SIMPLE MD TO HTML

Development documentation

Exposee

This document describes the simple markdown to html parser tool from developer view. The parsing process is shown and the several technologies are listed.

Ricardo Krause

Kaser23@gmx.de

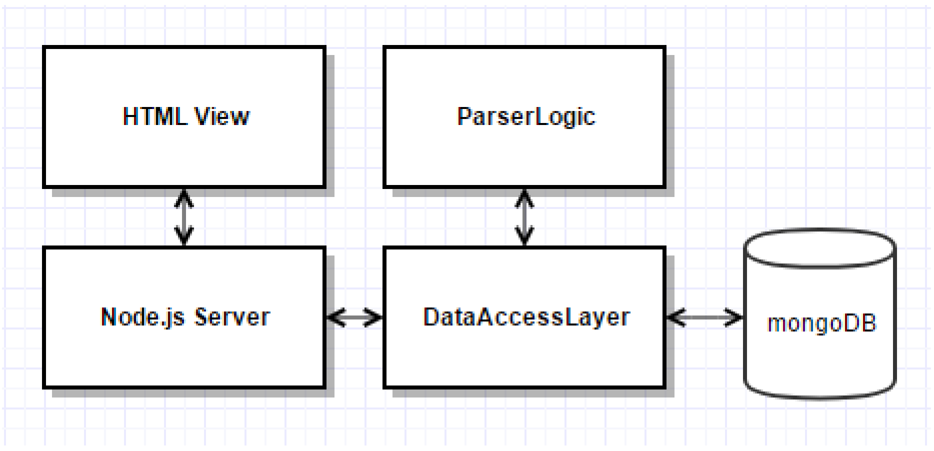
Content

Abstract Design

The application is on an abstract level separated in five functional parts that are loosely coupled and are easy to reuse.

- **HTML View**
  Contains the user interface code, including the index.ejs (main view) and the html, css and script files for the web ui.
- **Node.js Sever**
  Acts as application core logic and contains the glue code for the configuration of all software parts.
- **DataAccessLayer**
  Is in response for the communication between the application and the database, there the data are stored.
- **ParserLogic**
  This software part replaces the markdown strings to plain html strings.
- **MongoDB**
  Simple document based database to persist a data for the application.

The abstract design is shown in Figure1.



Detail Design

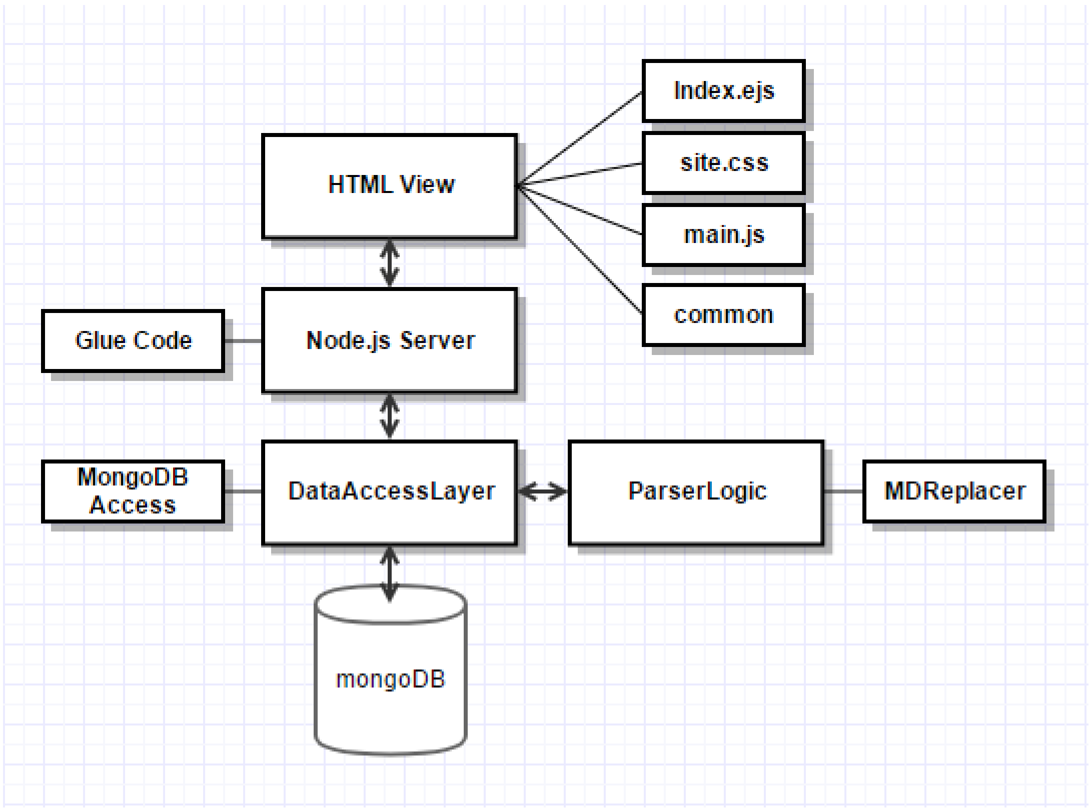


Figure 2: Detail Design

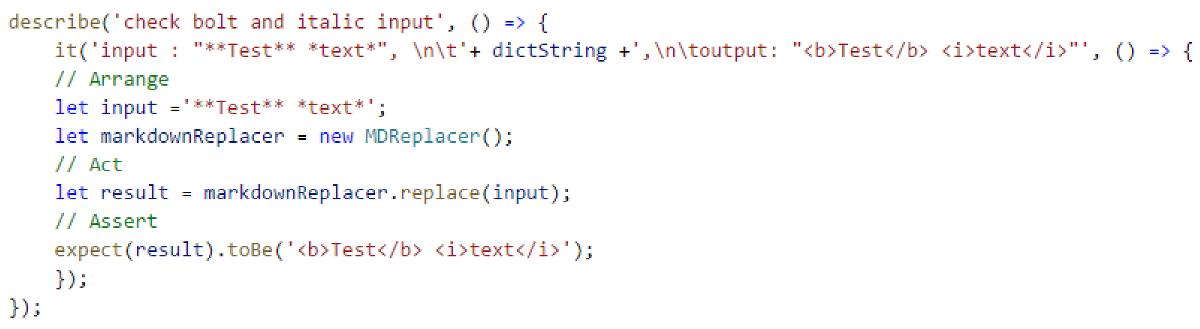
Concepts

This chapter describes the main concept that was used for the implementation of the simple markdown to html parser. It's mainly intended to provide a better understanding of the code.

Unit Tests

This project proceeded in test driven development, so the first step was to create a specification what fits the syntax constrains of the task. In this project, the testing framework jasmine was used. The test framework was running during the development process so it was easy to check if a parsing step was successful or needs to be changed.

The test is split in three parts, the arrangement of test environment, the act part there the code is running and an assertion where we compare the result with an expected value.



```
describe('check bolt and italic input', () => {
    it('input : "**Test** *text*", \n\t'+ dictString +',\n\toutput: "<b>Test</b> <i>text</i>"', () => {
    // Arrange
    let input ='**Test** *text*';
    let markdownReplacer = new MDReplacer();
    // Act
    let result = markdownReplacer.replace(input);
    // Assert
    expect(result).toBe('<b>Test</b> <i>text</i>');
    });
});
```

Figure 3: unit test example with jasmine

Parsing process

The parsing of the markdown input strings in done by the **MDReplacer**, that is used by the **DBAccess** class to combine the input string with the parsed output string in one transaction. The main feature of the replacer is to parse a markdown string to html. Intern use the replacer some small helper functions and properties to procced the process.

- **dictionary (): any**
  Contains the parsing syntax.
- **replace (input: string): string**
  This method splits the input string in blocks and parse each line separate.
- **replaceKey (input: string, key: string): string**
  Replace the md tokens to html tags by simple key replacement.
- **replaceDouble (input: string, key: string): string**
  If the markdown token has a start and end token, this method replaces both.
- **startsWith (input: string, key:string): boolean**
  A small helper method to check if the current token is a starting token
- **conbineListElements** (input: string): string
  Combine multiline list elements to one list

As you can see, the parsing is separated in different small steps. This has the advantage that we can reuse several methods for different steps.

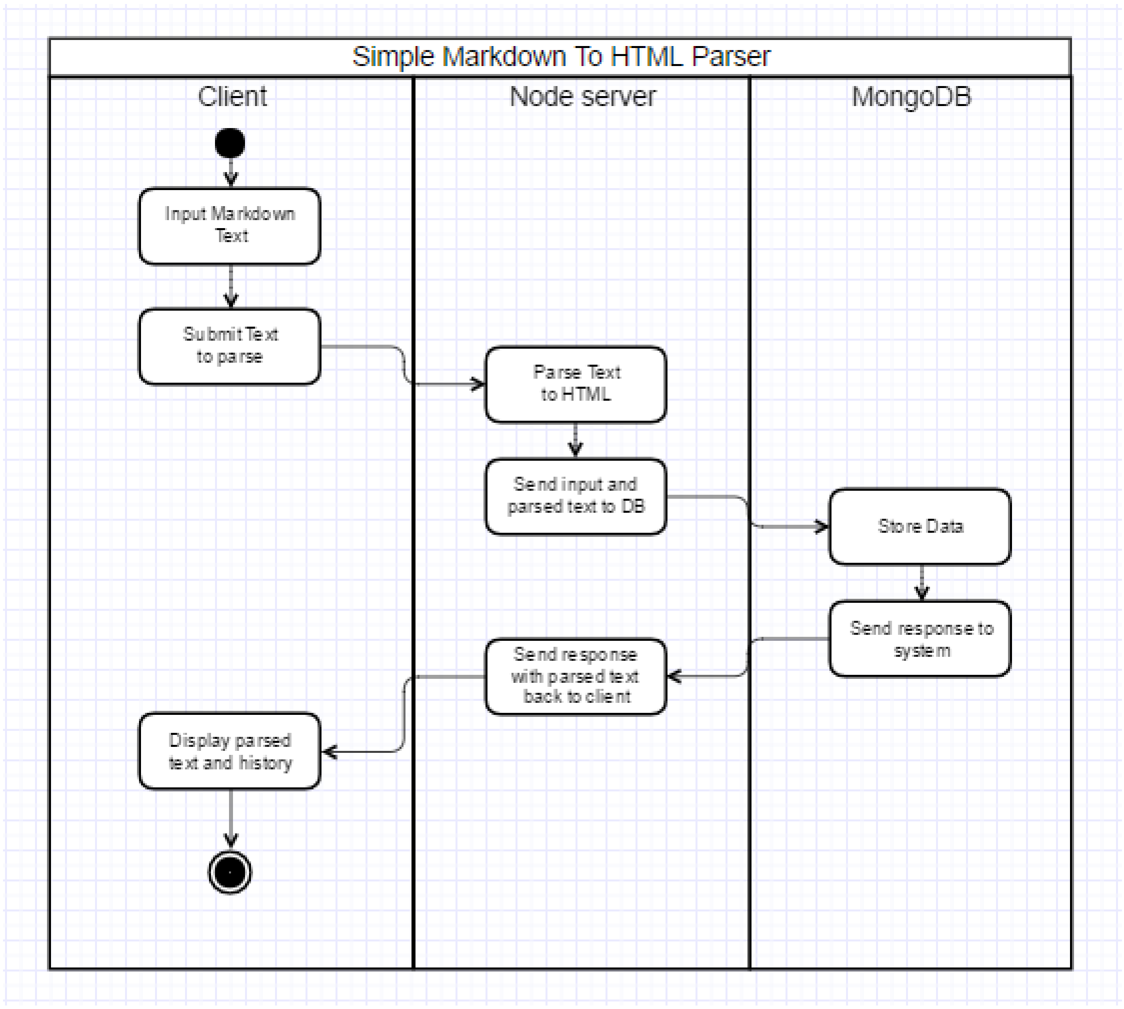

Simple Markdown To HTML Parser
Client
Node server
MongoDB
Input Markdown Text
Submit Text to parse
Parse Text to HTML
Send input and parsed text to DB
Store Data
Send response to system
Send response with parsed text back to client
Display parsed text and history

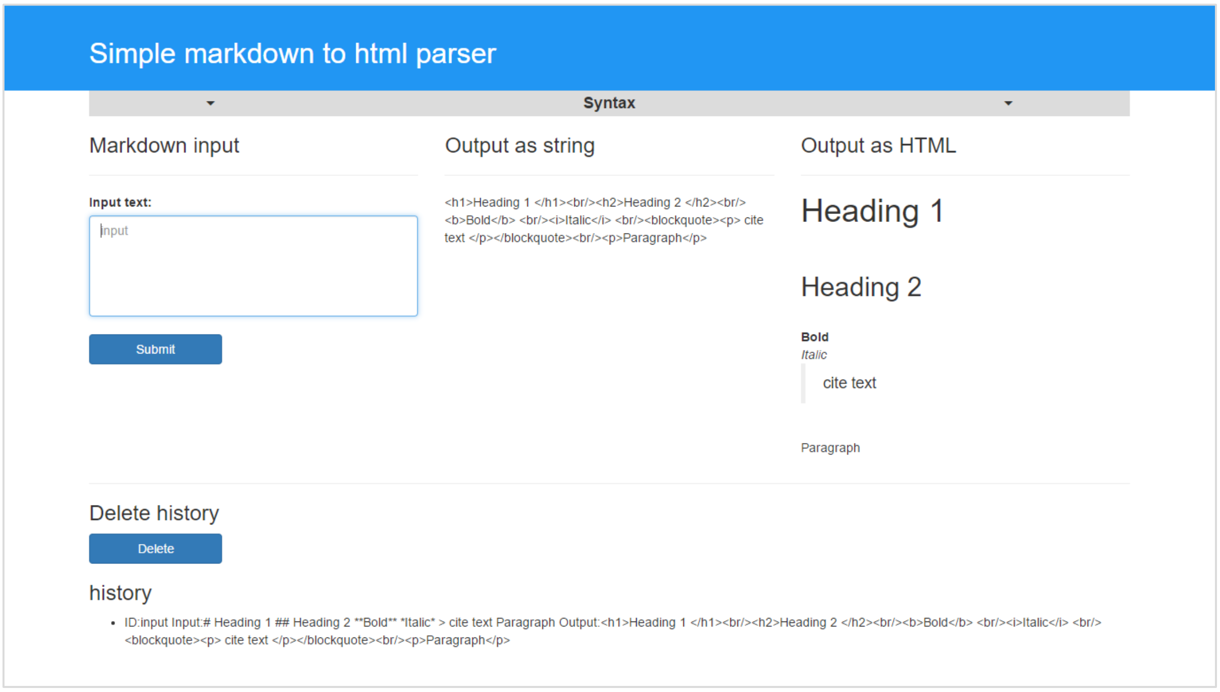
Figure 4: Parsing activity

Visualization

For the visualization, a node.js server with the express framework is running in the background. This server has the standard CRUD functionality implemented in form of http get, post, put and delete.

On the client side is the embeddedJS framework view engine running. This allows to use JavaScript directly in the view next to the HTML5. The view implements the twitter bootstrap framework to reach a responsive and modern design.

The view is column based oriented if there is enough space, on small screen the content is displayed only in one column.



Simple markdown to html parser

Syntax

Markdown input

Input text:

Output as string

```
<h1>Heading 1 </h1><br/><h2>Heading 2 </h2><br/><b>Bold</b> <br/><i>Italic</i> <br/><blockquote><p> cite text </p></blockquote><br/><p>Paragraph</p>
```

Output as HTML

Heading 1

Heading 2

Bold
Italic
cite text

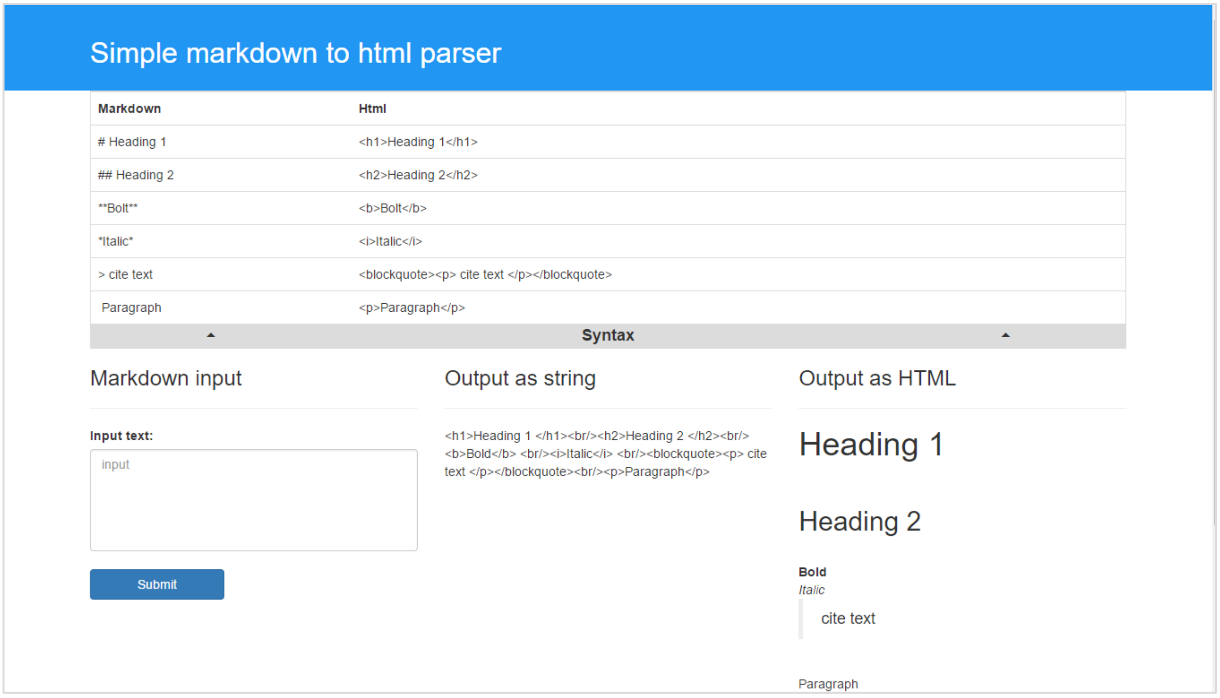
Paragraph

Delete history

history

- ID:input Input:# Heading 1 ## Heading 2 **Bold** *Italic* > cite text Paragraph Output:<h1>Heading 1 </h1><br/><h2>Heading 2 </h2><br/><b>Bold</b> <br/><i>Italic</i> <br/><blockquote><p> cite text </p></blockquote><br/><p>Paragraph</p>

Figure 5: HTML view



Simple markdown to html parser

Markdown	Html
# Heading 1	<h1>Heading 1</h1>
## Heading 2	<h2>Heading 2</h2>
Bolt	<b>Bolt</b>
Italic	<i>Italic</i>
> cite text	<blockquote><p> cite text </p></blockquote>
Paragraph	<p>Paragraph</p>

Syntax

Markdown input

Input text:

Output as string

```
<h1>Heading 1 </h1><br/><h2>Heading 2 </h2><br/><b>Bold</b> <br/><i>Italic</i> <br/><blockquote><p> cite text </p></blockquote><br/><p>Paragraph</p>
```

Output as HTML

Heading 1

Heading 2

Bold
Italic
cite text

Paragraph

Figure 6: Syntax view

Data storage

As additional feature a small history is implemented. For this purpose, a mongoDB cloud storage is implemented. Attention at startup the storage will be cleared, independent of the running instances.