공유 텍스트 편집 시스템 7팀

1. 프로젝트 개요

프로젝트 명  
: 공유 텍스트 편집 시스템 (Shared Editing System)

목표  
: 여러 사용자가 각각 다른 통신 환경에서도 실시간으로 텍스트를 함께 편집할 수 있도록 지원하는 분산 시스템 기반 협업 편집 플랫폼을 구축한다.

개발 동기  
: 단순한 중앙 집중형 서버가 아닌, 효율적인 이벤트 전송과 데이터 동기화를 통해 실제 사용 가능한 수준의 "공유 편집" 시스템을 만드는 것을 목표로 한다.

핵심 특징

* 클라이언트-서버 구조 (Event-driven architecture)
* 실시간 텍스트 편집 및 동기화
* 늦게 접속한 사용자도 현재 편집 내용을 즉시 불러오기 지원 (Late-comer support)
* 저장 및 불러오기 기능을 통한 지속성 유지
* 동시 편집 충돌 최소화 및 사용자 상태 표시
* 문서 삭제 및 수정 내역 관리

2. 개발 환경

개발 언어  
: Java 17

개발 도구 (IDE)  
: IntelliJ IDEA 2023.2

통신 및 네트워킹 라이브러리  
: KU CCSLAB의 CM (Communication Manager) Framework 사용

버전 관리 시스템  
: GitHub (Private Repository 운영, 협업자(ccslab) 추가)

GITHUB LINK : <https://github.com/DistributedSystemTeam15/Team15>

구성요소

* 서버 프로그램 (CMServerApp, CMServerEventHandler)
* 클라이언트 프로그램 (CMClientApp, CMClientEventHandler, MainFrame 등)
* GUI (Swing 기반)
* 유틸리티 클래스 (DialogUtil, DocumentMeta, ServerConfig 등)

2.1 기능적 요구사항

* 사용자 인증 및 세션 관리
  + 사용자는 고유 ID로 서버에 로그인할 수 있다.
  + 서버는 동일 ID로 중복 로그인 시도를 감지하고 차단한다.
  + 사용자는 자유롭게 접속 및 로그아웃할 수 있다.
* 문서 관리
  + 사용자는 새로운 텍스트 문서를 생성할 수 있다.
  + 사용자는 기존 문서를 선택하여 열 수 있다.
  + 사용자는 편집 중인 문서를 저장할 수 있다.
  + 사용자는 문서를 삭제할 수 있으며, 삭제 시 관련 클라이언트에 즉시 반영된다.
* 실시간 협업 편집
  + 여러 사용자가 동시에 같은 문서를 편집할 수 있다.
  + 모든 편집 변경사항은 실시간으로 서버를 거쳐 참여 클라이언트에게 브로드캐스트된다.
  + 사용자의 입력/삭제/붙여넣기 등 텍스트 변경사항이 모두 반영된다.
* 동기화 및 일관성 보장
  + 모든 클라이언트는 동일한 문서 상태를 유지한다.
  + 네트워크 지연 발생 시에도 최종 일관성(Eventual Consistency)을 보장한다.
* 늦게 접속한 사용자 동기화
  + 편집 도중 새로 접속한 사용자는 현재 최신 문서 상태를 즉시 받아 편집에 합류할 수 있다.
* 문서 저장 및 복원
  + 편집 중인 문서는 서버가 주기적으로 디스크에 저장하여 장애 발생 시 복원할 수 있다.
* 편집 상태 알림
  + 상단 패널을 통해 현재 열려 있는 문서명과 편집 참여자 목록이 실시간으로 표시된다.
* 접속자 관리
  + 실시간 온라인 사용자 목록이 클라이언트에 표시된다.
  + 사용자 접속 및 퇴장 이벤트가 자동 반영된다.

2.2 비기능적 요구사항

* 성능
  + 편집 동기화는 네트워크 지연 최대 100ms 이내로 전파되어야 한다.
  + 최소 3명 이상의 동시 편집자 환경을 안정적으로 지원해야 한다.
* 확장성
  + 코드 설계는 기능 추가(예: 채팅, Undo/Redo, 문서 버전 관리 등)를 고려해 모듈화되어야 한다.
* 신뢰성
  + 서버 장애 발생 시에도 저장된 문서 파일을 기반으로 복구할 수 있어야 한다.
  + 클라이언트 비정상 종료에도 서버 상태는 유지된다.
* 보안성
  + 현재 로그인 과정은 최소한의 검증(ID 중복 방지)만 적용되어 있으며, 향후 비밀번호 암호화와 같은 추가 보안 조치를 고려한다.
* 사용성
  + 모든 에러 상황(로그인 실패, 서버 연결 끊김, 문서 삭제됨 등)에 대해 명확한 메시지와 UI 안내가 제공되어야 한다.
* 장애 허용성
  + 클라이언트가 비정상 종료되더라도 서버는 세션을 안정적으로 유지하고, 다른 사용자에게 영향을 주지 않아야 한다.

4. 시스템 아키텍처 및 주요 구성

전체 구조

* Server
  + 클라이언트와의 통신을 담당하는 서버 스텁 (CMServerStub)
  + 클라이언트 요청을 처리하는 이벤트 핸들러 (CMServerEventHandler)
  + 문서 정보 (내용, 메타데이터)를 메모리와 파일 시스템에 저장 및 관리
* Client
  + 서버와 연결 및 이벤트 송수신을 담당하는 클라이언트 스텁 (CMClientStub)
  + 서버로부터 받은 이벤트를 처리하는 이벤트 핸들러 (CMClientEventHandler)
  + 문서 편집 및 목록 관리를 위한 GUI (MainFrame, DocumentEditScreen, DocumentListScreen)
  + 로그인 및 서버 연결을 위한 대화 상자 (LoginDialog, ConnectionDialog)

핵심 통신 방식

* CM Framework의 사용자 정의 이벤트(CMUserEvent) 를 통한 메시지 송수신
* 주요 이벤트: CREATE\_DOC, EDIT\_DOC, SAVE\_DOC, DELETE\_DOC, LIST\_DOCS, DOC\_CONTENT, USER\_LIST, LOGIN\_ACCEPTED, LOGIN\_REJECTED\_DUPLICATE 등

문서 동기화 흐름

* 편집된 내용은 서버로 실시간 전송되고, 서버는 이를 다른 클라이언트에 브로드캐스팅하여 동기화
* 서버는 모든 문서를 메모리와 파일 시스템에 이중 저장하여 장애 복구를 지원

5. 작동 원리

(1) 서버 시작

* CMServerApp 클래스 실행
  + 내부적으로 CMServerStub 객체를 초기화하고 startCM() 호출 → 서버가 대기 상태로 전환
  + 서버는 클라이언트의 접속 요청 및 사용자 이벤트를 수신할 준비 완료

(2) 클라이언트 접속

* CMClientApp 클래스 실행
  + ConnectionDialog를 통해 서버 IP, 포트 입력
  + 서버에 연결 성공 시, LoginDialog를 통해 사용자 ID, 비밀번호 입력
  + loginCM() 호출 → 서버로 로그인 요청
  + 서버 (CMServerEventHandler)는 중복 로그인 여부 검사 후 LOGIN\_ACCEPTED 또는 LOGIN\_REJECTED\_DUPLICATE 이벤트 전송
  + 로그인 성공 시 MainFrame (GUI) 오픈

(3) 문서 목록 조회 및 문서 열기

* 클라이언트는 서버에 LIST\_DOCS 이벤트 전송
* 서버는 현재 보유 중인 문서들의 메타데이터를 JSON 배열로 반환
* 클라이언트는 이를 파싱해 문서 목록 리스트를 화면에 출력
* 사용자가 문서를 선택하면 SELECT\_DOC 이벤트 전송 → 서버는 문서 내용을 클라이언트로 전송

(4) 실시간 편집 및 동기화

* 클라이언트가 텍스트를 수정하면 EDIT\_DOC 이벤트를 서버로 전송
* 서버는 해당 문서에 참여 중인 다른 클라이언트들에게 편집된 내용을 전송하여 즉시 동기화

(5) 문서 저장 및 불러오기

* 사용자가 저장을 요청하면 (SAVE\_DOC 이벤트)
* 서버는 현재 문서 내용을 documents/ 폴더에 파일로 저장 (.txt 파일)

(6) 문서 삭제

* 사용자가 문서 삭제 요청 (DELETE\_DOC 이벤트)
* 서버는 메모리와 파일 시스템에서 해당 문서를 삭제
* 현재 해당 문서를 열고 있던 다른 클라이언트에게 DOC\_CLOSED 이벤트를 전송해 편집 화면을 초기화

6. 분산 데이터 처리 방식

* 서버 중심 분산 모델 채택
  + 모든 데이터(문서 내용, 메타데이터)는 서버가 관리
  + 클라이언트는 편집 요청만 보내고, 데이터 저장은 서버가 담당
* 실시간 브로드캐스트
  + 서버가 편집 내용을 다른 참여자들에게 실시간으로 전송
* 중복 로그인 방지
  + 서버가 로그인 사용자 목록을 관리하고 중복 ID 접속을 거부
* Late-comer 지원
  + 나중에 접속하는 사용자는 현재까지의 문서 내용을 서버로부터 수신
* 장애 복구를 위한 파일 저장
  + 문서는 메모리에만 저장하지 않고 파일 시스템에도 저장

7. 클래스 및 메소드별 상세 동작 설명

7.1 서버 측

7.1.1 CMServerApp  
서버 프로그램의 진입점 클래스. 서버 스텁과 이벤트 핸들러를 초기화하고 서버를 구동한다.

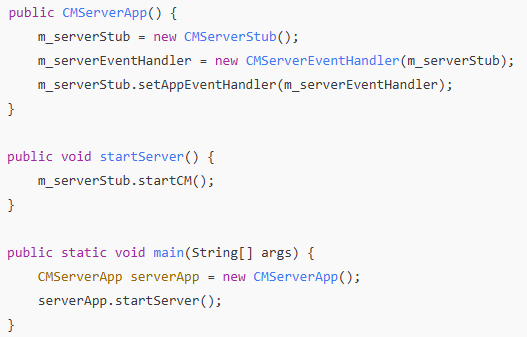


Figure 1

동작 원리 설명:

* CMServerApp() 생성자
  + CMServerStub 객체를 생성하여 서버 소켓을 초기화하고
  + CMServerEventHandler를 생성해 이벤트 수신을 준비한다.
  + 서버 스텁에 이벤트 핸들러를 연결하여 모든 서버 이벤트를 CMServerEventHandler가 처리하게 한다.
* startServer()
  + startCM() 호출 → 서버 포트 리스닝 시작, 클라이언트 연결 대기 상태로 진입.
* main()
  + 프로그램 진입점. 위 과정을 순차적으로 호출하여 서버 구동.

예시 흐름:

1. 운영자가 CMServerApp 실행
2. 서버가 포트 7777(기본값)에서 클라이언트 연결 대기
3. 클라이언트가 접속하면 서버가 로그인 이벤트를 수신하고 핸들러로 전달

7.1.2 CMServerEventHandler

서버 측에서 발생하는 모든 클라이언트 이벤트를 수신하고, 문서 생성, 편집, 저장, 삭제 등 공유 편집 시스템의 모든 핵심 로직을 처리하는 클래스이다.

주요 멤버 변수

documents: 문서 이름 ➔ 문서 내용

docUsers: 문서 이름 ➔ 편집 중인 사용자 리스트

userCurrentDoc: 사용자 ID ➔ 열고 있는 문서 이름

onlineUsers: 현재 접속한 사용자 집합

docMeta: 문서 이름 ➔ 메타데이터 (생성자, 최종 수정자, 시간 등)

문서 상태와 사용자 세션을 서버가 직접 메모리로 관리한다.

✨ 핵심 메소드: processEvent(CMEvent cme)



Figure 2

① 서버는 모든 클라이언트 이벤트를 processEvent()에서 수신.

② 이벤트 타입이 SESSION\_EVENT면 로그인/로그아웃 처리.

③ 이벤트 타입이 USER\_EVENT면 문서 작업 (CREATE\_DOC, EDIT\_DOC, 등) 수행.

④ 각 작업별로 handleXXX 메소드로 세부 로직을 위임.

각 이벤트별 동작 요약

|  |  |  |
| --- | --- | --- |
| 이벤트ID | 처리 메소드 | 동작 설명 |
| CREATE\_DOC | handleCreateDoc() | 새 문서 생성, 메타데이터 등록, 전체 사용자에게 문서 리스트 갱신 전송 |
| EDIT\_DOC | handleEditDoc() | 문서 편집 내용 업데이트, 편집 사용자 외 모든 사용자에게 변경 내용 브로드캐스트 |
| SAVE\_DOC | handleSaveDoc() | 현재 문서 내용을 디스크에 저장 (documents/ 폴더) |
| DELETE\_DOC | handleDeleteDoc() | 문서 삭제, 편집 참여자에게 DOC\_CLOSED 이벤트 전송 |
| LIST\_DOCS | handleListDocs() | 현재 서버에 존재하는 문서 목록 JSON 전송 |
| SELECT\_DOC | handleSelectDoc() | 특정 문서 선택 → 문서 전체 내용을 해당 클라이언트로 전송 |
| USER\_LIST | handleUserList() | 특정 문서의 참여자 리스트 조회 및 반환 |

예시 흐름 (코드 기반)

문서 편집 흐름 예시:

* 사용자 A가 편집 → EDIT\_DOC 이벤트 발생
* handleEditDoc() 메소드 호출
* 서버 documents 맵에서 해당 문서 내용 수정
* docUsers 맵을 참조하여 현재 문서를 열고 있는 모든 사용자 파악
* 변경된 내용을 나(A) 제외 다른 사용자들에게 브로드캐스트
* 다른 사용자 B, C는 서버로부터 TEXT\_UPDATE 이벤트를 수신하고 편집 내용 업데이트

저장 예시:

* 클라이언트가 저장 요청 → SAVE\_DOC 이벤트 전송
* handleSaveDoc() 호출
* 현재 문서 내용을 documents/ 폴더에 .txt 파일로 저장
* 문서 메타데이터(docMeta)의 수정 시간을 갱신
* 저장 완료 후 클라이언트에게 저장 완료 표시 (markSaved())

[클라이언트 측]

7.1.3 CMClientApp

클라이언트 프로그램의 진입점. 서버와 연결을 설정하고, 로그인 절차를 진행하고, 메인 화면(MainFrame)을 띄운다.  
또한 문서 생성, 저장, 삭제와 같은 클라이언트 측 주요 작업들을 서버로 요청하는 기능을 제공한다.

**✨ 핵심 소스코드 발췌**

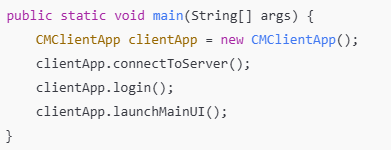


Figure 3

**✨ 주요 메소드**

|  |  |
| --- | --- |
| **메소드** | **설명** |
| connectToServer() | 서버 IP/포트 입력 → 서버 연결 |
| login() | 사용자 ID/비밀번호 입력 → 로그인 요청 |
| launchMainUI() | MainFrame 생성 및 표시 |
| createNewDocument(String docName) | 새 문서 생성 요청 |
| sendTextUpdate(String content) | 편집한 텍스트 서버로 전송 |
| saveDocument() | 서버에 저장 요청 (SAVE\_DOC 이벤트) |
| deleteDocument(String docName) | 서버에 문서 삭제 요청 (DELETE\_DOC 이벤트) |
| openDocument() | 서버에 문서 목록 요청 (LIST\_DOCS 이벤트) |
| notifyModified(boolean modified) | 문서 수정 상태(저장 여부)에 따라 상단 제목 업데이트 |

**✨ 주요 플로우**

🛠 **초기화 플로우 (main)**

1. connectToServer()
   * ConnectionDialog를 통해 서버 IP, 포트 입력
   * CMClientStub의 connectToServer() 호출하여 서버 연결
2. login()
   * LoginDialog를 통해 ID/비밀번호 입력
   * loginCM() 호출하여 서버로 로그인 요청
   * 서버로부터 LOGIN\_ACCEPTED 수신 시 로그인 성공
3. launchMainUI()
   * MainFrame 인스턴스 생성
   * 초기 화면 띄우기 (문서 리스트 표시 전)

🛠 **편집 플로우**

1. 사용자가 문서 생성 요청 (New Document)  
   → createNewDocument(docName) 호출 → 서버로 CREATE\_DOC 이벤트 전송
2. 사용자가 텍스트 편집  
   → sendTextUpdate(content) 호출 → 서버로 EDIT\_DOC 이벤트 전송
3. 사용자가 저장 요청  
   → saveDocument() 호출 → 서버로 SAVE\_DOC 이벤트 전송
4. 사용자가 삭제 요청  
   → deleteDocument(docName) 호출 → 서버로 DELETE\_DOC 이벤트 전송

**✨ 핵심 동작 요약**

|  |  |
| --- | --- |
| **기능** | **설명** |
| 서버 연결 | connectToServer() → ConnectionDialog 입력 기반 연결 |
| 사용자 로그인 | login() → ID/비밀번호 입력 → 서버로 로그인 요청 |
| 메인 UI 띄우기 | launchMainUI() → 문서 목록 및 편집기 화면 띄움 |
| 편집 내용 서버 전송 | sendTextUpdate() → 실시간 동기화 지원 |
| 저장 요청 | saveDocument() 호출 → 서버 저장 |
| 문서 삭제 요청 | deleteDocument() 호출 → 서버 문서 삭제 |

**7.1.4 CMClientEventHandler**

**✨ 클래스 목적**

클라이언트 측 이벤트 핸들러. 서버로부터 수신하는 모든 이벤트 (TEXT\_UPDATE, DOC\_CONTENT, USER\_LIST 등)를 처리하여,  
클라이언트 UI (MainFrame)에 반영한다.

**✨ 핵심 소스코드 발췌**



Figure 4

**✨ 주요 메소드**

|  |  |
| --- | --- |
| **메소드** | **설명** |
| updateOnlineList(CMUserEvent) | 서버로부터 수신한 온라인 사용자 목록 갱신 |
| openDocumentContent(CMUserEvent) | 선택한 문서 내용 받아서 편집창 표시 |
| updateTextContent(CMUserEvent) | 다른 사용자의 편집 내용을 반영 |
| updateDocumentList(CMUserEvent) | 서버로부터 문서 목록 수신 및 UI 갱신 |
| updateUserList(CMUserEvent) | 문서별 접속 사용자 목록 업데이트 |
| handleDocumentClosed(CMUserEvent) | 삭제된 문서 알림 수신 후 편집창 초기화 |
| handleLoginAccepted() | 로그인 성공 후 초기화 |
| handleLoginRejected() | 중복 로그인 거부 처리 |

**✨ 주요 플로우**

🛠 **문서 선택 흐름 예시**

1. 서버로부터 DOC\_CONTENT 수신
2. openDocumentContent() 호출
   * 현재 편집기를 초기화하고
   * 서버로부터 받은 문서 내용으로 채운다
   * 편집 가능 상태로 전환한다
   * 저장 버튼 활성화

🛠 **편집 반영 흐름 예시**

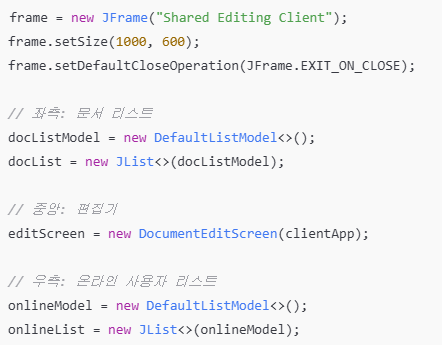
1. 서버로부터 TEXT\_UPDATE 수신
2. updateTextContent() 호출
   * 현재 편집창 내용 업데이트
   * (자기 자신이 보낸 내용은 반영하지 않음)

7.2 [GUI 및 유틸리티]

7.2.1 MainFrame

클라이언트 프로그램의 **메인 화면**을 구성한다. 문서 목록, 편집 창, 접속자 리스트, 상단 메뉴 등을 관리한다. 사용자의 주요 인터랙션(문서 열기, 저장, 삭제)을 담당하는 중앙 UI 클래스다.

**✨ 핵심 소스코드 발췌**



**✨ 주요 구성 요소**

|  |  |
| --- | --- |
| **구성 요소** | **설명** |
| JList<DocumentMeta> 문서 목록 | 서버에 존재하는 문서 리스트를 표시한다. |
| JTextArea 편집창 (DocumentEditScreen) | 실제 텍스트 편집이 이루어지는 공간이다. |
| JList<String> 온라인 사용자 목록 | 현재 서버에 접속 중인 사용자들을 표시한다. |
| 메뉴바 (File 메뉴) | 새 문서 생성, 저장 버튼 |
| 상단 툴바 | 저장 버튼 (💾) 배치 |
| 상단 패널 | 현재 문서명과 편집 참여자 목록 표시 |

**✨ 주요 메소드**

|  |  |
| --- | --- |
| **메소드** | **설명** |
| addNewDocumentAction(listener) | 새 문서 버튼 클릭 시 서버로 생성 요청 |
| addSaveDocumentAction(listener) | 저장 버튼 클릭 시 서버로 저장 요청 |
| setDocumentList(List<DocumentMeta>) | 서버로부터 수신한 문서 목록을 갱신 |
| updateTextContent(content) | 서버로부터 받은 문서 내용을 편집창에 표시 |
| updateDocumentUsers(docName, users) | 특정 문서의 편집 참여자 목록 업데이트 |
| markDocumentModified() | 문서가 수정된 경우 상단 제목에 \* 표시 |
| markDocumentSaved() | 문서 저장 시 \* 제거 |
| setCurrentDocument(name) | 상단에 현재 편집 중인 문서 제목 표시 |
| setCurrentDocumentUsers(users) | 상단에 현재 편집 참여자 표시 |

**✨ 주요 플로우**

🛠 **문서 열기 플로우**

1. 사용자가 문서 리스트를 더블클릭
2. SELECT\_DOC 이벤트 서버로 전송
3. 서버가 문서 내용을 전송
4. updateTextContent(content) 호출 → 편집 창에 문서 내용 로딩

🛠 **편집 플로우**

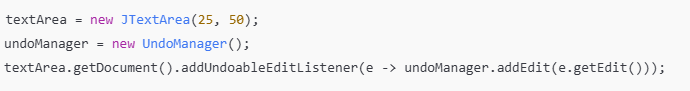
1. 사용자가 텍스트 편집
2. DocumentEditScreen이 자동으로 sendTextUpdate(content) 호출
3. 서버로 실시간 편집 내용 전송

🛠 **저장 플로우**

1. 사용자가 Ctrl+S 누르거나 저장 버튼 클릭
2. saveDocument() 호출 → 서버로 SAVE\_DOC 요청
3. 서버 저장 후 UI에서 markDocumentSaved() 호출 → \* 표시 제거
   * 1. **DocumentEditScreen**

텍스트 편집 기능을 담당하는 **편집 창** 컴포넌트. JTextArea를 중심으로 Undo/Redo 기능과 서버 전송 로직을 구현했다.

**✨ 핵심 소스코드 발췌**



**✨ 주요 기능**

|  |  |
| --- | --- |
| **기능** | **설명** |
| 텍스트 입력 감지 | 입력/삭제 발생 시 서버로 편집 내용 전송 (sendTextUpdate()) |
| Undo/Redo 지원 | Ctrl+Z를 통해 Undo 기능 제공 (Redo는 추가 가능성 있음) |
| 편집 모드 활성화 | 문서 선택 또는 새 문서 생성 시 편집 가능 상태로 전환 |
| 편집 무시 모드 (ignoreDocEvents) | 서버로 받은 업데이트 반영 시 이벤트 발생 방지 |

**✨ 주요 메소드**

|  |  |
| --- | --- |
| **메소드** | **설명** |
| updateTextContent(content) | 서버로부터 수신한 문서 내용으로 편집창 업데이트 |
| resetDocumentView() | 편집창을 초기화 (빈 화면 + 편집 불가) |
| markSaved() | 저장 상태로 전환하여 내부 플래그 초기화 |

**✨ 주요 플로우**

🛠 **텍스트 입력 감지 플로우**

1. 사용자가 키 입력
2. DocumentListener가 insertUpdate() 호출
3. sendTextUpdate(content) 실행 → 서버로 편집 내용 전송

🛠 **서버로부터 문서 업데이트 수신 플로우**

1. 서버가 DOC\_CONTENT 이벤트 전송
2. updateTextContent(content) 호출
3. ignoreDocEvents 플래그를 켜서 입력 이벤트 무시
4. 텍스트 업데이트 후 다시 이벤트 허용
   * 1. **DocumentListScreen**

문서 목록 선택 및 삭제 대화창을 제공하는 유틸리티 클래스. 사용자 입력을 받아 어떤 문서를 열거나 삭제할지 결정하게 한다.

**✨ 주요 메소드**

|  |  |
| --- | --- |
| **메소드** | **설명** |
| promptDocumentSelection(docNames, parent) | 문서 목록 중 하나를 선택받는다. |
| promptDocumentDeletion(docNames, parent) | 삭제할 문서를 선택받는다. |
| confirmDocumentDeletion(docName, parent) | 문서 삭제 여부를 확인받는다. |
| showNoDocumentsAvailable(parent) | 문서가 없을 때 알림창 표시 |
| showUserList(docName, users, parent) | 특정 문서의 참여자 리스트 표시 |

**✨ 주요 플로우**

🛠 **문서 열기 플로우**

1. 서버에서 문서 목록 수신
2. 사용자에게 선택 팝업 표시
3. 사용자가 문서를 선택하면 해당 이름을 반환

🛠 **문서 삭제 플로우**

1. 문서 리스트에서 삭제할 문서를 선택
2. 사용자가 삭제 확인 창에서 Yes를 누르면 서버로 삭제 요청 전송

**6.2.4 ClientUIController**

MainFrame에서 발생하는 사용자 액션을 서버 요청으로 연결하는 역할.

**✨ 주요 메소드**

|  |  |
| --- | --- |
| **메소드** | **설명** |
| 등록된 ActionListener를 통해 new document 클릭 시 서버에 새 문서 요청 |  |
| save document 클릭 시 서버에 저장 요청 |  |

**✨ 플로우**

* New Document 클릭 → 새 문서 생성 이벤트 발송
* Save 클릭 → 저장 요청 이벤트 발송

(버튼에 리스너를 등록하고, 내부적으로 CMClientApp 메소드를 호출하게 연결해준다.)

7.8 ClientUIController

* 클래스 목적: MainFrame의 버튼 액션을 서버 요청으로 연결
* 주요 메소드:
  + 새 문서 생성, 문서 저장 액션 등록

7.9 DialogUtil

* 클래스 목적: 팝업 메시지 유틸리티
* 주요 메소드:
  + 오류, 알림, 확인창 표시

7.10 ConnectionDialog, LoginDialog

* 클래스 목적: 서버 연결 설정 입력 / 로그인 입력
* 주요 메소드:
  + getServerConfig(): 서버 IP/포트 입력
  + showLoginDialog(): 로그인 ID/비밀번호 입력

7.11 DocumentMeta

* 클래스 목적: 문서에 대한 메타데이터 저장
* 주요 필드:
  + 이름, 생성자, 수정자, 생성시간, 수정시간, 현재 편집 참여자 리스트

8. 동작 시나리오 예시

(1) 서버 실행 및 초기화

* 서버 운영자가 CMServerApp을 실행.
* CMServerStub이 기동되어 클라이언트 접속을 대기.
* 파일 시스템(documents/ 폴더) 내 기존 문서를 스캔하여 관리 준비 완료.

(2) 클라이언트 접속 및 로그인

* 클라이언트 사용자는 프로그램을 실행하여 ConnectionDialog로 서버 IP/포트 입력.
* LoginDialog에서 ID/비밀번호 입력.
* 서버로 로그인 요청 전송 → 서버는 중복 접속 여부 검사 후 로그인 허용.

(3) 문서 목록 조회 및 문서 열기

* 로그인 후 클라이언트는 LIST\_DOCS 이벤트 전송.
* 서버는 현재 존재하는 문서들의 목록을 JSON 형태로 응답.
* 사용자는 DocumentListScreen에서 문서를 선택하거나 새 문서를 생성할 수 있음.

(4) 새 문서 생성

* 사용자가 메뉴 또는 버튼으로 "New Document" 선택.
* 문서 이름 입력 후 서버로 CREATE\_DOC 이벤트 전송.
* 서버는 새 문서를 메모리에 등록하고, 모든 클라이언트에 문서 목록 갱신을 통보.

(5) 문서 편집

* 사용자가 텍스트를 입력하거나 수정하면, EDIT\_DOC 이벤트가 서버로 전송.
* 서버는 해당 문서에 참여 중인 모든 다른 클라이언트에게 새로운 내용을 전송하여 실시간 동기화.

(6) 문서 저장

* 사용자가 Ctrl+S를 누르거나 저장 버튼을 클릭.
* 클라이언트가 SAVE\_DOC 이벤트를 서버로 전송.
* 서버는 현재 문서 내용을 파일 시스템에 저장하고 수정 시간을 갱신.

(7) 문서 삭제

* 사용자가 문서 삭제를 요청 (DELETE\_DOC 이벤트).
* 서버는 in-memory 및 파일 시스템 양쪽에서 문서를 삭제.
* 해당 문서를 열고 있던 다른 사용자들에게 DOC\_CLOSED 이벤트를 보내 편집 화면을 초기화.

9. 예외 상황 및 예외 처리

| 상황 | 처리 방법 |
| --- | --- |
| 중복 로그인 시도 | 서버가 LOGIN\_REJECTED\_DUPLICATE 이벤트 전송 및 차단 |
| 문서가 존재하지 않는 경우 선택 시도 | 서버가 문서 로드 실패 시 빈 문서로 초기화하여 대응 |
| 삭제된 문서를 편집하려 할 경우 | DOC\_CLOSED 이벤트 수신 후 편집 화면 초기화 |
| 서버 연결 실패 | 연결 재시도 안내 또는 프로그램 종료 |
| 저장 실패 (디스크 에러 등) | 서버 콘솔에 오류 출력, 클라이언트에는 영향 없음 |

10. 테스트 및 검증

테스트 환경

* Windows 11, Java 17 환경
* 동일 네트워크 상 2대 이상의 클라이언트 접속 테스트

주요 테스트 항목

* 서버 다중 클라이언트 접속 및 로그인 테스트
* 새 문서 생성 후 편집 및 저장 테스트
* 문서 편집 충돌 테스트 (동시 편집 시 서버 수용 여부)
* 문서 삭제 후 다른 클라이언트 동작 반응 테스트
* 서버 중단/재기동 후 문서 복원 테스트

검증 결과

* 실시간 편집 반영
* 문서 저장 및 복원
* 중복 로그인 차단
* 삭제된 문서 자동 초기화

11. 향후 개선 방향

| 개선 방향 | 설명 |
| --- | --- |
| 구역별 락 기반 편집 | 동일 문서 내에서 특정 범위만 편집권한 부여 (Conflict 감소) |