Markdown Parsing

CS 297 - Aditya Prajapati

Introduction

Markdown

- A lightweight markup language for formatting text
- Commonly used for documentation, websites, and readme files

Parser

- Processes and interprets structured data
- Converts Markdown into HTML in this case

Elements

Basic syntax elements:

- Headings (e.g., # H1, ## H2, ...)
- Emphasis (e.g., *italic*, **bold**)
- Lists (e.g., - Unordered and 1. Ordered)
- Links (e.g., [title](url))
- Images (e.g., ![alt text](image_url))
- Code Blocks (e.g., `inline code` and multiline with triple backticks)

Parsing

Tokenization

- Break down Markdown text into identifiable tokens for each syntax type
  - Ex. Recognize # at the beginning of a line as a heading token

- Use regular expressions to identify patterns, like ^#{1,6}\s for headings or *.** for emphasis.

Syntax Tree

- Organizes tokens hierarchically and preserves the structure of nested elements
- Each Markdown element (e.g., heading, list item, link) becomes a node
- Sub-elements, like list items within lists, are nested as child nodes

Parsing syntax tree to HTML

- Traverse the syntax tree and convert each node to HTML.
  - Example: Heading node <h1>Title</h1>, list node Item

- Nested elements: Properly manage lists, quotes, and other elements that can contain multiple levels

Challenges

- Edge Cases and Ambiguities
  - Handle syntax errors and mixed formatting like # Heading with *italic*
- Security and HTML Escaping
  - Prevent injection attacks by escaping HTML
- Extensibility and Performance
  - Add support for more Markdown features (e.g., tables, footnotes) and optimize for large files

Tips and tricks

- Handle Elements in the Correct Order - order matters
  - Capture most specific -> most general : Parse headings, lists, and tables before paragraphs to avoid unintended transformations
- Lazy Evaluation and Streaming
  - For large Markdown files, consider parsing with a streaming approach to handle content as it’s read, rather than loading it all into memory.
- Store markdown in db and parse it when needed?
- Store HTML as a temporary file on server
  - Store name as hash of content, so no need to parse again if hash has not changed