2022

프로젝트 요구사항 분석 및 설계서

팀 FLOOR

백 재웅, 백 현웅

목차(Index)

        - (#9) Document: Auto-Refresh
        - (#10) Chunk: autocomplete
        - (#11) Chunk: swap positions
        - (#12) Document: Share
        - (#13) Document: Navigator
        - (#14) File: create/delete/rename file
        - (#15) File: upload/download files
        - (#16) Search: Document Search
        - (#17) Stash: render
        - (#18) File: export document
        - (#19) Stash: add
        - (#20) Stash: remove
        - (#21) Stash: Drag and Drop to Document
        - (#22) Management: Login
        - (#23) Management: Configure
        - (#24) Management: Localization
        - (#25) Management: Theme
        - (#27) Chunk: edit
        - (#28) Extension: API
        - (#29) Extension: Plugin
        - (#30) Search: Find word
    3. 성능 요구사항(Performance requirements)
    4. 논리적 데이터베이스 요구사항(Logical database requirements)
    5. 설계 제약사항(Design constraints)
        1. 표준 준수(Standards compliance)
    6. 소프트웨어 시스템 속성(Software system attributes)
    7. 상세 요구사항의 구성(Organizing the specific requirements)
        1. 객체(Objects)
        2. 사용자 인터페이스 상세

4. 요구사항 명세 추가 이력 (Requirement Specification Supporting Information)

    1. 부록(Appendixes)
    2. 개발 환경(Development Environment)
    3. 일정표(Schedule)

5. 설계(Architecture)

    1. UML
        1. Server Side UML
        2. Client Side UML
    2. 의사코드(Pseudo Code)
        1. 서버 RPC 메세지 처리

        2. 클라이언트의 메세지 처리 동기화
        3. 다른 작업들
    3. 동기화 정책(Synchronization Policy)

6. 시험(Testing)

    1. 유닛 테스트(Unit test)
    2. 기능 테스트(Functional Test)

1. 소개(Introduction)

Version : 1.1.0

Version Hash : ef10264

본 문서는 전북대학교 컴퓨터공학과의 Floor 팀에서 Scrap Yard라는 어플리케이션을 설계 및 구현하기 위한 소프트웨어 요구사항 명세서(SRS)이자 어플리케이션의 구조를 나타내는 설계서이자 어플리케이션의 시험 결과 보고서이다. 정리하면 어플리케이션을 개발하면서 발생하는 산출물들을 정리한 문서이다.

1.1. 목적(Purpose)

본 문서의 목적은 첫째로 프로젝트의 관련된 모든 아이디어들을 정리하고 분석해서 나열하는 것이다. 또한 프로젝트를 더 잘 이해하기 위해 이 제품이 어떻게 사용될지 예측하고 분류하고, 나중에 개발될 요소를 설명하고, 고려 중이지만 폐기될 수 있는 요구사항들을 문서화한다. 둘째로 이러한 요구사항을 해결하기 위해 만들어진 설계를 나열하고 문서화하는 것이다. 셋째로 이러한 설계로 구현된 프로그램을 시험을 하고 그 결과를 정리해서 보기좋게 문서화하고 색인하는 것이다.

1.2. 범위(Scope)

본 문서의 범위는 ScrapYard의 기능들과 그 환경 그리고 상세 설계 및 인터페이스이다.

ScrapYard는 문서 작성 및 문서를 아카이빙 할 수 있는 웹 어플리케이션이다. 같이 제공되는 확장기능을 통해 북마크(즐겨찾기)를 구조적으로 보관할 수 있고 미리보기를 보여줄 수 있다.

또한 문서를 다른 사람과 링크로 공유할 수 있다.

개인정보의 관리를 자기 자신이 제어할 수 있도록 파일과 문서에 대한 메타데이터가 어떠한 외부 DB가 있는 것이 아닌 파일에서 사람이 읽을 수 있는 형태로 관리된다.

1.3. 용어 및 약어 정의(Definitions, acronyms and abbreviations)

용어 및 약어	정의
ScrapYard	현재 개발하는 앱의 명칭
DnD	드래그 앤 드롭의 약자

1.4. 참고자료(References)

- repository
- document repository
- react
- recoil
- MUI
- dndkit
- markdown

1.5. 개요(Overview)

2장과 3장은 SRS(소프트웨어 요구사항 명세서)의 양식에 맞추어 작성되었다. 2장에서는 종합적인 요구사항을 서술하고, 3장에서는 기능 및 UI에 대해서 상세한 요구사항을 설명한다. 4장은 SRS의 추가 이력사항에 대해서 서술한다. 여기에서는 어플리케이션 개발 일정표가 포함되어 있다. 5장은 어플리케이션의 상세한 설계에 대해서 서술한다. 6장은 개발된 어플리케이션의 시험과 그 결과에 대해서 서술한다.

2. 전체 시스템 개요(Overall description)

2.1. 제품 관점(Product perspective)

2.1.1. 시스템 인터페이스(System interfaces)

본 시스템은 Cross-platform 소프트웨어이다. 다음과 같은 브라우저가 원활히 실행될 수 있는 시스템에서 동작 할 수 있다.

- Chrome 버전 61 이상
- Firefox 버전 60 이상
- Edge 버전 79 이상
- Safari 버전 11 이상
- Chrome for Android 버전 100 이상
- Samsung internet 버전 8.2 이상

2.1.2. 사용자 인터페이스(User interfaces)

웹으로 동작하는 GUI이다. 키보드와 마우스, 터치 인터페이스로 동작할 수 있다. GUI의 디자인은 Material Design이나 Metro Design 같이 플랫한 디자인을 추구한다.

2.1.3. 하드웨어 인터페이스(Hardware interfaces)

해당되지 않음.

2.1.4. 소프트웨어 인터페이스(Software interfaces)

이 프로젝트의 결과물은 클립보드를 통해서 여러 타입의 데이터를 import/export한다.

2.1.5. 통신 인터페이스(Communications interfaces)

해당되지 않음.

2.1.6. 메모리 제약사항(Memory constraints)

서버는 2GB 메모리 환경에서 정상 작동해야 한다.

2.1.7. 운영(Operations)

해당되지 않음.

2.1.8. 사이트 적용 요건(Site adaption requirements)

해당되지 않음.

2.2. 제품 기능(Product functions)

본 프로젝트의 결과물은 다음과 같은 기능을 수행한다.

2.2.1 Chunk Operation

1. #1 Chunk: Focus/Unfocus
2. #2 Chunk: remove
3. #3 Chunk: render
4. #4 Chunk: previews
5. #10 Chunk: autocomplete
6. #11 Chunk: swap positions
7. #27 Chunk: edit

2.2.2 Document Operation

1. #5 Document: view Chunk
2. #6 Document: remove
3. #7 Document: add/delete tag
4. #8 Document: Drag And Drop Upload
5. #9 Document: Auto-Refresh
6. #12 Document: Share
7. #13 Document: Navigator

2.2.3 File Operation

1. #14 File: create/delete/rename file
2. #15 File: upload/download files
3. #18 File: export document

2.2.4 Search Operation

1. #16 Search: Document Search
2. #30 Search: Find word

2.2.5 Stash Operation

1. #17 Stash: render
2. #19 Stash: add
3. #20 Stash: remove
4. #21 Stash: Drag and Drop to Document

2.2.6 Management Operation

1. #22 Management: Login
2. #23 Management: Configure
3. #24 Management: Localization
4. #25 Management: Theme

2.2.7 Extension Operation

1. #28 Extension: API
2. #29 Extension: Plugin

2.3. 사용자 특성(User characteristics)

사용자는 기본적인 GUI 조작을 할 줄 알며 인터넷 사용을 원활히 할 수 있고 기본적인 영어를 읽고 쓸 줄 알며, markdown을 작성할 수 있는 사용자로 한정한다. 일반적으로 13세 이상 65세 이하의 사람을 사용자로 가정한다.

2.4. 제약사항(Constraints)

- 이 프로젝트는 MIT License로 개발되고 있으므로 라이브러리의 라이센스도 신경을 쓴다.
- XSS 공격에 안전해야 한다.

2.5. 가정 및 의존성(Assumptions and dependencies)

해당되지 않음.

2.6. 단계별 요구사항(Apportioning of requirements)

해당되지 않음.

3. 상세요구사항(Specific Requirements)

3.1. 외부 인터페이스 요구사항(External interface requirements)

해당되지 않음.

3.2. 기능 요구사항(Functional requirements)

(#1) Chunk: Focus/Unfocus

액터: 사용자
시작 조건: Chunk를 편집할 수 있는 권한을 가져야 한다.
목표: 지금 편집하고자 하는 Chunk를 보여준다.

1. 사용자가 Chunk의 영역에 클릭을 했을때, Focus 된다. 그때 다른 Chunk의 Focus를 사라지게 한다.
2. Focus를 얻었을때, Focus를 얻은 Chunk을 눈에 띄이도록 표시한다.
3. Focus가 사라졌을때, 변경되었으면 변경된 Chunk를 저장한다.

(#2) Chunk: remove

액터: 사용자
시작 조건: Chunk를 수정가능한 권한을 가지고 있어야함.
목표: Chunk를 지운다.

1. Chunk의 좌측 상단의 Context Menu에 삭제 아이콘을 클릭할 때나 빈 내용의 Chunk에서 `Backspace`나 `Del`를 입력할 때 시작한다.
2. 해당 Chunk를 삭제한다.
3. 서버에서 그 Chunk를 삭제한다.
4. 아래의 Chunk가 있다면 끌어 올린다.

(#3) Chunk: render

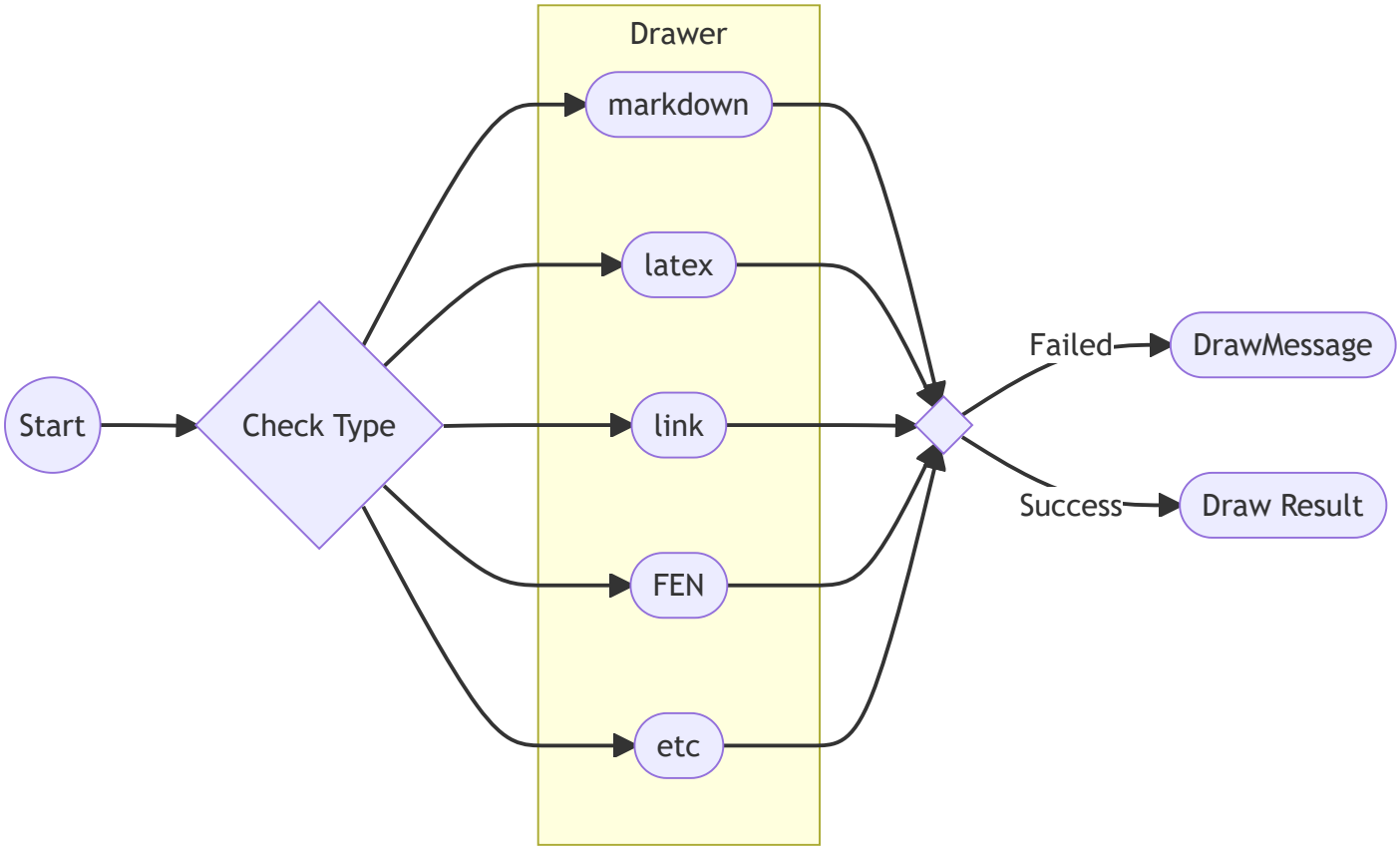
액터: 사용자
시작조건: 없음
목표: Chunk를 보여준다.

1. 내용을 읽는다.
2. 그 내용을 chunk안 영역에 사람이 보기 좋게 그 타입에 따라 렌더링한다. render하는 대상 목록은 다음과 같다.
    - markdown
    - latex
    - link (image, video, site)
    - FEN
    - etc

대안 흐름:

A. 렌더링 실패

1. 렌더링에 실패하면 실패의 이유를 보여준다.



(#4) Chunk: previews

액터: 사용자
사용조건: 편집 중일때
목표: 미리보기를 보여주어 편집을 편하게 한다.

1. Chunk의 내용을 바꾸면 시작된다.
2. 보기모드에서 어떻게 보여질지 미리보기 창을 띄워준다. 미리보기는 기본적으로 하단에 띄우고 밑에 공간이 없으면 상단에 띄운다.
3. 내용이 바뀌면 미리보기 창의 내용도 갱신한다.

(#5) Document: view Chunk

액터: 사용자
시작조건: 읽기 권한이 있어야 한다.

목표: Chunk들을 보여준다.

1. Document가 로딩되면 시작한다.
2. 경로가 주어지면 Document Component에서 그 경로의 문서를 읽고 파싱한다. 그동안 로딩 바를 보여준다.
3. 로딩이 완료되면 파싱된 결과물인 Chunk들을 보여준다.

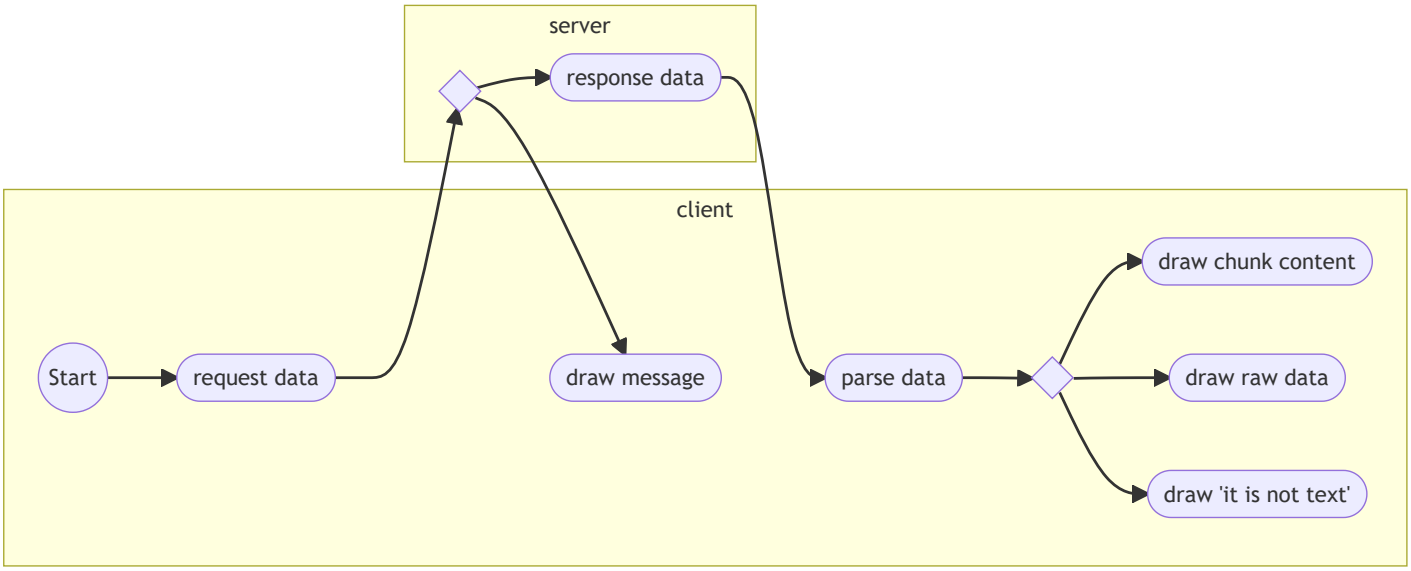
대안흐름:

A. 읽기 실패:

1. 읽기에 실패한 경우 읽기에 실패한 이유를 띄운다.

B. 파싱 실패:

1. 파싱에 실패한 경우 파싱에 실패한 이유를 띄우고 raw text가 담긴 Chunk로 렌더링한다.



(#6) Document: remove

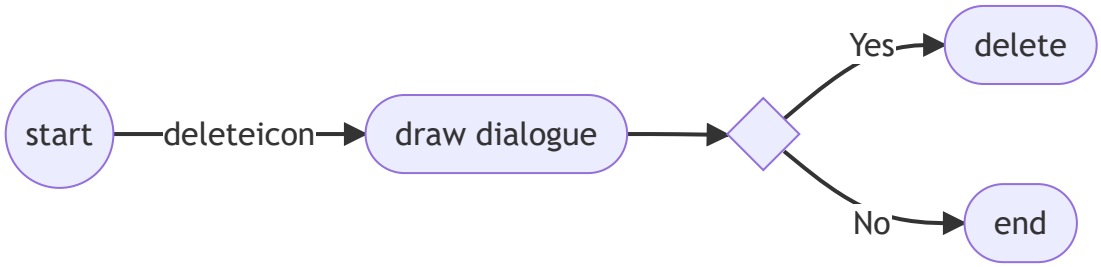
액터: 사용자
시작조건: 문서를 삭제할 권한이 있어야함.
목표: 앱에서 문서를 삭제한다.

1. Document의 AppBar에 놓여있는 삭제 아이콘을 클릭하면 시작한다.

2. 정말로 삭제하겠냐는 다이얼로고가 띄운다.
3. 거기서 예스를 누르면 Document를 삭제한다.

대안 흐름:

1. 다이얼로그에서 아니오를 누르면 다이얼로고를 닫고 종료한다.



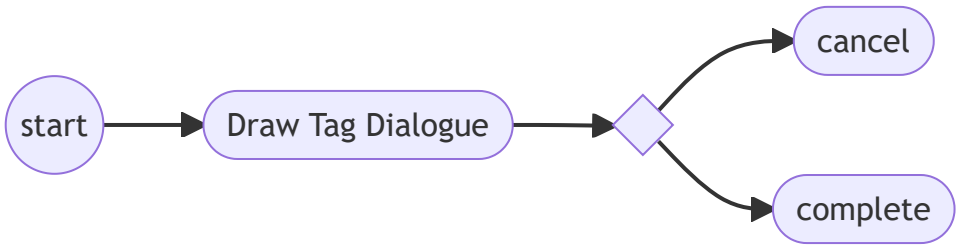
(#7) Document: add/delete tag

액터: 사용자
시작조건: 태그 수정 권한이 있을 때
목표: 문서의 태그를 추가/수정/삭제한다.

1. Document의 AppBar에 놓여있는 태그 수정 아이콘을 클릭하면 시작한다. 태그 수정 다이얼로그를 띄운다.
2. 태그 수정 다이얼로그에서 태그를 생성, 삭제한다.
3. 수정을 완료하고 저장 버튼을 누르면 태그 수정이 종료된다.

대안 흐름:

1. 취소 버튼을 누르면 다이얼로그를 닫고 종료한다.



(#8) Document: Drag And Drop Upload

액터: 사용자
시작조건: 문서 수정 권한이 있어야 한다.
목표: 업로드를 드래그 앤 드롭으로 한다.

1. 웹사이트의 그림이나 비디오 등의 파일을 Drag And Drop 해서 Chunk 사이에 놓으면 시작된다.
2. Drag And Drop된 파일을 서버에 업로드한다. 업로드시 파일 이름이 중복될 때 파일 이름이 밑줄과 숫자로 끝나지 않으면 이름의 뒤에 "_1"을 붙여 업로드 한다. 숫자로 끝나면 다음 숫자를 붙여서 업로드 한다. 파일 이름이 없다면 임의의 이름을 붙여서 업로드 한다.
3. 그 파일을 새로운 Chunk로 추가한다. 이때 웹에서 표시가능한 파일(이미지, 동영상)이면 파일을 표시하는 Chunk를 추가하고 아니면 다운로드 링크를 가진 Chunk를 추가한다.

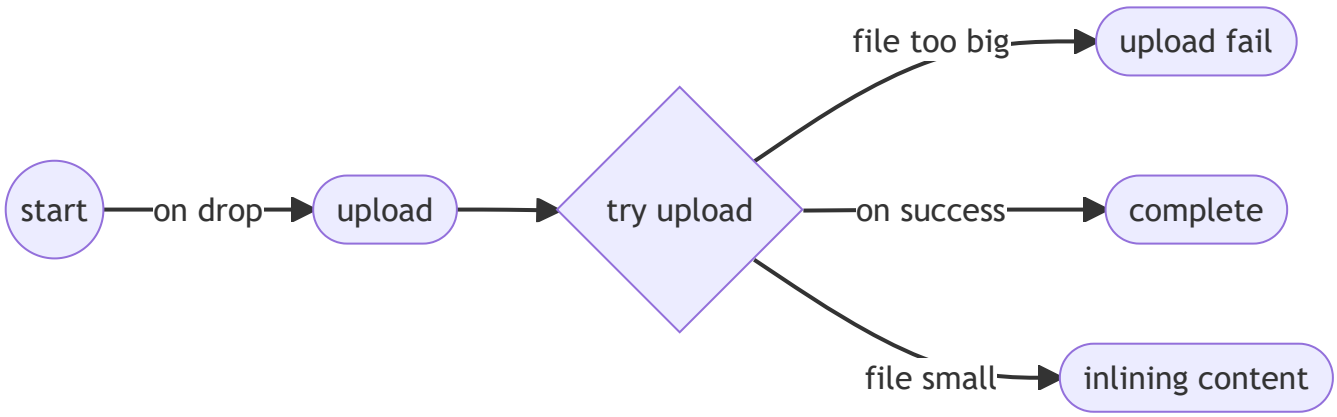
대체흐름:

A. 파일 사이즈 큼

1. 기본단계 2에서 파일 사이즈가 설정보다 크면 시작한다.
2. 파일 크기가 너무 크다는 메세지로 띄우고 종료한다.

B. 작은 파일 사이즈

1. 기본단계 2에서 파일 사이즈가 설정보다 작으면 시작한다.
2. 파일을 base64로 인코딩해서 Chunk로 삽입한다.



(#9) Document: Auto-Refresh

액터: 외부 편집기
시작조건: 없음
목적: 변화를 실시간으로 따라갈 수 있게 한다.

1. Document나 Document가 포함하는 미디어의 파일이 다른 편집 프로그램에 의해서 변경되었을 시에 시작한다.
2. 보고 있는 사용자의 Document 뷰를 변경 부분만 Refresh한다. 이때 보고 있던 스크롤이 변하지 않게 유의한다.

(#10) Chunk: autocomplete

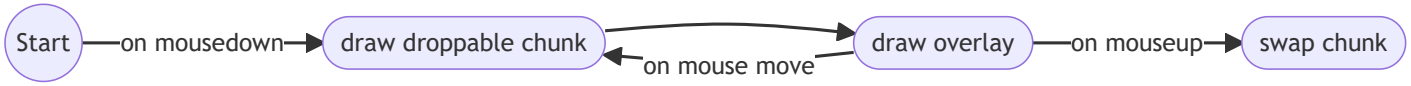
액터: 사용자
시작조건: chunk 수정를 하고 있어야 한다.
목표: 링크 등을 자동으로 완성해서 편집을 편하게 한다.

1. `Ctrl+Space`로 자동완성 창을 띄우는 명령을 내릴 때 시작한다.
2. caret cursor위치를 찾아서 그 위치에 자동완성 창을 띄운다.
3. 자동완성은 caret cursor 앞의 단어를 보고 문맥을 추론하여 추천 리스트를 만든다. 순서는 일반적으로는 abc순으로 한다.
4. 추천 리스트 중 알맞은 것을 방향키로 고르게 한다.
5. `Tab`이나 `Enter`를 통해 선택한다.
6. 선택한 단어로 완성시킨다.

(#11) Chunk: swap positions

액터: 사용자
시작 조건: Document 수정권한이 있어야 한다.
목적: 앱에서 Chunk 위치를 바꿀 수 있다.

1. Focus를 얻고 핸들 아이콘을 누를때 시작한다.
2. 이때 오버레이를 표시해서 놓여졌을 때의 상황을 미리 볼 수 있게 한다.
3. 드래그해서 원하는 장소에 놓으면 위치를 바꿀 수 있다.

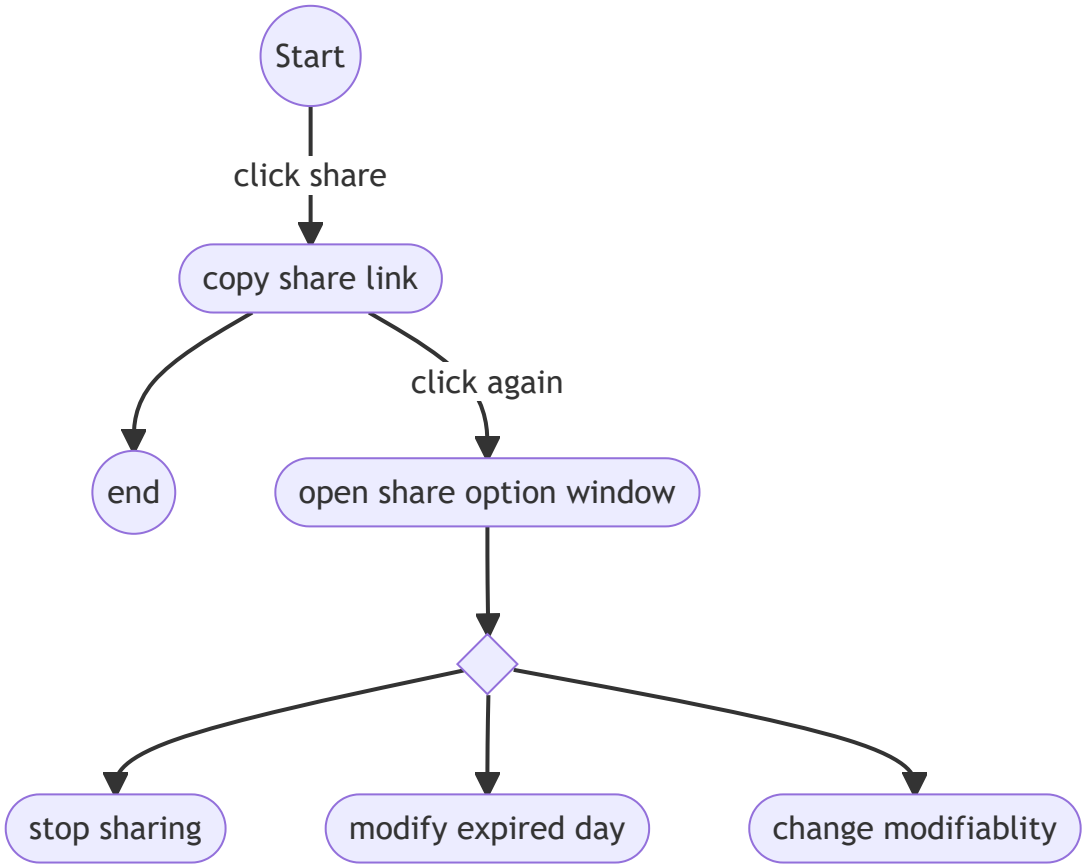


(#12) Document: Share

액터: 사용자
시작 조건: 문서를 공유할 수 있는 권한을 가져야한다.
목표: 문서를 공유한다.

1. Document의 AppBar에 놓여있는 공유 아이콘을 누르면 시작한다.
2. 공유 링크를 복사한다. 그리고 공유 설정아이콘을 띄워준다. 여기서 종료할 수 있다.

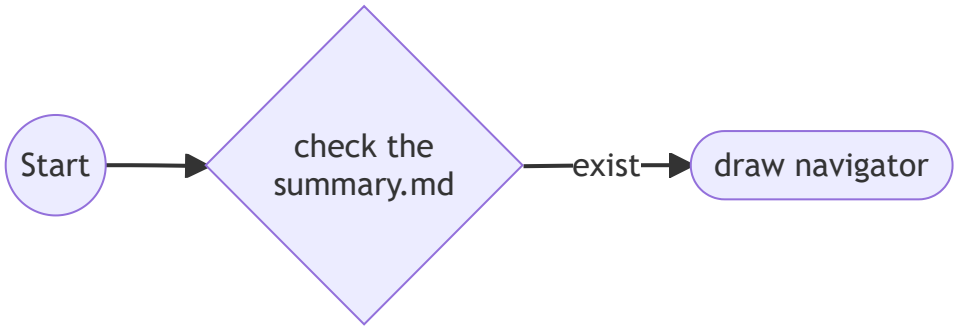
3. 공유 설정아이콘을 클릭하면 공유 설정 다이얼로그를 띄운다. 이 다이얼로그에서는 공유 기간과 편집 가능여부 등을 설정할 수 있고 공유 링크를 복사할 수 있다. 공유를 취소할 수도 있다.



(#13) Document: Navigator

액터: 사용자
시작 조건: 없음
목표: 문서들을 쉽게 이동할 수 있는 네비게이터를 보여준다.

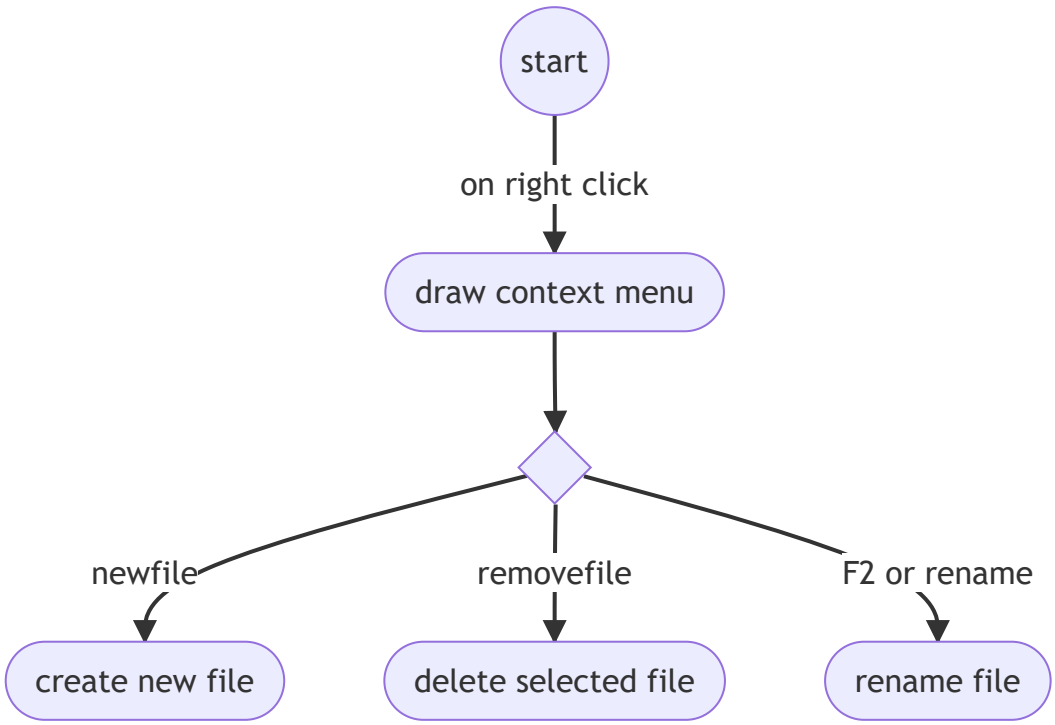
1. 만일 문서가 속하는 디렉토리에 Summary.md 가 있고 올바른 형식(링크와 리스트로 이루어져 있음)이면 시작한다.
2. Document의 왼쪽에 Summary의 내용을 네비게이터 역할로 표시한다.



(#14) File: create/delete/rename file

액터: 사용자
시작조건: 디렉토리에 대한 권한을 가지고 있어야한다.
목표: 앱상에서 파일을 생성하거나 삭제 할 수 있어야 한다.

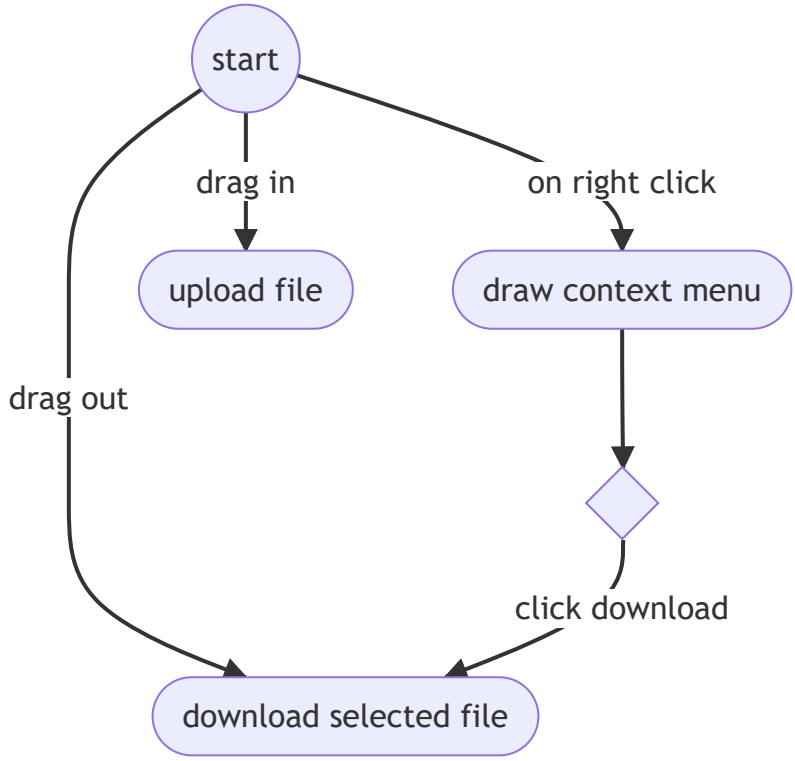
1. Treeview에 포커스가 간 상태에서 시작한다.
2. Treeview에서 오른쪽 클릭을 하면 Context Menu가 나오고 새파일을 클릭하면 이름을 지정해서 파일을 생성할 수 있다.
3. Context Menu에서 삭제를 클릭하면 해당 파일을 삭제한다.
4. 이름 바꾸기를 클릭하거나 Treeview에서 파일에 포커스가 간 상태에서 `F2`를 입력하면 이름을 바꿀 수 있도록 한다.



(#15) File: upload/download files

액터: 사용자
시작조건: 디렉토리의 권한이 있어야 한다.
목표: 파일을 업로드하거나 다운로드 할 수 있어야 한다.

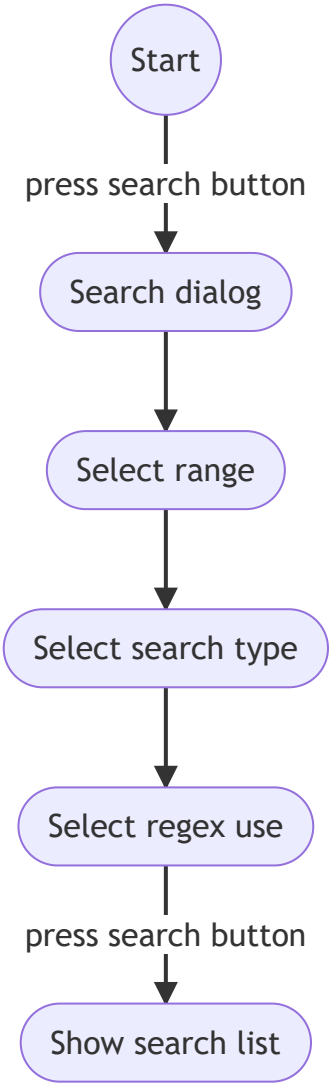
1. Treeview의 Context menu에서 다운로드 버튼을 클릭해서 다운로드 할 수 있다. 아니면 Treeview 파일을 Drag and Drop 하는 것으로도 가능하다.
2. Treeview에 파일을 드래그 앤 드롭하는 것으로 업로드 할 수 있다.



(#16) Search: Document Search

액터: 사용자
시작조건: 검색에는 문서를 읽을 수 있는 권한이 있어야 한다. 목표: 문서들을 검색할 수 있다.

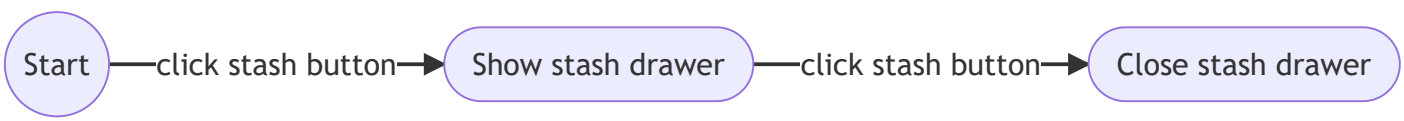
1. Drawer에서 문서 검색 버튼을 누르면 문서 검색 창이 뜬다.
2. 문서 검색창에서 범위를 지정한다. 기본 범위는 현재 보고 있는 문서의 디렉터리로 한다.
3. 문서 검색 창에서 태그를 검색할지 내용으로 검색을 할지 지정한다. 기본값은 내용이다.
4. 정규식을 사용할 것인지 지정한다. 기본값은 사용 안함이다.
5. 검색 버튼을 누르면 해당 조건을 만족하는 문서를 리스트로 보여준다.



(#17) Stash: render

액터: 사용자
시작조건: 화면크기가 768px 이상일때 보여준다.
목표: Stash를 보여준다.

1. 오른쪽에 작은 버튼을 두고 버튼을 클릭하면 시작한다.
2. 오른쪽 Drawer가 열려서 Stash의 내용을 보여준다. 최대 지정된 숫자만큼의 내용들을 보여준다.
3. 다시 버튼을 누르면 Stash Drawer를 닫는다.



(#18) File: export document

액터: 사용자
시작조건: 문서를 읽을 수 있는 권한이 있어야 한다. 문서 타입에만 가능하다.
목표: 문서를 보기 모드에서 보이는 것처럼 출력할 수 있다.

1. 메뉴의 Context menu에서 접근 할 때 시작한다.
2. 출력하기를 누르면 출력 다이얼로그가 뜬다.
3. 출력할 문서 타입을 문서 타입을 설정한다. 출력할 수 있는 문서 타입은 다음과 같다.
    - pdf
    - html
4. 출력하기를 누르면 출력된 문서를 다운로드한다.

대안흐름:

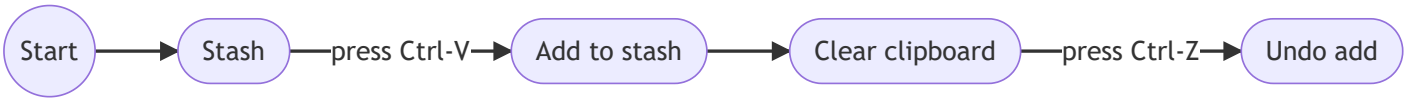
A. 취소

1. 단계 2에서 시작한다.
2. 취소하기를 누르면 다이얼로그 창을 닫고 종료한다.

(#19) Stash: add

액터: 사용자
시작 조건: Stash 창을 연 상태에서 포커스가 Stash 창에 주어져 있어야 한다.
목표: Stash를 추가한다.

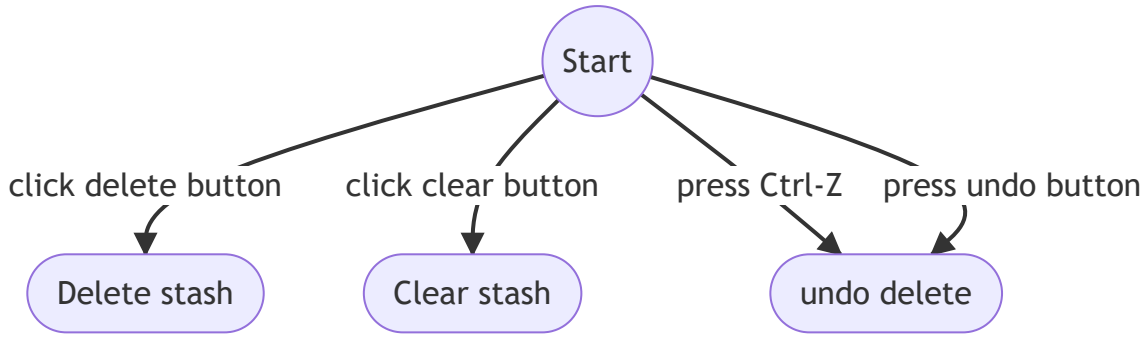
1. << 포함 #17 >>
2. `Ctrl+V`를 누르면 Stash에 클립보드의 내용이 Stash로 추가되고 클립보드는 비워진다.
3. `Ctrl+Z`를 눌러서 추가를 되돌릴 수 있다.



(#20) Stash: remove

액터: 사용자
시작조건: Stash 창을 연 상태에서 포커스가 Stash 창에 주어져 있어야 한다.
목표: 원하는 Stash를 삭제한다.

1. Stash 창의 각각 항목의 삭제 버튼을 클릭하면 선택된 항목을 삭제한다.
2. Stash에 붙어있는 삭제 버튼을 클릭하면 전체 항목을 삭제한다.
3. `Ctrl+Z`를 눌르거나 실행취소 버튼을 눌러 삭제를 되돌릴 수 있다.



(#21) Stash: Drag and Drop to Document

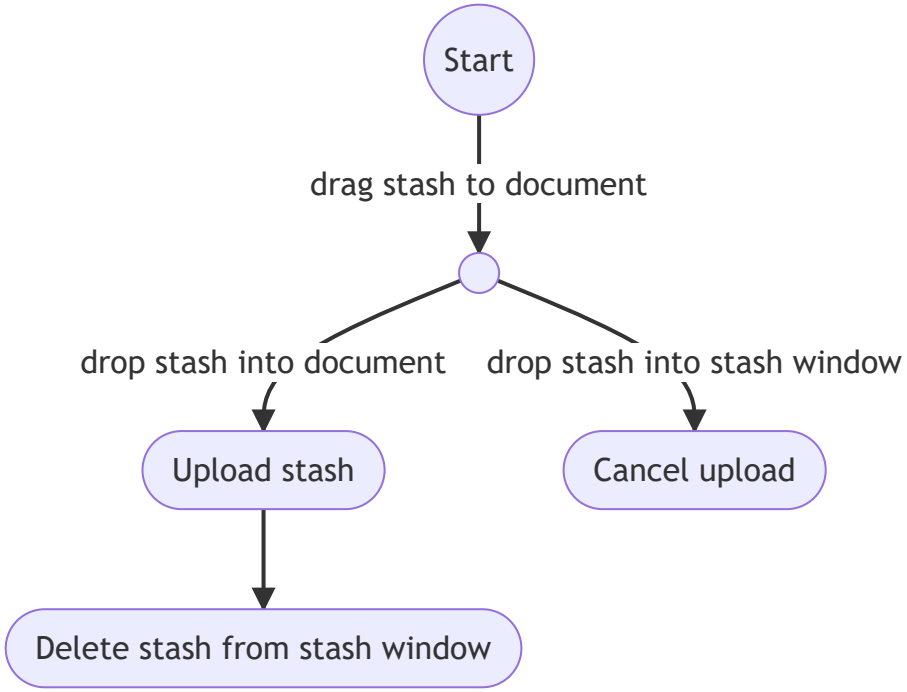
액터: 사용자
시작 조건: 문서 편집 권한이 있어야 한다.
목표: 드래그 앤 드롭으로 Chunk를 Stash에서 꺼내 삽입할 수 있다.

1. Stash 창의 Stash을 Document에 드래그하면 시작된다.
2. Document의 Chunk 사이에 Stash을 드롭하면 그 자리에 Chunk가 삽입된다. 사이의 결정은 제일 가까운 청크로 정한다.
3. Stash을 그 위치로 업로드시킨다. 업로드는 #8 와 한 것 같이 한다.
4. 해당 Stash를 Stash창에서 삭제한다.

대안흐름:

A. 취소

1. Stash 창에 놓으면 작업을 취소한다.



(#22) Management: Login

액터: 사용자
시작조건: 없음.
목표: 사용자가 액세스하기 위해 로그인할 수 있다.

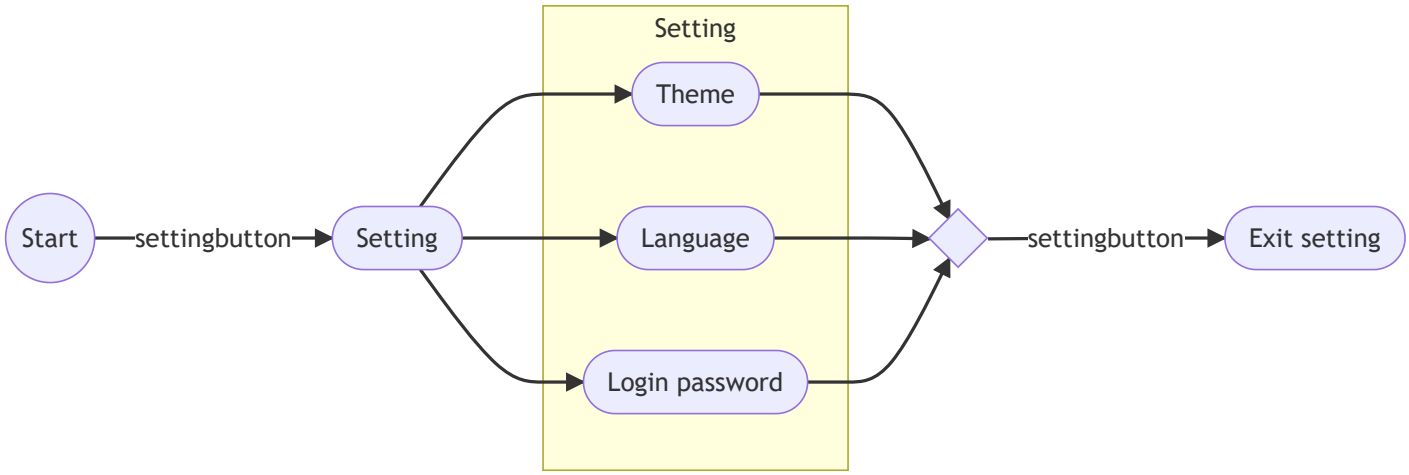
1. 사용자가 로컬에서 지정된 프로그램이 아닌 외부에서 접근한다면 시작한다.
2. 로그인 암호를 요구한다. 초기 로그인 암호는 환경변수에 의해서 결정된다.
3. 알맞은 암호를 입력했다면 권한을 부여한다.

(#23) Management: Configure

액터: 사용자
시작조건: 없음.
목표: 앱 정책등에 대해 앱에서 설정할 수 있다.

1. 편집기 내부에서 설정 아이콘을 클릭하면 설정창을 보여준다. 설정창은 프로그램의 속성을 설정할 수 있고 로그를 볼 수 있다. 설정창은 다음 항목들을 포함한다.
    - 로그인 암호
    - 테마 설정

    - 언어 설정 이것은 권한에 따라 선택적으로 렌더링된다.
2. 다시 설정 아이콘을 클릭하면 설정창을 닫는다.

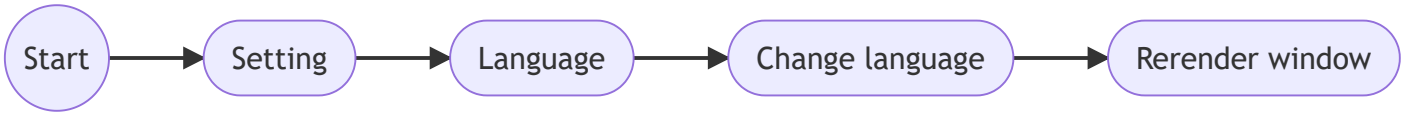


(#24) Management: Localization

어플리케이션을 여러가지 다양한 언어로 제공한다. 다음은 다른 언어로 바꾸는 사용 사례이다.

액터: 사용자
시작조건: 없음.
목표: 설정창에서 다른 언어로 바꿀 수 있다.

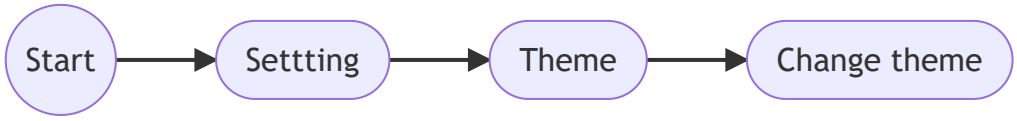
1. 설정창을 연다.
2. 언어 항목으로 간다. 지원되는 언어 리스트 창이 놓여저있다.
3. 언어 항목에서 다른 언어로 바꾼다. 이때 창을 다시 렌더링한다.



(#25) Management: Theme

액터: 사용자
시작조건: 없음.
목표: 설정창에서 테마를 설정할 수 있다.

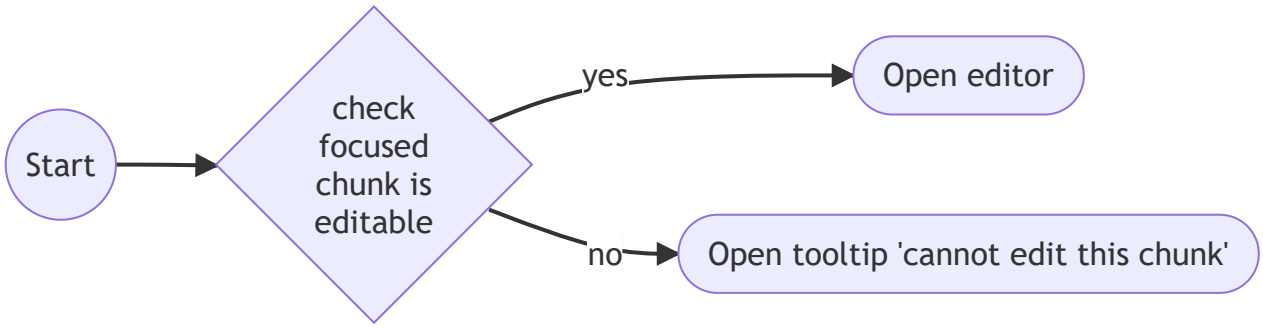
1. 설정창에서 테마로 이동한다.
2. 기본적으로 제공하는 밝은 테마과 어두운 테마를 고른다. 기본값은 밝은 색이다. 바꾸는 즉시 테마를 변경한다.
3. 커스텀 css 를 올려서 테마로 등록한다.



(#27) Chunk: edit

액터: 사용자
시작조건: Chunk의 Focus를 얻어야 한다.
목표: Chunk안의 컨텐츠를 수정할 수 있다.

1. 수정가능한 타입인지 확인한다.
2. 타입에 맞는 에디터를 띄운다. 예를 들어 text 타입이면 해당 Chunk 안의 text를 수정할 수 있게 한다.



(#28) Extension: API

액터: API 사용자
시작조건: 없음.
목표: 외부에서 API를 가지고 접근가능하도록 인터페이스를 제공한다.

1. 설정에 들어가서 API 토큰을 발급받는다.
2. api를 사용할때 Header에 발급받은 토큰을 넣고 통신한다.

API는 개발이 어느정도 진척되고 나서야 명세를 정할 수 있다. 그러므로 일단은 통신 방식만 명세한다.

(#29) Extension: Plugin

액터: 사용자
시작 조건: 없음
목표: Plugin으로 편집환경을 확장 가능하게 한다.

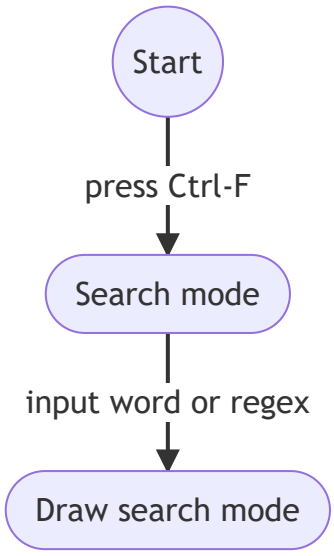
1. 플러그인을 폴더에 추가해 플러그인을 설치한다.
2. 플러그인을 사용한다.

> Plugin는 개발이 어느정도 진척되고 나서야 명세를 정할 수 있다. 그러므로 사용한다는 것만 명세한다.

(#30) Search: Find word

액터: 사용자
시작조건: 문서를 읽을 수 있는 권한을 가져야한다.
목표: 문서에서 단어를 검색한다.

1. `Ctrl+F`를 누르면 작은 검색 창이 오른쪽 상단에 표시된다.
2. 검색할 단어를 입력한다. 정규식도 가능하다.
3. 검색된 단어의 배경에 색깔을 칠해 구별 가능하게 한다.

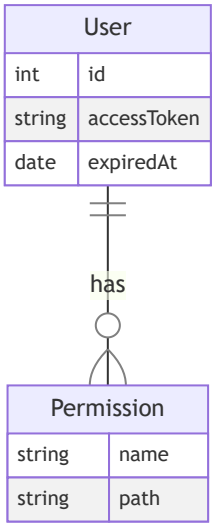


3.3. 성능 요구사항(Performance requirements)

1. 최소 1000 RPS를 보장해야한다.

2. 첫 로드후 로딩하면 0.5s 이내에 동작해야합니다
3. 동시 편집 이용자를 5명까지는 허용해야 합니다.
4. 적어도 400개의 파일을 관리할 수 있어야 합니다.

3.4. 논리적 데이터베이스 요구사항(Logical database requirements)



단순하게 공유를 위한 User 구조만 있다.

3.5. 설계 제약사항(Design constraints)

2.1.1. 에서 언급했듯이 다음 브라우저에서 동작해야 한다.

- Chrome 버전 61 이상
- Firefox 버전 60 이상
- Edge 버전 79 이상
- Safari 버전 11 이상
- Chrome for Android 버전 100 이상
- Samsung internet 버전 8.2 이상

그리고 모바일에서도 큰 불편함 없이 원활히 동작해야 한다.

3.5.1. 표준 준수(Standards compliance)

해당되지 않음.

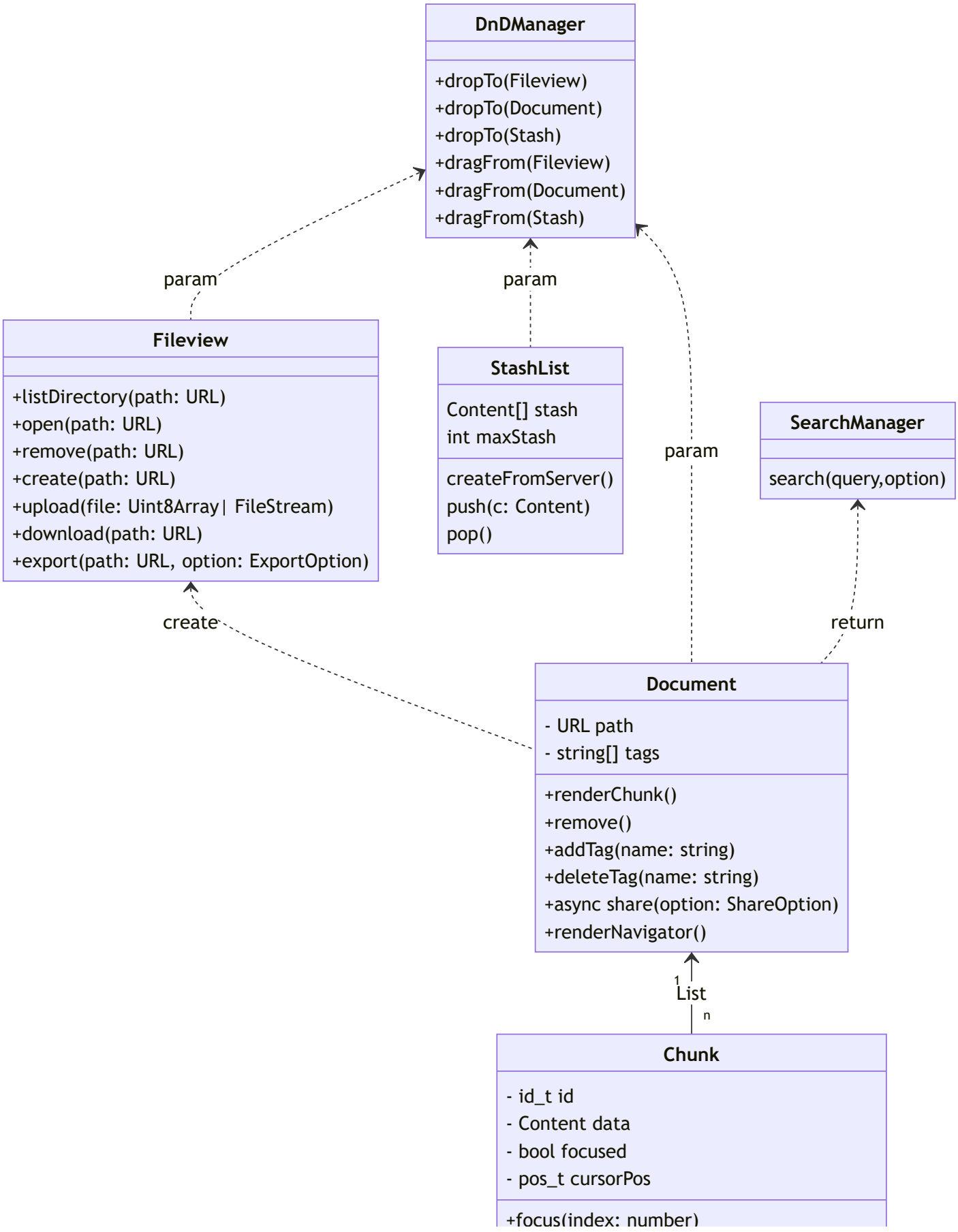
3.6. 소프트웨어 시스템 속성(Software system attributes)

해당되지 않음.

3.7. 상세 요구사항의 구성(Organizing the specific requirements)

3.7.1. 객체(Objects)

다음과 같은 UML을 그릴 수 있다.


DnDManager
+dropTo(Fileview)
+dropTo(Document)
+dropTo(Stash)
+dragFrom(Fileview)
+dragFrom(Document)
+dragFrom(Stash)
param
param
param
Fileview
+listDirectory(path: URL)
+open(path: URL)
+remove(path: URL)
+create(path: URL)
+upload(file: Uint8Array| FileStream)
+download(path: URL)
+export(path: URL, option: ExportOption)
StashList
Content[] stash
int maxStash
createFromServer()
push(c: Content)
pop()
SearchManager
search(query,option)
create
return
Document
- URL path
- string[] tags
+renderChunk()
+remove()
+addTag(name: string)
+deleteTag(name: string)
+async share(option: ShareOption)
+renderNavigator()
1
List
n
Chunk
- id_t id
- Content data
- bool focused
- pos_t cursorPos
+focus(index: number)

+unfocus(index: number)
+remove(index: number)
+insertBefore()
+draw()
+drawPreview()
+autoComplete(ctx: AutoCompleteContext)
+swapWith(p : Chunk)
+edit()

Management
login(auth:Auth) setServerConfigure(config: Configure) setLocale(lang: string) setTheme(theme: string) getLocale() getTheme() getServerConfigure()

Extension
registerPlugin(plugin: Plugin)

Chunk는 문서를 이루는 기본적인 단위이다. 글의 문단이라고 생각 할 수 있다. Document는 그런 Chunk의 리스트이다. FileView는 파일에 접근하고 Document를 열 수 있는 파일 브라우저이다. StashList는 단순한 파일이나 텍스트 등을 임시로 저장하고 꺼내쓰는 컨테이너이다.

3.7.2. 사용자 인터페이스 상세

다음과 같이 컴포넌트가 배치될 것이다. 배치될 컴포넌트는 Treeview, Chunk, ChunkList, Appbar, GeneralDialogue, Drawer, ContextMenu, StashList가 있다.

요구사항 명세 추가 이력 (Requirement Specification Supporting Information)

4.1. 부록(Appendixes)

내용 없음.

4.2. 개발 환경(Development Environment)

프론트엔드는 Vite로 개발한다. 그리고 개발 언어로는 typescript를 사용한다. 그리고 react를 사용한다.

백엔드는 Deno를 사용한다.

4.3. 일정표(Schedule)

일정은 다음 표와 같다.

주차	구현 기능
4.24~4.30	#1, #2, #3, #5, #14, #15, #27
5.1~5.7	#4, #6, #7, #8, #11
5.8~5.14	#9, #10, #12, #22, #23
5.15~5.21	#13, #16, #17, #19, #20, #21, #30
5.22~5.28	#18, , #24, #25
5.29~6.4	#28, #29

주차 4.24~4.30

- (#1) Chunk: Focus/Unfocus
- (#2) Chunk: remove
- (#3) Chunk: render

- (#5) Document: view Chunk
- (#14) File: create/delete/rename file
- (#15) File: upload/download files
- (#27) Chunk: edit

주차 5.1~5.7

- (#4) Chunk: previews
- (#6) Document: remove
- (#7) Document: add/delete tag
- (#8) Document: Drag And Drop Upload
- (#11) Chunk: swap positions

주차 5.8~5.14

- (#9) Document: Auto-Refresh
- (#10) Chunk: autocomplete
- (#12) Document: Share
- (#22) Management: Login
- (#23) Management: Configure

주차 5.15~5.21

- (#13) Document: Navigator
- (#16) Search: Document Search
- (#17) Stash: render
- (#19) Stash: add
- (#20) Stash: remove
- (#21) Stash: Drag and Drop to Document
- (#30) Search: Find word

주차 5.22~5.28

- (#18) File: export document
- (#24) Management: Localization
- (#25) Management: Theme

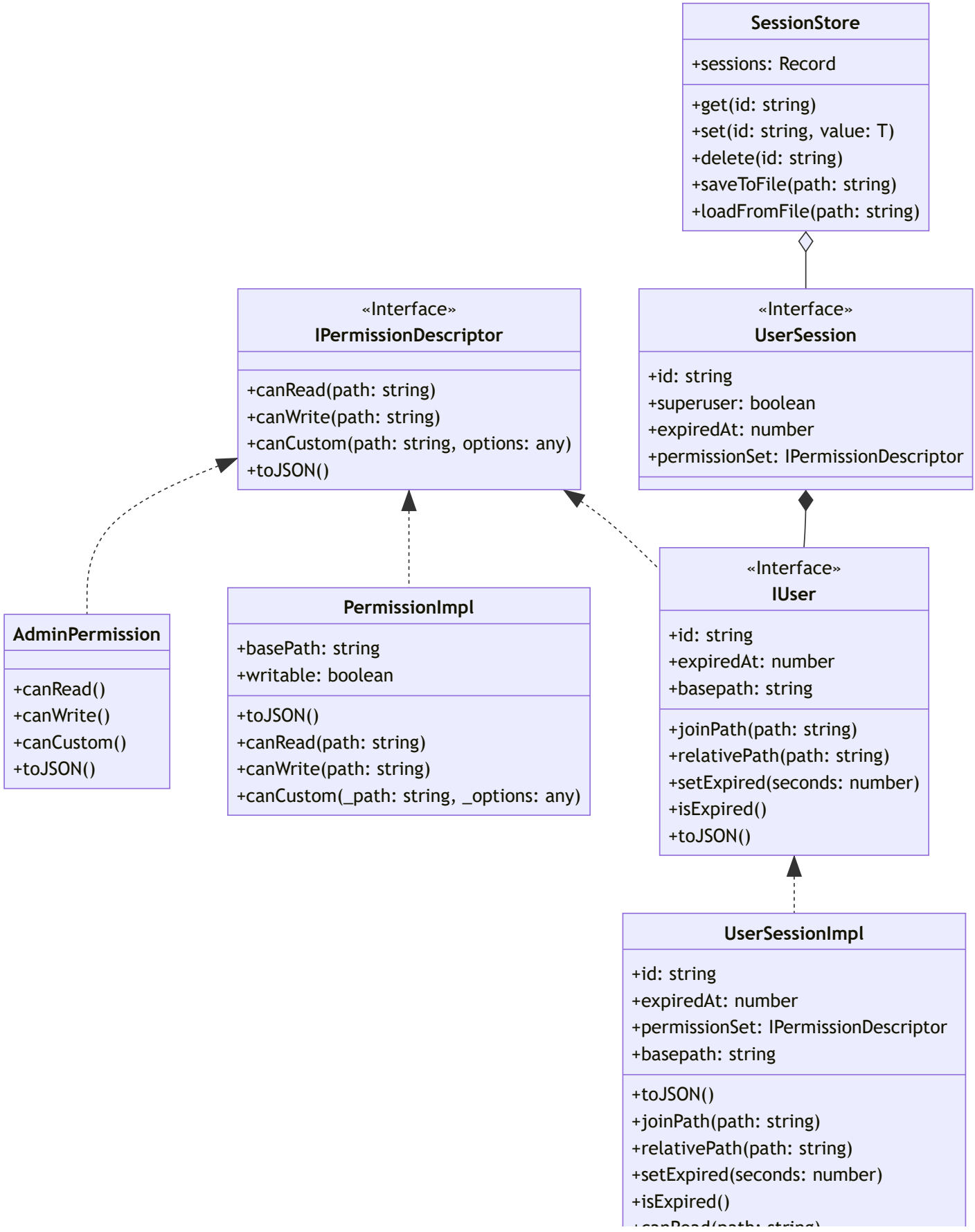
주차 5.29~6.4

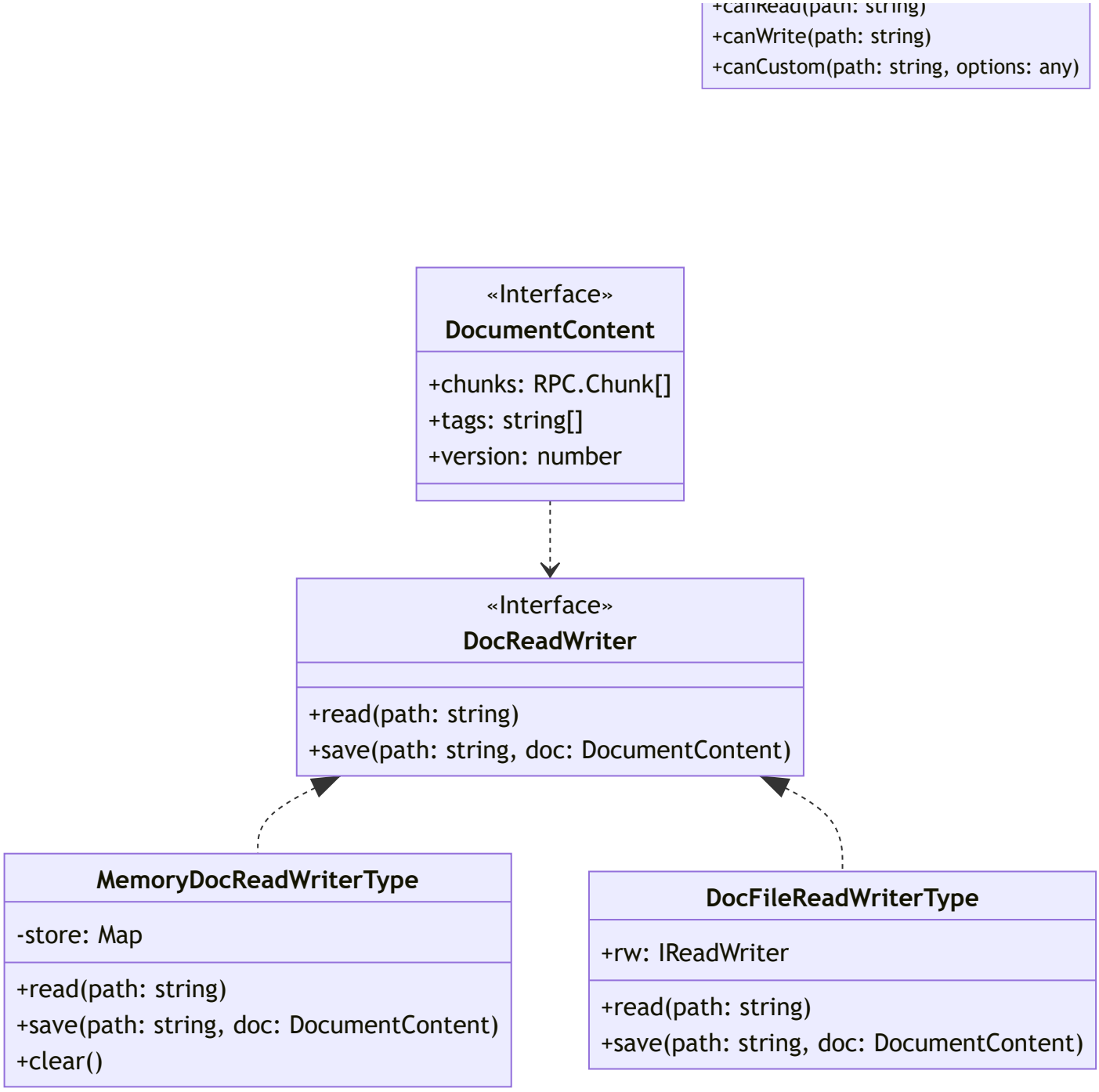
- (#28) Extension: API
- (#29) Extension: Plugin

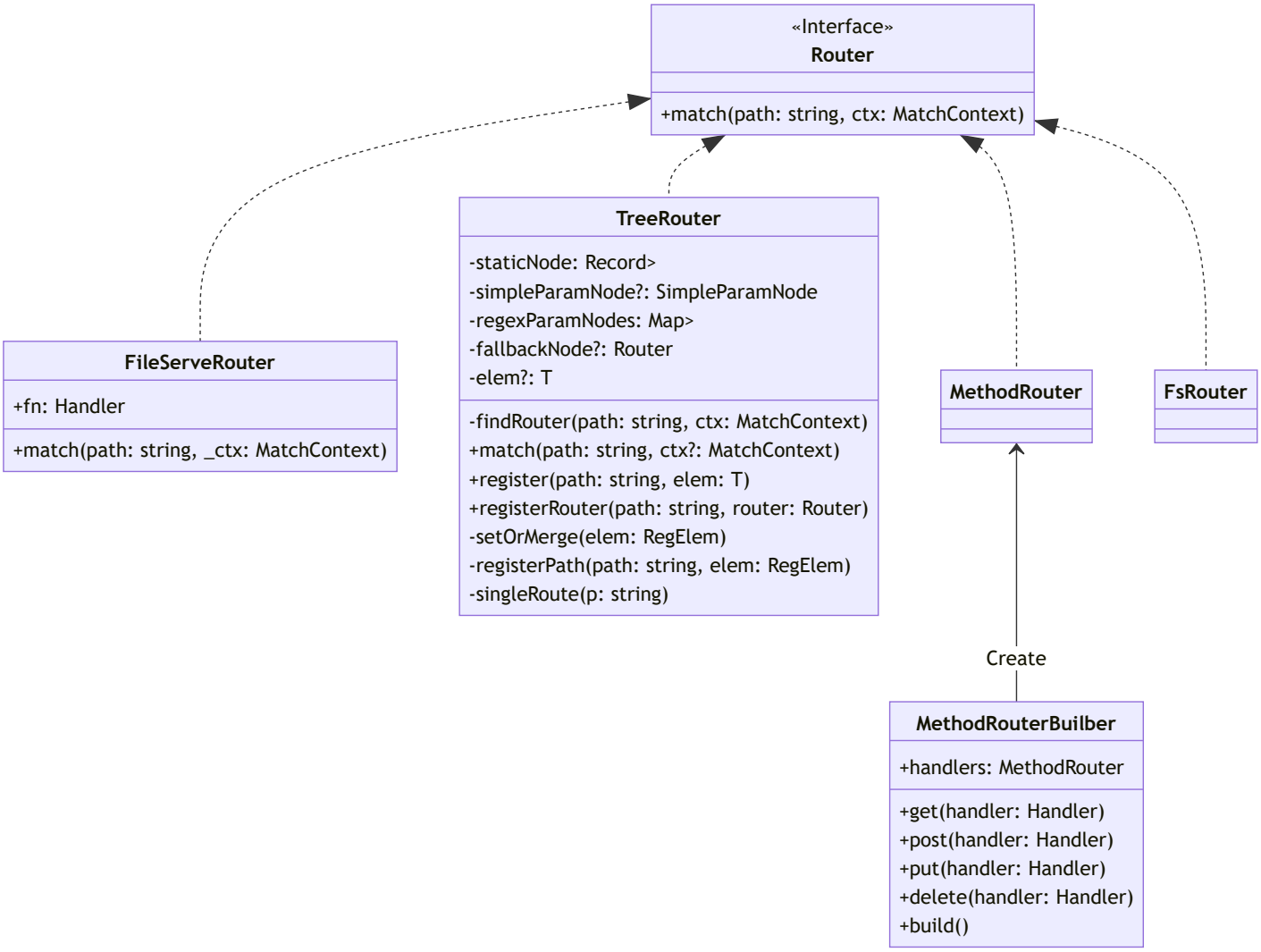
5. 설계(Architecture)

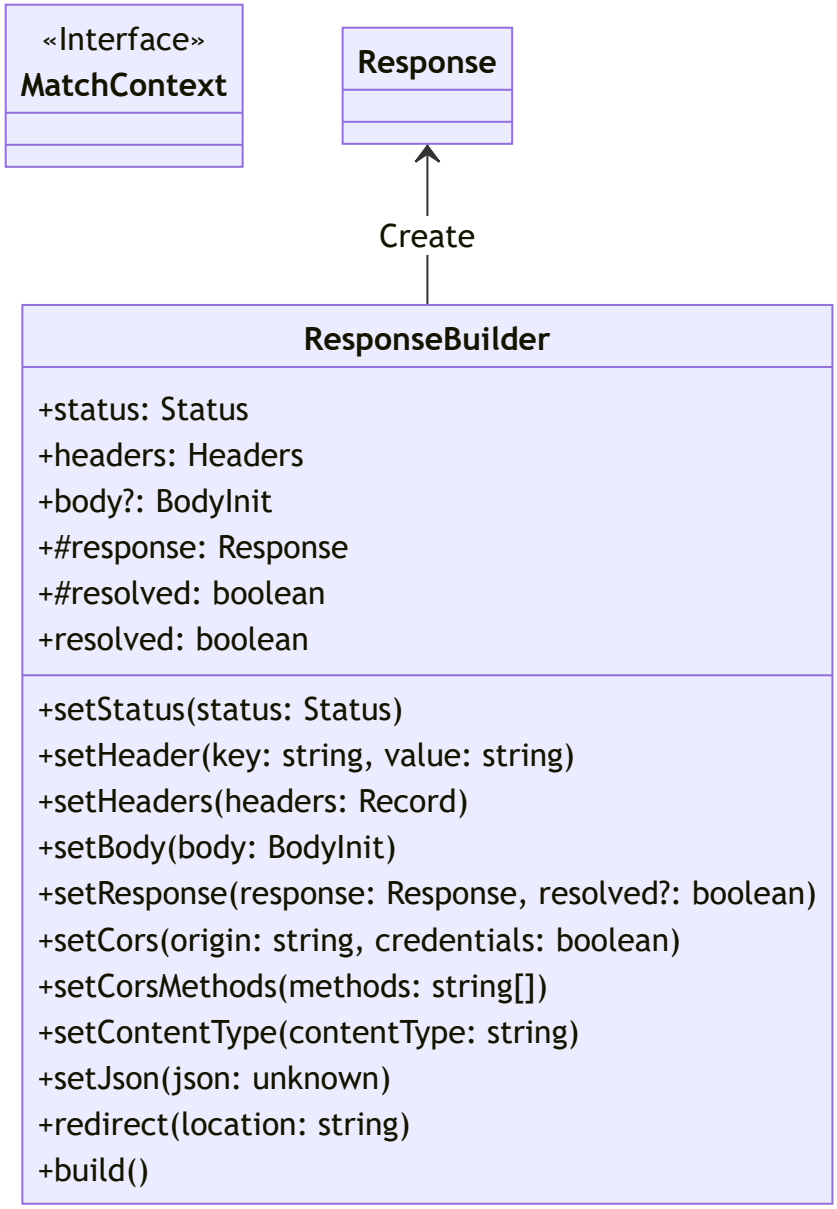
5.1 UML

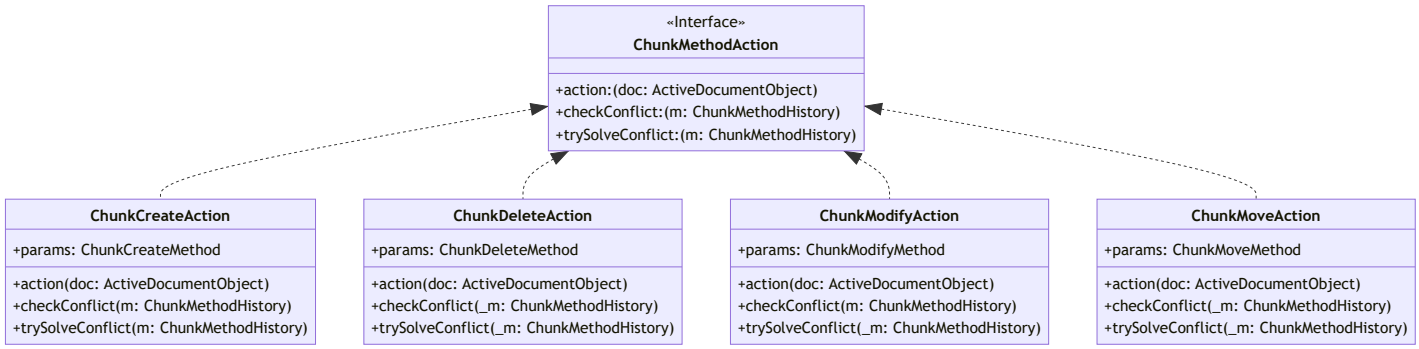
5.1.1 Server Side UML

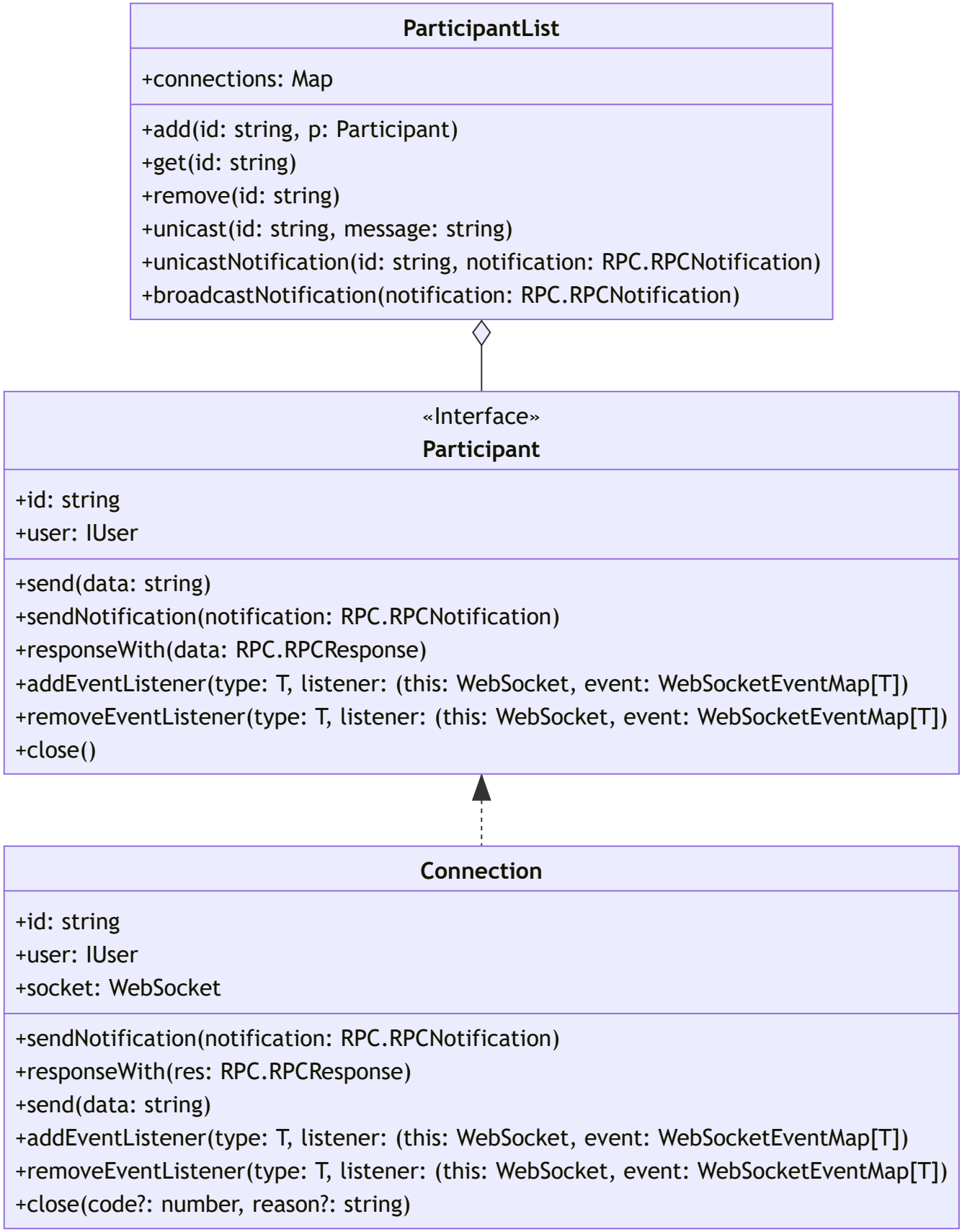

SessionStore
+sessions: Record
+get(id: string)
+set(id: string, value: T)
+delete(id: string)
+saveToFile(path: string)
+loadFromFile(path: string)
«Interface»
IPermissionDescriptor
+canRead(path: string)
+canWrite(path: string)
+canCustom(path: string, options: any)
+toJSON()
«Interface»
UserSession
+id: string
+superuser: boolean
+expiredAt: number
+permissionSet: IPermissionDescriptor
AdminPermission
+canRead()
+canWrite()
+canCustom()
+toJSON()
PermissionImpl
+basePath: string
+writable: boolean
+toJSON()
+canRead(path: string)
+canWrite(path: string)
+canCustom(_path: string, _options: any)
«Interface»
IUser
+id: string
+expiredAt: number
+basepath: string
+joinPath(path: string)
+relativePath(path: string)
+setExpired(seconds: number)
+isExpired()
+toJSON()
UserSessionImpl
+id: string
+expiredAt: number
+permissionSet: IPermissionDescriptor
+basepath: string
+toJSON()
+joinPath(path: string)
+relativePath(path: string)
+setExpired(seconds: number)
+isExpired()

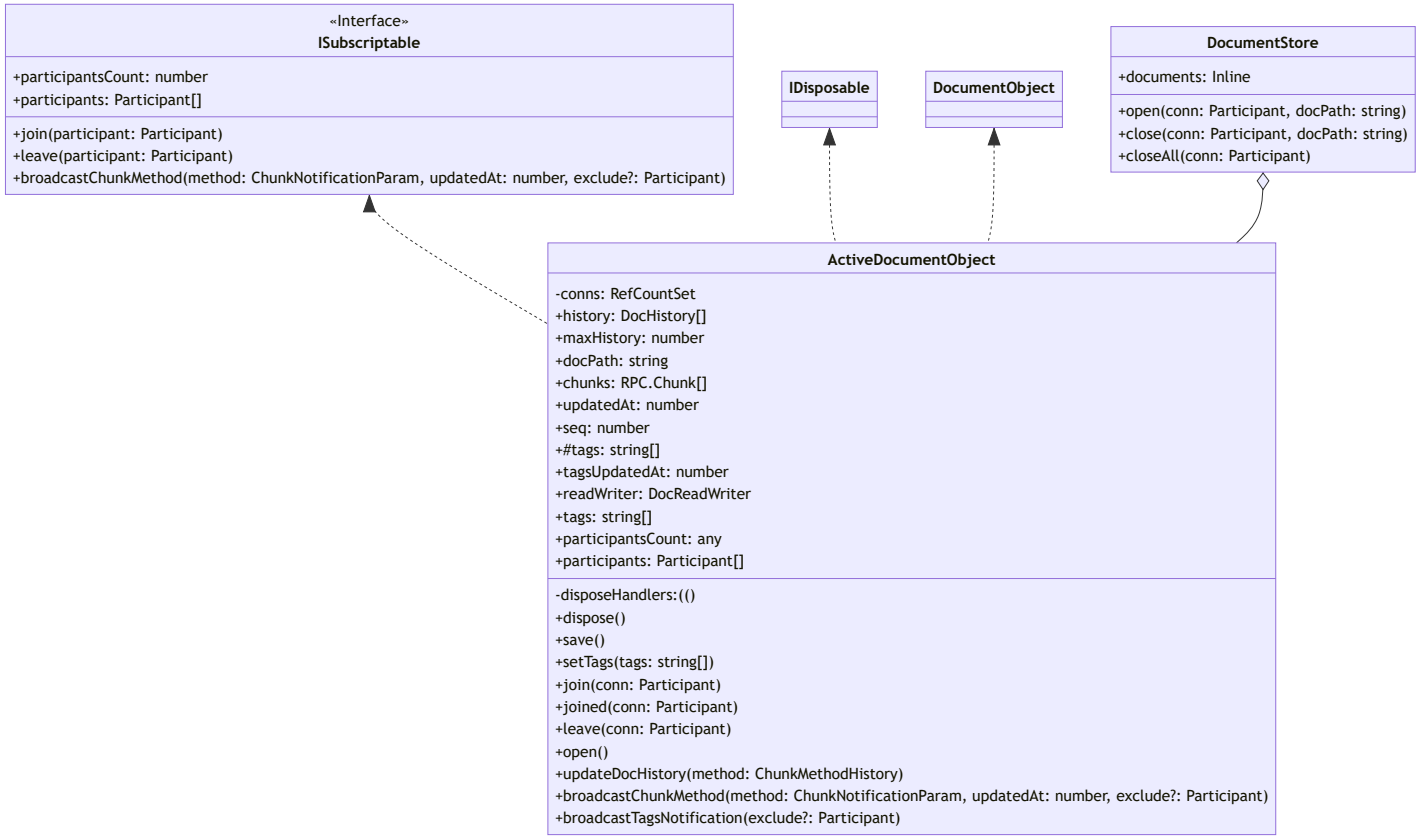

+canRead(path: string)
+canWrite(path: string)
+canCustom(path: string, options: any)
«Interface»
DocumentContent
+chunks: RPC.Chunk[]
+tags: string[]
+version: number
«Interface»
DocReadWriter
+read(path: string)
+save(path: string, doc: DocumentContent)
MemoryDocReadWriterType
-store: Map
+read(path: string)
+save(path: string, doc: DocumentContent)
+clear()
DocFileReadWriterType
+rw: IReadWriter
+read(path: string)
+save(path: string, doc: DocumentContent)

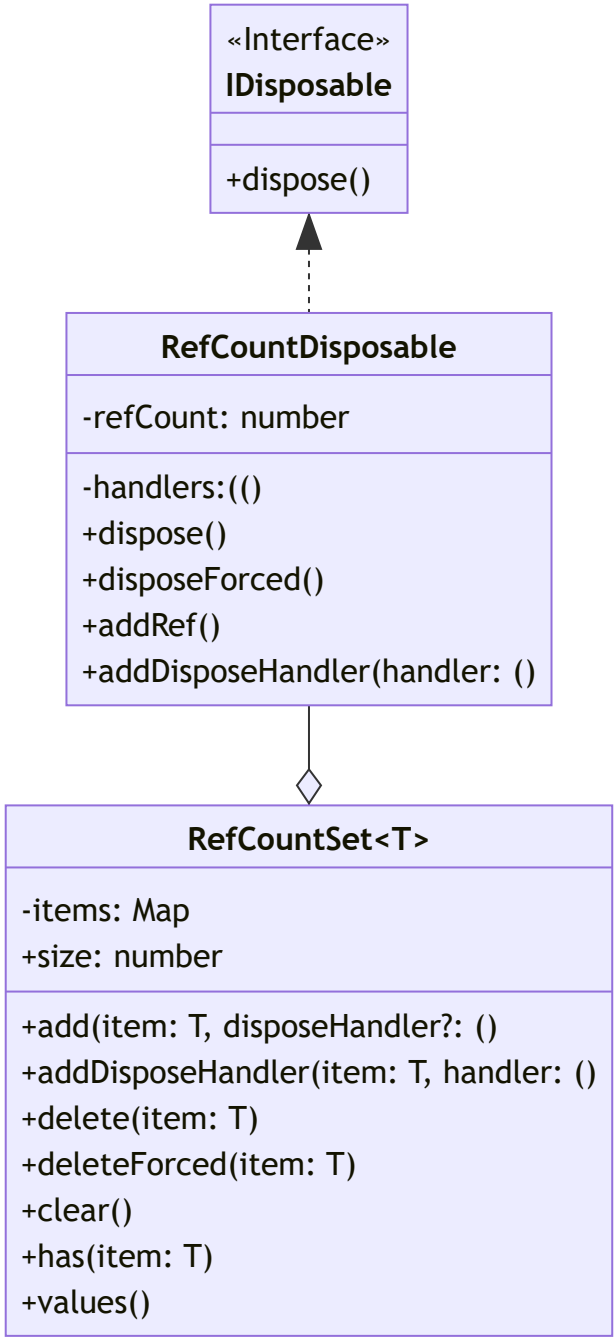

«Interface»
Router
+match(path: string, ctx: MatchContext)
TreeRouter
-staticNode: Record>
-simpleParamNode?: SimpleParamNode
-regexParamNodes: Map>
-fallbackNode?: Router
-elem?: T
-findRouter(path: string, ctx: MatchContext)
+match(path: string, ctx?: MatchContext)
+register(path: string, elem: T)
+registerRouter(path: string, router: Router)
-setOrMerge(elem: RegElem)
-registerPath(path: string, elem: RegElem)
-singleRoute(p: string)
FileServeRouter
+fn: Handler
+match(path: string, _ctx: MatchContext)
MethodRouter
FsRouter
Create
MethodRouterBuilber
+handlers: MethodRouter
+get(handler: Handler)
+post(handler: Handler)
+put(handler: Handler)
+delete(handler: Handler)
+build()

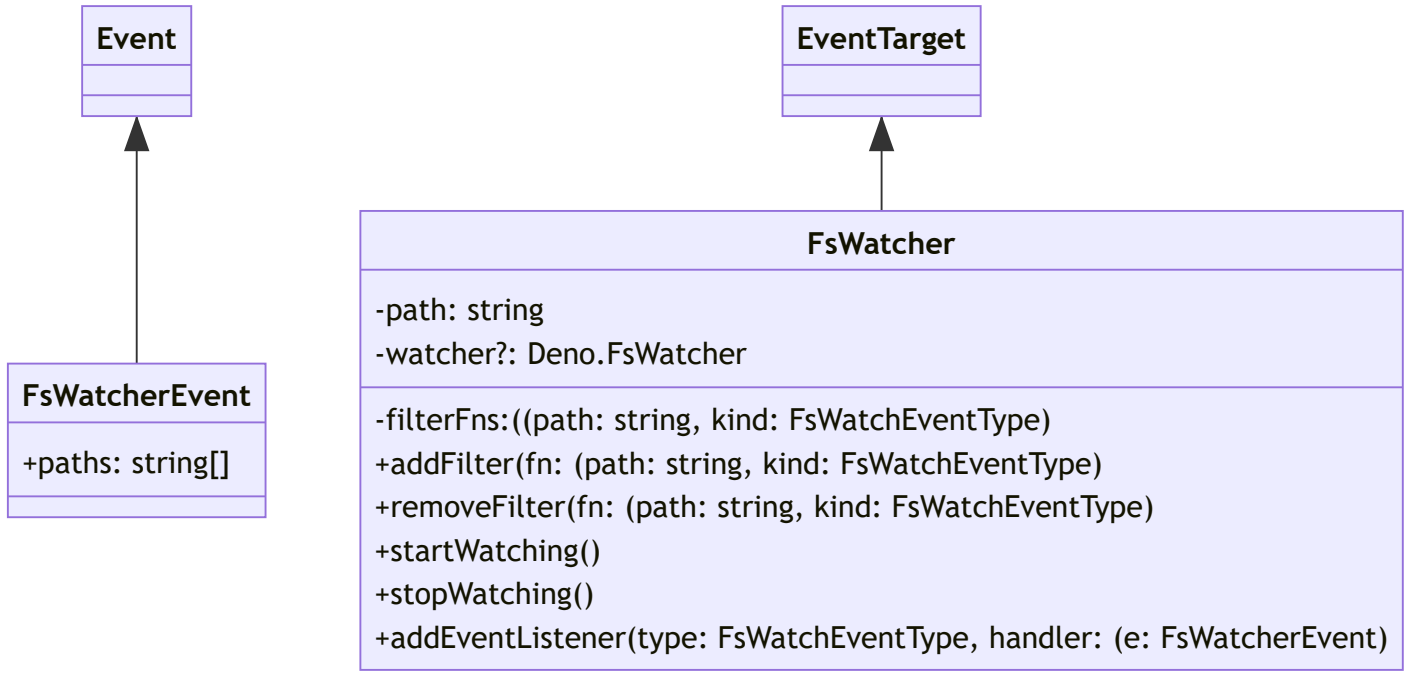

«Interface»
MatchContext
Response
Create
ResponseBuilder
+status: Status
+headers: Headers
+body?: BodyInit
+#response: Response
+#resolved: boolean
+resolved: boolean
+setStatus(status: Status)
+setHeader(key: string, value: string)
+setHeaders(headers: Record)
+setBody(body: BodyInit)
+setResponse(response: Response, resolved?: boolean)
+setCors(origin: string, credentials: boolean)
+setCorsMethods(methods: string[])
+setContentType(contentType: string)
+setJson(json: unknown)
+redirect(location: string)
+build()

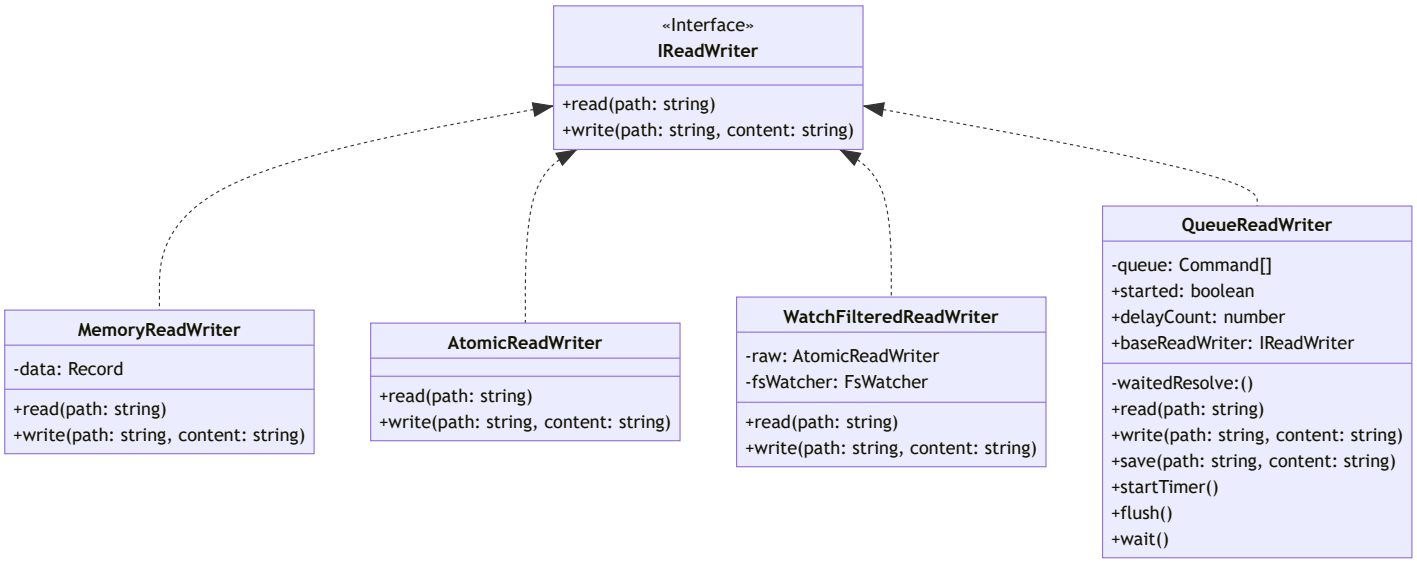

«Interface»
ChunkMethodAction
+action:(doc: ActiveDocumentObject)
+checkConflict:(m: ChunkMethodHistory)
+trySolveConflict:(m: ChunkMethodHistory)
ChunkCreateAction
+params: ChunkCreateMethod
+action(doc: ActiveDocumentObject)
+checkConflict(m: ChunkMethodHistory)
+trySolveConflict(m: ChunkMethodHistory)
ChunkDeleteAction
+params: ChunkDeleteMethod
+action(doc: ActiveDocumentObject)
+checkConflict(_m: ChunkMethodHistory)
+trySolveConflict(_m: ChunkMethodHistory)
ChunkModifyAction
+params: ChunkModifyMethod
+action(doc: ActiveDocumentObject)
+checkConflict(m: ChunkMethodHistory)
+trySolveConflict(_m: ChunkMethodHistory)
ChunkMoveAction
+params: ChunkMoveMethod
+action(doc: ActiveDocumentObject)
+checkConflict(_m: ChunkMethodHistory)
+trySolveConflict(_m: ChunkMethodHistory)


ParticipantList
+connections: Map
+add(id: string, p: Participant)
+get(id: string)
+remove(id: string)
+unicast(id: string, message: string)
+unicastNotification(id: string, notification: RPC.RPCNotification)
+broadcastNotification(notification: RPC.RPCNotification)
«Interface»
Participant
+id: string
+user: IUser
+send(data: string)
+sendNotification(notification: RPC.RPCNotification)
+responseWith(data: RPC.RPCResponse)
+addEventListener(type: T, listener: (this: WebSocket, event: WebSocketEventMap[T])
+removeEventListener(type: T, listener: (this: WebSocket, event: WebSocketEventMap[T])
+close()
Connection
+id: string
+user: IUser
+socket: WebSocket
+sendNotification(notification: RPC.RPCNotification)
+responseWith(res: RPC.RPCResponse)
+send(data: string)
+addEventListener(type: T, listener: (this: WebSocket, event: WebSocketEventMap[T])
+removeEventListener(type: T, listener: (this: WebSocket, event: WebSocketEventMap[T])
+close(code?: number, reason?: string)

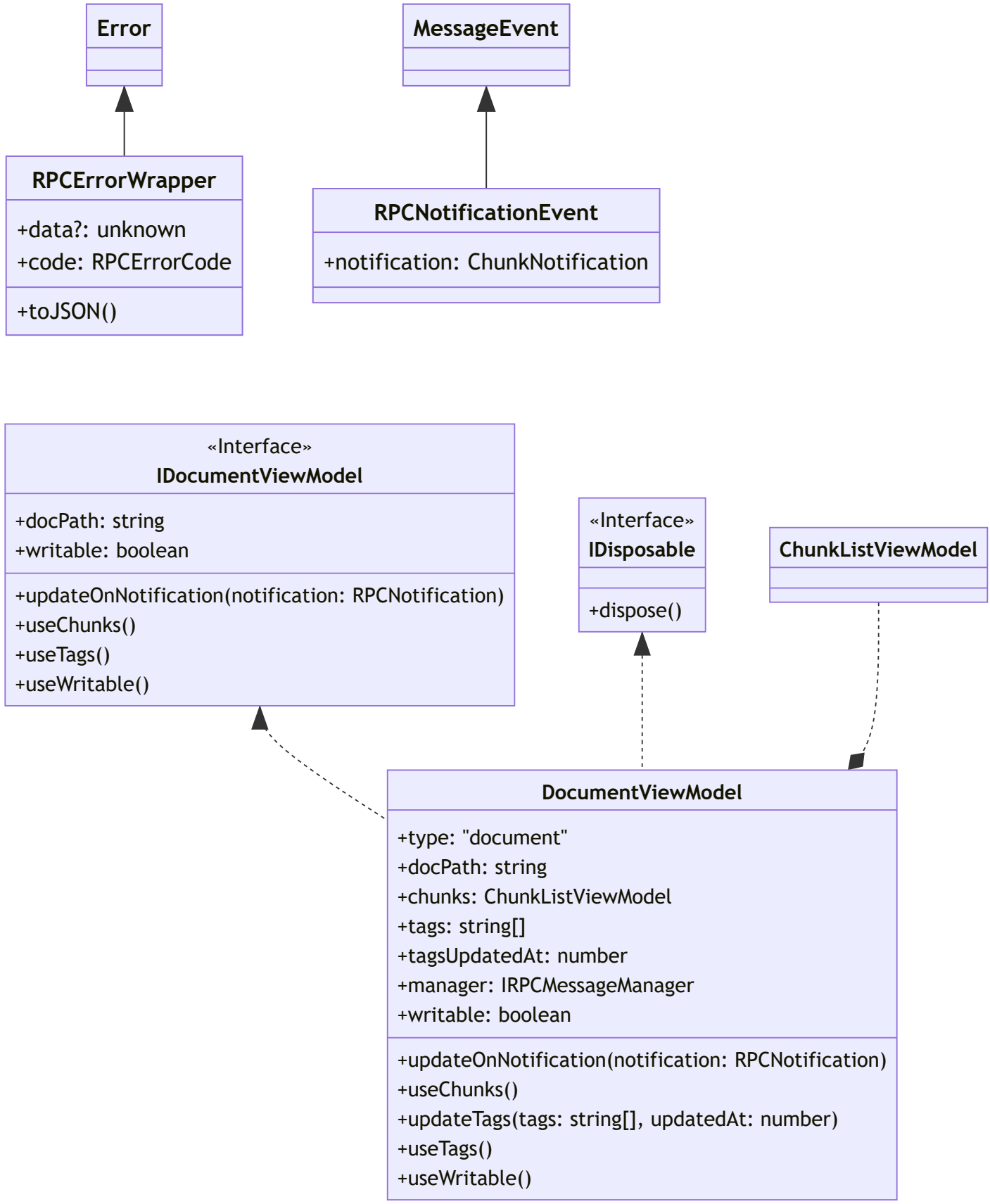

«Interface»
ISubscriptable
+participantsCount: number
+participants: Participant[]
+join(participant: Participant)
+leave(participant: Participant)
+broadcastChunkMethod(method: ChunkNotificationParam, updatedAt: number, exclude?: Participant)
IDisposable
DocumentObject
DocumentStore
+documents: Inline
+open(conn: Participant, docPath: string)
+close(conn: Participant, docPath: string)
+closeAll(conn: Participant)
ActiveDocumentObject
-conns: RefCountSet
+history: DocHistory[]
+maxHistory: number
+docPath: string
+chunks: RPC.Chunk[]
+updatedAt: number
+seq: number
+#tags: string[]
+tagsUpdatedAt: number
+readWriter: DocReadWriter
+tags: string[]
+participantsCount: any
+participants: Participant[]
-disposeHandlers:(()
+dispose()
+save()
+setTags(tags: string[])
+join(conn: Participant)
+joined(conn: Participant)
+leave(conn: Participant)
+open()
+updateDocHistory(method: ChunkMethodHistory)
+broadcastChunkMethod(method: ChunkNotificationParam, updatedAt: number, exclude?: Participant)
+broadcastTagsNotification(exclude?: Participant)

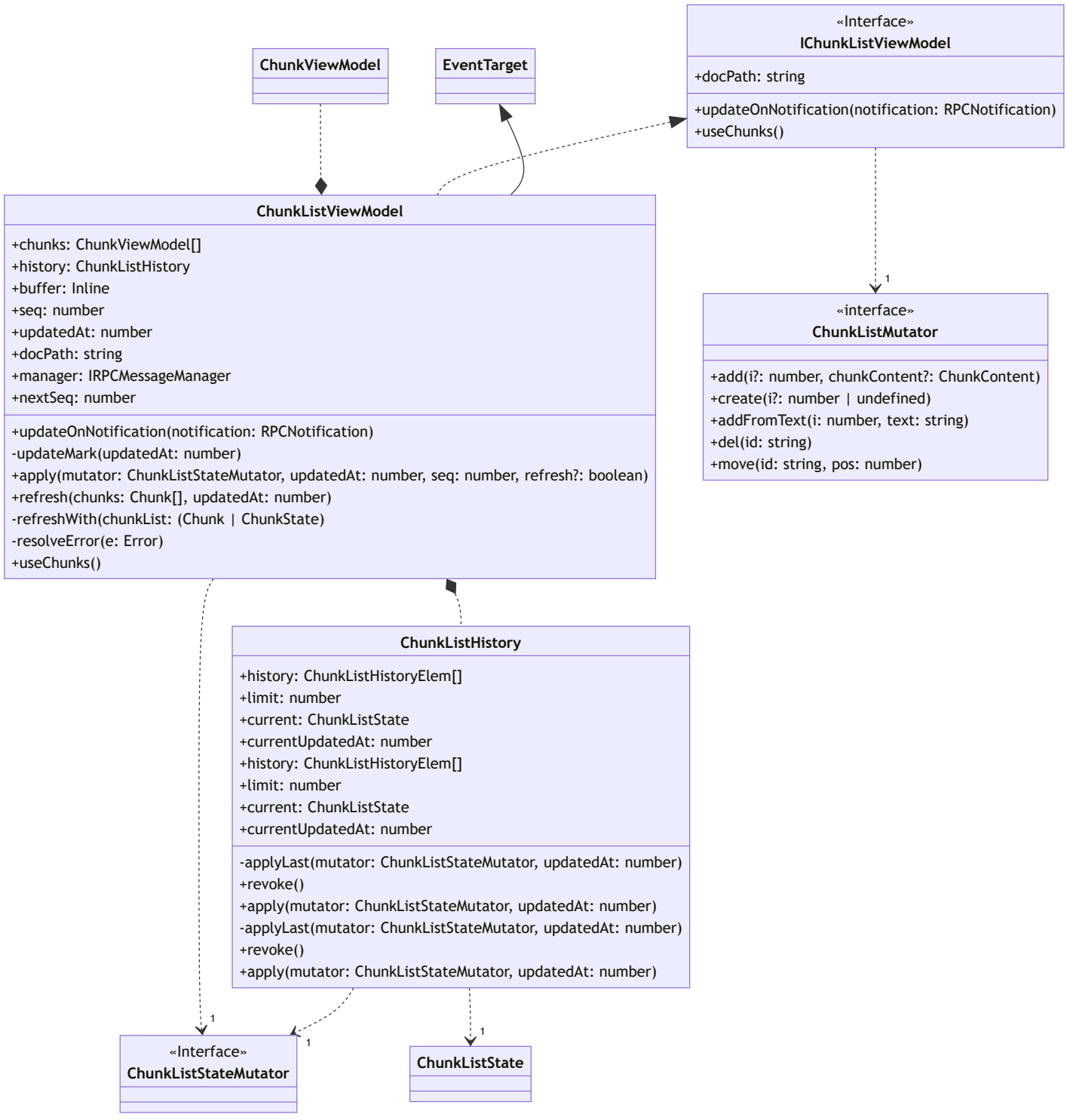

«Interface»
IDisposable
+dispose()
RefCountDisposable
-refCount: number
-handlers:(()
+dispose()
+disposeForced()
+addRef()
+addDisposeHandler(handler: ()
RefCountSet<T>
-items: Map
+size: number
+add(item: T, disposeHandler?: ()
+addDisposeHandler(item: T, handler: ()
+delete(item: T)
+deleteForced(item: T)
+clear()
+has(item: T)
+values()

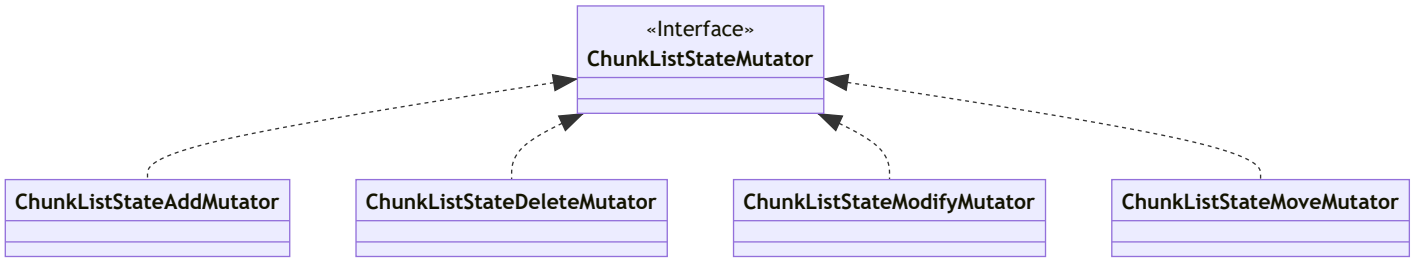

Event
EventTarget
FsWatcherEvent
+paths: string[]
FsWatcher
-path: string
-watcher?: Deno.FsWatcher
-filterFns:((path: string, kind: FsWatchEventType)
+addFilter(fn: (path: string, kind: FsWatchEventType)
+removeFilter(fn: (path: string, kind: FsWatchEventType)
+startWatching()
+stopWatching()
+addEventListener(type: FsWatchEventType, handler: (e: FsWatcherEvent)

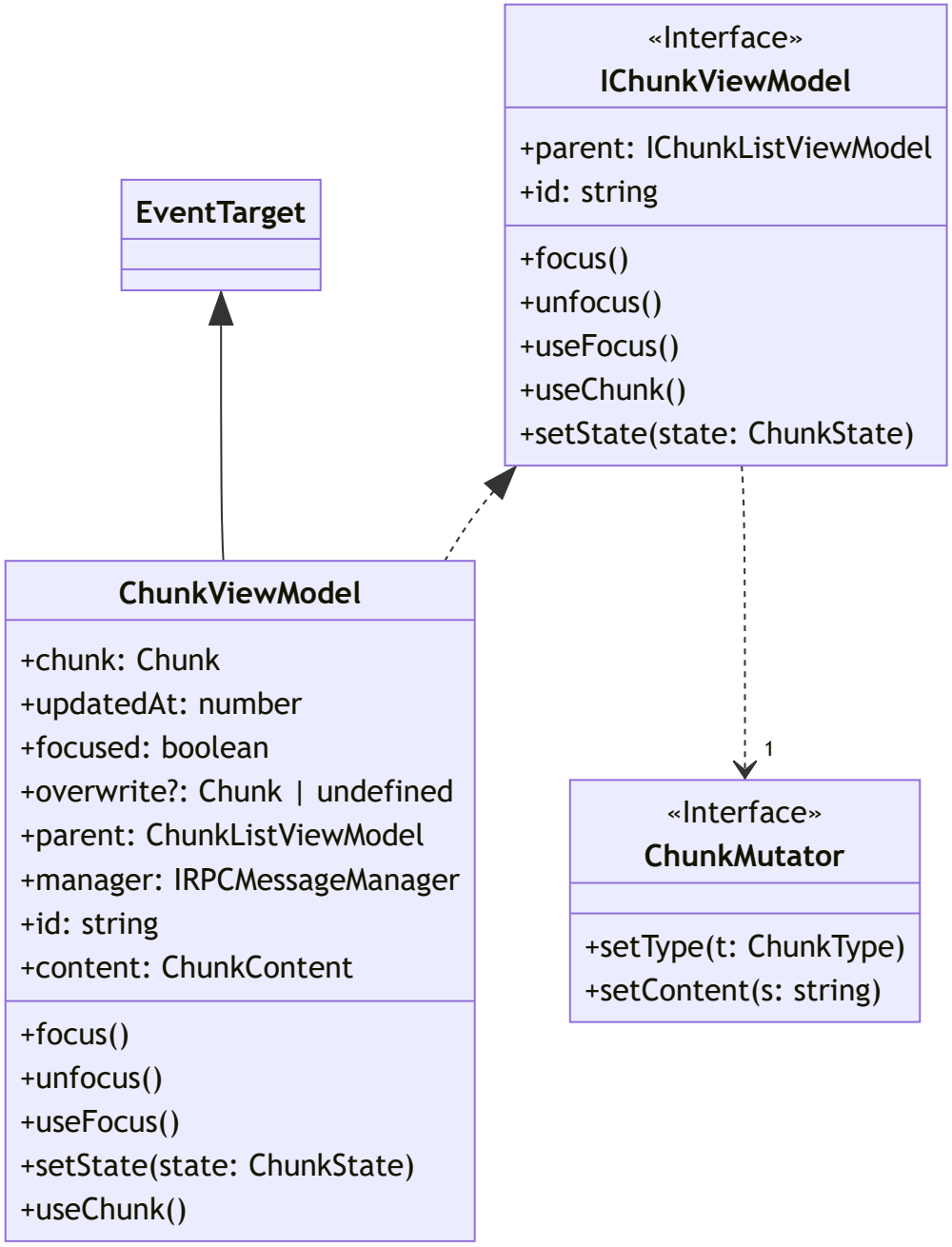

«Interface»
IReadWriter
+read(path: string)
+write(path: string, content: string)
MemoryReadWriter
-data: Record
+read(path: string)
+write(path: string, content: string)
AtomicReadWriter
+read(path: string)
+write(path: string, content: string)
WatchFilteredReadWriter
-raw: AtomicReadWriter
-fsWatcher: FsWatcher
+read(path: string)
+write(path: string, content: string)
QueueReadWriter
-queue: Command[]
+started: boolean
+delayCount: number
+baseReadWriter: IReadWriter
-waitedResolve:()
+read(path: string)
+write(path: string, content: string)
+save(path: string, content: string)
+startTimer()
+flush()
+wait()

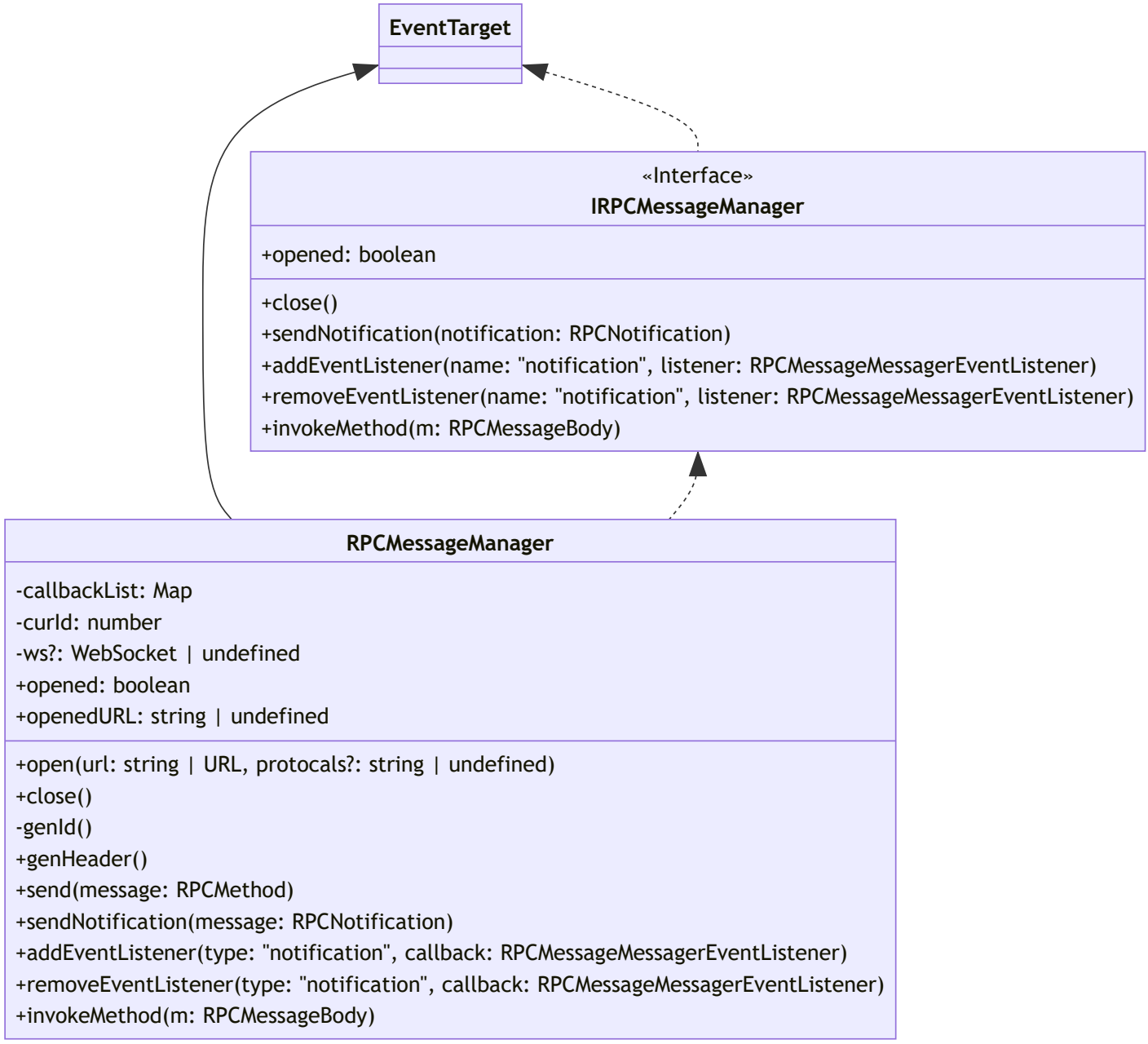
5.1.2 Client Side UML

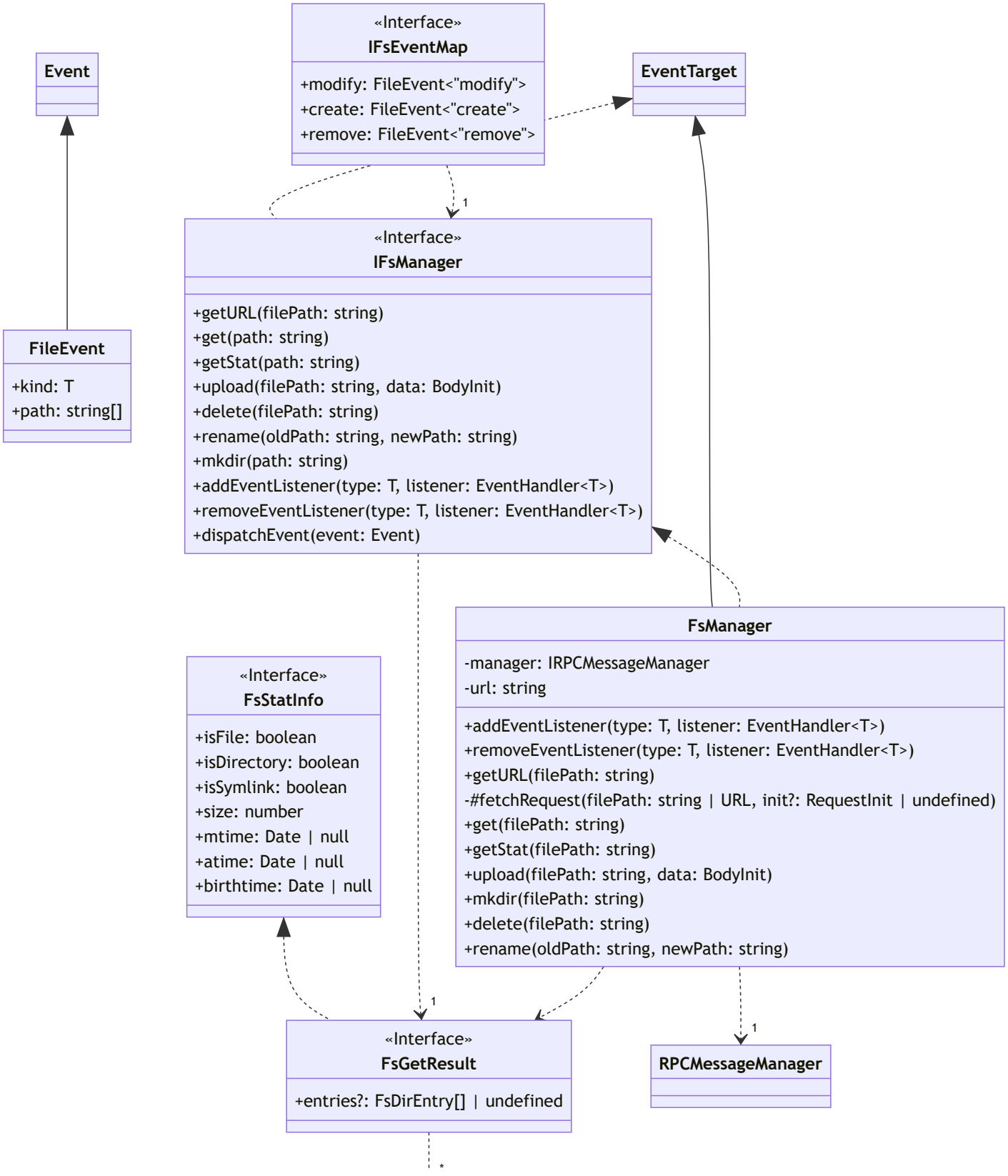


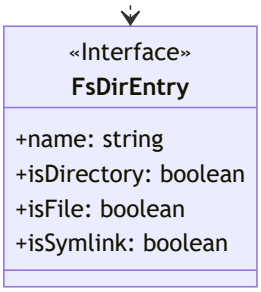

«Interface»
IChunkListViewModel
+docPath: string
+updateOnNotification(notification: RPCNotification)
+useChunks()
ChunkViewModel
EventTarget
ChunkListViewModel
+chunks: ChunkViewModel[]
+history: ChunkListHistory
+buffer: Inline
+seq: number
+updatedAt: number
+docPath: string
+manager: IRPCMessageManager
+nextSeq: number
+updateOnNotification(notification: RPCNotification)
-updateMark(updatedAt: number)
+apply(mutator: ChunkListStateMutator, updatedAt: number, seq: number, refresh?: boolean)
+refresh(chunks: Chunk[], updatedAt: number)
-refreshWith(chunkList: (Chunk | ChunkState)
-resolveError(e: Error)
+useChunks()
«interface»
ChunkListMutator
+add(i?: number, chunkContent?: ChunkContent)
+create(i?: number | undefined)
+addFromText(i: number, text: string)
+del(id: string)
+move(id: string, pos: number)
ChunkListHistory
+history: ChunkListHistoryElem[]
+limit: number
+current: ChunkListState
+currentUpdatedAt: number
+history: ChunkListHistoryElem[]
+limit: number
+current: ChunkListState
+currentUpdatedAt: number
-applyLast(mutator: ChunkListStateMutator, updatedAt: number)
+revoke()
+apply(mutator: ChunkListStateMutator, updatedAt: number)
-applyLast(mutator: ChunkListStateMutator, updatedAt: number)
+revoke()
+apply(mutator: ChunkListStateMutator, updatedAt: number)
«Interface»
ChunkListStateMutator
ChunkListState
1
1
1
1


«Interface»
ChunkListStateMutator
ChunkListStateAddMutator
ChunkListStateDeleteMutator
ChunkListStateModifyMutator
ChunkListStateMoveMutator


«Interface»
IChunkViewModel
+parent: IChunkListViewModel
+id: string
+focus()
+unfocus()
+useFocus()
+useChunk()
+setState(state: ChunkState)
EventTarget
ChunkViewModel
+chunk: Chunk
+updatedAt: number
+focused: boolean
+overwrite?: Chunk | undefined
+parent: ChunkListViewModel
+manager: IRPCMessageManager
+id: string
+content: ChunkContent
+focus()
+unfocus()
+useFocus()
+setState(state: ChunkState)
+useChunk()
1
«Interface»
ChunkMutator
+setType(t: ChunkType)
+setContent(s: string)


EventTarget
«Interface»
IRPCMessageManager
+opened: boolean
+close()
+sendNotification(notification: RPCNotification)
+addEventListener(name: "notification", listener: RPCMessageMessagerEventListener)
+removeEventListener(name: "notification", listener: RPCMessageMessagerEventListener)
+invokeMethod(m: RPCMessageBody)
RPCMessageManager
-callbackList: Map
-curId: number
-ws?: WebSocket | undefined
+opened: boolean
+openedURL: string | undefined
+open(url: string | URL, protocals?: string | undefined)
+close()
-genId()
+genHeader()
+send(message: RPCMethod)
+sendNotification(message: RPCNotification)
+addEventListener(type: "notification", callback: RPCMessageMessagerEventListener)
+removeEventListener(type: "notification", callback: RPCMessageMessagerEventListener)
+invokeMethod(m: RPCMessageBody)


«Interface»
IFsEventMap
+modify: FileEvent<"modify">
+create: FileEvent<"create">
+remove: FileEvent<"remove">
Event
EventTarget
1
«Interface»
IFsManager
+getURL(filePath: string)
+get(path: string)
+getStat(path: string)
+upload(filePath: string, data: BodyInit)
+delete(filePath: string)
+rename(oldPath: string, newPath: string)
+mkdir(path: string)
+addEventListener(type: T, listener: EventHandler<T>)
+removeEventListener(type: T, listener: EventHandler<T>)
+dispatchEvent(event: Event)
FileEvent
+kind: T
+path: string[]
FsManager
-manager: IRPCMessageManager
-url: string
+addEventListener(type: T, listener: EventHandler<T>)
+removeEventListener(type: T, listener: EventHandler<T>)
+getURL(filePath: string)
-#fetchRequest(filePath: string | URL, init?: RequestInit | undefined)
+get(filePath: string)
+getStat(filePath: string)
+upload(filePath: string, data: BodyInit)
+mkdir(filePath: string)
+delete(filePath: string)
+rename(oldPath: string, newPath: string)
«Interface»
FsStatInfo
+isFile: boolean
+isDirectory: boolean
+isSymlink: boolean
+size: number
+mtime: Date | null
+atime: Date | null
+birthtime: Date | null
1
«Interface»
FsGetResult
+entries?: FsDirEntry[] | undefined
1
RPCMessageManager
*



5.2 의사코드(Pseudo Code)

의사코드는 다음과 같이 진행된다.

5.2.1 서버 RPC 메세지 처리

클라이언트는 RPC를 진행하기 위해 웹소켓을 연결한다. 웹소켓의 주소 `/ws`에 도달하기 위해서 먼저 라우팅이 진행이 된다. 라우팅은 `TreeRouter` 클래스에서 진행된다.

```
Input: req Request
Output: Response
fn Route(req){
    for node in HandlerNodes {
        if(req.url prefix is matching){
            node.Handler[matching](req)
        }
    }
    response with 404 Not Found
}
```

마침내 엔드포인트에 도달하게 되면 웹소켓을 얻어내고 새로운 연결을 등록한다. 여기서 reqeust session으로 유저를 얻어내서 접속하는 것에 대해 주의하라.

```
Input: req Request
Output: Response
fn RPCHandleEndpoint(req){
    ws, res = upgradeWebSocket(req)
    user = getUserInfo(req.header)
    conn = new Connection(ws, user)
    registerParticipant(conn)
    res
}
```

이제부터 `RPCMethod` 메세지를 받을 수 있다. 메세지가 오면 함수 `handleMessage` 가 실행이 호출된다.

```
Input: message on send
Output: response
fn Connection.handleMessage(msg: string){
    data = JSON.parse(msg)
    check format of data
    dispatch(data.method, data, this)
}
```

디스패치가 성공적으로 이루어지면 RPC 작업를 처리하는 함수에 도착한다. 청크 작업을 예로 들면 이렇다. 청크 작업에서는 권한을 확인하고 명령의 충돌을 히스토리를 비교하며 다시 적용하며 해결한다. 그리고 요구된 작업을 처리하고 문서의 `updatedAt` 을 업데이트하고 문서 업데이트 사실을 이 문서를 보고 있던 참여자에게 전파한다.

```
Input: conn Connection
Input: m RPCChunkMethod
Output: response

fn ChunkOperation(conn, m){
    doc = DocStore.getDocument(m.docPath);
    updatedAt = m.updatedAt;
    action = getAction(m);
    if !conn.userpermissionSet.canDo(Action) {
        response with PermissionError;
    }

    appliedList = doc.history.filter(x => x.updatedAt > updatedAt)
    for m in appliedList {
        if action.checkConflict(m) {
            resolvedAction = action.tryResolveConflict(m)
            if resolvedAction == Fail {
                response with ConflictError
            }
            action = resolvedAction
        }
    }

    res = action.act(doc);
    doc.updateHistory(res);

    subscribers = doc.getSubscribers();
    subscribers.broadcastNotification(m, exclude = conn);
    response with res
}
```

문서 작업도 마찬가지로 이루어진다.

5.2.2 클라이언트의 메세지 처리 동기화

클라이언트에서는 다음과 같은 일이 일어난다.

먼저 `notification`을 받는다. 그러면 모든 `DocumentViewModel`에게 이벤트를 전달한다. 그리고 각각의 `DocumentViewModel`은 자기 문서에 일어난 일인지 확인하고 `ChunkListMutator`를 만들어서 문서에 적용한다.

```
Input: e RPCNotification
Input: this document view model

fn updateOnNotification(this,notification){
    { docPath, method, seq, updatedAt } = notification.params;
    if (docPath !== this.docPath) return;
    mutator =  ChunkListMutatorFactory.createFrom(method);
    this.apply(mutator, updatedAt, seq);
}
```

문서의 레디큐에 mutator를 집어넣고 `seq` 번호가 기다리는 것이면 실행하고 업데이트한다.

```
Input: mutator ChunkListMutator
Input: updatedAt Date
Input: seq number
Input: this Document View Model

//readyQueue is priority queue.
fn apply(this, mutator, updatedAt, seq){
    this.readyQueue.push({mutator, updatedAt, seq})

    if(this.readyQueue.lenght < some limit){
        while(!readyQueue.empty() &&
            this.readyQueue.top().seq === this.seq + 1)
        {
            mutator, updatedAt, seq = this.readyQueue.pop();
            mutator(this.chunks);
            this.seq = seq;
            this.updatedAt = updatedAt;
        }
        this.dispatch(new Event("chunksChange"));
    }
    else {
        document.refresh();
    }
}
```

5.2.3 다른 작업들

다음은 청크 컴포넌트에 관한 의사코드이다.

```
module chunk {
  type mode = Read | Write

  struct Chunk {
    id: string
    content: string
    type: string
  }

  newChunk() {
    { id      = uuid()
    ; content = ""
    ; type    = ""
    }
  }

  chunkViewer(chunk : Chunk, focusedChunk : State<string>,
   deleteThis : () => void) : Component {
    var mode = Read

    var c = new Component(
      content { value = chunk.content }
      settypebutton
      editbutton
      deletebutton
    )

    when mode becomes Read { chunk.content = content }
    when mode becomes Write { focusedChunk = chunk.id }
    when focusedChunk is changed { mode = Read }

    when editbutton is clicked { mode = (mode = Read) ? Write : Read }
    when deletebutton is clicked { deleteThis() }
    when settypebutton is clicked { chunk.type = prompt() }

    return c
  }
}
```

다음은 검색 기능에 관한 의사코드이다.

```
module search {
  searchWord(chunks, word) {
      return doc.chunks.concat_map((s) => s.matchAll(word))
  }

  searchWordPrompt(chunks: Chunk.chunk list) {
    var word = prompt()
    var results = searchWord(chunks, word)

    var c = new Component(results)

    when result in results is selected {
      moveto(result.location)
      close()
    }
  }

  when Ctrl-F is pressed { searchWordPrompt() }
}
```

다음은 문서 컴포넌트에 관한 의사코드이다.

```
module document {
  struct Document {
    title: string
    path: Path
    tags: string set
    chunks: chunk.Chunk array
  }

  documentViewer(doc: document) : Component {
    var focusedChunk = null

    var c = new Component(
      taglist { value: tags }
      chunklist
    )

    delete(id) {
      i = doc.chunks.find((c) => c.id = id)
      doc.chunks.remove(i)
    }

    chunklist = doc.chunks.concat_map((c, i) =>
      [ divider(i), chuknViewer(c, focusedChunk, () => delete(c.id)) ])

    when divider(i) clicked { doc.chunks.insert(i, c) }
    when chunkViewer(c) is dropped on divider(i) { doc.chunks.move(c, i) }

    return c
  }
}
```

다음은 파일리스트 컴포넌트에 관한 의사코드이다.

```
module filelist {
  fileList(dir : Directory, open: (File) => void) : Component {
    var c = new Component(
      filelist
    )

    filelist = dir.files().map((f) => button(f))

    when button(f) is clicked {
      open(f)
    }

    return c
  }
}
```

다음은 설정에 관한 의사코드이다.

```
module settings {
  settings() : Component {
    var c = new Component(
      language = select("korean", "english")
      theme = select("light", "dark")
    )

    when language(l) is selected {
      global context.lang = l
    }

    when theme(t) is selected {
      global context.theme = t
    }

    return c
  }
}
```

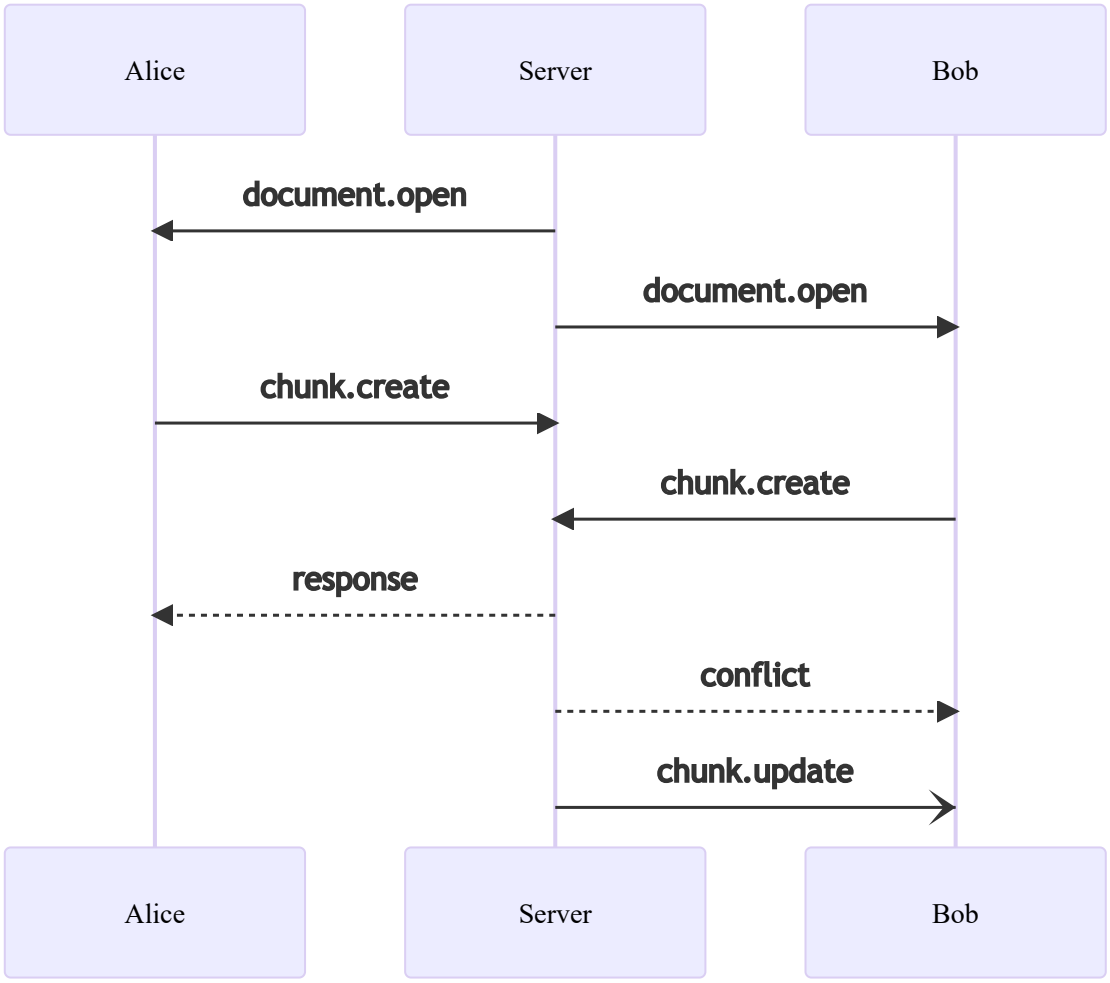
다음은 프론트엔드를 요약하는 의사코드이다.

```
module frontend {
  main() : Component {
    var docv

    var open = (f) => {
      with doc = openfile(f) {
        docv = document.documentViewer(doc)
      }
    }

    var filelist = filelist.fileList(rootdir, open)

    var c = new Component(
      document = docv
      filelist = filelist
    )

    return c
  }
}
```

5.3 동기화 정책(Synchronization Policy)

동기화 정책은 이렇다. 기본적으로 상대문서의 갱신 시간과 나의 갱신 시간을 비교하고 상대문서가 최신이 아니면 거부한다. 메소드가 실행 순서에 무관하면 언제나 실행하고 갱신한다. 메소드가 순서에 따라 조정될 수 있으면 조정한다.

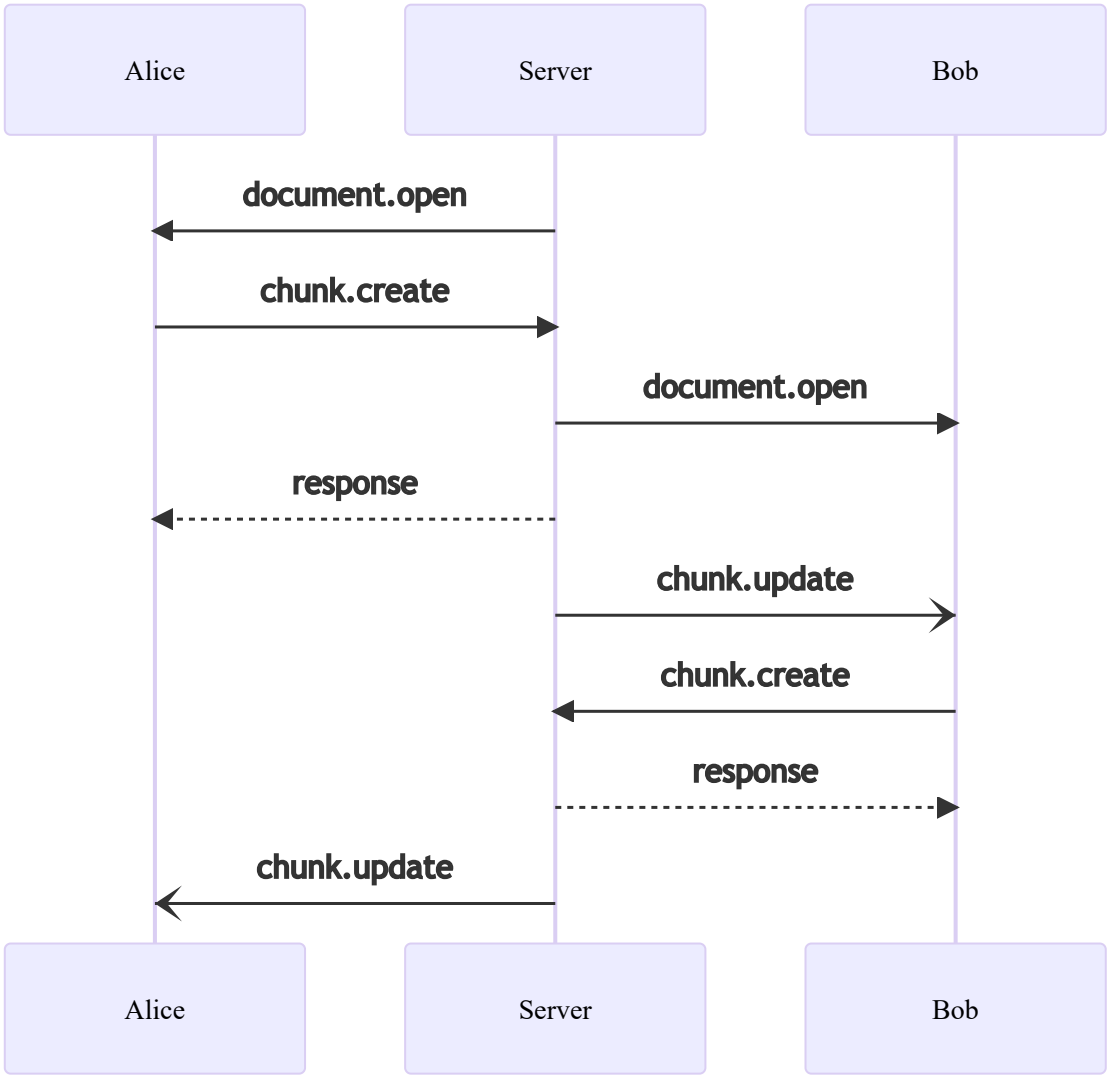
예를 들어, 다음과 같은 상황이 있다.



처음에 Server에서 Alice와 Bob이 문서를 받아왔다. 그리고 Alice가 `chunk.create` 메소드를 보내고 Bob이 `chunk.create` 메소드를 보냈다. 그러면 Server는 Alice가 먼저 도착했으므로 Alice의 메소드을 처리하고 최근 갱신 시간을 갱신한다. 그리고 Bob의 메소드를 처리할때 Bob의 문서의 최근 갱신 시간이 Server 문서의 최근 갱신 시간보다 작으므로 거부한다.

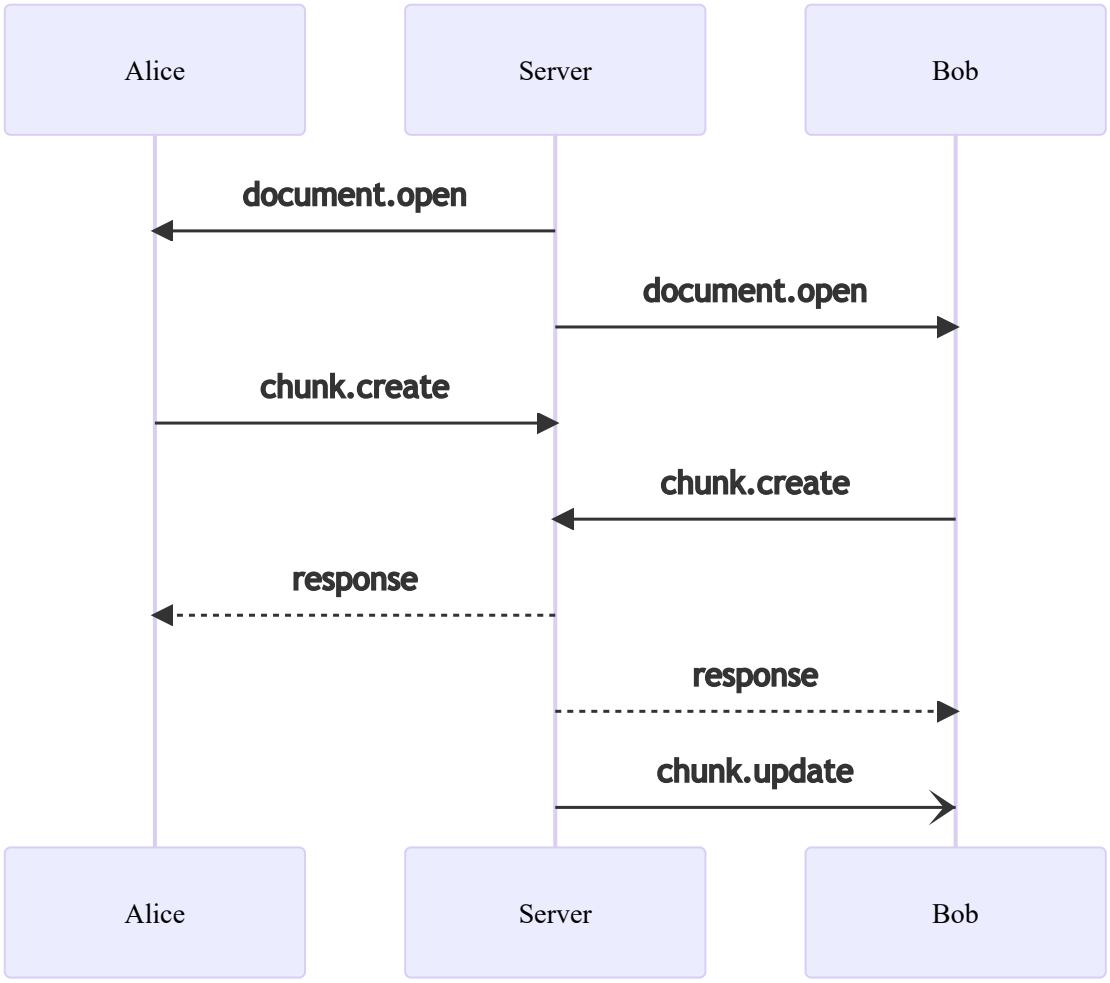
별개로 Alice의 갱신사실을 알리기 위해서 Bob에게 `chunk.update`를 보낸다.

다른 예로 다음과 같은 상황에서는 이렇다.



여기서는 Bob의 문서가 최신이기 때문에(**문서 갱신 시간이 Server와 같기 때문에**) 거부되지 않고 처리되는 것을 볼 수 있다. 그와 별개로 갱신 사실을 알리기 위해서 `chunk.update`가 Alice와 Bob에게 전달되는 것을 볼 수 있다.

메소드를 조정할 수 있으면 조정한다는 것은 인수를 바꾸어서 실행한다는 것이다. 예를 들어 다음의 예에 대해서



A는 1번째 위치에서 `chunk.create`에서 시도하고 B가 3번째 위치에서 `chunk.create`를 시도하면 A가 B보다 먼저 앞의 위치에서 삽입을 했으므로 B의 위치를 4번째로 조정하고 적용한다.

6. 시험(Testing)

6.1 유닛 테스트(Unit test)

유닛 테스트로 63.7%의 Line Coverage와 67.5%의 Function Coverage, 61.5%의 Branch Coverage를 달성했다.

permission.test.ts

name	result	duration
permission.test	✅	14ms
permission empty	✅	16ms

session.test.ts

name	result	duration
Session : set	✅	14ms
Session : delete	✅	16ms
Login Handler : login with invalid format	✅	15ms
Login Handler : login with invalid password	✅	16ms
Login Handler : login	✅	15ms
Login Handler : logout with no session	✅	17ms
Login Handler : logout	✅	16ms
getSession	✅	15ms
getSession with invalid cookie	✅	15ms

user.test.ts

name	result	duration
user.createAdminUser	✅	5ms

filedoc.test.ts

name	result	duration
readDocFile	✅	12ms
readDocFile: not found	✅	15ms
readDocFile: invalid json	✅	15ms
saveDocFile	✅	15ms

methodHandle.test.ts

name	result	duration
methodHandle: basic methods	✅	14ms
methodHandle: not found	✅	15ms
methodHandle: options	✅	17ms

route.test.ts

name	result	duration
route: basic route	✅	6ms
route: double slash route	✅	12ms
route: double match	✅	16ms
route: test context	✅	15ms
route: test regex	✅	15ms
route: test not found	✅	2ms
route: encode_route	✅	12ms
route: router in router	✅	15ms

chunk.test.ts

name	result	duration
basic chunk operation : create chunk	✅	13ms
basic chunk operation : delete chunk	✅	15ms
basic chunk operation : modify chunk	✅	16ms

name	result	duration
basic chunk operation : move chunk	✅	16ms
basic chunk operation : invalid chunk operation	✅	16ms
test chunk notification operation	✅	16ms
test chunk conflict	✅	15ms
test chunk conflict resolve with history	✅	32ms

doc.test.ts

name	result	duration
handleDocumentMethod	✅	5ms
handleTagMethod : setTag	✅	9ms
handleTagMethod : getTag	✅	16ms
handleTagMethod : conflict	✅	16ms

share.test.ts

name	result	duration
handleShareGetInfo	✅	16ms
handleShareDocMethod	✅	2ms
handleShareMethod with no existing share token	✅	12ms

server.test.ts

name	result	duration
server rpc test	✅	4s

fswatcher.test.ts

name	result	duration
WatchFilteredReadWriter	✅	79ms

readWriter.test.ts

name	result	duration
QueueReadWriter	✅	57ms

util.test.ts

name	result	duration
watcher util isHidden	✅	5ms

Total

name	passed	Steps	Failed	duration
ok	35	15	0	4726ms

6.2 기능 테스트(Functional Test)

Chunk

ID	Content	Procedure	Test Data	P/F
1	Focus/Unfocus	1. 청크를 클릭한다.		✅
2	remove	1. 청크를 삭제하는 버튼을 클릭한다.		✅
3-1	render - markdown	1. 마크다운 청크 렌더링을 확인한다.	# 제목	✅
3-2	render - latex	1. LaTex 청크 렌더링을 확인한다.	sum^n_{n=0}n = \frac{n(n+1)}2$$	✅
3-3	render - link	1. Image 청크 렌더링을 확인한다.	http://picsum.photos/200/300	✅

ID	Content	Procedure	Test Data	P/F
4	previews	1. Katex 청크의 미리보기를 본다.	sum^n_{n=0}n = \frac{n(n+1)}2$$	✅
10	autocomplete	1. 자동완성을 시험한다.		❌
11	swap positions	1. 청크의 위치를 바꾼다.		✅
27-1	edit	1. 청크를 수정한다.		✅
27-2	edit chunk conflict	1. 청크를 수정 모드에 들어간다.		❌

Document

ID	Content	Procedure	Test Data	P/F
5	view Chunk	1. 문서를 열어 청크가 렌더링되는지 본다.	test.syd	✅
7	add/delete tag	1. 문서에 태그를 추가한다. 2. 문서에 태그를 삭제한다.	A	✅
8	Drag And Drop Upload,	1. 텍스트를 드래그한다.		✅

File

ID	Content	Procedure	Test Data	P/F
14	create/delete/rename file	1. 파일을 만든다.	test.txt	✅
15	upload/download files	1. 파일을 업로드한다.	test.txt	✅
18	export document	1. export 버튼을 누른다.		❌

Search

ID	Content	Procedure	Test Data	P/F
16	Document Search	1. 검색해본다.		❌

ID	Content	Procedure	Test Data	P/F

Stash

ID	Content	Procedure	Test Data	P/F
17	render	1. 스태시가 그려지는지 확인한다		✅
19	add	1. 청크를 추가한다		✅
20	remove	1. 청크를 삭제한다		✅
21	Drag and Drop to Document	1. 청크로부터 문서로 청크를 옮긴다.		✅

Management

ID	Content	Procedure	Test Data	P/F
22	Login	1. 비밀번호를 입력한다.	admin	✅
24	Localization	1. 다른언어를 지원하는지 언어를 바꿔 확인한다		❌
25	Theme	1. Theme를 바꾸어서 제대로 변경되는지 시험해본다.		✅