서버는 전달 결과를 요약하여 200 OK 또는 202 Accepted 응답을 반환합니다.

9.

클라이언트는 전달 성공/실패 여부를 사용자에게 알립니다.

**Error/Alternative Scenarios:** 권한 없음(403), 대상 채팅방 없음(404), 원본 메시지 없음(404), 서버 오류(500). 일부 채팅방 전달 실패 시 부분 성공/실패 처리 필요.

## **✅ 4. 요청 DTO 예시**

```json  
{  
 "key": "value"  
}  
```?

## **✅ 5. 응답 형식 예시**

**성공 시 (예: 메시지 전송 성공 - 201 Created)**  
{  
 "status": 201,  
 "message": "Message sent successfully",  
 "data": {  
 "messageId": "msg-abc-12345",  
 "roomId": "room-xyz-789",  
 "senderId": "user-me-001",  
 "content": "안녕하세요! 답장입니다.",  
 "timestamp": "2025-04-15T16:55:00Z",  
 "originalMessageId": "msg-origin-67890",  
 "mentionedUserIds": ["user-friend-002"]  
 }  
}

* **실패 시 (예: 입력값 오류 - 400 Bad Request)**  
  {  
   "status": 400,  
   "message": "메시지 내용은 필수입니다.",  
   "data": null  
  }
* **실패 시 (예: 인증 실패 - 401 Unauthorized)**  
  {  
   "status": 401,  
   "message": "인증되지 않은 사용자입니다. 로그인이 필요합니다.",  
   "data": null  
  }
* **실패 시 (예: 권한 없음 - 403 Forbidden)**  
  {  
   "status": 403,  
   "message": "해당 채팅방에 접근할 권한이 없습니다.",  
   "data": null  
  }

## **✅ 6. 예외 케이스 / 실패 상황**

- 필수 값 누락 → 400 에러  
- 잘못된 형식 → 400 에러  
- 중복된 요청 → 409 에러  
- 인증되지 않은 사용자 요청 → 401 에러

## **✅ 7. 엔드포인트 정보**

| **Method** | **URL** | **Description** | **인증 필요** |
| --- | --- | --- | --- |
| POST | /api/v1/chatrooms | 채팅방 생성 | **Yes** |
| POST | /api/v1/chatrooms/{roomId}/members | 사용자 초대 | **Yes** |
| DELETE | /api/v1/chatrooms/{roomId}/members/me | 채팅방 나가기 | **Yes** |
| PATCH | /api/v1/chatrooms/{roomId} | 채팅방 정보 수정 (이름 등) | **Yes** |
| PUT | /api/v1/chatrooms/{roomId}/members/me/notification | 채팅방 알림 설정 | **Yes** |
| DELETE | /api/v1/chatrooms/{roomId}/messages/{messageId} | 메시지 삭제 | **Yes** |
| PATCH | /api/v1/chatrooms/{roomId}/messages/{messageId} | 메시지 수정 | **Yes** |
| POST | /api/v1/chatrooms/{roomId}/messages | 메시지 전송 (답장 기능 포함) | **Yes** |
| GET | /api/v1/chatrooms/{roomId}/search | 메시지 검색 | **Yes** |
| POST | /api/v1/messages/forward | 메시지 전달 | **Yes** |

## **✅ 8. 담당자**

박범진

## **📌 예시 코드 (Controller 예시)**

@RestController  
@RequestMapping("/api/v1/chatrooms/{roomId}/messages")  
public class ChatMessageController {

private final

ChatMessageService chatMessageService;

// 생성자를 통한 의존성 주입  
 public

ChatMessageController(ChatMessageService chatMessageService) {

this.chatMessageService = chatMessageService;  
 }

/\*\*  
 \* 채팅 메시지 전송 API  
 \* @param roomId 메시지를 보낼 채팅방 ID (Path Variable)  
 \* @param sendMessageRequestDto 메시지 내용 DTO (Request Body)  
 \* @param userDetails 현재 인증된 사용자 정보 (Spring Security)  
 \* @return 생성된 메시지 정보 또는 성공 응답  
 \*/  
 @PostMapping  
 public

ResponseEntity<ApiResponse<SendMessageResponseDto>> sendMessage(

@PathVariable String roomId,

@Valid @RequestBody SendMessageRequestDto sendMessageRequestDto,

@AuthenticationPrincipal UserDetails userDetails) { // Spring Security 등 통해 인증된 사용자 정보 주입

// 현재 사용자 ID 가져오기 (UserDetails 구현에 따라 다름)  
 String userId =

userDetails.getUsername(); // 예시

//

Service 레이어 호출하여 메시지 전송 로직 수행

SendMessageResponseDto responseDto = chatMessageService.sendMessage(roomId,

userId, sendMessageRequestDto);

// 표준 응답 형식으로 감싸서 반환

ApiResponse<SendMessageResponseDto> response =

ApiResponse.success(responseDto);

// HTTP

Status 201 Created 와 함께 응답 반환  
 return

ResponseEntity.status(HttpStatus.CREATED).body(response);  
 }  
}

## **📌 예시 코드 (DTO 예시)**

// --- Request DTO ---  
  
/\*\*  
 \* 메시지 전송 요청 DTO  
 \*/  
public class SendMessageRequestDto {  
  
 @NotBlank(message = "메시지 내용은 필수입니다.")  
 @Size(max = 1000, message = "메시지는 최대 1000자까지 입력 가능합니다.")  
 private String content; // 메시지 내용  
  
 private String originalMessageId; // 답장할 원본 메시지 ID (옵션)  
  
 private List<String> mentionedUserIds; // 멘션된 사용자 ID 목록 (옵션)  
  
 // Getters and Setters or Lombok @Data  
 public String getContent() { return content; }  
 public void setContent(String content) { this.content = content; }  
 public String getOriginalMessageId() { return originalMessageId; }  
 public void setOriginalMessageId(String originalMessageId) { this.originalMessageId = originalMessageId; }  
 public List<String> getMentionedUserIds() { return mentionedUserIds; }  
 public void setMentionedUserIds(List<String> mentionedUserIds) { this.mentionedUserIds = mentionedUserIds; }  
}  
  
  
// --- Response DTO ---  
  
/\*\*  
 \* 메시지 전송 응답 DTO (성공 시 data 필드에 포함될 내용)  
 \*/  
public class SendMessageResponseDto {  
 private String messageId; // 생성된 메시지 ID  
 private String roomId; // 채팅방 ID  
 private String senderId; // 보낸 사람 ID  
 private String content; // 메시지 내용  
 private LocalDateTime timestamp; // 메시지 전송 시각 (서버 기준)  
 private String originalMessageId; // 답장한 경우 원본 메시지 ID  
 private List<String> mentionedUserIds; // 멘션된 사용자 ID 목록  
  
 // Constructor, Getters or Lombok @Data  
 public SendMessageResponseDto(String messageId, String roomId, String senderId, String content, LocalDateTime timestamp, String originalMessageId, List<String> mentionedUserIds) {  
 this.messageId = messageId;  
 this.roomId = roomId;  
 this.senderId = senderId;  
 this.content = content;  
 this.timestamp = timestamp;  
 this.originalMessageId = originalMessageId;  
 this.mentionedUserIds = mentionedUserIds;  
 }  
 // Getters...  
 public String getMessageId() { return messageId; }  
 public String getRoomId() { return roomId; }  
 public String getSenderId() { return senderId; }  
 public String getContent() { return content; }  
 public LocalDateTime getTimestamp() { return timestamp; }  
 public String getOriginalMessageId() { return originalMessageId; }  
 public List<String> getMentionedUserIds() { return mentionedUserIds; }  
}  
  
  
// --- 공통 API 응답 DTO ---  
  
/\*\*  
 \* 표준 API 응답 형식 DTO  
 \* @param <T> 응답 데이터의 타입  
 \*/  
public class ApiResponse<T> {  
 private int status; // HTTP 상태 코드와 일치 또는 내부 정의 코드  
 private String message; // 응답 메시지 (성공/실패 등)  
 private T data; // 실제 응답 데이터 (성공 시)  
  
 // Public constructor (필요에 따라 private + static factory 유지 가능)  
 public ApiResponse(int status, String message, T data) {  
 this.status = status;  
 this.message = message;  
 this.data = data;  
 }  
  
 /\*\*  
 \* 성공 응답 생성 (기본 200 OK)  
 \* @param data 응답 데이터  
 \* @return ApiResponse 객체  
 \*/  
 public static <T> ApiResponse<T> success(T data) {  
 return new ApiResponse<>(HttpStatus.OK.value(), "Success", data);  
 }  
  
 /\*\*  
 \* 성공 응답 생성 (메시지 커스텀, 기본 200 OK)  
 \* @param data 응답 데이터  
 \* @param message 성공 메시지  
 \* @return ApiResponse 객체  
 \*/  
 public static <T> ApiResponse<T> success(T data, String message) {  
 return new ApiResponse<>(HttpStatus.OK.value(), message, data);  
 }  
  
 /\*\*  
 \* 실패 응답 생성  
 \* @param status HTTP 상태 코드  
 \* @param message 실패 메시지  
 \* @return ApiResponse 객체 (데이터는 null)  
 \*/  
 public static <T> ApiResponse<T> error(HttpStatus status, String message) {  
 return new ApiResponse<>(status.value(), message, null);  
 }  
  
 // Getters or Lombok @Getter  
 public int getStatus() { return status; }  
 public String getMessage() { return message; }  
 public T getData() { return data; }  
}