Міністерство освіти і науки України
Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки Кафедра обчислювальної техніки

Курсова робота
з дисципліни
“Компоненти програмної інженерії”

На тему: парсер Markdown у PDF
Виконав: студент групи ІМ-31 Соломка Андрій Романович

2025 рік

Design Document

Markdown to PDF Parser

---

1. Introduction and Project Goals

1.1 Project Overview

This project is a straightforward Markdown to PDF converter built with TypeScript and Node.js. The goal is to create a practical tool that takes markdown files and generates readable PDF documents with proper formatting for headings, lists, tables, code blocks, and images.

The parser uses a simple three-stage approach: parse markdown into tokens, convert tokens into a document structure, and render that structure to PDF. The implementation focuses on correctness and simplicity rather than architectural complexity.

1.2 Objectives & Deliverables

Type	Objective	Deliverable
Functional	• Parse CommonMark markdown syntax • Support tables, code blocks, and images • Generate readable PDF documents	• CLI tool that converts .md files to .pdf • Support for basic markdown features • Configurable page settings
Implementation	• Straightforward code structure • Minimal dependencies • Clear data flow	• ~2,000-2,500 lines of TypeScript • Standard npm packages • Simple module organization
Quality	• Handle common markdown correctly • Reasonable error messages	• Basic test coverage for main features • Handle malformed input gracefully

2. System Architecture and Design

2.1 Core Components

The system consists of four main modules:

Component	Responsibility	Main Functions	Dependencies
Parser	Convert markdown text to token list	`parse(markdown: string)` → Token[]	None (regex-based)
Converter	Build document structure from tokens	`convert(tokens: Token[])` → Document	Parser output
Renderer	Generate PDF from document structure	`render(doc: Document)` → PDF	PDFKit library

CLI	Handle command-line interface	Read files, write output, show progress	fs, path modules

2.2 Data Flow

```
Markdown File → Parser → Token List → Converter → Document Object → Renderer → PDF File
                  ↓          ↓             ↓             ↓             ↓
              Regex      Array of      Tree of       PDFKit API    Binary PDF
              matching   token objects  elements      calls         output
```

3. Markdown Support

3.1 Supported Markdown Features

Feature	Syntax	Implementation Notes
Headings	`#` to `######`	Detect # at line start, count level
Bold	`**text**` or `__text__`	Replace with bold markers
Italic	`*text*` or `_text_`	Replace with italic markers
Code Inline	`` `code` ``	Detect backticks, extract content
Code Blocks	```` ```language ... ``` ````	Detect triple backticks, extract language and code
Lists Unordered	`- item` or `* item`	Detect - or * at line start
Lists Ordered	`1. item`	Detect number + period at line start
Links	`[text](url)`	Regex match link pattern
Images	`![alt](url)`	Regex match, download and embed
Tables	Pipe-separated cells	Parse rows, detect header separator
Blockquotes	`> text`	Detect > at line start
Horizontal Rules	`---` or `***`	Detect three or more dashes/asterisks
Line Breaks	Double newline	Paragraph separation

4. Implementation Details

4.1 Parser Implementation

Token Types:

Token Type	Properties	Example
`HEADING`	level: 1-6, text: string	`# Title` → {type: 'HEADING', level: 1, text: 'Title'}

`PARAGRAPH`	text: string, formatting: InlineFormat[]	Plain text with bold/italic markers
`LIST_ITEM`	ordered: boolean, text: string, level: number	`- Item` → {type: 'LIST_ITEM', ordered: false, text: 'Item'}
`CODE_BLOCK`	language: string | null, code: string	```` ```js\ncode\n``` ````
`TABLE_ROW`	cells: string[], isHeader: boolean	`| A | B |`
`IMAGE`	alt: string, url: string	`![Logo](logo.png)`
`BLOCKQUOTE`	text: string	`> Quote`
`HORIZONTAL_RULE`	none	`---`

Parsing Algorithm:

1. Split markdown into lines
2. For each line, check patterns in order:
   - Heading: starts with #
   - List item: starts with - or * or number.
   - Code block: starts with ```
   - Table row: contains |
   - Blockquote: starts with >
   - Horizontal rule: line of ---, ***, ___
   - Otherwise: paragraph text
3. Group consecutive paragraph lines
4. Extract inline formatting (bold, italic, links, inline code) using regex
5. Return array of tokens

4.2 Document Converter

Conversion Process:

1. Take token array from parser
2. Group related tokens (e.g., consecutive LIST_ITEMs become one List)
3. Process inline formatting markers
4. Resolve image paths (relative to input file)
5. Build element tree
6. Generate IDs for headings (for potential TOC)
7. Return Document object

4.3 PDF Renderer

Rendering Strategy:

Element Type	Rendering Method	Styling
Heading	Draw text with larger font	fontSize based on level (28pt - 14pt), bold
Paragraph	Draw text with wrapping	fontSize 12pt, lineHeight 1.5
List	Draw bullet/number + indented text	Indent 20pt, bullet or number prefix

Code Block	Draw text in monospace font with background	Background gray, Courier font, no wrapping
Table	Draw cells with borders	Calculate column widths, draw borders
Image	Embed image, scale to fit	Max width: page width minus margins
Blockquote	Draw with left border and indent	Left border 3pt, indent 15pt, italic
Horizontal Rule	Draw line across page	Line from margin to margin

Page Layout:

Setting	Value	Description
Page Size	A4 (595 x 842 points)	Standard paper size
Margins	72 points (1 inch) all sides	Top, right, bottom, left
Header	Optional	Page numbers, document title
Footer	Optional	Page numbers
Line Height	1.5	Space between lines
Paragraph Spacing	12 points	Space between paragraphs

Font System:

Font	Usage	File
Helvetica	Body text, headings	Built-in PDF font
Helvetica-Bold	Bold text	Built-in PDF font
Helvetica-Oblique	Italic text	Built-in PDF font
Courier	Code blocks, inline code	Built-in PDF font

Built-in fonts are used to avoid embedding font files, reducing complexity.

4.4 Text Processing

Text Wrapping:

Simple greedy algorithm:

1. Split text into words
2. Measure each word width
3. Add words to current line until width exceeded
4. Start new line, repeat
5. Handle very long words by breaking mid-word if needed

Table Column Width:

1. Measure content width for each cell
2. Find maximum width per column
3. If total width fits page: use measured widths
4. If too wide: scale all columns proportionally to fit

Image Handling:

1. Check if image file exists
2. Load image using PDFKit
3. Get image dimensions
4. Calculate scale to fit page width
5. Embed image at scaled size
6. If image load fails: show placeholder text

5. Configuration and Customization

5.1 Configuration File

Configuration via JSON file or command-line options:

Option	Type	Default	Description
`pageSize`	string	"A4"	A4, Letter, Legal
`marginTop`	number	72	Top margin in points
`marginRight`	number	72	Right margin in points
`marginBottom`	number	72	Bottom margin in points
`marginLeft`	number	72	Left margin in points
`fontSize`	number	12	Base font size in points
`lineHeight`	number	1.5	Line spacing multiplier
`showPageNumbers`	boolean	false	Add page numbers in footer
`pageNumberPosition`	string	"bottom-center"	Position of page numbers
`headerText`	string	""	Optional header text
`footerText`	string	""	Optional footer text

5.2 Styling Options

Style	Configuration	Values
Heading sizes	`heading1Size` - `heading6Size`	Points (default: 28, 24, 20, 18, 16, 14)
Code background	`codeBackground`	Hex color (default: #f5f5f5)
Link color	`linkColor`	Hex color (default: #0366d6)
Blockquote border	`blockquoteBorderColor`	Hex color (default: #ddd)
Table border	`tableBorderColor`	Hex color (default: #000)

6. CLI Interface

6.1 Command-Line Usage

Basic Syntax:

```
md2pdf [options] <input-file>
```

Options:

Flag	Description	Example
`-o <file>`	Output file path	`-o output.pdf`
`--config <file>`	Load config from JSON file	`--config config.json`
`--page-size <size>`	Page size (A4, Letter, Legal)	`--page-size Letter`
`--margin <n>`	Set all margins to n points	`--margin 50`
`--no-page-numbers`	Disable page numbers	`--no-page-numbers`
`-h, --help`	Show help message	`-h`
`--version`	Show version number	`--version`

Examples:

```
md2pdf document.md
md2pdf document.md -o output.pdf
md2pdf document.md --page-size Letter --margin 50
md2pdf document.md --config my-config.json
```

7. Testing Strategy

7.1 Test Coverage

Test Type	Framework	Scope
Unit Tests	Jest	Individual functions: token parsing, text wrapping, element creation
Integration Tests	Jest	Full pipeline: markdown → PDF for various inputs
Benchmark Tests	Jest + Benchmark.js	Performance benchmarks for parsing, conversion, and rendering

7.2 Benchmark Tests

Benchmark	Input	Target
Small Document Parse	100 lines	<5ms

Medium Document Parse	1,000 lines	<50ms
Large Document Parse	10,000 lines	<500ms

7.3 Test Cases

Category	Test Cases
Parser	• All heading levels • Bold and italic combinations • Nested lists • Code blocks with various languages • Tables with different column counts • Links and images • Malformed markdown
Converter	• Token grouping (list items → list) • Inline formatting extraction • Image path resolution • Empty document handling
Renderer	• Text wrapping at page width • Page breaks • Table layout with long content • Image scaling • Multi-page documents
CLI	• File reading • Option parsing • Error messages • Config file loading

7.4 Example Markdown Files

Test with various markdown documents:

Document	Purpose
`simple.md`	Basic text, headings, lists
`formatting.md`	Bold, italic, code, links
`tables.md`	Various table layouts
`code.md`	Code blocks with different languages
`images.md`	Images with different sizes
`long.md`	Long document testing page breaks
`edge-cases.md`	Unusual formatting, edge cases