MCP SERVER

pdfcrowd-mcp-pdf-export

Project Overview & Architecture

836	2	39	6
LINES OF CODE	MCP TOOLS	UNIT TESTS	DEPENDENCIES

v1.0.1 • MIT License • Node.js ≥18 • TypeScript • Vitest

Contents

1 What This Project Does

2 Project Structure

3 Architecture

4 Key Components

5 Data Flow

6 Type System & Schemas

7 Error Handling Strategy

8 Dependencies & How They Connect

9 Build & Test Infrastructure

10 Design Patterns

1. What This Project Does

This is a **Model Context Protocol (MCP) server** that gives AI agents (Claude Code, Codex CLI, Gemini CLI) the ability to generate PDFs. It accepts HTML content, local files, or URLs, bundles any local assets, sends them to the PDFCrowd cloud API, and writes the resulting PDF to disk.

Core loop: AI Agent → MCP tool call → Validate input → Bundle local assets → POST to PDFCrowd API → Write PDF → Return metadata to agent

Two MCP Tools Exposed

Tool	Purpose	Key Inputs
`pdfcrowd_create_pdf`	Convert content to PDF	`html` | `url` | `file`, `output_path`, page options
`pdfcrowd_info`	Return usage guidance	`topic`: html_layout, mermaid_diagrams, local_assets, parameters

2. Project Structure

```
pdfcrowd-mcp-pdf-export/
  src/
    index.ts (227 loc) — MCP server entry, tool registration, topic content
    version.ts (6 loc) — Reads VERSION from package.json
    schemas/
      index.ts (56 loc) — Zod schemas, constants, types
    services/
      pdfcrowd-client.ts (336 loc) — API client, retries, error mapping
      asset-bundler.ts (211 loc) — Asset detection, ZIP bundling
  tests/
    unit/ — 39 test cases (577 loc)
    prompt/ — 5 integration tests with fixtures
  dist/ — Compiled JS, declarations, source maps
  package.json — Metadata, scripts, deps
  tsconfig.json — ES2022, Node16 modules, strict
  vitest.config.ts — 30s test timeout
  makefile — Build and publish targets
```

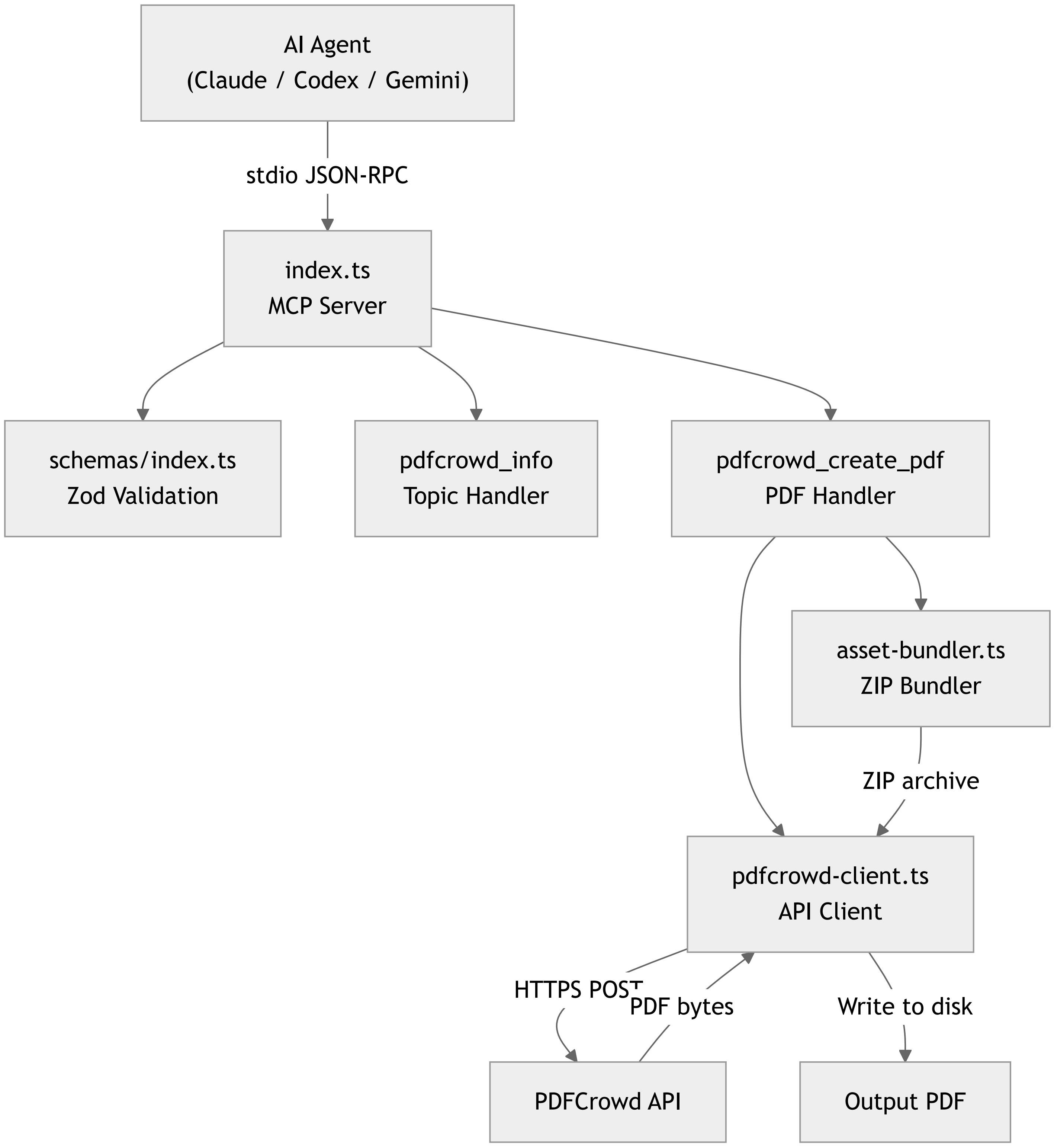
Lines of Code Breakdown

Layer	Files	LOC	Share
SRC Server & Tools	index.ts, version.ts	233	28%
SRC Schemas	schemas/index.ts	56	7%
SRC Services	pdfcrowd-client.ts, asset-bundler.ts	547	65%
TEST Unit Tests	3 files	577	—
Total Production		**836**	100%

3. Architecture

The server is organized in three layers: protocol (MCP), validation (Zod), and services (API client + asset bundler).

Component Diagram



Layer Responsibilities

Layer	File	Responsibility
Protocol	index.ts	MCP server lifecycle, tool registration, topic content, stdio transport
Validation	schemas/index.ts	Input validation (Zod strict + refine), type inference, JSON schema generation
Service: API	pdfcrowd-client.ts	Credential management, FormData construction, HTTPS requests, retry logic, error mapping
Service: Bundler	asset-bundler.ts	HTML/CSS parsing, local file detection, ZIP archive creation, reference rewriting

4. Key Components

4.1 MCP Server (index.ts)

Entry point. Creates the MCP server, validates env vars, registers both tools, and connects to stdio transport.

Static Topics

Topic Key	Content
`html_layout`	CSS reset, viewport width (1096px default), font sizing, page breaks, cover pages
`mermaid_diagrams`	CDN URL, init config, sizing limits (6-8 nodes), CSS/HTML template
`local_assets`	Auto-bundling docs, supported attributes, relative path resolution

Dynamic Topic

Topic Key	Generated At	Content
`parameters`	Runtime	Full JSON Schema from Zod via `zod-to-json-schema`, plus usage examples and temp file conventions

4.2 PDFCrowd Client (pdfcrowd-client.ts)

The largest component (336 loc). Handles all communication with the PDFCrowd API.

API Configuration

Constant	Value
API Base URL	`https://api.pdfcrowd.com/convert/24.04`
Auth	HTTP Basic (username:apiKey)
Max Retries	2 (3 total attempts)
Retry Delay	1,000 ms
Request Timeout	120,000 ms
User-Agent	`pdfcrowd-mcp-pdf-export/{VER} (Node.js)`

Key Functions

Function	Purpose
`getCredentials()`	Read env vars, throw if missing
`isDemo()`	Return true when username is "demo"
`buildForm()`	Construct multipart FormData with all fields
`parseMetadata()`	Extract jobId, credits, pageCount from response headers
`createPdf()`	Main entry: validate → bundle → API call → write file
`getErrorGuidance()`	Map API error codes to actionable guidance

4.3 Asset Bundler (asset-bundler.ts)

Detects local files referenced in HTML/CSS and bundles them into a ZIP archive for the API.

Detection Patterns

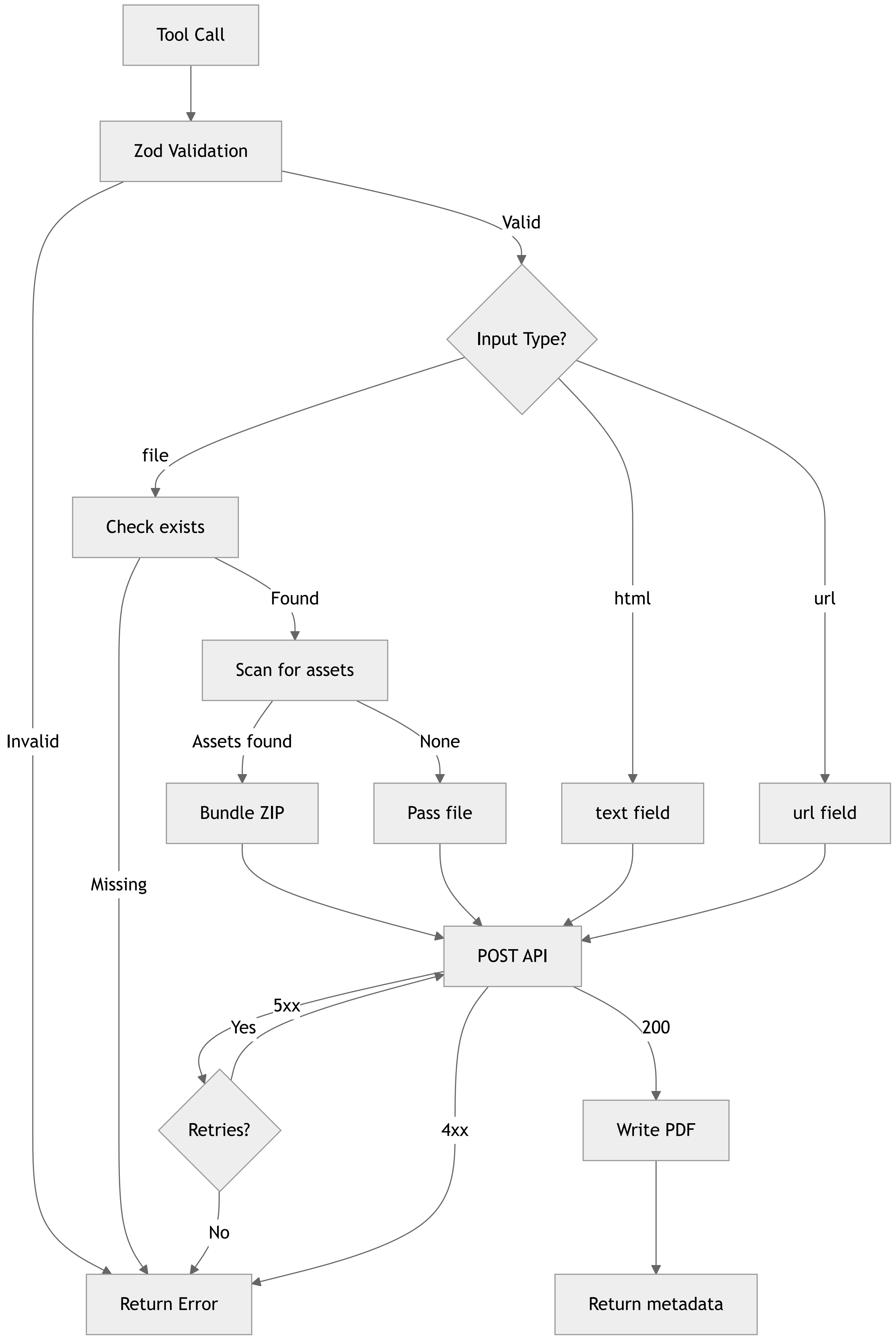
Source	Tags / Patterns
HTML tags	`img` , `script` , `video` , `audio` , `source` , `embed` , `input` (src), `link` (href), `object` (data)
CSS	`url()` patterns, recursively parsed for linked CSS files
Skipped	`http://` , `https://` , `data:` , `javascript:` , `mailto:` , `//`

Key Functions

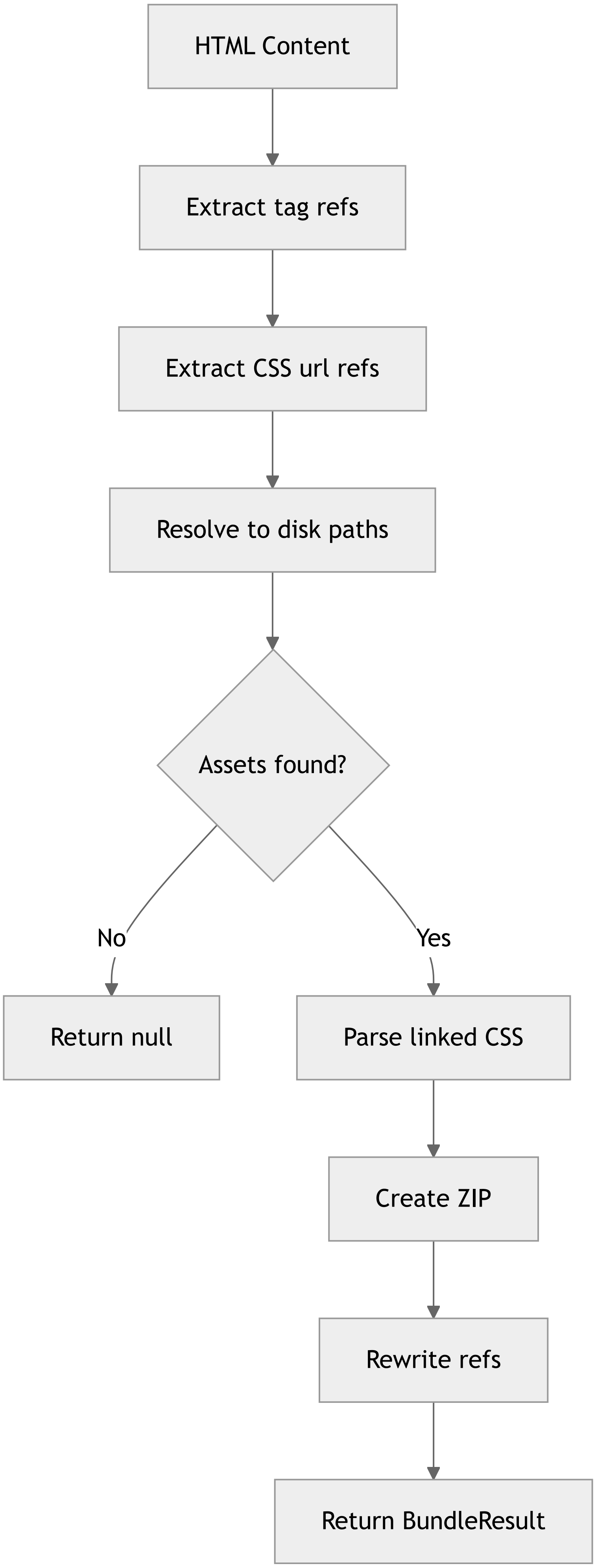
Function	Purpose
`extractHtmlRefs()`	Regex-based extraction from HTML tags and inline CSS
`extractUrlRefs()`	Extract `url()` patterns from CSS content
`resolveAssets()`	Resolve paths to disk, deduplicate, handle external paths
`rewriteRefs()`	Replace original refs with ZIP-relative paths (longest-first)
`bundleAssets()`	Main: extract → resolve → ZIP → return { zipPath, cleanup }

5. Data Flow

Conversion Pipeline


Tool Call
Zod Validation
Valid
Invalid
Input Type?
file
html
url
Check exists
Found
Missing
Scan for assets
Assets found
None
Bundle ZIP
Pass file
text field
url field
POST API
5xx
Yes
Retries?
No
4xx
200
Write PDF
Return Error
Return metadata

Asset Bundling Pipeline


HTML Content
Extract tag refs
Extract CSS url refs
Resolve to disk paths
Assets found?
No
Yes
Return null
Parse linked CSS
Create ZIP
Rewrite refs
Return BundleResult

FormData Fields Sent to API

Field	Source	Notes
`text` / `url` / `file`	Input parameter	Mutually exclusive; file sent as stream
`page_size`	page_size param	A3, A4, A5, Letter
`orientation`	orientation param	portrait, landscape
`margin_top/bottom/left/right`	margins param	Same value for all four; comma-to-period normalization
`content_viewport_width`	viewport_width param	Default: 1096px
`content_fit_mode`	Hardcoded	Always "content-width"
`zip_main_filename`	Bundler result	Only when ZIP bundling is used
`title`	title param	PDF metadata (optional)

6. Type System & Schemas

Zod Schema (CreatePdfSchema)

Defined in `schemas/index.ts` using strict mode with a custom refinement for mutual exclusivity.

```
z.object({
  html:           z.string().optional(),
  url:            z.string().url().optional(),
  file:           z.string().optional(),
  output_path:    z.string().min(1),
  page_size:      z.enum(["A3","A4","A5","Letter"]).default("A4"),
  orientation:    z.enum(["portrait","landscape"]).default("portrait"),
  margins:        z.union([z.string().regex(MARGIN_REGEX), z.literal(0)])
                   .default("10mm"),
  viewport_width: z.number().int().min(96).max(65000).optional(),
  title:          z.string().optional(),
}).strict()
  .refine(d => exactly one of html/url/file is set)
```

Schema Constants

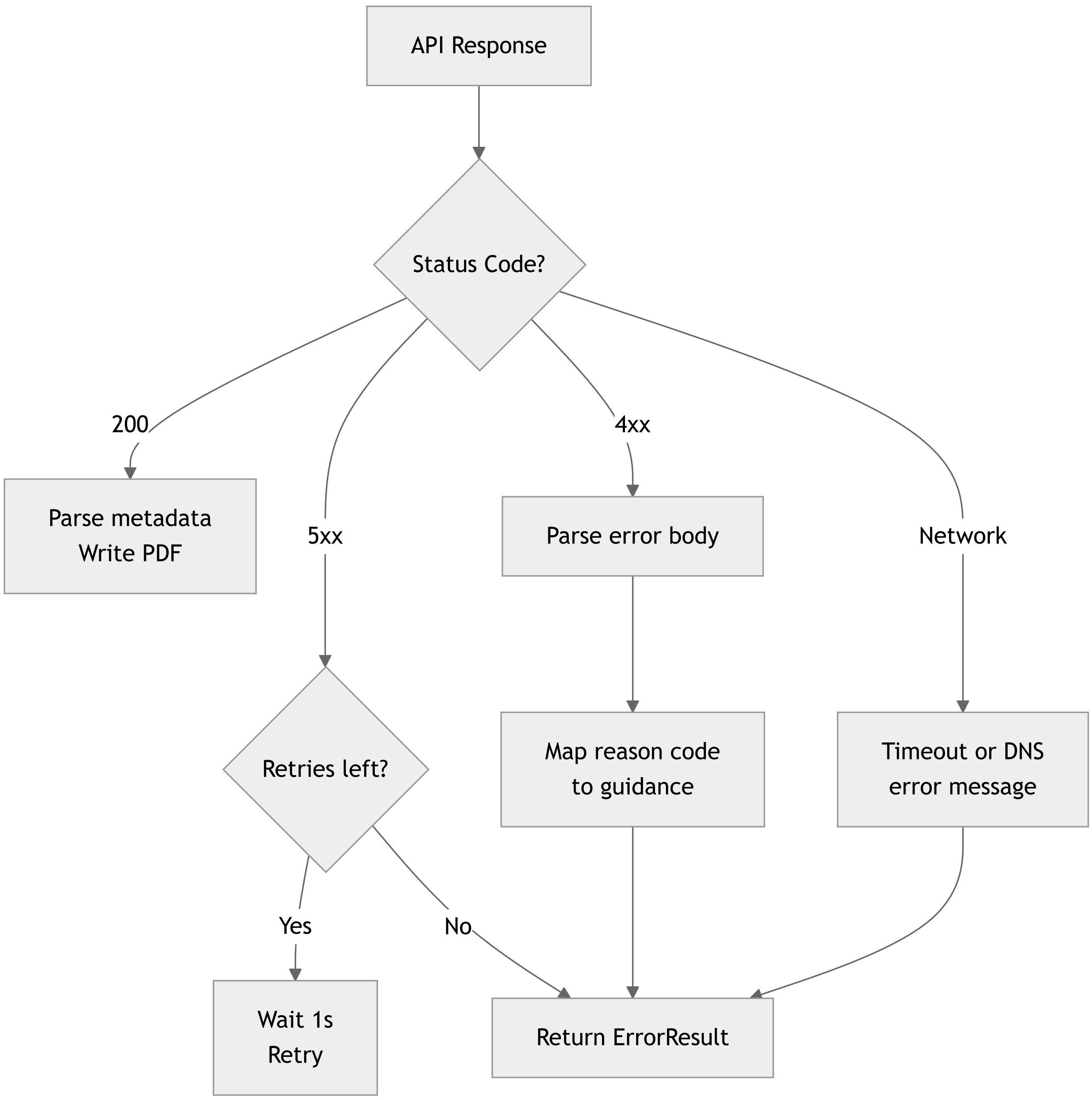
Constant	Value
PAGE_SIZES	`["A3", "A4", "A5", "Letter"]`
ORIENTATIONS	`["portrait", "landscape"]`
DEFAULT_MARGIN	`10` (mm)
DEFAULT_VIEWPORT_WIDTH	`"1096px"`
MIN/MAX_VIEWPORT_WIDTH	`96` / `65000`
MARGIN_REGEX	`/^\d+([.,]\d+)?(in|mm|cm|px|pt)$/`

Key Interfaces (pdfcrowd-client.ts)

Interface	Fields
`ConversionMetadata`	jobId, remainingCredits, consumedCredits, pageCount?, outputSize
`ConversionResult`	success: true, outputPath, metadata, isDemo
`ErrorResult`	success: false, error, httpCode?
`BundleResult`	zipPath, mainFilename, cleanup()

7. Error Handling Strategy

Error Flow



Error Code Mapping

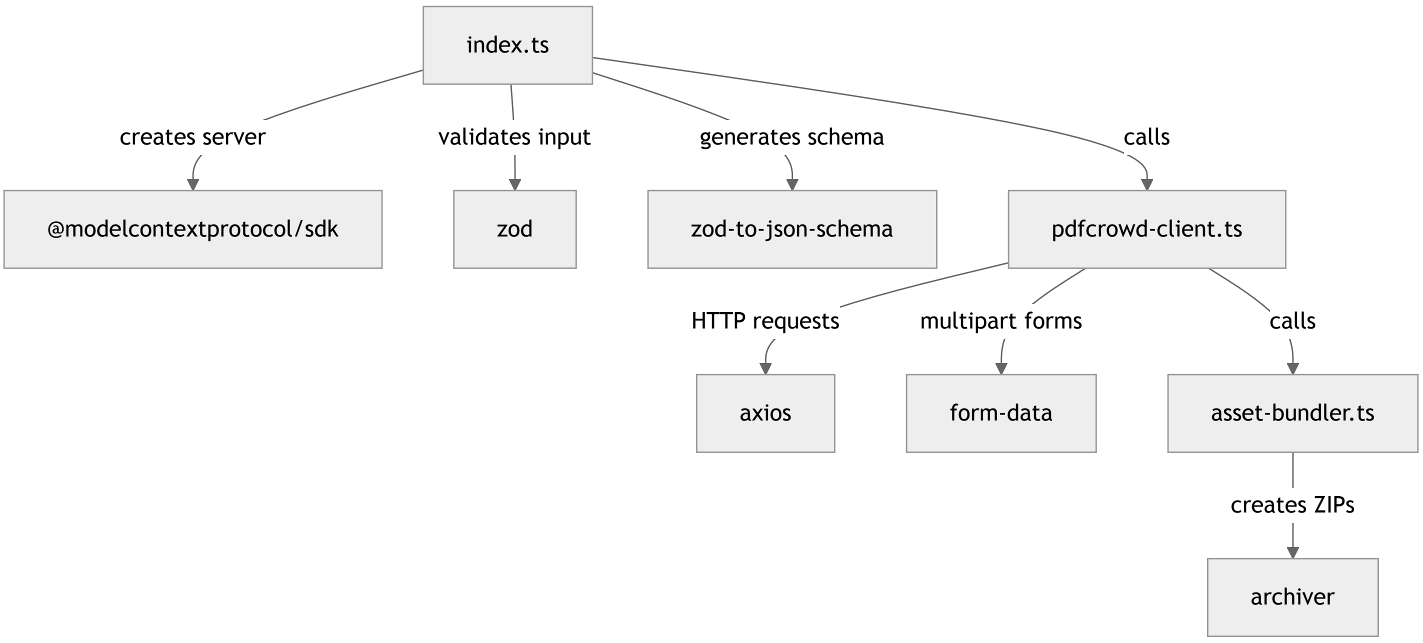
Code	Meaning	Guidance
103	License expired	Renew subscription
106	Invalid credentials	Check env vars

Code	Meaning	Guidance
105	No credits	Purchase credits
120	Rate limited	Reduce frequency
121	Concurrent limit	Wait for current jobs
122	Demo exhausted	Upgrade account
305/325	Invalid HTML	Check content
320	Invalid URL	Verify URL format
323	Timeout	Simplify layout
337	Invalid param	Check schema
357	Password protected	Remove password

Design principle: Every API error is mapped to an *actionable* guidance message so the AI agent can self-correct or inform the user. 4xx errors are never retried (user must fix config); only 5xx (transient server errors) trigger automatic retries.

8. Dependencies & How They Connect

Dependency Map



Production Dependencies

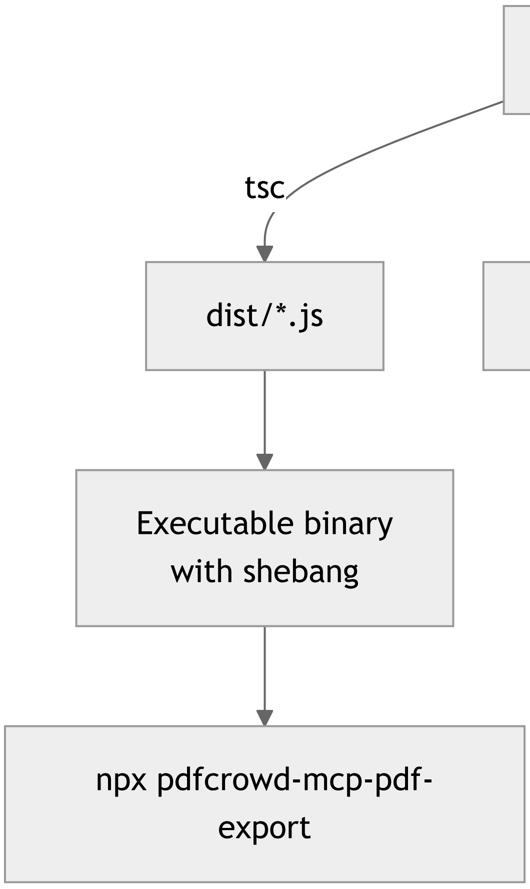
Package	Version	Used By	Purpose
`@modelcontextprotocol/sdk`	^1.6.1	index.ts	MCP server, tool registration, stdio transport
`zod`	^3.23.8	schemas/	Runtime validation, type inference, strict mode
`zod-to-json-schema`	^3.25.1	index.ts	Generate JSON Schema for MCP tool descriptions
`axios`	^1.7.9	pdfcrowd-client	HTTPS POST, Basic Auth, timeout handling
`form-data`	^4.0.0	pdfcrowd-client	Multipart form encoding for API requests
`archiver`	^7.0.1	asset-bundler	ZIP archive creation (zlib level 5)

Development Dependencies

Package	Version	Purpose
`typescript`	^5.7.2	Compiler (ES2022 target, Node16 modules, strict)
`vitest`	^4.0.18	Test runner (30s timeout, parallel execution)
`tsx`	^4.19.2	TypeScript executor for `npm run dev` with watch mode
`@types/node`	^22.10.0	Node.js type definitions (fs, path, os, etc.)
`@types/archiver`	^7.0.0	Archiver type definitions

9. Build & Test Infrastructure

Build Pipeline



npm Scripts

Script	Command	Purpose
`npm start`	`node dist/index.js`	Run compiled server
`npm run dev`	`tsx watch src/index.ts`	Dev mode with live reload
`npm run build`	`tsc`	Compile TypeScript
`npm test`	`vitest run`	Run unit tests
`npm run clean`	`rm -rf dist`	Remove build output

Test Coverage

Test File	Cases	Covers
schemas.test.ts	23	Validation, defaults, bounds, mutual exclusion, strict mode
pdfcrowd-client.test.ts	10	Retries, auth, errors, metadata, form fields, network failures

Test File	Cases	Covers
asset-bundler.test.ts	6	Bundling, CSS parsing, missing files, external paths, cleanup

Integration Tests (Prompt Tests)

Prompt	Validates
hello-world	Single full-bleed page, text content
html-layout	Multi-page (3 pages), titled sections
invoice-template	HTML template merged with JSON data
local-assets	Image bundling (house.png), file size >100KB
mermaid-flowchart	Diagram rendering, no mermaid version text

10. Design Patterns

Discriminated Union Results

The API client returns `ConversionResult | ErrorResult`, using the `success` boolean as a discriminant. This eliminates null checks and makes error handling explicit at every call site.

Schema-Driven Interfaces

A single Zod schema (`CreatePdfSchema`) serves three purposes simultaneously:

Purpose	Mechanism
Runtime validation	`schema.parse(input)`
TypeScript types	`z.infer<typeof schema>`
MCP tool description	`zodToJsonSchema(schema)`

Guaranteed Cleanup

The asset bundler returns a `cleanup()` function. The caller uses it in a `finally` block, ensuring temp directories are removed even when the API call fails or throws.

Error Guidance as Data

Rather than generic error messages, every PDFCrowd error code maps to a specific guidance string. This lets AI agents self-correct without human intervention—a critical design choice for an MCP tool that runs inside autonomous workflows.

Longest-First Replacement

When rewriting asset references in HTML/CSS, the bundler sorts replacements by length (longest first) to prevent partial substring matches—e.g., replacing `img/bg.png` before `img/bg.png?v=2`.

Stream-Based File Handling

File uploads use `fs.createReadStream()` instead of reading entire files into memory. This keeps memory usage bounded even for large HTML documents with many assets.