

readme.md for @push.rocks/smartmarkdow n

do more with markdown files

Install

To install `@push.rocks/smartmarkdown`, you can use npm or yarn. Run one of the following commands in your terminal:

```
npm install @push.rocks/smartmarkdown --save
```

or if you use yarn:

```
yarn add @push.rocks/smartmarkdown
```

This module is designed to be used in a Node.js environment or in a frontend project that supports ES Modules.

Usage

`@push.rocks/smartmarkdown` offers powerful tools to work with markdown files, including parsing markdown to HTML, extracting frontmatter data, and converting HTML back to markdown. Below, we'll explore how to utilize these features effectively.

Parsing Markdown to HTML

Let's start by converting a simple Markdown string to HTML:

```
import { SmartMarkdown } from '@push.rocks/smartmarkdown';

async function parseMarkdown() {
  const mdString = `# Hello World

This is a simple markdown string.`;
  const htmlString = await SmartMarkdown.easyMarkdownToHtml(mdString);
  console.log(htmlString); // Logs the HTML string
}

parseMarkdown();
```

In this example, `SmartMarkdown.easyMarkdownToHtml` is a convenient static method that takes a markdown string and returns its HTML representation.

Working with Frontmatter

Markdown files often contain frontmatter, which is metadata specified at the top of the file.

`@push.rocks/smartmarkdown` can extract this data along with converting the content.

```
import { SmartMarkdown } from '@push.rocks/smartmarkdown';

async function parseMarkdownWithFrontmatter() {
  const markdownWithFrontmatter = `---
title: "My First Post"
date: 2023-01-01
tags: ["blog", "post"]
---

# Hello World

This is a post with frontmatter.`;

  const smartmarkdownInstance = new SmartMarkdown();
  const mdParsedResult = await
smartmarkdownInstance.getMdParsedResultFromMarkdown(markdownWithFrontmatter);
  console.log(mdParsedResult.frontmatterData); // Logs: { title: "My First Post", date: "2023-
01-01", tags: ["blog", "post"] }
  console.log(mdParsedResult.html); // Logs: HTML content
}
```

```
parseMarkdownWithFrontmatter();
```

In the code above, we manually create an instance of `SmartMarkdown` to access the `getMdParsedResultFromMarkdown` method, which returns an object containing both the HTML and the frontmatter data.

HTML to Markdown Conversion

Sometimes, you may need to convert HTML back to Markdown. Here's how you can do it:

```
import { SmartMarkdown } from '@push.rocks/smartmarkdown';

const smartmarkdownInstance = new SmartMarkdown();
const htmlString = '<h1>Hello World</h1><p>This is a simple HTML string.</p>';
const markdownString = smartmarkdownInstance.htmlToMarkdown(htmlString);
console.log(markdownString);
```

In this example, we use the `htmlToMarkdown` method to convert an HTML string back to Markdown. This is particularly useful when working with content editing interfaces that support both formats.

Advanced Usage: Parsing Markdown with Custom Plugins

`@push.rocks/smartmarkdown` leverages Unified, Remark, and Turndown to provide its functionality. You can extend its capabilities by using custom plugins.

For instance, if you wanted to use a custom Remark plugin to highlight code blocks, you would:

1. Create a new `SmartMarkdown` instance.
2. Configure it to use your custom plugins.
3. Process your markdown content.

```
// This is a hypothetical example. Please refer to the respective plugin documentation for actual implementation details.
```

```
import { SmartMarkdown } from '@push.rocks/smartmarkdown';
import myRemarkPlugin from 'remark-myplugin';

async function parseMarkdownWithPlugin(mdString: string) {
```

```
const smartMarkdownInstance = new SmartMarkdown();
smartMarkdownInstance.useRemarkPlugin(myRemarkPlugin, pluginOptions);
const result = await smartMarkdownInstance.getMdParsedResultFromMarkdown(mdString);
// Now `result` will include transformations done by your custom plugin.
}
```

`@push.rocks/smartmarkdown` provides a solid base for performing various markdown related tasks in your projects. Whether parsing, transforming, or exporting content, its utilities and integrations offer flexibility for most needs. By leveraging ES Modules and TypeScript, it ensures type safety and enhances developer experience with excellent IntelliSense support.

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the [license](#) file within this repository.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH

Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Updated 2026-03-28 12:18:21 UTC by foss.global Team