# Accessibility and Page Testing

**Objectives**

1. Understand the A11Y checklist and the benefits of Lighthouse.
2. Clone a project from GitHub to allow testing via Lighthouse.
3. Be able to rectify code so it meets the WCAG standards that are set.
4. Explore other Lighthouse audit features such as Performance.

## What are Git, GitHub and GitHub Pages?

As with the previous lab, this lab will use GitHub pages to publish our site, this time to allow the auditing of the accessibility features.

## Fork the GitHub Repo

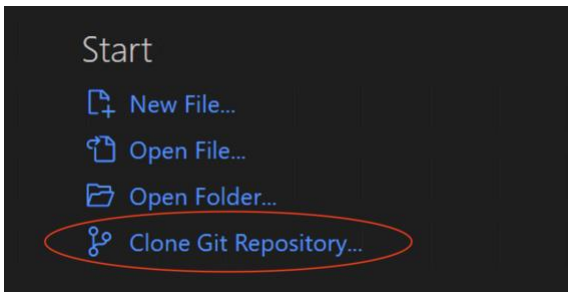
The files for this lab are already in a GitHub repo found at:

https://github.com/mustbebuilt/webDevAccessibilityLab

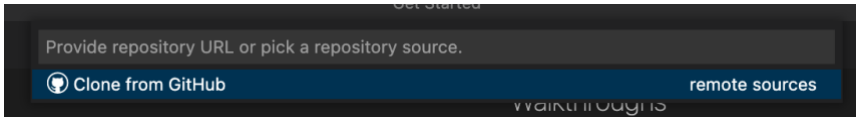
Fork this repo into a one of your own.


## Clone the Repository

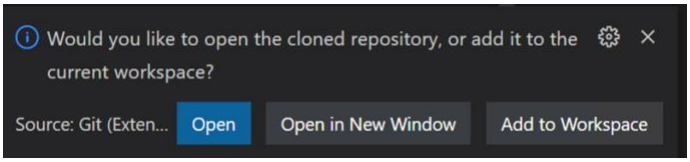
Use the option on the Splash Screen to Clone Git Repository.



Select **Clone from GitHub** and your repos should appear as you type your username into the prompt.




Visual Studio Code will then prompt you to select a target folder on your machine as the location to save your local copy of the repository. Choose somewhere sensible you will be able to easily locate again. Once cloned you are prompted to **Open** the cloned repository. Open it as prompted.




You should now see the project files. There will be several HTML files created for you based on the previous labs. We are doing to audit the files to improve the accessibility of the files.

## A11Y

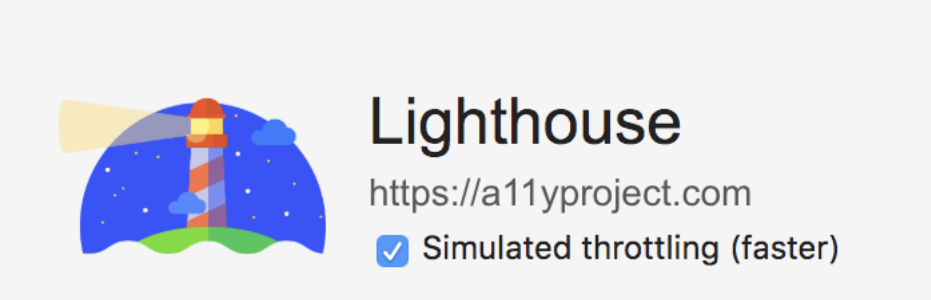
A11Y is a project that focuses on how accessible websites are for people. Websites can be created so they can be understood by screen readers, braille displays and screen magnifiers. There are also disabilities like colour-blindness, dyslexia, seizure triggers that need to be considered. A11y provide a checklist for checking the compliance to WCAG (Web Context Accessibility Guidelines):

https://www.a11yproject.com/checklist/

The above lists core concepts such as images, headings, tables and forms.

## Auditing Pages with Lighthouse

Built into Google Chrome is a tool called Lighthouse. This can automate some but not all of the A11Y checklist and is a useful starting point for compliance.

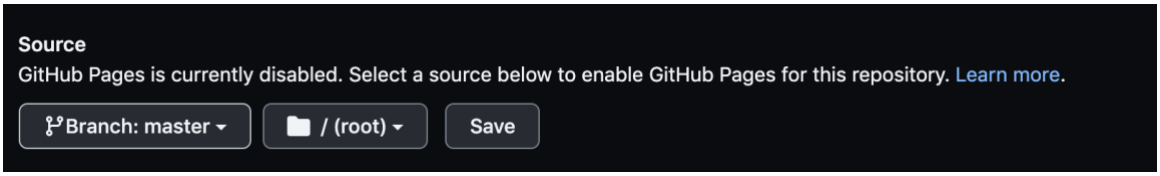



Lighthouse allows for reports to be generated against various criteria including SEO (Search Optimization), Performance and Accessibility. In order to use Lighthouse webpages need to be posted to a webserver, so that Lighthouse to access them and run the audit. Therefore, to test pages via Lighthouse we need to publish our pages to a webserver. There are various ways to do this but we are going to use GitHub Pages.

## Setting Up the local and remote Repos and enabling GitHub Pages

Navigate to your forked version of the GitHub repo and under *Settings* locate *Pages* to enable GitHub pages.

From here you can publish your pages. Under *Source* choose *main/master* and then *Save* your settings.




This will initially take a few seconds, but it will publish your repo to address based on your GitHub username and repo name, for example:

https://mustbebuilt.github.io/webDevAccessibilityLab/

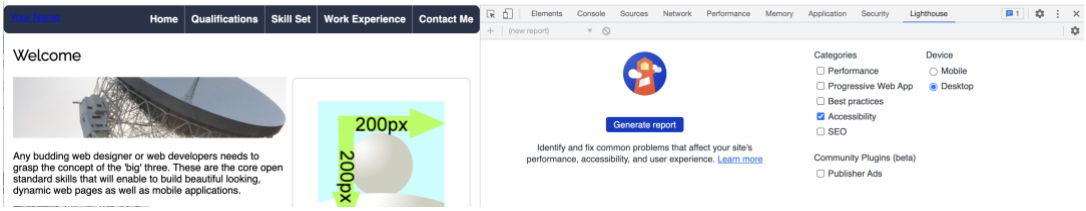
More information on GitHub pages can be found at:

https://pages.github.com/

## Running a Lighthouse Audit

Browse to GitHub pages hosted repo in Google Chrome. Now we have the pages on a webserver we can use Lighthouse to audit them.

Open the developer tool (right-click Inspect) and locate *Lighthouse*. Untick the other tests leaving just *Accessibility* and select *Desktop* for the first audit. You can now run an Accessibility audit against the page.



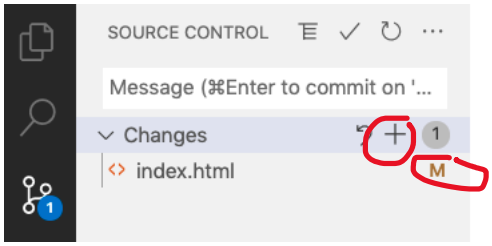

*Tip: To re-run an audit close the developer tools and reopen.*

## Publishing and Re-running a Lighthosue Audit

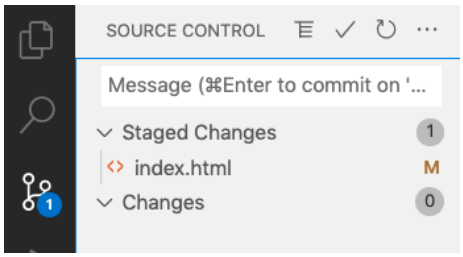
The *index.html* page does not score very highly. We are aiming for a score of 100%.

Lighthouse will give you tips on how to improve the page. As you make improvements save your file and then republish to GitHub Pages. This means you will need to Stage, Commit and Sync the files from Visual Studio Code to GitHub.

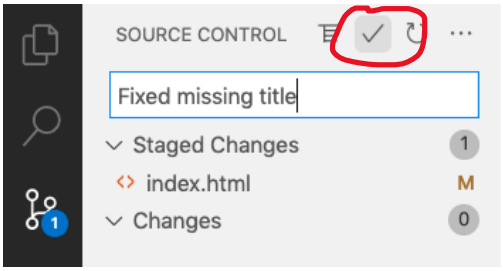
As you make changes, in the Source Code tool you will see your changes flagged as **M** for modified.



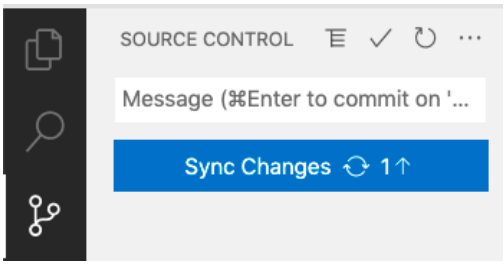

Click the + sign to stage the change, and it then shows in *Staged Changes* list:




Add a commit statement and press the tick to commit it.




Your local repo is now up to date however the GitHub repo is not, so use the *Sync Changes* button to update the remote repo.




GitHub Pages may be a little slow to update, so you may need to refresh the pages a few times to see the change filter through. Use the *Elements* feature in the developer tools to check your change has been made before re-running a Lighthouse test.

## Improving the Accessibility Score and Test the Pages

Follow the guidance given by Lighthouse to improve the score of the *all the* pages in the web site. The following provides some tip but beware these pages were put together in a hurry so there are other errors that you also need to test and identify that Lighthouse does not pick up.

## Language Attribute

Lighthouse will identify that the page is missing a language attribute on the `<html>` element. Lighthouse tells us:

*"If a page doesn't specify a lang attribute, a screen reader assumes that the page is in the default language that the user chose when setting up the screen reader."*

This matches the A11Y checklists comments on Global code:

- Use a lang attribute on the html element.

Fix this by ensuring each page has a value for the language attribute of the `<html>` element.

```
<!DOCTYPE html>
<html lang="en">
```

## Title

Lighthouse identify missing a `<title>` element. Lighthouse tells us:

*"The title gives screen reader users an overview of the page, and search engine users rely on it heavily to determine if a page is relevant to their search."*

This matches the A11Y checklists comments on Global code:

- Provide a unique title for each page or view.

Therefore, ensure each page has a `<title>`. This should be a child element of the `<head>` and is ideally unique per page.

```
<title>Your Name :: Portfolio</title>
```

Lighthouse won't pick up that all pages should have unique titles. Check that is the case.

## Images

Lighthouse identifies images that are missing alternate text `alt` attributes. A11Y checklist states:

- Make sure that all `img` elements have an `alt` attribute.
- Make sure that decorative images use null alt (empty) attribute values.
- For images containing text, make sure the alt description includes the image's text.

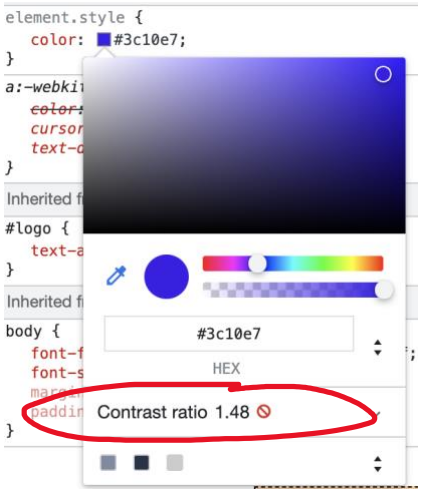
Images should have `alt` attributes if there are more than decorative.

```
<img src="images/square200px.png" alt="Some Other Picture" />
```

If an image is not integral to the information that the page contains, it is decorative and therefore does not need an `alt` value. The `alt` should still be present but left empty, in effect a `null` value. With the *index.html* all the images should really have alt applied.

## Colour Contrast

The A11Y checklist suggests that colour contrast should be checked across a range of features on a webpage. Lighthouse identifies poor colour contrast and has a handy tool to identify a good colour to use.




Target contrast ratios are outlined here:

https://web.dev/color-contrast/?utm_source=lighthouse&utm_medium=devtools

Note: Some issues within the pages can be fixed by applying styling via an improved heading hierarchy (see below).

## Mobile Best Practice

The A11Y checklist under Global code states:

- Ensure that viewport zoom is not disabled.

Lighthouse suggests that disabling zooming via the `<meta viewport>` is not good practice.

In the `<head>` you may find the `<meta viewport>` set as:

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1.0, user-scalable=no" />
```

This meta tag includes setting for restricting the ability of the user to scale the page. Change it to:

```
<meta name="viewport" content="width=device-width, initial-scale=1"/>
```

## Heading Hierarchy

Lighthouse is a good guide but it can only check a subset of accessibility issues automatically. The A11Y checklist states on the use of heading elements that webpages should:

- Use heading elements to introduce content.
- Use only one `h1` element per page or view.
- Heading elements should be written in a logical sequence.
- Don't skip heading levels.

If Lighthouse picks up on the poor colour contrast make sure you use appropriate rules to set the colour. For example if your have text as follows:

```
<h1><a href="index.html">Your Name</a>/</h1>
```

… you should use a rule which selects the `<a>` colour ie:

```
header h1 a {
  color: #fff;
  text-decoration: none;
}
```

## Tables

HTML tables can be made more accessible. The A11Y checklist highlights:

- Use the table element to describe tabular data.
- Use the `th` element for table headers (with appropriate scope attributes).
- Use the `caption` element to provide a title for the table.

Check any tables for accessibility and add `<th>` table header element for cells that act as table headers and use the `scope` attribute to indicate that they are heading for the columns ie:

```
<tr>
    <th scope="col">Qualification</th>
    <th scope="col">Subject</th>
    <th scope="col">Grade</th>
    <th scope="col">Date</th>
</tr>
```

You could also add a `<caption>` element that is placed immediately after the `<table>` element.

## Forms

HTML forms can be made more accessible. The A11Y checklist states:

- All inputs in a form are associated with a corresponding `<label>` element.

For example, the following is visually okay:

```
<div>
    <p>Name:</p>
    <input type="text" name="yourName" id="yourName">
</div>
```

But the above can be made more accessible with the use of `<label>`:

```
<div>
    <label for="yourName">Name:</label>
    <input type="text" name="yourName" id="yourName">
</div>
```

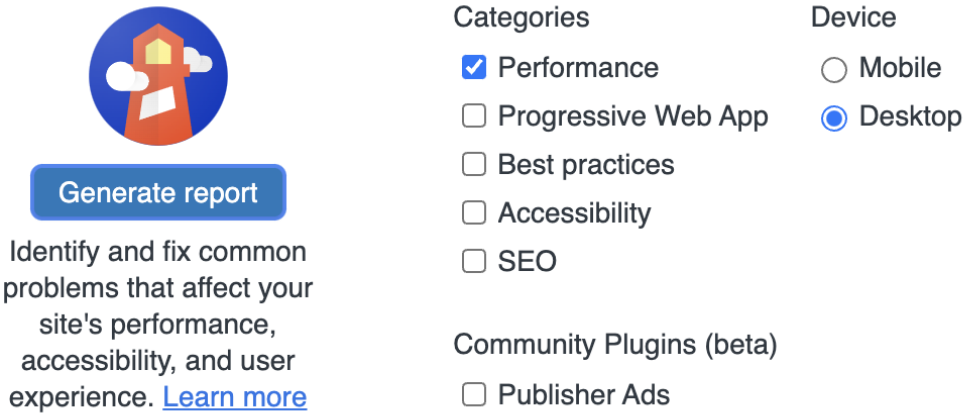
This is likely to impact on your page's appearance as we have swapped a block element `<p>` for an inline element `<label>`. How could you fix this?

## Page Zoom

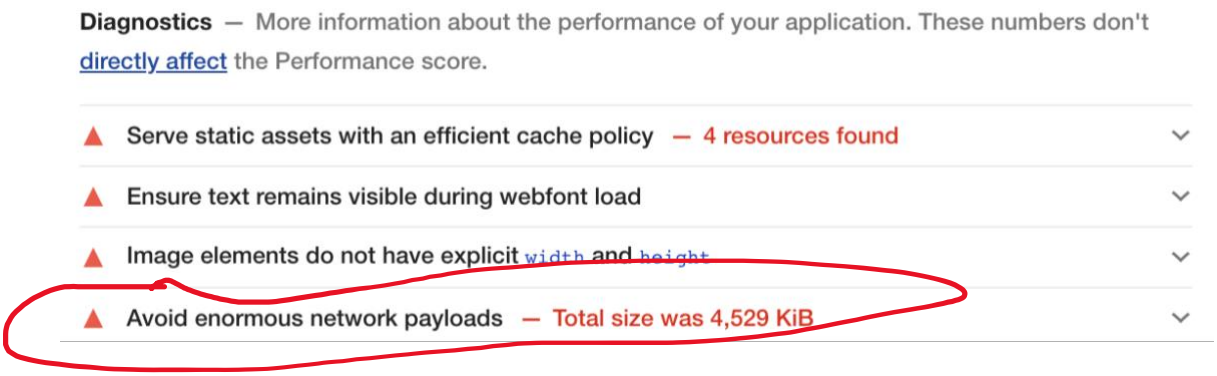
WCAG suggest that a page should zoom by 200% without any major on the usability of the page. Try using the Zoom tool in Chrome to test your project.

## Other Lighthouse Audits

It is useful to consider other useful Lighthouse audits. For example, the *SEO* audit gives advice on how to improve a webpage's search engine ranking. The *Performance* audit suggests ways a page can be made more efficient, that is smaller in file size / payload and therefore quicker to load. Run a Performance audit on the *skill-set.html* webpage.




Notice that Lighthouse picks up on the fact there is a large image file and suggests that we "**Avoid enormous network payloads".**




The image *flowers-big.jpg* is too large at 4.6Mb. As the *flowers-big.jpg* is straight from a phone camera, it is a massive 3648 pixels wide by 2736 pixels in height which explains the 4.6Mb of data.

We need to reduce the image size and dimensions. We will also take the opportunity to make the dimensions of the image better suited to our page as other images in the banner slot in the website are a mere 900px by 200px.

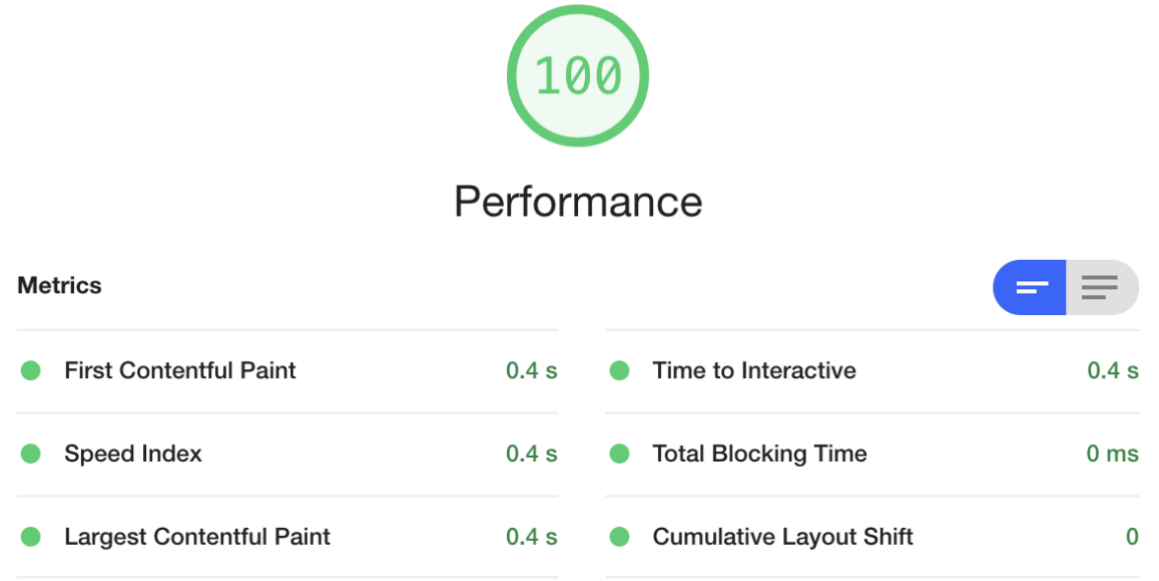
To crop the image, software such as Photoshop, Sketch or GIMP could be used. (These would also allow for artistic changes to be made). Alternatively, online tools can be used. Either way create a resized/cropped version of the image at 900px by 200px and save it as *flowers-sml.jpg* to differentiate it from the original.

Resizing / Cropping tool:
https://resizing.app/features/resize-jpeg/

Optimizing tools:
https://tinyjpg.com/
https://compressjpeg.com/

You should also be able to get the file size down to around 34Kb, a massive network saving. Replace the file used in the *skill-set.html* webpage and upload the new image to GitHub.

When rerunning the performance test, you should now get a much better score.




… and your visitors will be very grateful!