Accessibility Checker

General Description of Project

My project is a web-based accessibility checker aimed at identifying common accessibility issues on government websites in Puerto Rico. The tool will focus on two key principles of the Web Content Accessibility Guidelines (WCAG): Perceivable, which ensures that information and user interface components are presented in ways users can perceive, and Operable, which requires that interface components and navigation be usable for all (w3.org). This initial version will evaluate five critical accessibility features: color contrast, typography (size and weight), alternative text for images, heading structure, and link/button labeling. These elements were chosen because they are foundational to web accessibility and are often overlooked but do make a major difference in whether a website is usable for everyone, especially people with disabilities or older adults.

To build this accessibility checker, I will use Playwright, a tool that allows me to load and inspect real websites as if am using a browser. This makes it possible to analyze the actual content and styles of each page. For checking color contrast, I'll use a package called wcag-contrast-ratio, which applies WCAG 2.0 AA and AAA formulas to evaluate whether text is readable against its background. All other checks—such as font size, alt text for images, heading order, and button or link labels—will also be handled through Playwright by inspecting the structure and styling of each web page.

I will begin by making sure the tool works properly in the command line, which will help me test and debug the core features. Once the command-line version is working smoothly, I will shift focus to building a web interface where users can paste a website link and view the results in a more user-friendly way. This part of the project will likely be built with Flask or Streamlit, depending on time and customization needs. To enhance the reporting capabilities, I will use Pandas to organize the accessibility findings into structured tables and enable CSV exports for easy sharing or documentation.

Task Vignettes

Task 1: Submitting a Website for Analysis

The user arrives at the accessibility checker's homepage. They see a simple input field prompting them to paste a website URL. The user enters the link to a public government website and clicks the "Check Accessibility" button.

Technical Details:

- Frontend: Flask or Streamlit web form receives the input URL.
- Backend: Python script (using Playwright) loads the target website.

- Validates that the URL is reachable and returns HTML/CSS content.

Task 2: Processing and Displaying Results

After submitting the URL, the user sees a loading screen. Within a few seconds, the system generates and displays a summary of the accessibility audit. The results are organized into five focus areas:

- Color Contrast
- Typography (Size + Weight)
- Alt Text for Images
- Heading Structure
- Link/Button Labels

Each section highlights whether the page passes or fails the check and includes brief explanations.

Technical Details:

- Playwright evaluates HTML/CSS to extract relevant elements.
- wcag-contrast-ratio checks contrast ratios against WCAG standards.
- Results are compiled into a Pandas DataFrame.
- Visual summaries generated using Matplotlib (maybe).
- Output is rendered in the web interface and it can be downloadable as CSV.

Technical "flow"

Blocks and Flow

1) **User and web interface**
    - User inputs the URL, presses button and triggers analysis process.
    - Data type to collect will be a string.
2) **Input handler URL validator**
    - Checks if URL format is correct and if the site is reachable. Validates syntax.
    - Data in: URL(string)
    - Data out: bool, response object
    - If it fails returns an error message in the web interface.
3) **Page loader**
    - Loads page content and extracts relevant elements for analysis.
    - It will collect data HTML/CSS (str), computed styles (dict), list of elements(list).
    - Then it will be converted to JSON for better data analysis.

4) **Accessibility Analyzer**
    a) Contrast Checker (wcag-contrast-ratio)
        - Checks text/background pairs for contrast ratios.
        - Data type: list of tuples, float for results
    b) Typography Checker
        - Checks font sizes and weights.
        - Data type: list or dict of elements and computed styles
    c) Alt Text Checker for images
        - Identifies images with missing or empty alt tags.
        - Data type: list of image tag data
    d) Heading Structure Checker
        - Analyzes semantic order of heading tags H1to H6.
        - Data type: list of tag order
    e) Button/Link Checker
        - Detects missing or unclear labels for buttons/links.
        - Data type: list of tag data with aria attributes
5) **Results Format**
    - Combines all test results into a structured format
    - Builds a pandas DataFrame with columns such as check, element, status, summary.
    - Send summary to the front-end.
    - Data out: DataFrame (for CSV), dict/JSON (for web UI)
6) **User Interface Results**
    - Displays results in a table format.
    - Export result into CSV "Download CSV" button.

Final (self) assessment

The biggest change I had to make was simplifying the project to only analyze one page at a time instead of full websites, which made it more realistic for my current skills. I feel somewhat confident about building it, but the most important challenge is making sure I can extract all the necessary content from a webpage correctly. If that part fails, the rest won't work. Also, I need to make sure the accessibility guidelines are correctly implemented in the code and that the output reflects accurate results. I'm least familiar with handling computed styles and when it comes to Flask making sure the web interface is functional and user-friendly.

Resources:

W3C (World Wide Web Consortium). "w3c/wcag: Web Content Accessibility Guidelines." GitHub, https://github.com/w3c/wcag/.

World Wide Web Consortium. “Web Content Accessibility Guidelines (WCAG) 2.2: Perceivable.” W3C, 5 Oct. 2023, https://www.w3.org/TR/WCAG22/#perceivable.