웹 문서 뷰어(미리보기) 시스템 아키텍처 및 고도화 분석 보고서

클라이언트 사이드 렌더링 엔진 분석, 기술 사유 및 한계 대응 전략

문서 분류	기술 분석 및 아키텍처 보고서	**작성일자**	2026년 6월 19일
대상 시스템	프로젝트 관리 시스템(PMS) 문서 뷰어	**작성자**	시스템 아키텍트 팀

1. 개요 및 시스템 목적

본 보고서는 사내 프로젝트 관리 시스템(PMS) 내 첨부파일의 대략적인 내용을 다운로드 없이 웹 브라우저 상에서 즉각 확인할 수 있도록 구축된 **웹 문서 미리보기 시스템**의 아키텍처, 확장자별 적용 기술 및 사유, 클라이언트 단의 구현 방식, 그리고 태생적인 기술적 한계와 그에 대한 현실적인 대응 방안을 체계적으로 정리한 문서입니다.

본 시스템은 서버 리소스를 최소화하고 사용자에게 빠른 응답 속도를 제공하기 위해 **클라이언트 사이드 직접 렌더링 (Client-Side Rendering, CSR)** 방식을 주 축으로 채택하였으며, 직접 파싱이 불가능하거나 복잡한 포맷에 대해서는 **서버 사이드 PDF 변환 폴백(Fallback)** 또는 전문 오피스 서버 연동 구조를 유기적으로 결합하여 상시 지원 체계를 구축하였습니다.

2. 문서 뷰어 확장자별 지원 사양 및 기술 사유

시스템에서 감지 및 렌더링하는 방식은 파일 포맷의 스펙과 보안성, 그리고 라이선스 비용 등을 고려하여 크게 세 가지 파이프라인으로 분기 처리됩니다.

확장자 분류	적용 기술 및 뷰잉 방식	기술 적용 사유 및 특이사항
PDF	자체 `PDF.js` 오픈소스 뷰어로 브라우저 직접 출력	표준 문서 포맷으로서 별도의 변환 작업 없이 100% 원본 렌더링 및 고해상도 출력이 가능함.
HWP / HWPX	`hwp.js` 기반 직접 파싱 및 뷰잉 + LibreOffice PDF 변환 폴백	서버 리소스 절약을 위해 일차적으로 브라우저 단에서 직접 그리며, 복잡한 레이아웃 겹침 발생 시 상단 "PDF로 보기" 연동을 제공함.
DOCX	`docx-preview` 라이브러리 직접 뷰잉 + OfficeToPDF 변환 폴백	웹 표준 HTML 태그 구조로의 변환이 용이하여 직접 뷰잉을 제공하며, 서식 누락에 대비한 고정밀 PDF 보기 버튼을 배치함.
XLSX / XLS / XLSM	`LuckyExcel` / `Luckysheet` 결합 시트 뷰잉 + OfficeToPDF 변환 폴백	텍스트 데이터와 셀 그리드를 완벽히 재현하기 위해 그리드 렌더러를 도입함. 단, 부유형 도형(Shapes) 표현 한계를 보완하기 위한 PDF 뷰잉 폴백을 지원함.
DOC / PPT / PPTX	백엔드 `OfficeToPDF` 서버를 통한 PDF 자동 변환본 로드	구형 이진(Binary) 포맷 및 PPT의 복잡한 슬라이드 그래픽은 브라우저 자바스크립트 단독으로 레이아웃 처리가 불가능하므로 고해상도 PDF 변환 렌더링이 필수적임.

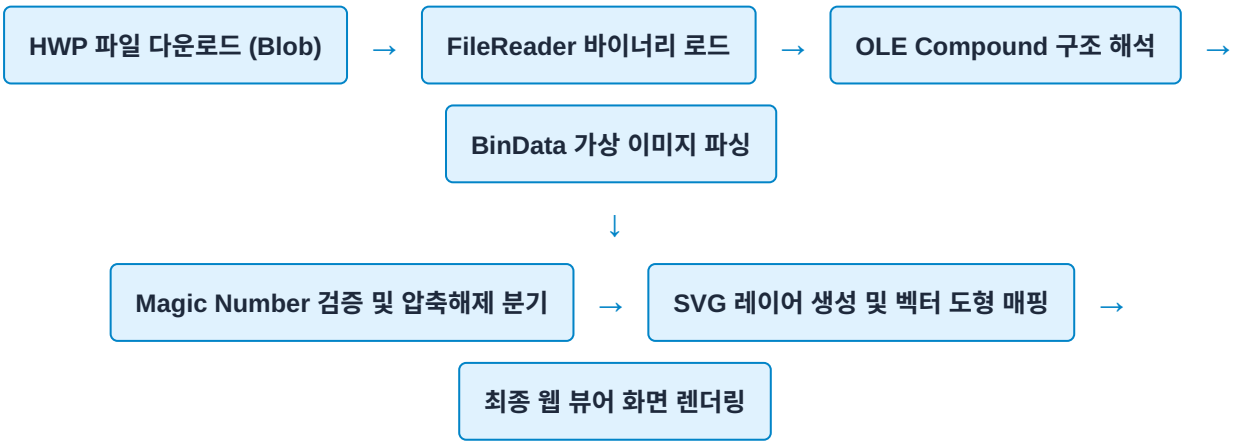
※ **미리보기 미지원 확장자 및 원천 차단 기술 사유**

- **실행 파일 (exe, msi, bat, sh):** 크로스 사이트 스크립팅(XSS) 및 악성코드 실행 방지를 위해 보안상 브라우저 샌드박스 내부 렌더링이 원천 차단됩니다.
- **압축 파일 (rar, 7z, tar, alz):** .zip을 제외한 독점 포맷은 알고리즘이 무겁고, 기가바이트 단위 대용량 파일 해제 시 브라우저 메모리 고갈(탭 다운)을 유발합니다.
- **데이터베이스 (db, sqlite, sql, accdb):** 실시간 파일 락킹(File Locking) 및 커넥션 세션 관리가 필요하며 민감 정보 유출 위험이 커 다운로드만 허용합니다.

3. 클라이언트 사이드 미리보기 핵심 구현 방식

가장 기술적 난이도가 높았던 **한글(HWP) 파일의 클라이언트 단 직접 파싱 및 렌더링 고도화** 를 기준으로 정립된 핵심 구현 로직과 아키텍처는 다음과 같습니다.

3.1 아키텍처 및 데이터 흐름



3.2 핵심 이슈 해결 및 코드 명세

1. **OLE 이미지 압축 해제(Decompression) 안정화 및 우회**

   기존 파서 라이브러리는 무조건 `pako.inflate({ windowBits: -15 })`를 실행하여 무압축 Raw 바이너리가 OLE 스트림에 기록된 경우 뷰어가 크래시되는 치명적 결함이 존재했습니다. 이를 해결하기 위해 파일 상위 4바이트 *Magic Number* 시그니처를 검증해 표준 이미지(PNG, JPEG, GIF, WMF)일 경우 압축 해제를 우회하는 로직을 구축하였고, 3단계 예외 처리 폴백(`windowBits: -15` → 표준 `inflate` → `inflateRaw` → 원본 복사)을 설계했습니다.

2. **MIME 타입 조정 및 바이트 유실 방지**

   기존 소스코드 내 오타(`images/png`)로 인한 브라우저 엑스박스 출력 오류를 표준 규격(`image/png`, `image/jpeg`)으로 전면 교정하였으며, 데이터 가공 과정에서 문자열이 깨지는 현상을 방지하기 위해 Blob 생성 직전 `Uint8Array` 인스턴스로 안전하게 바인딩되도록 강제했습니다.

3. **SVG 배경 레이어 적용을 통한 자체 생성 도형(Shape) 복원**

   문서 편집기 내 직사각형(Rectangle), 타원(Ellipse), 선(Line) 등의 벡터 도형 누락 문제를 해결하기 위해, 순회 엔진에 태그 ID를 추가 등록하고 부모 컨테이너 내부에 절대 배치된 `<svg>` 레이어를 생성하여 렌더링했습니다. 또한 세로 기준 속성(***vertRelTo***)을 명확히 파악하여 문단 기준(***vertRelTo = 2***)인 경우 부모 `div`에 `position: relative`를 주입하고, 좌표 오프셋을 ***position: absolute***로 매핑하여 페이지 바깥으로 도형이 밀려나는 현상을 교정했습니다.

4. **글로벌 CSS 충돌 조치 (줄 간격 보정)**

   프로젝트 공통 스타일시트의 간섭으로 문단 글씨가 위아래로 심하게 겹치던 가독성 오류를 해결하기 위해 개별 문자열 스팬 요소에 `line-height: 1.65` 인라인 스타일을 강제로 선언하여 스타일 오버라이드를 원천 차단했습니다.

5. **기타 호환성 처리**

클라이언트 OS 환경에 따른 한글 서체 유실 문제를 방지하기 위해 `@font-face` 문법으로 Google Noto 웹 폰트 매핑 체계를 반영했으며, 브라우저 표준 디코더가 읽지 못하는 구형 `.wmf` 벡터 그래픽 파일은 `wmf.js`를 연동해 HTML5 Canvas에 디코딩한 뒤 PNG로 치환하여 정상 노출시켰습니다.

4. 기술적 한계 원인 분석

클라이언트 사이드 직접 파싱 방식은 서버에 부하를 주지 않고 오프라인 상태에서도 빠른 속도로 구동되는 극강의 장점이 있으나, 근본적으로 다음과 같은 기술적 한계를 내포하고 있습니다.

4.1 HWP 포맷의 비공개 독점 규격성

한글(HWP) 파일 포맷은 국제 표준이 아닌 한글과컴퓨터사의 독점 바이너리 스펙입니다. 한컴에서 일부 공개한 `HWP 5.0 File Format Specification` 문서는 전체 스펙의 60~70% 수준에 불과하며, 정밀한 도형 속성, 수식, 필드 코드 및 고급 서식 세부 정보는 비공개 영역으로 남아있습니다. 오픈소스 커뮤니티의 역공학(Reverse Engineering)에 의존하는 현재의 `hwp.js` 라이브러리로는 원본 프로그램 수준의 100% 완벽한 복원이 구조적으로 불가능합니다.

4.2 엑셀 파일 도형(Shapes) 렌더링의 공수 한계

엑셀(.xlsx) 미리보기에 사용 중인 `LuckyExcel`과 `Luckysheet`는 세포(Cell) 중심의 데이터 그리드 표현에 특화되어 개발되었습니다. 시트 내 그리드 위에 공중 부양 형태로 독립 배치되는 드로잉 도형 객체(화살표, 프로세스 차트 등) 정보는 `xl/drawings/drawingX.xml` 내부에 **DrawingML** 규격으로 기록되나, 파서 단에서 이를 완벽히 무시하고 있습니다.

이를 클라이언트 단에서 직접 구현하려면 OpenXML 규격을 파싱하는 DrawingML Parser를 밑바닥부터 독자 개발해야 할 뿐만 아니라, 셀의 너비와 높이 변화에 유기적으로 가산 연동되는 **동적 셀-픽셀 좌표 매핑 엔진(Two Cell Anchor 구조 대응)**을 설계해야 하므로 웹 오피스를 완전히 새로 개발하는 수준의 과도한 리소스(공수)가 요구됩니다.

5. 한계에 대한 최선 대응 방안

구조적 한계를 극복하고 업무 시스템의 영속성과 데이터 정합성을 보장하기 위해 다음과 같은 단계별 대응 전략을 수립하여 적용 중입니다.

■ 전략적 대응 핵심 방안 (Multi-Layered Strategy)

[1단계] 하이브리드 UI 배치 (현재 적용 완료)
클라이언트 뷰어의 빠른 응답성을 기본 값으로 제공하되, 화면 상단에 **"PDF로 보기 (원본 고정밀 보기)" 변환 폴백 버튼**을 직관적으로 배치했습니다. 사용자가 복잡한 수식이나 도형이 누락되었음을 감지했을 때 이 버튼을 누르면 서버 사이드 백엔드 엔진이 구동되어 고화질 PDF 변환본을 화면에 재로드합니다.

[2단계] 백엔드 무인 자동 변환 엔진 구축
리눅스 서버 인프라에 `LibreOffice` 및 `OfficeToPDF` 프로세스를 백그라운드 서비스 데몬으로 탑재했습니다. 클라이언트가 특정 문서를 타겟팅하여 원본 보기를 요청하는 즉시 CLI 명령을 통해 오차 없는 표준 PDF 문서로 실시간 변환 연동되도록 조치했습니다.

[3단계] 차세대 설치형 전용 웹오피스 서버 도입 (중장기 권고안)
MS 오피스 및 한글 문서군의 완벽한 웹 협업 및 재현력 확보를 위해 사내 가상화(Docker) 인프라 내에 오픈소스 `OnlyOffice Document Server`를 독립 개설하여 iframe 형태로 연동하는 방안을 추진합니다. 이는 변환 대기 시간을 소거하고 웹상 실시간 편집 및 공동 작성 기능까지 확장 가능한 중대형 인트라넷을 위한 최상위 솔루션입니다.

6. 결론 및 유지보수 가이드

본 문서 뷰어 시스템은 클라이언트 사이드 엔진 고도화를 통해 가로 스크롤바를 소거하고 폰트 불일치 결함 및 SVG 배경 벡터 레이어를 성공적으로 안착시켜 일차적인 가독 레이아웃 일치율을 비약적으로 향상시켰습니다.

향후 운영 단계에서 스타일시트(`font.css`, `hwp.js`) 등이 업데이트될 경우 브라우저의 강력한 클라이언트 캐싱 메커니즘으로 인해 소스 코드가 반영되지 않는 현상이 일어날 수 있으므로, 반드시 배포 프로세스 상에 **파일 버전 쿼리 스트링(e.g., `hwp.js?v=20260619`)**을 결합하여 강제 리프레시되도록 관리해야 합니다. 정밀한 도면이나 계약서 등의 최종 검증 단계에서는 반드시 시스템 상단의 'PDF로 보기'를 활용하도록 사용자 가이드를 배포할 것을 권고합니다.