Accessibility4md

Demonstration of checking and decorating the accessibility in markdown file

Jae Dong Hwang

# Introduction

[Markdown](https://en.wikipedia.org/wiki/Markdown) is a type of language that's used to create formatted text using a simple text editor. It's commonly used in many text editing workflows because it's lightweight and easy to use. For example, it's often used for documents in a code repository, like the README.md file. People also use markdown when creating blog posts or webpages because it lets them focus on editing the content instead of worrying about the formatting. The markdown framework automatically transforms the markdown file into an HTML file, making it easier to publish the content online.

## Motivation

Two factors inspired this project. Firstly, the second homework assignment ([Website/App Accessibility Assessment](https://courses.cs.washington.edu/courses/csep590b/23wi/assignments/website.html)) in our course allowed us to gain knowledge about standard guidelines for accessibility and identify simple ways to address accessibility issues in websites and apps.

While exploring accessibility checking tools, we realized that instead of simply separating the publisher and validator (tool), it would be more beneficial to assist the publisher in improving accessibility of the contents during their workflow. Combined led to consider examples where we could address accessibility issues at an earlier stage.

Thus, the motivation for this project was born - can we integrate accessibility checks and improvements into the markdown workflow as an example?

# Related Work

After conducting preliminary research online, we could not find a project directly related to our goal. Most accessibility tools are available for download and focus on checking and evaluating accessibility in the final product, such as HTML files. This may be due to our project's unique focus on evaluating and improving accessibility within the markdown workflow. Additionally, adding more checkers could reduce the workflow flexibility that markdown offers.

However, we did find that many major programming languages provide accessibility APIs, such as the [Java Accessibility API (JAAPI)](https://docs.oracle.com/en/java/javase/11/access/java-accessibility-overview.html#GUID-17F9FD40-E191-41CE-BCF9-D956F1EF5111), which allows developers to create accessible applications. However, these APIs primarily target improving the accessibility of GUI applications and not the workflow of producing documents. Nonetheless, it's still beneficial that major programming languages have APIs that support application development with accessibility features

# Methodology

### Guidelines

Firstly, the project starts with identifying and scoping the requirements for the tool. The [WCAG2.1](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244&currentsidebar=%23col_overview&technologies=html%2Ccss%2Cjs%2Cserver%2Csmil%2Cpdf%2Cflash%2Csl#text-alternatives) and [W3 ATAG](Authoring%20Tool%20Accessibility%20Guidelines%20(ATAG)%202.0%20(w3.org)) guidelines were the main resource to filter out the applicable requirements for this project. Here are some highlight guidelines from ATAG for this project. The detailed requirements are in the Appendix.

* B.1.1.2 Content Auto-Generation During Authoring Sessions (WCAG): If the authoring tool provides the functionality for automatically generating web content during an authoring session, then at least one of the following is true: (Level A to meet WCAG 2.0 Level A success criteria; Level AA to meet WCAG 2.0 Level A and AA success criteria; Level AAA to meet all WCAG 2.0 success criteria)
  + Accessible: The content is accessible web content (WCAG) without author input;
* B.2.4.1 Accessible Template Options (WCAG): If the authoring tool provides templates, then there are accessible template (WCAG) options for a range of template uses. (Level A to meet WCAG 2.0 Level A success criteria; Level AA to meet WCAG 2.0 Level A and AA success criteria; Level AAA to meet all WCAG 2.0 success criteria)
* B.2.3.2 Automating Repair of Text Alternatives: The authoring tool does not attempt to repair text alternatives for non-text content or the following are all true: (Level A)
* B.3.2.1 Repair Assistance (WCAG): **If checking (see Success Criterion B.3.1.1) can detect that a WCAG 2.0 success criterion is not met, then repair suggestion(s) are provided**: (Level A to meet WCAG 2.0 Level A success criteria; Level AA to meet WCAG 2.0 Level A and AA success criteria; Level AAA to meet all WCAG 2.0 success criteria)

### High-level Implementation

Once defining the scope of requirements for the project, we designed the implementation.

* Extension class per requirement
* Processors for each extension class
* Implementation of visitor class for Markdown parsed tree structure.
* Façade class that iterates the sample files and produces the output.

The solution for the promise is realized as a set of extensions that handles accessibility scenarios individually. The implementation highlighted three aspects.

## Accessibility Check in markdown.

Capturing the knowledge of accessibility guidelines in the checker, we implemented a couple of extensions class that checks and provide feedback about the potential improvement.

One example is the missing alt check. The altcheckextension.py finds the image node to check whether the alternative text exists.



Another example is whether the alternative texts are duplicated. The extension also detects the duplicate alternative texts in the near element in the document.



## New syntax and decoration for HTML element.

The tool also demonstrates that we can add the markdown syntax to add the attributes in the produced HTML content. The tool allows a new syntax, a pair of ==, in the markdown. When an author use the syntax, the element in HTLM will have a role attributes.

|  |  |
| --- | --- |
| Markdown | **# Detecting Role Syntax ==article==** |
| HTML | <h1 role="article">Detecting Role Syntax </h1> |

One example demonstrates the chain of extensions to produce the desired output. The example below is the output of two extensions implemented: AreaTableExtension and AreaTableIndexExtension. The first constructs the HTML table and the later decorate the table with area-rowcount, area-rowindex, and role.

|  |  |
| --- | --- |
| Markdown | |Header1|Header2|  |-|-|  |Row0 Cell1|Row0 Cell2|  |Row1 Cell1|Row1 Cell2| |
| HTML | <table aria-rowcount="2" role="table">      <thead role="rowgroup">          <tr aria-rowindex="None" role="row">              <th role="columnheader">Header1</th>              <th role="columnheader">Header2</th>          </tr>      </thead>      <tbody role="rowgroup">          <tr aria-rowindex="1" role="row">              <td role="cell">Row0 Cell1</td>              <td role="cell">Row0 Cell2</td>          </tr>          <tr aria-rowindex="2" role="row">              <td role="cell">Row1 Cell1</td>              <td role="cell">Row1 Cell2</td>          </tr>      </tbody>  </table> |

## Expanding with media library

During development, we confirmed that there is the possibility of integrating the external libraries to validate accessibility more intensively. For example, openCv and numpy library allows the tool to check the contrast of the image.

We can embed more intense but light-weighted image processing. For example, extracting the texts and comparing them with their background. The implementation of the functionality should relatively be easy with some robust design.

# Disability Justice Perspective

This project mainly relates to the second principle, [Leadership of Those Most Impacted](https://static1.squarespace.com/static/5bed3674f8370ad8c02efd9a/t/5f1f0783916d8a179c46126d/1595869064521/10_Principles_of_DJ-2ndEd.pdf). This project targets to influence the authoring markdown workflow by providing the checker and decorator the accessibility. The tool, a set of extensions, understands the gap in the accessibility of the markdown files in authoring and provides feedback or produces decorative output. The tool can be played as a compile-time influencer in the workflow and is an example of how other content producer tools can play a similar role in many cases.

# Learnings and Future Work

The existing degree of flexibility in markdown workflow was a big consideration when defining this project's scope. We shouldn't impact the freedom of producing the markdown in form. Instead, the tool provides the author the soft feedback to check the accessibility and add accessibility attributes.

This project demonstrates the extensibility of the popular python-markdown library to support accessibility. One possible expansion would be the [Visual Studio Code](https://code.visualstudio.com/api) markdown preview plugin. It should quickly highlight and output the warning message in the console with potential improvements. The near-real-time feedback would help the author to improve the accessibility of the file.

# How you made your app accessible

The library is accessible in github as a public library and we are reaching out to the owner of python-markdown repository to include as part of their library.

# References

* [How to Meet WCAG (Quickref Reference) (w3.org)](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244&currentsidebar=%23col_overview&technologies=html%2Ccss%2Cjs%2Cserver%2Csmil%2Cpdf%2Cflash%2Csl#text-alternatives)
* [Authoring Tool Accessibility Guidelines (ATAG) 2.0 (w3.org)](https://www.w3.org/TR/ATAG20/#part_b)
* [Markdown style guide | styleguide (google.github.io)](https://google.github.io/styleguide/docguide/style.html)
* [GitHub - Python-Markdown/markdown: A Python implementation of John Gruber’s Markdown with Extension support.](https://github.com/Python-Markdown/markdown)
* [10\_Principles\_of\_DJ-2ndEd.pdf (squarespace.com)](https://static1.squarespace.com/static/5bed3674f8370ad8c02efd9a/t/5f1f0783916d8a179c46126d/1595869064521/10_Principles_of_DJ-2ndEd.pdf)

# Appendix I. Functional Requirements

This section captures the results of feasibility studies and lists out the guidelines applicable to this project.

## Applicable Criteria and Functional Requirements

### Perceivable

Text Alternatives - Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.

### 1.1 Text Alternatives

1.1.1 [Non-Content – Level A](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C146%2C149%2C244&currentsidebar=%23col_overview&technologies=html%2Ccss%2Cjs%2Cserver%2Csmil%2Cpdf%2Cflash%2Csl#non-text-content)

The tool iterates any links and checks whether the alternative text is provided.

### 1.3 Adaptable

[1.3.1 Info and Relationships and 1.3.2 Meaningful Sequence – Level A](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C146%2C149%2C244&currentsidebar=%23col_overview&technologies=html%2Ccss%2Cjs%2Cserver%2Csmil%2Cpdf%2Cflash%2Csl#info-and-relationships)

The tool checks the structure in the markdown and whether there is a missing sequence.

### 1.4 Distinguishable

[1.4.3 Contrast (Minimum) – Level AA](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C146%2C149%2C244&currentsidebar=%23col_overview&technologies=html%2Ccss%2Cjs%2Cserver%2Csmil%2Cpdf%2Cflash%2Csl#contrast-minimum)

The tool checks whether the linked image of text has a contrast ratio of at least 4.5:1.

[1.4.11 Non-text Contrast – Level AA](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C146%2C149%2C244&currentsidebar=%23col_overview&technologies=html%2Ccss%2Cjs%2Cserver%2Csmil%2Cpdf%2Cflash%2Csl#non-text-contrast)

The tool checks the image in the file has ta contrast ratio of at least 3:1 against adjacent color(s).

### Operable

User interface components and navigation must be operable.

### 2.4 Navigable

[2.4.2 Page Titled – Level A](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=143%2C145%2C146%2C149%2C241%2C244&currentsidebar=%23col_overview&technologies=html%2Ccss%2Cjs%2Cserver%2Csmil%2Cpdf%2Cflash%2Csl#page-titled)

The first heading should be a level one heading and should be the same or nearly the same as the file name. The first level on heading is used as the page <title>. Tool warns on missing title.

[2.4.4 Link Purpose (In Context) – Level A](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=143%2C145%2C146%2C149%2C241%2C244&currentsidebar=%23col_overview&technologies=html%2Ccss%2Cjs%2Cserver%2Csmil%2Cpdf%2Cflash%2Csl#link-purpose-in-context) and [2.4.9 Link Purpose (Link Only)](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145,244#link-purpose-link-only)

The purpose of each link can be determined from the link text alone or from the link text together with its programmatically determined link context, except where the purpose of the link would be ambiguous to users in general. The tool should check whether any link provided text has the description of the purpose of a link.

TBD: Use [aria-label](https://www.w3.org/WAI/WCAG21/Techniques/aria/ARIA8.html) for link purposes.

[2.4.10 Section Heading – Level AAA](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244#section-headings)

TBD. This might be little tricky to determine the missing section heading in markdown file.

### Understandable

### 3.1 Readable

[3.1.4 Abbreviations – Level AAA](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244%2C2410%2C253%2C314%2C412%2C413#abbreviations)

TBD: This might be the extended feature if time is allowed. Tool checks the abbreviation in the contents and provides feedback on whether it has the definition in place.

[GitHub - philgooch/abbreviation-extraction: Python3 implementation of the Schwartz-Hearst algorithm for extracting abbreviation-definition pairs](https://github.com/philgooch/abbreviation-extraction)

[GitHub - allenai/scispacy: A full spaCy pipeline and models for scientific/biomedical documents.](https://github.com/allenai/scispacy)

### Robust

Content must be robust enough that it can be interpreted by a wide variety of user agents, including assistive technologies.

Majority of criteria for Robust automatically meet in the markdown scenario. The library ([python-markdown](https://github.com/Python-Markdown/markdown)) should guarantee to produce the valid format of document (either in html or pdf).

# Appendix II. Comment on Non-Applicable Criteria

Although we can embed the video/audio link in the markdown file, we scope out the [1.2 Time-based media](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244%2C2410%2C253%2C314%2C412%2C413#time-based-media) criteria in the project. It often uses captions or alternative description features from the provider, such as YouTube.

The input handling is not a common scenario where people use the markdown, so the project scopes out the criteria in [2.5 Input Modalities](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244%2C2410%2C253%2C314%2C412%2C413#input-modalities),  [3.2 Predictable](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244%2C2410%2C253%2C314%2C412%2C413#predictable), and [3.3 Input Assistance](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244%2C2410%2C253%2C314%2C412%2C413#input-assistance).

The workflow in using markdown allows people to focus on their writing. All converter tools automatically produce a valid file format, especially in HTML ([4.1.1 Parsing](https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=145%2C244%2C2410%2C253%2C314%2C412%2C413#parsing)).