CST8914 Accessible by Design

Lab 5: Accessibility Testing: Tools and Strategies

A comprehensive (also called hybrid) accessibility testing approach combines automated tools and manual assessment methods. Your task is to conduct hybrid accessibility testing to identify and fix WCAG accessibility issues.

Objectives

In this lab you will:

- Utilize automated accessibility testing browser tools to identify accessibility defects.
- Implement manual testing strategies to uncover accessibility issues that are not identified by automated tools.
- Fix failures detected through both automated and manual testing approaches.

Instructions

- Fork & clone the Accessible University Demo site - GitHub
  - Fork the provided accessible GitHub repository to create your copy.
  - Clone the forked repository to your local machine.
- Add your GitHub repository to your favorite https://github.com/[username]/accessible_u
- Add your GitHub before_u.html published page in your browser favorites for easy access https://[username].github.io/accessible_u/before_u.html
- Use WCAG 2.1 Level A and Level AA references below to confirm your understanding
  - On Brightspace,
    - Week4 - WCAG checklist
    - Week4 - WCAG checklist cheat sheet WebAIM – Accessibility Insights for web,
  - WebAIM's WCAG Checklist,
  - WCAG quick reference

Lab Submission

On Brightspace, upload one Microsoft Word file with answers to questions.

- Include your first name and last name and student number in the filename i.e., Lab4_YourName_YourStudentNumber.docx
- Remember to include URL of your GitHub repository and published fixed page,

Question 1: Automated accessibility testing (browsers extensions)

Use the three accessibility browser extensions below to find and fix **three unique accessibility failures**.

- Wave,
- Axe Dev tools,
- Accessibility insights>fastpath

Tip: On Google Chrome, pin the automated accessibility testing browser extensions so that they remain visible during the lab exercise. (How to Pin and Unpin Extensions from the Chrome Toolbar, YouTube video 3:36min)

Instructions

- Open your before_u.html published page on Google Chrome.
- Run the browser automated accessibility tests on the page using the extension.
    1. Wave,
    2. Axe Dev tools,
    3. Accessibility insights
- Review the scan results.
- Identify and fix each issue using the following steps:
- Step 1: Analyse the issue.
    - Determine the specific accessibility failures in the code to fix.
    - Understand the failure, why it matters and how to fix it.
    - Use Chrome DevTools during your analysis.
- Step 2: Fix the code.
    - Make the necessary changes on your local copy to address the identified issues.
    - Initiate the commit (add a clear description of the error and the fix in the comment).
    - Commit and push fixes to GitHub.
- Step 3: Validate the fix.
    - Re-run the automated accessibility tests on your updated code.
    - Verify that the automated tests now pass without detecting the issue.
- Step 4: Document the failure in the answers sheet
- Repeat the steps 1-4 for all the three issues.

Question 2: Manual accessibility testing on rendered web pages.

Use manual testing strategies (explained in slides Manual accessibility testing section) focused to find **four unique accessibility failures**.

- keyboard functionality,
- visually focused reviews,
- general content checks,
- Accessibility Insights >Assessment (any sub step from steps 2-24).

Instructions

- Open your before_u.html published page on Google Chrome.
- Identify and fix each issue using the following steps:
- Step 1: Find the issue.
    - Employ the manual testing strategies and tools demonstrated in the lecture to find the issue.
    - Determine the specific accessibility failures using Chrome DevTools
- Step 2: Fix the code.
    - Make the necessary changes on your local copy to address the identified issues.
    - Initiate the commit (add a clear description of the error and the fix in the comment).
    - Commit and push fixes to GitHub.
- Step 3: Validate the fix.
    - Re-run the manual tests strategy on your updated code.
    - Run the screen reader if required to confirm issue is fixed.
- Step 4: Document the failure in the answer sheet.
- Repeat the steps 1-4 for all the three issues.

Answer sheet

- **GitHub repository**
  i.e., https://github.com/RababGomaa/accessible_u
- **GitHub published page after fixes**:
  i.e., https://rababgomaa.github.io/accessible_u/before_u.html

Muna Adan Github links:

General accessible repo
https://github.com/Muna0027/accessible_u

Before.u after changes

https://github.com/Muna0027/accessible_u/blob/main/before_u.html

Question 1: Three issues from automated accessibility testing (browsers extensions)

	Name of tools used	**WCAG 2.1 AA Success criteria(s) failing**	**Description of Issue**	**[Sample of compliant code]**
Issue #1	Wave	3.3.2	• [No label] • Missing type=”search” and the text in the search bar should read “text” so user knows they can type in the search bar. [Code snippet] <input id="search-input" class="form-control mr-sm-2" type="search" placeholder="Search">	• Add <label> • Type attribute should read “text” • Implement aria-label <label for="search-input">Search:</label> <input id="search-input" class="form-control mr-sm-2" type="text" role="search" aria-label="Search" placeholder="Search">

Issue #2	Axe Dev tools	3.3.2	• No form label <input type="text" name="name" class="form-control">	• Added form label <label for="name">Name:</label> <input type="text" id="name" name="name" class="form-control">
Issue #3	Accessibility insights>Fastpath	1.4.3	Very low contrast between the background and text colour click here 	Made the contrast between the colour of the background and the colour of the text more vivd by changing the text colour to a complaint colour of blue based on a contrast ratio checker. Click here

Question 2: Manual accessibility testing on rendered web pages.

	Manual testing strategy used	**WCAG 2.1 AA Success criteria (s) failing**	**Description of Issue**	**[Sample of compliant code]**
Issue #1	Keyboard checks	2.1.1	• Tabbing does not take you to start your application <input type="text" name="email" class="form-control" />	• Added form label to <label for="email">Email:</label> <input type="email" id="email" name="email" class="form-control">

Issue #2	Visual checks	1.4.3	• The contrast between the background and the text is low New years eve party 	• Changed the colour of the text so contrast was visible. New Year's Eve Party
Issue #3	Content checks	2.4.4	• Just an unordered list <ul class="list-unstyled"> Contact Us Directions	• Wrapped in a nav element. Should help AT tech to see nav links <nav> Contact Us Directions </nav>
Issue #4	Accessibility Insights >Assessment (any sub step from steps 2-24).	1.3.1	This should be a heading Bienvenido!	Made it a heading <h1 class="heading">Bienvenido!</h1>