test

BookStack PDF Export 전처리 파이프라인 분석

> **대상**: BookStack v25.02.4 커스터마이징 참여 개발자 **범위**: Mermaid 다이어그램이 포함된 페이지의 PDF 내보내기 경로 **최종 수정**: 2026-03-24

1. 개요

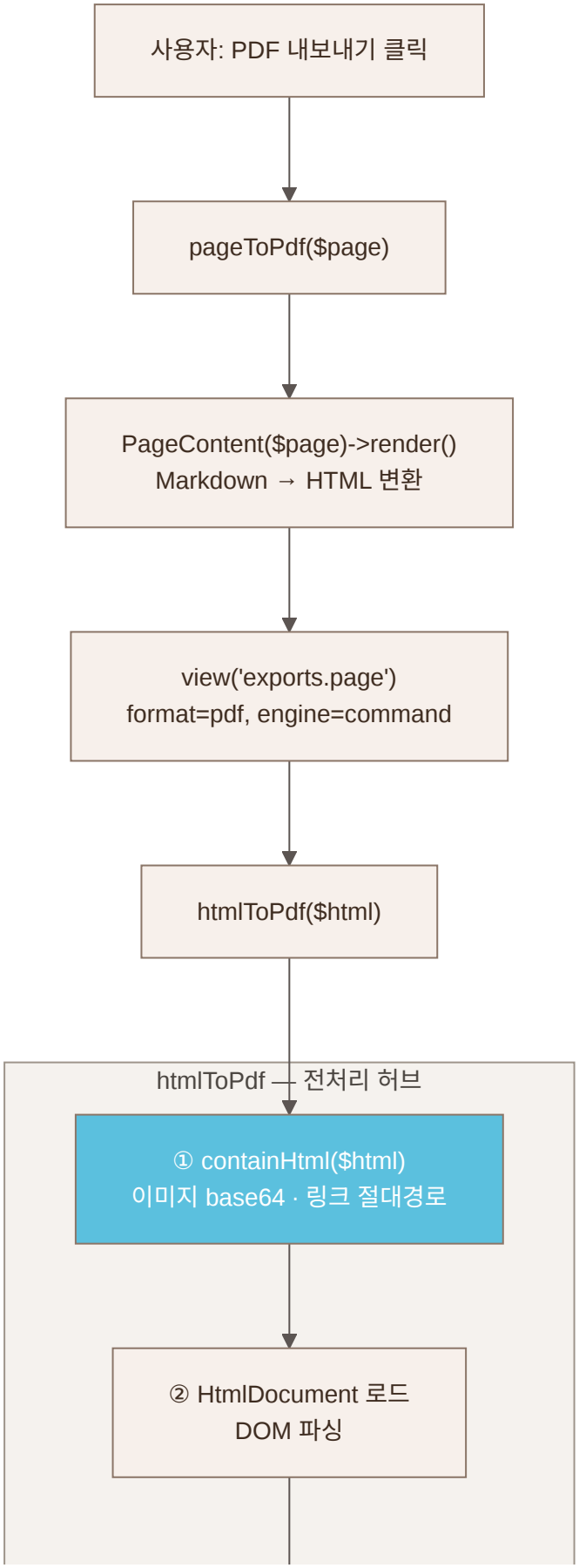
BookStack의 PDF 내보내기는 **WeasyPrint**(CSS 기반 PDF 렌더링 엔진)를 사용한다. WeasyPrint는 브라우저가 아니기 때문에 JavaScript를 실행할 수 없고, SVG 내부의 `<foreignObject>`도 지원하지 않는다. 이 제약이 Mermaid 다이어그램 PDF 출력의 핵심 문제였으며, 최종적으로 **PNG 이미지 방식**으로 해결했다.

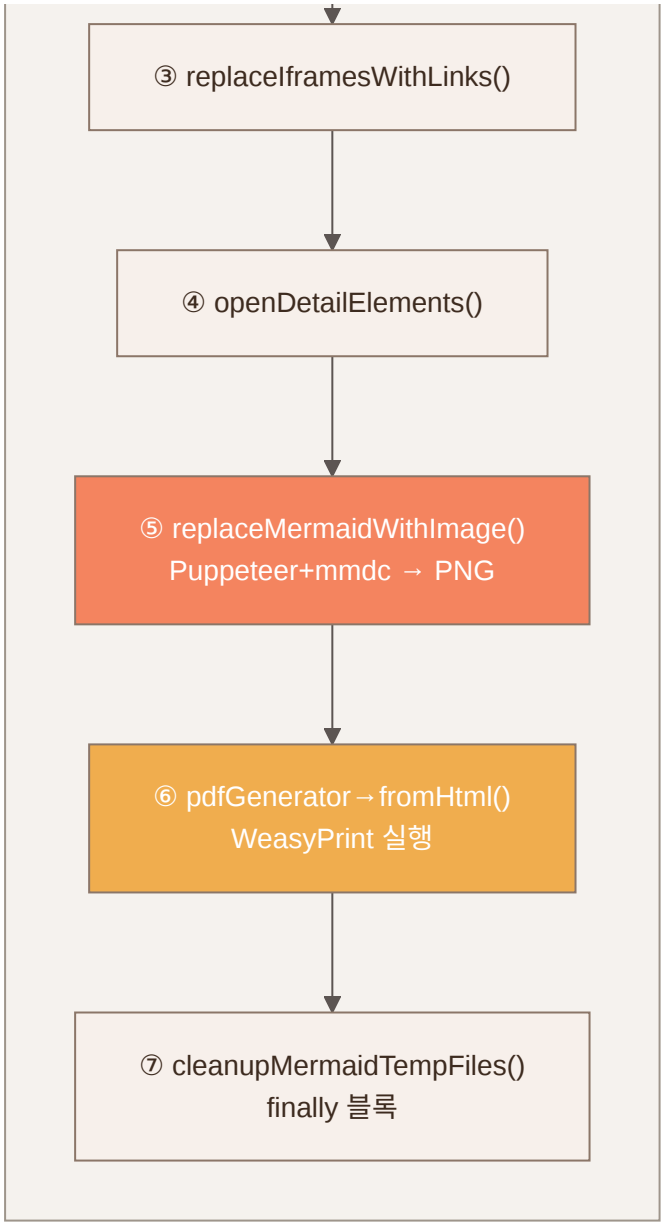
이 문서는 PDF export 요청이 들어왔을 때 HTML이 어떤 전처리를 거쳐 WeasyPrint에 도달하는지를 단계별로 분석한다.

2. 전체 파이프라인 흐름

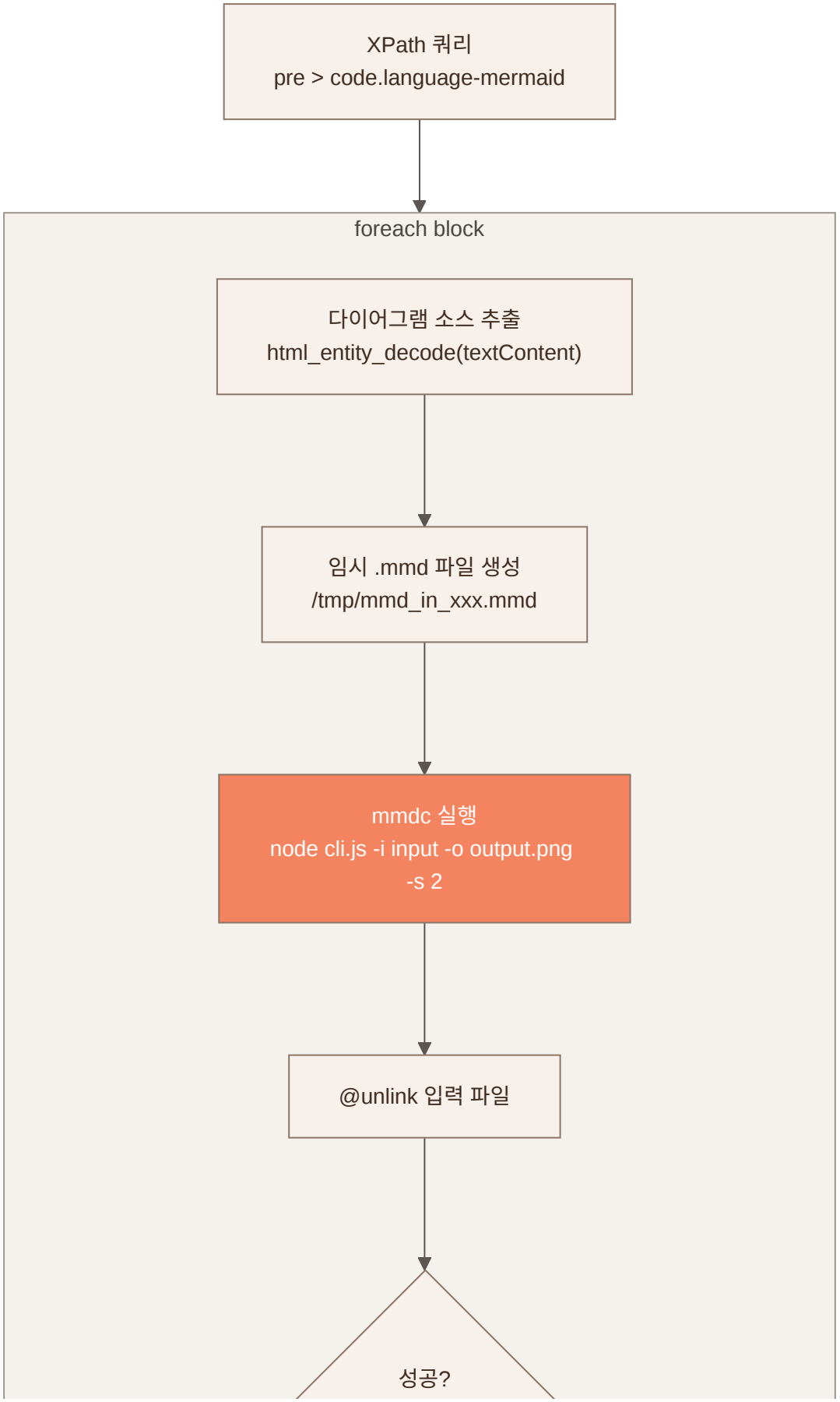
```
사용자 → PDF 내보내기 클릭
  |
  ▼
pageToPdf($page)                          [ExportFormatter.php]
  |
  ├─ PageContent($page)->render()          Markdown → HTML 변환
  |
  ├─ view('exports.page')                  Blade 템플릿 렌더링
  |    └─ format=pdf, engine=command         (CSS/레이아웃 포함)
  |
  └─ htmlToPdf($html)                     ★ 전처리 시작
       |
       ├─ 1. containHtml($html)            이미지 base64, 링크 절대경로
       |
       ├─ 2. HtmlDocument 로드             DOM 파싱
       |
       ├─ 3. replaceIframesWithLinks()     iframe → 링크 텍스트
       |
       ├─ 4. openDetailElements()          <details> 태그 열기
       |
       ├─ 5. replaceMermaidWithImage()     ★ Mermaid → PNG 변환
       |
       ├─ 6. pdfGenerator->fromHtml()      WeasyPrint 실행
       |
       └─ 7. cleanupMermaidTempFiles()     임시 PNG 삭제 (finally)
```

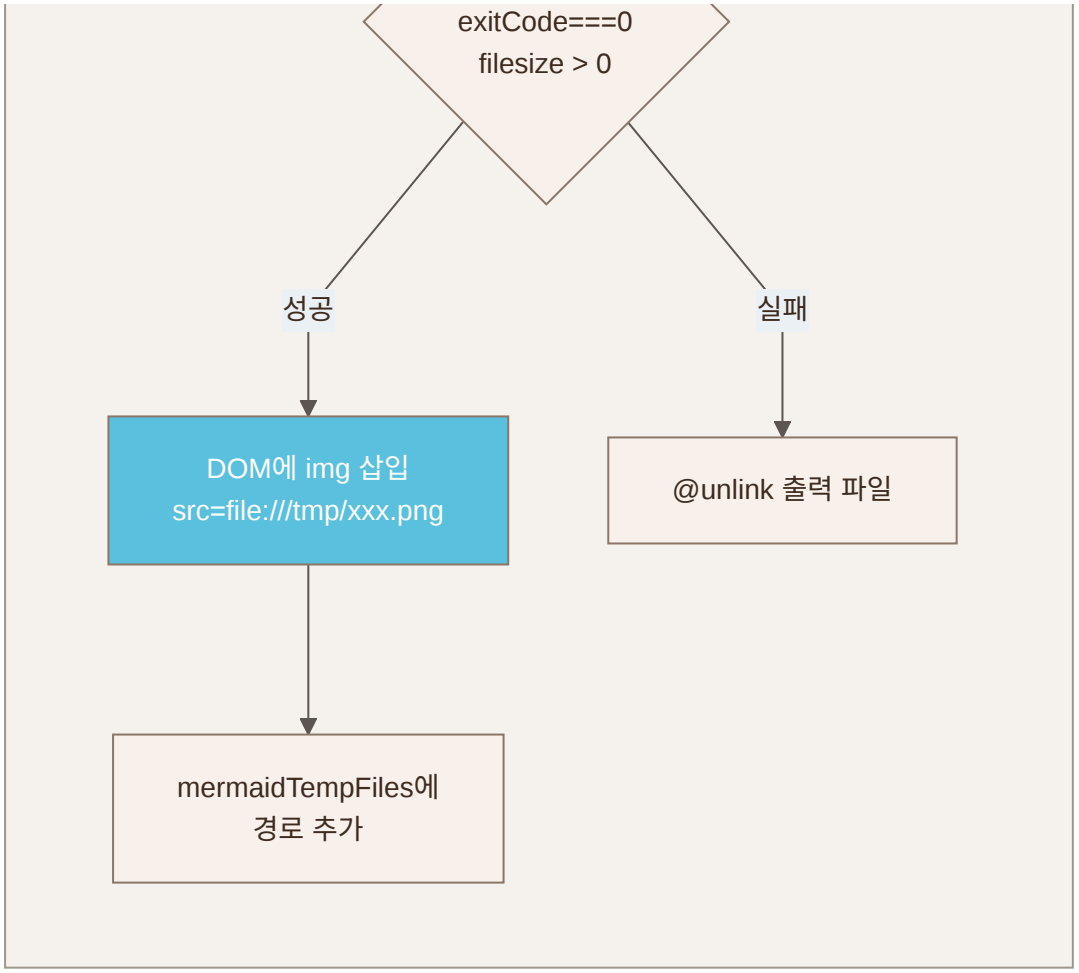
2.1 전체 파이프라인 다이어그램



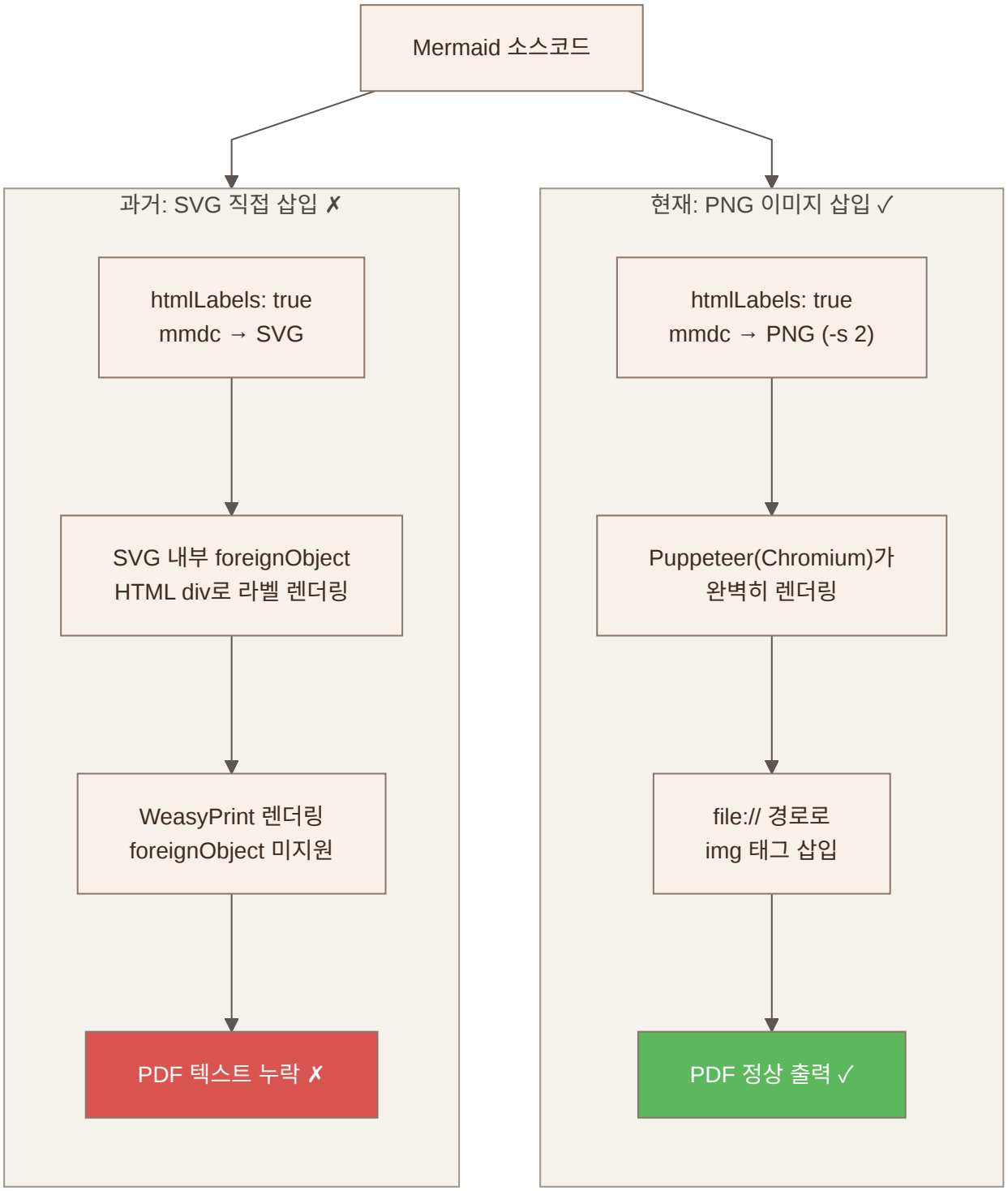


2.2 replaceMermaidWithImage 내부 흐름





2.3 htmlLabels 설정별 렌더링 경로 비교



3. 각 단계 상세

3.1 pageToPdf — 진입점

```
// app/Exports/ExportFormatter.php
public function pageToPdf(Page $page): string
{
    $page->html = (new PageContent($page))->render();
    $html = view('exports.page', [
        'page'   => $page,
        'format' => 'pdf',
        'engine' => $this->pdfGenerator->getActiveEngine(),
        'locale' => user()->getLocale(),
    ])->render();

    return $this->htmlToPdf($html);
}
```

`PageContent->render()`는 마크다운 소스를 HTML로 변환한다. 이 시점에서 Mermaid 코드 블록은 아직 `<pre><code class="language-mermaid">` 형태로 남아 있다. JavaScript 실행 없이 순수 HTML 문자열이다.

3.2 htmlToPdf — 전처리 허브

이 메서드가 전처리의 핵심이다. 모든 변환이 여기서 순서대로 실행된다.

```
protected function htmlToPdf(string $html): string
{
    // 1단계: 이미지 base64 인코딩 + 링크 절대경로 변환
    $html = $this->containHtml($html);

    // DOM 파싱
    $doc = new HtmlDocument();
    $doc->loadCompleteHtml($html);

    // 2단계: iframe을 링크 텍스트로 교체
    $this->replaceIframesWithLinks($doc);

    // 3단계: <details> 요소를 열린 상태로
    $this->openDetailElements($doc);

    // 4단계: Mermaid → PNG (반드시 containHtml 이후에 실행)
    $this->replaceMermaidWithImage($doc);

    $cleanedHtml = $doc->getHtml();
```

```
    // 5단계: WeasyPrint로 PDF 생성 → 6단계: 임시 파일 정리
    try {
        return $this->pdfGenerator->fromHtml($cleanedHtml);
    } finally {
        $this->cleanupMermaidTempFiles();
    }
}
```

실행 순서가 중요한 이유: `containHtml()`은 `<img>` 태그의 `src`를 base64로 변환하는데, `replaceMermaidWithImage()`가 삽입하는 PNG는 `file://` 경로를 사용한다. 만약 순서가 뒤바뀌면 `containHtml()`이 `file://` 경로를 base64 변환하려 시도할 수 있다. 이를 방지하기 위해 `containHtml()`에 `file://` 가드를 추가했다.

3.3 containHtml — 이미지/링크 전처리

```
protected function containHtml(string $htmlContent): string
{
    // 이미지 처리: src를 base64 data URI로 변환
    preg_match_all("/\<img.*?src\=(\'|\")(.*?)(\'|\").*?\>/i",
        $htmlContent, $imageTagsOutput);

    if (isset($imageTagsOutput[0]) && count($imageTagsOutput[0]) > 0) {
        foreach ($imageTagsOutput[0] as $index => $imgMatch) {
            $oldImgTagString = $imgMatch;
            $srcString = $imageTagsOutput[2][$index];

            // ★ file:// 경로 가드 — Mermaid PNG 임시 파일 보호
            if (str_starts_with($srcString, 'file://')) {
                continue;
            }

            $imageEncoded = $this->imageService->imageUrlToBase64($srcString);
            if ($imageEncoded === null) {
                $imageEncoded = $srcString;
            }
            $newImgTagString = str_replace(
                $srcString, $imageEncoded, $oldImgTagString);
            $htmlContent = str_replace(
                $oldImgTagString, $newImgTagString, $htmlContent);
        }
    }

    // 링크 처리: 상대 경로를 절대 URL로 변환
    // 단, #으로 시작하는 앵커 링크는 제외 (각주 등 문서 내 링크 보존)
    preg_match_all("/\<a.*href\=(\'|\")(.*?)(\'|\").*?\>/i",
```

```
        $htmlContent, $linksOutput);

    if (isset($linksOutput[0]) && count($linksOutput[0]) > 0) {
        foreach ($linksOutput[0] as $index => $linkMatch) {
            $oldLinkString = $linkMatch;
            $srcString = $linksOutput[2][$index];
            if (!str_starts_with(trim($srcString), 'http')
                && !str_starts_with(trim($srcString), '#')) {
                $newSrcString = url($srcString);
                $newLinkString = str_replace(
                    $srcString, $newSrcString, $oldLinkString);
                $htmlContent = str_replace(
                    $oldLinkString, $newLinkString, $htmlContent);
            }
        }
    }
    return $htmlContent;
}
```

주의 사항:

- `file://` 가드가 없으면 Mermaid PNG 경로를 `imageUrlToBase64()`에 넘기게 되어, 스토리지 경로 변환 실패로 null이 반환된다.
- 앵커 링크(`#fn1`, `#fnref1`)를 절대 URL로 변환하면 PDF 내 각주 링크가 깨진다. 이 수정은 각주 기능 구현 시 추가되었다.

3.4 replaceMermaidWithImage — 핵심 변환

이 메서드가 Mermaid 코드 블록을 PNG 이미지로 교체하는 핵심 로직이다.

```
protected function replaceMermaidWithImage(HtmlDocument $doc): void
{
    $blocks = $doc->queryXPath(
        '//pre/code[contains(@class, "language-mermaid")]'
    );
    \Log::debug('Mermaid blocks: ' . $blocks->length);

    $nodePath       = '/usr/local/bin/node';
    $mmdcScript     = '/var/www/bookstack/node_modules/'
                    . '@mermaid-js/mermaid-cli/src/cli.js';
    $puppeteerConfig = base_path('puppeteer-config.json');

    if (!file_exists($mmdcScript)) {
        \Log::debug('mmdc script not found');
```

```
        return;
    }

    $blockArray = iterator_to_array($blocks);
    foreach ($blockArray as $block) {
        $pre  = $block->parentNode;
        $code = html_entity_decode($block->textContent, ENT_QUOTES);

        // 임시 파일 생성
        $tmpIn  = tempnam(sys_get_temp_dir(), 'mmd_in_') . '.mmd';
        $tmpOut = tempnam(sys_get_temp_dir(), 'mmd_out_') . '.png';

        file_put_contents($tmpIn, $code);

        // mmdc 실행: Puppeteer(Chromium) → PNG
        $cmd = sprintf(
            'PUPPETEER_CACHE_DIR=/var/www/bookstack/storage/puppeteer '
            . '%s %s -i %s -o %s -c %s --puppeteerConfigFile %s -s 2 2>&1',
            $nodePath,
            $mmdcScript,
            escapeshellarg($tmpIn),
            escapeshellarg($tmpOut),
            escapeshellarg(base_path('mermaid-config.json')),
            escapeshellarg($puppeteerConfig)
        );

        $output = [];
        exec($cmd, $output, $exitCode);
        \Log::debug('exit: ' . $exitCode
            . ' png exists: ' . (file_exists($tmpOut) ? 'yes' : 'no')
            . ' output: ' . implode(' ', $output));

        // 입력 파일은 즉시 삭제
        @unlink($tmpIn);

        if ($exitCode === 0 && file_exists($tmpOut)
            && filesize($tmpOut) > 0) {
            // DOM에 <img> 삽입 — file:// 경로 참조
            $img = $doc->createElement('img');
            $img->setAttribute('src', 'file://' . $tmpOut);
            $img->setAttribute('style',
                'max-width:100%; height:auto;');
            $img->setAttribute('class', 'mermaid-diagram');

            $wrapper = $doc->createElement('div');
            $wrapper->setAttribute('class',
                'mermaid-diagram-wrapper');
```

```
            $wrapper->setAttribute('style',
                'text-align:center; margin:1em 0;');
            $wrapper->appendChild($img);

            $pre->parentNode->replaceChild($wrapper, $pre);

            // WeasyPrint가 읽을 때까지 PNG 유지
            $this->mermaidTempFiles[] = $tmpOut;
            \Log::debug('PNG file referenced: ' . $tmpOut
                . ' (' . filesize($tmpOut) . ' bytes)');
        } else {
            @unlink($tmpOut);
        }
    }
}
```

mmdc 명령어 옵션 설명

옵션	값	역할
`-i`	입력 .mmd 파일	Mermaid 소스
`-o`	출력 .png 파일	PNG 형식 지정 (확장자로 결정)
`-c`	`mermaid-config.json`	Mermaid 설정 (theme 등)
`--puppeteerConfigFile`	`puppeteer-config.json`	Chromium 실행 옵션
`-s 2`	스케일 팩터	2배 해상도 출력

설정 파일 내용

```
// mermaid-config.json — htmlLabels: true(기본값) 사용
{
  "theme": "default"
}

// puppeteer-config.json
{
  "args": ["--no-sandbox", "--disable-setuid-sandbox"],
  "cacheDirectory": "/var/www/bookstack/storage/puppeteer"
}
```

3.5 cleanupMermaidTempFiles — 임시 파일 정리

```
protected function cleanupMermaidTempFiles(): void
{
    foreach ($this->mermaidTempFiles as $tmpFile) {
        if (file_exists($tmpFile)) {
```

```
            @unlink($tmpFile);
            \Log::debug('Cleaned up Mermaid temp file: ' . $tmpFile);
        }
    }
    $this->mermaidTempFiles = [];
}
```

`try/finally`로 호출되므로 PDF 생성 성공/실패와 무관하게 실행된다. 이전에 대용량 이미지로 인한 메모리 초과 경험이 있어, 임시 파일이 `/tmp`에 누적되는 것을 방지하기 위해 이 패턴을 사용한다.

4. 왜 PNG인가 — 의사결정 기록

4.1 시도한 접근들

시도	방식	결과	실패 원인
1차	SVG + `htmlLabels: true`	텍스트 누락	WeasyPrint가 SVG 내 `<foreignObject>` 미지원
2차	SVG + `htmlLabels: false`	특수문자 깨짐	Mermaid 자체 버그 — ` `, `\`, `&`, `<>` 처리 불량
최종	**PNG +** `htmlLabels: true`	**정상**	Puppeteer가 완벽히 렌더링한 결과를 래스터 이미지로 캡처

4.2 htmlLabels 설정의 영향

`htmlLabels`는 Mermaid가 노드 라벨을 렌더링하는 방식을 결정한다.

`htmlLabels: true` **(기본값)** — HTML/CSS로 라벨 렌더링:

```
<svg>
  <foreignObject>
    <div xmlns="http://www.w3.org/1999/xhtml">
      <span>노드 텍스트</span>
    </div>
  </foreignObject>
</svg>
```

`htmlLabels: false` — 순수 SVG 텍스트 요소:

```
<svg>
  <text>
    <tspan>노드 텍스트</tspan>
  </text>
</svg>
```

PNG 방식에서는 Puppeteer(Chromium)가 렌더링을 완료한 뒤 스크린샷을 찍으므로, `<foreignObject>` 문제가 발생하지 않는다. 따라서 `htmlLabels: true`(기본값)를 사용하여 줄바꿈, 특수문자, 긴 텍스트 등을 정상 처리한다.

4.3 임시 파일 참조 vs base64 인라인

	base64 인라인	임시 파일 참조 (채택)
PHP 메모리	높음 (이미지 데이터 전체 로드)	낮음 (경로 문자열만)

	base64 인라인	임시 파일 참조 (채택)
PDF 생성 속도	느림 (HTML 문자열 비대)	빠름
과거 문제	base64로 HTML 크기 폭증 → 변환 중 뻗음	없음
임시 파일 관리	불필요	필요 (finally로 해결)

5. 실행 환경 의존성

컴포넌트	경로 / 설정	용도
Node.js	`/usr/local/bin/node`	mmdc 실행 (nvm이 아닌 복사본 — www-data 접근 가능)
mmdc	`node_modules/@mermaid-js/mermaid-cli/src/cli.js`	Mermaid → PNG 변환
Puppeteer cache	`/var/www/bookstack/storage/puppeteer`	Chromium 바이너리 저장
chrome-headless-shell	storage/puppeteer 하위	Puppeteer가 사용하는 브라우저
WeasyPrint	시스템 명령어	HTML → PDF 최종 변환
mermaid-config.json	`/var/www/bookstack/mermaid-config.json`	Mermaid 렌더링 설정
puppeteer-config.json	`/var/www/bookstack/puppeteer-config.json`	Chromium 실행 옵션

중요: nvm으로 설치한 Node.js는 `www-data`(PHP-FPM 사용자)가 접근할 수 없다. `/usr/local/bin/node`에 바이너리를 복사해야 한다.

6. 로그 확인

PDF export 시 아래 로그가 `storage/logs/laravel.log`에 기록된다.

정상 동작 시:

```
[2026-03-23] production.DEBUG: Mermaid blocks: 2
[2026-03-23] production.DEBUG: exit: 0 png exists: yes output: Generating single mermaid chart
[2026-03-23] production.DEBUG: PNG file referenced: /tmp/mmd_out_xxx.png (45231 bytes)
[2026-03-23] production.DEBUG: Cleaned up Mermaid temp file: /tmp/mmd_out_xxx.png
```

mmdc 실패 시:

```
[2026-03-23] production.DEBUG: exit: 1 png exists: no output: Error: ...
```

7. 브라우저 렌더링과의 관계

PDF export 파이프라인은 **브라우저 렌더링과 완전히 독립적**이다.

항목	브라우저 (페이지 보기)	PDF 내보내기
Mermaid 로드	`mermaid-init.js` (Custom HTML Head)	mmdc CLI
렌더링 엔진	브라우저 JS 엔진	Puppeteer (Chromium headless)
출력 형식	SVG (인라인)	PNG (file:// 참조)
뷰어	`mermaid-viewer.js` (클릭 확대)	없음 (이미지)
최종 변환	없음 (브라우저가 직접 표시)	WeasyPrint → PDF
설정 파일	`mermaid-init.js` 내 initialize()	`mermaid-config.json`

브라우저 쪽 파일(`mermaid-init.js`, `mermaid-viewer.js`, `mermaid-viewer.css`)을 수정해도 PDF export에는 영향이 없고, 그 반대도 마찬가지다.

8. 수정 이력

날짜	파일	변경 내용
2026-02-23	`ExportFormatter.php`	`replaceMermaidWithSvg` 메서드 신규 추가 (SVG 방식)
2026-03-11	`ExportFormatter.php`	SVG → PNG 전환, `htmlToPdf`에 try/finally 래핑
2026-03-11	`mermaid-config.json`	`htmlLabels: false` 제거, 기본값 사용
2026-03-23	`ExportFormatter.php`	메서드명 `replaceMermaidWithSvg` → `replaceMermaidWithImage`
2026-03-23	`ExportFormatter.php`	`containHtml()`에 `file://` 경로 가드 추가

9. 관련 문서

- Mermaid Part 1 ~ Part 4: 브라우저 렌더링, 에디터 프리뷰, Interactive Viewer
- Mermaid Part 5 — PDF Export PNG 전환: 이 문서의 의사결정 배경 상세
- 각주 PDF Export: `containHtml()` 앵커 링크 수정
- 이미지 메모리 제한: `ImageService.php` 50MB 제한 패치