

PDF Reporter MCP

Technical Overview & Demo Report

February 17, 2026

Table of Contents

Architecture

How It Works

Key Features

Callout Blocks Showcase

Code Distribution

Code Highlighting

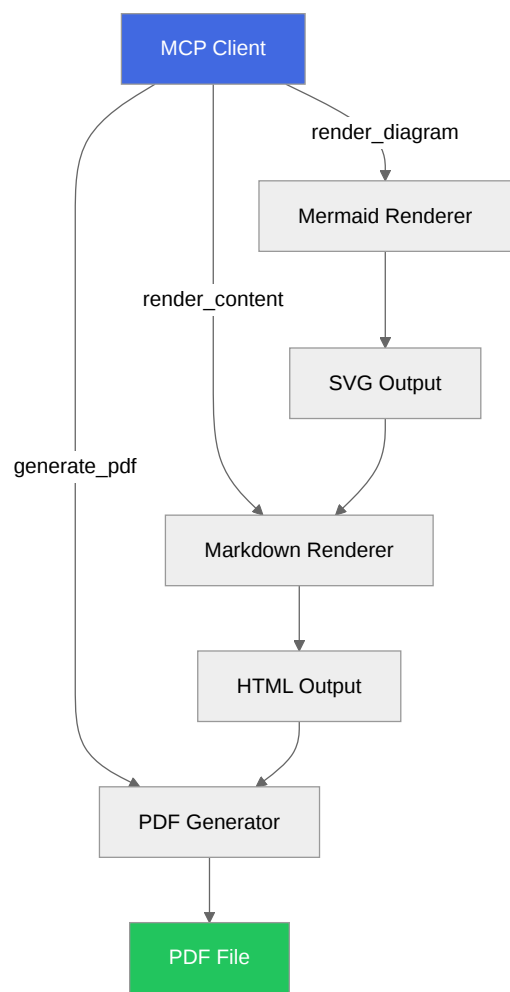
Technical Stack

Summary

PDF Reporter MCP — Overview

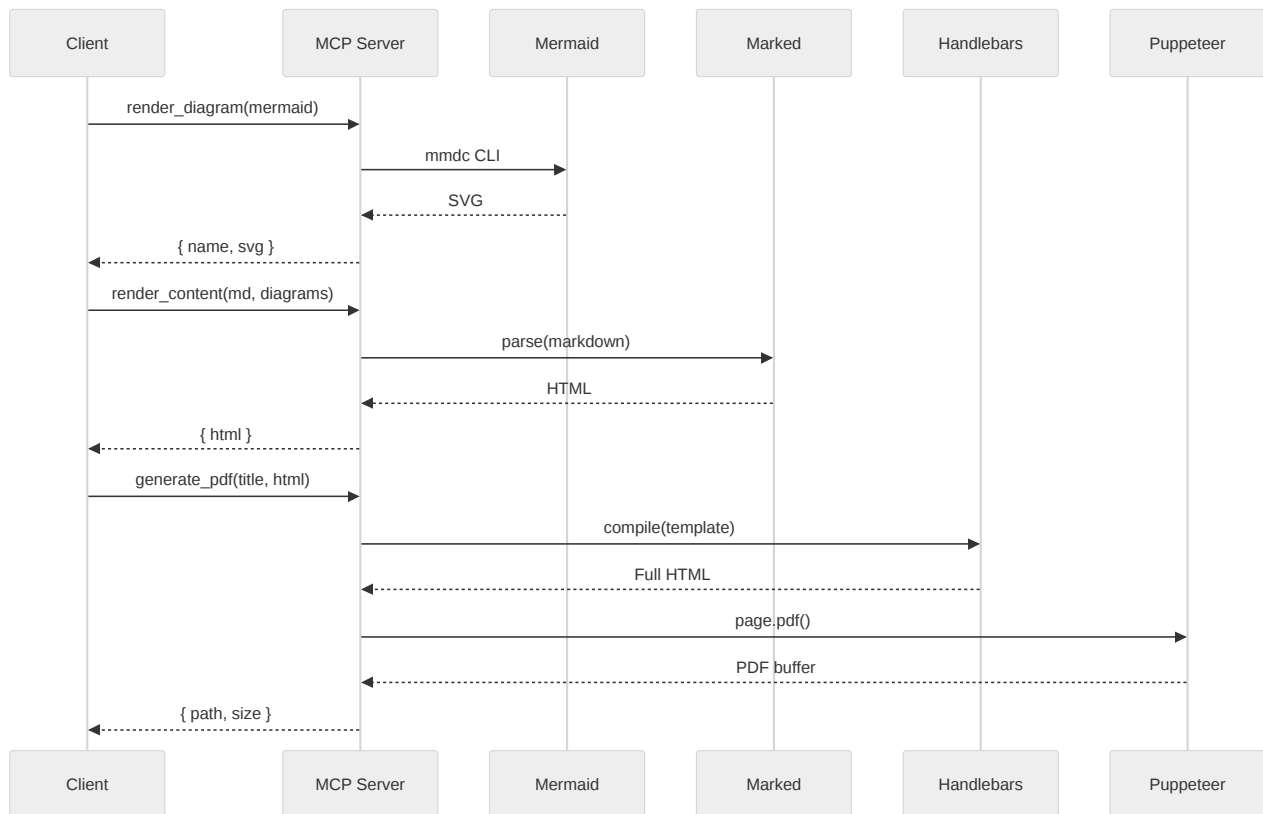
Architecture

The system is built as a modular MCP server with three main tools that can be composed together:



How It Works

The pipeline follows a clear sequence of operations:



Key Features

Callout Blocks Showcase

The system supports 9 types of callout blocks for rich document formatting:



Creative Insight

Every great report starts with a clear structure. Use **headings** for navigation, **callouts** for emphasis, and **diagrams** for visualization. The combination creates documents that are both informative and visually engaging.



Automation Opportunity

This entire report was generated programmatically through MCP tools. The workflow can be integrated into any CI/CD pipeline:

1. Collect data from APIs
2. Render diagrams from metrics
3. Generate PDF automatically
4. Distribute via email or Slack



Browser Dependency

Puppeteer requires Chrome/Chromium to be installed. In Docker, the Dockerfile handles this automatically. Locally, Puppeteer downloads Chrome on `npm install`.



All Tests Passing

The project achieves **77 tests** across 7 test suites with 100% pass rate. All tests run fully offline using mocked dependencies — no external services required.



Architecture Decision

The MCP server exposes three granular tools instead of one monolithic function. This allows AI assistants to build reports **incrementally** — rendering diagrams first, then composing content, and finally generating the PDF.



Security Consideration

User-provided content is rendered through Handlebars with HTML escaping enabled by default. However, the `content` field accepts raw HTML — always sanitize untrusted input before passing it to `generate_pdf`.



Return on Investment

A single MCP server replaces custom PDF generation code in every project. **One deployment** serves unlimited clients — reducing development time from days to minutes per new report type.



Technical Deep Dive

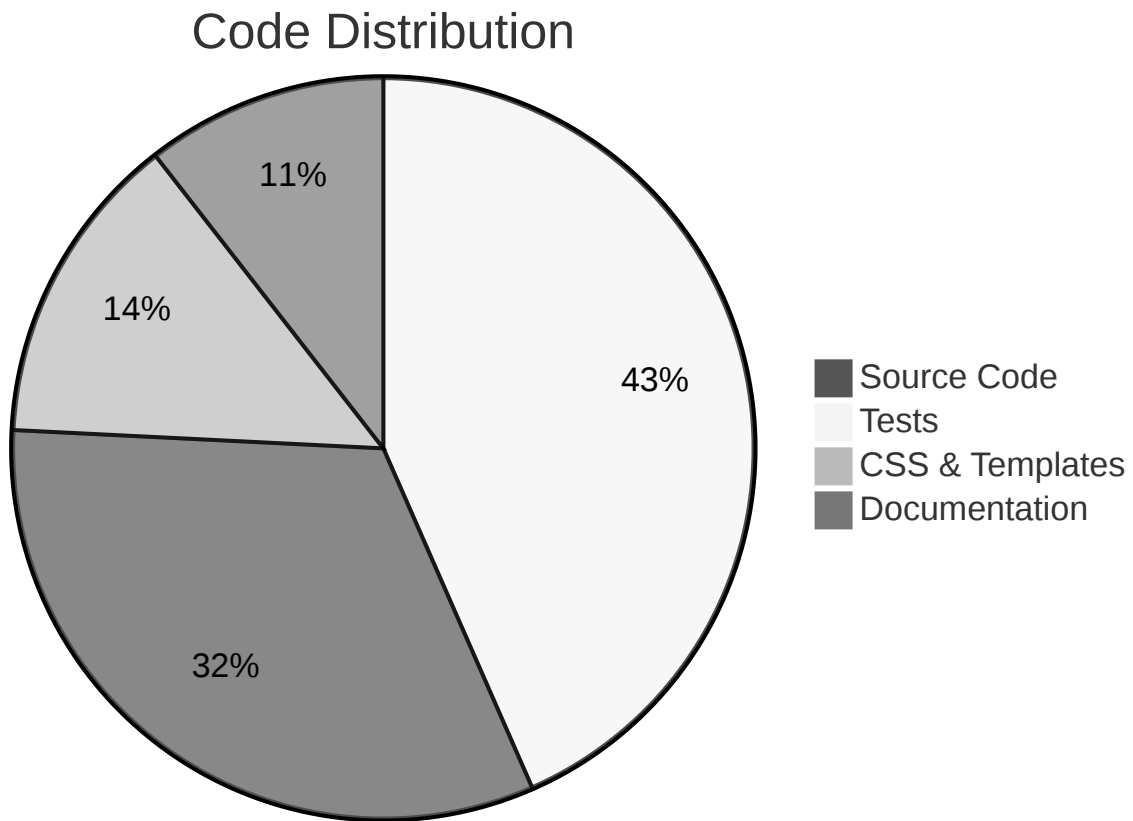
The callout parser uses a two-phase approach: first, fenced code blocks are identified and protected from processing. Then, `:::type` blocks are extracted via regex, their body content is recursively rendered through `marked`, and the result is wrapped in styled HTML divs with inline CSS for print reliability.



Best Practice

Use `render_diagram` separately to pre-render each diagram, then pass the SVG results to `render_content`. This gives you full control over the rendering pipeline and makes debugging easier.

Code Distribution



Code Highlighting

The system supports syntax highlighting in code blocks:

```
// Example: Using the MCP tools
const diagram = await renderDiagram({
  name: 'flow',
  mermaid: 'graph TD; A-->B',
});

const content = await renderContent({
  markdown: '# Report\n{{diagram:flow}}',
  diagrams: [diagram],
});

const pdf = await generatePdfFromHtml({
  title: 'My Report',
  content: content.html,
});
```

Technical Stack

Component	Technology	Purpose
Runtime	Node.js 20	Server platform
Language	TypeScript (strict)	Type safety
PDF Engine	Puppeteer	Chrome-based PDF generation
Diagrams	Mermaid CLI	Diagram rendering
Templates	Handlebars	HTML templating
Markdown	marked + highlight.js	Content rendering
Protocol	MCP SDK	Tool interface
Tests	Vitest	77 tests, all offline

Summary

PDF Reporter MCP transforms markdown content into professional PDF documents with:

- **Royal Blue themed** cover pages and table headers
- **9 callout types** for structured information blocks
- **Mermaid diagrams** rendered to crisp SVG
- **Syntax-highlighted** code blocks
- **Configurable themes** via environment variables
- **Docker-ready** deployment with GitHub Actions CI