AI-Powered Markdown Parser with Mermaid Support

Andrea Mussari

January 22, 2026

Contents

1 Introduction

The goal of this project is to use AI to design and implement a Markdown parser that supports standard Mardown features across multiple blocks of Mermaid code.

I chose to use Claude (Sonnet 4.5) because in my personal experience using AI in programming, such as AdventCode or Leetcode challenges, it is the one that produces the best results.

For the choice of programming language, I asked Claude directly which one is best for him.

2 Development Process

The section is divided into paragraphs where each paragraph represents a stage in the development of the project.

2.1 Prompt #1

"I would like your help designing and implementing a Markdown parser that supports standard Mardown features across multiple blocks of Mermaid code, producing a structured representation that can be rendered in HTML with special handling for Mermaid diagrams.
I would like your opinion on the best programming language to carry out a project of this type."

With this prompt, my aim was to make the goal as clear as possible to Claude and to verify whether he had correctly identified the context of the project.

Claude replied that in his opinion the most suitable programming language is Javascript/Typescript, giving several reasons:

- Ecosystem fit.
- Excellent existing libraries such as *marked*, *mardown-it* or *remark* for Markdown and *mermaid* library for diagram rendering.
- Universal deployment.
- Strong typing with Typescript.

This list of reasons is a compelling argument for choosing Javascript/Typescript as the programming language for the project, but Claude still gave me a list of programming language alternatives: Python, Rust, and Go; and for each, he gave advantages and disadvantages over choosing Javascript/Typescript.

He then suggested a scheme for the architecture, independent of the choice of programming language, divided into:

1. Lexer/Tokenizer.
2. Parser.
3. Renderer.
4. Post-processor.

Finally he asked me if I preferred a full parser from scatch or a pratical solution using existing libraries.

2.2 Prompt #2

**"I would prefer a practical solution in Typescript that uses existing libraries.
Set up the code so that each component has a corresponding file. If the files share features, create a module file to optimize, and finally create a main project file."**

The intent of this prompt is to test its most practical and fastest solution while still giving it an idea of code organization that would allow me to more easily identify component problems.

In fact, the following files were eventually generated:

- README.md
- src/types.ts - Core type definitions and interfaces.
- src/utils.ts - Shared utility functions (ID generation, validation, escaping).
- src/lexer.ts - Tokenization using the *marked* library.
- src/mermaidProcessor.ts - Mermaid diagram extraction and processing.
- src/renderer.ts - HTML rendering with custom Mermaid handling.
- src/parser.ts - Main orchestrator that coordinates all components.
- src/index.ts - Public API with convenience functions and demo code.
- package.json
- tsconfig.json

I decided to try the demo he created. To do this, you need to run three commands in order: *npm install*; *npm run build*; *npm test*.

The *npm run build* command that matches tsc, transpiler typescript to javascript, stops finding 11 errors in 3 files.

2.2.1 Error analysis

ReferencesErrorAnalysis

As previously mentioned, tsc has identified 11 errors, 9 of which are repetitions of the same error, so we can say that there are 4 distinct errors:

- `error TS2694: Namespace ‘‘/.../node_modules/marked/lib/marked’’.marked’ has no exported member ‘Token’.`
- `error TS2694: Namespace ‘‘/.../node_modules/marked/lib/marked’’.marked’ has no exported member ‘Tokens’.`
- `error TS2749: ‘marked.Renderer’ refers to a value, but is being used as a type here. Did you mean ‘typeof marked.Renderer’?`
- `error TS2353: Object literal may only specify known properties, and ‘headerIds’ does not exist in type ‘MarkedOptions’.`

2.2.1.1 Error #1 and #2

This error occurs in the `lexer.ts` and `mermaidProcessor.ts` files. After some research, checking the package.json file, I discovered that the package marked in the "dependencies" entry was not set to the latest available version, which resulted in the @types/marked package being added to the "devDependencies" entry, which is deprecated because in the latest marked versions it provides its own type definitions.

Claude's error can be corrected by updating the import statement and replace all instances of marked.Token and marked.Tokens with Token and Tokens respectively.

Listing 1: lexer.ts and mermaidProcessor.ts

```
import { marked } from 'marked'; // Original import causing errors
import { marked, type Token, Tokens } from 'marked'; // Corrected import
```

2.2.1.2 Error #3

This error occurs in `renderer.ts` file. The issue is analogous to the previous one, and can be resolved by updating the import statement and replace all instances of marked.Renderer with Renderer respectively.

Listing 2: renderer.ts

```
import { marked } from 'marked'; // Original import causing errors
import { marked, Renderer } from 'marked'; // Corrected import
```

2.2.1.3 Error #4

This error occurs in `renderer.ts` file. The issue arises because the headerIds and mangle options doesn't exist in the MarkedOptions type. To resolve this, simply remove the line that sets the headerIds and mangle options in the marked.setOptions call.

Listing 3: renderer.ts

```
marked.setOptions({
    renderer: this.renderer,
    gfm: this.options.gfm ?? true,
    breaks: this.options.breaks ?? true,
    headerIds: this.options.headerIds ?? true, // Remove to fix error
    mangle: false // Remove to fix error
});
```

After making these corrections, the code compile successfully without any errors.

3 Testing

After making the changes described in the 2.2.1 section, I ran the *npm run build* command again and this time the compilation was successful without errors.

To run my tests I created a test set consisting of:

- Big.md - A large markdown file with multiple mermaid diagrams and standard markdown features.
- CodeBlocks.md - A markdown file focusing on code blocks, including some programming languages.
- Emphasis.md - A markdown file testing various emphasis styles (bold, italic, together).
- Links.md - A markdown file testing different links (my github page, project's chat, ...).
- Lists.md - A markdown file testing ordered and unordered lists, including nested lists.
- Mermaid.md - A markdown file focusing on various mermaid diagrams (flowchart, sequence diagram, class diagram, ...).
- for each of these files, I created the corresponding expected HTML output file.

I created the HTML files using the site `https://convertmarkdowntohtml.com/`, the site does not natively support *Mermaid.js* so to make it compatible I replaced every occurrence of "language-mermaid", which was automatically generated by the site, with mermaid.

I created `test.js` that imports the parser generated by Claude and uses it to convert markdown files to HTML and generate `.report.html` files structured like this:

- Markdown Input.
- Expected HTML.
- Actual HTML.
- Diff.

The test results were almost perfectly positive, the only exceptions coming from the `Big.md` and `Mermaid.md` files whose actual HTML had small discrepancies with the expected HTML, as far as the Mermaid Blocks part is concerned, however these tests are also considered positive because even if different, the HTML generated by Claude renders the various diagrams correctly.

4 Conclusion

Claude's practical solution proved to be almost completely correct, without requiring further code generation I identified some important errors, because they did not allow the execution of npm run build, and other less important ones such as the use of packages not updated to the latest version and deprecated and unnecessary packages. That said, as reported above, by making minimal corrections, correcting imports from libraries, it makes npm run build not only executable but also allows you to verify the correct behavior of the code.

In conclusion, we can say that although not perfect from the beginning, Claude's code was more than sufficient and correct for the purpose of this project, thus demonstrating that the use of AI in the development phase of small projects like this greatly speeds up the writing of the code and involves a debugging phase small enough that it is not a burden for the programmer.

5 Links

- Project Repository: `https://github.com/Muxy03/AI-Parser`
- Claude chat: `https://claude.ai/share/d2c001c4-bb66-4cdf-97c4-05acc550336c`